

Andre Pratama

Laravel Uncover

Panduan Belajar Framework Laravel 9

Duniailkom

Laravel Uncover

Panduan Belajar Framework Laravel

Andre Pratama

Buku ini ditulis dan diterbitkan secara mandiri oleh DuniaIlkom (www.duniailkom.com)

06 Juli 2022

~ Update Log ~

- ✓ First Release (Laravel 6) – 29 Desember 2019
- ✓ Update ke Laravel 7 – 06 Maret 2020
- ✓ Update ke Laravel 8 – 06 Oktober 2020
- ✓ Update ke Laravel 9 – 10 Februari 2022
- ✓ Mini Update (Laravel 9.19) – 06 Juli 2022

Cover Photo by [PAUL SMITH](#) on [Unsplash](#)

© 2022 DuniaIlkom

Daftar Isi

Ucapan Terima kasih.....	14
Tentang Penulis.....	15
Lisensi.....	16
Kata Pengantar.....	19
Asumsi / Pengetahuan Dasar.....	21
Contoh Kode Program.....	22
1. Berkenalan Dengan Laravel.....	23
1.1. Pengertian Framework.....	23
1.2. Sejarah Laravel.....	25
Penomoran Versi Laravel.....	27
1.3. Kenapa Harus Laravel?.....	28
1.4. Framework vs PHP Native.....	30
1.5. Code Igniter vs Laravel.....	32
Kemudahan Penggunaan.....	32
Kecepatan Update.....	33
Dukungan Pengembang.....	34
Jadi, Lebih Baik Belajar Laravel?.....	35
2. Arsitektur MVC.....	37
2.1. Pengertian MVC.....	37
2.2. Contoh Penggunaan MVC.....	38
Model.....	41
View.....	43
Controller.....	45
2.3. Diagram Alur MVC.....	46
2.4. Mengenal Routing.....	47
3. Instalasi Laravel.....	50
3.1. Instalasi XAMPP.....	50
3.2. Mengakses PHP dari CMD.....	51

Menjalankan php.exe Secara Global.....	53
Mengatur PATH system variable.....	54
3.3. Instalasi Composer.....	59
Pengertian Composer, Package Manager dan Dependency.....	59
Menginstall Composer.....	62
3.4. Instalasi Laravel.....	65
Install Laravel dengan Composer Create-Project.....	66
Install Laravel dengan Laravel Installer.....	68
Mengatur versi Laravel.....	70
3.5. Instalasi Visual Studio Code.....	72
4. Mengakses Laravel.....	74
4.1. Folder Instalasi Laravel.....	74
4.2. Cara Mengakses Laravel.....	76
4.3. Membuat Shortcut Untuk Laravel Server.....	79
5. Route.....	82
5.1. Pengertian Routing.....	83
5.2. Route Bawaan Laravel.....	84
5.3. Membuat Route.....	87
5.4. Route Parameter.....	90
5.5. Route dengan Optional Parameter.....	91
5.6. Route Parameter dengan Regular Expression.....	92
5.7. Route Redirect.....	94
5.8. Route Group.....	94
5.9. Route Fallback.....	96
5.10. Route Priority.....	96
5.11. Penulisan URL Route.....	97
5.12. Melihat Daftar Route.....	98
6. Error Display & Proses Debugging.....	102
6.1. Tampilan Error Laravel.....	102
6.2. File Konfigurasi Laravel.....	106
Pengaturan File .env.....	107
Pengaturan dari Folder Config.....	109
6.3. Pencarian Kesalahan (Debugging).....	111
6.4. Menghapus Tambahan Tanda Caret (^).....	115

7. View.....	117
7.1. Pengertian View.....	117
7.2. View Bawaan Laravel.....	118
7.3. Membuat View.....	120
7.4. Membuat Struktur Folder View.....	123
7.5. Kode PHP di dalam View.....	124
7.6. Mengirim Data ke View.....	125
Mengirim Data ke View Sebagai Argumen.....	125
Mengirim Data ke View Menggunakan method With.....	129
Mengenal Function compact().....	131
7.7. Pengelolaan Assets.....	134
Folder Public.....	134
Tanda Forward Slash di Awal URL Assets.....	140
7.8. Bootstrap CSS Framework.....	143
 8. Blade Template Engine.....	149
8.1. Pengertian Blade.....	149
8.2. Menampilkan Data.....	150
8.3. Kondisi If Else.....	154
8.4. Kondisi Switch.....	158
8.5. Perulangan For.....	160
8.6. Perulangan While.....	161
8.7. Perulangan Foreach.....	161
8.8. Perulangan Forelse.....	165
8.9. Perintah Continue dan Break.....	165
8.10. Baris Komentar dan PHP mode.....	167
8.11. Merancang Layout.....	168
8.12. View Include.....	176
8.13. Mengirim Data ke Include.....	179
8.14. Layout Extends.....	181
Mengatasi Masalah Menu Navigasi.....	187
8.15. Default Value.....	189
8.16. Section Extends.....	190
8.17. Components dan Slots.....	194
8.18. Named Routes.....	201
Relative URL vs Absolute URL.....	203
Named Routes Parameter.....	205

8.19. Asset dan Url Function Helper.....	207
8.20. VS Code Laravel Extension Pack.....	209
9. Laravel Mix (Compiling Assets).....	212
9.1. Pengertian Laravel Mix.....	212
Penggabungan Assets.....	212
Memproses Kode Pre-processor.....	213
File Minification.....	214
Cache Busting (Versioning).....	214
9.2. Instalasi Node.js.....	215
Mengenal npm.....	218
9.3. Instalasi Laravel Mix.....	219
9.4. Menjalankan Laravel Mix.....	221
Penggabungan File.....	222
Memproses Pre-processor.....	228
Membuat File Minify.....	232
Membuat Cache Busting (File Versioning).....	233
Menjalankan Laravel Mix Secara Otomatis.....	235
9.5. Compile Bootstrap dengan Laravel Mix.....	235
10. Laravel UI.....	239
10.1. Pengertian Laravel UI.....	239
10.2. Instalasi Laravel UI.....	240
10.3. Menjalankan Laravel UI.....	243
11. Controller.....	248
11.1. Pengertian Controller.....	248
11.2. Cara Mengakses Controller.....	249
Folder app\Http\Controllers.....	249
11.3. Membuat Controller Secara Manual.....	251
Aturan Penamaan Controller.....	253
11.4. Cara Penulisan Route Untuk Controller.....	254
11.5. Mengakses View dari Controller.....	255
11.6. Memindahkan File Controller.....	258
11.7. Mengakses Laravel Facade.....	260
Akses Facade Secara Langsung.....	261
Akses Facade Dengan Perintah use.....	262

11.8. Mengakses External Class.....	263
11.9. Membuat Controller dari php artisan.....	266
11.10. Menggunakan Penulisan Route Laravel 7 (Opsional).....	274
12. Collection.....	278
12.1. Pengertian Collection.....	278
12.2. Cara Pembuatan Collection.....	278
12.3. Mengakses Collection.....	282
12.4. Collection Method.....	285
Method sum(), avg(), max(), min() dan median().....	285
Method random().....	285
Method concat().....	286
Method contains().....	286
Method unique().....	286
Method all().....	287
Method first() dan last().....	288
Method count().....	288
Method sort().....	288
Method get().....	289
Method has().....	289
Method replace().....	290
Method forget().....	290
Method flip().....	291
Method keys() dan values().....	292
Method search().....	292
Method each().....	293
12.5. Nested Array Collection Method.....	294
Method sortBy() dan sortByDesc().....	295
Method filter().....	297
Method where() dan firstWhere().....	298
Method whereBetween() dan whereNotBetween().....	300
Method whereIn() dan whereNotIn().....	301
Method pluck().....	302
12.6. Object Array Collection Method.....	303
Method groupBy().....	307
13. Migration.....	312

13.1. Pengaturan Database Laravel.....	312
File Konfigurasi .env dan config\database.php.....	312
Jalankan Apache dan MySQL dari XAMPP.....	315
13.2. Pengertian Migration.....	316
13.3. File Migration Bawaan Laravel.....	317
13.4. Menjalankan Migration.....	320
13.5. Migration Rollback.....	323
13.6. Membuat Migration.....	325
13.7. Alter Table Migration.....	334
14. DB Facade (Raw SQL Queries).....	338
14.1. Pengertian DB Facade (Raw SQL Queries).....	338
Membuat Migration dan Controller.....	339
14.2. Menginput Data (DB::insert).....	341
Membuat Prepared Statement.....	344
14.3. Mengupdate Data (DB::update).....	346
14.4. Menghapus Data (DB::delete).....	347
14.5. Menampilkan Data (DB::select).....	347
14.6. Menjalankan Query (DB::statement).....	352
15. Query Builder.....	354
15.1. Pengertian Query Builder.....	354
Membuat Migration dan Controller.....	354
15.2. Menginput Data.....	355
15.3. Mengupdate Data.....	358
15.4. Menghapus Data.....	361
15.5. Menampilkan Data.....	362
Method get().....	362
Method where() dan orderBy().....	365
Method select().....	366
Method skip() dan take().....	367
Method first().....	367
Method find().....	370
Method selectRaw().....	370
15.6. Mini Case Study.....	371
16. Eloquent ORM.....	376

16.1. Pengertian Eloquent ORM.....	376
Membuat Migration dan Controller.....	378
16.2. Pembuatan Model.....	379
16.3. Pengaksesan Model.....	381
16.4. Menginput Data.....	382
Mass Assignment.....	384
16.5. Mengupdate Data.....	389
Mass Update.....	391
16.6. Menghapus Data.....	392
Mass Delete.....	394
16.7. Menampilkan Data.....	395
Method All().....	396
Mengirim Data ke View.....	398
Method where().....	400
Method first().....	401
Method find().....	403
Method latest().....	404
Method limit().....	405
Method skip() dan take().....	406
16.8. Soft Delete.....	406
Restore Soft Delete.....	410
Menghapus Data Permanen.....	411
17. Form Processing dan Form Validation.....	413
17.1. Pembuatan Form.....	413
17.2. Request Object.....	417
17.3. Post Request dan CRFS Token.....	420
17.4. Validasi From dari Request Object.....	423
17.5. Mengakses Pesan Error.....	426
17.6. Pesan Error Terpisah.....	429
17.7. Repopulate Form.....	433
Checked / Selected Blade Directives.....	435
17.8. Validator Class.....	436
17.9. Membuat Custom FormRequest Class.....	442
18. Localization.....	447
18.1. Pengertian Localization.....	447

18.2. Folder lang.....	448
18.3. Function __() dan Perintah @lang.....	449
18.4. Membuat 2 Bahasa Localization.....	451
18.5. Mengatur Localization secara Dinamis.....	457
18.6. Localization dengan Route Parameter.....	459
18.7. Localization Untuk Validasi Form.....	460
18.8. Translation Strings.....	466
19. CRUD.....	470
19.1. Persiapan Awal.....	470
19.2. Create.....	477
19.3. Read.....	482
Menampilkan Semua Isi Tabel.....	482
Menampilkan Satu Data Tabel.....	485
Route Model Binding.....	488
19.4. Refactoring File.....	489
Refactoring Form Pendaftaran.....	489
Refactoring View Index.....	490
19.5. Flash Data.....	493
19.6. Validasi Duplikasi Data.....	497
19.7. Update.....	500
Menampilkan Form Update.....	500
Proses Update Tabel.....	506
Modifikasi File View.....	512
19.8. Delete.....	514
19.9. Resource Controllers.....	516
20. File Upload.....	521
20.1. Persiapan Awal.....	521
20.2. Informasi File Upload.....	524
20.3. Validasi File Upload.....	526
20.4. Memindahkan File Upload.....	529
20.5. Membuat Symlink.....	533
20.6. Method move().....	538
20.7. Mini Case Study: File Upload Rename.....	539
21. Middleware.....	543

21.1. Pengertian Middleware.....	543
21.2. Persiapan Awal.....	543
21.3. Membuat Middleware.....	545
21.4. Mendaftarkan Middleware.....	547
21.5. Mengaktifkan Middleware.....	550
Mengaktifkan Middleware dari Route.....	550
Mengaktifkan Middleware dari Controller.....	551
21.6. Redirect Middleware.....	553
21.7. Laravel Maintenance Mode.....	554
22. Session.....	557
22.1. Persiapan Awal.....	557
22.2. Membuat Session.....	559
22.3. Membaca Session.....	561
22.4. Menghapus Session.....	563
22.5. Flash Session.....	564
23. Case Study: Login Middleware.....	566
23.1. Pembuatan Route.....	566
23.2. Pembuatan Controller.....	567
23.3. Pembuatan Middleware.....	570
23.4. Pembuatan View.....	571
24. Authentication.....	576
24.1. Instalasi Laravel Authentication.....	576
24.2. Users Table.....	580
24.3. File Laravel Authentication.....	582
File Route.....	582
File Controller.....	584
File Model.....	588
File View.....	589
24.4. Case Study: Menambah Halaman Baru.....	591
24.5. Menampilkan Data User.....	595
25. Case Study: CRUD dan Authentication.....	598
25.1. Tampilan Akhir "CRUD Jurusan".....	598
25.2. Membuat Migration.....	599
25.3. Menyiapkan Route.....	599

25.4. Menyiapkan Model.....	600
25.5. Menyiapkan JurusanController.....	600
25.6. Modifikasi Auth Controller.....	603
25.7. Membuat View.....	603
26. Policy.....	612
26.1. Pengertian Laravel Policy.....	612
26.2. Membuat Policy.....	612
26.3. Membatasi Proses Create.....	615
26.4. Membatasi Tampilan di View.....	618
26.5. Membatasi Proses Delete.....	620
26.6. Pembatasan di Route.....	622
27. Case Study: ILKOOM Profile Manager.....	626
27.1. Instalasi Laravel Authentication.....	629
27.2. Modifikasi Tabel Users.....	630
27.3. Instalasi Localization.....	632
27.4. Persiapan File Gambar.....	633
27.5. Instalasi Font Awesome.....	633
27.6. Membuat File Sass.....	634
27.7. Modifikasi View layout\app.blade.php.....	639
27.8. Modifikasi View auth\register.blade.php.....	644
27.9. Membuat View layout\form.blade.php.....	646
27.10. Membuat File JavaScript.....	655
27.11. Validasi Form Pendaftaran.....	658
27.12. Modifikasi User Model.....	662
27.13. Membuat symlink.....	664
27.14. Modifikasi HomeController.....	665
27.15. Modifikasi View home.blade.php.....	666
27.16. Membuat UserController.....	670
27.17. Modifikasi Route.....	671
27.18. Modifikasi Property \$redirectTo.....	672
27.19. Menyiapkan Data Update.....	672
27.20. Membuat Form Update.....	673
27.21. Memproses Update User.....	675
27.22. Membuat Proses Delete User.....	677
27.23. Membuat Policy.....	678

27.24. Tambahan JavaScript Untuk Delete.....	681
27.25. Modifikasi View auth\login.blade.php.....	686
27.26. Modifikasi View home.blade.php.....	688
Penutup Laravel Uncover.....	692
Daftar Pustaka.....	694

Ucapan Terima kasih

Dalam kesempatan ini saya ingin mengucapkan terima kasih kepada Allah SWT karena dengan karuniaNya saya masih diberi kesempatan dan kesehatan untuk bisa menulis buku kesembilan DuniaIlkom: **Laravel Uncover**.

Selanjutnya kepada keluarga yang terus memberi motivasi dan dukungan tiada henti untuk terus mengembangkan DuniaIlkom.

Terakhir kepada rekan-rekan pembaca dan pengunjung setia DuniaIlkom. Terutama bagi yang telah memberikan donasi untuk membeli buku saya sebelumnya: HTML, CSS, PHP, JavaScript, MySQL, Pascal, Bootstrap, dan OOP PHP Uncover. Karena dari *feedback* dan dukungan rekan-rekan lah saya bisa lanjut menulis buku Laravel ini. Terima kasih :)

Padang Panjang, 2022

Penulis

Andre Pratama

www.duniaIlkom.com

Tentang Penulis



Andre Pratama

Andre memiliki background S1 Ilmu Komputer dari Universitas Sumatera Utara (angkatan 2005). Sempat terjun ke dunia kerja sebagai Assistant Manager di Bank Mandiri tahun 2010 - 2014.

Di akhir 2014, memutuskan untuk menjadi praktisi dan penulis buku programming. Saat ini full time mengelola web duniaIlkom yang sudah dirintis sejak tahun 2012. Harapannya, web duniaIlkom bisa menjadi sebagai salah satu media belajar programming dan ilmu komputer terbaik di Indonesia.

Andre berdomisili di kota Padang Panjang, Sumatera Barat. Jika ada pertanyaan, saran, kritik yang membangun bisa menghubungi duniaIlkom@gmail.com atau WA ke 083180285808.

Lisensi

Terima kasih untuk tidak memperbanyak / mengedarkan / mencopy eBook ini

Menulis sebuah buku hingga ratusan halaman butuh waktu yang tidak sebentar. Belum lagi saya harus berjuang mempelajari referensi yang kebanyakan dalam bahasa inggris. Ini saya lakukan agar pembaca bisa mendapatkan materi yang detail, update, dan berkualitas.

Saya menyadari kekurangan sebuah ebook adalah mudah dicopy-paste dan disebarluaskan. Tapi dengan eBook, harga buku bisa ditekan. Selain tidak perlu mencetak, eBook DuniaIlkom ini bisa di dapat dengan mudah dan murah, termasuk bagi teman-teman di daerah yang ongkos kirimnya lumayan mahal (jika berbentuk buku fisik).

Atas dasar itulah saya mohon kerjasamanya dari rekan-rekan semua untuk **tidak memperbanyak, menggandakan, atau mengupload ulang buku ini di forum, situs maupun media lain dalam bentuk apapun (termasuk tidak membuat video youtube dari materi buku).**

Saya juga berharap rekan-rekan tidak memposting materi apapun yang ada di dalam buku ini. Jika ingin sebagai bahan artikel untuk postingan blog/situs, silahkan ambil materi yang ada di website duniaIlkom (jangan yang dari buku).

Apabila rekan-rekan memperoleh buku ini **bukan** dari DuniaIlkom, saya mohon bantuan donasinya untuk membeli versi asli. Donasi pembelian buku ini adalah sumber mata pencarian saya untuk menafkahi keluarga. Lisensi atau hak guna buku ini hanya untuk 1 orang, yakni yang telah membeli langsung ke duniaIlkom@gmail.com.

Dengan kualitas yang ditawarkan, harga buku ini cukup terjangkau. Buku ini saya buat dengan waktu yang tidak sebentar, hingga berbulan-bulan, kadang sampai tengah malam. Bantuan donasi dari rekan-rekan yang membeli buku secara resmi sangat saya hargai, selain mendapat ilmu yang berkah, ini juga bisa menjadi penyemangat saya untuk terus berkarya dan menghadirkan ebook-ebook programming berkualitas lainnya.

Untuk yang membeli dari DuniaIlkom, saya ucapan banyak terimakasih :)

Anda diperbolehkan untuk:

- ✓ Mencetak eBook ini untuk keperluan pribadi dan dibaca sendiri.
- ✓ Mencopy eBook ini ke laptop/smartphone/tablet milik sendiri.
- ✓ Membuat ringkasan buku untuk digunakan sebagai bahan ajar (bukan keseluruhan isi buku).

Anda tidak dibolehkan untuk:

- ✗ Mencetak eBook ini untuk dibaca oleh orang lain, walaupun gratis.
- ✗ Mencopy eBook ini untuk dijual ulang, maupun dibagikan kepada orang lain dengan gratis.
- ✗ Membeli buku ini untuk dibaca bersama-sama (lisensi buku ini hanya untuk 1 orang).
- ✗ Mengambil sebagian atau seluruh isi buku untuk di publish ke blog, situs, artikel, dan media lain dalam bentuk apapun.
- ✗ Menjadikan materi buku sebagai bahan video YouTube / media public lain.
- ✗ Membagikan eBook ini kepada murid/siswa/mahasiswa (jika digunakan untuk bahan pengajaran).

Setiap pelanggaran dari lisensi ini akan dituntut sesuai undang-undang yang berlaku di Republik Indonesia, terutama **Pasal 12 UU No. 19 Tahun 2002** tentang **Hak Cipta**.

Penjelasan lebih lanjut bisa ke: [Apakah Mengunduh E-book Termasuk Perbuatan Illegal?](#)

Khusus untuk pembaca muslim bisa ke: [Hukum Memakai Barang Bajakan](#). Mari kita jaga agar ilmu yang di dapat berkah dan bermanfaat, bukan dari sumber yang haram.

REPUBLIK INDONESIA
KEMENTERIAN HUKUM DAN HAK ASASI MANUSIA

SURAT PENCATATAN CIPTAAN

Dalam rangka pelindungan ciptaan di bidang ilmu pengetahuan, seni dan sastra berdasarkan Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta, dengan ini menerangkan:

Nomor dan tanggal permohonan :

Pencipta

Nama : Andre Pratama

Alamat :

Kewarganegaraan :

Indonesia

Pemegang Hak Cipta

Nama : Andre Pratama

Alamat :

Kewarganegaraan :

Indonesia

Jenis Ciptaan :

e-Book

Judul Ciptaan :

Laravel Uncover

Tanggal dan tempat diumumkan untuk pertama kali di wilayah Indonesia atau di luar wilayah Indonesia

: 30 Oktober 2019, di Padang Panjang

Jangka waktu pelindungan

: Berlaku selama hidup Pencipta dan terus berlangsung selama 70 (tujuh puluh) tahun setelah Pencipta meninggal dunia, terhitung mulai tanggal 1 Januari tahun berikutnya.

Nomor pencatatan :

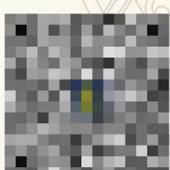
adalah benar berdasarkan keterangan yang diberikan oleh Pemohon.

Surat Pencatatan Hak Cipta atau produk Hak terkait ini sesuai dengan Pasal 72 Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta.

a.n. MENTERI HUKUM DAN HAK ASASI MANUSIA
DIREKTUR JENDERAL KEKAYAAN INTELEKTUAL



Dr. Freddy Harris, S.H., LL.M., ACCS.
NIP. 196611181994031001



Kata Pengantar

Laravel merupakan framework PHP paling populer saat ini, setidaknya itulah yang saya dapat dari hasil [Google Trends](#). Salah satu faktor yang membuat perkembangan Laravel sedemikian pesat adalah selalu update dengan kebutuhan programmer. Saking updatenya, setiap 1 tahun sekali akan hadir Laravel versi terbaru.

Bagi programmer web yang ingin fokus ke back-end PHP (atau memilih jalur full stack), Laravel menjadi salah satu materi yang wajib dikuasai. Mayoritas lowongan kerja web programmer back-end di Indonesia mewajibkan paham sampai ke framework, yang biasanya salah satu dari **Code Igniter** atau **Laravel**.

Framework memang bukan hal wajib untuk bisa membuat web, tapi untuk aplikasi yang besar dan butuh kerjasama tim, framework seperti Laravel akan sangat membantu. Penerapan design arsitektur **MVC** (Model-View-Controller) dirancang sedemikian rupa agar tidak saling tercampur antara design tampilan (HTML, CSS, JavaScript) dengan logika program (PHP, MySQL).

Buku **Laravel Uncover** ini saya tujuhan bagi rekan-rekan yang ingin mempelajari framework Laravel dari awal. Kita akan bahas berbagai fitur dasar yang disediakan Laravel.

Disebut 'dasar' karena ekosistem Laravel itu sendiri sangat luas dan mencakup tidak hanya PHP saja, tapi juga berbagai teknologi web lain yang saling terhubung. Karena kita sudah sampai ke materi framework (yang termasuk kategori *advanced PHP*), maka *dasar* yang dimaksud juga tidak benar-benar dasar, melainkan fitur awal dari framework Laravel.

Sama seperti buku duniaIlkom lain, buku **Laravel Uncover** ini lebih menekankan ke pemahaman materi, tidak sekedar "asal projectnya jadi" sehingga kadang 1 baris kode program butuh penjelasan hingga beberapa halaman. Karena saking banyaknya materi, bahasan yang lebih advanced akan menjadi jatah seri **Laravel In Depth**.

Di buku Laravel Uncover ini kita akan bahas inti dari framework Laravel terutama konsep MVC, kemudian di seri Laravel In Depth baru akan masuk ke materi yang lebih detail serta fitur-fitur tambahan yang sayang untuk dilewatkan.

Sebagai contoh, dalam buku ini akan dibahas tentang **Eloquent ORM**, yakni fitur Laravel untuk mengakses database, tapi isinya baru mengakses 1 tabel saja. Di buku **Laravel In Depth #1** kita akan pelajari cara penggunaan Eloquent untuk mengakses beberapa tabel yang saling terhubung (*eloquent relationship*)

Di buku ini juga dibahas tentang fitur **authentication**, yakni proses login, logout dan pendaftaran user. Di seri in depth lainnya akan dipelajari dengan lebih detail sampai proses aktivasi user serta *reset/forgot password*.

Harapan saya, buku **Laravel Uncover** ini bisa menjadi salah satu buku terbaik berbahasa Indonesia untuk mempelajari framework Laravel. Jika pun anda memutuskan hanya membaca buku ini saja (tidak lanjut ke seri *in depth*), setidaknya sudah punya dasar yang cukup untuk memahami bagaimana cara kerja Laravel.

Akhir kata, semoga buku Laravel Uncover bisa menjadi buku pengantar terbaik dalam langkah anda menjadi seorang web programmer. Sampai jumpa di bab terakhir :)

Asumsi / Pengetahuan Dasar

Laravel adalah framework PHP, sehingga saya berasumsi rekan-rekan sudah paham tentang PHP sebelum membaca buku ini. PHP yang dimaksud bukan hanya PHP dasar (*procedural*), tapi juga mencakup materi pemrograman object PHP (**OOP PHP**).

Tidak hanya itu, sepanjang pembahasan nanti kita juga banyak mengakses database MySQL, termasuk menulis query dan istilah-istilah lain terkait database. Sehingga saya menganggap juga sudah paham perintah untuk membuat tabel MySQL serta query dasar seperti **SELECT**, **INSERT**, **UPDATE** dan **DELETE**.

Untuk sisi design, dalam buku ini saya akan memakai **Bootstrap** agar tampilan program menjadi lebih sedap untuk dipandang. Ini tidak wajib, tapi agar bisa memahami seluruh materi, sebaiknya juga punya dasar Bootstrap terlebih dahulu. Dan, tentu saja agar bisa ke Bootstrap harus dari **HTML** dan **CSS** terlebih dahulu.

Huff... syarat yang cukup banyak... Ini karena Laravel memang bukan ditujukan untuk pemula.

Materi-materi di atas juga tidak harus dipelajari dari buku DuniaIlkom, tapi bisa juga dari tutorial/buku lain. Namun yang paling ideal adalah dari 7 buku duniaIlkom berikut:

1. **HTML Uncover**
2. **CSS Uncover**
3. **PHP Uncover**
4. **MySQL Uncover**
5. **JavaScript Uncover**
6. **Bootstrap Uncover**
7. **OOP PHP Uncover**

Contoh Kode Program

Seluruh kode program yang ada dalam buku **Laravel Uncover** ini bisa di download dari folder sharing Google Drive yang di kirim pada saat pembelian: **belajar_laravel.zip**. Isinya berupa file PHP yang disusun sesuai dengan bab tempat kode tersebut dibahas.

File yang tersedia juga bukan sebuah aplikasi laravel lengkap, tapi hanya beberapa file saja. Agar bisa berjalan, setiap file harus ditempatkan ke dalam folder yang sesuai di aplikasi Laravel. Sebagai contoh, file route `web.php` harus ditempatkan ke dalam folder `routes`, dan file `MahasiswaController` harus disimpan ke dalam folder `app\Http\Controllers\`. Terkait lokasi folder ini akan dibahas sesuai bab yang bersangkutan.

Penomoran baris (*line numbering*) pada contoh kode program dalam buku ini berguna untuk memudahkan pembahasan. Jika ingin men-copy kode ini langsung dari eBook **pdf**, gunakan kombinasi tombol **ALT + tahan tombol mouse selama proses seleksi** agar *line numbering* tidak ikut di-copy. Namun hanya bisa dilakukan dari aplikasi pdf reader tertentu seperti Adobe Acrobat Reader.

Alternatif yang lebih saya sarankan adalah dengan mengetik ulang seluruh kode program yang ada supaya lebih cepat paham sekaligus bisa menghafal kegunaan dari setiap kode. Jika setelah diketik ternyata tidak jalan, besar kemungkinan ada penulisan yang salah. Solusinya, samakan kode yang di tulis dengan file yang tersedia di **belajar_laravel.zip**.

1. Berkenalan Dengan Laravel

Dalam bab pertama buku **Laravel Uncover** ini kita akan berkenalan dengan Laravel, sejarah singkat Laravel, membahas Laravel sebagai PHP Framework, serta perbandingan Laravel dengan framework lain.

1.1. Pengertian Framework

Laravel adalah **sebuah framework PHP**. Untuk bisa memahami ini maka kita harus membahas terlebih dahulu apa yang dimaksud dengan *framework*.

Secara sederhana, **framework** adalah kumpulan kode program siap pakai dengan aturan penulisan tertentu yang bertujuan untuk memudahkan serta mempercepat pembuatan aplikasi. Lebih spesifik lagi, PHP framework adalah framework yang dibuat menggunakan bahasa pemrograman PHP.

Tujuan utama kenapa menggunakan framework adalah untuk **mempercepat pembuatan aplikasi**, karena di dalam framework sudah tersedia berbagai fitur siap pakai. Kita tinggal menggunakan fitur ini tanpa perlu membuat semuanya dari nol. Selain itu aturan penulisan di framework akan memaksa kita menggunakan cara penulisan yang baik (mengikuti standar *best practice*).

Framework pada awalnya berasal dari kebutuhan programmer untuk mengurangi pembuatan kode yang sama berulang kali.

Sebagai contoh, misalkan saya bekerja sebagai programmer di sebuah perusahaan software. Project pertama disuruh membuat sistem informasi sekolah, dimana perlu fitur pendaftaran mahasiswa, pendaftaran dosen, login mahasiswa dan login dosen.

Setelah selesai, lanjut ke project kedua untuk membuat aplikasi perpustakaan yang diantaranya perlu halaman pendaftaran karyawan serta login karyawan. Project ketiga berupa website berita yang juga perlu fitur pendaftaran editor dan login editor.

Sampai di sini bisa dilihat bahwa untuk ketika project, saya perlu membuat fitur pendaftaran dan login. Dan besar kemungkinan untuk project keempat dan seterusnya juga butuh fitur serupa.

Daripada membuat dari nol terus menerus, akan lebih efisien jika saya menyiapkan sebuah

kode dasar untuk halaman pendaftaran dan login. Jika project baru butuh fitur yang sama, tinggal copy kode ini dan edit sedikit di bagian tertentu. Pekerjaan selesai dengan jauh lebih cepat.

Seiring bertambahnya pengalaman, kode dasar ini saya modifikasi dengan tambahan berbagai fitur baru agar semakin lengkap, misalnya menambah cara koneksi ke database, penanganan cookie, validasi form, dst. Akhirnya, jadilah sebuah **framework**.

Beberapa waktu kemudian, saya merasa framework ini sangat memudahkan dan ingin berbagi dengan programmer lain. Selain berbuat baik (karena banyak yang akan terbantu), saya berharap programmer lain juga bisa memberikan koreksi serta menambah fitur-fitur baru.

Jadi, sebenarnya siapa saja bisa membuat framework (tentunya selama memiliki skill), namun di antara sekian banyak framework, ada yang lebih populer dari yang lain. Alasannya bisa jadi karena fitur yang disediakan lebih banyak, penggunaannya lebih mudah, serta memiliki komunitas yang aktif.

Jika framework itu dibuat dengan struktur yang rapi dan mudah digunakan, maka secara perlahan makin banyak programmer lain ikut berkontribusi. Mayoritas framework juga bagian dari *open source project* sehingga bisa kita pakai dengan gratis.

Terdapat berbagai framework untuk keperluan yang berbeda-beda. Sebagai contoh, **Bootstrap** adalah sebuah framework CSS yang berisi kumpulan kode CSS untuk mempercepat pembuatan design web. Selain Bootstrap masih banyak framework CSS lain seperti **Tailwind CSS**, **Materialize**, **Bulma**, dan **Semantic UI**.

Begitu juga di PHP, terdapat berbagai pilihan framework seperti **Code Igniter**, **Symfony**, **Yii**, **Zend** dan tentu saja **Laravel**.

Dengan menggunakan framework, ibaratnya kita memanfaatkan keahlian ribuan programmer yang sudah lebih dahulu memikirkan apa yang harus dibuat dan bagaimana cara terbaik untuk membuatnya.

Framework vs Library

Selain framework, terdapat juga istilah **library**. Keduanya sama-sama berisi kode program yang dibuat oleh programmer lain dan bisa dipakai untuk mempercepat pembuatan aplikasi.

Library umumnya berisi kumpulan kode program untuk tugas yang lebih spesifik (biasanya untuk 1 fungsi saja) sehingga lebih sederhana daripada framework.

Misalnya kita butuh membuat perhitungan statistika, maka bisa mencari library yang berisi rumus-rumus statistik siap pakai. Kode program ini biasanya berbentuk *function* atau *class*. Prinsip utama dari library adalah, **kita tetap memiliki kontrol tentang cara penulisan kode program**.

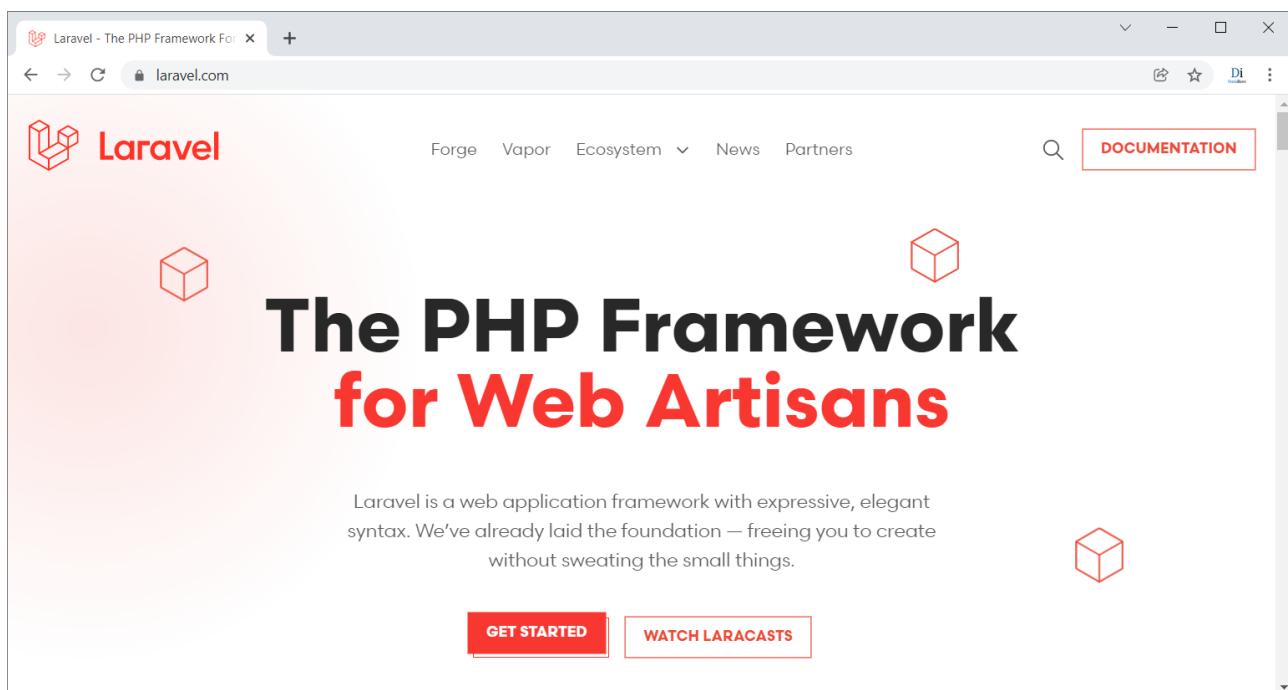
Sebagai contoh, [dompdf](#) adalah library PHP yang bisa dipakai untuk membuat file pdf. Atau [jQuery](#), yang merupakan salah satu library populer dari JavaScript.

Framework lebih kompleks daripada library dan dipakai untuk membuat sebuah aplikasi utuh, tidak hanya untuk 1 tugas saja. Pada prakteknya, di dalam framework bisa jadi terdapat puluhan library yang saling bekerja sama.

Prinsip utama dari framework adalah, **kita tidak memiliki kontrol tentang cara penulisan kode program**. Framework sudah memiliki aturan penulisan baku yang harus di ikuti.

Tips praktis untuk bisa membedakan library dan framework adalah, jika kode tambahan tersebut ada di dalam kode kita, maka besar kemungkinan itu adalah sebuah **library**. Sedangkan jika kita menginput sesuatu ke kode program yang sudah ada maka besar kemungkinan itu adalah sebuah **framework**.

1.2. Sejarah Laravel



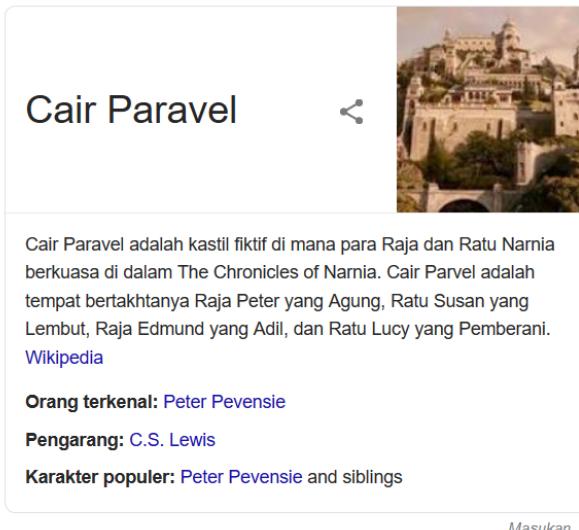
Gambar: Tampilan halaman web resmi Laravel (laravel.com)

Ide awal pembuatan framework **Laravel** berasal dari masalah yang dihadapi oleh **Taylor Otwell** pada tahun 2011. Saat itu sebenarnya sudah tersedia cukup banyak framework PHP, salah satu yang paling populer adalah **Code Igniter** (sering disingkat sebagai CI).

Framework CI sudah cukup baik, namun **Taylor** merasa masih kurang sesuai dengan kebutuhannya. Salah satu fitur yang tidak tersedia di CI adalah *user authentication* dan *user*

authorization bawaan, yakni sebuah mekanisme pendaftaran dan login user. Padahal hampir semua aplikasi web butuh fitur ini.

Maka Taylor mencoba merancang framework PHP yang dinamakan **Laravel**. Pemilihan nama Laravel sendiri juga cukup unik karena terinspirasi dari kastil **Cair Paravel** dari novel / film **The Chronicles of Narnia** (sumber: [Where did the name Laravel come from?](#)).



Gambar: Cuplikan penjelasan dari Cair Paravel

Secara resmi Laravel versi beta di rilis pertama kali pada 9 Juni 2011, yang segera diikuti dengan **Laravel 1.0** sebulan sesudahnya. Segera setelah di rilis, kalangan open source tertarik untuk mengembangkan framework Laravel agar lebih lengkap lagi.

Masih di tahun yang sama, **Laravel 2.0** di rilis pada September 2011, yang kemudian diikuti **Laravel 3.0** di Februari 2012, **Laravel 4.0** di Mei 2013, dan **Laravel 5.0** pada Februari 2015.

Untuk versi 5.0 ke atas, tim pengembang Laravel memutuskan membuat update terjadwal setiap 6 bulan sekali. Yang kemudian menjadi 1 tahun sekali sejak Laravel 8.

Dengan adanya jadwal rilis, Laravel diharapkan bisa berinovasi dengan lebih cepat. Namun ini juga jadi kendala bagi kita sebagai programmer, karena setiap 1 tahun harus selalu update mengenai fitur yang berubah dan ditambah ke Laravel terbaru.

Dukungan perbaikan bug dan celah keamanan juga jadi pertimbangan lain. Sebagai contoh, buku ini menggunakan Laravel versi 9 yang rilis Februari 2022, maka di Februari 2023 akan hadir Laravel 10.

Jadi bagaimana nasib aplikasi yang menggunakan versi 9? Normalnya, tim Laravel merilis dukungan perbaikan bug selama 18 bulan dan perbaikan celah keamanan selama 2 tahun untuk setiap versi.

Jika dukungan ini dirasa terlalu singkat, terdapat versi **LTS** (*long-term support*) yang dirilis berkala. Khusus untuk versi LTS, dukungan perbaikan bug disediakan selama 2 tahun dan

perbaikan celah keamanan selama 3 tahun. Untungnya, Laravel 9 merupakan versi LTS sehingga mendapat support yang lebih lama.

Tabel berikut merangkum jadwal rilis Laravel 1 hingga (perkiraan) Laravel 11:

Version	Release date	PHP version
1.0	June 2011	
2.0	September 2011	
3.0	February 22, 2012	
3.1	March 27, 2012	
3.2	May 22, 2012	
4.0	May 28, 2013	≥ 5.3.0
4.1	December 12, 2013	≥ 5.3.0
4.2	June 1, 2014	≥ 5.4.0
5.0	February 4, 2015	≥ 5.4.0
5.1 LTS	June 9, 2015	≥ 5.5.9
5.2	December 21, 2015	≥ 5.5.9
5.3	August 23, 2016	≥ 5.6.4
5.4	January 24, 2017	≥ 5.6.4
5.5 LTS	August 30, 2017	≥ 7.0.0
5.6	February 7, 2018	≥ 7.1.3
5.7	September 4, 2018	≥ 7.1.3
5.8	February 26, 2019	≥ 7.1.3
6 LTS	September 3, 2019	≥ 7.2 and ≤ 8.0 ^[18]
7	March 3, 2020 ^[19]	≥ 7.2 and ≤ 8.0 ^[18]
8	September 8, 2020	≥ 7.3 and ≤ 8.1 ^[18]
9 LTS	February 8, 2022 ^[18]	≥ 8.0 and ≤ 8.1 ^[18]
10	February 7, 2023	≥ 8.0 and ≤ 8.1 ^[18]
11	January 2024	

Legend:	Old version, not maintained	Older version, still maintained	Current stable version	Latest preview version	Future release
---------	-----------------------------	---------------------------------	------------------------	------------------------	----------------

Gambar: Tanggal perilisan Laravel (sumber: [wikipedia](#))

Tipsnya, jika sedang mengerjakan project besar dan sensitif dengan keamanan, sebaiknya pilih versi LTS agar mendapat dukungan perbaikan bug dan celah keamanan yang lebih lama. Namun jika ingin melihat dan memakai fitur terbaru Laravel, pilihlah versi paling akhir.

Pindah versi Laravel untuk aplikasi yang sudah jadi juga kurang disarankan, karena besar kemungkinan ada fitur yang berubah dari versi lama. Kecuali anda siap meluangkan waktu untuk mengubah kode di aplikasi lama agar sesuai dengan update terbaru Laravel.

Penomoran Versi Laravel

Pada saat Laravel masuk ke versi 5, penomoran yang dipakai adalah di digit kedua setiap 6 bulan sekali. Maksudnya setelah **Laravel 5.0** rilis, 6 bulan kemudian akan rilis **Laravel 5.1**, lalu 6 bulan berikutnya **Laravel 5.2**, dst hingga versi 5.8.

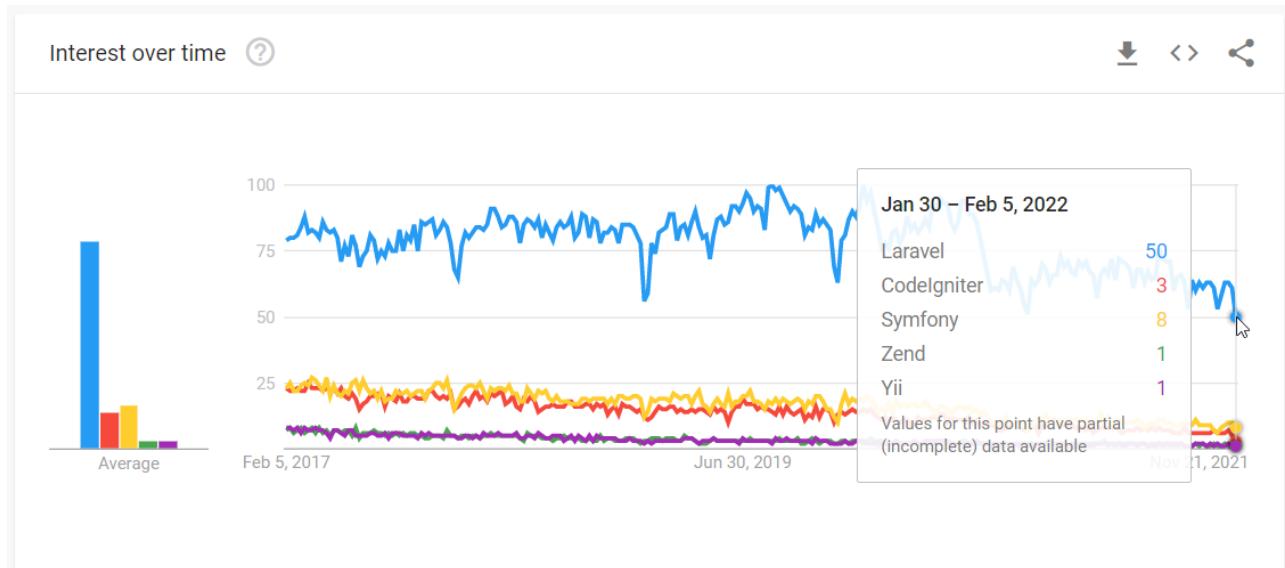
Namun mulai dari **Laravel 6**, terdapat perubahan sistem penomoran. Saat ini Laravel memakai semantic versioning, dimana digit kedua berfungsi untuk update minor yang bisa saja rilis setiap minggu atau bulan (seperti Laravel 9.1, Laravel 9.2, dst). Update minor ini tidak banyak mengubah fitur yang sudah ada, lebih ke perbaikan bug atau penambahan fitur-fitur kecil.

1.3. Kenapa Harus Laravel?

Di awal bab telah kita singgung bahwa penggunaan framework bisa mempercepat pembuatan sebuah aplikasi. Namun di luar sana tersedia cukup banyak pilihan framework PHP. Jadi kenapa harus menggunakan **Laravel**?

Bagi saya pribadi, jawaban singkatnya adalah karena Laravel merupakan **framework PHP paling populer** saat ini.

Berikut hasil Google Trends untuk 5 framework PHP dalam 5 tahun terakhir (2017 – 2022):



Gambar: Perbandingan hasil Google Trends 5 framework PHP (sumber: trends.google.com)

Google Trends menggambarkan apa yang paling banyak di search dari kotak pencarian Google. Terlihat bahwa Laravel sangat mendominasi jika dibandingkan dengan 4 framework PHP lain, yakni **CodeIgniter**, **Symfony**, **Zend** dan **Yii**.

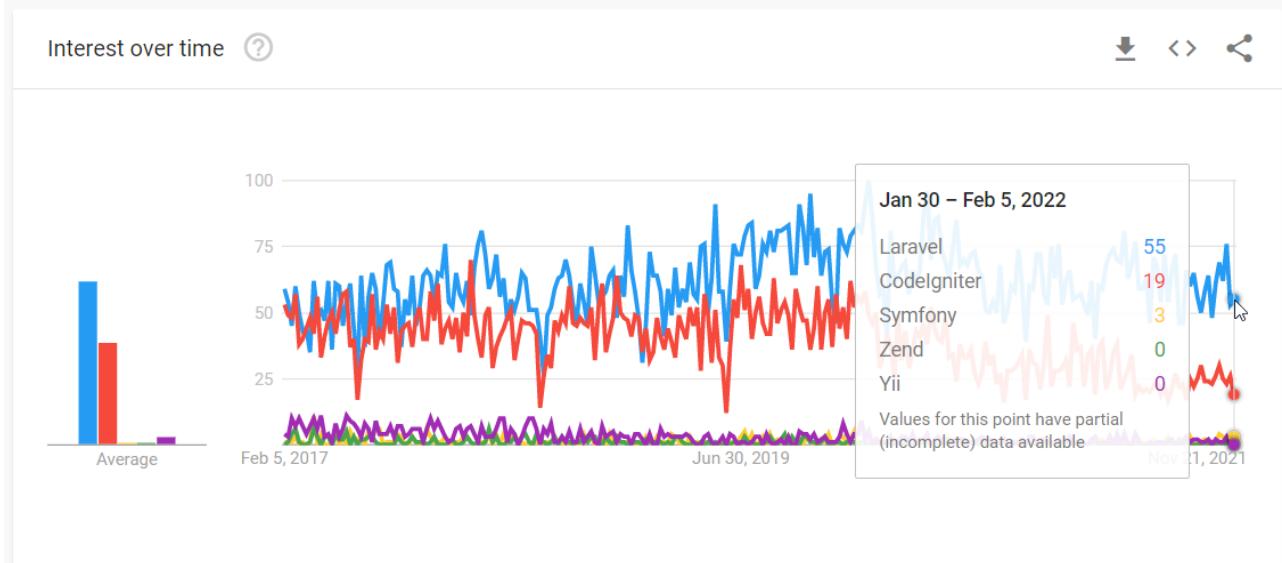
Meskipun tidak bisa dibuktikan secara langsung, tapi semakin populer sebuah teknologi maka bisa dikatakan teknologi tersebut juga memiliki fitur yang relatif baik.

Hasil Google Trends ini bisa saja berasal dari blunder atau hal negatif yang membuat orang banyak mencari keyword tersebut, tapi kasus seperti itu biasanya hanya berefek singkat, tidak dalam waktu tahunan seperti grafik di atas.

Kepopuleran Laravel juga bisa dilihat dari kebutuhan industri. Jika anda mencari lowongan kerja programmer back-end, maka besar kemungkinan lowongan tersebut mensyaratkan

harus menguasai PHP hingga framework Laravel. Ini juga salah satu alasan untuk belajar Laravel.

Sebagai info tambahan, grafik di atas adalah tingkat popularitas framework PHP di seluruh dunia. Bagaimana dengan di Indonesia? Hasilnya jadi lebih menarik:



Gambar: Perbandingan hasil Google Trends 5 framework PHP di Indonesia (sumber: trends.google.com)

Terlihat 2 garis yang saling mengejar, yakni garis biru dan garis merah. Artinya, di Indonesia terdapat 2 buah framework PHP yang sama-sama populer: **Laravel** dan **CodeIgniter**. Meskipun di luar negeri Laravel sangat mendominasi, tapi di Indonesia CodeIgniter masih jadi salah satu pilihan framework PHP.

Jika anda memutuskan fokus ke bidang PHP back-end, maka bisa pelajari 2 framework ini: **Laravel** dan **CodeIgniter**. Lowongan kerja keduanya cukup besar. Dari grafik di atas, popularitas CI terlihat terus menurun dalam beberapa tahun terakhir, namun tetap ada perusahaan yang butuh CI terutama untuk me-maintenance kode-kode lama.

Bagaimana dengan framework lain seperti **Symfony**, **Yii**, atau **Zend**? Kita tidak bisa mengatakan kalau framework ini tidak bagus karena pada dasarnya setiap framework punya kelebihan dan kekurangan masing-masing. Hanya kebetulan saja framework tersebut kalah populer dari Laravel dan CodeIgniter. Ini sedikit miris karena di dalam Laravel banyak menggunakan komponen yang berasal dari framework Symfony.

Tujuan akhir dari penggunaan framework adalah agar kerjaan cepat selesai, tidak masalah ingin dibuat dengan framework apa saja. Beberapa perusahaan software juga membuat framework khusus yang tidak di-publish.

Web perusahaan besar seperti **Tokopedia** atau **Bukalapak** sangat mungkin menggunakan framework sendiri, karena mereka punya kebutuhan khusus yang bisa jadi belum tersedia di Laravel atau Code Igniter.

Meskipun nantinya kita bisa saja tidak menggunakan framework, namun pemahaman alur kerja framework tetap bermanfaat. Framework modern dirancang untuk skala industri dan mengikuti standar terbaik dalam pembuatan aplikasi. Kita bisa belajar banyak dalam mengelola kode program dengan tingkat kerumitan menengah ke atas.

1.4. Framework vs PHP Native

Menggunakan framework memang akan mempersingkat waktu pembuatan aplikasi, namun tidak semua project cocok memakai framework. Perdebatan apakah sebaiknya memakai framework atau tidak (yakni menggunakan kode "PHP saja" atau dikenal sebagai "PHP Native") menjadi perdebatan abadi di berbagai forum diskusi.

Ada beberapa alasan kenapa kita sebaiknya tidak menggunakan framework:

- x **Aplikasi yang dibuat cukup sederhana**

Meskipun bisa, tapi kurang pas jika menggunakan framework hanya untuk membuat program menghitung luas segitiga. Untuk yang seperti ini sebaiknya pakai PHP native saja karena akan jauh selesai lebih cepat.

- x **Belum memiliki dasar web programming yang cukup**

Ini sering terjadi terutama bagi pemula yang ingin cepat-cepat masuk ke framework dengan melewatkannya banyak materi dasar. Framework termasuk materi PHP tingkat *advanced* yang perlu dasar yang kuat. Tidak hanya PHP saja, tapi juga materi dasar lain seperti HTML, CSS, MySQL dan JavaScript.

Sebagai contoh, di Laravel nanti ada yang namanya **Eloquent ORM**, yakni sebuah *abstraction layer* untuk mengakses database. Di sini kita tidak lagi menggunakan query MySQL seperti `SELECT`, `INSERT` atau `JOIN`, tapi cukup memakai fitur dari Eloquent. Jika tidak memiliki basic di perintah SQL, bagian ini akan terasa susah untuk di pahami.

- x **Ingin mengejar performa**

Sebuah framework PHP umumnya terdiri dari ratusan hingga ribuan file PHP yang saling bekerja sama. Bahkan Laravel 9 terdiri dari sekitar 7.100 file PHP dengan total ukuran 44MB. Memang tidak semua file ini terpakai untuk setiap project, tapi performa yang dihasilkan mungkin bisa lebih tinggi jika dibuat tanpa framework.

Namun juga bukan berarti tanpa framework otomatis aplikasi akan berjalan dengan lebih cepat. Hal ini bergantung keahlian kita dalam meramu kode program PHP.

Performa juga merupakan salah satu indikator terpenting dari keberhasilan sebuah framework. Tim dibalik Laravel tentu selalu berusaha untuk mendapatkan performa terbaik.

Kemudian apa alasan sebaiknya menggunakan framework?

✓ **Tersedia fitur siap pakai**

Framework sudah menyediakan berbagai komponen siap pakai untuk membantu kita dalam merancang aplikasi. Sebagai contoh, di Laravel dengan 1 perintah sederhana kita bisa meng-generate form register lengkap dengan fitur login dan logout. Jika menggunakan PHP native, membuat fitur ini bisa butuh waktu sehari penuh.

Tidak hanya itu, umumnya framework PHP menyediakan cara singkat untuk membuat fitur-fitur lain, seperti pembuatan form, validasi, menampilkan pesan error, mengakses database, pembuatan layout, dsb.

✓ **Mengikuti best practice**

Framework memiliki aturan penulisan baku yang tidak bisa diubah. Dengan demikian kita akan "dipaksa" mengikuti cara penulisan framework. Cara penulisan ini sudah dipikirkan oleh ribuan programmer profesional yang merancang framework tersebut.

Sebagai contoh, sebagian besar framework PHP menggunakan konsep **M-V-C**, yakni singkatan dari **Model**, **View** dan **Controller**. Konsep MVC bertujuan untuk memisahkan 3 bagian program: kode untuk mengakses database (disebut sebagai *model*), kode untuk tampilan (*view*) dan kode untuk mengatur alur logika program (*controller*). Dengan pemisahan ini, aplikasi kita menjadi lebih rapi dan mudah di kelola.

Bagi pemula, pembagian ini tampak merepotkan. Karena untuk sekedar membuat tampilan 1 halaman saja, kita butuh mengedit minimal 3 file serta mengatur berbagai konfigurasi. Namun dalam jangka panjang, MVC sangat memudahkan pengelolaan website terutama untuk project besar.

✓ **Mudah untuk kolaborasi**

Menggunakan framework juga memudahkan pembuatan kode program yang dibuat oleh tim. Framework memiliki aturan penulisan yang sudah baku sehingga setiap anggota tim bisa dengan mudah membaca alur kode program yang dibuat oleh programmer lain. Misalnya jika ada masalah dengan database, maka problem utama kemungkinan besar ada di *model* (bagian 'M' dari MVC).

Selain itu, konsep MVC juga membuat pemisahan antara front-end dan back-end yang lebih baku. Tim front-end yang mengembangkan design web bisa fokus ke sisi tampilan saja (*view*), sedangkan tim back-end bisa fokus ke alur logika aplikasi (*controller* dan *model*).

✓ **Mudah membaca kode program**

Konsep yang sudah baku juga bermanfaat jika ada pergantian anggota tim. Tanpa framework, butuh waktu lama bagi programmer baru untuk memahami kode program

yang sudah ada. Sangat mungkin konsep berfikir programmer sebelumnya berbeda jauh dengan programmer yang akan menggantikannya.

Namun jika aplikasi tersebut dibuat dengan framework, setiap programmer harus mengikuti cara yang sudah diatur oleh framework tersebut. Misalnya kode program untuk pengaturan tampilan ada di View, pengaturan database ada di Model, dsb. Sehingga lebih mudah bagi programmer baru untuk melakukan modifikasi.

✓ Keamanan Aplikasi

Keamanan untuk sebuah aplikasi "real world" merupakan hal wajib, namun kadang kita tidak tau apa yang harus diamankan dan bagaimana membuat kode programnya.

Sebagai contoh, ada celah kelemahan web yang dikenal sebagai **CSRF** atau Cross-Site Request Forgery, yakni teknik mengisi form (seperti form login), dengan kode program yang bukan berasal dari website kita. Celah ini bisa dimanfaatkan untuk membuat bot atau program yang terus-menerus mencoba mengisi form login.

Mayoritas framework PHP (termasuk Laravel), sudah menyediakan cara untuk mengatasi masalah ini, yaitu dengan memaksa kita untuk membuat sebuah **CSRF token**. Laravel akan menampilkan error jika sebuah form tidak memiliki CSRF token. Cara input token ini juga sangat sederhana, hanya perlu satu perintah singkat.

Fitur keamanan seperti ini sudah lebih dahulu dipikirkan oleh tim pengembang Laravel, sehingga aplikasi yang kita buat relatif lebih aman.

Melihat plus dan minus sebuah framework, keuntungan yang didapat jauh lebih banyak. Saya tidak akan heran jika setelah menamatkan materi di buku ini, anda akan merasa "tergantung" dengan framework dan ingin membuat aplikasi dengan framework saja, tidak lagi dari nol menggunakan native PHP.

1.5. Code Igniter vs Laravel

Di Indonesia, framework PHP yang paling populer ada 2, yakni **Code Igniter** dan **Laravel**. Tidak jarang bagi yang ingin belajar framework bingung dengan kedua pilihan ini (mau belajar yang mana dulu?)

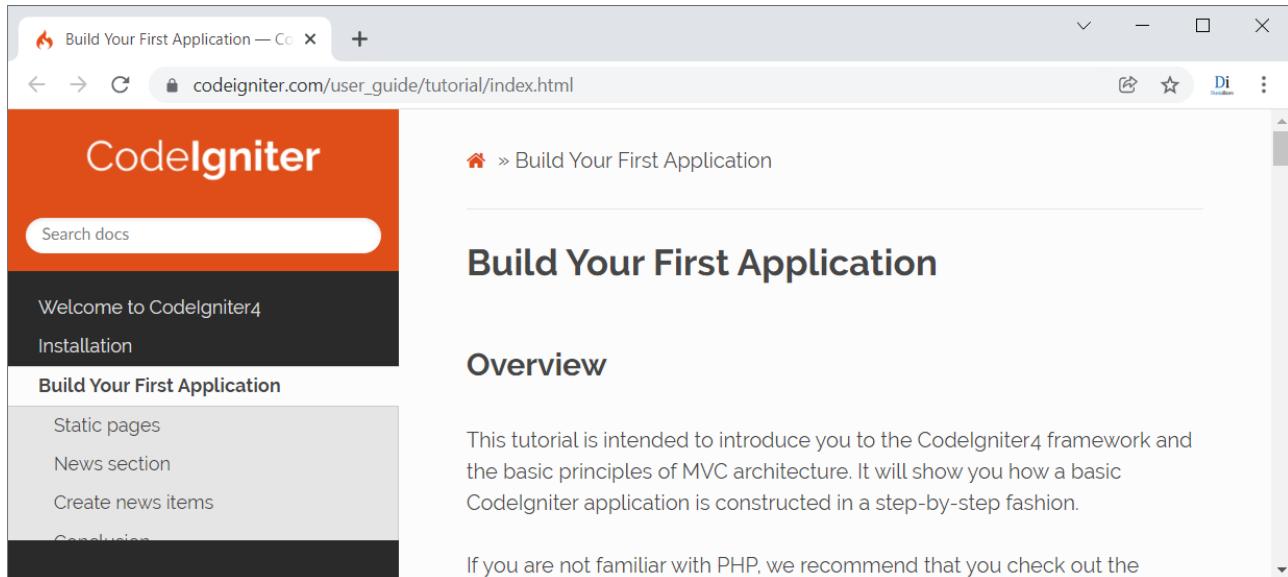
Setiap framework pada dasarnya punya kelebihan dan kekurangan masing-masing. Bagi saya, kesimpulannya tetap: "Selama bisa menyelesaikan project, pakai framework apapun tidak masalah". Namun tidak ada salahnya kita bahas sedikit perbedaan dari kedua framework ini.

Kemudahan Penggunaan

Secara umum Code Igniter lebih mudah dipelajari dan digunakan, terutama bagi pemula. Mungkin ini karena Code Igniter juga lebih sederhana dan masih kental dengan "PHP tradisional". Misalnya untuk bisa menggunakan CI 3, cukup download file asli dalam bentuk

zip, extract ke folder htdocs XAMPP, dan CI sudah bisa langsung dipakai.

Dokumentasi resmi Code Igniter juga sangat mudah diikuti dan tersedia tutorial step-by-step pembuatan aplikasi sederhana.



Gambar: Tutorial penggunaan Code Igniter dalam dokumentasi resminya

(codeigniter.com/user_guide/tutorial/index.html)

Sedangkan Laravel terkesan lebih modern sekaligus agak sedikit rumit. Untuk bisa menggunakan Laravel, web resminya tidak menyediakan link download file. Kita harus menginstall **composer** terlebih dahulu, kemudian mengetik perintah install dari cmd (*command prompt*).

Selama pembuatan aplikasi, perintah cmd ini juga akan sering diakses, yang bagi sebagian pengguna Windows terlihat agak menakutkan. Yup, di Laravel kita akan bergantian mengetik kode program di teks editor, serta mengakses perintah-perintah Laravel dari cmd.

Dokumentasi resmi Laravel juga relatif susah dipelajari bagi pemula, karena tidak disediakan tutorial singkat cara penggunaannya secara berurutan. Laravel juga banyak menggunakan istilah-istilah yang relatif asing seperti *facade*, *artisan*, *migrate*, atau *middleware*.

Jadi, *learning curve* (proses belajar) untuk Code Igniter lebih mudah dibandingkan Laravel.

Kecepatan Update

Di sisi lain, Laravel juga sangat update. Seperti yang kita bahas sebelumnya, jadwal rilis Laravel saat ini setiap 1 tahun (dulu bahkan 6 bulan sekali). Hasilnya, Laravel bisa menerapkan konsep programming terkini dengan sangat cepat. Jika PHP menambah fitur baru, beberapa saat kemudian akan di implementasikan oleh Laravel.

Namun ini juga bikin repot karena belum selesai kita belajar versi lama, eh sudah keluar Laravel yang lebih baru.

Akhir-akhir ini perubahan fitur di setiap update Laravel memang tidak terlalu drastis, tapi tetap saja ada 1 atau 2 hal yang tidak akan jalan. Ketika menemui kesulitan dan mencarinya di internet, harus dilihat juga apakah itu bisa jalan di versi Laravel saat ini atau tidak.

Sedangkan untuk Code Igniter, update ini berjalan sedikit lambat. Pada akhir 2019 versi stabil Code Igniter adalah versi 3.1.11, dimana mayor update dari 3.0 ke 3.1 terakhir terjadi di Juli 2016. Artinya, antara 2016 - 2019 tidak banyak perubahan di CI. Sepanjang 3 tahun itu hanya ada update minor yang lebih ke perbaikan bug.

Hal ini juga terlihat untuk mayor update CI berikutnya, yakni **Code Igniter 4**. Code Igniter 4 versi alpha sudah diumumkan sejak Desember 2018, namun butuh waktu 1 tahun lebih untuk full release di akhir Februari 2020. Jika berdasarkan pada aspek historis ini, maka update mayor ke CI 5 belum akan hadir dalam beberapa tahun ke depan.

Lambatnya proses update ini membuat fitur di CI seolah-olah "terkunci" sejak tahun 2016. Ini mendatangkan keuntungan tersendiri karena mayoritas programmer CI masih memakai versi yang sama.

Di sini, meski menghadapi tantangan agar terus mengikuti perkembangan, framework Laravel terlihat lebih menarik. Kecuali anda memang lebih menyukai framework yang tidak banyak perubahan dan tidak masalah dengan fitur-fitur lama.

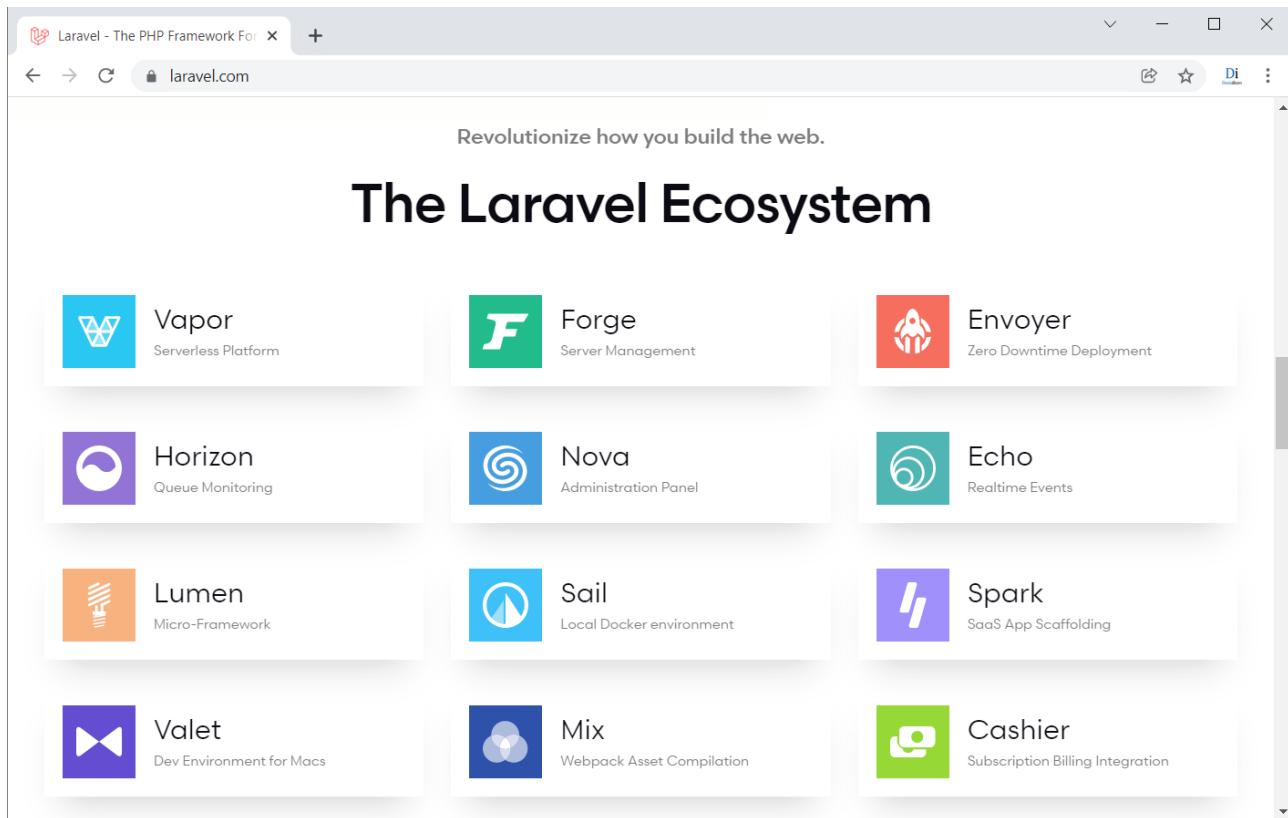
Dukungan Pengembang

Lambatnya proses update Code Igniter menurut saya juga dikarenakan faktor tidak adanya perusahaan komersial yang benar-benar bisa mendukung pengembangan CI. Bahkan pada 2013, CI sempat hampir mati karena perusahaan **EllisLab** selaku pengembang utama merasa tidak memiliki sumber daya lagi untuk mengelola CI: [EllisLab Seeking New Owner for CodeIgniter](#).

Untungnya **British Columbia Institute of Technology**, sebuah universitas teknologi asal Kanada, tertarik melanjutkan pengembangan CI dimulai dari tahun 2014 hingga sekarang. Namun tetap saja ini bukanlah sebuah perusahaan, sehingga pengembangan CI lebih banyak secara sukarela dari komunitas.

Berbeda halnya dengan Laravel yang dari awal sudah memiliki cara untuk "mendatangkan uang". Misalnya, terdapat [Laracasts](#) sebuah layanan kursus online yang berisi berbagai video tutorial seputar Laravel (berbayar sekitar \$15 per bulan), ada juga [Forge](#) dan [Laravel Vapor](#), semacam hosting berbayar untuk aplikasi yang dibuat dengan Laravel. [Laravel Nova](#), sebuah halaman admin untuk memudahkan pengelolaan aplikasi CRUD, dan masih banyak lagi.

Semua produk premium ini disebut Laravel sebagai "**The Laravel Ecosystem**", yang jumlahnya juga cukup banyak.



Gambar: Berbagai layanan premium dari Laravel

Di sini kita bisa saja khawatir dengan banyaknya layanan berbayar dari Laravel, karena sangat mungkin produk gratis nya (framework Laravel itu sendiri) banyak yang disunat. Namun kenyataannya tidak seperti itu, fitur di framework Laravel sangat banyak. Apa yang akan kita bahas dalam buku ini juga baru *segelintir* dari semua fitur yang tersedia di Laravel.

Layanan berbayar yang disediakan Laravel secara khusus ditujukan untuk perusahaan atau kalangan profesional yang ingin mendapat dukungan teknis dari pengembang framework secara langsung. Harapannya, perusahaan besar juga nyaman menggunakan Laravel.

Bagi kita sebagai pengguna framework, ini semua menjadi "jaminan" bahwa Laravel akan tetap ada dan terus dikembangkan. Jika tidak, semua layanan berbayar itu tidak banyak yang menggunakan. Dengan banyaknya perusahaan yang memakai Laravel, maka kebutuhan programmer dengan skill Laravel juga meningkat. Hasilnya, komunitas Laravel juga terus bertambah.

Produk premium seperti ini tidak tersedia (atau tepatnya *belum tersedia*) di web resmi Code Igniter. Tapi mudah-mudahan Code Igniter 4 bisa melanjutkan eksistensi CI sebagai salah satu framework PHP populer.

Jadi, Lebih Baik Belajar Laravel?

Jika patokannya ke masa depan, saya merasa prospek sebagai programmer Laravel lebih cerah dibandingkan Code Igniter, terutama karena masifnya dukungan update. Namun tetap saja

tujuan akhir adalah selama aplikasi jadi dan client senang dengan aplikasi yang kita buat, tidak masalah mau menggunakan framework apa saja.

Alur belajar Laravel sendiri juga lebih rumit daripada Code Igniter, sehingga tidak jarang ada yang kurang suka dengan Laravel dan lebih memilih Code Igniter yang tidak terlalu sering update. Ini juga tidak masalah karena kesimpulan akhirnya lebih ke selera masing-masing.

Tapi karena anda sudah membeli buku ini, maka sudah saatnya kita lanjut masuk ke **Dunia Laravel**.

Dalam bab ini kita telah membahas pengertian dasar seperti apa itu framework PHP, sejarah singkat Laravel, plus minus penggunaan framework, serta perbandingan dari 2 framework PHP populer: Code Igniter dan Laravel.

Sebelum mulai praktik menginstall Laravel, mari belajar sejenak tentang konsep **MVC**. MVC sendiri menjadi arsitektur utama dari hampir semua framework PHP (tidak hanya Laravel saja). Pemahaman MVC di awal ini akan memudahkan kita dalam mempelajari alur kerja Laravel.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

2. Arsitektur MVC

Sebelum masuk ke praktik penggunaan Laravel, saya ingin membahas konsep dasar yang dipakai oleh Laravel (dan juga banyak framework PHP lain), yakni arsitektur **MVC**.

Materi kali ini memang lebih banyak tentang teori, tapi sangat penting agar bisa memahami "*kode apa saja yang kita tulis ke dalam Laravel*".

2.1. Pengertian MVC

MVC adalah sebuah arsitektur perancangan kode program. Tujuannya untuk memecah kode program utama menjadi 3 komponen terpisah dengan tugas yang spesifik. Ketiga komponen tersebut adalah:

- Pengaksesan database, disebut sebagai **Model**
- Tampilan design (user interface), disebut sebagai **View**
- Alur logika program, disebut sebagai **Controller**

Model-View-Controller inilah yang disingkat sebagai **MVC**.

MVC tidak hanya dipakai untuk membuat website, tapi juga aplikasi komputer secara umum. Konsep MVC sendiri sudah hadir sejak tahun 1970 yang di populerkan oleh [Trygve Reenskaug](#), seorang ilmuwan komputer asal Norwegia. Ketika itu MVC diterapkan memakai bahasa pemrograman **Smalltalk**.

Ide awal dari perlunya konsep MVC adalah agar aplikasi yang dibuat bisa mudah dikelola dan dikembangkan, terutama untuk aplikasi besar.

Sebagai contoh, seorang web designer bisa fokus merancang bagian **View** saja, yakni tampilan design website yang terdiri dari kode HTML dan CSS plus sedikit JavaScript. Kode program untuk berkomunikasi dengan database bisa ditangani oleh programmer yang secara khusus bagian **Model**. Serta programmer lain mengatur alur logika program di bagian **Controller**.

Dengan pemisahan seperti ini, kerja tim menjadi mudah dikelola. Selain itu setiap bagian tidak saling bergantung sama lain. Jika ada perubahan atau modifikasi, cukup edit di bagian yang diperlukan saja, tidak harus merombak ulang semua aplikasi.

Dibalik keunggulan ini, kendala utama dari konsep MVC adalah cukup rumit untuk dipahami (terutama bagi pemula), serta file kode program menjadi banyak karena setiap bagian dari M-

V-C harus ditulis dalam file terpisah. Namun keuntungan yang didapat sebanding dengan "usaha" untuk mempelajari MVC tersebut, karena kode program kita menjadi lebih fleksibilitas dan mudah dikelola.

2.2. Contoh Penggunaan MVC

Arsitektur **MVC** hanya menekankan pemisahan antara Model, View dan Controller. Penerapannya di setiap bahasa pemrograman bisa berbeda-beda karena lingkungan pemrograman desktop berbeda dengan pemrograman web.

Khusus untuk PHP, hampir semua framework PHP populer menggunakan arsitektur MVC, termasuk Laravel. Setiap framework juga menerapkan MVC dengan level "keketatan" yang berbeda-beda.

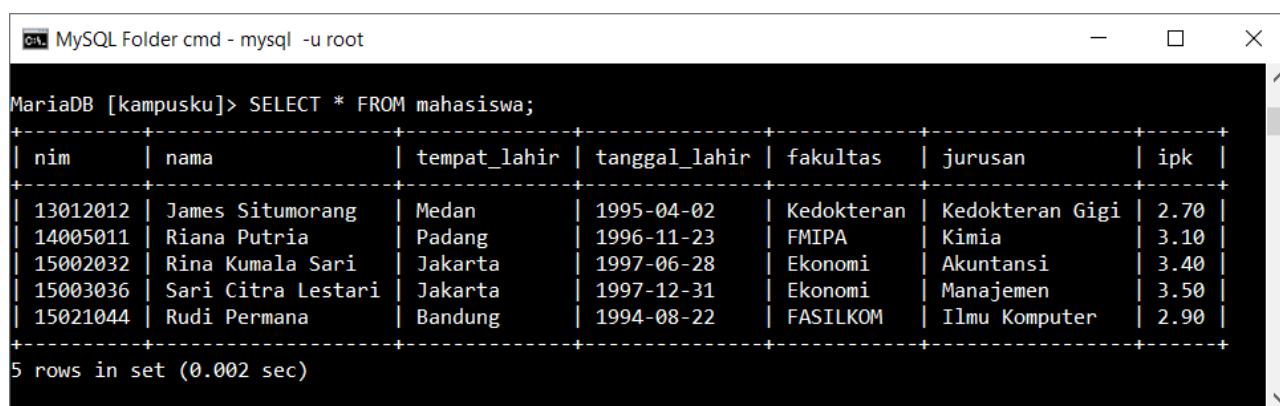
Agar lebih memahami peranan **Model**, **View** dan **Controller**, kita akan bahas dengan sebuah praktek sederhana tanpa framework. Contoh kasusnya adalah menampilkan isi tabel MySQL menggunakan PHP.

Sebenarnya ini bisa dibuat dengan 1 file PHP saja, dan sudah sering kita lakukan, yakni:

1. Buat koneksi ke MySQL dengan fungsi `mysqli_connect()`,
2. Ambil isi tabel menggunakan `mysqli_query()`,
3. Tampilkan hasilnya menggunakan perulangan `while` ke dalam tabel HTML.

Sebagai bahan praktek, saya akan memakai tabel **mahasiswa** di dalam database **kampusku**. Tabel ini merupakan tabel yang sama dari studi kasus bab CRUD buku **PHP Uncover**. Tabel apapun sebenarnya tidak masalah, karena tujuan kita hanya sekedar menampilkan isi tabel.

Berikut isi dan struktur tabel **mahasiswa**:



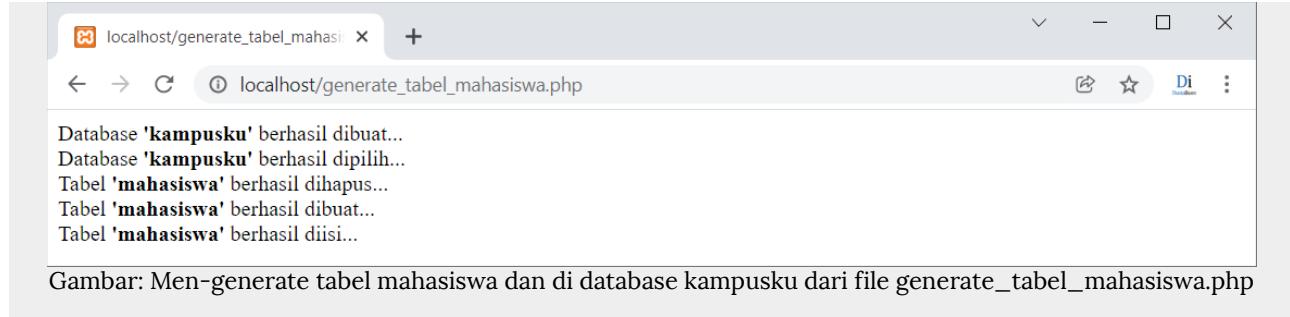
```

MySQL Folder cmd - mysql -u root
MariaDB [kampusku]> SELECT * FROM mahasiswa;
+-----+-----+-----+-----+-----+-----+-----+
| nim   | nama        | tempat_lahir | tanggal_lahir | fakultas | jurusan    | ipk   |
+-----+-----+-----+-----+-----+-----+-----+
| 13012012 | James Situmorang | Medan      | 1995-04-02  | Kedokteran | Kedokteran Gigi | 2.70 |
| 14005011 | Riana Putria   | Padang      | 1996-11-23  | FMIPA     | Kimia       | 3.10 |
| 15002032 | Rina Kumala Sari | Jakarta    | 1997-06-28  | Ekonomi    | Akuntansi   | 3.40 |
| 15003036 | Sari Citra Lestari | Jakarta    | 1997-12-31  | Ekonomi    | Manajemen   | 3.50 |
| 15021044 | Rudi Permana   | Bandung    | 1994-08-22  | FASILKOM  | Ilmu Komputer | 2.90 |
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)

```

Gambar: tampilan isi tabel mahasiswa

Kode program untuk membuat tabel **mahasiswa** bisa diakses dari [bab_02/generate_tabel_mahasiswa.php](#) yang tersedia di Google Drive.



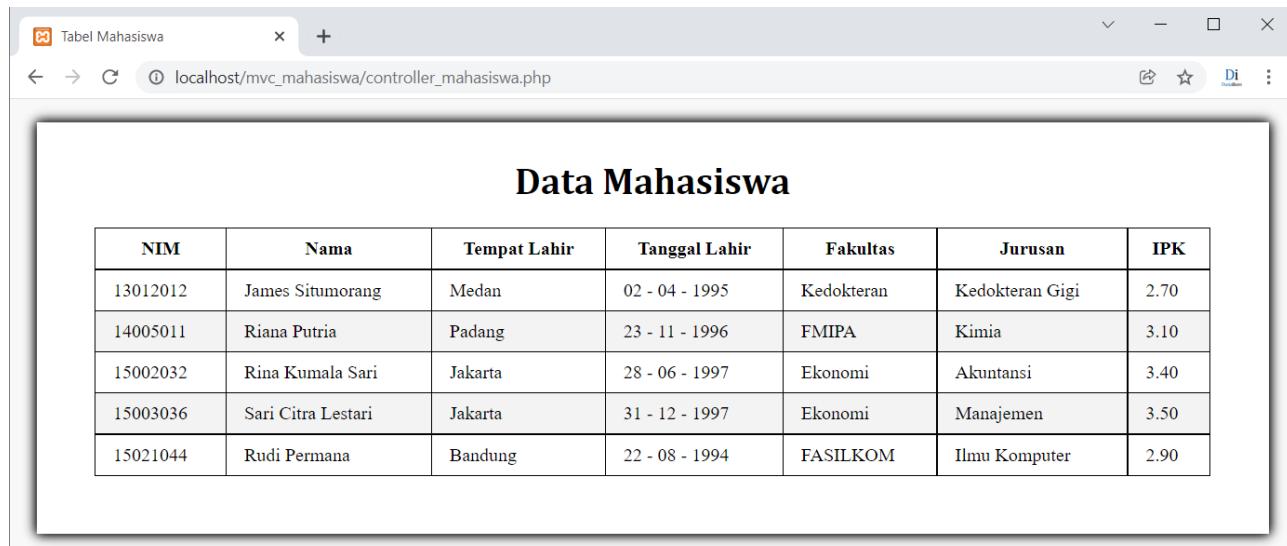
Gambar: Men-generate tabel mahasiswa dan di database kampusku dari file generate_tabel_mahasiswa.php

Dan berikut kode program yang saya rancang untuk menampilkan isi tabel menggunakan perpaduan kode HTML, CSS dan PHP:

tabel_mahasiswa.php

```
1 <?php
2     // buat koneksi dengan database mysql
3     $dbhost = "localhost";
4     $dbuser = "root";
5     $dbpass = "";
6     $dbname = "kampusku";
7     $link = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname);
8 ?>
9 <!DOCTYPE html>
10 <html lang="id">
11 <head>
12     <meta charset="UTF-8">
13     <title>Tabel Mahasiswa</title>
14     <style>
15         /* =====GLOBAL STYLE===== */
16         body {
17             background-color: #F8F8F8;
18         }
19         div.container {
20             width: 960px;
21             padding: 10px 50px 50px;
22             background-color: white;
23             margin: 20px auto;
24             box-shadow: 1px 0px 10px, -1px 0px 10px ;
25         }
26         h1 {
27             text-align: center;
28             font-family: Cambria, "Times New Roman", serif;
29             clear: both;
30         }
31
32         /* =====TABLE===== */
33         table {
34             border-collapse: collapse;
35             border-spacing: 0;
36             border: 1px black solid;
37             width: 100%
38         }
39         th, td {
```

```
40      padding:8px 15px;
41      border:1px black solid;
42  }
43  tr:nth-child(2n+3) {
44      background-color: #F2F2F2;
45  }
46  </style>
47 </head>
48 <body>
49 <div class="container">
50 <h1>Data Mahasiswa</h1>
51 <table border="1">
52 <tr>
53 <th>NIM</th>
54 <th>Nama</th>
55 <th>Tempat Lahir</th>
56 <th>Tanggal Lahir</th>
57 <th>Fakultas</th>
58 <th>Jurusan</th>
59 <th>IPK</th>
60 </tr>
61 <?php
// jalankan query
63 $query = "SELECT * FROM mahasiswa";
64 $result = mysqli_query($link, $query);
65
//buat perulangan untuk element tabel dari data mahasiswa
66 while($data = mysqli_fetch_assoc($result)) {
67     // konversi date MySQL (yyyy-mm-dd) menjadi dd-mm-yyyy
68     $tanggal_php = strtotime($data["tanggal_lahir"]);
69     $tanggal = date("d - m - Y", $tanggal_php);
70
71     echo "<tr>";
72     echo "<td>$data[nim]</td>";
73     echo "<td>$data[nama]</td>";
74     echo "<td>$data[tempat_lahir]</td>";
75     echo "<td>$tanggal</td>";
76     echo "<td>$data[fakultas]</td>";
77     echo "<td>$data[jurusan]</td>";
78     echo "<td>$data[ipk]</td>";
79     echo "</tr>";
80
81 }
82 ?>
83 </table>
84 </div>
85 </body>
86 </html>
```



The screenshot shows a Microsoft Edge browser window with the title bar 'Tabel Mahasiswa'. The address bar contains 'localhost/mvc_mahasiswa/controller_mahasiswa.php'. The main content area displays a table with the following data:

NIM	Nama	Tempat Lahir	Tanggal Lahir	Fakultas	Jurusan	IPK
13012012	James Situmorang	Medan	02 - 04 - 1995	Kedokteran	Kedokteran Gigi	2.70
14005011	Riana Putria	Padang	23 - 11 - 1996	FMIPA	Kimia	3.10
15002032	Rina Kumala Sari	Jakarta	28 - 06 - 1997	Ekonomi	Akuntansi	3.40
15003036	Sari Citra Lestari	Jakarta	31 - 12 - 1997	Ekonomi	Manajemen	3.50
15021044	Rudi Permana	Bandung	22 - 08 - 1994	FASILKOM	Ilmu Komputer	2.90

Gambar: tampilan tabel mahasiswa

Di bagian awal terdapat kode PHP untuk membuat koneksi dengan MySQL. Kemudian diikuti kode HTML dan CSS dari baris 9 – 60. Sebagian kode HTML ini dipakai untuk membuat struktur awal tag <table>. Di baris 61 – 82 kembali terdapat kode PHP untuk menampilkan setiap baris data isi tabel mahasiswa.

Saya yakin anda sudah bisa memahami kode program ini yang merupakan kode standar untuk menampilkan tabel MySQL menggunakan PHP. Untuk menyederhanakan kode program, saya tidak memakai fungsi pemeriksa kesalahan seperti `mysqli_error()`.

Di sini, ketiga bagian dari MVC masih "saling tercampur" satu sama lain. Maksudnya, kode program untuk tampilan (*view*), pengaksesan database (*model*) serta logika program (*controller*) semua ada di 1 file. Ini tidak masalah untuk program sederhana seperti contoh kita. Namun seiring kompleksitas website, penulisan campuran seperti ini susah di kelola.

Misalnya jika web designer ingin mengubah design tabel, ia harus hati-hati agar tidak merusak kode program PHP. Sebaliknya, jika ada perubahan pada kode PHP, si programmer juga harus memahami struktur design yang ada agar tidak merusak tampilan akhir.

Tugas kita sekarang adalah, mengkonversi kode program ini menjadi MVC sederhana, yakni memecah file di atas menjadi 3 bagian: 1 file sebagai model, 1 file sebagai view, dan 1 file sebagai controller.

Model

Model adalah bagian MVC yang mengurus database. Semua hal yang terkait database harus ditulis di dalam model.

Dalam contoh kita, kode yang berhubungan dengan database ada di proses pembuatan koneksi serta pengambilan isi tabel mahasiswa. Dua bagian kode program ini akan saya pisah ke dalam 1 file:

mvc_mahasiswa/model_mahasiswa.php

```

1 <?php
2 // buat koneksi dengan database mysql
3 function buatKoneksi(){
4     $dbhost = "localhost";
5     $dbuser = "root";
6     $dbpass = "";
7     $dbname = "kampusku";
8     return mysqli_connect($dbhost,$dbuser,$dbpass,$dbname);
9 }
10
11 // ambil tabel mahasiswa
12 function getTableMahasiswa() {
13     $link = buatKoneksi();
14     $query = "SELECT * FROM mahasiswa";
15     $result = mysqli_query($link, $query);
16
17     // ambil semua isi tabel ke dalam bentuk array 2 Dimensi
18     $hasil = mysqli_fetch_all($result, MYSQLI_ASSOC);
19     return $hasil;
20 }
```

File `model_mahasiswa.php` terdiri dari 2 buah function. Function pertama adalah `buatKoneksi()` di baris 3 – 9. Sesuai dengan namanya, fungsi ini dipakai untuk membuka koneksi dengan database MySQL, kemudian me-return hasil pemanggilan `mysqli_connect()` seperti yang terlihat di baris 8.

Fungsi kedua adalah `getTableMahasiswa()` di baris 13 – 19. Fungsi ini dipakai untuk mengambil semua isi dari tabel mahasiswa.

Di baris 13 terdapat pemanggilan fungsi `buatKoneksi()` yang disimpan ke dalam variabel `$link`. Lalu disambung dengan menjalankan query `"SELECT * FROM mahasiswa"` memakai fungsi `mysqli_query()` di baris 14 – 15.

Isi tabel mahasiswa saya ambil menggunakan fungsi `mysqli_fetch_all()` di baris 18. Fungsi ini akan mengembalikan semua isi tabel dalam bentuk array 2 dimensi. Inilah yang menjadi hasil dari fungsi `getTableMahasiswa()`.

Untuk menguji seperti apa hasil pemanggilan fungsi `getTableMahasiswa()`, bisa ditambahkan kode berikut di bagian bawah file:

mvc_mahasiswa/model_mahasiswa.php

```

1 <?php
2     function buatKoneksi(){
3         ...
4     }
5
6     function getTableMahasiswa() {
7         ...
8     }
```

```

9
10 echo "<pre>";
11 print_r(getTableMahasiswa());
12 echo "</pre>";

```

```

Array
(
    [0] => Array
        (
            [nim] => 13012012
            [nama] => James Situmorang
            [tempat_lahir] => Medan
            [tanggal_lahir] => 1995-04-02
            [fakultas] => Kedokteran
            [jurusan] => Kedokteran Gigi
            [ipk] => 2.70
        )

    [1] => Array
        (
            [nim] => 14005011
            [nama] => Riana Putria
            [tempat_lahir] => Padang
            [tanggal_lahir] => 1996-11-23
            [fakultas] => FMIPA
            [jurusan] => Kimia
            [ipk] => 3.10
        )
)

```

Gambar: Tampilan isi tabel mahasiswa dalam bentuk array 2 dimensi

Dari kode ini bisa terlihat bahwa file `model_mahasiswa.php` saya batasi hanya sampai mengambil isi tabel saja. Mengenai cara memproses array ini ke bentuk tabel HTML akan menjadi jatah bagian lain dari MVC, yakni **View**.

View

View adalah bagian MVC yang mengurus design tampilan. Mayoritas isi view berupa kode HTML, CSS dan JavaScript (jika dibutuhkan). Namun kadang juga terdapat sedikit kode PHP yang berisi data hasil pemrosesan dari bagian lain.

Untuk contoh kode program kita, berikut isi dari file view:

`mvc_mahasiswa/view_mahasiswa.php`

```

1  <!DOCTYPE html>
2  <html lang="id">
3  <head>
4      <meta charset="UTF-8">
5      <title>Tabel Mahasiswa</title>
6      <style>
7          /* =====GLOBAL STYLE===== */
8          body {
9              background-color: #F8F8F8;
10         }
11         div.container {
12             width: 960px;
13             padding: 10px 50px 50px;

```

```
14     background-color: white;
15     margin: 20px auto;
16     box-shadow: 1px 0px 10px, -1px 0px 10px ;
17 }
18 h1 {
19     text-align: center;
20     font-family: Cambria, "Times New Roman", serif;
21     clear: both;
22 }
23
24 /* =====TABLE===== */
25 table {
26     border-collapse: collapse;
27     border-spacing: 0;
28     border: 1px black solid;
29     width: 100%
30 }
31 th, td {
32     padding: 8px 15px;
33     border: 1px black solid;
34 }
35 tr:nth-child(2n+3) {
36     background-color: #F2F2F2;
37 }
38 </style>
39
40 </head>
41 <body>
42 <div class="container">
43 <h1>Data Mahasiswa</h1>
44 <table border="1">
45     <tr>
46         <th>NIM</th>
47         <th>Nama</th>
48         <th>Tempat Lahir</th>
49         <th>Tanggal Lahir</th>
50         <th>Fakultas</th>
51         <th>Jurusan</th>
52         <th>IPK</th>
53     </tr>
54 <?php
55     foreach ($isiTabelMahasiswa as $data ) {
56         // konversi date MySQL (yyyy-mm-dd) menjadi dd-mm-yyyy
57         $tanggal_php = strtotime($data["tanggal_lahir"]);
58         $tanggal = date("d - m - Y", $tanggal_php);
59
60         echo "<tr>";
61         echo "<td>$data[nim]</td>";
62         echo "<td>$data[nama]</td>";
63         echo "<td>$data[tempat_lahir]</td>";
64         echo "<td>$tanggal</td>";
65         echo "<td>$data[fakultas]</td>";
66         echo "<td>$data[jurusan]</td>";
67         echo "<td>$data[ipk]</td>";
68         echo "</tr>";
```

```
69      }
70      ?>
71    </table>
72  </div>
73 </body>
74 </html>
```

Seperti yang terlihat, dalam file `view_mahasiswa.php` masih terdapat sedikit kode PHP di baris 54 – 70. Bagian ini masih termasuk design tampilan (view) karena berisi kode untuk menampilkan isi array ke dalam tag `<table>` HTML.

Awal dari kode PHP ini berupa perulangan `foreach` di baris 55. Ini dipakai untuk menampilkan isi variabel `$isiTabelMahasiswa`. Programmer atau web designer yang merancang view cukup diberitahu bahwa variabel `$isiTabelMahasiswa` adalah array 2 dimensi. Selanjutnya, menjadi tugas web designer ingin menampilkan array ini dalam bentuk apa, misalnya menjadi tabel atau bisa juga dalam bentuk list.

Dalam penerapannya di framework PHP, biasanya ada yang disebut sebagai **template engine**. Template engine ini berfungsi untuk menampilkan isi variabel PHP dengan lebih singkat. Sebagai contoh, untuk menampilkan isi variabel `$data` di dalam tag `<p>` kita bisa menulis:

```
<p><?php echo $data ?></p>
```

Di Laravel, hal yang sama bisa dibuat dengan cara berikut:

```
<p>{{ $data }}</p>
```

Nantinya kita akan bahas dengan detail cara penggunaan *template engine* di Laravel.

Controller

Controller adalah bagian MVC yang berisi alur logika program dan juga sebagai penghubung antara model dengan view.

Dalam contoh kita, file model hanya berisi pendefinisian 2 buah function, yakni function `buatKoneksi()` dan `getTableMahasiswa()`. Kemudian di dalam view sudah langsung diawali dengan perulangan `foreach` untuk menampilkan isi dari variabel `$isiTabelMahasiswa`. Maka tugas controller adalah menghubungkan kedua file ini.

Berikut isi dari file controller:

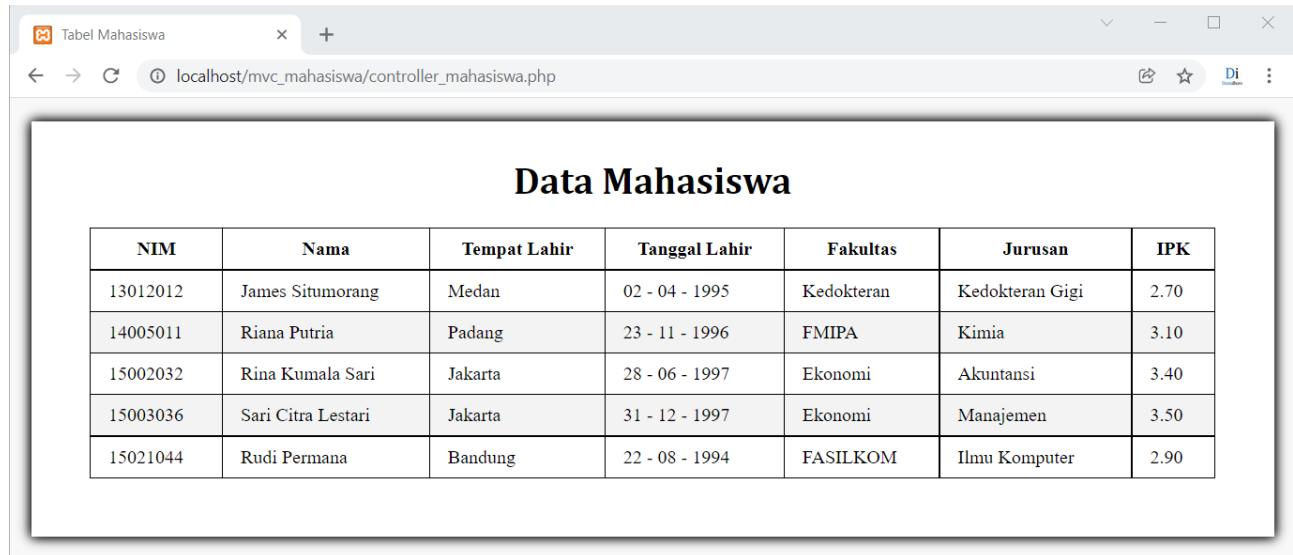
```
mvc_mahasiswa/controller_mahasiswa.php
```

```
1  <?php
2  include 'model_mahasiswa.php';
3  $isiTabelMahasiswa = getTableMahasiswa();
4  include 'view_mahasiswa.php';
```

Yup, hanya 3 baris.

Di baris 2 terdapat perintah untuk include file `model_mahasiswa.php` yang di dalamnya berisi pendefinisian fungsi `getTableMahasiswa()`. Fungsi ini kemudian dipanggil di baris 3 yang isinya di simpan ke dalam `$isiTabelMahasiswa`. Artinya, variabel `$isiTabelMahasiswa` akan berisi array. Terakhir di baris 4 saya meng-include file `view_mahasiswa.php`.

Dengan demikian, lengkap sudah proses pemisahan kode program menjadi MVC. Tempatkan ketiga file di dalam sebuah folder yang sama dan akses file `controller_mahasiswa.php`.



The screenshot shows a web browser window titled "Tabel Mahasiswa". The address bar indicates the URL is "localhost/mvc_mahasiswa/controller_mahasiswa.php". The main content area displays a table with the following data:

NIM	Nama	Tempat Lahir	Tanggal Lahir	Fakultas	Jurusan	IPK
13012012	James Situmorang	Medan	02 - 04 - 1995	Kedokteran	Kedokteran Gigi	2.70
14005011	Riana Putria	Padang	23 - 11 - 1996	FMIPA	Kimia	3.10
15002032	Rina Kumala Sari	Jakarta	28 - 06 - 1997	Ekonomi	Akuntansi	3.40
15003036	Sari Citra Lestari	Jakarta	31 - 12 - 1997	Ekonomi	Manajemen	3.50
15021044	Rudi Permana	Bandung	22 - 08 - 1994	FASILKOM	Ilmu Komputer	2.90

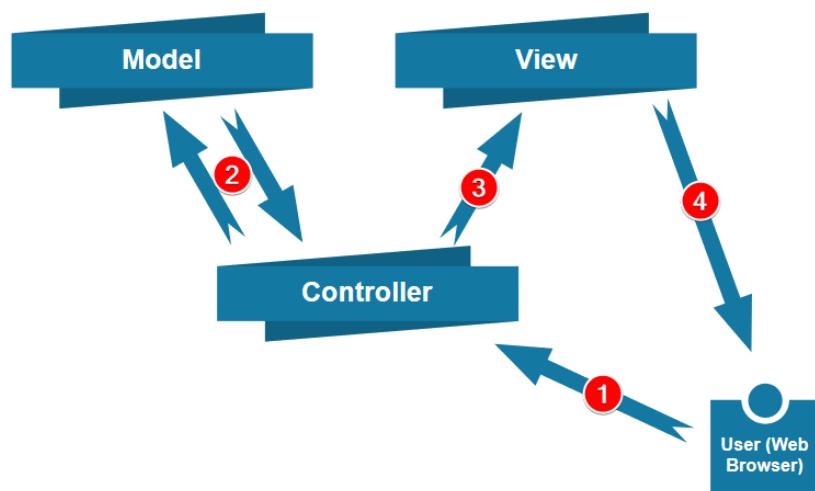
Gambar: Tampilan isi tabel mahasiswa menggunakan MVC

Dari sisi tampilan memang kita tidak bisa dilihat perbedaan apakah sebuah web menggunakan arsitektur MVC atau tidak, karena MVC ini posisinya ada di server, yakni memudahkan programmer dalam membuat dan menata kode program.

Contoh yang saya tampilkan ini merupakan versi sangat sederhana dari MVC. Jika anda sudah pernah mempelajari MVC sebelumnya, mungkin ada bagian yang kurang pas. Misalnya di dalam controller nyaris tidak berisi kode logika apapun, hanya sekedar include file dan pemanggilan fungsi. Tapi ini semata-mata agar kode kita tidak terlalu kompleks.

2.3. Diagram Alur MVC

Kita telah melihat contoh praktik sederhana dari **Model**, **View**, dan **Controller**. Dimana model berisi kode untuk mengakses database, view untuk design tampilan, serta controller sebagai logika dan penghubung antara model dan view. Jika dibuat dalam bentuk diagram, berikut alur proses dari sebuah arsitektur MVC:



Gambar: Diagram Alur MVC

Kita akan bahas mulai dari kanan bawah, yakni dari user yang membuka website kita menggunakan web browser.

Dalam panah 1, setiap interaksi yang dilakukan user akan ditangani oleh controller. Misalnya ketika user mengetik alamat situs www.duniaikom.com, maka sebuah controller di server duniaikom akan menangkap "request" tersebut. Atau ketika user selesai mengisi form register dan men-klik tombol submit, file controller akan menerima proses tersebut.

Pada contoh kita sebelum ini, tahap 1 adalah ketika user mengetik nama file `controller_mahasiswa.php` di web browser, kemudian menekan tombol Enter.

Controller pada dasarnya berisi logika program. Seandainya perlu mengambil data dari database, maka controller akan memanggil Model (panah nomor 2). Model inilah yang bertanggung jawab mengakses database lalu mengembalikan hasilnya kembali ke controller.

Dalam contoh kita, model ada di file `model_mahasiswa.php` yang berisi fungsi `getTableMahasiswa()`. Poin penting di sini adalah, **model hanya berkomunikasi dengan controller**.

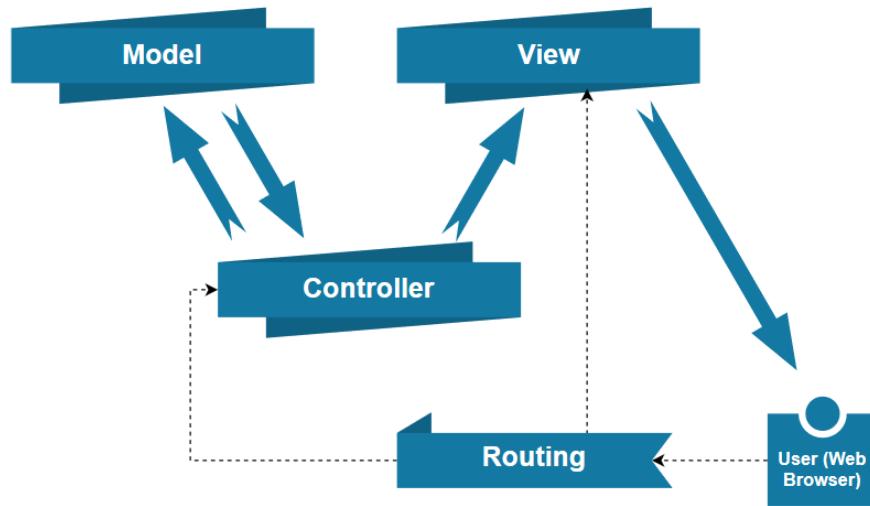
Setelah data dari model diterima kembali oleh controller, controller kemudian meneruskan data tersebut ke dalam View (panah 3). Data ini kemudian diproses sebagai kode HTML dan CSS di dalam view. Inilah yang dilihat oleh user di dalam web browser (panah 4).

Jika user men-klik halaman lain, maka itu akan diproses lagi oleh controller, yakni kembali ke langkah 1, demikian seterusnya alur kerja dari arsitektur MVC.

2.4. Mengenal Routing

Selain View, Model dan Controller, masih terdapat komponen lain yang hampir selalu ada di

dalam framework PHP, yakni **Routing** atau **Route**. Posisi routing ini berada di antara user dan controller, seperti diagram berikut:



Gambar: Diagram Alur MVC + Routing

Secara sederhana, routing berfungsi untuk "menyembunyikan" serta mempercantik nama file controller.

Dalam contoh kita sebelumnya, untuk menampilkan isi tabel mahasiswa, user harus mengetik lengkap nama file PHP, yakni `controller_mahasiswa.php`. Nama ini terasa kurang ideal dan akan lebih *user friendly* jika diganti menjadi `mahasiswa.php` atau `index.php`.

Dengan menggunakan routing, kita bisa melakukan hal tersebut. Misalnya jika yang diketik berupa `satu.php`, maka routing akan me-*redirect* ke file `controller_dosen.php`. Atau jika diketik file `pegawai.php`, maka akan di *redirect* ke `controller_pegawai.php`.

Di Laravel, routing juga bisa langsung dipakai untuk mengakses View. Jadi ketika yang diketik adalah `tiga.php`, bisa di rancang agar langsung *redirect* ke `view_siswa.php` tanpa harus melalui controller terlebih dahulu.

File routing pada dasarnya berisi daftar pasangan alamat URL dengan nama file controller atau view seperti contoh berikut:

```

satu.php -> controller_dosen.php
dua.php  -> controller_pegawai.php
tiga.php -> view_siswa.php
    
```

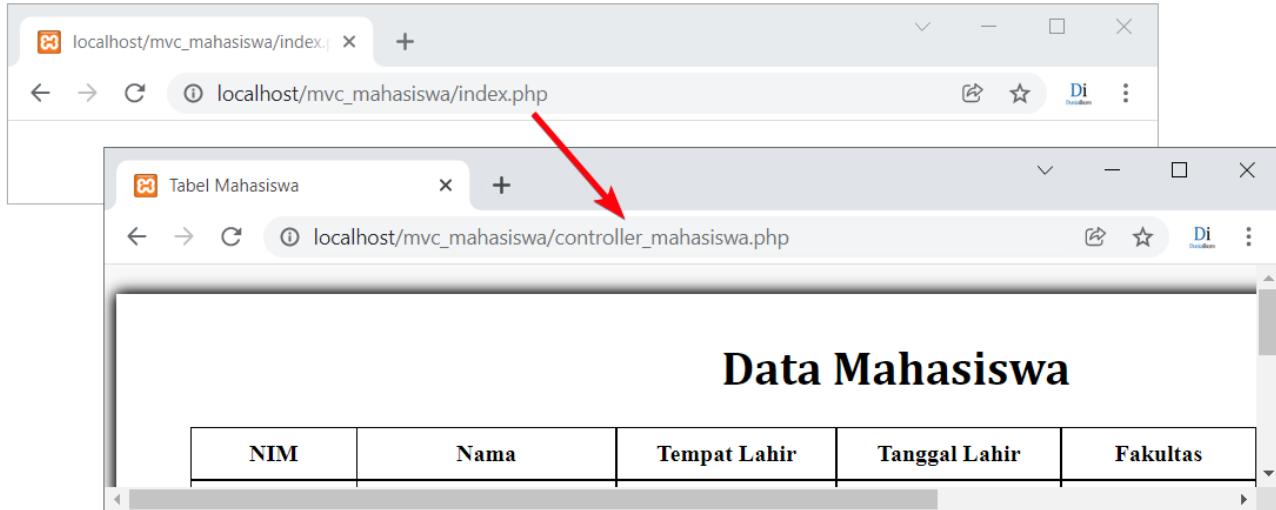
Untuk membuat fitur seperti ini caranya cukup kompleks, tapi kita bisa rancang versi yang sederhana. Yakni dengan membuat file `index.php` yang berisi kode berikut:

```

mvc_mahasiswa/index.php

1  <?php
2      header('Location: controller_mahasiswa.php');
    
```

Yup, hanya sebuah fungsi `header()`. Hasilnya, ketika file `index.php` di akses, maka halaman langsung pindah (*redirect*) ke `controller_mahasiswa.php`. Tempatkan file `index.php` ke folder yang sama dengan file MVC sebelumnya.



Gambar: Proses redirect dari index.php ke controller_mahasiswa.php

Pada prakteknya di Laravel, routing ini memiliki banyak fitur lain, termasuk membuat alamat URL dinamis.

Dalam bab ini kita telah membahas teori serta praktik **arsitektur MVC**. Meskipun contoh yang dipakai sangat sederhana, semoga bisa memberi bayangan tentang fungsi dari **Model**, **View** dan **Controller**. Konsep MVC sangat populer dan dipakai oleh banyak framework, termasuk Laravel dan juga Code Igniter.

Pada prakteknya nanti, 90% kode program yang akan kita tulis di dalam Laravel ada di 3 bagian ini saja: Model, View dan Controller.

Berikutnya kita akan masuk ke persiapan proses instalasi Laravel.

3. Instalasi Laravel

Dalam bab ini kita akan membahas cara instalasi Laravel serta aplikasi yang diperlukan, yakni **XAMPP**, teks editor (**VS Code**), dan **Composer**.

Karena keterbatasan perangkat, saya hanya membahas cara instalasi Laravel di sistem operasi Windows, terutama Windows 10 64-bit. Jika anda menggunakan Linux atau Mac OS, bisa cari panduan instalasinya di Google atau bisa ke sini:

- [How to install Laravel on Linux](#)
- [How to install Laravel on Mac OS](#)

3.1. Instalasi XAMPP

XAMPP merupakan aplikasi standar untuk memproses kode PHP secara local. Saya yakin aplikasi ini sudah tersedia di komputer anda. Jika butuh file installer XAMPP, bisa download dari web resminya di [apachefriends.org](#), kemudian install seperti biasa.

Laravel versi terbaru (versi 9 pada saat buku ini di revisi), butuh minimum PHP 8.0.2. Artinya kita harus menggunakan **XAMPP 8.0.2** ke atas. Saya sendiri akan memakai **XAMPP 8.1.4**.



Gambar: Tampilan XAMPP Control Panel

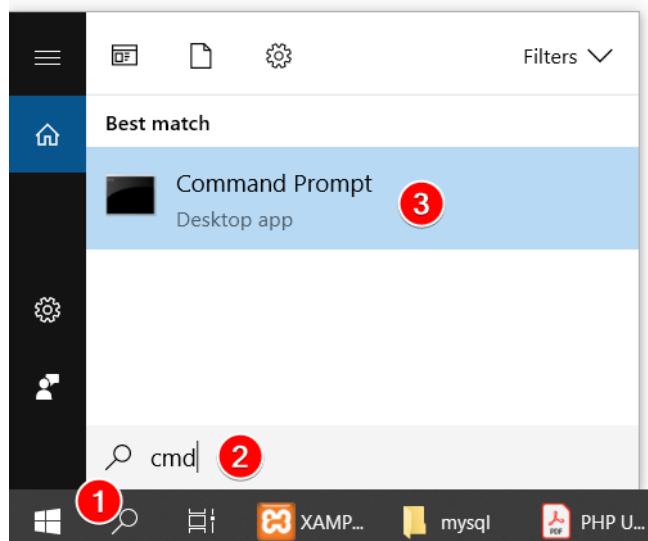
3.2. Mengakses PHP dari CMD

Secara "tradisional", kita biasa mengakses file PHP dari web browser (menggunakan XAMPP). Selain itu sebenarnya PHP juga bisa diakses dari **cmd** (*command prompt*), atau **terminal** kalau di Linux/Mac OS.

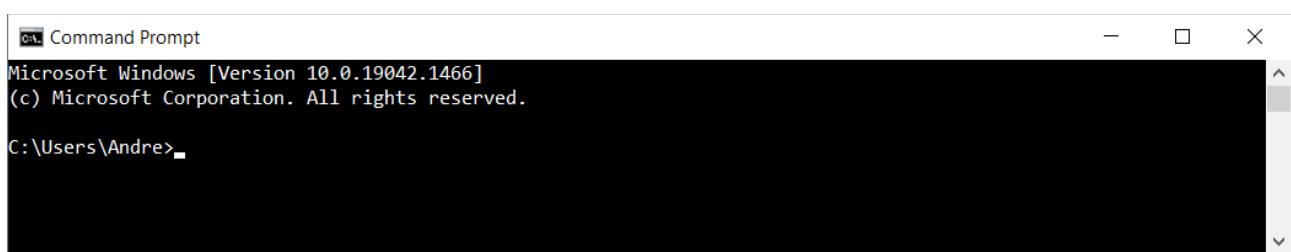
Bagi pemula, mengetik perintah di cmd terasa agak menakutkan, karena kita tidak bisa melihat apa-apa, hanya layar hitam tanpa tombol apapun yang bisa diklik. Namun cepat atau lambat, cmd akan jadi sahabat yang selalu menemani dalam proses pembuatan aplikasi di Laravel, karena Laravel banyak menggunakan fitur ini.

Untuk menjalankan PHP dari cmd, cukup buka aplikasi cmd kemudian masuk ke folder tempat file **php.exe** berada. Mari kita coba.

Pertama, silahkan buka **cmd** Windows. Untuk Windows 10, cara paling cepat adalah dengan klik tombol icon kaca pembesar (**search**) yang terletak di sebelah icon start menu (**tombol Windows**), lalu ketik **cmd** dan klik “**Command Prompt**“:



Gambar: Cara membuka aplikasi cmd

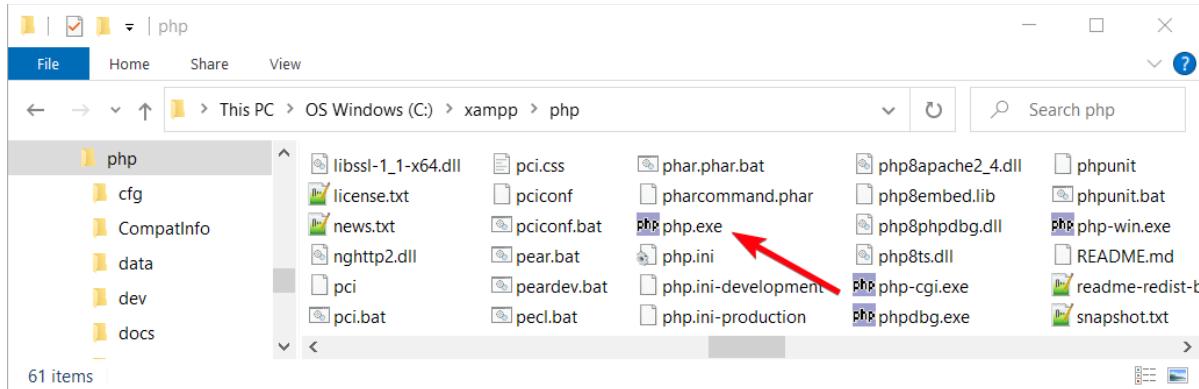


Gambar: Tampilan awal jendela cmd Windows

Jika dibuka dengan cara seperti ini, *directory* aktif cmd berada di `C:\Users\<nama_user_windows>`. Karena saya menggunakan nama user Andre, maka lokasinya ada di `C:\Users\Andre`. Artinya, cmd Windows saat ini aktif di folder `C:\Users\Andre`. Seluruh perintah yang akan diketik hanya berdampak ke folder ini saja.

Untuk mengakses file `php.exe`, kita harus pindah ke folder `C:\xampp\php`, yakni tempat instalasi PHP bawaan XAMPP berada. Jika anda menginstall XAMPP di lokasi lain, bisa disesuaikan saja.

Di dalam folder `C:\xampp\php` inilah terdapat file **`php.exe`** yang akan kita akses dari cmd:

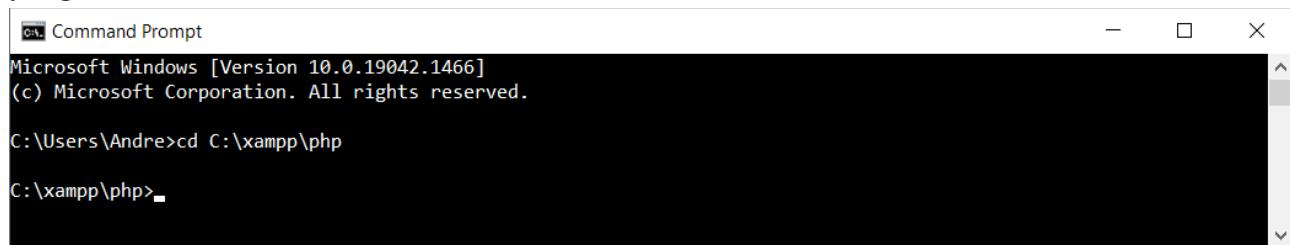


Gambar: Lokasi file `php.exe` di folder `C:\xampp\php`

Kembali ke cmd, ketik perintah berikut lalu tekan tombol Enter:

```
cd C:\xampp\php
```

Perintah **`cd`** merupakan singkatan dari **change directory**. Ini dipakai sebagai instruksi kepada cmd untuk menukar folder aktif. Hasilnya, teks di sisi kiri akan berubah menjadi `C:\xampp\php` yang menandakan kita sudah berada di dalam folder `C:\xampp\php`.

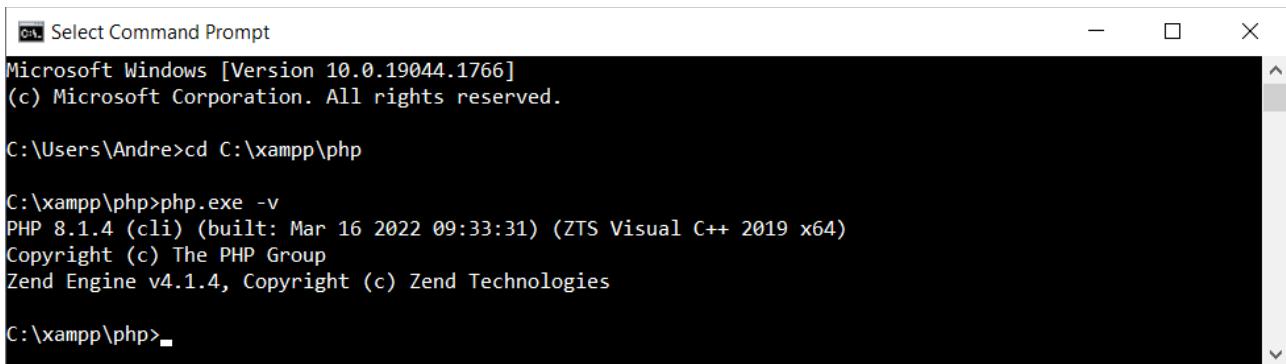


Gambar: Directory aktif sudah pindah ke `C:\xampp\php`

Sekarang mari coba akses file `php.exe` dari dalam cmd. Ketik perintah berikut dan akhiri dengan menekan tombol Enter:

```
php.exe -v
```

Ini artinya kita menjalankan perintah `-v` ke file `php.exe`. Perintah `-v` sendiri berfungsi untuk menampilkan versi PHP:



```
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>cd C:\xampp\php

C:\xampp\php>php.exe -v
PHP 8.1.4 (cli) (built: Mar 16 2022 09:33:31) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.4, Copyright (c) Zend Technologies

C:\xampp\php>
```

Gambar: Melihat versi php.exe

Dari teks yang tampil terlihat bahwa di komputer yang saya pakai, terinstall **PHP 8.1.1**. Hasil yang anda dapat bisa berbeda sesuai dengan versi PHP yang di install bersama XAMPP.

Di dalam cmd, akhiran .exe ini boleh tidak ditulis, sehingga hasil yang sama juga bisa di dapat dengan mengetik perintah berikut:

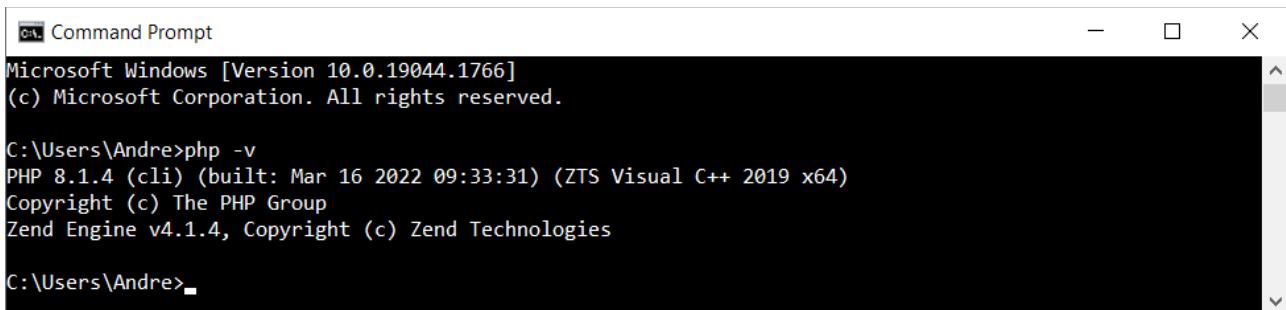
```
php -v
```

Sebagai contoh perintah lain, silahkan ketik `php -h` yang akan menampilkan teks *help* tentang perintah-perintah yang tersedia untuk file `php.exe`.

Menjalankan php.exe Secara Global

Apa yang baru saja kita praktekkan adalah menjalankan `php.exe` dari dalam folder tempat file tersebut berada. Agar lebih praktis, file `php.exe` ini bisa di set supaya dapat dipanggil secara global dari directory mana saja, tidak harus masuk terlebih dahulu ke folder `C:\xampp\php`.

Jika anda menginstall XAMPP yang relatif baru, file `php.exe` sebenarnya sudah bisa dipanggil secara global. Untuk uji coba, silahkan tutup cmd Windows kemudian buka kembali. Pastikan directory aktif tidak di `C:\xampp\php` (boleh dimana saja). Lalu langsung ketik `php -v`:



```
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

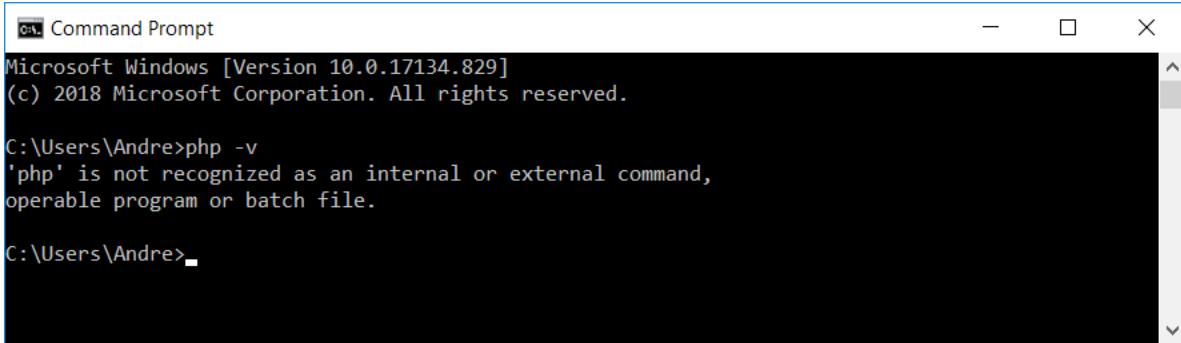
C:\Users\Andre>php -v
PHP 8.1.4 (cli) (built: Mar 16 2022 09:33:31) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.4, Copyright (c) Zend Technologies

C:\Users\Andre>
```

Gambar: Perintah `php -v` bisa diakses secara global

Jika terlihat versi PHP seperti tampilan di atas, maka berarti file `php.exe` sudah bisa diakses secara global.

Namun jika tampil pesan "`php` is not recognized as an internal or external command, operable program or batch file", artinya `php.exe` belum bisa diakses secara global:



```
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Andre>php -v
'php' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Andre>
```

Gambar: Perintah php -v tidak dikenal

Agar cmd bisa mengenal file `php.exe` secara global, kita harus mengubah sedikit konfigurasi sistem Windows, tepatnya di pengaturan PATH system variable.

Jika anda sudah berhasil menjalankan perintah di atas, boleh lewati bahasan berikutnya: "**Mengatur PATH system variable**". Namun saya sarankan untuk tetap membaca materi ini karena bersifat umum dan akan sering di pakai jika ada masalah dengan perintah cmd yang tidak dikenal.

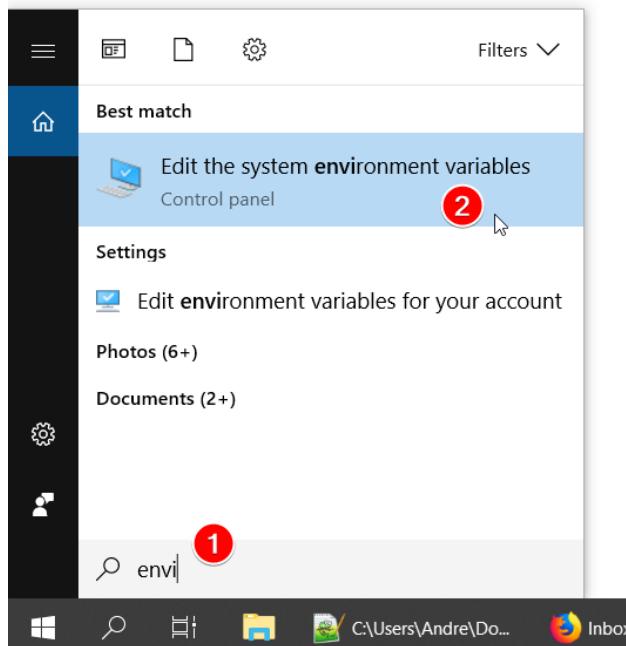
Mengatur PATH system variable

Dalam pengertian secara umum, **path** adalah istilah untuk menyebut alamat sebuah file di dalam *directory* komputer. Misalnya kita memiliki dokumen `index.php`. Alamat pathnya bisa saja tersimpan di `C:\xampp\htdocs\belajar_laravel\index.php`.

Tetapi jika disebut dengan "**mengatur path di sistem operasi Windows**", itu maksudnya kita menginput sebuah alamat folder agar **cmd** Windows bisa menjalankan aplikasi yang tersimpan dari mana saja.

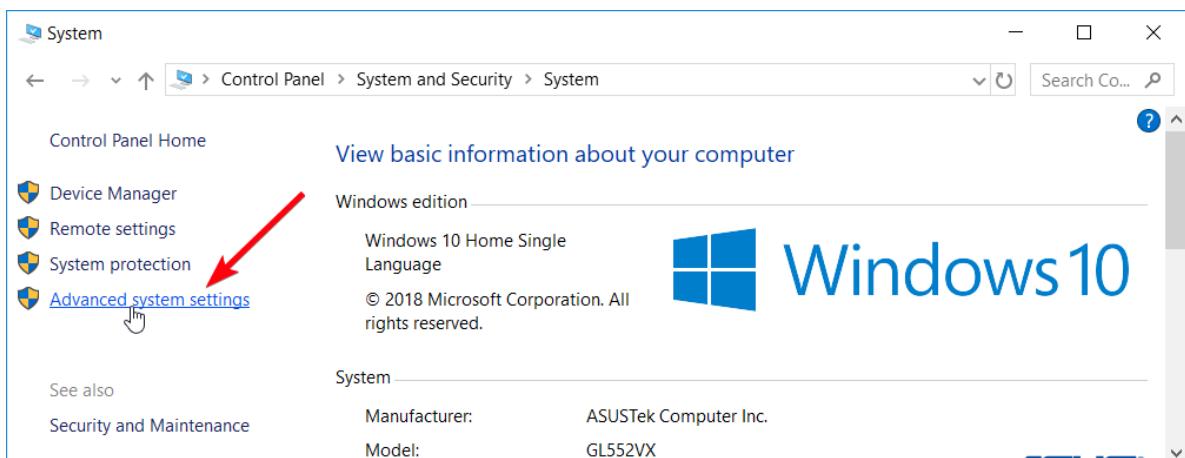
File `php.exe` berada di folder `C:\xampp\php`. Maka setiap kali mengakses file ini, sebenarnya kita harus membuka dahulu folder `C:\xampp\php`. Namun dengan mengatur alamat **path**, nantinya file `php.exe` bisa diakses dari alamat mana saja (bersifat global).

Untuk mengatur path, silahkan klik icon **search** di sebelah Start Menu (Logo Windows) lalu ketik "**environment**". Akan tampil pilihan "**Edit the system environment variables**", klik menu ini.



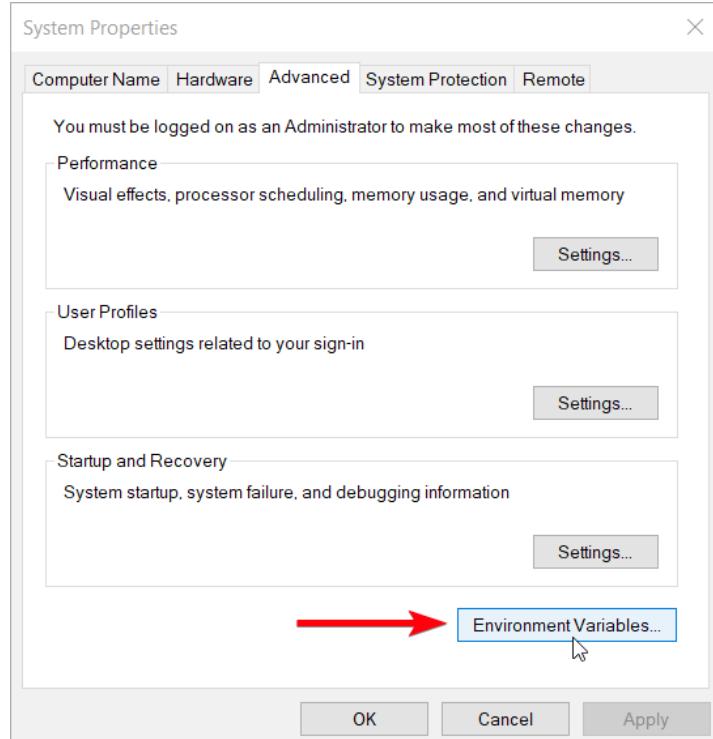
Gambar: Klik menu "Edit the system environment variables"

Cara lain bisa juga dari **Control Panel**, pilih **System and Security**, masuk ke menu **System**, lalu di menu bagian kiri klik "**Advanced System Settings**".

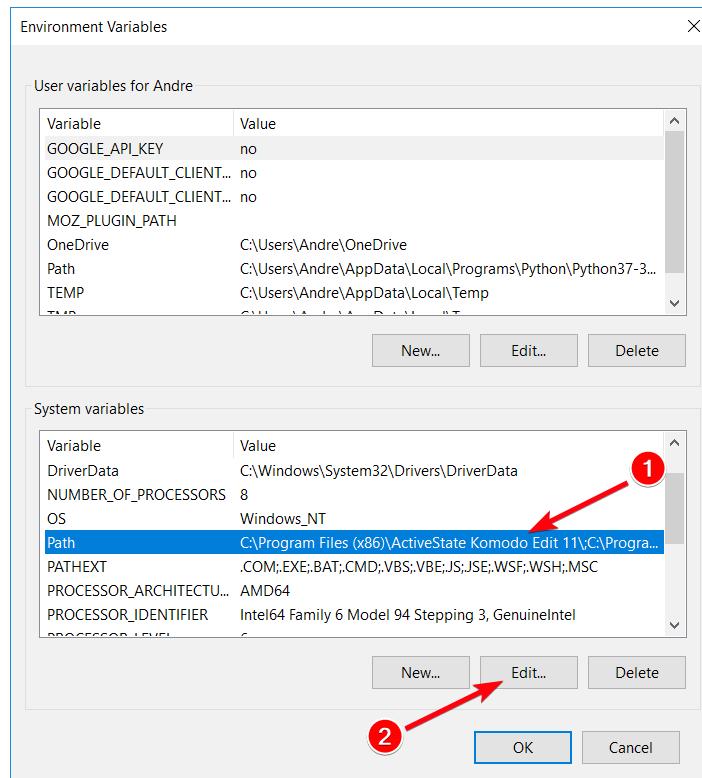


Gambar: Cara alternatif dari "Advanced System Settings" di Control Panel

Kemudian pilih tab "**Advanced**" di jendela "**System Properties**". Di jendela "**Advanced**" ini, klik tombol **Environment Variable** di bagian kanan bawah.

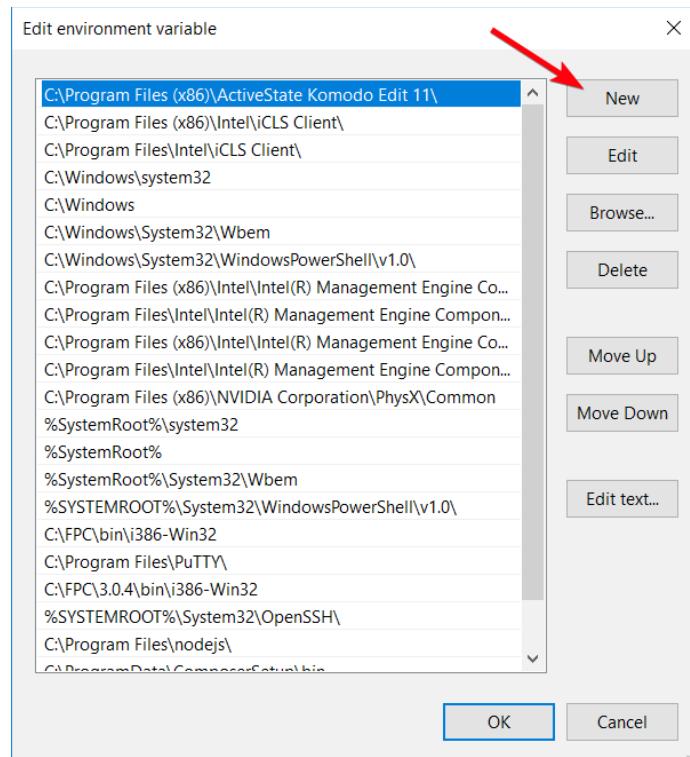


Gambar: Klik tombol Environment Variable



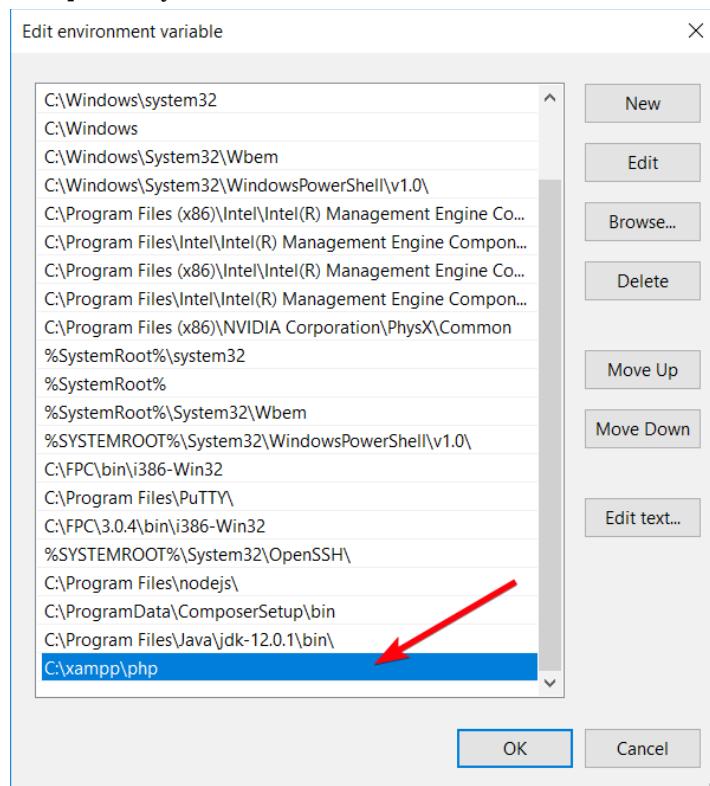
Di jendela **Environment Variable**, pilih baris **Path** di bagian bawah, lalu klik tombol **Edit**. Gambar: Pilih Path lalu klik tombol Edit

Sekarang akan tampil jendela **Edit Environment Variable** seperti berikut:



Gambar: Klik tombol New untuk menambah Path

Klik tombol **New** di kanan atas. Cursor akan beralih ke sisi kiri bawah, lalu ketik alamat path folder tempat file `php.exe` berada, yakni `C:\xampp\php`. Jika anda menginstall XAMPP di alamat lain, sesuaikan penulisan path-nya.



Gambar: Tambahkan alamat Path `C:\xampp\php`

Setelah path C:\xampp\php ada di dalam tabel, klik **OK** untuk mengakhiri, lalu klik kembali tombol **OK** beberapa kali untuk menutup semua jendela **Environment Variable dan System Properties**.

Sekarang pengaturan **Path** sudah selesai. Silahkan tutup cmd jika masih terbuka, lalu buka kembali dan ketik perintah php -v. Seharusnya perintah ini sudah bisa dikenali.

Pengetahuan untuk mengatur dan menambah PATH system variable ini sangat penting untuk dipahami, terutama jika sudah berhubungan dengan **cmd**.

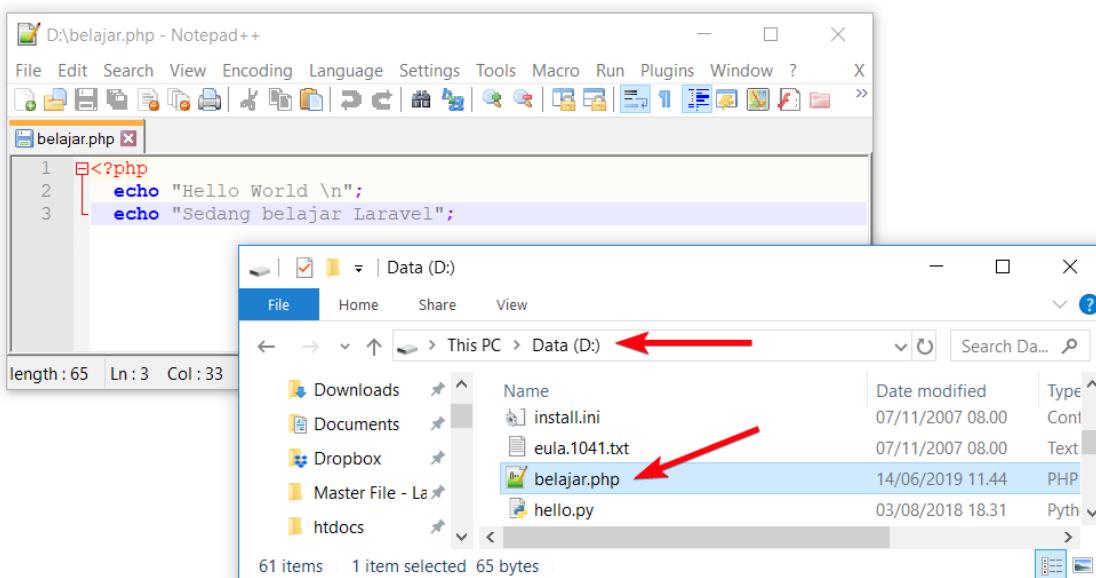
Aplikasi pemrograman lain seperti **Java**, **Python**, dan **NodeJS**, semuanya butuh konfigurasi PATH di system variable Windows. Cara yang dipakai sama saja, yang beda nanti hanya lokasi folder yang ingin ditambah.

Berikutnya, mari kita coba jalankan file PHP dari cmd. Silahkan buat sebuah file PHP dengan nama bebas dan simpan di folder mana saja, tidak harus di htdocs. Kali saya akan membuat file belajar.php di drive D, sehingga alamat file ada di D:/belajar.php.

Berikut isi dari file belajar.php:

belajar.php

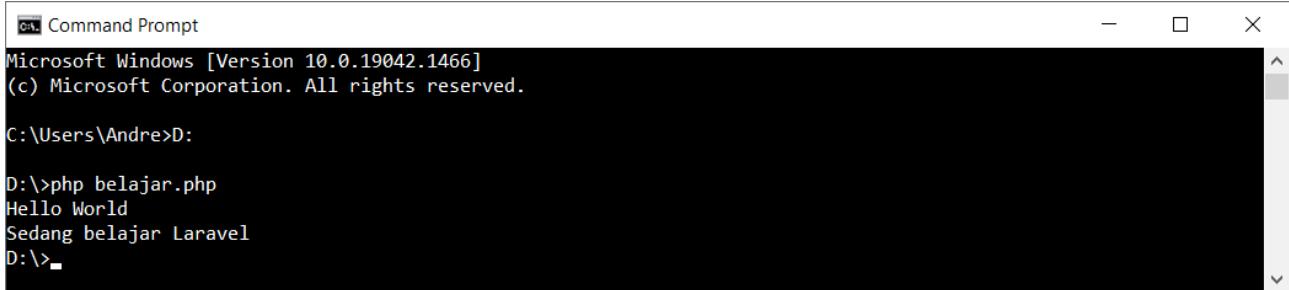
```
1 <?php
2 echo "Hello World \n";
3 echo "Sedang belajar Laravel";
```



Gambar: Simpan file belajar.php di drive D

Kode program kita hanya berisi 2 buah perintah echo. Silahkan buka cmd, masuk ke drive D dengan mengetik D: lalu tekan enter. Kemudian jalankan file belajar.php dengan perintah berikut:

```
php belajar.php
```



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>D:
D:\>php belajar.php
Hello World
Sedang belajar Laravel
D:\>
```

Gambar: File belajar.php tampil di cmd

Di cmd akan tampil 2 baris teks, yakni Hello World dan Sedang belajar Laravel. Inilah hasil dari perintah echo yang terdapat di file belajar.php.

Menjalankan file PHP dari cmd sebenarnya bukanlah hal wajib untuk bisa menggunakan Laravel, tapi pengetahuan ini sangat bermanfaat dan agar kita lebih familiar menggunakan cmd. Karena, instalasi Laravel juga dijalankan menggunakan cmd.

3.3. Instalasi Composer

Ketika ingin menggunakan file PHP yang dibuat programmer lain, cara yang biasa di pakai adalah download file .php tersebut, kemudian letakkan ke dalam folder **htdocs** XAMPP. Cara ini sangat praktis dan masih jadi pilihan untuk menginstall banyak aplikasi PHP.

Misalnya ketika ingin menginstall WordPress, caranya download terlebih dahulu file `wordpress.zip` dari web `wordpress.org`, kemudian unzip ke dalam folder `htdocs`. Begitu pula cara instalasi framework Code Igniter 3, yakni download file source code `codeigniter.zip` dari `codeigniter.com`, lalu unzip ke folder `htdocs`.

Namun tidak untuk proses instalasi Laravel, atau setidaknya dokumentasi resmi Laravel tidak menyarankan cara seperti itu.

Pengertian Composer, Package Manager dan Dependency

Untuk menginstall Laravel, kita disarankan memakai aplikasi **Composer**. Composer merupakan sebuah aplikasi *package manager* untuk menginstall berbagai file PHP, terutama library dan framework PHP.

Package manager sendiri adalah sebutan untuk aplikasi yang bertugas mengelola *dependency* atau ketergantungan antar aplikasi. Composer terinspirasi dari **npm** (singkatan dari *node package manager*), yakni aplikasi *package manager* yang lebih dahulu hadir untuk bahasa pemrograman JavaScript.

Pertanyaan, **kenapa harus pakai Composer? Dan apa itu dependency?**

Mayoritas website modern tidak lagi dibuat dari nol, tapi memanfaatkan berbagai library atau

kode program lain yang sudah ada. Laravel juga menggunakan prinsip yang sama. Istilahnya "*don't reinvent the wheel*", maksudnya jika orang lain sudah menyediakan alat yang tinggal pakai (dan diizinkan untuk dipakai), kenapa kita harus membuatnya lagi dari awal?

Jika anda pernah belajar **Bootstrap** (sebuah framework CSS), tentu paham betapa mudahnya membuat design web responsive menggunakan Bootstrap. Namun Bootstrap sendiri juga tidak dibuat dari nol. Secara internal, Bootstrap butuh library **jQuery** dan **Popper JS**. Di sini kita bisa sebut bahwa Bootstrap memiliki *dependency* atau ketergantungan kepada jQuery dan Popper JS. Tanpa kedua file ini, efek JavaScript Bootstrap tidak bisa jalan.

Untuk skala yang lebih besar, sebuah library atau framework bisa saja memiliki *dependency* kepada puluhan atau bahkan ratusan file lain. File lain itu kadang juga memiliki *dependency* lanjutan, dst. Dalam kasus Bootstrap, bisa saja file jQuery memiliki *dependency* ke file lain. Di sinilah **composer** berperan sebagai aplikasi yang mengurus *dependency* antar library.

Framework Laravel juga cukup unik karena banyak menggunakan komponen atau library yang sudah ada, termasuk library yang dikembangkan oleh framework PHP lain. Bisa disebut bahwa tim pengembang laravel juga menggunakan prinsip "*don't reinvent the wheel*". Jika fitur yang diinginkan sudah tersedia (yang juga *open source*), kenapa tidak menggunakananya saja?

Berikut tampilan beberapa library yang dipakai Laravel:

```

17   "require": {
18     "php": "^8.0.2",
19     "ext-mbstring": "*",
20     "ext-openssl": "*",
21     "doctrine/inflector": "^2.0",
22     "dragonmantank/cron-expression": "^3.1",
23     "egulias/email-validator": "^3.1",
24     "laravel/serializable-closure": "^1.0",
25     "league/commonmark": "^2.2",
26     "league/flysystem": "^3.0",
27     "monolog/monolog": "^2.0",
28     "nesbot/carbon": "^2.53.1",
29     "psr/container": "^1.1.1|^2.0.1",
30     "psr/log": "^1.0|^2.0|^3.0",
31     "psr/simple-cache": "^1.0|^2.0|^3.0",
32     "ramsey/uuid": "^4.2.2",
33     "symfony/console": "^6.0",
34     "symfony/error-handler": "^6.0",
35     "symfony/finder": "^6.0",
36     "symfony/http-foundation": "^6.0",
37     "symfony/http-kernel": "^6.0",
38     "symfony/mailer": "^6.0",
39     "symfony/mime": "^6.0",
40     "symfony/process": "^6.0",
41     "symfony/routing": "^6.0",
42     "symfony/var-dumper": "^6.0",
43     "tijsverkoyen/css-to-inline-styles": "^2.2.2",
44     "vlucas/phpdotenv": "^5.4.1",
45     "voku/portable-ascii": "^1.6.1"
46   },

```

Sebagian library yang di pakai Laravel

Teks di atas merupakan potongan dari file `composer.json` yang ada di installer Laravel (`laravel/framework`). Di setiap library atau framework yang nantinya kita install dari composer, file `composer.json` akan selalu ada. Di dalam file inilah bisa dilihat daftar semua dependency.

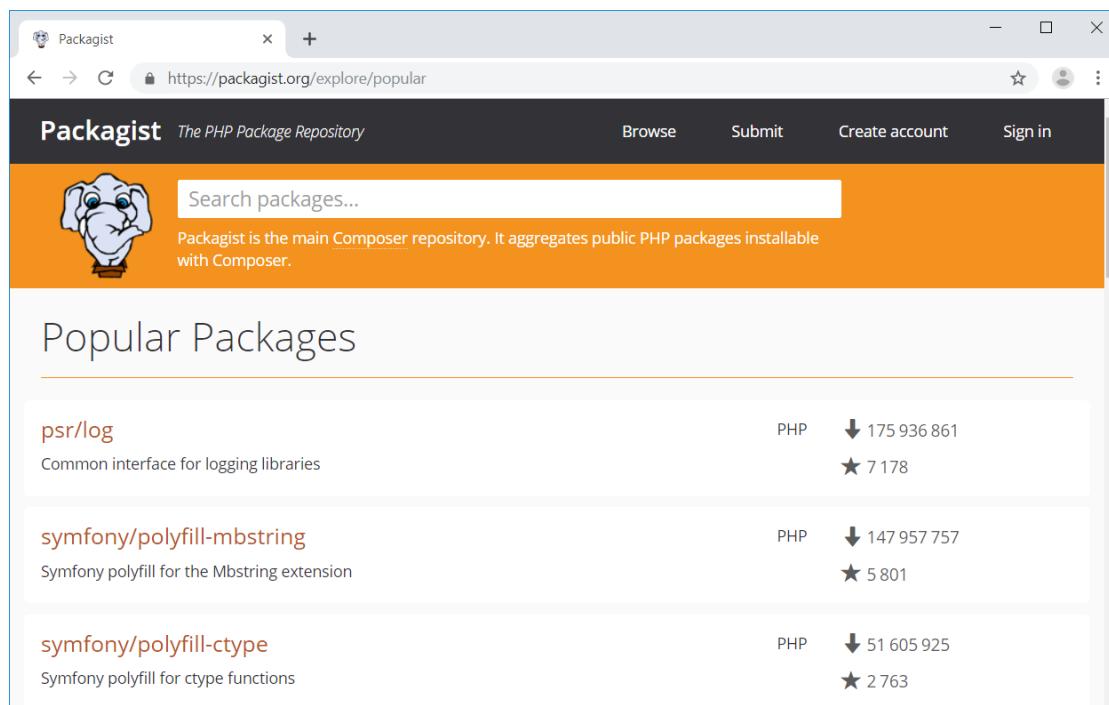
Terlihat Laravel butuh setidaknya 25 library (baris 21 – 45). Sehingga bisa juga disebut bahwa sebagian besar file yang ada di dalam Laravel bukan dibuat oleh tim pengembang Laravel itu sendiri, tapi gabungan dari library yang sudah ada. Perhatikan juga baris 33 – 42, di bagian awal terdapat kata "`symfony/`", yang artinya Laravel menggunakan beberapa komponen kepunyaan framework PHP lain, yakni **Symfony**.

Tapi ini bukan berarti bahwa Laravel hanya sebuah framework copy-paste. Tim pengembang Laravel sudah menyaring apa saja library yang layak dipakai. Jika library tersebut memang bagus (dan sudah teruji), maka tidak perlu dibuat ulang.

Ini juga salah satu alasan kepopuleran Laravel karena berusaha mengumpulkan *best practice* yang sudah tersedia di industri. Selain itu tentu saja tetap butuh kode untuk menggabungkan semua library ini menjadi paket framework PHP yang lengkap.

Package manager seperti **composer** berperan dalam mengelola semua library ini. Composer akan mengumpulkan semua file di satu tempat, dan mengatur jika ada salah satu library yang di update.

Sebagai informasi tambahan, aplikasi composer hanya bertugas mengumpulkan library. Library itu sendiri berada di tempat lain, yakni di packagist.org.



Gambar: Tampilan web packagist.org

Packagist merupakan repository atau "gudang" dari berbagai library dan framework PHP. Jika anda ingin mencari berbagai library PHP, di sinilah tempatnya.

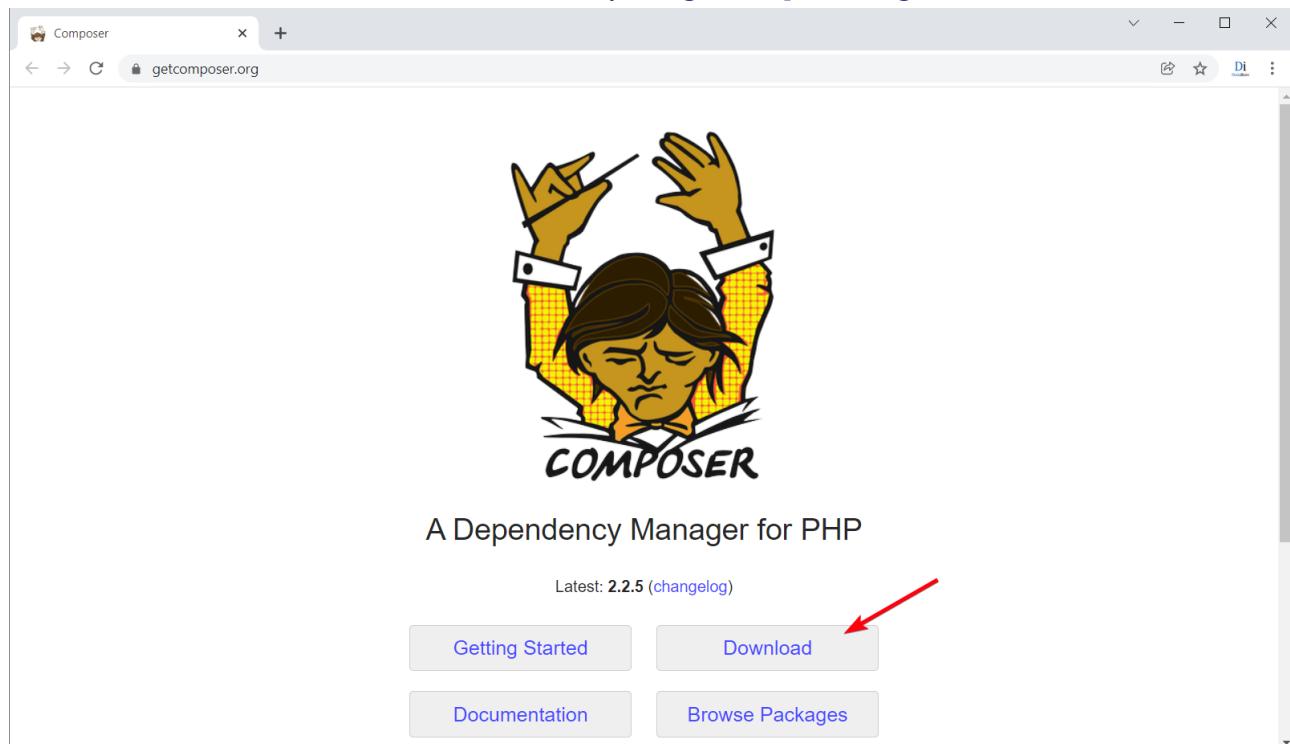
Sebenarnya tempat terbaik untuk mencari (dan mempublikasikan) library untuk bahasa pemrograman apapun adalah di [GitHub](#). Hanya saja khusus untuk composer, secara default menggunakan packagist sebagai repository.

Selain itu jika anda ingin mempublikasikan kode program yang sudah di tulis ke packagist, jalurnya memang ke GitHub terlebih dahulu, baru kemudian dihubungkan ke Packagist agar bisa diinstall dari Composer.

Semoga anda bisa memahami pengertian dari *composer*, *package manager* dan *dependency*. Ketiga istilah ini memang agak asing, namun lambat laun akan familiar.

Menginstall Composer

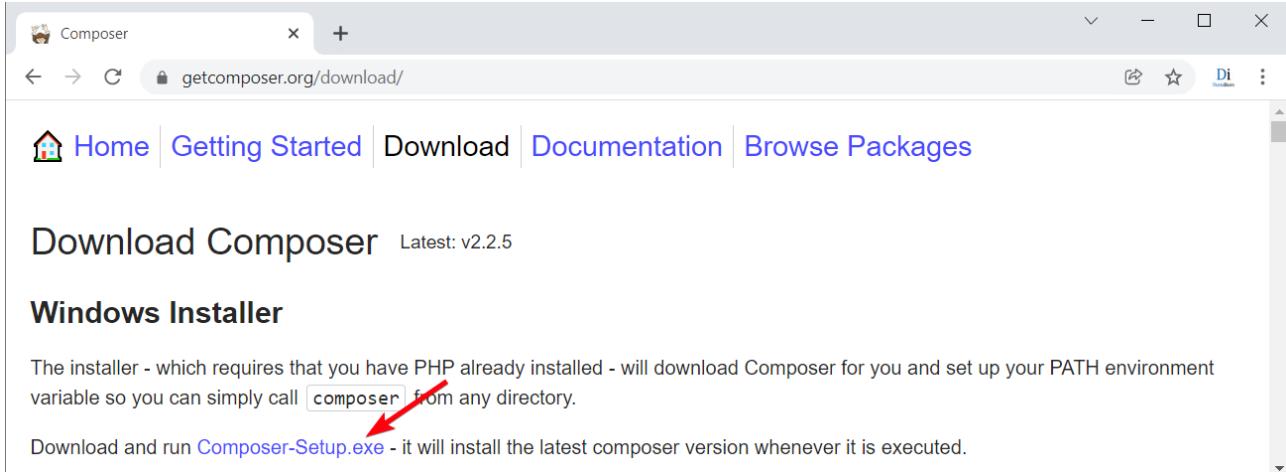
Pada dasarnya cara menginstall Composer sama seperti proses instalasi program lain. Pertama, ambil file installer dari web resminya di [getcomposer.org](#).



Gambar: Tampilan halaman awal getcomposer.org

Dari halaman home, klik tombol **Download**. Jika menggunakan Windows, lanjutkan dengan klik link [Composer-Setup.exe](#) dan save file Composer-Setup.exe di folder mana saja untuk proses instalasi.

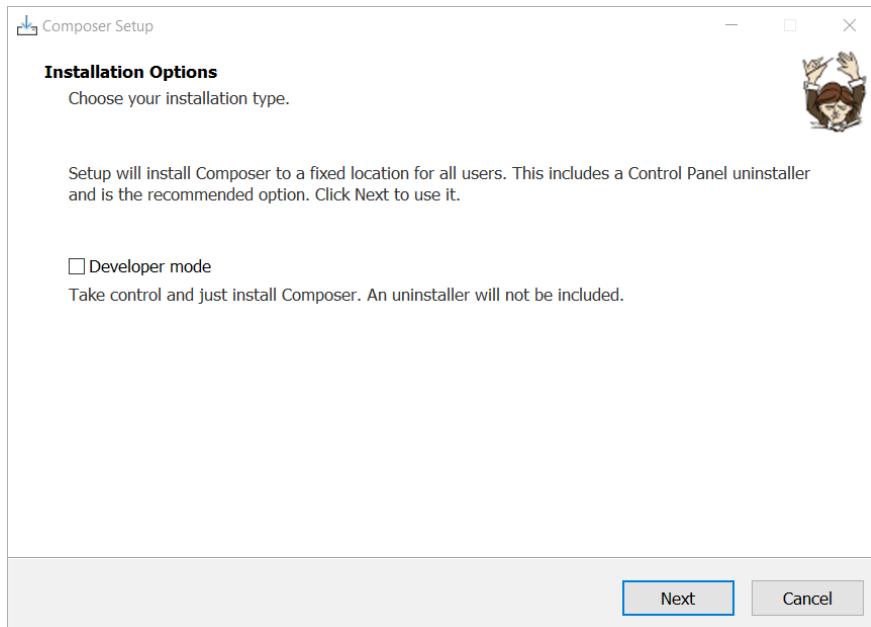
Instalasi Laravel



Gambar: Link untuk download Composer-Setup.exe

Jika anda menggunakan **Linux** atau **Mac OS**, bisa ikut panduannya ke: [How To Install and Use PHP Composer on Ubuntu 18.04](#) atau [Installing Composer on Mac OSX](#)

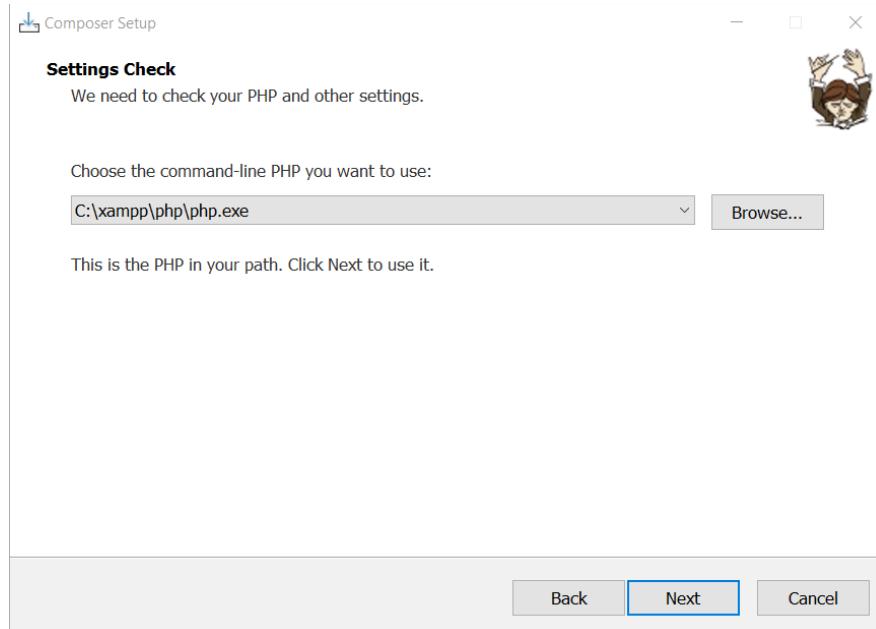
Untuk mulai proses instalasi, double klik file **Composer-Setup.exe**. Di jendela pertama, biarkan checkbox **Developer mode** kosong dan langsung saja klik tombol **Next**.



Gambar: Jendela pertama proses intalasi Composer

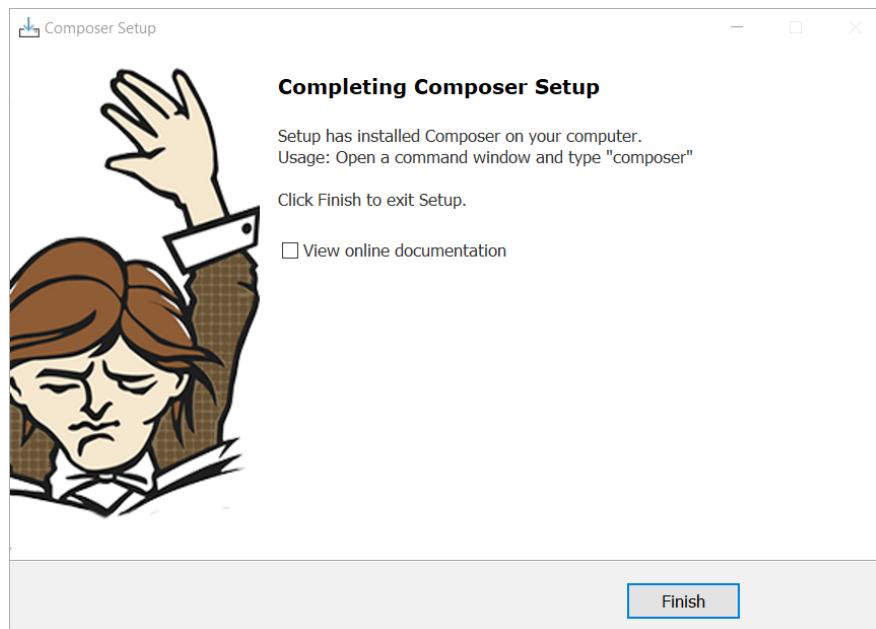
Di jendela kedua tampil pilihan checkbox untuk mencari file **php.exe**. Composer akan mencoba "menebak" lokasi ini. Pastikan alamat tersebut benar, yakni path dari file **php.exe**. Karena saya menginstall XAMPP di Drive C, maka alamat **php.exe** ini ada di **C:\xampp\php\php.exe**. Lanjut dengan klik tombol **Next**.

Instalasi Laravel



Gambar: Composer akan mencari alamat dari file php.exe

Untuk jendela inputan proxy tidak perlu diisi dan langsung saja klik tombol **Next**. Kemudian klik tombol **Install**. Tunggu beberapa saat, klik tombol **Next** dan **Finish**. Sip, composer sudah sukses terinstall.



Gambar: Composer selesai di install

Composer adalah aplikasi yang diakses dari cmd. Maka untuk menguji apakah composer sudah berhasil di install atau tidak, buka cmd, ketik **composer** dan tekan **Enter**.

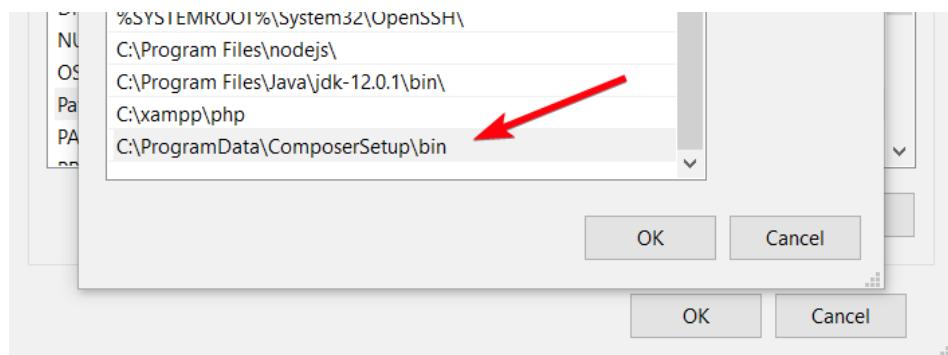
Jika tampil teks seperti berikut, maka artinya composer sudah berhasil di install dan bisa diakses secara global.

Instalasi Laravel

Gambar: Composer sudah bisa dijalankan

Namun apabila tampil pesan '*composer* is not recognized as an internal or external command, operable program or batch file', maka ada 2 kemungkinan. Pertama, installer Composer gagal, silahkan coba install ulang.

Kemungkinan kedua, composer sudah berhasil diinstall tapi tidak bisa diakses secara global. Solusinya, tambah folder C:\ProgramData\ComposerSetup\bin ke dalam **PATH system variable**. Mengenai caranya, sudah kita bahas ketika mengatur php.exe secara global.



Gambar: Tambah folder C:\ProgramData\ComposerSetup\bin ke dalam PATH system variable

Sekarang kita bisa masuk ke bagian yang ditunggu-tunggu, yakni proses instalasi Laravel.

3.4. Instalasi Laravel

Dalam [dokumentasi resmi Laravel](#) dijelaskan bahwa terdapat 2 cara instalasi Laravel. Pertama dengan perintah **composer create-project**, dan yang kedua dengan **laravel installer**. Kita akan bahas kedua cara ini.

Pastikan komputer anda terhubung ke internet karena proses instalasi Laravel (dengan

composer) tidak bisa dilakukan secara offline. File Laravel yang akan di download juga berukuran cukup besar, yakni sekitar **33MB** untuk 1 kali instalasi Laravel.

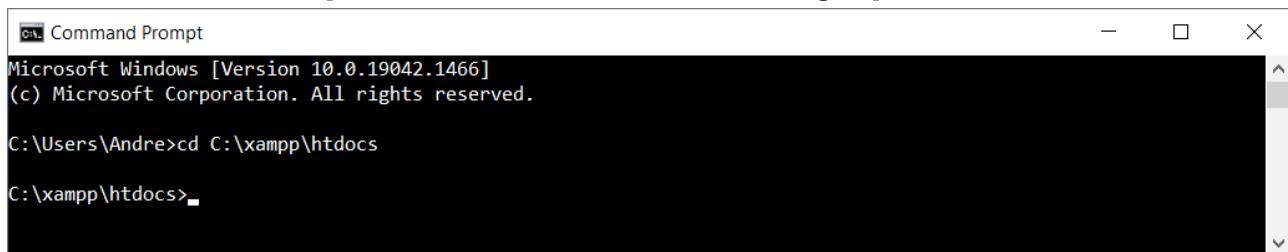
Dalam tutorial ini saya akan menjalankan beberapa kali proses instalasi. Jika anda memakai paket internet yang cukup mahal (berbatas quota), cukup install Laravel 1 kali saja, lalu copy seluruh file yang sudah di download ke tempat lain sebagai "file master". Jika nanti ingin membuat project Laravel baru, cukup copy lagi file ini (tidak perlu lagi dari Composer).

Saya juga telah menyertakan master file Laravel di Google Drive dengan nama file **master_laravel_9.zip**. Anda bisa download file ini lalu unzip ke **htdocs**.

Install Laravel dengan Composer Create-Project

Proses instalasi Laravel dengan perintah **composer create-project** ini kadang disebut juga sebagai cara singkat, karena kita hanya perlu menulis 1 perintah saja ke dalam cmd. Cara ini memang praktis tapi perintahnya agak susah di hafal karena cukup panjang.

Silahkan buka **cmd** lalu pindah ke folder **htdocs** XAMPP dengan perintah `cd C:\xampp\htdocs`.



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>cd C:\xampp\htdocs

C:\xampp\htdocs>
```

Gambar: Buka cmd lalu pindah ke C:\xampp\htdocs

Kemudian ketik atau copy perintah berikut:

```
composer create-project laravel/laravel coba1
```

Perintah ini terdiri dari beberapa bagian:

- ◆ **composer**: jalankan composer
- ◆ **create-project**: buat sebuah project
- ◆ **laravel/laravel**: nama library atau framework yang akan di install
- ◆ **coba1**: simpan semua file ke dalam folder **coba1**

Yang agak unik adalah, nama framework Laravel di panggil sebagai **laravel/laravel**. Dalam *repository packagist*, sebuah library harus memiliki nama dengan format: **vendor_name/project_name**.

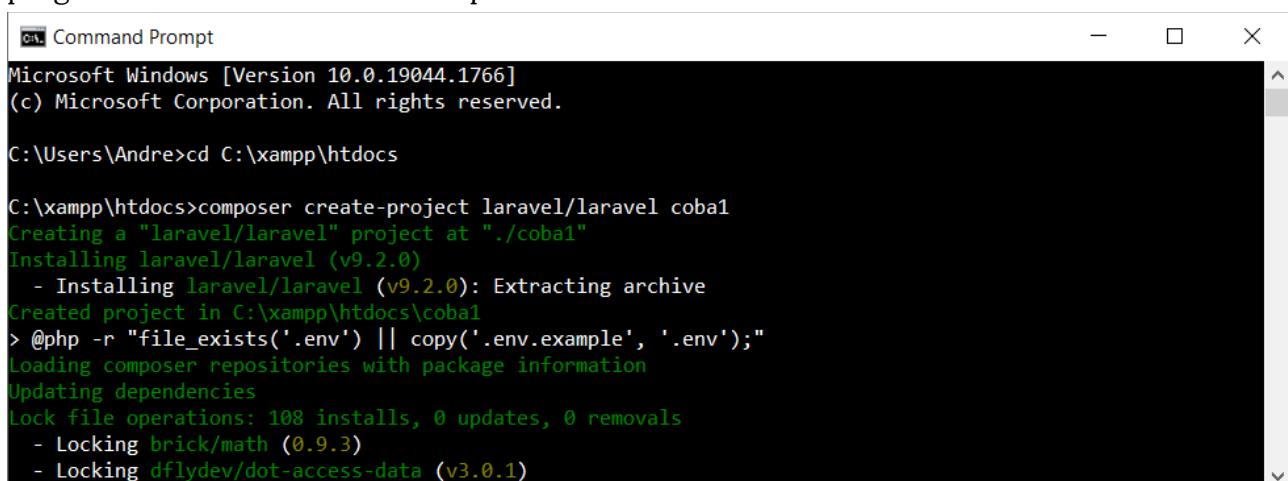
Vendor name adalah sebutan untuk programmer atau tim yang membuat library. Sedangkan **project name** adalah nama dari library yang dibuat.

Sebagai contoh, jika saya ingin membuat sebuah library dengan nama `validator`, maka di packagist nama paketnya menjadi `andre/validator`. Atau jika alex membuat library yang ia beri nama `larachoy` maka di packagist namanya menjadi `alex/larachoy`. Aturan penamaan ini dipakai untuk mencegah bentrok 2 nama library dengan nama yang sama.

Dalam perintah di atas `laravel/laravel` artinya kita ingin menginstall library dengan nama `laravel` yang dibuat oleh tim `laravel`.

Selain framework Laravel, tim laravel juga membuat beberapa library lain yang masih berhubungan dengan framework Laravel, seperti `laravel/helpers`, `laravel/homestead`, `laravel/tinker`, dan masih banyak lagi.

Kembali ke perintah `composer create-project`, ketika kita menekan tombol **Enter**, maka composer akan bekerja dan mendownload semua file Laravel. Proses ini bisa butuh waktu beberapa menit karena ukuran file Laravel memang cukup besar (sekitar 35MB). Anda bisa pergi sebentar untuk membuat kopi.

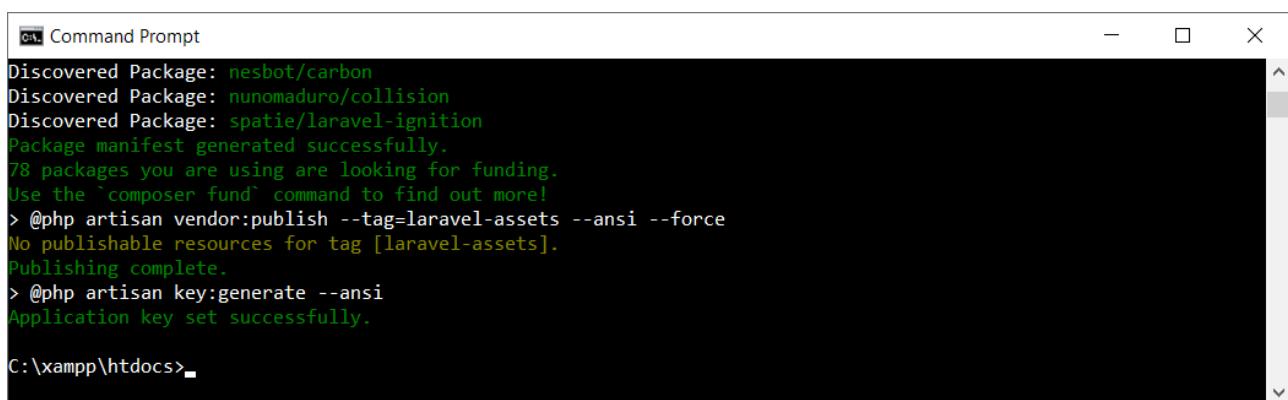


```
Command Prompt
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>cd C:\xampp\htdocs

C:\xampp\htdocs>composer create-project laravel/laravel coba1
Creating a "laravel/laravel" project at "./cba1"
Installing laravel/laravel (v9.2.0)
  - Installing laravel/laravel (v9.2.0): Extracting archive
Created project in C:\xampp\htdocs\cba1
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 108 installs, 0 updates, 0 removals
  - Locking brick/math (0.9.3)
  - Locking dflydev/dot-access-data (v3.0.1)
```

Gambar: Proses instalasi Laravel dimulai



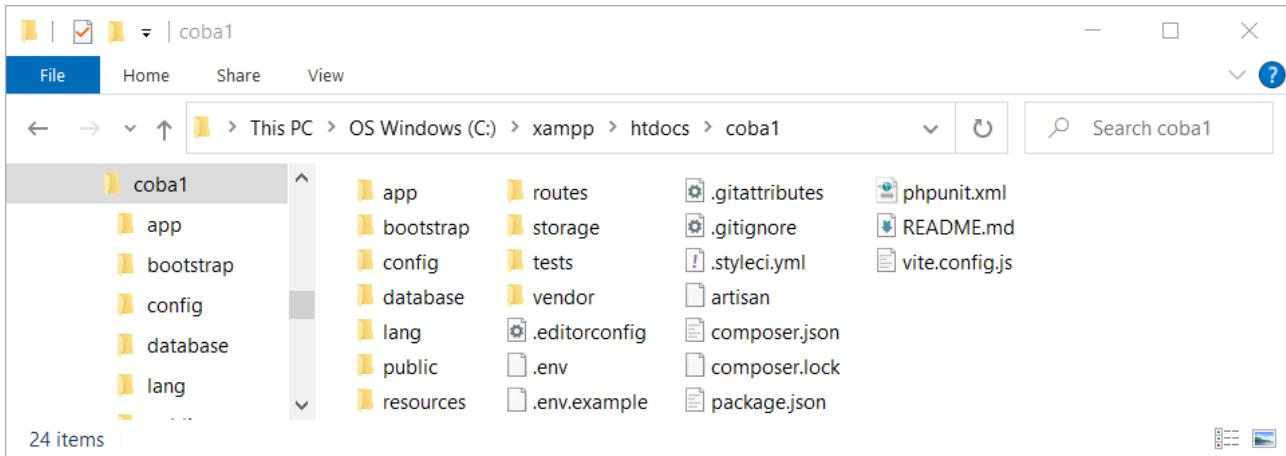
```
Command Prompt
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: spatie/laravel-ignition
Package manifest generated successfully.
78 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
No publishable resources for tag [laravel-assets].
Publishing complete.
> @php artisan key:generate --ansi
Application key set successfully.

C:\xampp\htdocs>
```

Gambar: Proses instalasi Laravel selesai

Apabila sudah tampil teks 'Application key set successfully' dan cursor kembali ke `C:\xampp\htdocs`, maka artinya **proses instalasi Laravel sudah selesai**. Silahkan buka folder `htdocs` dari

Windows Explorer, akan terlihat folder **coba1** yang berisi berbagai file Laravel.



Gambar: isi folder coba1

Laravel memiliki cara tersendiri untuk bisa diakses dari web browser (akan kita bahas dalam bab berikutnya). Jadi untuk sementara anda bisa biarkan file-file ini "apa adanya".

Apabila kita ingin membuat project baru, maka tinggal menjalankan perintah yang sama. Yang perlu diubah hanya nama folder saja, Misalkan saya ingin membuat web sistem informasi SMA, maka bisa mengetik perintah berikut:

```
composer create-project laravel/laravel sistem_informasi_sma
```

Composer akan kembali menginstall Laravel ke folder *sistem_informasi_sma*.

Composer Cache

Saat menjalankan perintah `composer create-project` untuk pertama kali, composer sebenarnya juga menyimpan file-file tersebut ke dalam *cache* internal, yakni sebuah tempat penyimpanan sementara di harddisk kita.

Ketika perintah yang sama dijalankan untuk kali kedua, composer tetap akan memeriksa ke packagist terlebih dahulu, yakni apakah ada versi update dari komponen yang sedang di install, jika tidak ada, composer cukup mengambil file dari cache, tidak perlu mendownload ulang semuanya dari internet.

Jadi ketika kita menginstall Laravel untuk kedua, ketiga dan seterusnya, akan terasa lebih cepat, tidak selama waktu menginstall pertama kali karena file tersebut diambil dari cache. Ini juga menghemat kuota internet yang terpakai.

Install Laravel dengan Laravel Installer

Kita akan bahas cara kedua untuk proses instalasi Laravel, yakni via **Laravel Installer**. Cara ini kadang disebut juga dengan cara yang (sedikit) panjang karena kita perlu menginstall aplikasi Laravel Installer terlebih dahulu.

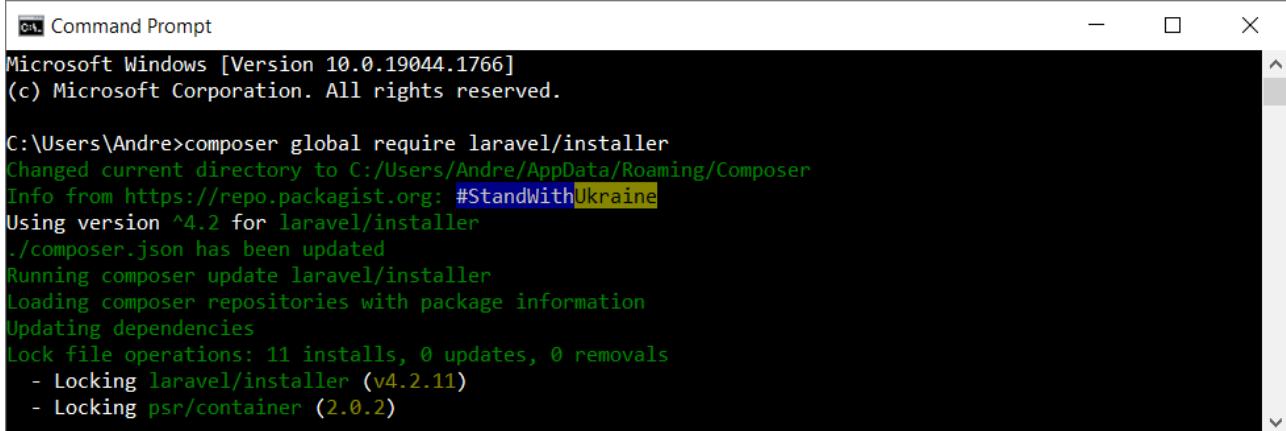
Instalasi Laravel

Namun ini sangat cocok jika anda sering membuat project laravel karena perintahnya lebih singkat daripada `composer create-project`. Selain itu Laravel Installer hanya perlu di install sekali di awal saja.

Proses instalasi Laravel Installer juga dilakukan dari composer, yakni dengan perintah berikut:

```
composer global require laravel/installer
```

Jalankan dengan menekan tombol Enter.

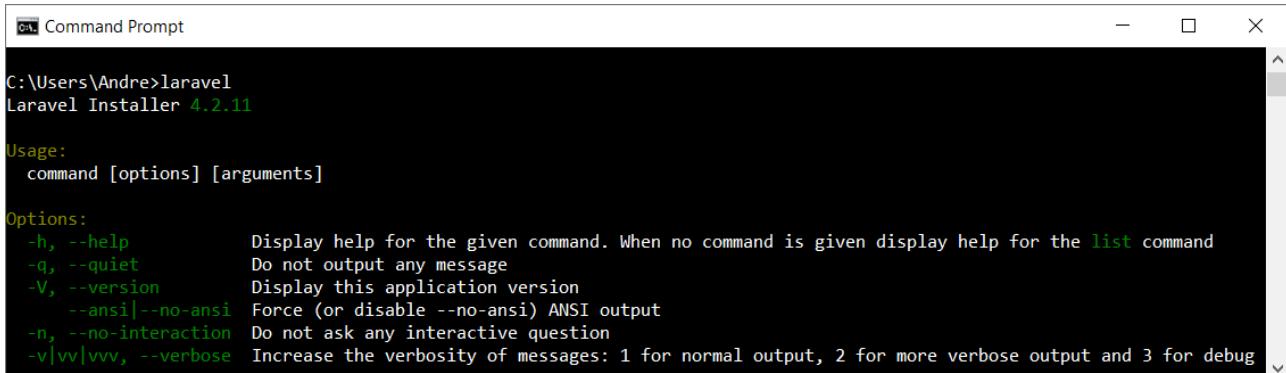


```
Command Prompt
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>composer global require laravel/installer
Changed current directory to C:/Users/Andre/AppData/Roaming/Composer
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^4.2 for laravel/installer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
Lock file operations: 11 installs, 0 updates, 0 removals
- Locking laravel/installer (v4.2.11)
- Locking psr/container (2.0.2)
```

Gambar: Proses instalasi Laravel Installer dimulai

Setelah selesai, kita akan mendapat "program baru" untuk cmd, yakni **laravel.exe**. Untuk uji coba, silahkan tutup cmd, lalu buka kembali dan ketik `laravel` kemudian tekan Enter:



```
Command Prompt
C:\Users\Andre>laravel
Laravel Installer 4.2.11

Usage:
  command [options] [arguments]

Options:
  -h, --help           Display help for the given command. When no command is given display help for the list command
  -q, --quiet          Do not output any message
  -V, --version         Display this application version
  --ansi|--no-ansi     Force (or disable --no-ansi) ANSI output
  -n, --no-interaction Do not ask any interactive question
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Gambar: Perintah laravel bisa diakses dari cmd

Jika terlihat tampilan di atas, artinya **Laravel Installer** sudah sukses terinstall. Sekarang untuk menginstall Laravel perintahnya jauh lebih singkat:

```
laravel new <nama_folder>
```

Sebagai contoh, saya ingin menginstall Laravel di folder `C:\xampp\htdocs\coba2\`, maka perintahnya adalah:

```
cd C:\xampp\htdocs\
laravel new coba2
```

Instalasi Laravel

Perintah `cd C:\xampp\htdocs\` dipakai untuk pindah ke folder `C:\xampp\htdocs\`. Setelah itu barulah proses instalasi Laravel dilakukan dengan perintah `laravel new coba2`.



```
Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

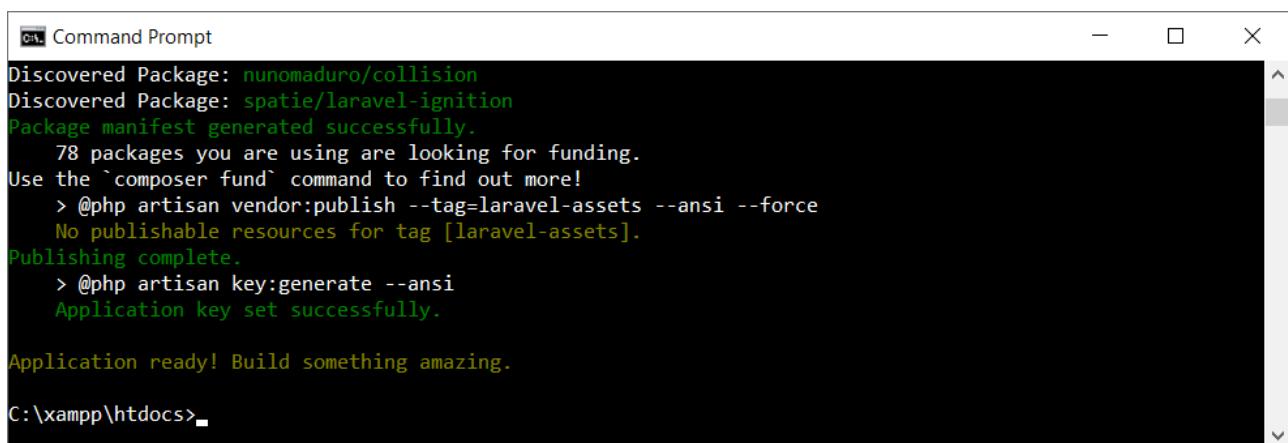
C:\Users\Andre>cd C:\xampp\htdocs\

C:\xampp\htdocs>laravel new coba2

[Progress Bar]

Creating a "laravel/laravel" project at "./coba2"
Installing laravel/laravel (v9.2.0)
- Installing laravel/laravel (v9.2.0): Extracting archive
  Created project in C:\xampp\htdocs\coba2
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
  Loading composer repositories with package information
```

Gambar: Proses instalasi Laravel dimulai



```
Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

Discovered Package: nunomaduro/collision
Discovered Package: spatie/laravel-ignition
Package manifest generated successfully.
  78 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
  > @php artisan vendor:publish --tag=laravel-assets --ansi --force
    No publishable resources for tag [laravel-assets].
Publishing complete.
  > @php artisan key:generate --ansi
    Application key set successfully.

Application ready! Build something amazing.

C:\xampp\htdocs>
```

Gambar: Proses instalasi Laravel selesai

Setelah proses ini selesai, folder `coba2` akan berisi file Laravel yang sama persis dengan folder `coba1`. Anda bebas ingin menggunakan perintah yang mana saja.

Mengatur versi Laravel

Sebagaimana layaknya sebuah software, Laravel secara berkala terus di update (saat ini setiap 1 tahun sekali). Dalam setiap update terdapat perbaikan bug, penambahan fitur baru, serta penghapusan fitur lama.

Update versi Laravel tidak selalu diikuti perubahan besar. Bisa jadi hanya peningkatan kinerja yang tidak terlalu berpengaruh ke cara penggunaan, namun tetap ada kemungkinan 1 atau 2 perintah yang berubah.

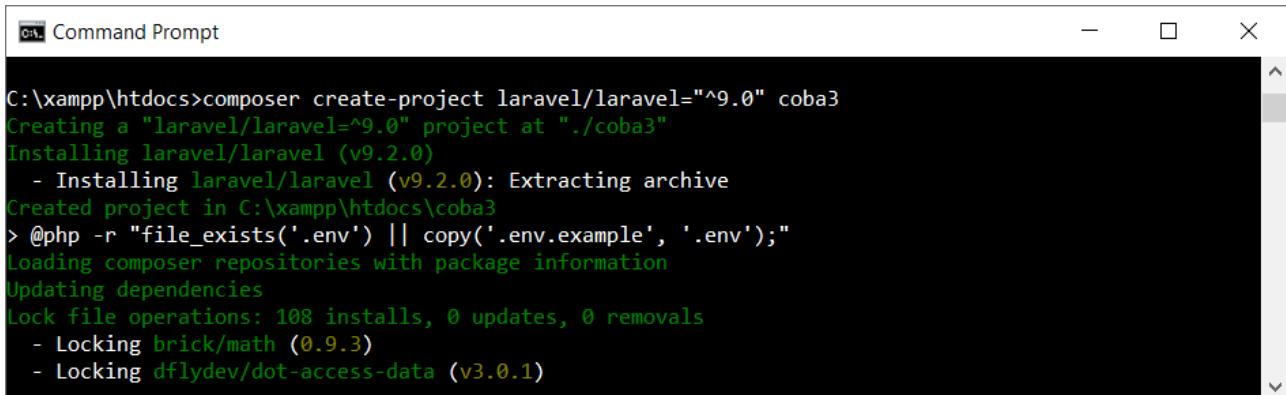
Pada saat buku ini di revisi, versi terakhir Laravel adalah versi 9. Agar semua materi buku bisa

diikuti dengan baik, maka **saya mengharuskan anda untuk menggunakan Laravel 9**. Ini untuk menghindari perbedaan fitur antar versi Laravel, karena bisa jadi ketika membaca buku ini Laravel sudah masuk ke versi 10, versi 11 atau yang lebih baru lagi.

Proses instalasi yang kita lakukan sebelumnya secara otomatis akan menginstall Laravel versi terbaru. Apabila saat ini sudah hadir Laravel 10, maka perintah tersebut akan menginstall Laravel 10 ke folder `coba1` dan `coba2`.

Jadi, bagaimana cara untuk menginstall Laravel untuk versi tertentu? Kita bisa modifikasi sedikit perintah `composer create-project`:

```
composer create-project laravel/laravel="^9.0" coba3
```



```
C:\xampp\htdocs>composer create-project laravel/laravel="^9.0" coba3
Creating a "laravel/laravel:^9.0" project at "./coba3"
Installing laravel/laravel (v9.2.0)
- Installing laravel/laravel (v9.2.0): Extracting archive
Created project in C:\xampp\htdocs\coba3
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 108 installs, 0 updates, 0 removals
- Locking brick/math (0.9.3)
- Locking dflydev/dot-access-data (v3.0.1)
```

Gambar: Proses instalasi Laravel dengan versi tertentu

Sebagaimana yang pernah kita bahas di bab 1, Laravel saat ini memakai sistem penomoran **semantic versioning**, dimana digit kedua berfungsi sebagai penanda update minor seperti 9.1, 9.2, dst. Update mayor baru akan terjadi untuk Laravel 10.

Tambahan `= " ^9.0 "` artinya saya ingin menginstall Laravel 9 dengan versi update minor terakhir. Misalnya saat ini sudah tersedia Laravel 9.4, maka perintah di atas akan menginstall versi tersebut. Seharusnya, update minor tidak akan berpengaruh banyak ke kode program yang ada dalam buku ini.

Perintah instalasi di atas tidak berlaku untuk Laravel 5, karena pada versi tersebut update mayor terjadi di digit ke dua. Misalkan anda menemukan tutorial yang hanya bisa berjalan di Laravel 5.6, maka perintah yang dipakai adalah:

```
composer create-project laravel/laravel="5.6.*" coba4
```

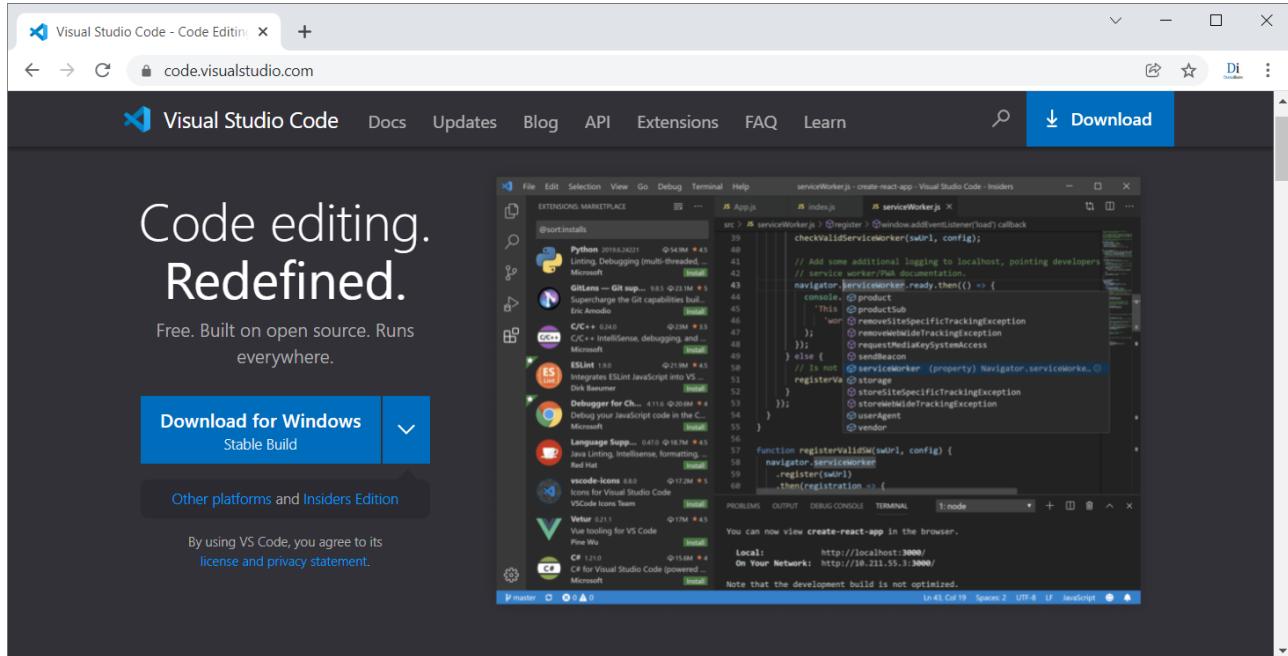
Sayangnya, proses install Laravel versi lama ini tidak tersedia jika kita menggunakan **Laravel Installer**. Yang bisa dipakai hanya dari perintah `composer create-project` saja.

Bagi yang terbatas quota internet, silahkan copy folder `coba3` yang berisi **Laravel 9** ke folder lain, misalnya folder `laravel9`. Dengan demikian ketika ingin memulai project baru, tinggal copy folder tersebut, tidak perlu menjalankan ulang perintah `composer`.

3.5. Instalasi Visual Studio Code

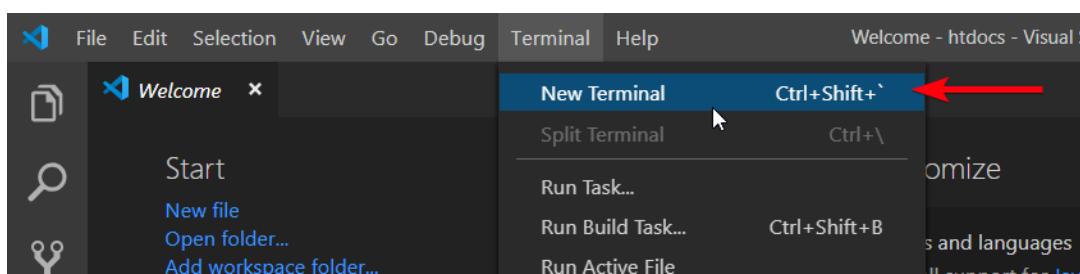
Saya yakin anda sudah memiliki pilihan text editor sendiri, apakah itu **Sublime Text**, **Atom**, **Visual Studio Code**, atau **Notepad++**. Semua teks editor ini tetap bisa dipakai karena Laravel pada dasarnya hanya terdiri dari file PHP biasa.

Dalam buku ini saya akan memakai Visual Studio Code atau sering disingkat sebagai VS Code (code.visualstudio.com). Ini bukan pilihan wajib karena kembali, teks editor apapun tidak masalah. Untuk beberapa komputer, VS Code mungkin terasa berat, jadi anda bisa pilih Notepad++ yang lebih ringan.



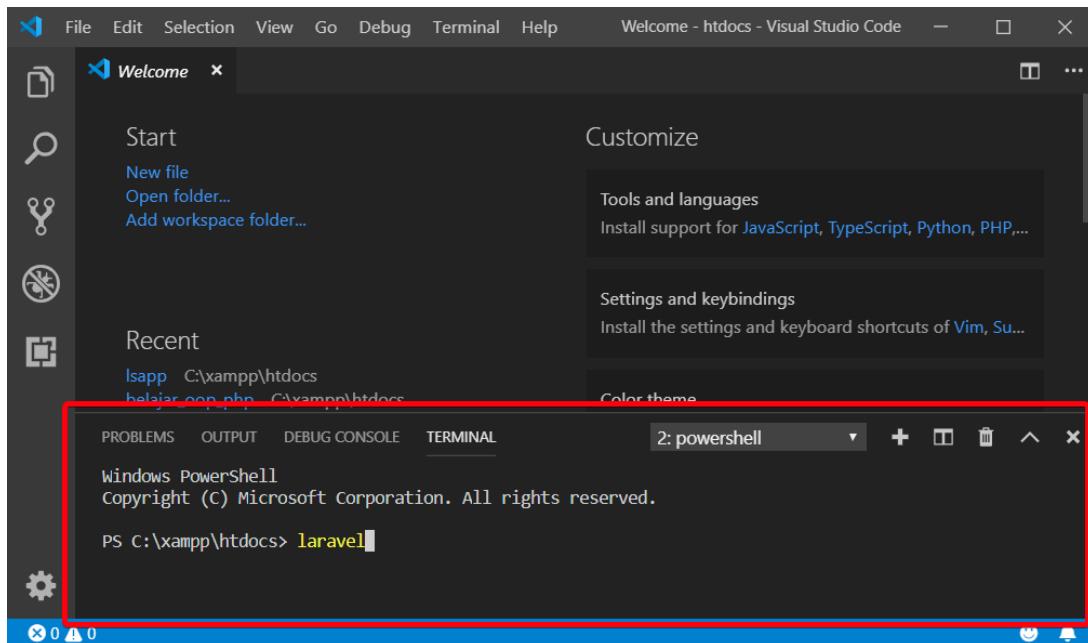
Gambar: Tampilan website resmi VS Code di code.visualstudio.com

Salah satu fitur VS Code yang berguna adalah bisa mengakses cmd atau terminal dari dalam text editor. Caranya, pilih menu **Terminal -> New Terminal**, atau bisa juga melalui shortcut **Ctrl + Shift + `**. Tombol untuk tanda ` ada di sudut kiri atas keyboard, yakni di atas tombol Tab.



Gambar: Menu untuk menampilkan terminal di VS Code

Setelah menu ini di klik, pada bagian bawah akan tampil jendela kecil dimana kita bisa mengetik perintah cmd:



Gambar: Tampilan jendela terminal di VS Code

Fitur ini memang tidak wajib karena kita tetap bisa membuka cmd seperti biasa. Tapi jika anda merasa repot harus bolak-balik dari cmd ke teks editor, fitur ini sangat bermanfaat.

Selain itu VS Code juga memiliki plugin atau extension khusus Laravel, yang akan memudahkan kita dalam penulisan kode program. Extension ini akan kita pelajari pada materi yang sesuai nantinya.

Dalam bab ini kita telah menyiapkan 'perangkat perang' untuk belajar Laravel, yakni: **XAMPP**, **Composer**, dan **Teks Editor** (VS Code).

Aplikasi lain yang juga dibutuhkan tentu saja **Web Browser**. Untuk pilihan web browser tidak ada syarat khusus, anda bisa pakai Google Chrome, Mozilla Firefox, Microsoft Edge atau Opera, selama aplikasi tersebut update agar tidak bermasalah dengan tampilan (bisa memproses CSS dengan sempurna).

Selain itu kita juga sudah mempelajari cara instalasi Laravel menggunakan **composer** dengan 2 cara, yakni menggunakan perintah **composer install project** serta **Laravel Installer**.

Dan agar bisa mengikuti semua pembahasan di buku ini dengan nyaman, **saya mengharuskan untuk memakai Laravel 9**, meskipun ternyata sudah ada versi Laravel yang lebih baru. Ini untuk menghindari perubahan fitur yang kadang bisa membuat bingung. Setelah selesai membaca buku ini, silahkan update Laravel dan pelajari perubahan yang terdapat di versi terbaru. Next, kita akan bahas cara mengakses dan menjalankan Laravel.

4. Mengakses Laravel

Dalam bab ini kita akan membahas cara mengakses Laravel. Laravel memiliki cara tersendiri agar bisa dijalankan.

4.1. Folder Instalasi Laravel

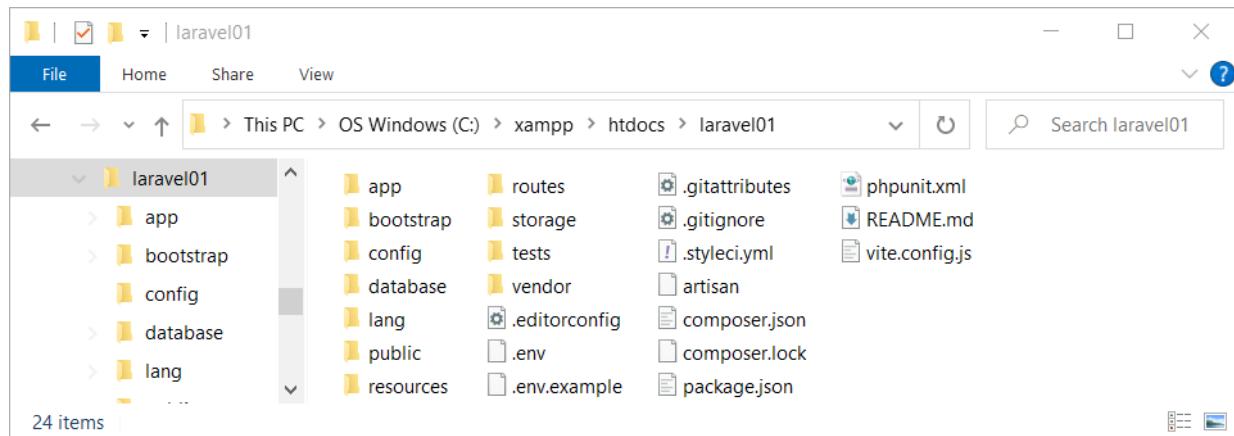
Dalam bab sebelumnya kita telah berhasil menginstall Laravel. Jika anda mengikuti semua langkah yang ada, maka di `htdocs` terdapat 3 folder Laravel 9: `coba1`, `coba2`, dan `coba3`. Pilih salah satu folder lalu rename menjadi **laravel01**. Perubahan nama folder ini tidak wajib, hanya agar lebih rapi saja. Atau juga bisa menginstall Laravel 9 baru dengan perintah berikut:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Sepanjang buku ini, saya akan terus memakai folder dengan nama "**laravel01**". Anda bisa saja menggunakan nama lain karena itu tidak pengaruh ke kode program. Yang penting, di dalam folder tersebut terinstall **Laravel**.

Selain itu hampir di semua awal bab saya akan menginstall ulang Laravel ke dalam folder ini. Tujuannya agar kita bisa fokus ke materi yang dibahas, tidak terganggu proses "utak-atik" dari bab sebelumnya.

Baik, mari lihat apa saja isi folder Laravel ini:



Gambar: Isi folder Laravel

Untuk Laravel 9 yang saya gunakan, total terdapat sekitar **7.259 file** di dalam **1.218 folder**. Yup, terbilang luar biasa banyak, dan itulah alasan kenapa file installer Laravel mencapai 34MB.

Namun dari gambar terlihat hanya ada 11 folder serta beberapa file. Mayoritas file Laravel memang tersembunyi karena perlu kita utak-atik. Sepanjang pembuatan project nanti, kita hanya butuh mengubah kurang dari 10 file bawaan Laravel (tidak termasuk file yang akan di tambah).

Berikut penjelasan singkat dari isi setiap folder:

- **App:** berisi berbagai file untuk aplikasi yang akan dibangun. Dalam folder inilah nantinya kita membuat controller dan model.
- **Bootstrap:** berisi file `app.php` yang berfungsi sebagai `bootstrap` atau file pengaturan awal dari Laravel. Selain itu terdapat juga folder `cache` untuk meningkatkan performa aplikasi.

Sebagai info tambahan, nama folder ini tidak ada kaitannya dengan framework CSS **Bootstrap**. Dalam istilah komputer, `bootstrap` memiliki makna yang sama dengan `booting` yakni proses awal dari sebuah aplikasi.
- **Config:** berisi berbagai file konfigurasi Laravel. Folder ini cukup sering kita akses karena di sinilah berbagai pengaturan Laravel tersimpan.
- **Database:** berisi folder dan file yang 'mengurus' database seperti `migrations`, `factories` dan `seeds`. Folder database ini juga bisa dipakai sebagai tempat menyimpan file tabel database SQLite (jika kita menggunakan SQLite).
- **Lang:** berisi folder `language` untuk mengubah bahasa (localization).
- **Public:** berisi file `index.php` sebagai file awal dari semua request ke aplikasi Laravel. File `index.php` inilah yang nanti memanggil berbagai file-file lain. Namun kita tidak akan meng-edit file `index.php` ini secara langsung.
- **Resource:** berisi file `resource` 'mentah' seperti file CSS dan JavaScript. Di sini juga nantinya kita membuat view.
- **Routes:** berisi file yang akan menangani proses routing.
- **Storage:** berisi tempat penyimpanan file yang di generate oleh Laravel, seperti file log. Jika kita membuat form upload, file tersebut juga bisa disimpan di sini.
- **Test:** berisi file untuk proses testing seperti PHPUnit.
- **Vendor:** Berisi berbagai file internal framework Laravel. Di sinilah semua library `dependency` Laravel berada. Folder vendor ini mencakup 95% ukuran framework laravel (33MB yang terdiri lebih dari 7400an file). Meskipun berisi banyak file, folder ini tidak perlu kita utak-atik.

Sepanjang pembuatan aplikasi, folder yang akan sering kita akses adalah **app** (untuk membuat model dan controller), **resource** (untuk membuat view) dan **routes** (untuk membuat routing). Penjelasan lebih lengkap akan kita bahas ketika masuk ke setiap materi tersebut.

4.2. Cara Mengakses Laravel

Selama ini, cara kita mengakses sebuah file PHP adalah dengan menjalankan XAMPP lalu membukanya dari alamat localhost. Ini sebenarnya juga masih bisa dipakai untuk mengakses Laravel.

Namun Laravel menyediakan cara khusus dengan memanfaatkan web server bawaan PHP. Mulai dari PHP versi 5.4 ke atas, PHP sudah menyertakan web server internal sendiri. Inilah yang dipakai Laravel untuk memudahkan proses pembuatan aplikasi.

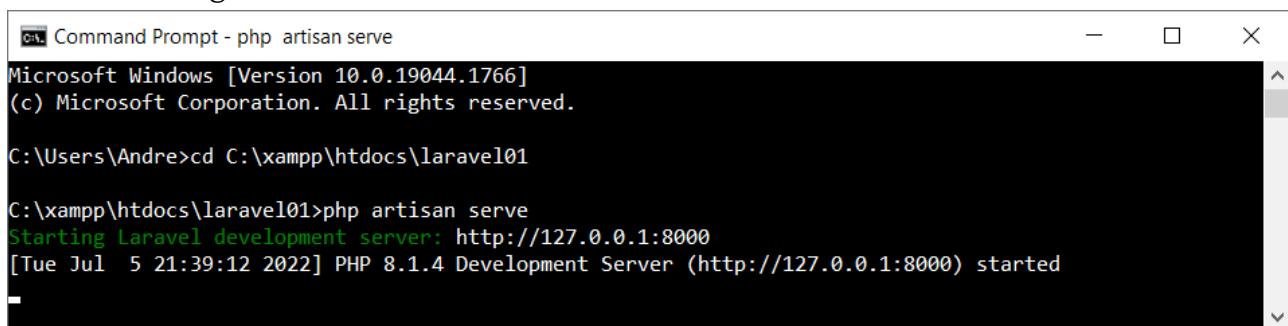
Untuk menjalankan server bawaan PHP ini, kita harus menggunakan cmd. Silahkan buka cmd lalu masuk ke folder instalasi Laravel. Karena file Laravel yang saya gunakan ada di C:\xampp\htdocs\laravel01, maka perintah untuk pindah directory adalah:

```
cd C:\xampp\htdocs\laravel01
```

Setelah berada di dalam folder Laravel, jalankan web server dengan perintah berikut:

```
php artisan serve
```

Lalu akhiri dengan tombol Enter.



```
Command Prompt - php artisan serve
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Andre>cd C:\xampp\htdocs\laravel01
C:\xampp\htdocs\laravel01>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Tue Jul  5 21:39:12 2022] PHP 8.1.4 Development Server (http://127.0.0.1:8000) started
```

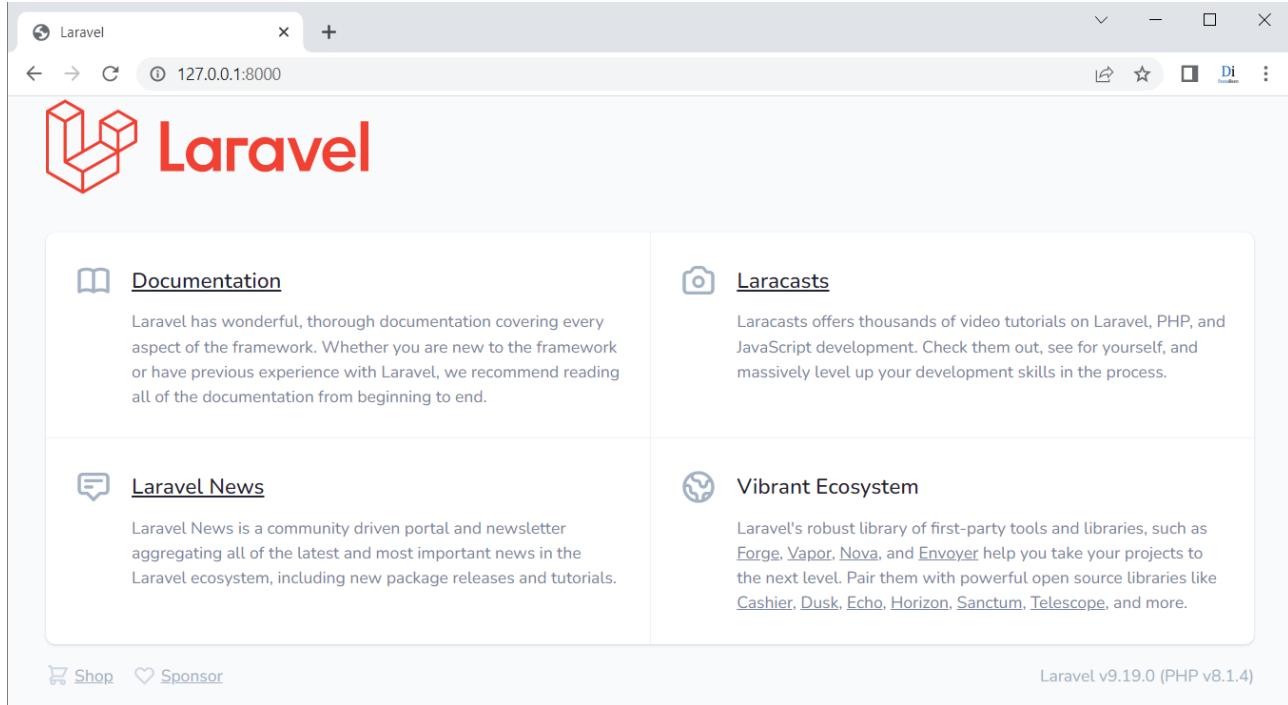
Gambar: Menjalankan Laravel server

Sesaat kemudian cursor cmd akan berhenti, maksudnya kita tidak bisa mengetik apa-apa lagi di cmd. Ini merupakan hal normal dan itu artinya server PHP sudah aktif. Selain itu juga akan tampil teks:

```
PHP 8.1.4 Development Server (http://127.0.0.1:8000) started
```

Artinya server sudah aktif dan bisa diakses dari <http://127.0.0.1:8000>. Silahkan buka web browser dan ketik alamat ini.

Jika anda menggunakan aplikasi anti virus atau firewall, ada kemungkinan perintah `php artisan serve` di blokir oleh aplikasi tersebut. Solusinya izinkan hal ini dengan cara memasukkan file `php.exe` sebagai *exception*. Atau bisa juga dengan mematikan anti virus sementara waktu (meskipun kurang disarankan).



Gambar: Tampilan awal Laravel

Sip, Laravel sudah berhasil diakses! Dan karena alamat `127.0.0.1` tidak lain adalah `localhost`, maka halaman di atas juga bisa diakses dari `http://localhost:8000`.

Ada beberapa hal yang perlu kita bahas:

Pertama, kita tidak butuh menjalankan web server Apache bawaan XAMPP, karena yang sedang berjalan saat ini adalah server internal bawaan PHP.

Kedua, karena tidak perlu menjalankan Apache bawaan XAMPP, maka sebenarnya kita tidak harus menginstall Laravel di folder `htdocs`. Bisa saja Laravel di install ke drive D atau tempat lain kemudian jalankan perintah `php artisan serve` dari folder tersebut.

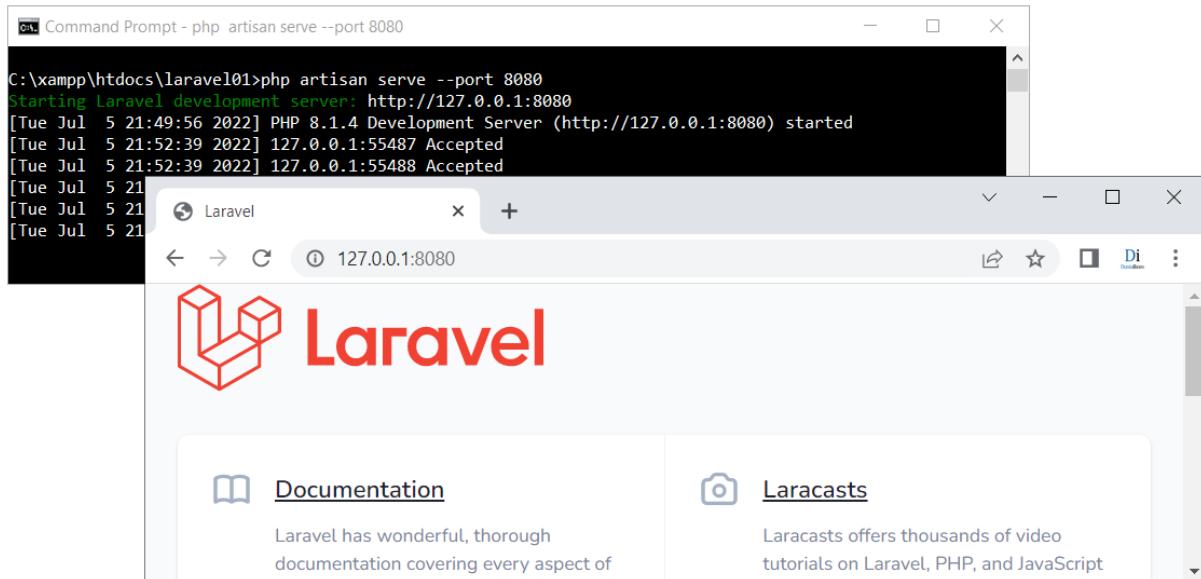
Ketiga, jendela cmd yang kita pakai menjalankan perintah `php artisan serve` tidak boleh ditutup! Apabila jendela ini ditutup, server otomatis juga akan berhenti.

Keempat, secara default port yang dipakai adalah 8000. Kita bisa ganti port ini dengan nomor lain dengan tambahan perintah `--port nomor_port`. Sebagai contoh, jika saya ingin menjalankan server di port 8080, bisa menjalankan perintah berikut:

```
php artisan serve --port 8080
```

Untuk OS Linux atau Mac OS, perintahnya adalah `php artisan serve -port:8080`

Mengakses Laravel

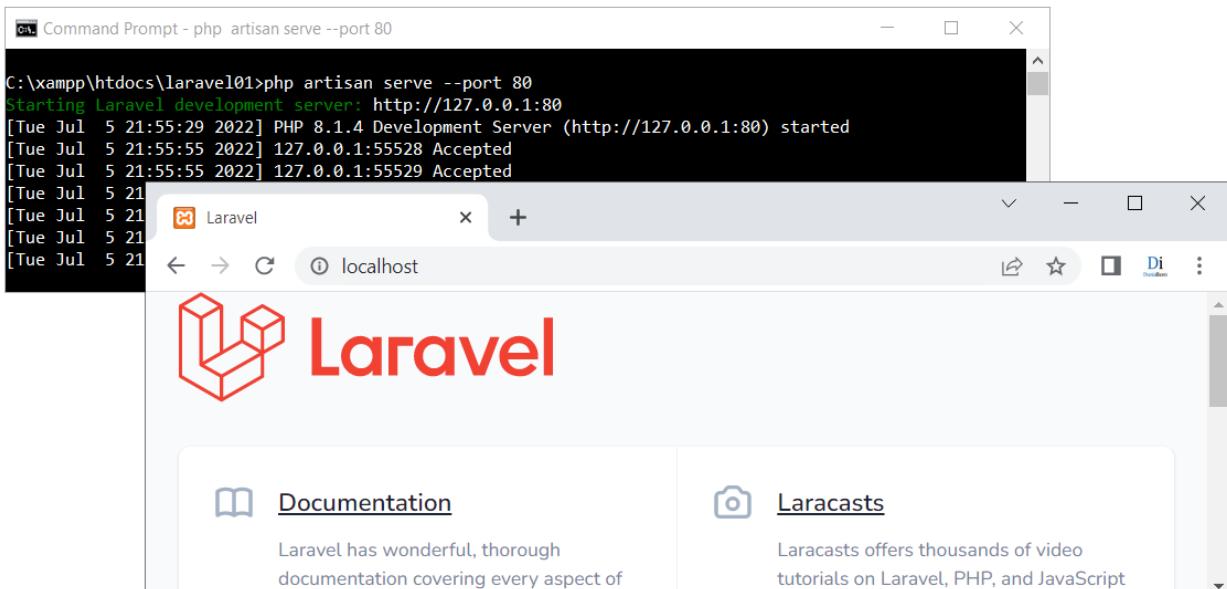


Gambar: Menjalankan PHP server di nomor port 8080

Kita juga bisa menjalankan `php artisan serve` di port 80, sehingga bisa diakses dari alamat `http://localhost` tanpa harus menulis nomor port. Namun syaratnya port 80 tidak sedang dipakai oleh aplikasi lain, termasuk web server Apache bawaan XAMPP.

Perintah untuk menjalankannya adalah:

```
php artisan serve --port 80
```



Gambar: Menjalankan PHP server di nomor port 80

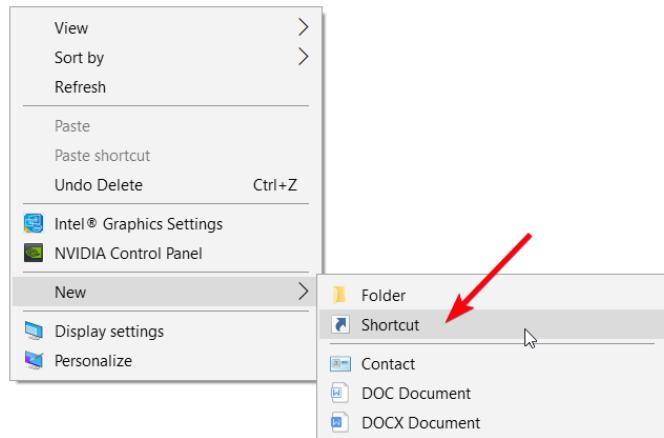
Terakhir, dengan menggunakan nomor port yang berbeda-beda, maka kita bisa menjalankan beberapa project sekaligus, misalnya project 1 diakses di alamat `http://localhost:8000`, project 2 di `http://localhost:8001`, project 3 di `http://localhost:8002`, dst. Setiap server nantinya perlu dijalankan dari cmd terpisah.

4.3. Membuat Shortcut Untuk Laravel Server

Setiap kali ingin mengakses Laravel, maka server juga harus dijalankan dari cmd. Agar lebih mudah, saya akan buat shortcut cmd di desktop agar bisa langsung menjalankan server Laravel.

Catatan: langkah ini sepenuhnya opsional dan tidak harus dibuat. Jika anda mengalami kendala, boleh dilewati saja dan jalankan server seperti yang kita bahas sebelumnya. Pembuatan shortcut ini saya praktekkan di Windows 10 64-bit. Bisa jadi caranya sedikit berbeda untuk versi Windows lain.

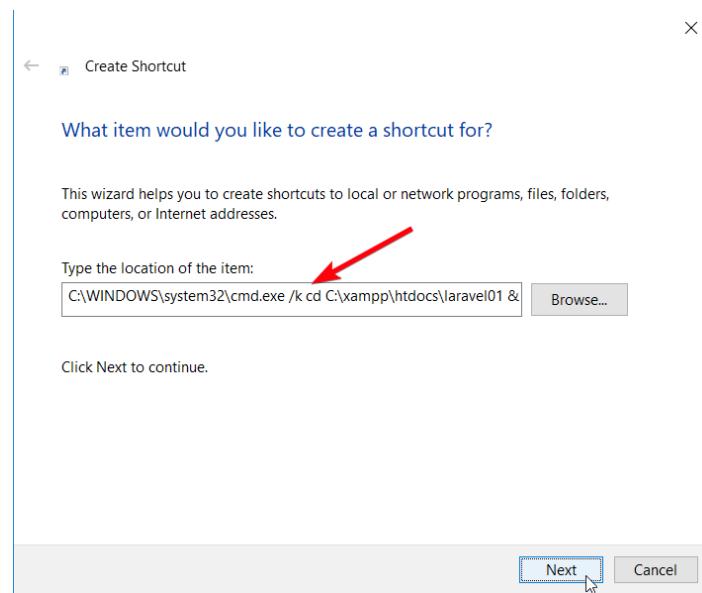
Untuk membuat shortcut cmd, klik kanan pada sembarang tempat di desktop, lalu pilih menu **New -> Shortcut**.



Gambar: Pilih menu New -> Shortcut

Di kolom inputan 'Type the location of the item', ketik perintah berikut (atau copy-paste saja):

```
C:\WINDOWS\system32\cmd.exe /k cd C:\xampp\htdocs\laravel01 & php artisan serve
```



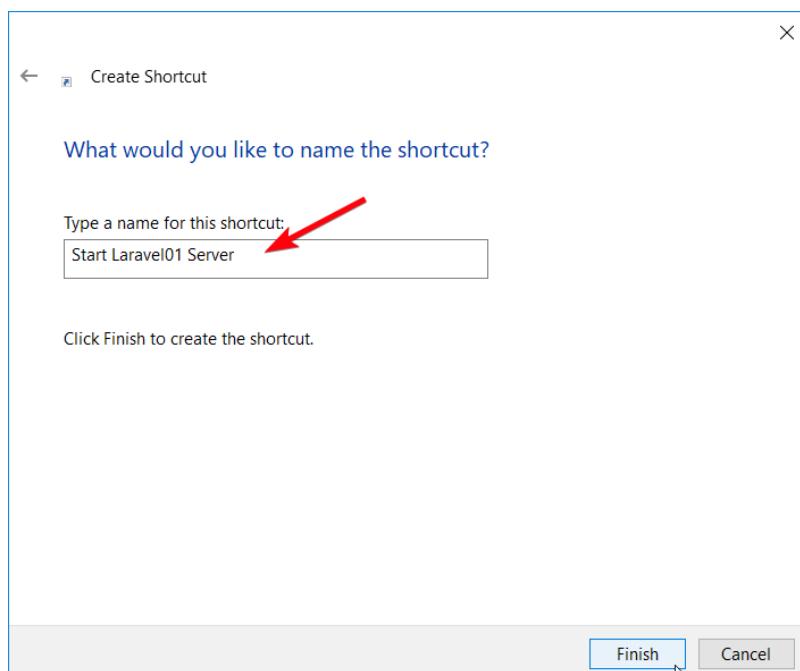
Gambar: Tulis lokasi dan perintah shortcut

Perintah di atas bisa dibaca sebagai berikut:

Buat shortcut dari program yang ada di **C:\WINDOWS\system32\cmd.exe**, yakni lokasi aplikasi cmd Windows. Kode **/k** adalah instruksi untuk cmd bahwa akan ada perintah tambahan, yakni **cd C:\xampp\htdocs\laravel01** untuk memindahkan directory aktif ke **C:\xampp\htdocs\laravel01**, kemudian **& php artisan serve** yang akan langsung menjalankan perintah **php artisan serve** di dalam folder tersebut.

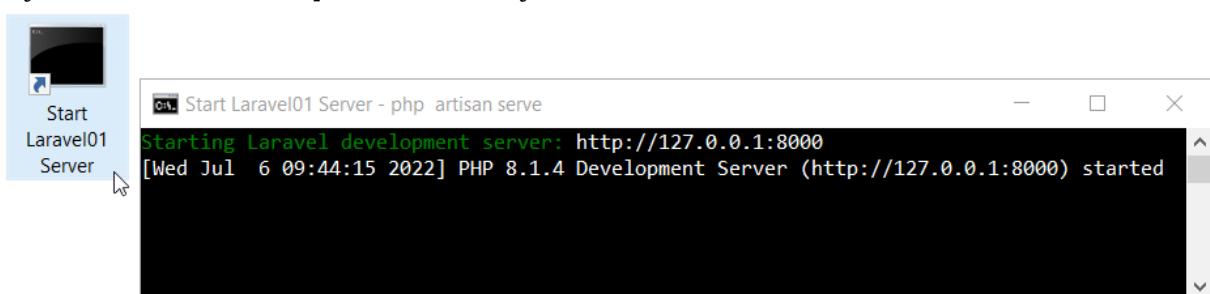
Jika folder Laravel berada di tempat lain, anda bisa sesuaikan perintah **cd C:\xampp\htdocs\laravel01**. Akhiri dengan men-klik tombol **Next**.

Jendela berikutnya berupa inputan kolom 'Type a name for this shortcut', yakni nama dari shortcut. Anda boleh bebas ingin menamakan shortcut ini, dimana saya akan menulis "Start Laravel01 Server". Akhiri dengan klik tombol **Finish**.



Gambar: Tulis nama shortcut

Sekarang di desktop akan tampil shortcut "**Start Laravel01 Server**". Setiap kali ingin menjalankan server, cukup double klik saja:



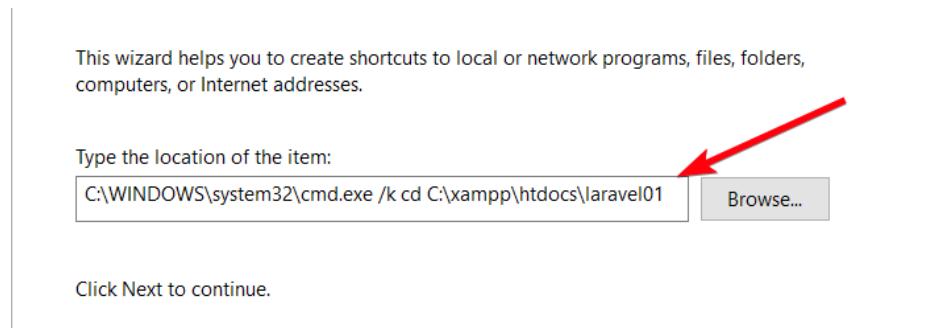
Gambar: shortcut untuk menjalankan php artisan serve

Mengakses Laravel

Cara yang sama juga bisa dipakai untuk membuat shortcut cmd agar langsung membuka folder cd C:\xampp\htdocs\laravel01. Nantinya kita akan banyak mengetik perintah cmd dari dalam folder ini.

Kembali klik kanan pada sembarang tempat di desktop, lalu pilih menu **New -> Shortcut**. Di kolom inputan 'Type the location of the item', ketik perintah berikut (atau copy-paste saja):

```
C:\WINDOWS\system32\cmd.exe /k cd C:\xampp\htdocs\laravel01
```



Gambar: Tulis lokasi dan perintah shortcut

Klik tombol **Next** dan untuk nama shortcut akan saya tulis "Laravel01 Project". Sekarang jika perlu masuk ke folder C:\xampp\htdocs\laravel01 dari cmd, cukup klik shortcut saja:



Gambar: shortcut untuk masuk ke folder C:\xampp\htdocs\laravel01 dari cmd

Kedua shortcut ini akan memudahkan kita sepanjang pembuatan project di Laravel.

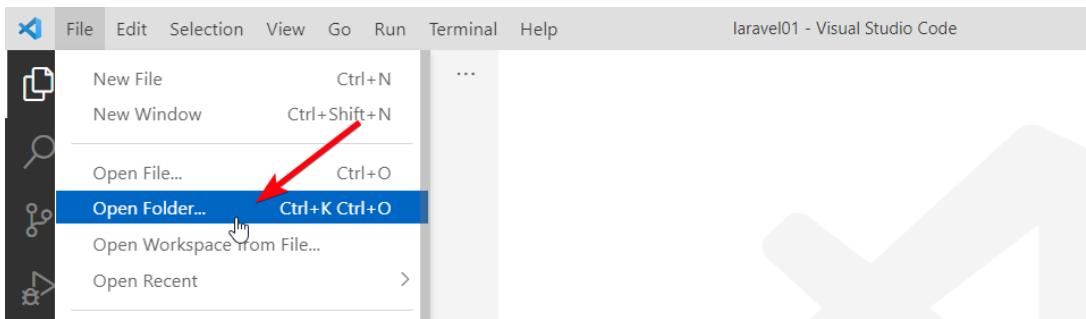
Dalam bab ini telah dibahas cara akses Laravel menggunakan server bawaan PHP, yakni dari perintah `php artisan serve`. **Artisan** sendiri merupakan perintah cmd bawaan Laravel yang akan sering di pakai.

Berikutnya kita akan mulai *coding* di Laravel.

5. Route

Semua persiapan sudah selesai, maka kita bisa masuk ke pembahasan kode program Laravel itu sendiri. Materi pertama adalah tentang **Route** atau **Routing**.

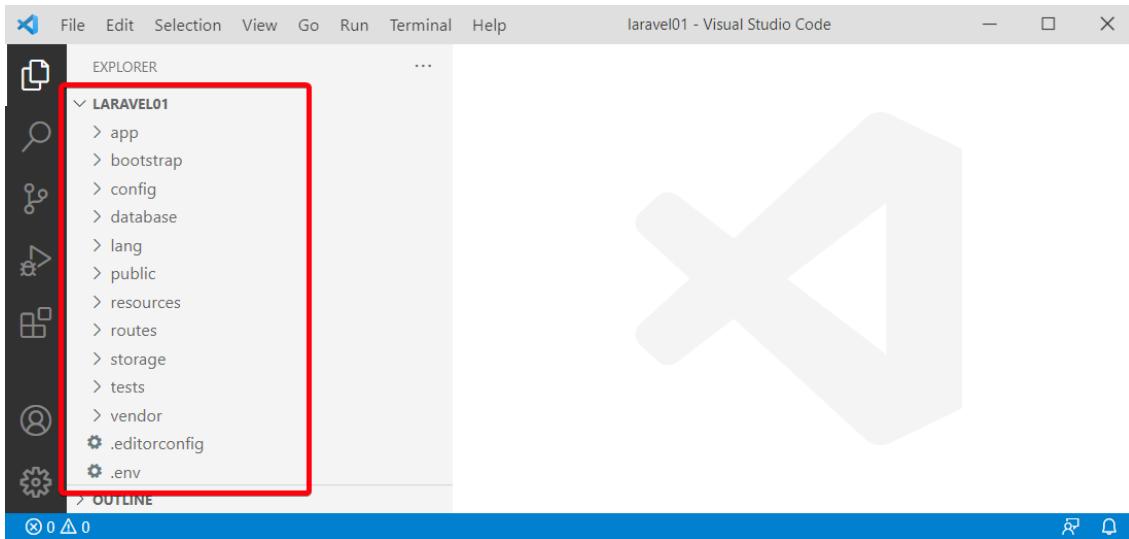
Sebelum itu, silahkan import / buka folder Laravel ke dalam text editor agar mudah beralih dari satu file ke file lain. Jika anda menggunakan **VS Code**, pilih menu **File -> Open Folder...** atau bisa juga dengan kombinasi **Ctrl + O**.



Gambar: Membuka Folder di VS Code

Akan tampil jendela Windows Explorer, kemudian cari folder Laravel yang akan dipakai. Dalam contoh ini saya menggunakan folder **laravel01** yang berada di **C:\xampp\htdocs**, tidak masalah jika anda menggunakan lokasi lain. Pilih folder ini lalu klik tombol **Open**.

Kita kembali ke aplikasi VS Code, di sisi kiri tampil semacam jendela explorer yang berisi semua file dari folder Laravel tersebut. Nantinya cukup klik jendela ini jika ingin berpindah dari satu file ke file lain.



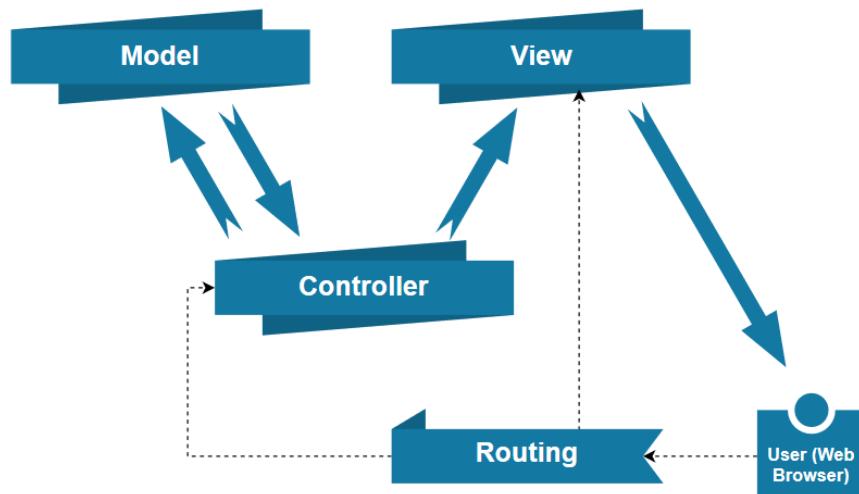
Gambar: Jendela explore di VS Code

Jika anda menggunakan text editor lain, kemungkinan besar juga memiliki menu yang sama. Atau jika pun tidak ada (seperti di **NotePad++**), juga tidak masalah. Silahkan pakai Windows Explorer untuk beralih dari satu file ke file lain.

Sepanjang buku ini saya juga menggunakan theme warna putih untuk VS Code (*light color theme*). Ini semata-mata agar lebih mudah terbaca di buku versi cetak. Jika tertarik mengubah warna theme VS Code, bisa mengaksesnya dari menu **File -> Preferences -> Color Theme**.

5.1. Pengertian Routing

Di bab 2 kita telah membahas konsep **MVC + Routing**. Sebagai pengingat, berikut saya tampilkan kembali diagram alur kerja MVC:



Gambar: Diagram Alur MVC + Routing

Route atau **Routing** berperan sebagai penghubung antara user dengan keseluruhan framework. Dalam Laravel, setiap alamat web yang kita ketik di web browser akan melewati route terlebih dahulu. Route-lah yang menentukan ke mana proses akan dibawa, apakah ke Controller atau ke View.

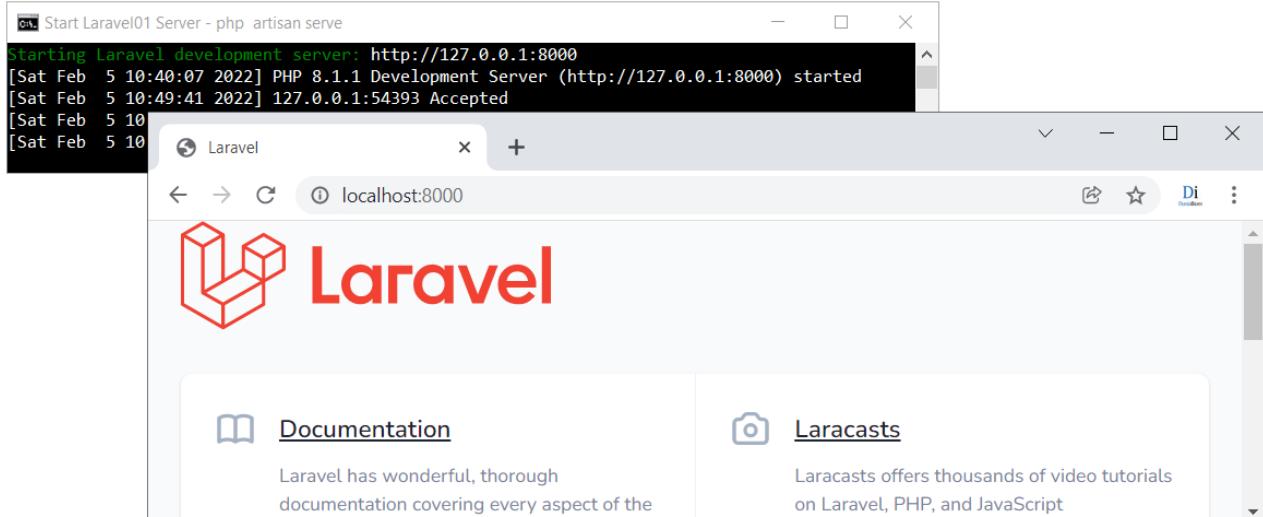
Khusus dalam bab ini, kita juga akan melihat bahwa di dalam Laravel, route juga bisa dikembalikan langsung ke user (tanpa melewati view maupun controller).

Istilah **Route**, **Router** dan **Routing** kadang saling di pertukarkan. Ketiga istilah ini memiliki perbedaan dari cara penggunaannya dalam konteks bahasa Inggris, yakni apakah merujuk ke kata benda atau kata kerja. Dalam pembahasan buku ini, kita anggap saja ketiganya merujuk ke hal yang sama.

5.2. Route Bawaan Laravel

Secara default, Laravel sudah menyertakan 1 route bawaan. Sebelum masuk ke kode program, mari kita lihat hasil tampilan dari route ini. Jalankan server PHP dengan perintah `php artisan serve` di **cmd**, atau klik shortcut *Start Laravel01 Server* (seperti yang kita praktekkan dalam bab sebelumnya).

Kemudian buka web browser dan ketik alamat `http://localhost:8000`. Hasil tampilan ini sebenarnya berasal dari sebuah proses route di dalam Laravel.



Gambar: Laravel diakses dari alamat `http://localhost:8000`

Pada saat kita mengetik alamat `http://localhost:8000` dan menekan tombol Enter, maka sebuah kode program route Laravel akan dipanggil untuk memproses alamat tersebut.

Kode program ini berada di dalam file `routes/web.php`. Silahkan buka file ini:

The screenshot shows the Visual Studio Code interface with the file `routes/web.php` open in the editor. The code is as follows:

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 /*
6 |-----|
7 | Web Routes
8 |-----|
9 |
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19

```

The file is located in the `LARAVEL01/routes` directory. The `web.php` file is highlighted in the Explorer sidebar with a red arrow pointing to it.

Gambar: Tampilan isi file `routes/web.php`

Di dalam folder **route** sebenarnya terdapat 3 file lain, yakni `api.php`, `channels.php` dan `console.php`. Semuanya juga dipakai untuk membuat route, tapi bukan route yang ditujukan untuk web browser.

Sebagai contoh, file `routes/api.php` dipakai untuk membuat route **API** (Application Programming Interface). API ini berguna jika Laravel digabung dengan front-end JavaScript seperti **React** atau untuk interface aplikasi android.

Sepanjang buku ini, kita hanya fokus ke file `routes/web.php` saja, karena inilah route yang di pakai untuk memproses alamat web.

Secara bawaan, file `routes/web.php` sudah berisi beberapa baris kode program:

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 /*
6 |-----[ Web Routes ]-----|
7 |
8 |-----[-----|
9 |
10| Here is where you can register web routes for your application. These
11| routes are loaded by the RouteServiceProvider within a group which
12| contains the "web" middleware group. Now create something great!
13|
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19

```

Di baris 3 terdapat perintah untuk proses import **Route facade**. Secara sederhana, **facade** adalah istilah Laravel untuk menyebut class bantu. Route class terdiri dari berbagai method yang akan kita bahas secara bertahap. Dalam penggunaan normal, perintah di baris 3 ini tidak perlu kita utak-atik.

Berikutnya di baris 5 - 14 terdiri dari komentar yang menjelaskan isi file. Hampir setiap file bawaan Laravel berisi komentar tentang maksud dan cara menggunakan file. Kadang baris komentar malah lebih panjang daripada file kode program sebenarnya. Ini sangat memudahkan kita dalam mempelajari setiap perintah yang ada.

Kode program dari `route.php` ini terdapat di 3 baris terakhir, yakni:

```

1 Route::get('/', function () {
2     return view('welcome');
3 });

```

Inilah kode program yang dipanggil untuk memproses alamat `http://localhost:8000`. Secara sederhana, penulisan route Laravel mengikuti format berikut:

```
Route::<jenis method>(<alamat URL>, <proses yang dijalankan>)
```

Pertama, kita harus menulis perintah 'Route::'. Di dalam konsep OOP PHP, tanda `::` merupakan perintah untuk mengakses sebuah **static method** (atau bisa juga **static property**) kepunyaan sebuah class, yang dalam contoh ini adalah milik class Route.

Selanjutnya diikuti dengan penulisan `<jenis method>`. Jenis method adalah 'cara sebuah URL diakses'. 'get' merupakan salah satu jenis method yang akan dijalankan ketika alamat URL diakses secara normal.

Maksud "normal" di sini adalah dengan mengetik langsung alamat web di *address bar* web browser atau ketika kita men-klik sebuah link. Nantinya terdapat jenis method lain seperti `post`, `put`, dan `delete`. Kita akan bahas jenis method ini pada bab tersendiri.

Untuk saat ini bisa disimpulkan bahwa jika ingin membuat route untuk sebuah alamat web, method yang dipakai adalah `get`.

Setelah itu dalam tanda kurung terdapat `<alamat URL>`. Ini merupakan alamat URL yang akan di proses oleh route. Dalam contoh route bawaan Laravel, alamat URL ini berisi 1 karakter saja, yakni: `'/'`. Tanda *forward slash* dipakai untuk merujuk ke alamat *home* atau *root*, yakni alamat ketika web dipanggil tanpa menulis tambahan apapun seperti `http://localhost:8000`.

Terakhir terdapat inputan untuk `<proses yang dijalankan>`. Pada bagian inilah kita menulis kode program yang akan dijalankan oleh route, yakni apakah memanggil view, memanggil controller, atau hal lain. Dalam contoh bawaan Laravel, proses ini berbentuk *anonymous function*, atau kadang disebut sebagai *closure* atau *callback function*:

```
1 function () {
2     return view('welcome');
3 }
```

Anonymous function atau *closure* adalah sebuah function yang tidak memiliki nama.

Umumnya *closure* dipakai ketika membuat function yang hanya perlu dijalankan 1 kali saja.

Perintah di atas akan menjalankan sebuah **View** yang bernama '`welcome`'. View ini pada dasarnya berbentuk file php yang akan kita bahas secara terpisah dalam 1 bab khusus. Untuk saat ini, boleh abaikan terlebih dahulu maksud dari perintah `return view('welcome')`.

Sebagai kesimpulan, tiga baris kode program yang ada di dalam `route.php` bisa dibaca: **Jika halaman home diakses, jalankan view yang bernama welcome.**

Nantinya, format penulisan route bisa lebih kompleks dari ini dan isi file route sendiri tidak hanya berbentuk *anonymous function* saja, tapi juga bisa nama view, nama controller, atau hal lain. Kita akan bahas hal ini secara bertahap.

Jika anda sedikit bingung dengan penjelasan ini, jangan khawatir. Kita akan coba bahas kembali dengan membuat route dari nol.

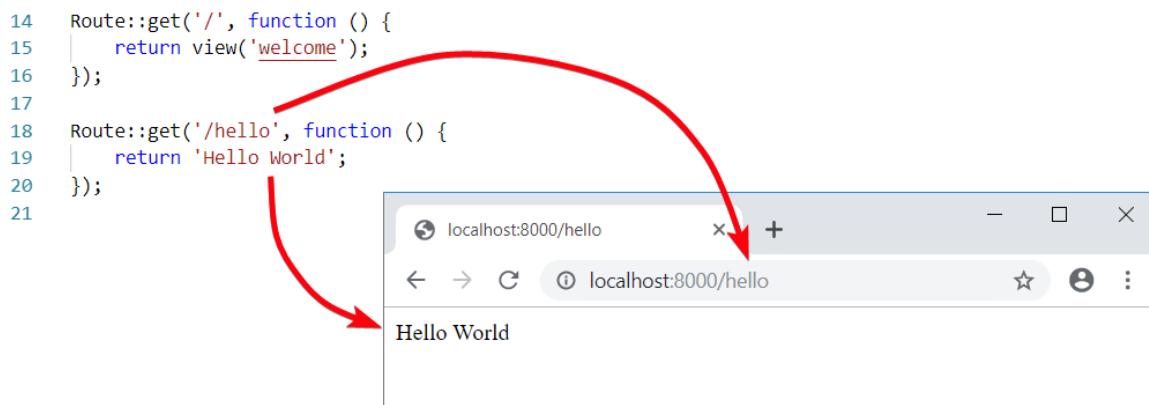
5.3. Membuat Route

Untuk membuat route, cukup tulis pemanggilan static method **Route** baru ke dalam file `routes/web.php`. Sebagai contoh, tulis kode program berikut di bawah bawaan Laravel:

`routes/web.php`

```
1 Route::get('/hello', function () {
2     return 'Hello World';
3});
```

Di sini saya menulis '/hello' sebagai argument pertama method `Route::get`. Sebelumnya sudah kita pelajari bahwa argument pertama ini adalah tempat untuk <alamat URL>. Jika ditulis seperti ini, maka sebuah route dengan alamat `http://localhost:8000/hello` sudah bisa diakses. Berikut hasilnya:



Gambar: Hasil tampilan dari `http://localhost:8000/hello`

Ketika alamat `http://localhost:8000/hello` diakses, maka perintah `return 'Hello World'` akan dijalankan. Hasilnya di web browser tampil teks 'Hello World'.

Isi dari *anonymous function* ini juga bisa berbentuk kode PHP biasa, tidak harus perintah `return`. Sebagai contoh kedua, saya akan tambah route baru untuk alamat `http://localhost:8000/belajar`:

`routes/web.php`

```
1 Route::get('/belajar', function () {
2     echo '<h1>Hello World</h1>';
3     echo '<p>Sedang belajar Laravel</p>';
4});
```

Gambar: Hasil tampilan dari `http://localhost:8000/belajar`

Route

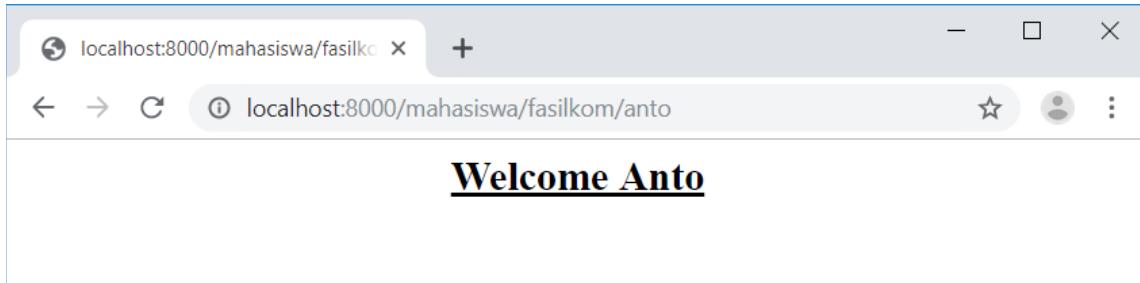
Sekarang isi dari *anonymous function* terdiri dari 2 buah perintah echo. Keduanya berisi tag <h1> dan <p> yang akan di proses web browser sebagai tag HTML biasa.

Alamat URL di dalam route juga bisa terdiri dari beberapa segmen, seperti contoh berikut:

routes/web.php

```
1 Route::get('/mahasiswa/fasilkom/anto', function () {
2     echo '<h2 style="text-align: center"><u>Welcome Anto</u></h2>';
3 });
```

Jika ditulis seperti ini, maka alamat URL-nya adalah <http://localhost:8000/mahasiswa/fasilkom/anto>.



Gambar: Hasil tampilan dari <http://localhost:8000/mahasiswa/fasilkom/anto>

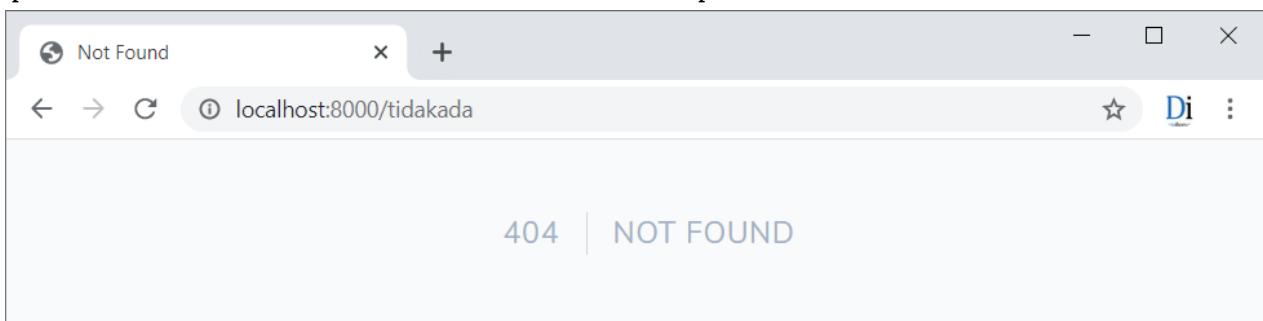
Sampai di sini semoga anda bisa memahami prinsip dasar cara kerja route Laravel. Intinya, route akan "menampung" semua alamat yang diketik di web browser, lalu menampilkan hasilnya.

Dengan route, alamat sebuah URL tidak harus bersesuaian dengan file fisik yang ada di server.

Ketika mempelajari PHP dasar, kita biasa pahami bahwa jika ditulis alamat <http://localhost/belajar.php>, maka bisa di tebak itu akan mencari file *belajar.php* di folder *htdocs*.

Namun di Laravel, alamat URL ini hanyalah 'alamat virtual'. Jika ditulis <http://localhost:8000/mahasiswa/fasilkom/anto>, maka belum tentu di server terdapat folder dengan nama *mahasiswa/fasilkom/anto*, karena alamat tersebut sebenarnya di proses oleh route.

Artinya, dalam Laravel harus ada route yang bertugas untuk "menangkap" setiap alamat. Apabila route tidak ditemukan, Laravel akan menampilkan **halaman 404**:



Gambar: Tampilan halaman 404 bawaan Laravel

Setelah kita membuat beberapa route tambahan, berikut hasil akhir dari file routes\web.php:

```

1 Route::get('/', function () {
2     return view('welcome');
3 });
4
5 Route::get('/hello', function () {
6     return 'Hello World';
7 });
8
9 Route::get('/belajar', function () {
10    echo '<h1>Hello World</h1>';
11    echo '<p>Sedang belajar Laravel</p>';
12 });
13
14 Route::get('/mahasiswa/fasilkom/anto', function () {
15    echo '<h2 style="text-align: center"><u>Welcome Anto</u></h2>';
16 });

```

Isi route ini nantinya bisa cukup banyak, tergantung kebutuhan kita. Route di Laravel juga memiliki berbagai fitur lanjutan seperti route dinamis, redirect hingga menggunakan *regular expression*. Kita akan bahas satu per satu.

It's just work!

Salah satu tips dasar yang sering terdengar ketika kita belajar framework seperti Laravel adalah: "Lupakan mayoritas cara kerja PHP standar yang pernah anda ketahui".

Tips ini terdengar aneh karena Laravel itu juga dibuat dari PHP. Namun Laravel memang memiliki cara kerja yang 'ajaib'. Misalnya bagaimana bisa alamat yang ditulis di web browser diterima oleh route? Lalu kenapa hasil return dari sebuah *anonymous function* bisa tampil di web browser?

Jawaban untuk pertanyaan ini bisa diketahui jika kita benar-benar memahami cara kerja dari setiap komponen dasar Laravel. Namun ini sangat rumit dan terdiri dari puluhan ribu baris kode program.

Karena itu kebanyakan orang cukup 'menerima Laravel apa adanya'. Maksudnya jika Laravel sudah mengatur cara penulisan tertentu, terima saja lah karena hasilnya memang sesuai. *It's just work!*

5.4. Route Parameter

Dalam situasi tertentu, kadang kita ingin mengambil nilai yang ada di dalam alamat URL. Menggunakan PHP dasar, ini bisa dilakukan dengan *query string*. Sebagai contoh, jika alamat URL ditulis sebagai `http://localhost/mahasiswa.php?nama=Rani`. Maka nilai 'Rani' ini bisa diambil dari variabel `$_GET['nama']` di halaman `mahasiswa.php`.

Route

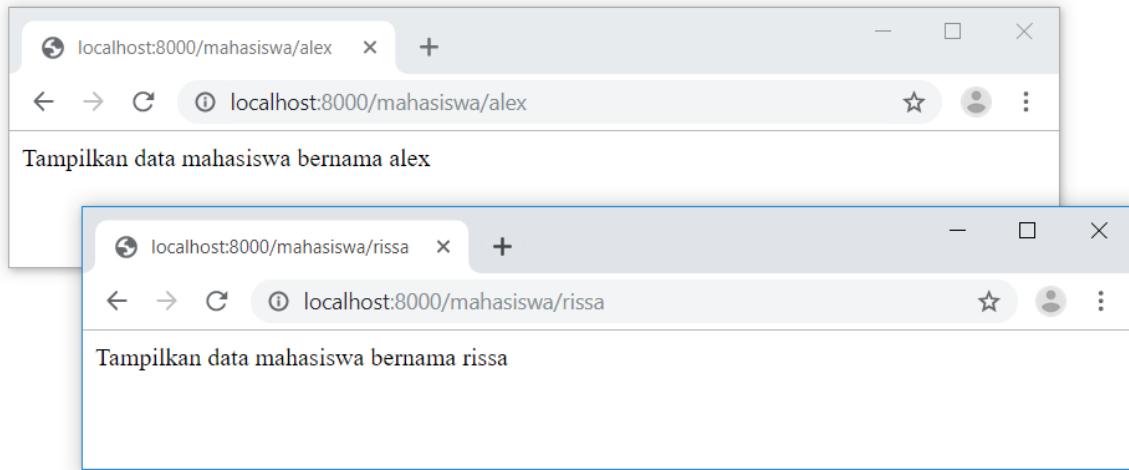
Dalam Laravel, caranya adalah sebagai berikut:

routes/web.php

```
1 Route::get('/mahasiswa/{nama}', function ($nama) {  
2     return "Tampilkan data mahasiswa bernama $nama";  
3 });
```

Di argument pertama method `Route::get()`, terdapat string '`mahasiswa/{nama}`'. Bagian segmen URL yang ditulis dalam kurung kurawal ini akan berfungsi sebagai **route parameter**. **Route parameter** bisa 'ditangkap' oleh kode program dalam *anonymous function*. Syarat lain, argument pertama dari anonymous function harus diisi dengan nama variabel yang ditangkap, yakni `function($nama)`.

Dengan penulisan seperti ini, kita bisa mengakses variabel `$nama` yang ditulis dalam URL:



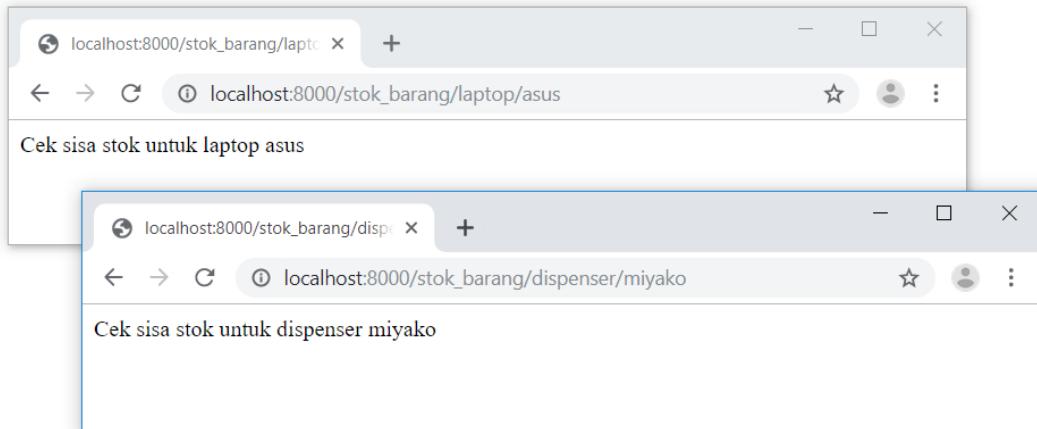
Gambar: Penggunaan route parameter mahasiswa

Lebih jauh lagi, kita bisa membuat 2 atau lebih route parameter. Dimana setiap segmen bisa diisi dengan nama variabel yang berbeda:

routes/web.php

```
1 Route::get('/stok_barang/{jenis}/{merek}', function ($jenis,$merek) {  
2     return "Cek sisa stok untuk $jenis $merek";  
3 });
```

Route



Gambar: Penggunaan route parameter stok_barang

Route parameter ini menyederhanakan cara pengambilan nilai URL, selain itu tampilan alamat web menjadi lebih rapi karena kita tidak lagi menggunakan *query string*.

Penulisan nama variabel di *anonymous function* sebenarnya tidak harus sama dengan nilai route parameter. Kita bisa menulis sebagai berikut:

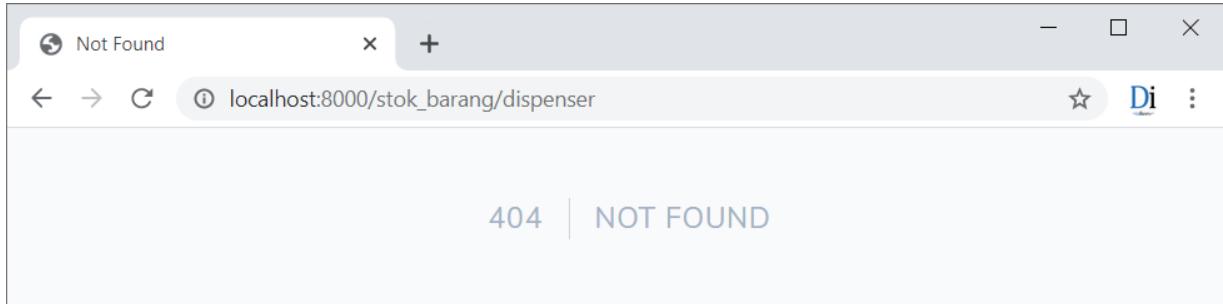
routes/web.php

```
1 Route::get('/stok_barang/{jenis}/{merek}', function ($a,$b) {
2     echo "Cek sisa stok untuk $a $b";
3 });
```

Meskipun route parameter menggunakan `{jenis}/{merek}`, tapi dalam *anonymous function* saya ganti menjadi `($a,$b)`. Ini tidak masalah karena yang dibutuhkan Laravel hanyalah urutan variabel saja. Variabel `$a` akan berpasangan dengan `{jenis}`, serta variabel `$b` dengan `{merek}`.

5.5. Route dengan Optional Parameter

Dalam contoh sebelum ini, jika salah satu route parameter tidak diisi akan tampil halaman 404:



Gambar: Halaman 404 karena route tidak ditemukan

Di sini saya mencoba mengakses `http://localhost:8000/stok_barang/dispenser/`, hasilnya tampil halaman 404 karena tidak ada route untuk alamat tersebut. Route `stok_barang` harus ditulis dengan 2 segmen, yakni untuk `jenis` dan `merek`.

Route

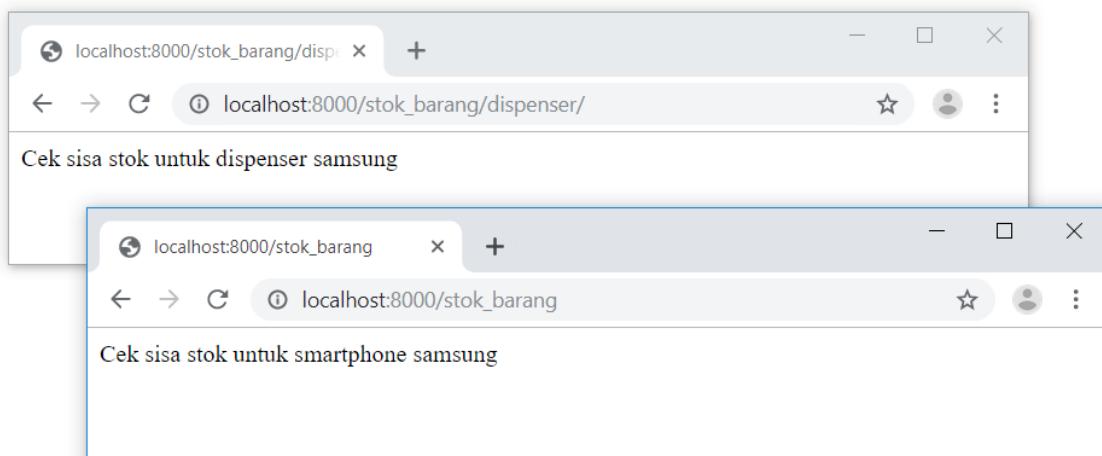
Kita bisa membuat salah satu atau kedua segmen ini menjadi opsional. Maksudnya jika nama segmen tidak ditulis, pakai nilai default. Inilah yang disebut sebagai **optional parameters**. Berikut cara penulisannya:

routes/web.php

```
1 Route::get('/stok_barang/{jenis?}/{merek?}',  
2     function ($a = 'smartphone', $b = 'samsung') {  
3         return "Cek sisa stok untuk $a $b";  
4     });
```

Untuk penulisan parameter, sekarang ditambah dengan tanda tanya, yakni `{jenis?}/{merek?}`. Kemudian di dalam *anonymous function* ditulis menjadi `function ($a = 'smartphone', $b = 'samsung')`.

Artinya jika segmen yang menampung nilai jenis barang tidak ditulis, pakai 'smartphone' sebagai nilai default, begitu pula jika segmen yang menampung merek barang tidak ditulis, pakai nilai 'samsung'. Berikut hasilnya:



Gambar: Hasil penggunaan optional parameter

Terlepas dari apakah ada dispenser bermerek samsung, mudah-mudahan anda bisa memahami cara penggunaan *optional parameter* ini.

5.6. Route Parameter dengan Regular Expression

Route parameter juga bisa dibatasi dengan *regular expression*. Ini dipakai jika kita ingin penulisan alamat URL harus memenuhi pola tertentu. Sebagai bahan praktik, perhatikan route berikut:

routes/web.php

```
1 Route::get('/user/{id}', function ($id) {  
2     return "Tampilkan user dengan id = $id";  
3 });
```

Route

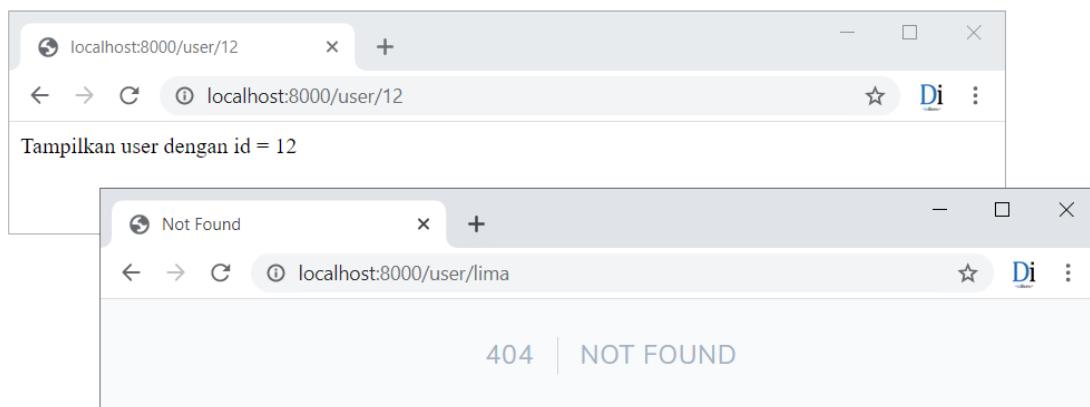
Route ini bisa diakses dari alamat `localhost:8000/user/12`, `localhost:8000/user/3402` maupun `localhost:8000/user/lima`. Artinya, apapun nilai segmen terakhir tetap bisa diproses oleh Laravel.

Namun biasanya, `id` terdiri dari angka saja. `Id` yang diinput dengan huruf seperti 'tujuh', seharusnya tidak bisa diakses. Untuk pembatasan seperti ini, kita bisa menggunakan regular expression dengan contoh berikut:

`routes/web.php`

```
1 Route::get('/user/{id}', function ($id) {
2     return "Tampilkan user dengan id = $id";
3 })->where('id', '[0-9]+');
```

Perhatikan tambahan di baris 3, yakni perintah `->where('id', '[0-9]+')`. Kode ini artinya saya ingin agar parameter '`id`' hanya bisa diakses jika memenuhi pola *regular expression* `[0-9]+`. Pola ini berarti segmen '`id`' hanya bisa diisi dengan angka 0 – 9 sebanyak 1 karakter atau lebih. Selain itu, route tidak bisa diakses.



Gambar: Membatasi route dengan pola regular expression

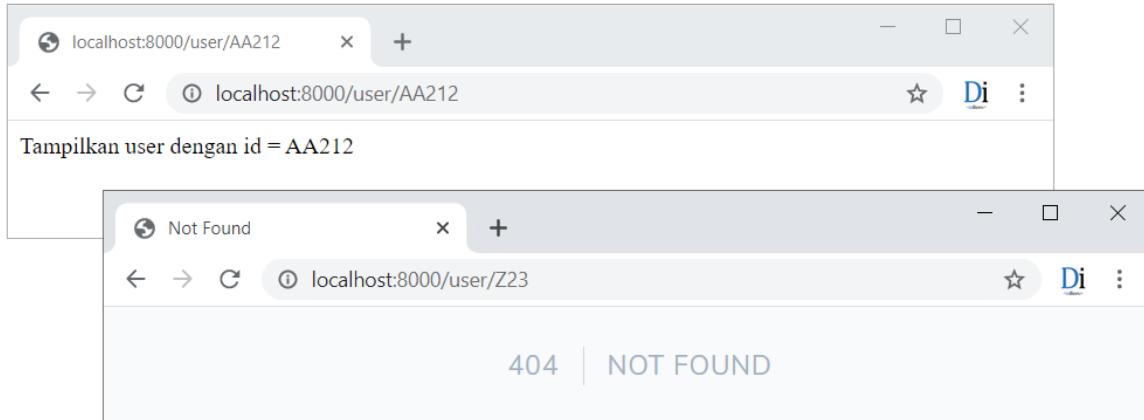
Pola regular expression `[0-9]+` akan cocok dengan semua angka minimal 1 digit, seperti 4, 12, maupun 9999. Akan tetapi pola ini tidak akan cocok dengan karakter huruf seperti `lima`. Hasilnya, ketika diakses `localhost:8000/user/lima` akan tampil halaman 404.

Batasan regular expression ini juga bisa dibuat lebih rumit, seperti contoh berikut:

`routes/web.php`

```
1 Route::get('/user/{id}', function ($id) {
2     return "Tampilkan user dengan id = $id";
3 })->where('id', '[A-Z]{2}[0-9]+');
```

Pola `[A-Z]{2}[0-9]+` berarti segmen `id` harus diawali dengan 2 huruf besar, lalu diikuti dengan 1 atau lebih angka. Contoh pola yang memenuhi syarat seperti AA123, JI9999, atau ZE1, namun tidak untuk A12 atau aa12.



Gambar: Membatasi route dengan pola regular expression

Pembatasan route dengan *regular expression* mungkin tidak terlalu sering kita pakai, tapi fitur ini sangat berguna dalam situasi tertentu.

5.7. Route Redirect

Laravel juga menyediakan fitur untuk proses *redirect* antar route. Caranya, gunakan method **Route::redirect** seperti contoh berikut:

```
routes/web.php

1 Route::get('/hubungi-kami', function () {
2     return '<h1>Hubungi Kami</h1>';
3 });
4
5 Route::redirect('/contact-us', '/hubungi-kami');
```

Di sini terdapat 2 route. Route di baris 1 – 3 merupakan route normal untuk alamat `localhost:8000/hubungi-kami`. Di baris 5 terdapat kode untuk membuat *redirect*. Sehingga ketika diakses alamat `localhost:8000/contact-us`, maka halaman langsung dialihkan ke `localhost:8000/hubungi-kami`.

5.8. Route Group

Jika kita memiliki beberapa route yang ingin di proses secara kelompok, bisa menggunakan **route group**. Dengan menggunakan route group, beberapa route bisa di proses sebagai satu kesatuan.

Apa yang bisa diproses untuk route group ini akan kita pelajari secara bertahap, misalnya nanti bisa menjalankan *middleware* tertentu, menggunakan *namespace* tertentu, atau mengakses *subdomain* tertentu.

Sebagai contoh sederhana, kita bisa menggunakan method **Route::prefix()** untuk menambah

Route

awalan ke setiap anggota route. Perhatikan 3 route berikut:

routes/web.php

```
1 Route::get('/admin/mahasiswa', function () {
2     return "<h1>Daftar Mahasiswa</h1>";
3 });
4
5 Route::get('/admin/dosen', function () {
6     return "<h1>Daftar Dosen</h1>";
7 });
8
9 Route::get('/admin/karyawan', function () {
10    return "<h1>Daftar Karyawan</h1>";
11});
```

Ketiga route ini bisa diakses dari alamat:

```
localhost:8000/admin/mahasiswa
localhost:8000/admin/dosen
localhost:8000/admin/karyawan
```

Tidak ada yang salah dari cara ini. Namun karena semua route menggunakan awalan segment yang sama, yakni 'admin/', kita bisa memanfaatkan route group dengan tambahan method

`Route::prefix():`

```
1 Route::prefix('/admin')->group(function () {
2
3     Route::get('/mahasiswa', function () {
4         echo "<h1>Daftar Mahasiswa</h1>";
5     });
6
7     Route::get('/dosen', function () {
8         echo "<h1>Daftar Dosen</h1>";
9     });
10
11    Route::get('/karyawan', function () {
12        echo "<h1>Daftar Karyawan</h1>";
13    });
14
15});
```

Ketiga route sekarang berada di dalam *anonymous function* `Route::prefix('admin')->group(function () { })`.

Method ini akan menambah awalan (*prefix*) '/admin' ke semua route yang ada di dalam group. Ini memudahkan kita jika ingin mengubah '/admin' menjadi prefix lain karena cukup meng-edit di group saja, tidak perlu mengubah route satu per satu.

Gambar: Route dengan prefix '/admin'

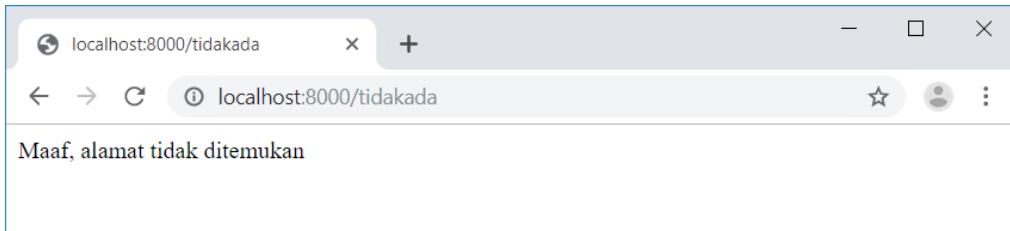
5.9. Route Fallback

Laravel menyediakan route khusus yang akan dijalankan jika tidak ditemukan route untuk sebuah alamat URL. Secara default, laravel akan menampilkan halaman 404 | Not Found, namun kita bisa menimpanya menggunakan method `Route::fallback()`.

Berikut contoh penulisannya:

`routes/web.php`

```
1 Route::fallback(function () {
2     return "Maaf, alamat tidak ditemukan";
3 });
```



Gambar: Route fallback akan dijalankan untuk alamat URL yang tidak memiliki route

Menggunakan view, nantinya kita bisa membuat tampilan halaman 404 yang lebih menarik.

5.10. Route Priority

Urutan penulisan route memiliki efek prioritas yang berbeda tergantung jenis route.

Jika kita menggunakan route biasa (tanpa parameter), maka apabila terdapat lebih dari 1 route dengan alamat yang sama, route paling akhir yang akan dijalankan:

```
1 Route::get('/buku/1', function () {
2     return "Buku ke-1";
3 });
4
5 Route::get('/buku/1', function () {
6     return "Buku saya ke-1";
7 });
8
9 Route::get('/buku/1', function () {
10    return "Buku kita ke-1";
11 });
12
```

Di sini saya menulis 3 buah route yang semuanya berisi alamat URL yang sama, yakni '/buku/1'. Ketika alamat `http://localhost:8000/buku/1` diakses, yang akan tampil di web browser adalah "Buku kita ke-1", yakni hasil dari route di baris 9 – 11.

Namun ketika menggunakan route parameter, hasilnya jadi berbeda:

Route

```
1 Route::get('/buku/{a}', function ($a) {
2     return "Buku ke-$a";
3 });
4
5 Route::get('/buku/{b}', function ($b) {
6     return "Buku saya ke-$b";
7 });
8
9 Route::get('/buku/{c}', function ($c) {
10    return "Buku kita ke-$c";
11});
```

Ketika mengakses `http://localhost:8000/buku/1`, yang tampil adalah "Buku ke-1", yakni hasil dari route di baris 1 – 3.

Urutan prioritas route ini perlu menjadi perhatian, karena bisa jadi kita bingung kenapa hasil tampilan tidak sesuai dengan yang tertulis di route. Bisa jadi itu karena alamat URL sudah "ditangkap" oleh route di baris sebelumnya.

5.11. Penulisan URL Route

Dalam beberapa referensi, penulisan route juga bisa diawali tanpa forward slash, seperti berikut:

```
1 Route::get('mahasiswa/andi', function () {
2     echo "Halaman mahasiswa andi";
3 });
```

Penulisan ini tidak ada bedanya dengan:

```
1 Route::get('/mahasiswa/andi', function () {
2     echo "Halaman mahasiswa andi";
3 });
```

Kedua route sama-sama merujuk ke alamat URL `localhost:8000/mahasiswa/andi`. Tambahan tanda '/' di awal penulisan route hanya "kesukaan saja". Penjelasan lebih lanjut bisa ke: [Routes in Laravel with or without a forward slash?](#)

Saya pribadi lebih suka menggunakan tambahan tanda '/' di awal agar seragam dengan penulisan alamat home yang juga ditulis sebagai '/'.

5.12. Melihat Daftar Route

Ketika project yang kita buat sudah cukup besar, file `routes/web.php` akan berisi banyak route. Selain itu nantinya juga terdapat beberapa perintah yang secara tidak langsung menjalankan route tertentu (akan kita bahas pada bagian terpisah).

Maka Laravel menyediakan perintah `php artisan` untuk melihat semua route yang tersedia.

Route

Artisan sendiri adalah perintah **cmd** bawaan Laravel yang bisa dipakai untuk melakukan banyak hal. Sepanjang buku ini kita akan sering memakai perintah `php artisan` terutama jika sudah masuk ke materi tentang controller dan model.

Sebelum itu, mari seragamkan daftar route. Berikut isi kode program dari file `routes/web.php` yang saya pakai:

`routes/web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 /*
6 |-----|
7 | Web Routes
8 |-----|
9 |
10| Here is where you can register web routes for your application. These
11| routes are loaded by the RouteServiceProvider within a group which
12| contains the "web" middleware group. Now create something great!
13|
14*/
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20 Route::get('hello', function () {
21     return 'Hello World';
22 });
23
24 Route::get('belajar', function () {
25     echo '<h1>Hello World</h1>';
26     echo '<p>Sedang belajar Laravel</p>';
27 });
28
29 Route::get('mahasiswa/fasilkom/anto', function () {
30     return '<h2 style="text-align: center"><u>Welcome Anto</u></h2>';
31 });
32
33 Route::get('mahasiswa/{nama}', function ($nama) {
34     return "Tampilkan data mahasiswa bernama $nama";
35 });
```

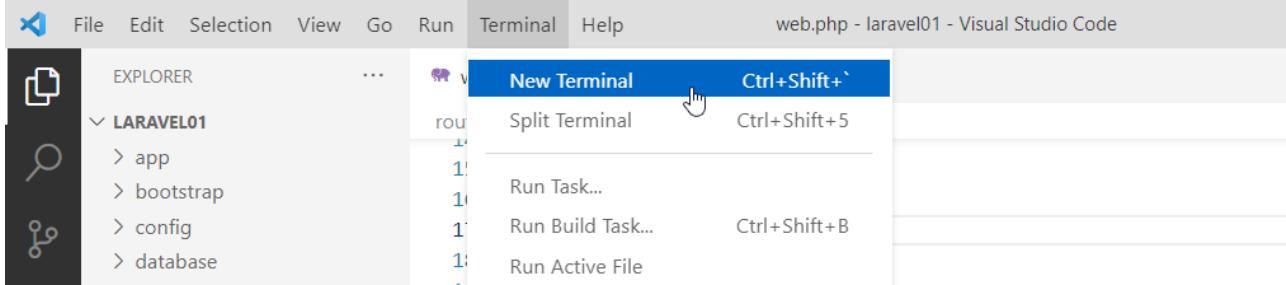
Ini merupakan sebagian route yang sudah kita buat sejak awal bab. Di sini saya memiliki 5 route, namun tidak masalah jika anda memiliki route lain.

Sekarang kita akan coba tampilkan daftar route menggunakan perintah **artisan**. Silahkan buka cmd baru (cmd yang menjalankan `php artisan serve` dibiarkan saja). Lalu masuk ke dalam folder `laravel01`, kemudian ketik perintah `php artisan route:list`

Gambar: Melihat daftar route

Route

Laravel 9 mengupdate tampilan daftar list dengan warna biru tua. Di cmd bawaan Windows, warna ini bentrok dengan background hitam sehingga agak susah di baca. Alternatif lain, bisa buka terminal di VS code (pilih menu Terminal -> New Terminal):



Lalu jalankan perintah `php artisan route:list` dari jendela terminal di sisi bawah:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\xampp\htdocs\laravel01> php artisan route:list

GET|HEAD / .....
POST _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user .....
GET|HEAD belajar .....
GET|HEAD hello .....
GET|HEAD mahasiswa/fasilkom/anto .....
GET|HEAD mahasiswa/{nama} .....
GET|HEAD sanctum/csrf-cookie .....
Laravel\Sanctum > CsrfCookieController@show

PS C:\xampp\htdocs\laravel01>
```

Gambar: Tampilan terminal di VS Code

Inilah daftar route aktif yang ada di aplikasi Laravel saat ini. Terdapat 10 baris yang mewakili 10 route. Kita memang hanya menulis 5 route di dalam file web.php (baris 1, 6, 7, 8 dan 9), tambahan 5 route lain berasal dari route bawaan Laravel.

Setiap baris terdiri dari 3 bagian:

- ◆ Sisi paling kiri berisi **route method** yang dalam tampilan di atas berisi GET|HEAD dan POST. Ini tidak lain merujuk ke HTTP method yang dikirim oleh web server. Nantinya kita bisa menambah method lain seperti PUT atau DELETE yang akan dibahas dalam bab terpisah.
- ◆ Sisi tengah berisi alamat **URL** yang perlu di tulis di *address bar* web browser.
- ◆ Sisi kanan berisi nama **controller** yang yang memproses route tersebut. Meskipun idealnya route merujuk ke controller, tapi ini tidak harus. Semua route yang kita tulis sejak awal bab tidak menggunakan controller sama sekali. Materi controller nantinya

juga akan dibahas dalam bab tersendiri.

Daftar route ini menjadi sangat berguna seiring kompleksitas web yang dibangun. Apalagi nantinya terdapat peningkatan penulisan route, seperti yang akan kita bahas dalam bab CRUD di pertengahan buku nanti.

Dalam bab ini kita telah membahas tentang **route**, yang tidak lain merupakan "pintu masuk" ke aplikasi Laravel.

Sebenarnya masih banyak materi lanjutan tentang route, namun materi tersebut belum bisa saya bahas karena butuh pemahaman tentang konsep lain. Kita akan pelajari secara bertahap ketika waktunya pas, misalnya cara mengakses **View** dari route akan dibahas di dalam bab tentang **View**, serta cara mengakses **Controller** dari route akan dibahas di dalam bab tentang **Controller**.

Berikutnya, kita akan belajar tentang menampilkan pesan error di Laravel serta fungsi bawaan Laravel untuk proses pencarian kesalahan (*debugging*).

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

=====

6. Error Display & Proses Debugging

Dalam menulis kode program, **error** adalah hal yang sangat dimaklumi. Entah itu lupa mengetik titik koma di akhir baris, salah penulisan nama fungsi, salah menulis nama file, salah tipe data, dsb.

Bisa membaca dan memahami pesan error juga salah satu skill programming yang sangat penting. Proses mencari kesalahan ini biasa disebut sebagai **debugging**. Skill ini akan meningkat seiring banyaknya latihan kode program, karena biasanya pesan error yang keluar "hanya itu-itu" saja.

Namun jika sudah menggunakan framework seperti Laravel, pesan error yang tampil memang lebih susah dipahami, karena bisa saja PHP menampilkan pesan error yang merujuk ke file lain (bukan file yang saat ini kita ketik).

Tips yang pastinya sudah sering kita lakukan adalah copy paste pesan error tersebut ke Google. Syukur-syukur ketemu artikel yang membahas persis apa yang kita alami. Website forum tanya jawab seperti stackoverflow.com bisa menjadi tumpuan yang sangat diharapkan.

Salah satu keuntungan menggunakan framework populer seperti Laravel adalah, cukup banyak programmer yang paham sehingga relatif lebih mudah mencari bantuan. Terutama jika dibandingkan framework lain yang masih relatif baru.

6.1. Tampilan Error Laravel

Apabila pada saat menulis kode program terdapat error, Laravel menampilkan error tersebut dengan tampilan yang cukup cantik, tidak sekedar text biasa seperti error di PHP Native.

Tampilan error pada Laravel 9 menggunakan library [Ignition](#), sebuah update dari versi Laravel sebelumnya. Mari kita lihat tampilan error ini.

Pertama, saya akan buat sebuah route sederhana:

```
routes/web.php
1 Route::get('/hello', function () {
2     return 'Hello World';
3 });
```

Route ini bisa diakses dari alamat `http://localhost:8000/hello`, yang akan menampilkan teks 'Hello World'. Tidak ada masalah karena ini sudah kita bahas pada bab sebelumnya.

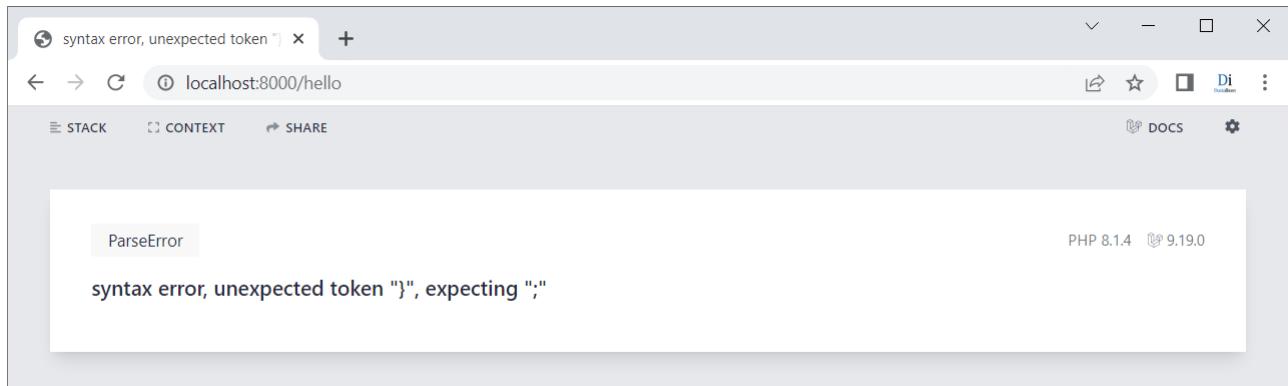
Error Display & Proses Debugging

Sekarang saya akan hapus tanda titik koma di akhir baris `return 'Hello World'`:

`routes/web.php`

```
1 Route::get('/hello', function () {
2     return 'Hello World'
3 });
```

Save, dan buka di web browser. Berikut tampilan halaman `http://localhost:8000/hello`:



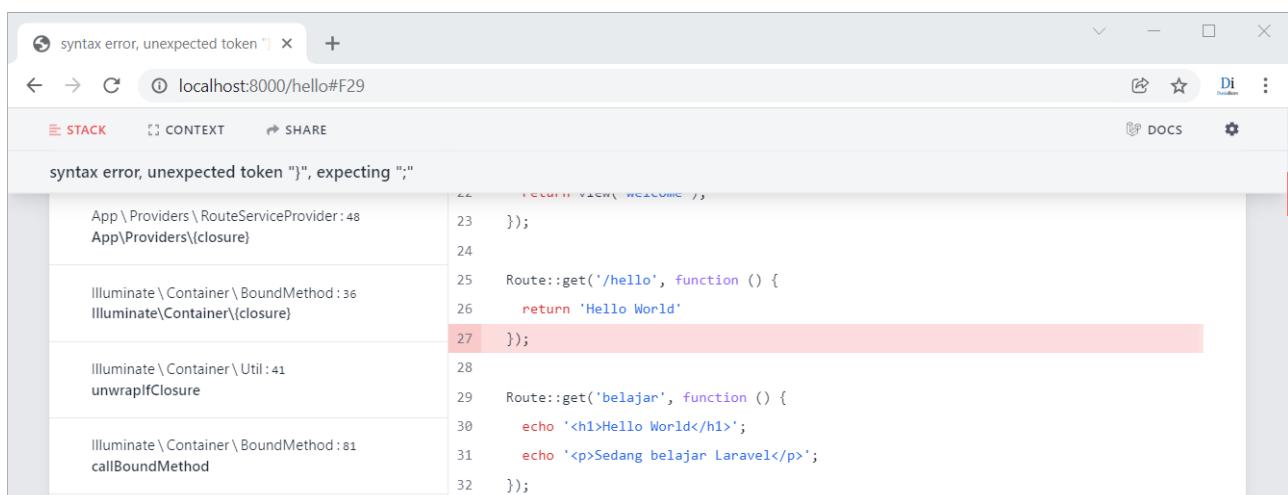
Gambar: Tampilan error Laravel

Inilah *error reporting* bawaan Laravel yang terdiri dari beberapa bagian.

Di sisi atas (1) terdapat teks kecil '*ParseError*' yang berarti PHP gagal membaca kode program. Biasanya ini disebabkan salah tulis perintah.

Penjelasan error yang lebih detail ada di bagian tengah (2), yakni *syntax error, unexpected token "}"*, *expecting ";"*, dimana PHP komplain karena terdapat tanda penutup kurung kurawal '}', seharusnya PHP mengharapkan tanda titik koma ';'. Ini merupakan pesan error classic yang sering tampil ketika kita lupa menulis tanda titik koma di akhir baris.

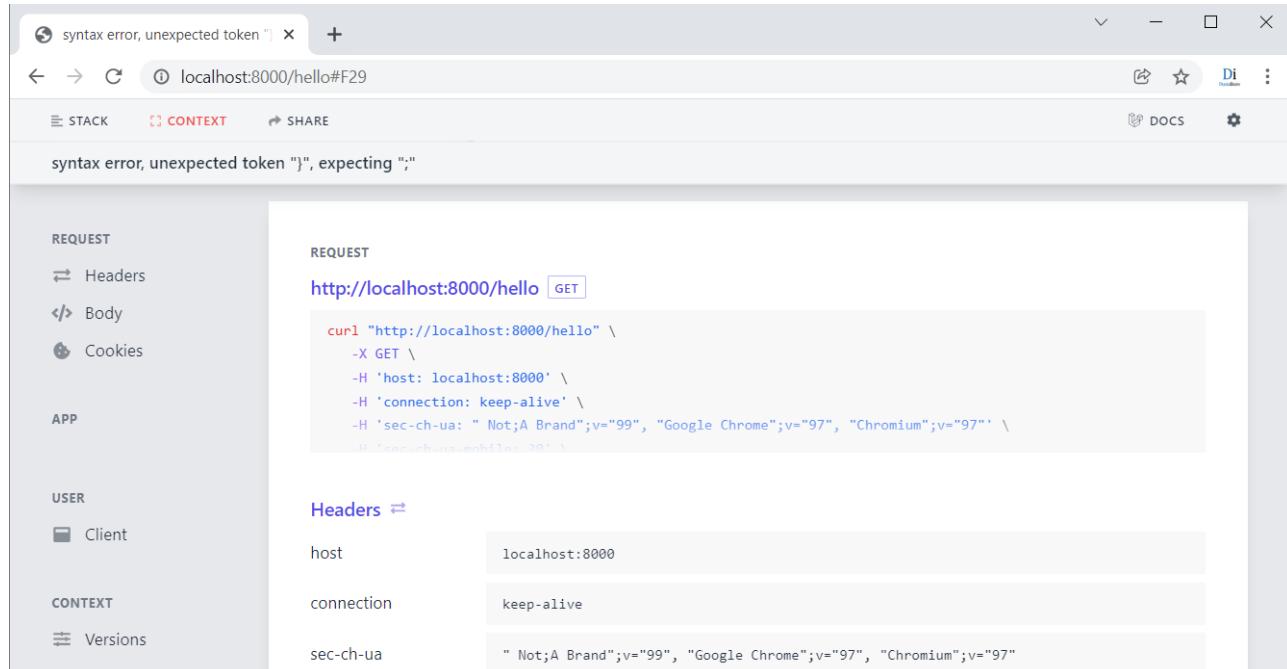
Scroll lagi sedikit ke bawah, akan terlihat cuplikan kode yang menjadi sumber error, yakni baris 27 di file `C:\xampp\htdocs\laravel01\routes\web.php`.



Gambar: Posisi error yang di deteksi Laravel

Error Display & Proses Debugging

Jika di scroll lagi ke sisi paling bawah, terlihat info tambahan seperti HTTP header, HTTP body, cookies, serta berbagai informasi lain. Bagian ini bisa dimanfaatkan untuk menggali info lebih jauh terkait error yang terjadi. Namun ini semua baru bisa kita pahami setelah masuk ke materi yang berhubungan seperti form processing atau cookie.



Gambar: Info tambahan tentang error

Jika anda menggunakan teks editor yang cukup modern seperti VS Code, kadang web editor sudah bisa mendeteksi salah ketik sederhana seperti lupa tanda titik koma, lupa menutup kurung atau lupa menutup tanda kutip.

Tampilan warna yang berbeda merupakan salah satu indikator kita salah ketik, meskipun ini tidak selalu pas karena ada kemungkinan teks editor belum bisa mendeteksi fitur terbaru yang ada di sebuah bahasa pemrograman.



Gambar: Tampilan error di VS Code

Teks editor lain mungkin punya fitur yang sama atau harus dikonfigurasi terlebih dahulu.

Sekarang, mari lakukan percobaan kedua. Silahkan tulis kembali tanda titik koma setelah baris

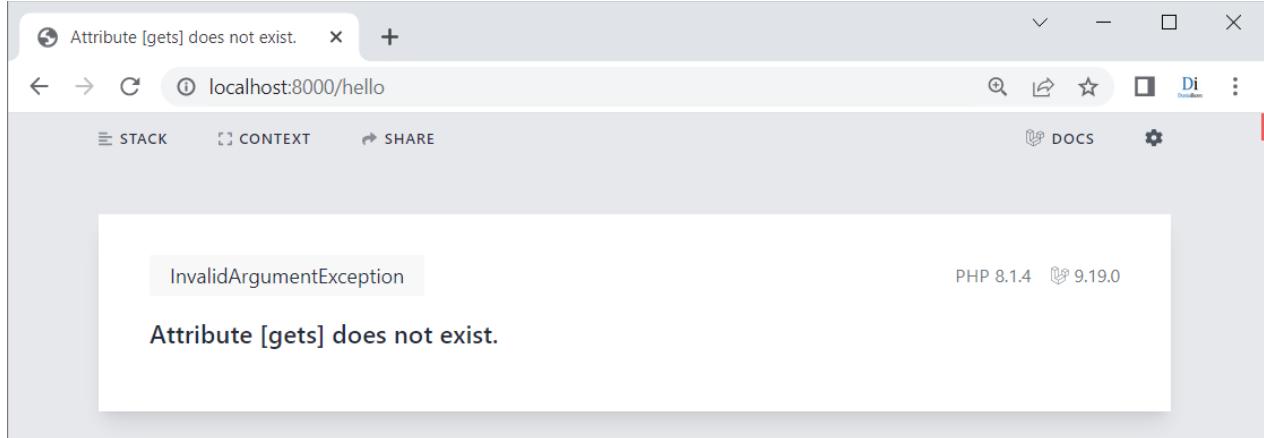
Error Display & Proses Debugging

return 'Hello World';, namun tambah 1 huruf **s** ke pemanggilan method `Route::get()`, menjadi `Route::gets()`:

`routes/web.php`

```
1 Route::gets('/hello', function () {
2     return 'Hello World';
3 });
```

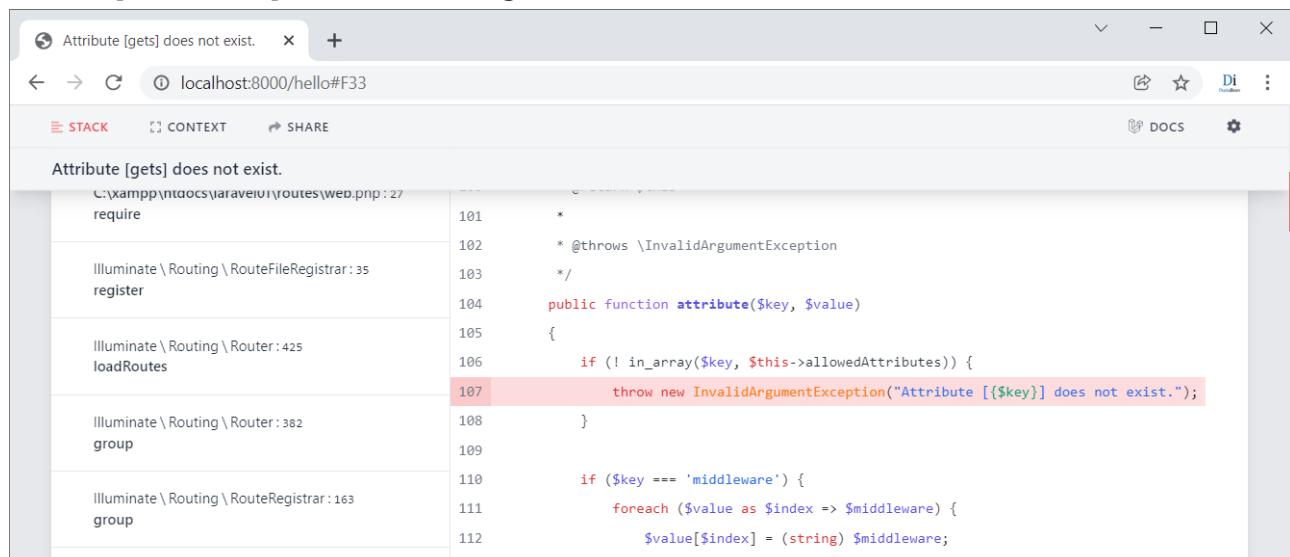
Dan berikut tampilan dari `http://localhost:8000/hello`:



Gambar: Tampilan error Laravel

Hasilnya kembali error karena di dalam Laravel memang tidak ada method `Route::gets()`. Pesan error yang tampil adalah "Attribute [gets] does not exist". Pesan ini cukup mudah dipahami dimana PHP komplain karena tidak menemukan atribut bernama gets.

Namun perhatikan posisi error di bagian bawah:



Gambar: Posisi error yang di deteksi Laravel

Kode yang tampil bukanlah isi dari file `routes/web.php` tempat kita sengaja menulis `Route::gets()`, tapi baris 107 dari file internal bawaan Laravel di `C:\xampp\htdocs\laravel01\`

vendor\laravel\framework\src\Illuminate\Routing\RouteRegistrar.php. Huff, path file yang cukup panjang. Di dalam file inilah Laravel melakukan proses pencarian method `Route::gets()`.

Hasil percobaan error kedua memperlihatkan bahwa bisa saja Laravel menampilkan detail error yang kurang pas, padahal sumber masalah kita hanya di penambahan 1 huruf 's' saat penulisan route.

Trivia

Jika anda perhatikan, struktur folder internal Laravel ada yang bernama "**Illuminate**". Bagi yang suka membaca teori konspirasi, bisa jadi langsung menghubungkannya dengan kelompok illuminati, *new world order*, *freemason*, dsb.

Saya tidak akan membahas hal ini lebih jauh karena itu lebih ke arah "cocoklogi". Tapi **Illuminate** merupakan *codename* dari Laravel 4, dimana tim Laravel saat itu merombak ulang inti framework dan masih menggunakan nama tersebut hingga sekarang.

Salah satu artikel di [quora](#) menyatakan bahwa **Taylor Otwell**, pembuat Laravel pernah berkata: "Nama Illuminate berarti menerangi, yakni menerangi pembuatan aplikasi dengan PHP dan membuatnya lebih baik" ("Illuminate means to light up, like to light up PHP development and make it better").

6.2. File Konfigurasi Laravel

Tampilan pesan error sangat membantu kita pada saat pembuatan kode program. Akan tetapi ketika project sudah selesai, pesan error ini harus disembunyikan. Selain bisa membuat bingung pengguna aplikasi (user), pesan error juga bisa menjadi sumber berharga bagi pihak yang ingin membobol aplikasi kita.

Untungnya, menyembunyikan pesan error di Laravel sangat mudah. Kita hanya perlu mengatur 1 baris perintah di file konfigurasi Laravel.

Laravel memiliki 2 tempat mengatur file konfigurasi, yakni di file `.env` dan di folder `config`. File `.env` berisi pengaturan ringkas, sedangkan konfigurasi utama ada di folder `config`. Kenapa dirancang seperti ini?

Alasannya agar memudahkan pembuatan aplikasi ketika dilakukan bersama tim. Jika sebuah project dikerjakan lebih dari 1 orang, maka ada kemungkinan *environment* atau lingkungan kerja setiap anggota tim berbeda-beda.

Misalnya seorang *web designer* lebih suka menggunakan MacOS, sedangkan *back-end developer* lebih suka menggunakan Linux. Kedua sistem operasi ini butuh konfigurasi yang berbeda agar bisa menjalankan Laravel. Konfigurasi tersebut mencakup tampilan error, alamat

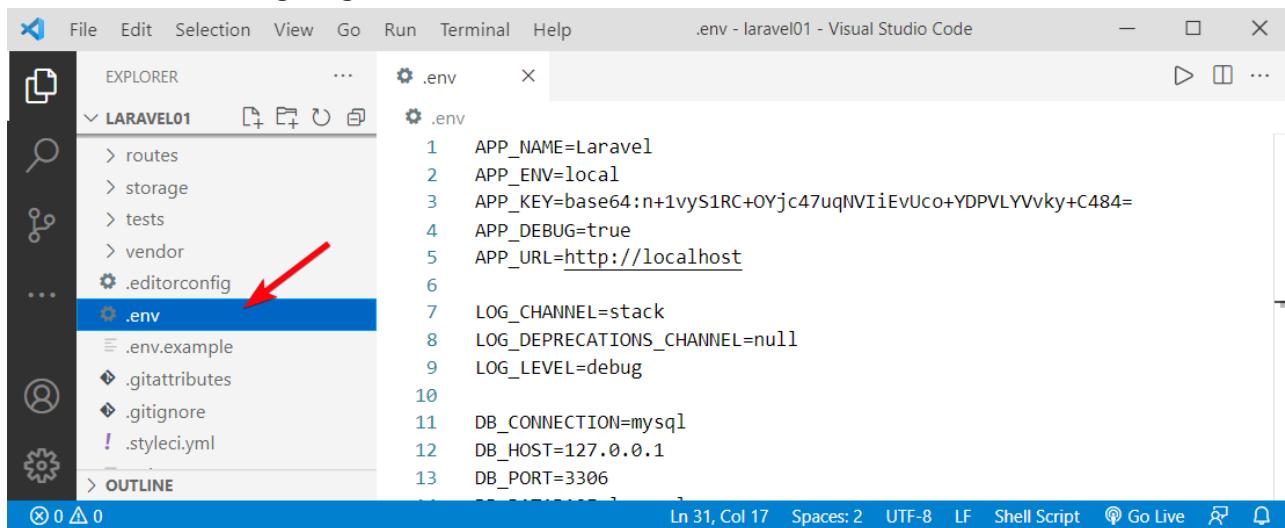
host, alamat server database, password database, dsb.

Pengaturan di folder **config** dirancang sebagai *master*, yakni pengaturan default dari sebuah project. Pada saat satu anggota tim mengambil file project asli, bisa jadi ada pengaturan yang tidak sesuai dengan sistem operasi yang dipakainya. Supaya tidak mengganggu pengaturan default, maka pengaturan individu dari setiap anggota tim bisa disimpan ke dalam file `.env`.

Dengan demikian, isi file konfigurasi `.env` bisa berbeda-beda antar satu anggota tim dengan anggota lainnya.

Pengaturan File `.env`

File `.env` berada langsung di bawah folder utama Laravel. Mari kita buka:



The screenshot shows the Visual Studio Code interface with the title bar ".env - laravel01 - Visual Studio Code". The left sidebar shows a tree view of the "LARAVEL01" folder containing "routes", "storage", "tests", "vendor", ".editorconfig", ".env" (which is selected and highlighted with a red arrow), ".env.example", ".gitattributes", ".gitignore", and ".styleci.yml". The main editor area displays the contents of the ".env" file:

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:n+1vyS1RC+OYjc47uqNVIiEvUco+YDPVLYVvky+C484=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
```

At the bottom, status bar text includes "Ln 31, Col 17 Spaces: 2 UTF-8 LF Shell Script".

Gambar: Isi dari file `.env`

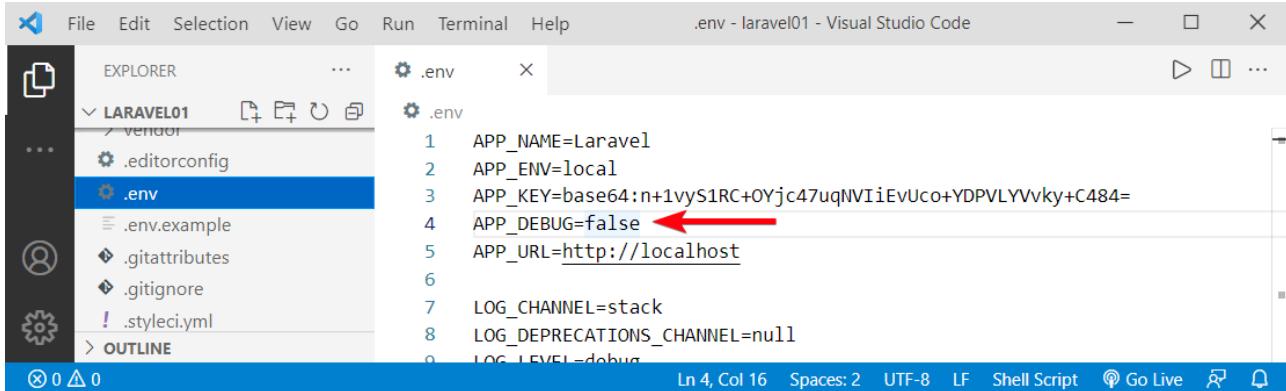
Penjelasan lengkap tentang setiap pengaturan ini bisa dibaca dari folder **config** (akan kita buka sesaat lagi). Selain itu beberapa pengaturan akan dibahas dalam bagian terpisah, misalnya ketika sudah masuk ke materi database.

File `.env` berisi sekitar 46 baris pengaturan dalam format berikut:

```
NAMA_PENGATURAN = nilai
```

Khusus untuk pengaturan tampilan error ada di baris ke-4, yakni `APP_DEBUG=true`. Jika pengaturan `APP_DEBUG` diisi nilai `true`, maka Laravel akan menampilkan semua pesan error (pesan debug). Agar pesan error tidak tampil, ubah pengaturan ini menjadi `APP_DEBUG=false` lalu save file `.env`.

Error Display & Proses Debugging

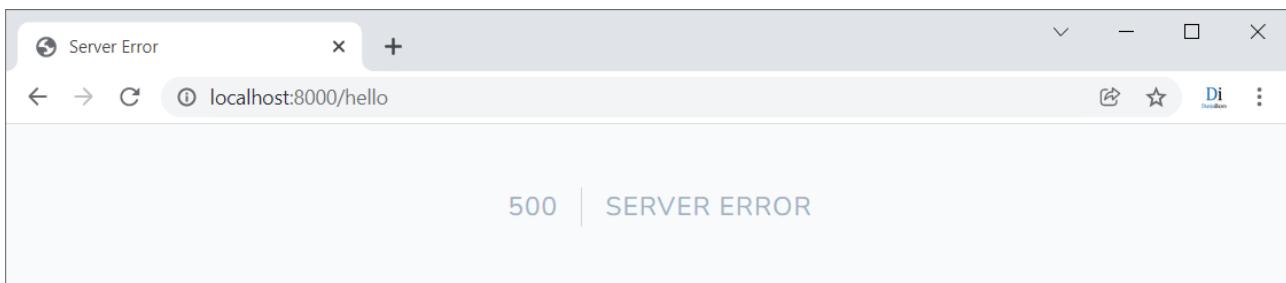


The screenshot shows the Visual Studio Code interface with the '.env' file open. The file contains environment variables for a Laravel application. A red arrow points to the line 'APP_DEBUG=false'. The file also includes other variables like APP_NAME=Laravel, APP_ENV=local, APP_KEY, APP_URL=http://localhost, LOG_CHANNEL=stack, and LOG_DEPRECATIONS_CHANNEL=null.

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:n+1vyS1RC+OYjc47uqNVIiEvUco+YDPVLYVvky+C484=
4 APP_DEBUG=false
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATED_CHANNEL=null
9 LOG_LEVEL=debug
```

Gambar: Tukar pengaturan APP_DEBUG=false

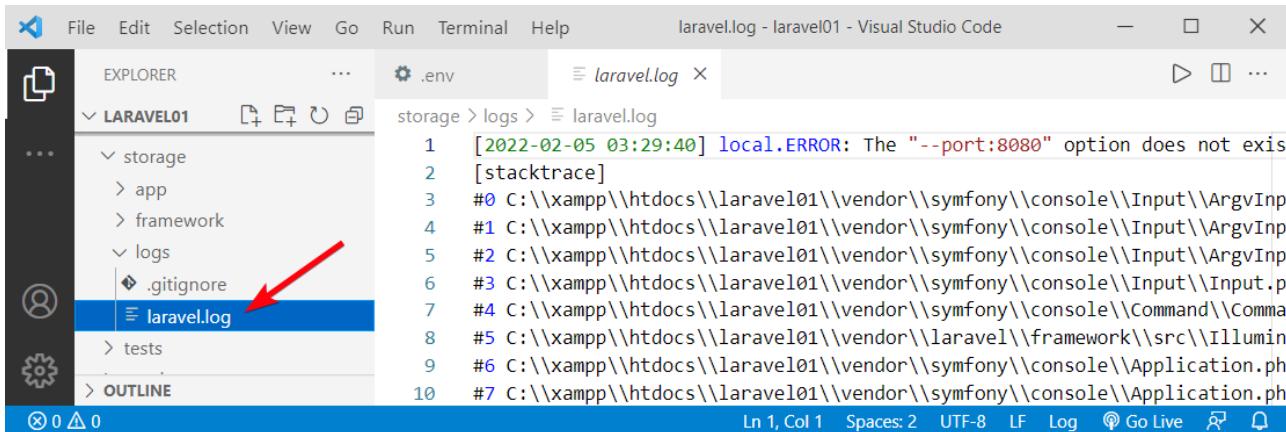
Setelah itu akses kembali <http://localhost:8000/hello> di web browser:



Gambar: Pesan error 'generik' Laravel

Sekarang yang tampil hanya pesan error 'generik' 500 | Server Error. Tampilan error seperti ini lebih manusiawi bagi pengguna aplikasi kita. Apapun error yang terjadi, yang tampil tetap berupa teks 500 | Server Error.

Namun masalahnya, dari mana kita bisa tau apa yang menyebabkan error? Laravel menyimpan history error ke dalam file laravel.log yang ada di folder storage\logs\.



The screenshot shows the Visual Studio Code interface with the 'laravel.log' file open. The file is located in the storage/logs directory. A red arrow points to the 'laravel.log' file in the Explorer sidebar. The log file contains several lines of error messages, starting with '[2022-02-05 03:29:40] local.ERROR: The "--port:8080" option does not exist'.

```
1 [2022-02-05 03:29:40] local.ERROR: The "--port:8080" option does not exist
2 [stacktrace]
3 #0 C:\xampp\htdocs\laravel01\vendor\symfony\console\Input\ArgvInput::handleUnrecognizedOption()
4 #1 C:\xampp\htdocs\laravel01\vendor\symfony\console\Input\ArgvInput::handleUnknownOption()
5 #2 C:\xampp\htdocs\laravel01\vendor\symfony\console\Input\ArgvInput::handleUnknownOption()
6 #3 C:\xampp\htdocs\laravel01\vendor\symfony\console\Input\ArgvInput::handleUnknownOption()
7 #4 C:\xampp\htdocs\laravel01\vendor\symfony\console\Command\Command::execute()
8 #5 C:\xampp\htdocs\laravel01\vendor\laravel\framework\src\Illuminate\Console\Command::execute()
9 #6 C:\xampp\htdocs\laravel01\vendor\symfony\console\Application.php:32
10 #7 C:\xampp\htdocs\laravel01\vendor\symfony\console\Application.php:32
```

Gambar: Lokasi file log Laravel

File log hanya berupa file teks biasa, namun isi teks memang sedikit teknis karena Laravel tidak hanya menyimpan pesan error, tapi juga seluruh daftar file yang diduga menjadi sumber masalah.

Error Display & Proses Debugging

Dalam contoh sebelumnya, saya sengaja mengubah method `Route::get` menjadi `Route::gets`. Pesan error ini bisa terbaca di dalam file log lengkap dengan waktu kapan error tersebut terjadi. Error paling akhir berada di baris paling bawah.

The screenshot shows the Visual Studio Code interface with the title bar "laravel.log - laravel01 - Visual Studio Code". The left sidebar shows a project structure under "LARAVEL01" with "storage", "logs", and ".env" files. The "logs" folder contains "laravel.log". The main editor area displays the contents of "laravel.log". A red arrow points to the error message at the bottom of the log:

```
#30 C:\xampp\htdocs\laravel01\public\index.php(52): Illuminate\Foundation\...
#31 C:\xampp\htdocs\laravel01\vendor\laravel\framework\src\Illuminate\...
esources\server.php(16): require_once('C:\xampp\htdocs...')
#32 {main}
"
[2022-02-05 05:33:25] local.ERROR: Attribute [gets] does not exist. {"exception":...
[stacktrace]
#0 C:\xampp\htdocs\laravel01\vendor\laravel\framework\src\Illuminate\...
#1 C:\xampp\htdocs\laravel01\vendor\laravel\framework\src\Illuminate\...
#2 C:\xampp\htdocs\laravel01\
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF Log ⚡ Go Live ⌂

Gambar: Isi dari file log, error karena method Route::gets tidak ditemukan

Folder log ini harus selalu di kontrol terutama jika aplikasi sudah di pakai user. Di sinilah kita sebagai developer bisa melihat kronologis error yang terjadi. Jika user tiba-tiba komplain aplikasinya bermasalah, folder log inilah yang harus diperiksa pertama kali.

Untuk Laravel 7 ke bawah, setiap kali file `.env` di edit, kita harus restart server dengan cara menutup jendela cmd yang menjalankan perintah `php artisan serve` atau tekan tombol **Ctrl + C** di cmd, lalu jalankan kembali perintah `php artisan serve`.

Mulai dari Laravel 8 ke atas, ini tidak lagi diperlukan. Perubahan di file `.env` secara otomatis me-restart server:

The terminal window shows the command "Start Laravel01 Server - php artisan serve". The output log shows several messages, with a red arrow pointing to the line "Environment modified. Restarting server...".

```
[Sat Feb 5 12:23:56 2022] 127.0.0.1:49628 Closing
[Sat Feb 5 12:25:05 2022] 127.0.0.1:49629 Closed without sending a request; it was probably just an unused
speculative preconnection
[Sat Feb 5 12:25:05 2022] 127.0.0.1:49629 Closing
Environment modified. Restarting server...
[Sat Feb 5 12:32:39 2022] PHP 8.1.1 Development Server (http://127.0.0.1:8000) started
[Sat Feb 5 12:33:25 2022] 127.0.0.1:49705 Accepted
[Sat Feb 5 12:33:25 2022] 127.0.0.1:49706 Accepted
```

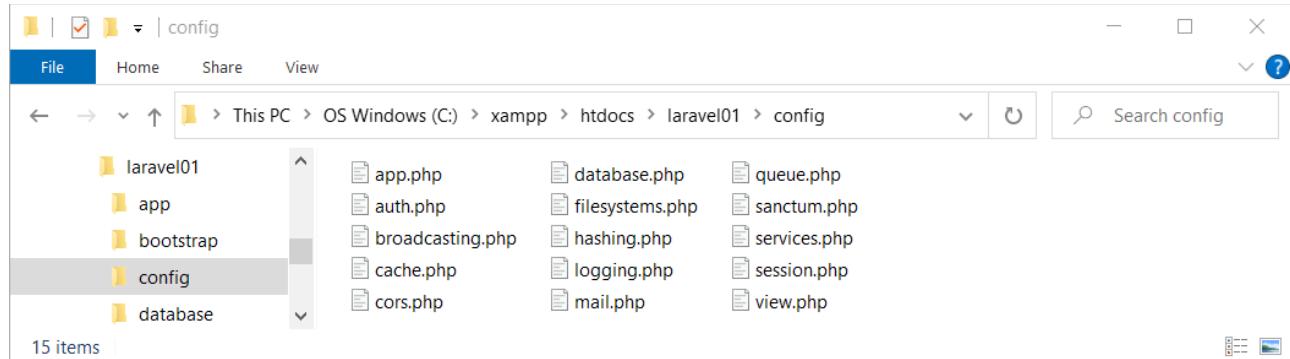
Gambar: Laravel 9 langsung restart server jika file .env di modifikasi

Pengaturan dari Folder Config

Baik, kita sudah melihat cara mengubah konfigurasi dari file `.env`. Sekarang kita beralih ke folder **config**.

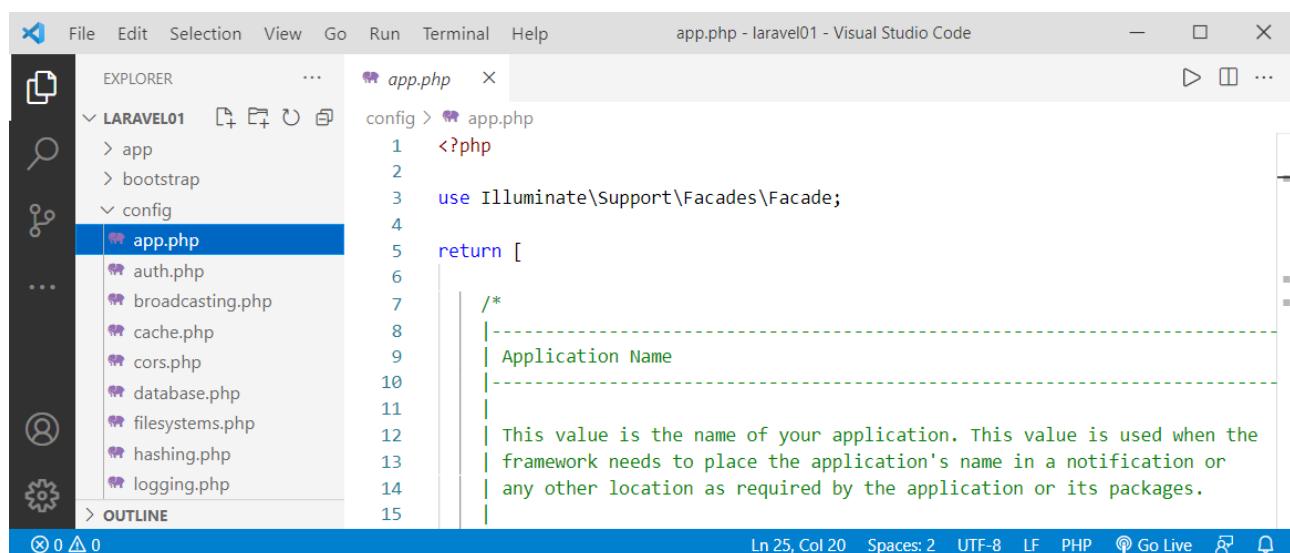
Kita sudah bahas sekilas bahwa folder **config** berisi pengaturan *master* dari aplikasi Laravel. Pengaturan ini dikelompokkan menurut fungsinya dan berada di file terpisah. Sebagai contoh, pengaturan tentang database ada di file `database.php`, pengaturan tentang email ada di `mail.php`, dst.

Error Display & Proses Debugging



Gambar: Isi folder config Laravel

Untuk mengatur tampilan error, ada di file `app.php`, silahkan buka file ini:



Gambar: Isi file app.php

Mayoritas isi file yang ada di folder **config** berupa teks komentar yang menjelaskan fungsi dari setiap konfigurasi. Selain itu penulisan pengaturan juga sedikit berbeda dengan file `.env`. Kali ini konfigurasi ditulis dalam format *associative array*.

Jika seluruh baris komentar yang terdapat pada file `app.php` dihapus, tampilannya menjadi sebagai berikut:

config/app.php

```
1 <?php
2
3 return [
4     'name' => env('APP_NAME', 'Laravel'),
5     'env' => env('APP_ENV', 'production'),
6     'debug' => (bool) env('APP_DEBUG', false),
7     'url' => env('APP_URL', 'http://localhost'),
8     'asset_url' => env('ASSET_URL', null),
9     'timezone' => 'UTC',
10    ...
11 ]
```

Beberapa pengaturan ada yang diambil dari fungsi `env()`. Sebagai contoh, perhatikan kode pengaturan berikut:

```
'debug' => (bool) env('APP_DEBUG', false),
```

Kode ini bisa dibaca: "untuk mengatur debug, ambil nilai `APP_DEBUG` yang ada di file `.env`. Namun jika di dalam file `.env` tidak ditemukan, maka isi dengan nilai `false`".

Di sini `env()` merupakan *function helper* bawaan Laravel. Fungsi ini dipakai untuk mengambil nilai pengaturan dari file `.env`.

Dengan penulisan seperti ini, Laravel akan memprioritaskan pengaturan dari file `.env` terlebih dahulu. Jika ada, nilai itulah yang dipakai. Jika tidak ada, maka gunakan nilai yang ditulis dalam argument kedua, yakni `false`.

Di sini juga bisa terlihat bahwa secara bawaan pengaturan tampilan error di file `app.php` sudah bernilai `false`, yang artinya pesan kesalahan tidak akan tampil. Namun karena di dalam file `.env` pengaturan ini bernilai `true` (secara bawaan), maka pesan error Laravel tetap tampil.

Sebelum lanjut ke bahasan berikutnya, silahkan aktifkan kembali pengaturan `debug` menjadi `true` di file `.env`. Karena sepanjang proses belajar dan pembuatan aplikasi, tentu kita ingin semua pesan error bisa langsung terlihat.

6.3. Pencarian Kesalahan (Debugging)

Tampilan pesan kesalahan memang sangat membantu, namun sering kali pesan error tersebut belum bisa memecahkan masalah yang hadapi.

Dalam **PHP native** (sebutan untuk "PHP biasa" / PHP tanpa framework), kita sering menggunakan fungsi `var_dump()` atau `print_r()` untuk menampilkan struktur detail dari sebuah variabel, terutama variabel yang berisi data kompleks seperti array. Kedua fungsi ini juga masih bisa dipakai di Laravel. Berikut contoh penggunaannya:

```
routes/web.php

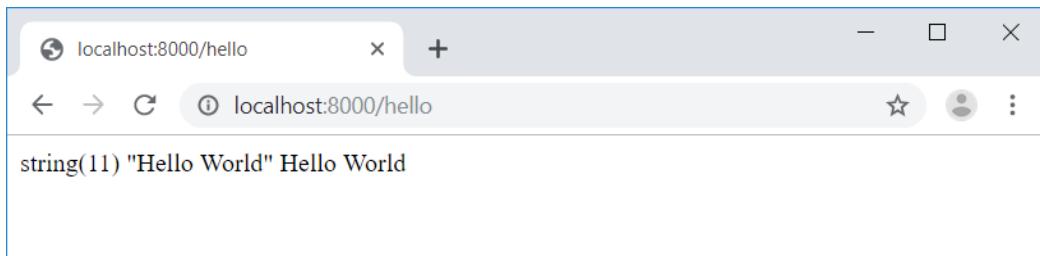
1 Route::get('/hello', function () {
2     $hello = 'Hello World';
3     var_dump($hello);
4
5     return $hello;
6 });
```

Di sini saya merancang sebuah route '`hello`'. Dalam fungsi *closure* di baris 2, saya membuat variabel `$hello` dan mengisinya dengan string '`Hello World`'.

Pada prakteknya nanti, nilai untuk variabel `$hello` ini bisa saja berasal dari sumber lain dan berisi tipe data yang lebih kompleks seperti array atau object.

Error Display & Proses Debugging

Untuk melihat detail isi variabel \$hello, saya menggunakan fungsi `var_dump($hello)` seperti yang terlihat di baris 3. Berikut hasil yang tampil di web browser:

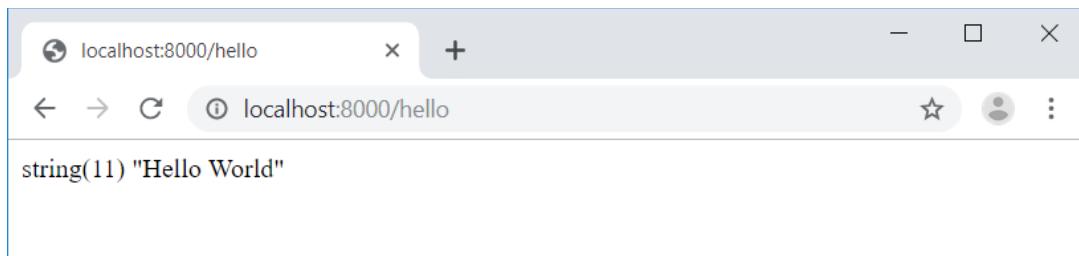


Gambar: Hasil tampilan `var_dump()`

Kenapa tampil 2 kali teks 'Hello World'? Alasannya karena selain hasil `var_dump()`, juga tampil hasil dari perintah `return $hello` di baris 5. Untuk menghindari hal ini, dan juga untuk menghentikan PHP tepat setelah perintah `var_dump()`, bisa di tambah fungsi `die()`:

routes/web.php

```
1 Route::get('/hello', function () {
2     $hello = 'Hello World';
3     var_dump($hello);
4     die();
5
6     return $hello;
7});
```



Gambar: Hasil tampilan `var_dump()` dan `die()`

Penggunaan fungsi `die()` cukup penting karena pada kode yang kompleks, bisa saja tampil error lain sebab Laravel terus lanjut memproses kode program setelah baris `var_dump()`.

Teknik seperti ini akan memudahkan kita mencari sumber error, dan karena alasan itu pula Laravel menyediakan fungsi khusus yang men-upgrade tampilan `var_dump()` bawaan PHP. Fungsi tersebut adalah `dd()`, singkatan dari "**d**ump and **d**ie".

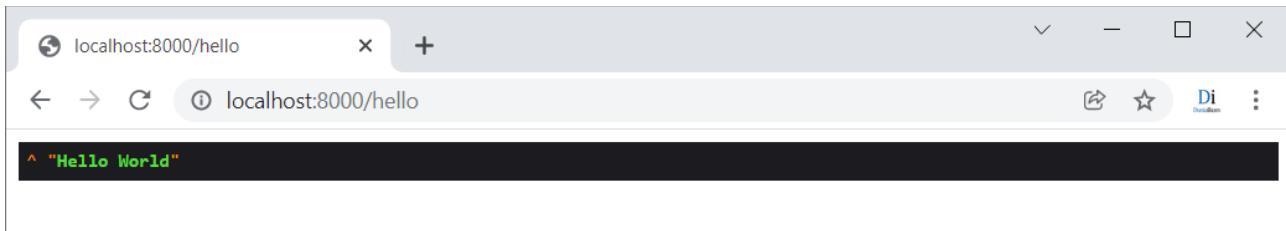
Sesuai dengan namanya, fungsi `dd()` akan menampilkan isi dari sebuah variabel dan langsung menghentikan kode program. Berikut contoh penggunaannya:

routes/web.php

```
1 Route::get('/hello', function () {
2     $hello = 'Hello World';
3     dd($hello);
```

Error Display & Proses Debugging

```
4
5     return $hello;
6 );
```



Gambar: Tampilan dari fungsi dd()

Perbedaan paling mencolok ada di hasil tampilan dd() yang lebih *colorfull* dengan warna teks hijau dan background hitam. Karena isi dari variabel \$hello hanya string sederhana, tampilan dd() masih kurang menarik. Mari kita coba dengan variabel yang lebih kompleks:

routes/web.php

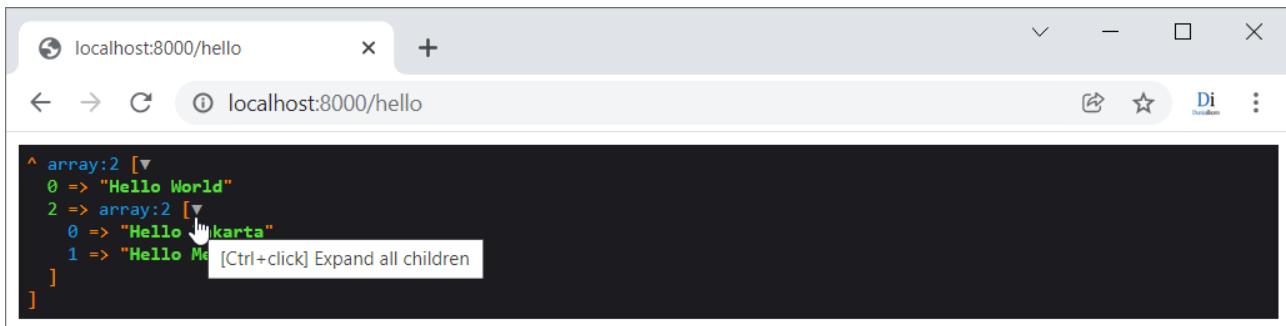
```
1 Route::get('hello', function () {
2     $hello = ['Hello World', 2 => ['Hello Jakarta', 'Hello Medan']];
3     dd($hello);
4
5     return $hello;
6 });
```

Di sini saya mengisi variabel \$hello dengan sebuah *nested array* atau array bersarang, dan berikut hasilnya:



Gambar: Tampilan dari fungsi dd() untuk nested array

Hasil tampilan fungsi dd() ini juga interaktif. Kita bisa klik tanda segitiga panah untuk membuka dan menutup isi array:



Gambar: Klik tanda panah untuk menampilkan seluruh element array

Error Display & Proses Debugging

Alternatif lain, Laravel juga menyediakan function `dump()`. Bedanya dengan `dd()` adalah, function `dump()` tidak langsung menghentikan kode program, hanya menampilkan isi variabel saja:

routes/web.php

```
1 Route::get('/hello', function () {
2     $hello = ['Hello World', 2 => ['Hello Jakarta', 'Hello Medan']];
3     dump($hello);
4
5     return $hello;
6 });
```



Gambar: Hasil penggunaan function `dump()`

Di web browser terlihat bahwa setelah proses tampilan dari `dump($hello)`, ada 1 baris teks yang berasal dari perintah `return $hello;`. Ini juga menarik karena jika di perhatikan, baris ini merupakan sebuah string dalam format **JSON** (JavaScript Object Notation).

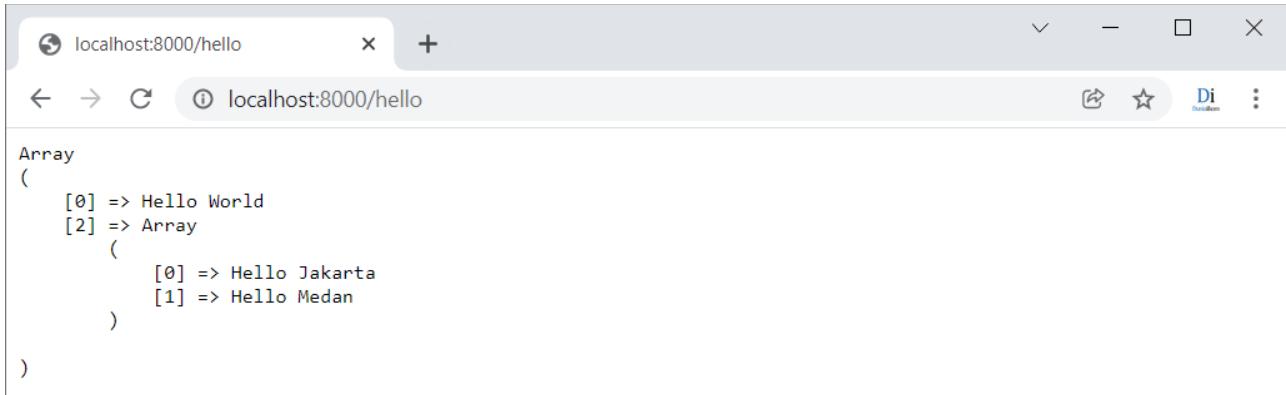
Artinya, jika sebuah array langsung di echo atau di return, Laravel otomatis menampilkan dalam format JSON. JSON sendiri sering dipakai sebagai sarana pertukaran data antar web atau untuk membuat **API**.

Alternatif terakhir, kita masih tetap bisa menggunakan fungsi `print_r()` bawaan PHP untuk menampilkan isi array:

routes/web.php

```
1 Route::get('/hello', function () {
2     $hello = ['Hello World', 2 => ['Hello Jakarta', 'Hello Medan']];
3
4     echo '<pre>';
5     print_r($hello);
6     echo '</pre>';
7     die();
8
9     return $hello;
10});
```

Error Display & Proses Debugging



Gambar: Tampilan print_r

Sepanjang pembuatan aplikasi nanti, function `dd()` dan `dump()` akan sangat membantu kita dalam menganalisis error yang terjadi.

Jika fungsi `dd()` bawaan Laravel juga tidak mencukupi, tersedia berbagai package atau plugin tambahan untuk proses debugging di Laravel, salah satunya [Laravel Debugbar](#). Namun package ini bukan untuk pemula karena butuh proses instalasi khusus.

6.4. Menghapus Tambahan Tanda Caret (^)

Di versi Laravel 9 yang saya pakai, ada sedikit bug (atau mungkin fitur?) yang ditambahkan tim Laravel. Ketika menampilkan hasil kode program dengan `dd()` dan `dump()`, terdapat tambahan tanda caret atau topi " ^ " di sisi kiri atas:



Gambar: Tambahan tanda caret di kiri atas hasil dd() dan dump()

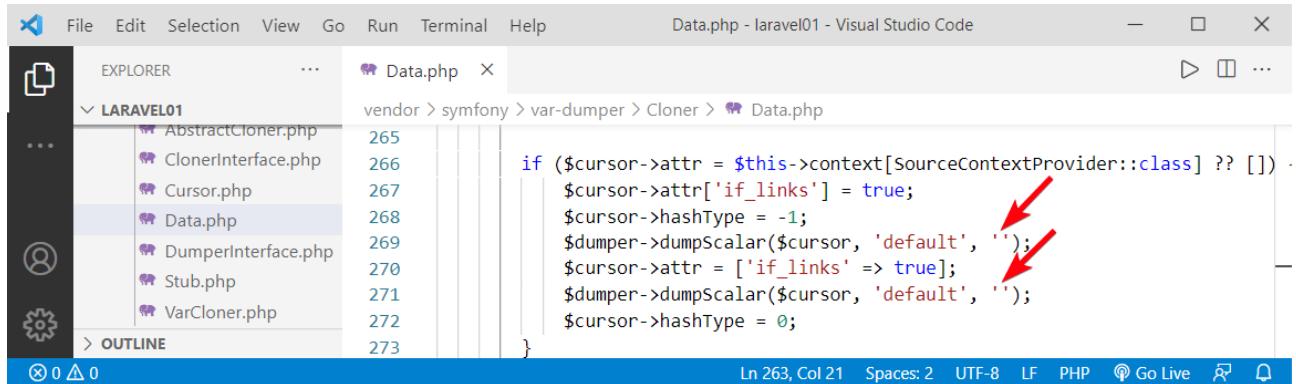
Ini sebenarnya tidak terlalu masalah, hanya sedikit mengganggu saja karena di versi Laravel sebelumnya karakter ini tidak terlihat. Setelah penelusuran lebih lanjut, karakter " ^ " ternyata berasal dari [library bawaan symfony](#), yakni `symfony VarDumper component`.

Untungnya, terdapat cara mudah untuk memperbaiki masalah ini. Silahkan buka file `\vendor\symfony\var-dumper\Cloner\Data.php`, lalu cari kode berikut di baris 269:

```
1 ...  
2     $dumper->dumpScalar($cursor, 'default', '^');  
3     $cursor->attr = ['if_links' => true];  
4     $dumper->dumpScalar($cursor, 'default', ' ');  
5 ...
```

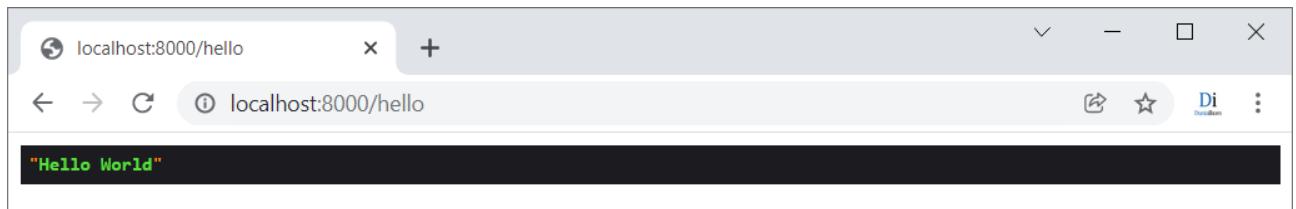
Lalu ganti menjadi:

```
1 ...
2     $dumper->dumpScalar($cursor, 'default', '');
3     $cursor->attr = ['if_links' => true];
4     $dumper->dumpScalar($cursor, 'default', '');
5 ...
```



Gambar: Hapus tambahan karakter "^" dan " " di file Data.php

Simpan perubahan di atas. Sekarang hasil dd() dan dump() tidak lagi diawali karakter " ^ ":



Gambar: Tampilan dd() dan dump() sudah kembali "bersih"

Saya kurang tau apakah ini memang bug atau fitur normal di Laravel 9. Jika ini sebuah bug, mudah-mudahan segera di perbaiki.

Dalam bab ini kita telah mempelajari tentang tampilan error Laravel, mengatur tampilan error, serta melihat fungsi yang sering dipakai untuk proses debugging.

Selanjutnya, kita masuk ke bagian pertama dari konsep MVC Laravel, yakni **View**.

7. View

Kita akan masuk ke bagian pertama dari konsep MVC, yakni **View**. Nantinya akan dipelajari tentang cara pembuatan view, mengirim data ke view, serta pemrosesan data di view.

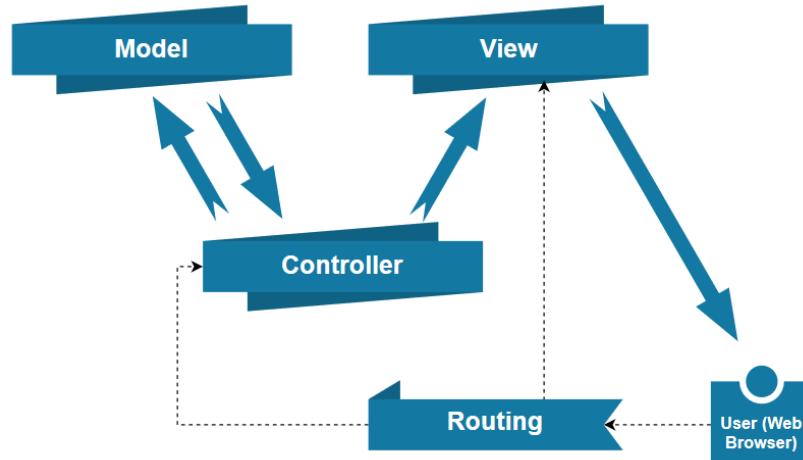
Jika anda membaca bab sebelum ini sekaligus mempretekannya (yang memang sangat disarankan), maka file `routes\web.php` sudah berisi beberapa route hasil praktek kita.

Agar tidak bermasalah, silahkan hapus semua route tersebut dan sisakan 1 route bawaan Laravel. Atau juga bisa mulai dari instalasi Laravel 9 yang baru.

7.1. Pengertian View

Pada awal buku kita telah membahas tentang teori dan juga sedikit praktek konsep **MVC**. Di sana di jelaskan bahwa **View** adalah komponen MVC yang menangani tampilan. Di dalam viewlah kode HTML, CSS dan juga JavaScript berada.

Sebelum kita masuk ke praktek pembuatan view, mari lihat kembali diagram alur MVC berikut:



Gambar: Diagram Alur MVC + Routing

Dalam gambar ini terlihat bahwa view sebenarnya diakses dari controller, namun Laravel mengizinkan kita untuk langsung mengakses view dari route. Jalur **Route->View** ini biasa dipakai untuk menampilkan halaman statis yang tidak perlu pemrosesan kompleks.

Jika nantinya kita butuh mengolah data yang rumit, maka jalur untuk menampilkan view harus ke controller terlebih dahulu. Mengenai hal ini akan di bahas dalam bab controller nanti, di

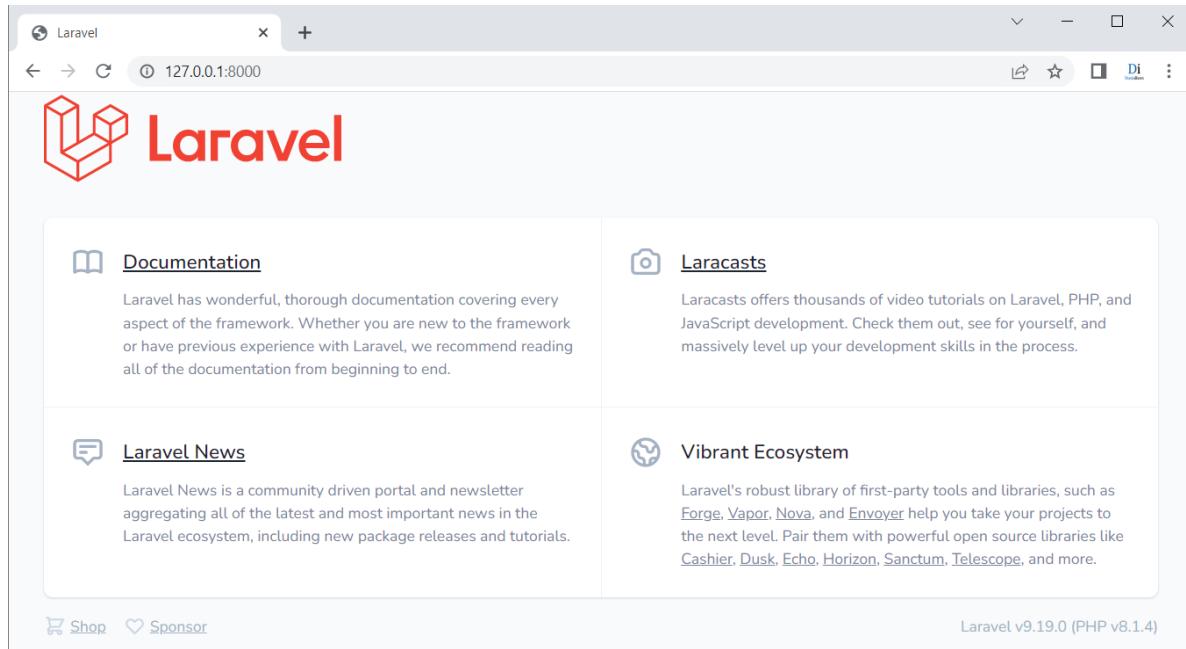
sini kita akan fokus mempelajari cara pengaksesan view langsung dari route saja.

7.2. View Bawaan Laravel

Laravel sebenarnya memiliki 1 view bawaan yang terlihat saat kita mengakses halaman homepage atau halaman root Laravel. Pemanggilan view ini berasal dari kode berikut:

routes/web.php

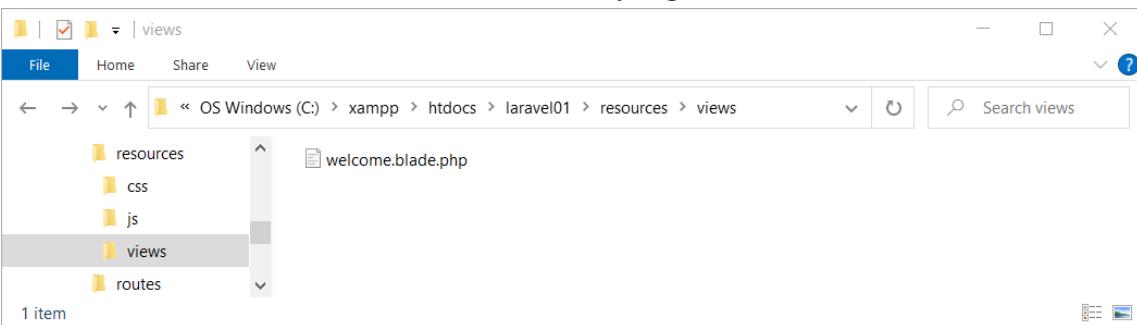
```
1 Route::get('/', function () {
2     return view('welcome');
3 });
```



Gambar: Tampilan view 'welcome' bawaan Laravel

Kode ini bisa dibaca: "Jika halaman root '/' diakses, tampilkan view bernama `welcome`".

Di dalam Laravel, nama sebuah view mewakili nama suatu file, artinya harus terdapat file bernama `'welcome'` di dalam folder instalasi Laravel. Di manakah letaknya? Laravel menyimpan view di dalam folder **resources\views**. Jika anda membuka folder ini, akan menemukan file bernama `welcome.blade.php`. Inilah view `'welcome'` yang dimaksud.



Gambar: Lokasi View 'welcome'

Dalam Laravel, setiap view harus ditulis dengan nama file berikut:

`<nama_file>.blade.php`

Sehingga jika kita ingin membuat view 'welcome', nama filenya adalah `welcome.blade.php`.

Blade adalah sebuah *templating engine* bawaan Laravel. Materi lebih lengkap tentang blade akan kita bahas dalam bab berikutnya. Untuk saat ini, cukup pahami bahwa semua file view Laravel harus diakhiri dengan `.blade.php`.

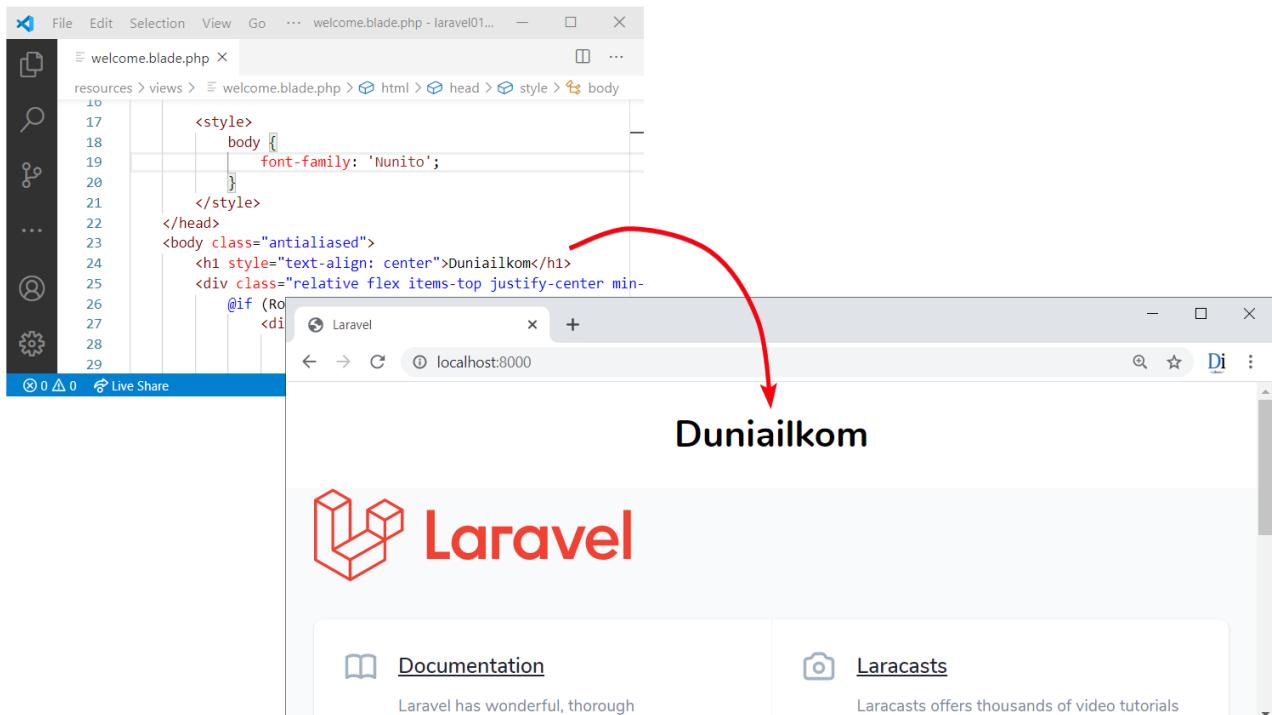
Selain itu file view juga **harus** berada di dalam folder `resources\views`.

File `welcome.blade.php` bawaan Laravel terdiri dari kode HTML, CSS, dan beberapa perintah blade. Untuk membuktikan bahwa inilah file yang memproses tampilan home Laravel, silahkan buka file `welcome.blade.php` lalu scroll hingga baris 23, yakni tepat pada pembuka tag `<body>`.

Kemudian tambah baris `<h1 style="text-align: center">DuniaIlkom</h1>` antara tag `<body>` dan tag `<div>` seperti contoh berikut :

```
23 <body class="antialiased">
24     <h1 style="text-align: center">DuniaIlkom</h1>
25     <div class="relative flex items-top justify-center min-h...
26     ...
```

Save file `welcome.blade.php` lalu buka kembali alamat `http://localhost:8000` di web browser:



Gambar: Mengubah kode dalam file `welcome.blade.php`

Sekarang terdapat teks "DuniaIlkom" di bagian paling atas.

7.3. Membuat View

Sampai di sini, kita sudah bisa mengambil beberapa kesimpulan tentang cara membuat view:

1. Buat route yang akan mengembalikan view.
2. Buat file view di folder `resources\views` dengan format `<nama_file>.blade.php`

Untuk langkah pertama, cukup tulis perintah `return view('nama_view')` ke dalam *anonymous function* di route.

Sebagai contoh, saya ingin ketika mengakses URL `http://localhost:8000/mahasiswa` akan ditampilkan view `mahasiswa`. Maka kode route-nya adalah sebagai berikut:

`routes/web.php`

```
1 Route::get('/mahasiswa', function () {
2     return view('mahasiswa');
3 });
```

Alamat URL dan nama view tidak harus sama, boleh suka-suka tergantung keperluan. Bisa saja kita membuat URL '`mahasiswa`' yang akan menampilkan view '`dosen`'.

`routes/web.php`

```
1 Route::get('/mahasiswa', function () {
2     return view('dosen');
3 });
```

Dengan penulisan seperti, maka di dalam folder `resources\views` harus terdapat file bernama `dosen.blade.php`.

Dalam Laravel, sering terdapat lebih dari 1 cara untuk menjalankan sebuah perintah. Ini membuat kode program menjadi lebih fleksibel, sekaligus membuat bingung karena ada banyak cara untuk melakukan sesuatu. Sebagai contoh, perhatikan kode berikut:

`routes/web.php`

```
1 Route::get('/mahasiswa', function () {
2     return View::make('mahasiswa');
3 });
```

Pada contoh sebelumnya, kita menggunakan perintah `return view('mahasiswa')` sebagai instruksi agar Laravel menampilkan view `mahasiswa`, namun kali ini perintah yang dipakai adalah `return View::make('mahasiswa')`.

Kedua penulisan ini sama-sama bisa dipakai dan tidak ada perbedaan. Secara teknis, `view()` adalah sebuah **helper function**, yakni function bantu yang disediakan oleh Laravel. Nantinya kita akan lihat berbagai *helper function* lain. Sedangkan pemanggilan `View::make()` dikenal dengan istilah **facade**.

Helper function vs Facade

Ketika membaca dokumentasi atau tutorial terkait Laravel, besar kemungkinan anda akan ketemu 2 istilah ini: *helper function* dan *facade*.

Helper function adalah sebutan untuk fungsi bawaan laravel yang memproses tugas tertentu. Sesuai dengan namanya, *helper function* berbentuk fungsi sederhana seperti `view()`, `dd()` dan `dump()`.

Facade adalah istilah *design pattern* yang merujuk kepada 'teknik membungkus' sebuah fitur dalam bentuk static method. Penjelasan detailnya cukup rumit, tapi bisa dibilang bahwa di dalam laravel, *facade* ini sebutan untuk '*helper function*' dalam bentuk static method. Sebagai contoh, `View::make()` adalah sebuah *facade*.

Dalam kebanyakan situasi, Laravel menyediakan *helper function* dan *facade* yang berfungsi sama. Misalnya untuk menampilkan view, tidak ada perbedaan antara *helper function* `view()` dengan *facade* `View::make()`. Secara internal, keduanya menjalankan kode program yang sama.

Setelah membuat route `mahasiswa` yang mengembalikan View '`mahasiswa`', maka kita harus menyiapkan file untuk view tersebut. Caranya, buat file dengan nama `mahasiswa.blade.php` di folder `resources\views`.

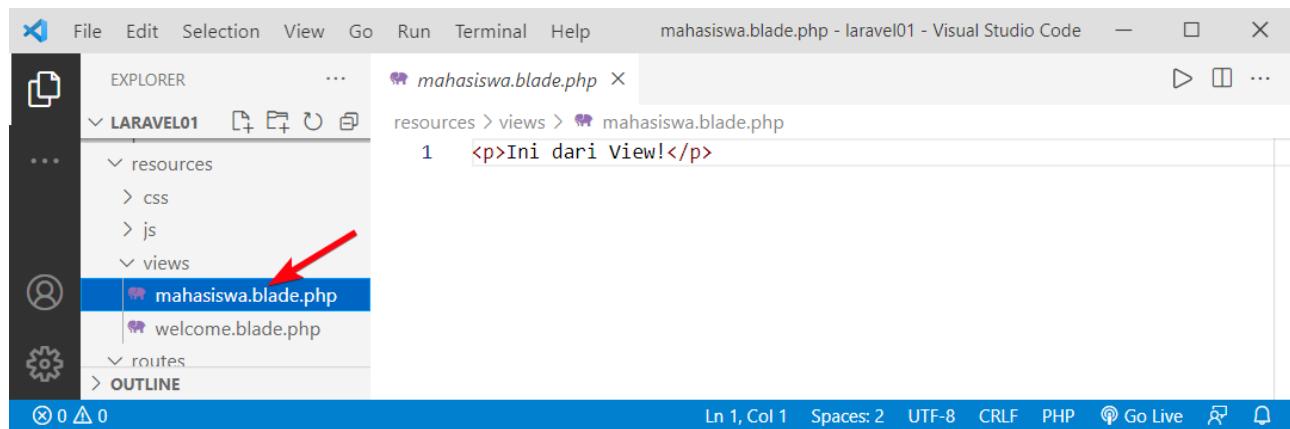
Berikut isi dari file `routes\web.php` dan file `resources\views\mahasiswa.blade.php`:

`routes\web.php`

```
1 Route::get('/mahasiswa', function () {
2     return view('mahasiswa');
3 });
```

`resources\views\mahasiswa.blade.php`

```
1 <p>Ini dari View!</p>
```



Gambar: Buat file mahasiswa.blade.php

View

Selanjutnya, akses route mahasiswa ini di web browser:



Gambar: Tampilan file mahasiswa.blade.php

Jika tampil teks 'Ini dari View!', maka artinya kita sudah sukses mengakses view.

Meskipun nama file view adalah `mahasiswa.blade.php`, tapi untuk mengaksesnya dari route cukup tulis sebagai `return view('mahasiswa')`. Kita tidak perlu menambah akhiran `.blade.php`. Laravel malah akan mengeluarkan pesan error jika view ditulis secara lengkap seperti `return view('mahasiswa.blade.php')`.

Agar penggunaan view ini lebih terlihat, saya akan isi dengan kode HTML lengkap. Silahkan edit kembali file `resources\views\mahasiswa.blade.php` dengan kode berikut:

`resources/views/mahasiswa.blade.php`

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Daftar Mahasiswa</title>
8  </head>
9  <body>
10     <h1>Daftar Mahasiswa</h1>
11     <ol>
12         <li>Rudi Permana</li>
13         <li>Sari Citra Lestari</li>
14         <li>Rina Kumala Sari</li>
15         <li>James Situmorang</li>
16     </ol>
17 </body>
18 </html>
```



Gambar: Daftar Mahasiswa

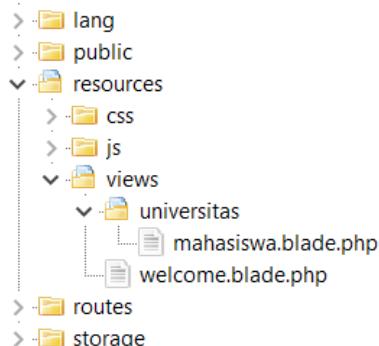
Sekarang isi dari view sudah terdiri dari sebuah kode HTML lengkap. Kita juga bisa menambah kode HTML, CSS dan JavaScript apapun ke dalam file ini.

7.4. Membuat Struktur Folder View

Pada praktik sebelum ini, file `mahasiswa.blade.php` berada langsung di bawah folder `resources\views`. Ini sebenarnya tidak masalah, tapi untuk aplikasi yang kompleks kita butuh sebuah manajemen file agar view lebih terstruktur.

Misalnya untuk membuat aplikasi sistem informasi universitas, halaman view yang dibutuhkan bisa terdiri dari puluhan file. Semua view ini perlu diatur ke dalam folder `mahasiswa`, folder `dosen`, folder `karyawan`, dst.

Sebagai contoh praktik, saya akan memindahkan file `mahasiswa.blade.php` ke dalam folder `universitas`. Sehingga alamat lengkap file ini ada di `resources\views\universitas\mahasiswa.blade.php`.



Gambar: File `mahasiswa.blade.php` sekarang berada di bawah folder `universitas`

Karena perubahan struktur folder, kita harus perbaiki lagi kode program untuk route.

Ketika menemukan perintah `return view()`, route akan mencari file view tersebut ke folder `resources\views`. Apabila file itu ada di dalam sub-folder (bukan langsung berada di folder `resources\views`), maka penulisan nama view juga harus menyertakan nama sub-folder tadi.

Berikut modifikasi route untuk perubahan ini:

```
routes/web.php

1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa');
3 });
```

Perhatikan cara penulisan perintah `return` di baris 2. Di sini saya menulis nama file view sebagai `'universitas.mahasiswa'`. Ini adalah instruksi kepada route bahwa view `mahasiswa` ada di dalam sub-folder `universitas`. Tanda titik di dalam penulisan `view()` berfungsi sebagai penanda nama folder.

Alternatifnya, kita juga bisa memisah penulisan folder dengan tanda *forward slash* seperti alamat URL biasa:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas/mahasiswa');
3 });
```

Mayoritas programmer Laravel lebih suka menggunakan cara pertama, yakni dengan menulis tanda titik sebagai pemisah nama sub-folder, namun cara kedua ini juga tidak salah.

Bisa juga dilihat bahwa perubahan nama view tidak akan mempengaruhi alamat URL. Inilah salah satu keuntungan dari pemisahan antara route dengan view. Jika menggunakan PHP native, perubahan nama file akan mengubah alamat URL karena nama file langsung di akses dari web browser.

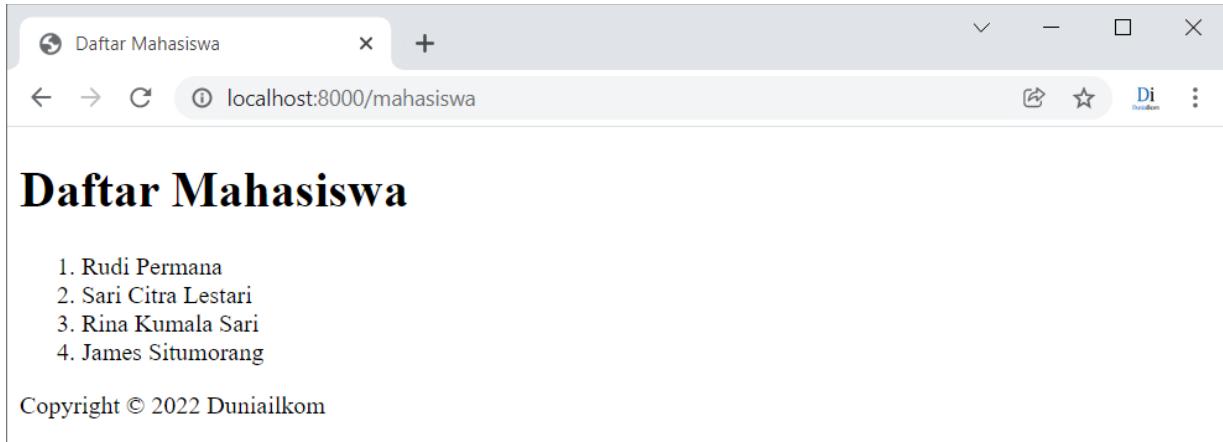
7.5. Kode PHP di dalam View

File view pada dasarnya tetap merupakan file PHP biasa. Meskipun mayoritas berisi kode front-end seperti HTML dan CSS, namun kita bisa menulis perintah PHP native ke dalam view.

Sebagai contoh, saya ingin menampilkan angka tahun di bagian bawah view `mahasiswa`. Agar tahun ini bisa update secara dinamis, saya ingin memakai fungsi `date()` bawaan PHP. Berikut perubahan dari file `mahasiswa.blade.php`:

resources/views/universitas/mahasiswa.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Daftar Mahasiswa</title>
8 </head>
9 <body>
10    <h1>Daftar Mahasiswa</h1>
11    <ol>
12        <li>Rudi Permana</li>
13        <li>Sari Citra Lestari</li>
14        <li>Rina Kumala Sari</li>
15        <li>James Situmorang</li>
16    </ol>
17    <div>
18        Copyright © <?php echo date("Y"); ?> DuniaIlkom
19    </div>
20 </body>
21 </html>
```



Gambar: Tampilan View mahasiswa dengan tambahan footer tahun

Perubahan dari kode sebelumnya ada di baris 17 – 19. Di sini saya membuat sebuah tag <div> yang berisi kode PHP echo date("Y"). Hasilnya, terlihat teks copyright di bagian bawah lengkap dengan tampilan tahun.

Selain perintah echo, kita juga bisa menulis kode PHP lain yang lebih kompleks seperti perulangan foreach, if else, dst. Cara penulisannya juga sama seperti kode PHP biasa, yakni dengan tag pembuka <?php dan tag penutup ?>.

7.6. Mengirim Data ke View

Apa yang kita pelajari sebelum ini baru sekedar menampilkan isi view tanpa perubahan apapun. Selain itu, kita juga bisa mengirim data dari route untuk ditampilkan secara dinamis oleh view.

Terdapat 2 cara pengiriman:

1. Menulis data sebagai argument kedua dari function `view()`.
2. Menggunakan method `with()`.

Kita akan bahas keduanya.

Pada prakteknya nanti, data yang dikirim ke view lebih banyak berasal dari controller. Untuk menyederhanakan pembahasan, kita akan pakai data yang diinput dari route terlebih dahulu. Di controller nanti, cara pengirimannya juga hampir sama.

Mengirim Data ke View Sebagai Argumen

Cara pertama untuk mengirim data dari route ke view adalah dengan menulis data tersebut sebagai argumen kedua pada saat pemanggilan function `view()`. Berikut contoh penggunaannya:

View

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa',[ "mahasiswa01" => "Risa Lestari"]);
3 });
```

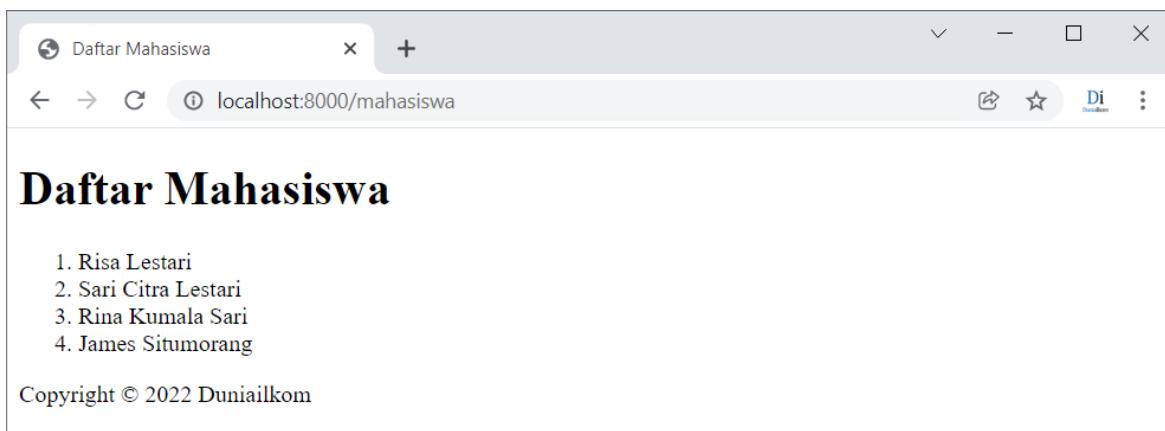
Perhatikan cara pemanggilan function `view()` di baris 2. Sebagai argumen pertama terdapat nama view yang akan ditampilkan, yang dalam contoh ini saya ingin mengakses view `'universitas.mahasiswa'`.

Selanjutnya sebagai argument kedua terdapat associative array berbentuk `["mahasiswa01" => "Risa Lestari"]`. Inilah data yang ingin saya kirim ke view `'universitas.mahasiswa'`.

Agar data ini bisa "ditangkap" oleh view, kita harus modifikasi kode program `mahasiswa.blade.php`. Buka kembali file ini dan ubah sebagai berikut:

resources/views/universitas/mahasiswa.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Daftar Mahasiswa</title>
8 </head>
9 <body>
10    <h1>Daftar Mahasiswa</h1>
11    <ol>
12        <li><?php echo $mahasiswa01; ?></li>
13        <li>Sari Citra Lestari</li>
14        <li>Rina Kumala Sari</li>
15        <li>James Situmorang</li>
16    </ol>
17    <div>
18        Copyright © <?php echo date("Y"); ?> DuniaIlkom
19    </div>
20 </body>
21 </html>
```



Gambar: Menampilkan hasil kiriman data dari Route

View

Di baris 12 terdapat perintah echo \$mahasiswa01, inilah cara menampilkan data yang dikirim dari Route.

Pada saat penulisan route, array ["mahasiswa01" => "Risa Lestari"] diinput sebagai argumen kedua function view(). Array ini akan berubah jadi variabel \$mahasiswa01 ketika sampai di dalam view.

Untuk mengirim lebih banyak data, tambah element baru ke dalam array yang akan dikirim:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa',
3     [
4         "mahasiswa01" => "Risa Lestari",
5         "mahasiswa02" => "Rudi Hermawan",
6         "mahasiswa03" => "Bambang Kusumo",
7         "mahasiswa04" => "Lisa Permata"
8     ]);
9 });
```

Jika ditulis seperti ini, bisakah anda merancang view untuk menampilkan seluruh nama mahasiswa?

Yup, di dalam view universitas.mahasiswa, setiap element array akan "dibuka" dan berubah menjadi variabel \$mahasiswa01, \$mahasiswa02, \$mahasiswa03, dan \$mahasiswa04. Untuk mengaksesnya kita bisa gunakan cara berikut:

resources/views/universitas/mahasiswa.blade.php

```
1 ...
2 <h1>Daftar Mahasiswa</h1>
3 <ol>
4     <li><?php echo $mahasiswa01; ?></li>
5     <li><?php echo $mahasiswa02; ?></li>
6     <li><?php echo $mahasiswa03; ?></li>
7     <li><?php echo $mahasiswa04; ?></li>
8 </ol>
9 ...
```

Hasil kode program:

Daftar Mahasiswa

Risa Lestari
Rudi Hermawan
Bambang Kusumo
Lisa Permata

Untuk menghemat tempat, saya tidak lagi menampilkan kode HTML di mahasiswa.blade.php secara lengkap, kita akan fokus ke bagian list-nya saja.

Agar lebih rapi, kode program di bagian route bisa ditulis sebagai berikut:

routes/web.php

```

1 Route::get('/mahasiswa', function () {
2
3     $arrMahasiswa = [
4         "mahasiswa01" => "Risa Lestari",
5         "mahasiswa02" => "Rudi Hermawan",
6         "mahasiswa03" => "Bambang Kusumo",
7         "mahasiswa04" => "Lisa Permata"
8     ];
9
10    return view('universitas.mahasiswa', $arrMahasiswa);
11 });

```

Di sini saya memisahkan kode untuk pembuatan array ke luar function `view()`. Sehingga kita cukup menulis `$arrMahasiswa` sebagai argument kedua yang akan dikirim ke view.

Proses menampilkan data seperti ini tidak salah, tapi masih kurang efisien. Di dalam view kita harus men-echo satu per satu variabel `$mahasiswa01`, `$mahasiswa02`, dst. Karena ini merupakan data yang berulang dan sama (berisi nama-nama mahasiswa), maka seharusnya bisa diproses menggunakan perulangan `foreach` PHP.

Namun struktur array yang dikirim juga harus diubah. Agar bisa diakses menggunakan perulangan `foreach`, kita harus rancang supaya data yang sampai di view masih berbentuk array, bukan variabel yang sudah dipecah. Solusinya, kirim data dalam bentuk nested array seperti contoh berikut:

routes/web.php

```

1 Route::get('/mahasiswa', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                       "Lisa Permata"];
4
5     return view('universitas.mahasiswa', ['mahasiswa' => $arrMahasiswa]);
6 });

```

Di baris 2 saya membuat `$arrMahasiswa` yang berisi daftar nama mahasiswa. Ketika array ini di kirim ke view, ditulis lagi sebagai `['mahasiswa' => $arrMahasiswa]`. Sehingga begitu sampai di View, nama mahasiswa tetap berbentuk array dan bisa diakses dengan cara berikut:

resources/views/universitas/mahasiswa.blade.php

```

1 <h1>Daftar Mahasiswa</h1>
2 <ol>
3     <li><?php echo $mahasiswa[0]; ?></li>
4     <li><?php echo $mahasiswa[1]; ?></li>
5     <li><?php echo $mahasiswa[2]; ?></li>
6     <li><?php echo $mahasiswa[3]; ?></li>
7 </ol>

```

Anda boleh luangkan waktu sejenak untuk memahami alur proses pengiriman data di atas, karena konsep ini sangat sangat penting sepanjang pembuatan aplikasi dengan Laravel.

Karena data yang sampai masih berbentuk array, bisa kita *loop* menggunakan perulangan `foreach`:

`resources/views/universitas/mahasiswa.blade.php`

```

1 <h1>Daftar Mahasiswa</h1>
2 <ol>
3 <?php
4   foreach ($mahasiswa as $nama) {
5     echo "<li> $nama </li>";
6   }
7 <?>
8 </ol>
```

Dengan teknik ini, kode program PHP di view menjadi lebih *simple*. Tidak peduli berapa banyak data yang dikirim dari route, selama data tersebut berbentuk array, maka bisa di-looping menggunakan `foreach`.

Mengirim Data ke View Menggunakan method With

Cara kedua yang bisa dipakai untuk mengirim data dari route ke view adalah menggunakan method `with()`. Berikut contoh penulisannya:

`routes/web.php`

```

1 Route::get('/mahasiswa', function () {
2   return view('universitas.mahasiswa')->with('mahasiswa01', 'Risa Lestari');
3 });
```

Method `with()` dipanggil dari function `view()`, teknik seperti ini dikenal dengan nama *method chaining*. Jika anda sudah membaca buku **OOP PHP Uncover**, di sana kita pernah membahas cara membuat struktur seperti ini.

Method `with()` butuh 2 buah argument, argument pertama berupa nama variabel yang akan dikirim, serta argument kedua untuk menampung nilai yang dikirim.

Dengan route di atas, maka di dalam view `universitas.mahasiswa` akan terdapat variabel `$mahasiswa01` yang berisi string '`Risa Lestari`'. Sehingga bisa di tampilkan dengan kode berikut:

`resources/views/universitas/mahasiswa.blade.php`

```

1 <h1>Daftar Mahasiswa</h1>
2 <ol>
3   <li><?php echo $mahasiswa01; ?></li>
4 </ol>
```

Berikutnya, bagaimana cara mengirim data agar bisa di loop menggunakan `foreach`? Tidak

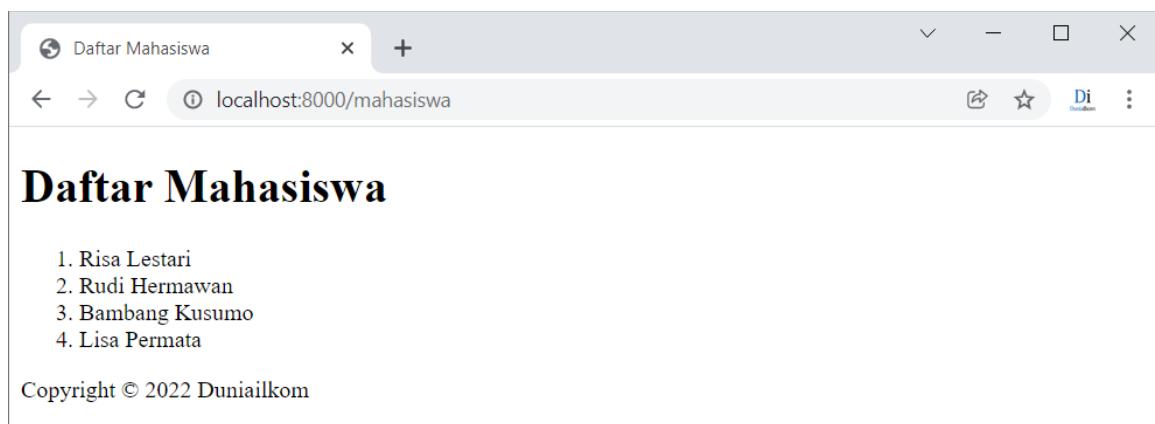
masalah, kita tinggal mengirim data dalam bentuk array seperti contoh sebelumnya:

```
1 Route::get('/mahasiswa', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                       "Lisa Permata"];
4
5     return view('universitas.mahasiswa')->with('mahasiswa', $arrMahasiswa);
6 });
```

Hasilnya, di dalam view bisa diakses dengan perulangan `foreach`:

`resources/views/universitas/mahasiswa.blade.php`

```
1 <h1>Daftar Mahasiswa</h1>
2 <ol>
3   <?php
4     foreach ($mahasiswa as $nama) {
5       echo "<li> $nama </li>";
6     }
7   ?>
8 </ol>
```



Gambar: Hasil tampilan data di View

Jika ingin mengirim banyak variabel, method `view()` ini bisa dipanggil beberapa kali dengan cara di *chaining* (di sambung) satu sama lain. Berikut contoh penggunaannya:

`routes/web.php`

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa')->with('mahasiswa01', 'Risa Lestari')
3       ->with('mahasiswa02', 'Rudi Hermawan')->with('mahasiswa03', 'Bambang Kusumo');
4 });
```

Di sini saya mengirim 3 buah data, yakni `mahasiswa01`, `mahasiswa02` dan `mahasiswa03`. Di dalam view, data tersebut akan menjadi `$mahasiswa01`, `$mahasiswa02`, dan `$mahasiswa03`.

Agar lebih rapi, penulisan method *chaining* bisa disusun sebagai berikut:

View

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa')
3     ->with('mahasiswa01', 'Risa Lestari')
4     ->with('mahasiswa02', 'Rudi Hermawan')
5     ->with('mahasiswa03', 'Bambang Kusumo');
6 });
```

Cara penulisan lain dari method `with()` adalah dengan menyambung langsung setiap nama variabel. Sebagai contoh, untuk mengirim data `mahasiswa01` ke dalam view, bisa menggunakan kode berikut:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa')->withmahasiswa01('Risa Lestari');
3 });
```

Penulisan seperti ini memang terlihat agak aneh, tapi bisa dilakukan. Perhatikan bahwa yang di chaining adalah `->withmahasiswa01('Risa Lestari')`, bukan hanya sekedar `->with()` saja. Di dalam view, variabel `$mahasiswa01` bisa diakses dan akan berisi string `'Risa Lestari'`.

Aturan penulisan penulisan method ini adalah sebagai berikut:

`with<nama_variabel>(nilai_variabel)`

Kita juga bisa membuat teknik *chaining* yang saling bersambung:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     return view('universitas.mahasiswa')
3     ->withmahasiswa01('Risa Lestari')
4     ->withmahasiswa02('Rudi Hermawan')
5     ->withmahasiswa03('Bambang Kusumo');
6 });
```

Sama seperti sebelumnya, ketika sudah sampai di view, data ini bisa diambil dengan mengakses `$mahasiswa01`, `$mahasiswa02`, dan `$mahasiswa03`.

Mengenal Function `compact()`

Function `compact()` bukanlah *helper function* dari Laravel, tapi function bawaan PHP yang sudah tersedia sejak PHP 4. Namun memang fungsi ini relatif jarang dipakai.

Function `compact()` berguna untuk membuat *associative array* dari variabel biasa. Karena data yang dikirim ke view biasanya berbentuk *associative array*, maka fungsi `compact()` cukup sering dipakai dalam Laravel.

Berikut contoh cara kerja dari fungsi `compact()`:

compact.php

```

1 <?php
2 $mahasiswa01 = "Risa Lestari";
3 $mahasiswa02 = "Rudi Hermawan";
4 $mahasiswa03 = "Bambang Kusumo";
5 $mahasiswa04 = "Lisa Permata";
6
7 $arr = compact("mahasiswa01", "mahasiswa02", "mahasiswa03", "mahasiswa04");
8
9 echo $arr["mahasiswa01"]; // Risa Lestari
10 echo $arr["mahasiswa02"]; // Rudi Hermawan
11 echo $arr["mahasiswa03"]; // Bambang Kusumo
12 echo $arr["mahasiswa04"]; // Lisa Permata

```

Kode ini bukanlah bagian dari Laravel, tapi kode PHP biasa yang bisa di jalankan dari `htdocs` XAMPP.

Di baris 2 – 5 saya membuat 4 variabel dengan nama `$mahasiswa01` s/d `$mahasiswa04`.

Kemudian di baris 7 semua variabel ini diinput ke dalam fungsi `compact()` dan hasilnya disimpan ke dalam variabel `$arr`.

Perhatikan bahwa argument untuk fungsi `compact()` ini berbentuk string sesuai dengan nama variabel, misalkan kita memiliki variabel `$mahasiswa01`, maka di dalam fungsi `compact()` ditulis menjadi `compact("mahasiswa01")`. Hasilnya, variabel `$arr` berisi *associative array* dengan nama key berupa nama variabel.

Di dalam Laravel, fungsi `compact()` ini sering dipakai untuk mempersiapkan array sebelum dikirim ke dalam view. Berikut contoh penggunaannya:

routes/web.php

```

1 Route::get('/mahasiswa', function () {
2
3     $mahasiswa01 = "Risa Lestari";
4     $mahasiswa02 = "Rudi Hermawan";
5     $mahasiswa03 = "Bambang Kusumo";
6     $mahasiswa04 = "Lisa Permata";
7
8     return view('universitas.mahasiswa', compact("mahasiswa01", "mahasiswa02",
9         "mahasiswa03", "mahasiswa04"));
10 });

```

Atau jika menggunakan method `with()`, bisa dituliskan sebagai berikut:

routes/web.php

```

1 Route::get('/mahasiswa', function () {
2
3     $mahasiswa01 = "Risa Lestari";
4     $mahasiswa02 = "Rudi Hermawan";
5     $mahasiswa03 = "Bambang Kusumo";
6     $mahasiswa04 = "Lisa Permata";

```

```

7
8   return view('universitas.mahasiswa')->with(compact("mahasiswa01",
9     "mahasiswa02", "mahasiswa03", "mahasiswa04"));
10 });

```

Dengan pemanggilan seperti ini, di dalam view `universitas.mahasiswa` kita bisa langsung mengakses variabel `$mahasiswa01`, `$mahasiswa02`, dst.

Exercise

Dengan modal materi seputar route dan mengirim data ke view, kita akan buat latihan sederhana. Saya ingin membuat tampilan view yang berasal dari alamat URL.

Jika diketik `http://localhost:8000/mahasiswa/Lisa/19/Bandung`, maka akan tampil teks:

```

Daftar Mahasiswa
Nama: Lisa
Umur: 19
Kota Asal: Bandung

```

Jika diketik `http://localhost:8000/mahasiswa/Rudi/24/Jakarta`, akan tampil teks:

```

Daftar Mahasiswa
Nama: Rudi
Umur: 24
Kota Asal: Jakarta

```

Hasil teks tidak boleh langsung ditampilkan dari route saja, tapi dikirim ke view terlebih dahulu. Di sini kita harus memakai *route parameter* untuk "menangkap" nilai dari URL.

Answer

Pertama untuk route, bisa menggunakan kode program berikut:

```

1 Route::get('/mahasiswa/{nama}/{umur}/{kotaAsal}', 
2   function ($nama, $umur, $kotaAsal) {
3     return view('universitas.mahasiswa')
4       ->with('nama', $nama)
5       ->with('umur', $umur)
6       ->with('kotaAsal', $kotaAsal);
7 });

```

Setiap segmen dari URL akan ditampung oleh variabel `$nama`, `$umur` dan `$kotaAsal`. Kemudian semua variabel dikirim menggunakan 3 buah method `with()` ke dalam view.

Sesampainya di view `universitas.mahasiswa`, ketiga variabel bisa ditampilkan sebagai berikut:

```

1 <h1>Daftar Mahasiswa</h1>
2 <ul>
3   <li>Nama: <?php echo $nama; ?></li>
4   <li>Umur: <?php echo $umur; ?></li>
5   <li>Kota Asal: <?php echo $kotaAsal; ?></li>
6 </ul>

```

Dengan mengubah isi segmen URL, teks yang tampil juga akan berubah.

7.7. Pengelolaan Assets

Dalam web programming terdapat istilah **assets**, atau **web assets**. Ini sebutan untuk file external seperti file CSS, file JavaScript, gambar, dan icon. Untuk project besar, kita butuh sebuah struktur pengelolaan assets agar semua file tersimpan rapi.

Cara sederhana yang biasa dilakukan adalah dengan membuat folder **css**, folder **js**, dan folder **img** untuk menyimpan file assets sesuai jenis filenya. Di Laravel kita juga bisa membuat struktur folder seperti ini.

Pengelolaan assets masih berhubungan dengan materi tentang view, karena view adalah tempatnya membuat tampilan akhir. Selain HTML, di dalam view juga bisa terdapat kode CSS dan JavaScript. Untuk kode internal tidak ada masalah, karena kita bisa langsung menulis tag `<style>` dan tag `<script>` ke dalam file view.

Tapi bagaimana dengan file CSS dan file JavaScript external? yakni file CSS dan JavaScript yang disimpan ke dalam file tersendiri? Di manakah file ini harus diletakkan?

Pengelolaan assets, terutama file CSS dan JavaScript sebenarnya ditangani oleh fitur khusus Laravel yang disebut sebagai **Laravel Mix**. Namun kali ini kita akan bahas cara penyimpanan dan pengaksesan file assets secara manual terlebih dahulu. Untuk Laravel Mix akan saya bahas dalam bab terpisah.

Folder Public

Sesuai namanya, folder **public** adalah folder Laravel yang bisa diakses oleh public. Di sinilah semua file yang boleh diakses pengunjung web berada:



Gambar: Isi folder public Laravel

Bawaan Laravel, folder **public** berisi 5 file:

- **.htaccess** dan **web.config**: Kedua file ini merupakan file konfigurasi untuk web server. File **.htaccess** ditujukan untuk web server Apache serta beberapa web server berbasis Linux lain seperti Nginx. Sedangkan file **web.config** ditujukan untuk web server Microsoft IIS. Untuk saat ini kedua file tidak perlu kita utak-atik.
- **favicon.ico**: File gambar untuk tampilan favicon, yakni gambar kecil di sudut kiri atas web browser. Bawaan laravel, file **favicon.ico** hanya gambar icon dummy, file ini berukuran 0 byte yang artinya tidak ada gambar sama sekali. Nantinya kita bisa ganti dengan gambar favicon lain.
- **robots.txt**: File khusus yang ditujukan untuk mesin pencari seperti Google. Di dalam file ini kita bisa menulis daftar halaman web yang tidak ingin di index oleh Google.
- **index.php**: Inilah file yang diakses Laravel pertama kali pada setiap pemrosesan halaman. Misalnya ketika kita mengetik alamat URL `http://localhost:8000/mahasiswa`, maka file **index.php** ini akan di proses pertama kali. Di dalamnya terdapat kode program khusus yang akan memanggil berbagai file Laravel. Kita juga tidak perlu mengutak-atik file ini, biarkan saja apa-adanya.

Hampir semua file yang ada di folder ini tidak perlu kita buka, jadi apa fungsinya?

Kunci utama dari ini semua ada di file **index.php**. Inilah file awal yang selalu berjalan ketika kita mengakses sebuah halaman web, sehingga view-pun "berangkat" dari file ini. Dengan demikian, di dalam folder public inilah kita bisa menyimpan file assets, seperti file CSS, file JavaScript serta file gambar. Kita akan lihat cara penggunaannya dengan praktik.

Pertama, siapkan kode route dan view. Saya akan menggunakan route **mahasiswa** sebelumnya:

`routes/web.php`

```

1 Route::get('/mahasiswa', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                      "Lisa Permata"];
4
5     return view('universitas.mahasiswa', ['mahasiswa' => $arrMahasiswa]);
6 });

```

`resources/views/universitas/mahasiswa.blade.php`

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Daftar Mahasiswa</title>
8 </head>
9 <body>

```

```

10 <h1>Daftar Mahasiswa</h1>
11 <ol>
12   <?php
13     foreach ($mahasiswa as $nama) {
14       echo "<li> $nama </li>";
15     }
16   ?>
17 </ol>
18 <div>
19   Copyright © <?php echo date("Y"); ?> DuniaIlkom
20 </div>
21 </body>
22 </html>

```

Tidak ada hal baru di sini, dimana saya mengirim array mahasiswa ke dalam view, yang kemudian di loop menggunakan perulangan foreach.

Berikutnya, silahkan buat folder **css** dan folder **js** di dalam folder **public**. Dalam kedua folder inilah file CSS dan JavaScript akan kita simpan.

Setelah itu, buka teks editor dan ketik kode berikut:

```
public/css/my-style.css
```

```

1 body {
2   background-color: aliceblue;
3 }

```

Ini adalah kode CSS sederhana. Isinya dipakai untuk mengubah warna background dari tag `<body>` menjadi *aliceblue*, yakni warna biru langit. Simpan file di atas dengan nama `my-style.css` ke dalam folder `public\css`.

Berikutnya, buat file kedua dengan kode program berikut:

```

1 let listMahasiswa = document.getElementsByTagName("ol");
2 listMahasiswa[0].addEventListener("click",tampilkan);
3
4 function tampilkan(event){
5   alert("Cek data mahasiswa"+event.target.innerHTML);
6 }

```

Ini merupakan kode JavaScript. Jika anda sudah pernah belajar JavaScript, misalnya dari buku **JavaScript Uncover**, saya yakin sudah bisa memahami kode ini, meskipun mungkin butuh waktu yang agak lama (eh...).

Di baris 1 saya menggunakan method `getElementsByName()` untuk mencari semua tag ol yang ada di halaman, lalu menyimpan seluruh element yang ditemukan ke dalam variabel `listMahasiswa`.

Di baris 2, method `addEventListener()` dipakai untuk 'menempelkan' event **click**. Perintah `listMahasiswa[0]` artinya saya ingin mengakses element pertama saja dari semua tag ol yang

ditemukan. Ini perlu dilakukan karena hasil dari `getElementsByName()` berbentuk array, meskipun di halaman hanya ada 1 tag ``.

Pada saat isi element tag "ol" di klik, function `tampilkan()` di baris 4 – 6 akan dipanggil. Isinya, tampilkan pesan `alert` dengan nilai yang diambil dari atribut `innerHTML`. Dalam teori DOM JavaScript, atribut `innerHTML` berisi teks yang terdapat di antara tag pembuka dan penutup element. Dalam contoh kita, `innerHTML` akan berisi teks di antara tag `` dan ``.

Jika anda kurang paham dengan penjelasan ini, tidak masalah. Isinya tidak berhubungan dengan pembahasan materi Laravel kita, hanya sekedar menguji jalan atau tidaknya kode JavaScript. Save file di atas sebagai `my-script.js` ke dalam folder `public\js`.

Sekarang, kita masuk ke inti dari materi, yakni bagaimana caranya mengakses kedua file ini dari dalam view?

Kembali, file yang dijalankan pertama kali oleh Laravel pada saat sebuah alamat di akses adalah file `index.php` yang ada di dalam folder **public**. Maka, semua alamat URL assets yang ditulis akan berangkat dari folder ini.

Berikut perubahan file view `mahasiswa.blade.php` dengan penambahan link ke file CSS dan file JavaScript external:

`resources/views/universitas/mahasiswa.blade.php`

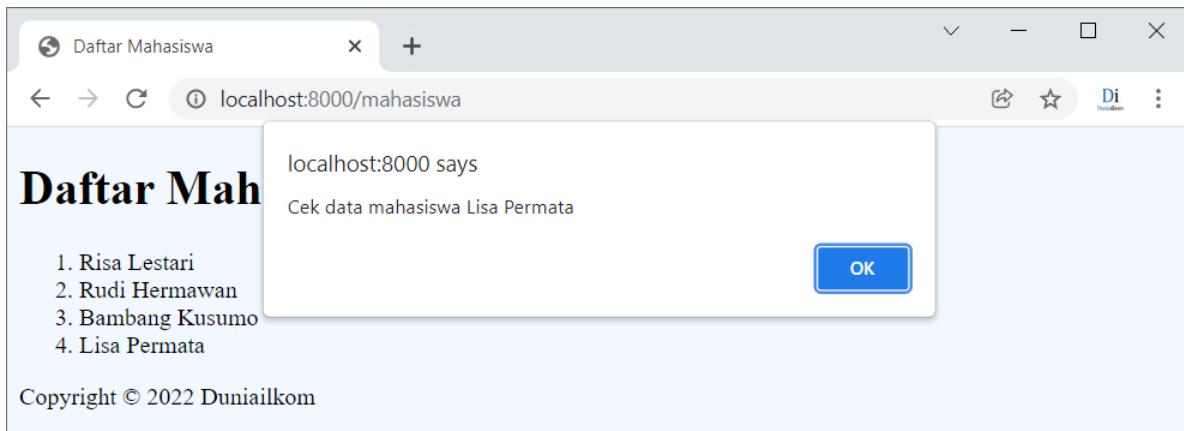
```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Daftar Mahasiswa</title>
8      <link href="/css/my-style.css" rel="stylesheet">
9  </head>
10 <body>
11     <h1>Daftar Mahasiswa</h1>
12     <ol>
13         <?php
14             foreach ($mahasiswa as $nama) {
15                 echo "<li> $nama </li>";
16             }
17         ?>
18     </ol>
19     <div>
20         Copyright © <?php echo date("Y"); ?> DuniaIlkom
21     </div>
22     <script src='/js/my-script.js'></script>
23 </body>
24 </html>

```

Perhatikan tambahan di baris 8 dan 22. Inilah cara mengakses file `my-style.css` dan file `my-script.js` yang tersimpan di folder `public`.

Berikut tampilannya di web browser:



Gambar: tampilan view dengan efek CSS dan alert JavaScript

Meskipun tidak terlalu terlihat, tapi warna background sudah berubah menjadi biru muda, yang berasal dari file `my-style.css`.

Serta juga tampil pesan alert 'Cek data mahasiswa Lisa Permata' sewaktu nama mahasiswa `Lisa Permata` di klik, ini berasal dari file `my-script.js`. Jika anda men-klik nama mahasiswa lain, pesan alert yang tampil juga akan berubah.

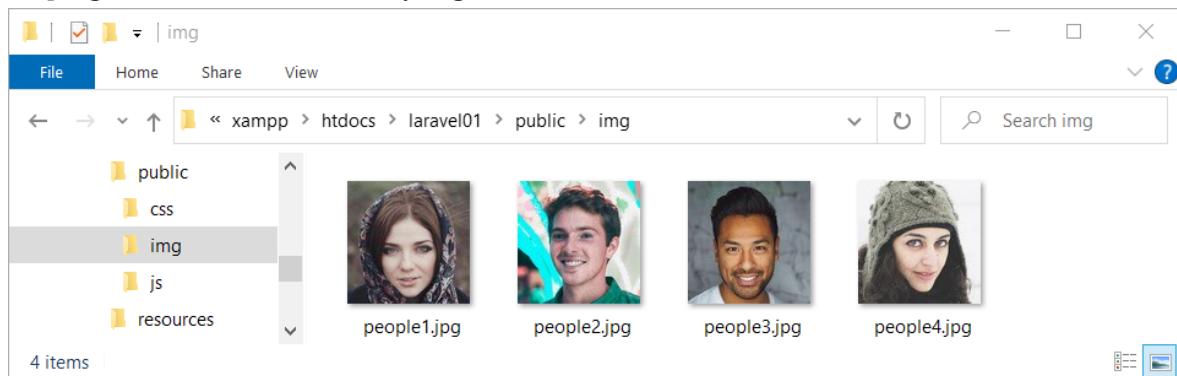
Kesimpulan dari percobaan ini adalah, alamat file assets dihitung relatif kepada file `index.php`. Meskipun file view berada di folder yang berbeda, yakni di `resources/views`. Di sini Laravel kembali memainkan 'magic' nya.

Sebagai contoh lain, misalkan saya memiliki file `bootstrap.css` yang berada di folder `public/css/file_bootstrap/bootstrap.css`, maka penulisan tag `<link>` di dalam view adalah sebagai berikut:

```
<link href="/css/file_bootstrap/bootstrap.css" rel="stylesheet">
```

Ini juga berlaku untuk file lain, termasuk gambar. Mari kita coba.

Pertama, saya akan buat folder `img` di dalam folder `public`, kemudian mengisinya dengan beberapa gambar. Gambar inilah yang akan kita akses dari dalam view.

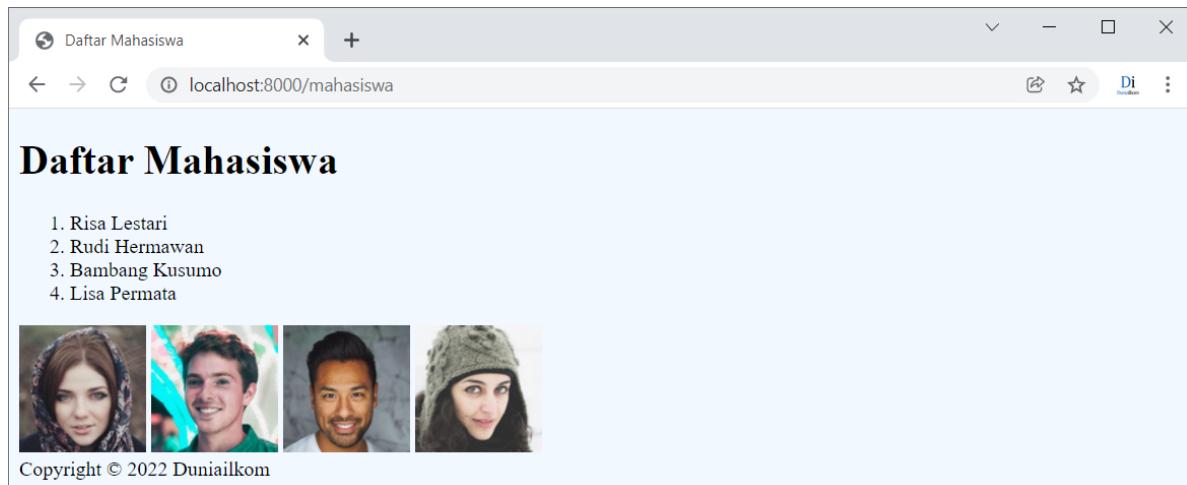


Gambar: isi beberapa gambar ke dalam folder img

Saya mengisi folder **img** dengan 4 gambar: `people1.jpg`, `people2.jpg`, `people3.jpg`, serta `people4.jpg`. Dan berikut isi view `mahasiswa.blade.php` dengan tambahan kode untuk menampilkan gambar-gambar ini:

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Daftar Mahasiswa</title>
8      <link href="/css/my-style.css" rel="stylesheet">
9  </head>
10 <body>
11     <h1>Daftar Mahasiswa</h1>
12     <ol>
13         <?php
14             foreach ($mahasiswa as $nama) {
15                 echo "<li> $nama </li>";
16             }
17         ?>
18     </ol>
19
20     <div>
21         
22         
23         
24         
25     </div>
26
27     <div>
28         Copyright © <?php echo date("Y"); ?> DuniaIlkom
29     </div>
30 <script src='/js/my-script.js'></script>
31 </body>
32 </html>
```



Gambar: Di dalam view mahasiswa, sudah tampil gambar yang tersimpan di public\img

Tambahan kode program ada di baris 20 – 25, dimana saya mengakses file gambar menggunakan alamat path seperti `/img/people1.jpg`. Kembali, patokannya berangkat dari file `index.php` yang ada di folder `public`.

Tanda Forward Slash di Awal URL Assets

Ketika menulis alamat URL untuk assets, selalu tambahkan tanda forward slash atau `/` sebelum penulisan nama file atau folder. Contohnya:

```
<link href="/css/my-style.css" rel="stylesheet">
```

Dan **bukan**:

```
<link href="css/my-style.css" rel="stylesheet">
```

Tambahan tanda `/` di awal harus ditulis agar web browser mencari file mulai dari root folder asset, yakni dari folder **public**. Jika tanda `/` tidak ditulis, maka web browser akan menerjemahkan URL tersebut sebagai alamat yang **relatif terhadap URL saat ini**.

Agar lebih mudah dijelaskan, kembali kita buat sedikit percobaan. Silahkan modifikasi view `mahasiswa.blade.php`, dan hapus semua awalan `/` yang merujuk ke file assets:

`resources/views/universitas/mahasiswa.blade.php`

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Daftar Mahasiswa</title>
8      <link href="css/my-style.css" rel="stylesheet">
9  </head>
10 <body>
11     <h1>Daftar Mahasiswa</h1>
12     <ol>
13         <?php
14             foreach ($mahasiswa as $nama) {
15                 echo "<li> $nama </li>";
16             }
17         ?>
18     </ol>
19
20     <div>
21         
22         
23         
24         
25     </div>
26
27     <div>
28         Copyright © <?php echo date("Y"); ?> DuniaIlkom
29     </div>

```

View

```
30 <script src='js/my-script.js'></script>
31 </body>
32 </html>
```

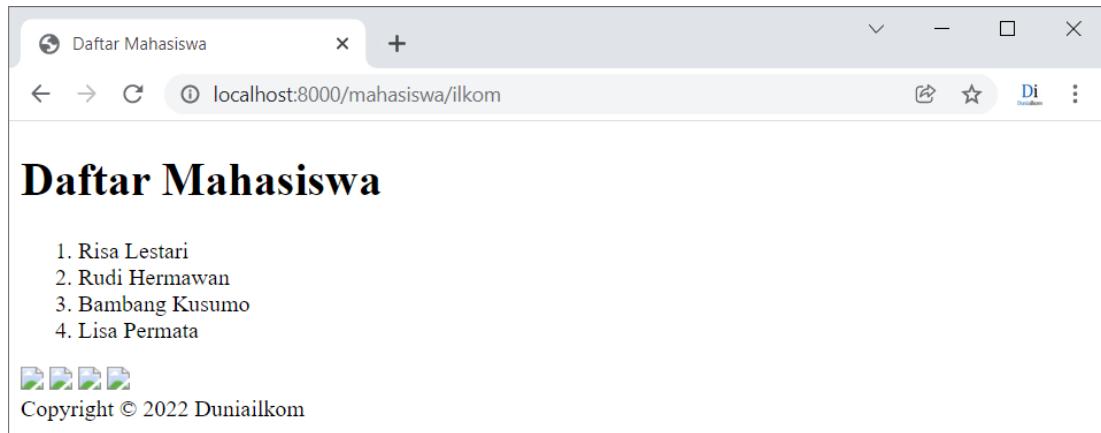
Perhatikan baris 8, 21-24, dan 30. Itulah tempat kita mengakses file assets, sekarang semuanya tidak memakai awalan forward slash lagi. Save view `mahasiswa` dan cek di web browser.

Betul, tidak tampak perubahan apa-apa. File CSS, JS dan gambar tetap tampil sempurna. Kemudian, modifikasi route untuk halaman mahasiswa menjadi sebagai berikut:

routes/web.php

```
1 Route::get('/mahasiswa/ilkom', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                       "Lisa Permata"];
4
5     return view('universitas.mahasiswa', ['mahasiswa' => $arrMahasiswa]);
6 });
```

Perubahannya ada di baris 1. Sekarang URL view `mahasiswa` saya tukar dari `'/mahasiswa'` menjadi `'/mahasiswa/ilkom'`. Sehingga alamat view juga berubah dari `http://localhost:8000/mahasiswa` menjadi `http://localhost:8000/mahasiswa/ilkom`.



Gambar: File assets tidak tampil

Hasilnya, tidak ada gambar yang tampil. Warna background juga kembali putih yang menandakan efek CSS tidak aktif. Termasuk saat nama mahasiswa di klik juga tidak terjadi apa-apa yang berarti file JS juga gagal diakses.

Untuk melihat apa yang terjadi, silahkan buka hasil source code dari web browser:



Gambar: URL untuk assets

Saya ingin mencari tau apa sebenarnya alamat URL untuk file CSS. Ternyata yang diakses oleh web browser adalah: <http://localhost:8000/mahasiswa/css/my-style.css>. Alamat ini salah, karena file `my-style.css` seharusnya diakses dari <http://localhost:8000/css/my-style.css>, yakni tidak ada tambahan "`mahasiswa`".

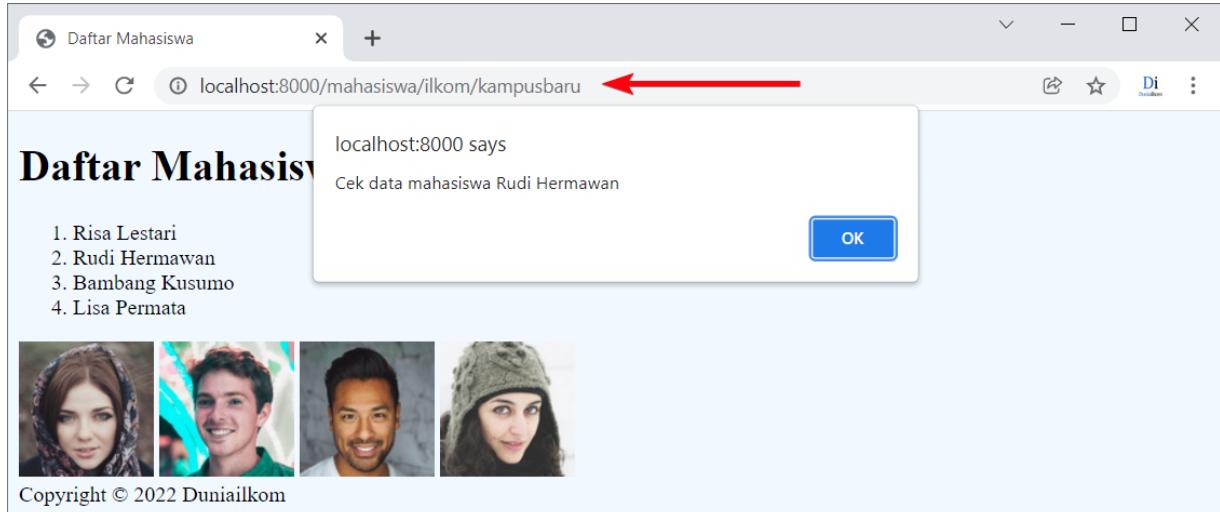
Hal yang sama jika terjadi untuk file gambar dan file JavaScript. Alamat yang akan diakses oleh web browser adalah <http://localhost:8000/mahasiswa/img/people1.jpg>, dimana seharusnya <http://localhost:8000/img/people1.jpg>. Kembali, terdapat tambahan segmen "`mahasiswa`" dalam URL.

Jika kita tukar route menjadi sebagai berikut:

```
1 Route::get('/mahasiswa/ilkom/kampusbaru', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                      "Lisa Permata"];
4
5     return view('universitas.mahasiswa', ['mahasiswa' => $arrMahasiswa]);
6 });
```

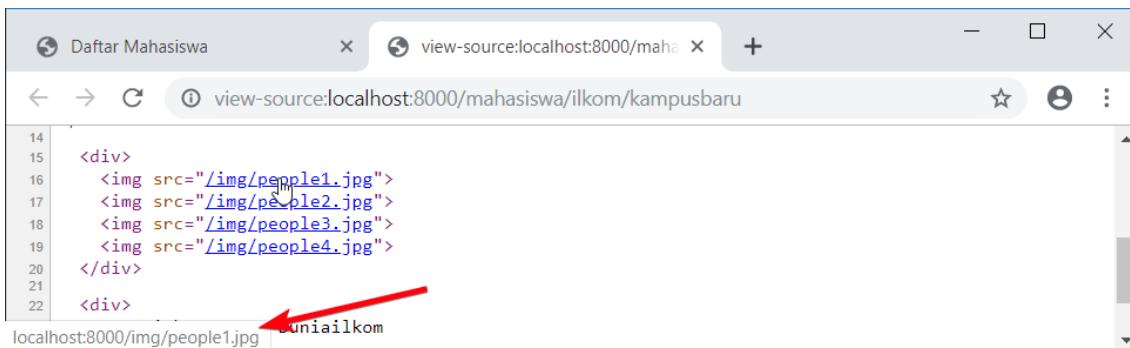
Maka alamat file assets akan berubah lagi menjadi <http://localhost:8000/mahasiswa/kampusbaru/img/people1.jpg>. Inilah yang saya maksud dengan web browser akan menerjemahkan URL tersebut sebagai alamat yang **relatif terhadap URL saat ini**. Ketika URL route diubah, alamat file assets juga ikut berubah.

Dengan penambahan tanda "/" di awal setiap alamat assets, maka URL akan selalu diakses dari folder **public**, bukan relatif ke URL saat ini. Silahkan tambah kembali tanda "/" di setiap URL assets, dan buka alamat <http://localhost:8000/mahasiswa/ilkom/kampusbaru>.



Gambar: Semua assets kembali tampil

Hasilnya, semua assets kembali tampil. Jika kita buka source code di web browser, alamatnya juga sudah sesuai, tanpa tambahan "mahasiswa/ilkom/kampusbaru":



Gambar: URL untuk assets

Masalah seperti ini cukup sering terjadi. Jika anda mendapati file CSS, file JS atau gambar tidak tampil, cek apakah alamat URLnya sudah sesuai atau belum.

7.8. Bootstrap CSS Framework

Agar hasil latihan kita tampak lebih menarik, saya memutuskan untuk memakai framework CSS Bootstrap dalam merancang tampilan front-end sepanjang buku ini. Bootstrap sepenuhnya opsional dan tidak berhubungan langsung dengan materi Laravel.

Bagi yang sudah pernah belajar Bootstrap, tentu bisa merasakan betapa mudahnya merancang tampilan yang menarik (serta responsive) menggunakan Bootstrap. Kita tidak perlu membuat semua kode CSS dari nol.

Dan sebenarnya, nanti ada satu materi khusus yang juga mengharuskan kita memakai Bootstrap, yakni **authentication**. Authentication adalah materi dimana kita akan mempelajari cara membuat form login dan form register bawaan Laravel, yang tampilannya (secara default)

menggunakan Bootstrap.

Proses mengintegrasikan Bootstrap ke dalam Laravel juga tidak sulit, cukup menambah link ke file CSS dan JavaScript milik Bootstrap saja.

Cara paling *simple*, kita bisa memakai alamat file CDN Bootstrap 5.1 yang ada di halaman [dokumentasi Bootstrap](#).

Untuk file CSS alamat CDNnya ada di:

<https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css>

Dan untuk file JavaScript bisa diakses dari:

<https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js>

CDN (Content Delivery Network) adalah sebutan untuk kumpulan server yang bisa dipakai untuk menampung file web agar bisa diakses dengan lebih cepat. Selain menyediakan layanan berbayar, umumnya CDN juga menyediakan link ke file library populer yang bisa dipakai secara gratis, termasuk file-file Bootstrap.

Berikut hasil modifikasi file view `mahasiswa.blade.php` dengan penambahan link ke file CDN Bootstrap:

`resources/views/universitas/mahasiswa.blade.php`

```

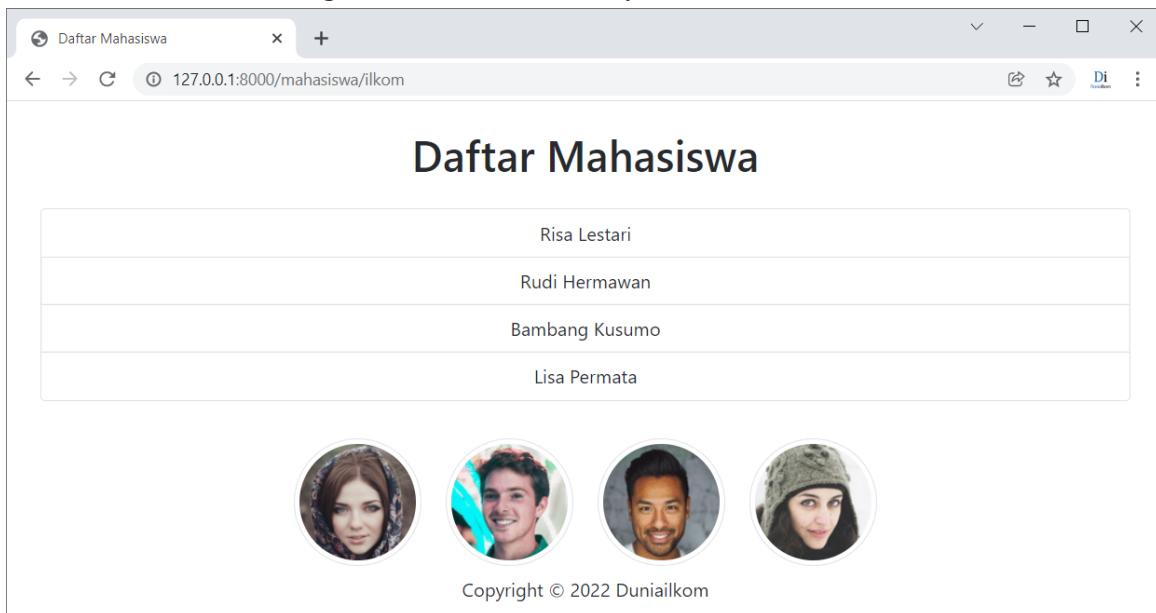
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Daftar Mahasiswa</title>
8      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/
dist/css/bootstrap.min.css">
9  </head>
10 <body>
11     <div class="container text-center mt-4">
12         <h1>Daftar Mahasiswa</h1>
13         <ol class="list-group my-4">
14             <?php
15                 foreach ($mahasiswa as $nama) {
16                     echo "<li class=\"list-group-item\"> $nama </li>";
17                 }
18             ?>
19         </ol>
20
21         <div>
22             
23             
24             
```

View

```
25      
26  </div>
27
28  <div>
29      Copyright © <?php echo date("Y"); ?> DuniaIlkom
30  </div>
31 </div>
32 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/
  bootstrap.bundle.min.js"></script>
33 </body>
34 </html>
```

Tambahan link ke file Bootstrap ada di baris 8 untuk file CSS, dan baris 32 untuk file JS. Alamat CDN Bootstrap memang cukup panjang sehingga dalam tampilan di atas terpaksa saya tulis "melimpah" ke baris di bawahnya.

Selain perubahan ini, di dalam kode HTML juga terdapat tambahan sedikit class bawaan Bootstrap, diantaranya untuk membuat tag `<div>` dengan class **container** di baris 11, menambah class `.list-group` ke dalam tag ``, serta menambah class `.rounded-circle` dan `.img-thumbnail` ke dalam tag ``. Berikut hasilnya:



Gambar: view mahasiswa dengan tambahan design Bootstrap

Terlihat lebih menarik dibandingkan tanpa Bootstrap.

Namun jika menggunakan CDN, kita harus selalu terkoneksi ke internet karena file Bootstrap tersebut berada di internet (tepatnya di server CDN).

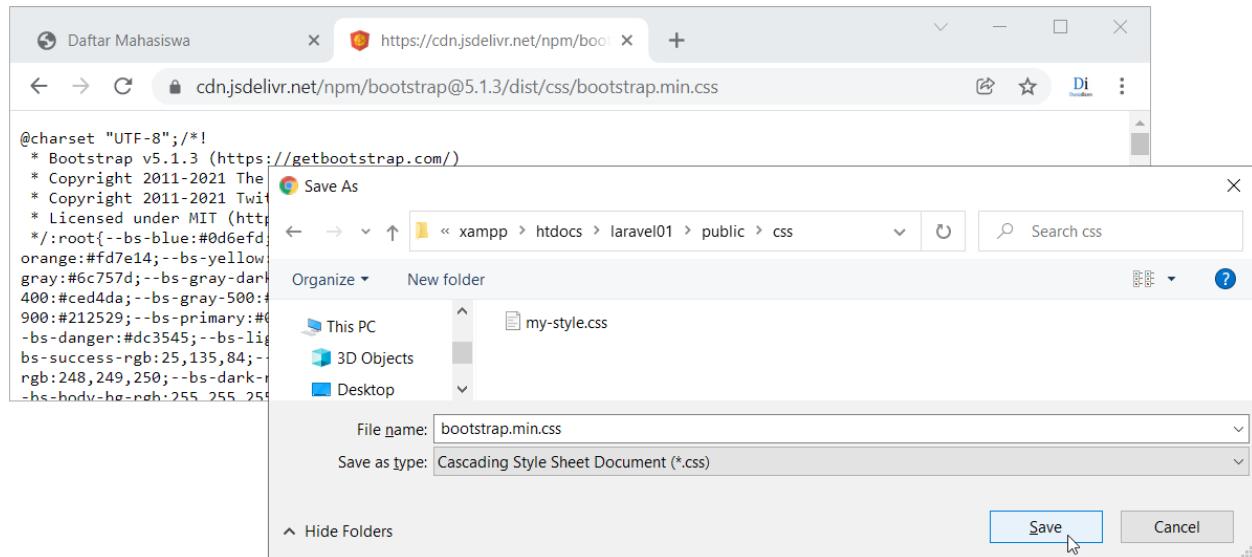
Bagaimana agar halaman ini bisa diakses secara offline? Tidak masalah, kita sudah mempelajari caranya, yakni download file-file Bootstrap, simpan ke folder **public**, kemudian akses menggunakan alamat relatif dari dalam view.

Pertama, kita harus kumpulkan dan simpan file-file Bootstrap ini. Cara yang paling cepat

View

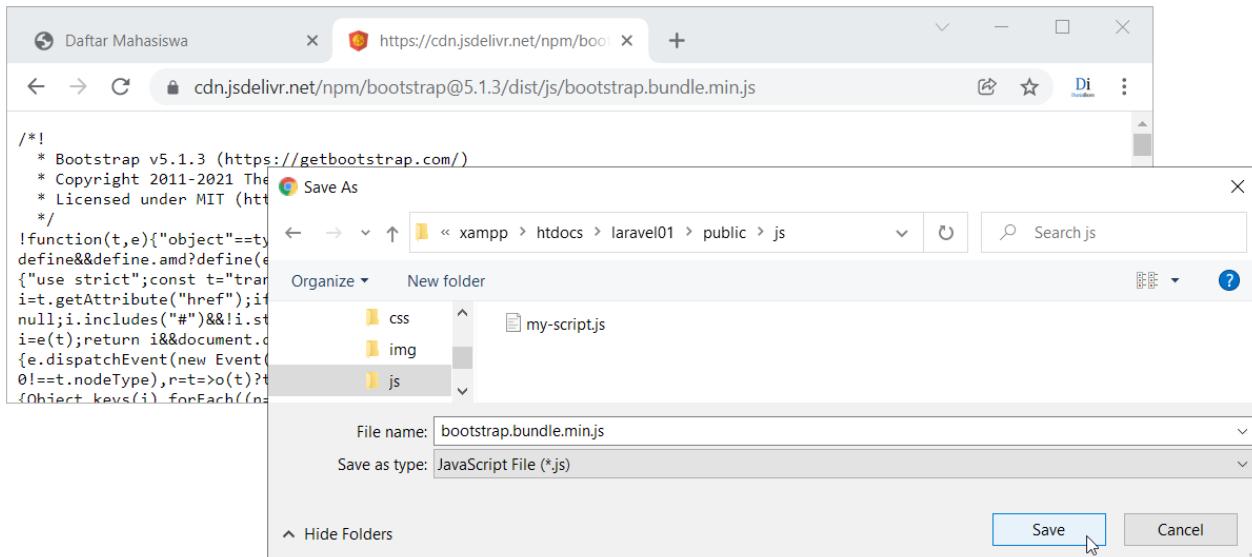
adalah dengan mengambil file dari alamat CDN tadi.

Untuk file CSS, buka alamat <https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css> di web browser, lalu **Save As** (atau tekan tombol **Ctrl + S**) untuk di simpan ke dalam folder public/css.



Gambar: Simpan file bootstrap.min.css ke dalam folder public/css

Lakukan hal yang sama dengan file JavaScript Bootstrap di <https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js>, namun kali ini simpan ke dalam folder public\js.



Gambar: Simpan file jquery-3.3.1.slim.min.js, popper.min.js dan bootstrap.min.js ke dalam folder public/css

Setelah semuanya selesai, maka di dalam folder public\css akan terdapat 2 file:

- ✓ **bootstrap.min.css** (file CSS Bootstrap)
- ✓ **my-style.css** (file CSS yang kita buat sebelumnya)

Dan di folder public\js juga terdapat 2 file:

- ✓ **bootstrap.bundle.min.js** (file JavaScript Bootstrap)
- ✓ **my-script.js** (file CSS yang kita buat sebelumnya)

Sip, sekarang kita bisa edit view mahasiswa agar mengambil file Bootstrap secara local, tidak lagi dari CDN:

resources/views/universitas/mahasiswa.blade.php

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Daftar Mahasiswa</title>
8      <link rel="stylesheet" href="/css/bootstrap.min.css">
9  </head>
10 <body>
11     <div class="container text-center mt-4">
12         <h1>Daftar Mahasiswa</h1>
13         <ol class="list-group my-4">
14             <?php
15                 foreach ($mahasiswa as $nama) {
16                     echo "<li class=\"list-group-item\"> $nama </li>";
17                 }
18             ?>
19         </ol>
20
21         <div>
22             
23             
24             
25             
26         </div>
27
28         <div>
29             Copyright © <?php echo date("Y"); ?> DuniaIlkom
30         </div>
31     </div>
32     <script src="/js/bootstrap.bundle.min.js"></script>
33 </body>
34 </html>
```

Silahkan akses kembali view mahasiswa di web browser, jika tampilannya masih sama seperti sebelumnya, berarti kita sudah sukses mengakses file Bootstrap secara offline.

Karena Bootstrap akan sering di pakai, silahkan copy semua file Bootstrap ke folder lain di luar Laravel. Tujuannya, ketika kita menginstall ulang file Laravel, tinggal copy kembali file-file ini ke dalam folder **public**.

Sebenarnya nanti terdapat cara yang lebih modern untuk mengintegrasikan Bootstrap ke dalam Laravel, yakni menggunakan **Laravel UI** dan **Laravel Mix**. Kita akan bahas keduanya dalam bab terpisah. Untuk sementara waktu, silahkan pakai cara manual seperti ini terlebih dahulu.

Peranan view sangat vital sebagai penentu tampilan akhir dari web, yakni hasil yang di lihat user di web browser.

Sepanjang bab ini kita telah membahas dasar penggunaan view Laravel, terutama cara pengiriman data dari route (dan juga nantinya, dari controller), membahas file assets, serta menginput file Bootstrap ke dalam View.

View juga masih punya fitur tambahan fitur yang tidak kalah menarik, yakni **blade template engine**. Inilah yang akan kita bahas dalam bab selanjutnya.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

8. Blade Template Engine

Bab kali ini masih berhubungan dengan view, dimana kita akan membahas **blade**, sebuah template engine bawaan Laravel. Blade sangat powerful dan akan memudahkan dalam memproses tampilan front-end Laravel.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder baru, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Alternatif lain, jika anda memiliki file asli Laravel 9 bisa juga di copy atau di rename menjadi folder laravel01.

Setelah itu input juga file Bootstrap (folder **css** dan folder **js**) ke dalam folder **public**, karena kita akan banyak menggunakan Bootstrap dalam bab ini.

8.1. Pengertian Blade

Blade adalah *template engine* bawaan Laravel. Secara sederhana, template engine berisi perintah tambahan untuk mempermudah pembuatan template. Template sendiri bisa disebut sebagai *kerangka dasar tampilan*.

Secara garis besar, terdapat 2 fungsi utama blade di dalam Laravel:

1. Mempersingkat penulisan perintah PHP.
2. Pemecahan file template (proses pembuatan layout).

Sebagai contoh, untuk menampilkan variabel di dalam view sebelumnya kita menggunakan perintah berikut:

```
<?php echo $nama; ?>
```

Dengan blade, bisa ditulis menjadi:

```
{{ $nama }}
```

Terlihat kodennya menjadi lebih rapi dan lebih singkat. Selain itu masih banyak fitur dari blade

yang akan kita pelajari secara bertahap.

8.2. Menampilkan Data

Kegunaan paling dasar dari **blade** adalah untuk mempermudah proses menampilkan data di dalam view.

Seperti yang sudah kita praktekkan sebelumnya, data ini bisa berasal dari route atau sumber lain (seperti controller). Ketika sampai ke view, data tersebut bisa ditampilkan dengan perintah echo PHP.

Mari lihat kembali contoh prakteknya, silahkan buat route dengan kode berikut:

routes/web.php

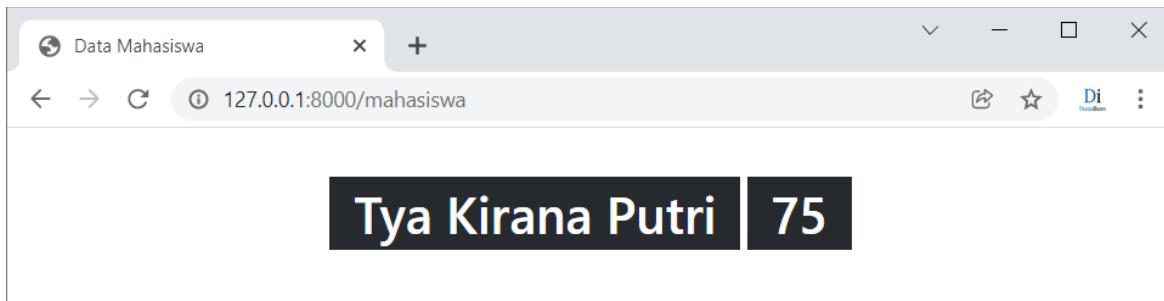
```
1 Route::get('/mahasiswa', function () {
2     $nama = 'Tya Kirana Putri';
3     $nilai = 75;
4     return view('mahasiswa', compact('nama', 'nilai'));
5 });
```

Saya yakin anda sudah bisa memahami route ini, dimana ketika alamat `http://localhost:8000/mahasiswa` diakses, view 'mahasiswa' akan dijalankan serta mengirim data `$nama` dan `$nilai` ke dalam view tersebut.

Berikut kode program untuk file `mahasiswa.blade.php`:

resources/views/mahasiswa.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Data Mahasiswa</title>
9 </head>
10 <body>
11     <div class="container text-center mt-3 pt-3 bg-white">
12         <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
13             <?php echo $nama ?>
14         </h1>
15         <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
16             <?php echo $nilai ?>
17         </h1>
18     </div>
19 </body>
20 </html>
```



Gambar: Tampilan view mahasiswa

Dalam view ini terdapat pemanggilan file CSS external /css/bootstrap.min.css (baris 7). Terkait file Bootstrap ini sudah kita bahas di akhir bab sebelumnya.

Sepanjang sisa buku saya juga akan terus menggunakan Bootstrap. Selain agar tampilan web tidak monoton, ini juga sebagai latihan penerapan Bootstrap terutama bagi yang memutuskan belajar **full-stack web development** (paham front-end dan back-end).

Ini semata-mata hanya untuk mempercantik tampilan web. Jika anda belum mempelajari Bootstrap, boleh diabaikan saja tambahan class CSS yang ada.

Fokus utama kita ada di baris 13 dan 16, yakni perintah echo PHP untuk menampilkan variabel \$nama dan \$nilai yang berasal dari route. Tidak ada hal yang baru di sini.

Menggunakan blade, bagian <body> dari view bisa ditulis sebagai berikut:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 py-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
3     {{$nama}}
4   </h1>
5   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
6     {{$nilai}}
7   </h1>
8 </div>
```

Hasilnya akan sama seperti sebelumnya. Penulisan seperti ini membuat kode program di dalam view menjadi lebih singkat dan lebih rapi. Secara internal, perintah {{\$nama}} akan diproses oleh Laravel sebagai <?php echo \$nama ?>.

Kita juga bisa menambah spasi sebelum dan sesudah tanda kurung kurawal:

resources/views/mahasiswa.blade.php

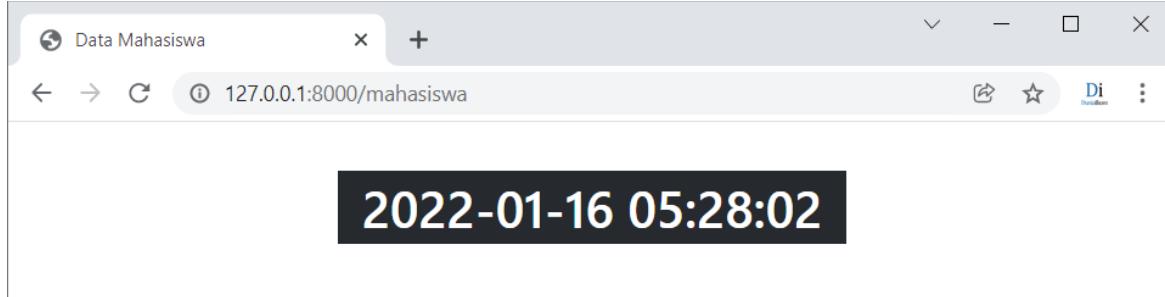
```
1 <div class="container text-center mt-3 py-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
3     {{ $nama }}
4   </h1>
5   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
6     {{ $nilai }}
```

```
7   </h1>
1   </div>
```

Selain menampilkan variabel, perintah PHP lain yang menghasilkan nilai juga bisa ditulis ke dalam tanda {{ dan }}, termasuk pemanggilan function seperti contoh berikut:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 py-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
3     {{ date(now()) }}
4   </h1>
5 </div>
```



Gambar: Tampilan hasil fungsi date() PHP

Di sini saya menjalankan fungsi `date(now())` bawaan PHP dalam tanda kurung kurawal blade. Ini sama artinya dengan perintah `<?php echo date(now()) ?>`.

Tanda kurung blade {{ dan }} sebenarnya tidak hanya menggantikan perintah echo saja, tapi juga menjalankan function `htmlspecialchars()` sebelum menampilkan data tersebut.

Function `htmlspecialchars()` merupakan fungsi bawaan PHP yang dipakai untuk men-filter data dari kode-kode aneh seperti tag HTML atau kode JavaScript. Tujuannya untuk mencegah serangan **XSS** (*cross-site scripting*), yakni teknik menyisipkan kode JavaScript ke dalam web kita.

Sebagai percobaan, saya akan buat route yang mengirim data berisi kode HTML:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2   $nama = '<u>Tya Kirana Putri</u>';
3   $nilai = 75;
4   return view('mahasiswa', compact('nama', 'nilai'));
5 });
```

Perhatikan variabel `$nama` berisi string '`<u>Tya Kirana Putri</u>`', dan berikut hasilnya:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 py-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
3     {{ $nama }}
```

```
4   </h1>
5   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
6     {{ $nilai }}
7   </h1>
8 </div>
```



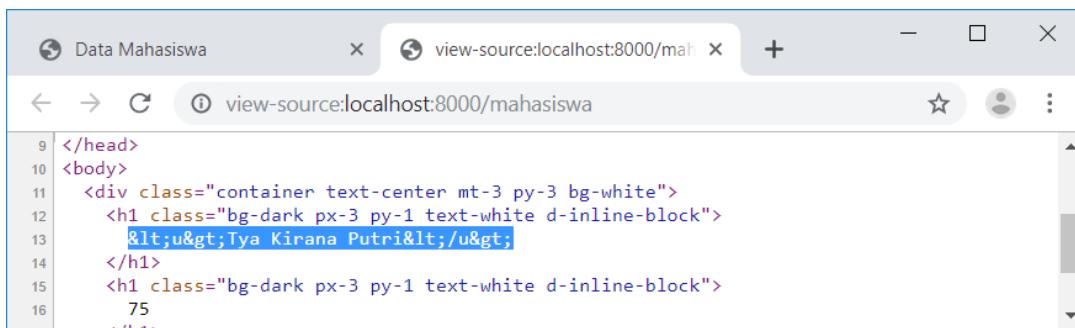
Gambar: Tag HTML tidak di proses oleh Blade

Terlihat tag `<u>` tampil "apa adanya", tidak dianggap sebagai bagian dari tag HTML. Di dalam HTML, tag `<u>` sendiri dipakai untuk membuat garis bawah (*underline*).

Apa yang akan dijalankan oleh blade adalah sebagai berikut:

```
<?php
echo htmlspecialchars($nama);
?>
```

Jika anda melihat source code tampilan dari web browser, maka teks `<u>` ini sudah di-*encode* menjadi HTML entity: `<u>`;



Gambar: Tag `<u>` di encode menjadi HTML entity

Namun bagaimana jika yang kita inginkan adalah agar tag `<u>` tersebut diproses sebagai bagian dari kode HTML? Atau dengan kata lain supaya data tidak masuk ke fungsi `htmlspecialchars()`?

Blade menyediakan cara penulisan alternatif, yakni dengan tanda kurung `{!!` dan `!!}`. Berikut contoh penggunaannya:

`resources/views/mahasiswa.blade.php`

```
1 <div class="container text-center mt-3 py-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
```

```
3      {!! $nama !!}
4  </h1>
5  <h1 class="bg-dark px-3 py-1 text-white d-inline-block">
6    {!! $nilai !!}
7  </h1>
8 </div>
```



Gambar: Tag <u> di proses oleh Blade sebagai bagian dari HTML

Sekarang, tanda <u> yang dikirim dari route akan di proses sebagai bagian dari kode HTML. Akan tetapi harap hati-hati menggunakan {!! dan !!}, karena data yang tampil tidak di proteksi dengan function `htmlspecialchars()`.

8.3. Kondisi If Else

Selain menampilkan data, blade juga menyediakan penulisan singkat untuk struktur kondisi PHP lain seperti **if else**, **switch** dan juga perulangan. Kita akan bahas struktur if else terlebih dahulu.

Sebagai bahan praktik, untuk route masih sama seperti sebelumnya:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2   $nama = 'Tya Kirana Putri';
3   $nilai = 75;
4   return view('mahasiswa', compact('nama', 'nilai'));
5 });
```

Namun di dalam view, saya ingin membuat sebuah kondisi if, yakni jika \$nilai mahasiswa antara 0 – 49, tampilkan pesan "Maaf, anda tidak lulus". Jika nilainya antara 50 – 100, maka tampilkan pesan "Selamat, anda lulus".

Kita bisa membuat logika ini menggunakan kondisi if PHP biasa:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nama }}</h1>
3   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nilai }}</h1>
4   <br>
5   <div class="alert alert-secondary d-inline-block">
```

```
6 <?php
7 if (($nilai >= 0) and ($nilai < 50))
8 {
9     echo "Maaf, anda tidak lulus";
10 }
11 else if (($nilai >= 50) and ($nilai <= 100))
12 {
13     echo "Selamat, anda lulus";
14 }
15 ?>
16 </div>
17 </div>
```



Gambar: Hasil kondisi if dari nilai

Di baris 2 – 3 saya menampilkan isi variabel \$nama dan \$nilai menggunakan blade.

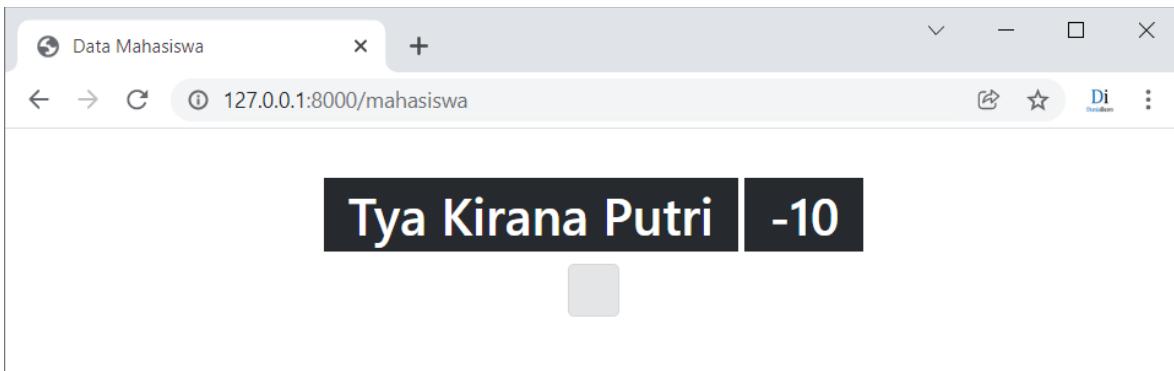
Kemudian di baris 6 – 15 terdapat kondisi if PHP, yakni jika variabel \$nilai berisi angka lebih atau sama dengan 0 DAN kurang dari 50, jalankan perintah echo "Maaf, anda tidak lulus". Sedangkan jika variabel \$nilai berisi angka lebih atau sama dengan 50 DAN kurang atau sama dengan 100, jalankan perintah echo "Selamat, anda lulus".

Dari route, variabel \$nilai berisi angka 75 sehingga hasil yang tampil adalah "Selamat, anda lulus". Anda bisa uji logika di atas dengan mengirim angka lain, misalnya 30, 10, atau 99.

Tapi mari kita test dengan mengirim angka -10:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2     $nama = 'Tya Kirana Putri';
3     $nilai = -10;
4     return view('mahasiswa', compact('nama', 'nilai'));
5});
```



Gambar: Hasil tampilan ketika dikirim nilai -10

Logika program kita sudah benar, ketika variabel \$nilai berisi angka -10, tidak akan tampil pesan apapun. Tapi kenapa ada kotak kosong berwarna abu-abu?

Ini berasal dari cara penulisan kode HTML, dimana tag <div> akan selalu tampil meskipun tidak berisi teks apapun:

```
<div class="alert alert-secondary d-inline-block">
    //... teks kosong disini
</div>
```

Agar tag <div> hanya tampil pada saat nilai berisi angka 0 - 100 saja, kita bisa pindahkan penulisan tag tersebut ke dalam blok if:

resources/views/mahasiswa.blade.php

```
1  <div class="container text-center mt-3 pt-3 bg-white">
2      <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nama }}</h1>
3      <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nilai }}</h1>
4      <br>
5      <?php
6          if (($nilai >= 0) and ($nilai < 50))
7          {
8              ?>
9              <div class="alert alert-danger d-inline-block">
10                 Maaf, anda tidak lulus
11             </div>
12         <?php
13         }
14         else if (($nilai >= 50) and ($nilai <= 100))
15         {
16             ?>
17             <div class="alert alert-success d-inline-block">
18                 Selamat, anda lulus
19             </div>
20         <?php
21         }
22     ?>
23 </div>
```

Apabila anda sudah sering membuat project atau studi kasus menggunakan PHP, tentu tidak asing dengan cara penulisan seperti ini. Kode HTML di baris 9 – 11 hanya akan dijalankan jika kondisi if di baris 6 bernilai true. Begitu juga dengan kode HTML di baris 17 – 19 hanya akan di eksekusi jika kondisi if di baris 13 bernilai true.

Dengan penulisan seperti ini, tag `<div>` tidak akan tampil jika variabel `$nilai` berisi angka selain 0 – 100. Selain itu saya juga bisa mengubah class alert Bootstrap untuk setiap kondisi, yakni class `.alert-danger` jika si mahasiswa tidak lulus, dan `.alert-success` jika lulus.

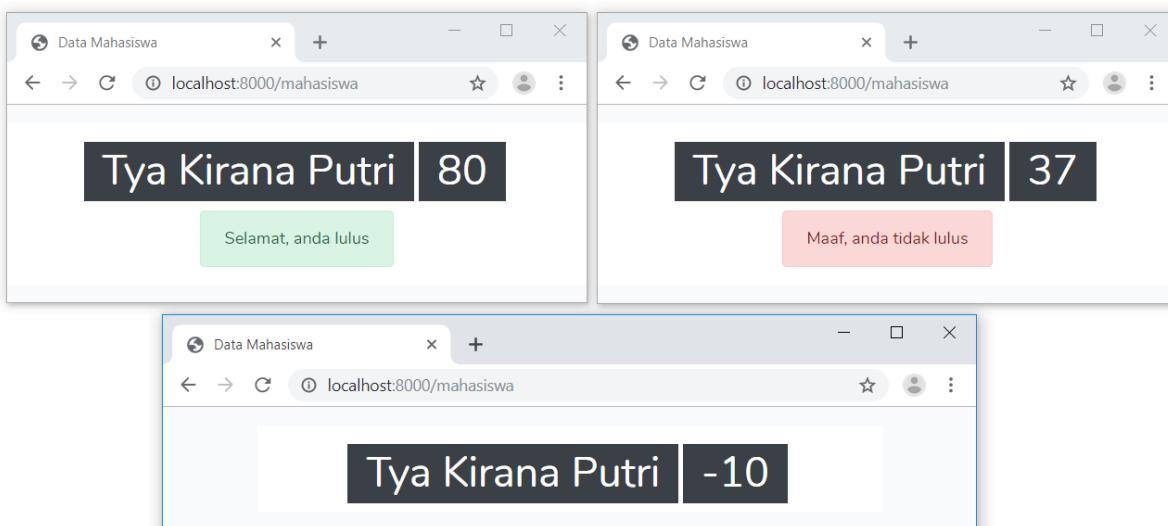
Bagi sebagian programmer, penulisan campuran PHP dan HTML ini terlihat kurang rapi. Selain itu juga susah di telusuri, misalnya kita harus teliti mencari tanda penutup blok if yang ternyata ada di baris 21.

Oleh karena itulah blade menyediakan cara penulisan alternatif, yakni dengan kode `@if` untuk memulai blok if, `@elseif` untuk blok else if, serta `@endif` untuk menutup blok if.

Berikut hasil konversi view sebelumnya menggunakan blade:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nama }}</h1>
3   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nilai }}</h1>
4   <br>
5   @if (($nilai >= 0) and ($nilai < 50))
6     <div class="alert alert-danger d-inline-block">
7       Maaf, anda tidak lulus
8     </div>
9   @elseif (($nilai >= 50) and ($nilai <= 100))
10    <div class="alert alert-success d-inline-block">
11      Selamat, anda lulus
12    </div>
13  @endif
14 </div>
```



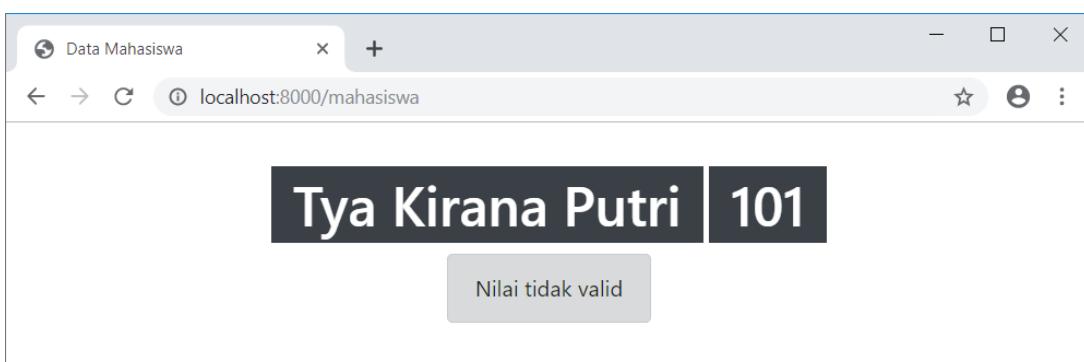
Gambar: Hasil tampilan ketika dikirim nilai 80, 37 dan -10

Menggunakan format blade, kode program menjadi lebih rapi dan lebih mudah dibaca. Kita juga tidak perlu menulis tag pembuka dan penutup php <?php ?>.

Tantangan berikutnya, bagaimana jika variabel \$nilai berada di luar 0 – 100, tampilkan pesan "Nilai tidak valid". Untuk hal ini, kita tinggal tambah satu lagi blok if else:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nama }}</h1>
3   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nilai }}</h1>
4   <br>
5   @if (($nilai >= 0) and ($nilai < 50))
6     <div class="alert alert-danger d-inline-block">Maaf, anda tidak lulus</div>
7   @elseif (($nilai >= 50) and ($nilai <= 100))
8     <div class="alert alert-success d-inline-block">Selamat, anda lulus</div>
9   @else
10    <div class="alert alert-dark d-inline-block">Nilai tidak valid</div>
11  @endif
12 </div>
```



Gambar: Hasil tampilan ketika dikirim nilai 101

Blade hanya mempersingkat penulisan kode program. Jika anda merasa "lebih pas" dan sudah terbiasa menggunakan struktur if else bawaan PHP, itu juga tidak masalah. Tapi seperti yang terlihat, kode program kita terlihat lebih rapi jika menggunakan blade.

8.4. Kondisi Switch

Selain struktur kondisi if else, blade menyediakan alternatif cara penulisan struktur kondisi **switch**.

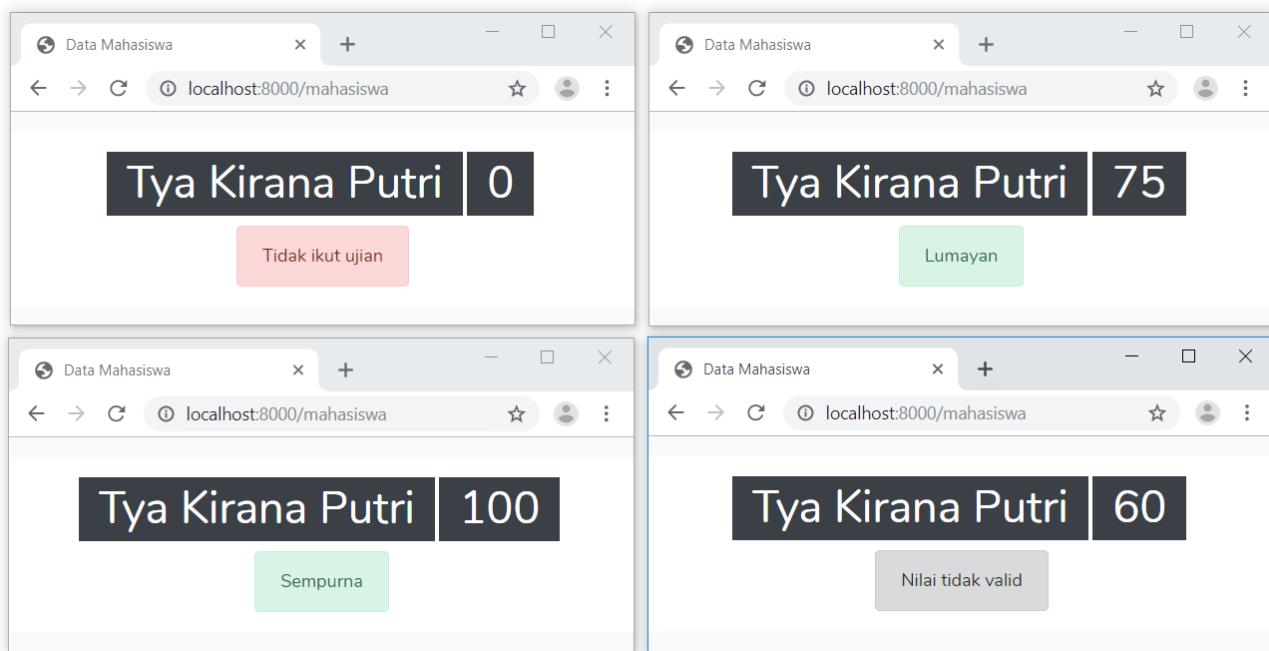
Untuk bahan praktik, saya ingin membuat 4 kondisi case:

- ◆ Jika \$angka bernilai 0, tampilkan teks "Tidak ikut ujian"
- ◆ Jika \$angka bernilai 75, tampilkan teks "Lumayan"
- ◆ Jika \$angka bernilai 100, tampilkan teks "Sempurna"
- ◆ Jika \$angka bernilai selain 3 angka di atas, tampilkan teks "Nilai tidak valid"

Berikut kode program view untuk keperluan ini:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nama }}</h1>
3   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{ $nilai }}</h1>
4   <br>
5   @switch($nilai)
6     @case(0)
7       <div class="alert alert-danger d-inline-block">Tidak ikut ujian</div>
8     @break
9     @case(75)
10      <div class="alert alert-success d-inline-block">Lumayan</div>
11    @break
12    @case(100)
13      <div class="alert alert-success d-inline-block">Sempurna</div>
14    @break
15    @default
16      <div class="alert alert-dark d-inline-block">Nilai tidak valid</div>
17 @endswitch
18 </div>
```



Gambar: Hasil tampilan nilai dengan 4 kondisi

Contoh ini memang kurang pas karena jika menggunakan switch-case, kita hanya bisa memeriksa kondisi dengan nilai yang sama persis, seperti 0, 75, atau 100. Struktur switch tidak bisa dipakai untuk memeriksa nilai jangkauan seperti 0 – 50 sebagaimana layaknya if else.

Di sini terdapat penggunaan 5 perintah blade, yakni: @switch, @case, @break, @default dan @endswitch. Saya yakin anda bisa paham maksud setiap perintah ini karena sangat mirip dengan perintah yang sama di dalam PHP native.

8.5. Perulangan For

Blade juga menyediakan cara penulisan singkat untuk struktur perulangan for. Berikut contoh penggunaannya:

resources/views/mahasiswa.blade.php

```

1 <div class="container text-center mt-3 pt-3 bg-white">
2   @for ($i = 0; $i < 5; $i++)
3     <div class="alert alert-info d-inline-block">
4       Laravel
5     </div>
6   @endfor
7 </div>
```



Gambar: Teks 'Laravel' di ulang sebanyak 5 kali

Kode view di atas akan menampilkan teks 'Laravel' sebanyak 5 kali yang berasal dari perintah `@for($i = 0; $i < 5; $i++)`. Kembali, ini sangat mirip seperti perulangan for PHP.

Selain itu kita juga bisa mengakses variabel counter `$i` seperti halnya di PHP native:

resources/views/mahasiswa.blade.php

```

1 <div class="container text-center mt-3 pt-3 bg-white">
2   @for ($i = 0; $i < 5; $i++)
3     <div class="alert alert-info d-inline-block">
4       {{ $i }}
5     </div>
6   @endfor
7 </div>
```



Gambar: Perulangan for untuk menampilkan angka

Untuk menampilkan variabel counter `$i` di baris 4, saya juga menggunakan penulisan kurung

kurawal blade.

8.6. Perulangan While

Blade menyediakan perintah @while dan @endwhile untuk membuat perulangan while. Akan tetapi kita tetap butuh kode PHP native untuk menaikkan nilai variabel counter seperti contoh berikut:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <?php $i = 0; ?>
3   @while($i < 5)
4     <div class="alert alert-info d-inline-block">
5       {{ $i }}
6     </div>
7     <?php $i++ ?>
8   @endwhile
9 </div>
```

Hasil dari kode ini sama seperti contoh pada perulangan for, yakni angka yang menaik dari 0, 1, 2, 3 dan 4.

Di baris 2 terdapat kode PHP untuk memberikan nilai awal kepada \$i . Variabel \$i ini berfungsi sebagai variabel counter. Pada prakteknya, nilai \$i bisa saja berasal dari luar view, seperti dikirim dari route.

Kemudian di baris 3 terdapat perintah @while yang akan melakukan perulangan selama kondisi \$i < 5 terpenuhi (bernilai true).

Karena ini adalah perulangan while, maka harus ada kode program untuk memodifikasi nilai variabel counter \$i. Untuk itu saya menjalankan operator increment di baris 7. Proses increment harus dilakukan dari PHP karena blade tidak menyediakannya.

8.7. Perulangan Foreach

Dibandingkan for dan while, perulangan **foreach** lebih banyak kita pakai dalam Laravel. Alasannya karena nilai yang dikirim dari route (dan nantinya dari controller), lebih banyak berbentuk array. Perulangan foreach sangat cocok untuk memproses array dibandingkan for dan while.

Sebagai contoh praktek, saya akan mengirim sebuah array dari route:

routes/web.php

```
1 Route::get('/mahasiswa', function () {
2   $nama = 'Tya Kirana Putri';
3   $nilai = [80,64,30,76,95];
```

```
4
5     return view('mahasiswa',compact('nama','nilai'));
6 );
```

Sekarang variabel \$nilai berisi array dengan 5 element. Ketika sampai di route, array \$nilai bisa ditampilkan dengan kode berikut:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3   <br>
4   @foreach ($nilai as $val)
5     <div class="alert alert-info d-inline-block">
6       {{$val}}
7     </div>
8   @endforeach
9 </div>
```



Gambar: Menampilkan array dengan foreach blade

Kembali, format penulisan perintah foreach blade sangat mirip dengan versi PHP. Kita hanya perlu menulis @foreach untuk memulai perulangan dan @endforeach untuk mengakhiri perulangan. Tersedia juga format @foreach(\$nilai as \$key => \$val) jika ingin mengakses key dan nilai array sekaligus.

Agar lebih menarik lagi, saya akan gabung perulangan foreach dengan kondisi if. Idenya adalah, jika salah satu nilai berada antara 0 – 49, tampilkan dengan alert warna merah, yakni dengan tambahan class .alert-danger. Sedangkan jika nilai berada di antara 50 – 100, tampilkan dengan alert warna hijau, yakni class .alert-success bawaan Bootstrap.

Berikut kode programnya:

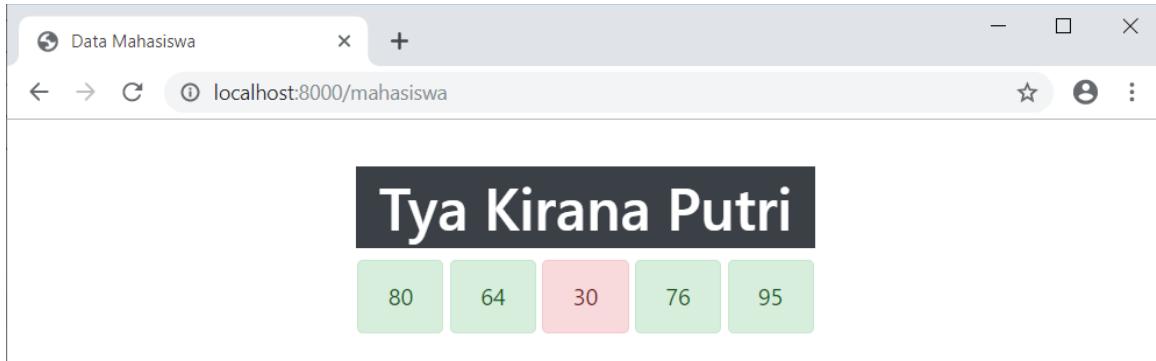
resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3   <br>
4   @foreach ($nilai as $val)
5     @if (($val >= 0) and ($val < 50))
6       <div class="alert alert-danger d-inline-block">
7         {{$val}}
```

```

8      </div>
9      @elseif (($val >= 50) and ($val <= 100))
10     <div class="alert alert-success d-inline-block">
11       {{$val}}
12     </div>
13   @endif
14 @endforeach
15 </div>

```



Gambar: Hasil gabungan foreach dan if blade

Sekarang di dalam blok @foreach terdapat 2 buah percabangan @if sesuai dengan syarat nilai. Hasilnya, angka 30 tampil dengan warna merah karena di eksekusi oleh blok if di baris 6 - 8.

Anda bisa coba mengubah element array dengan nilai lain. Jika diinput angka yang tidak valid seperti -10, nilai tersebut tidak akan tampil.

Exercise

Tambah satu lagi logika if ke dalam view mahasiswa, dimana jika route mengirim variabel \$nilai sebagai array kosong (tidak ada satu pun element), tampilkan teks "Tidak ada data..."

Answer

Kita perlu mencari cara untuk menentukan apakah \$nilai berisi sesuatu atau tidak. Pemeriksaan kondisi ini harus dilakukan sebelum perulangan foreach. Jika setelah pemeriksaan array \$nilai berisi sesuatu, maka jalankan perulangan foreach untuk menampilkan semua element. Namun jika \$nilai tidak berisi apa-apa, tampilkan teks "Tidak ada data...".

Berikut kode program yang bisa dipakai:

```

1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3   <br>
4   @if (count($nilai)>0)
5     @foreach ($nilai as $val)

```

```

6      @if (($val >= 0) and ($val < 50))
7          <div class="alert alert-danger d-inline-block">
8              {{$val}}
9          </div>
10     @elseif (($val >= 50) and ($val <= 100))
11         <div class="alert alert-success d-inline-block">
12             {{$val}}
13         </div>
14     @endif
15     @endforeach
16     @else
17         <div class="alert alert-dark d-inline-block">
18             Tidak ada data...
19         </div>
20     @endif
21 </div>
```

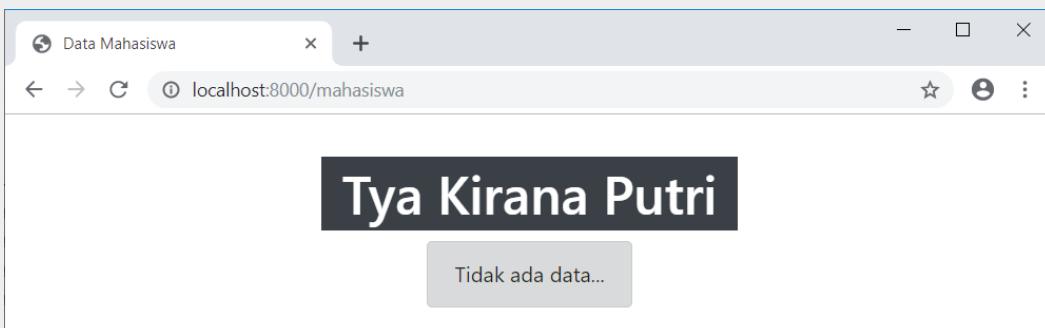
Di baris 4 terdapat tambahan kondisi if untuk memeriksa hasil `count($nilai)>0`. Fungsi `count()` sendiri merupakan function bawaan PHP untuk menampilkan jumlah element dari sebuah array.

Jika array `$nilai` berisi minimal 1 element, maka kondisi if akan bernilai **true** sehingga perulangan foreach di baris 5 – 15 akan dijalankan. Namun jika `count($nilai)` menghasilkan angka 0, yang artinya `$nilai` adalah sebuah string kosong, kondisi di baris 4 akan bernilai **false** sehingga kode program langsung lompat ke blok else di baris 17 – 19.

Berikut tampilan akhir ketika saya mengirim array `$nilai` sebagai array kosong:

```

1 Route::get('/mahasiswa', function () {
2     $nama = 'Tya Kirana Putri';
3     $nilai = [];
4     return view('mahasiswa', compact('nama', 'nilai'));
5 });
```



Gambar: Tampilan jika variabel \$nilai dikirim sebagai array kosong

Proses pemeriksaan array seperti ini sering kita lakukan untuk menghindari error, terutama jika nilai yang sampai di view di generate secara dinamis yang tidak pasti akan selalu berisi element.

Cara lain untuk menentukan apakah sebuah array berisi element atau tidak bisa juga

menggunakan perintah `@if(empty($nilai))`.

8.8. Perulangan Forelse

Forelse adalah perulangan khusus yang disediakan blade dengan struktur `@forelse`, `@empty` dan `@endforelse`. Ini sebenarnya gabungan dari proses pemeriksaan apakah sebuah array berisi nilai atau tidak, kurang lebih sama seperti exercise yang baru saja kita buat.

Berikut format dasar dari perulangan **forelse** blade:

```
1 @forelse ($nilai as $val)
2     // akan dijalankan jika $nilai berisi sesuatu
3 @empty
4     // akan dijalankan jika $nilai berupa array kosong
5 @endforelse
```

Berikut hasil konversi kode program exercise sebelumnya dengan perulangan forelse:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2     <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3     <br>
4     @forelse ($nilai as $val)
5         @if (($val >= 0) and ($val < 50))
6             <div class="alert alert-danger d-inline-block">
7                 {{$val}}
8             </div>
9         @elseif (($val >= 50) and ($val <= 100))
10            <div class="alert alert-success d-inline-block">
11                {{$val}}
12            </div>
13        @endif
14        @empty
15            <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
16        @endforelse
17    </div>
```

Blade secara khusus menyediakan perulangan forelse karena kita akan sangat sering memproses array di dalam view Laravel.

8.9. Perintah Continue dan Break

Di dalam PHP native, terdapat perintah **continue** dan **break** yang bisa dipakai untuk mengatur alur perulangan. Blade juga menyediakan perintah `@continue` dan `@break` sebagai alternatif penulisan.

Untuk contoh praktik, mari isi kembali isi variabel `$nilai` di dalam route:

Blade Template Engine

routes/web.php

```
1 Route::get('mahasiswa', function () {
2     $nama = 'Tya Kirana Putri';
3     $nilai = [80, 64, 30, 76, 95];
4
5     return view('mahasiswa', compact('nama', 'nilai'));
6 });
```

Dan berikut contoh penggunaan perintah @continue:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2     <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3     <br>
4     @foreach ($nilai as $val)
5         @if ($val < 50)
6             @continue
7         @endif
8         <div class="alert alert-success d-inline-block">
9             {{$val}}
10        </div>
11    @endforeach
12 </div>
```



Gambar: Hasil penggunaan perintah @continue

Di sini saya kembali menampilkan isi array `$nilai` menggunakan perulangan `foreach`, namun terdapat tambahan pemeriksaan kondisi di baris 5 – 7. Kondisi tersebut adalah, jika ditemukan angka yang kurang dari 50, maka jalankan perintah `@continue`.

Hasilnya, setiap kali `foreach` memproses angka yang kurang dari 50, perulangan langsung lompat ke iterasi selanjutnya.

Berikut contoh jika menggunakan perintah `@break`:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2     <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3     <br>
4     @foreach ($nilai as $val)
```

```
5  @if ($val < 50)
6      @break
7  @endif
8  <div class="alert alert-success d-inline-block">
9      {{$val}}
10     </div>
11 @endforeach
12 </div>
```



Gambar: Hasil penggunaan perintah @break

Terlihat hanya ada 2 nilai yang tampil, karena begitu perulangan memproses angka 30, perintah @break di baris 6 akan dijalankan dan perulangan foreach langsung berhenti.

8.10. Baris Komentar dan PHP mode

Untuk membuat baris komentar, blade menyediakan cara penulisan khusus, yakni dengan tanda pembuka {{-- dan penutup --}}. Berikut contoh penggunaannya:

resources/views/mahasiswa.blade.php

```
1  <div class="container text-center mt-3 pt-3 bg-white">
2      <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3      <br>
4      {{--
5      @foreach ($nilai as $val)
6          @if ($val < 50)
7              @break
8          @endif
9          <div class="alert alert-success d-inline-block">
10             {{$val}}
11            </div>
12        @endforeach
13    --}}
14 </div>
```

Dengan penulisan seperti ini, mulai dari baris 4 – 13 tidak akan diproses oleh Laravel.

Kita tidak bisa menggunakan kode komentar PHP biasa seperti // atau /* karena tidak sedang berada di PHP mode. Maksudnya, kode di atas tidak berada di dalam tag <?php dan ?>.

Menggunakan tag komentar HTML <!-- dan --> juga bisa, tapi kode akan terlihat di source

code web browser:

```
10 <body>
11   <div class="container text-center mt-3 pt-3 bg-white">
12     <h1 class="bg-dark px-3 py-1 text-white d-inline-block">Tya Kirana Putri</h1>
13     <br>
14     <!--
15       80
16     -->
17     <div class="alert alert-success d-inline-block">
18       64
19     </div>
20     <!--
21   </div>
22 </body>
```

Gambar: kode di dalam view terlihat jika menggunakan tag komentar HTML

Karena hal ini, sebaiknya gunakan tanda {{-- dan --}} jika ingin membuat baris komentar di dalam view.

Selain itu blade juga menyediakan perintah pengganti untuk tag <?php dan ?>, yakni dengan @php dan @endphp. Berikut contoh penggunaannya:

resources/views/mahasiswa.blade.php

```
1 <div class="container text-center mt-3 pt-3 bg-white">
2   <h1 class="bg-dark px-3 py-1 text-white d-inline-block">{{$nama}}</h1>
3   <br>
4   @php
5     var_dump($nilai);
6   @endphp
7 </div>
```



Gambar: Hasil penggunaan perintah @php dan @endphp

Di antara perintah @php dan @endphp kita bisa menulis perintah PHP seperti biasa.

8.11. Merancang Layout

Apa yang baru saja kita pelajari merupakan fungsi pertama dari blade, yakni mempersingkat penulisan perintah PHP. Selain itu blade juga memiliki fitur untuk mempermudah proses pembuatan layout/template.

Sebagaimana yang kita ketahui, di setiap halaman web terdapat banyak element yang terus berulang di setiap file, seperti bagian header, menu navigasi, sidebar dan footer. Teknik yang

umum dipakai adalah memecah bagian tersebut menjadi file terpisah, lalu disatukan kembali menggunakan fungsi `include()` bawaan PHP.

Cara itu juga bisa diterapkan dalam Laravel, yakni dengan memecah view menjadi beberapa file. Blade sendiri menyediakan berbagai perintah tambahan agar proses pembuatan layout menjadi lebih fleksibel.

Sebagai bahan praktik, saya akan buat 3 file view: `mahasiswa.blade.php`, `dosen.blade.php`, serta `gallery.blade.php`. Di ketiga halaman ini terdapat banyak komponen berulang yang akan kita pecah menggunakan blade.

Berikut route yang diperlukan:

`routes/web.php`

```
1 Route::get('mahasiswa', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                       "Lisa Permata"];
4     return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
5 });
6
7 Route::get('dosen', function () {
8     $arrDosen = ["Maya Fitrianti, M.M.", "Prof. Silvia Nst, M.Farm.",
9                  "Dr. Umar Agustinus", "Dr. Syahrial, M.Kom."];
10    return view('dosen')->with('dosen', $arrDosen);
11 });
12
13 Route::get('gallery', function () {
14     return view('gallery');
15 });
```

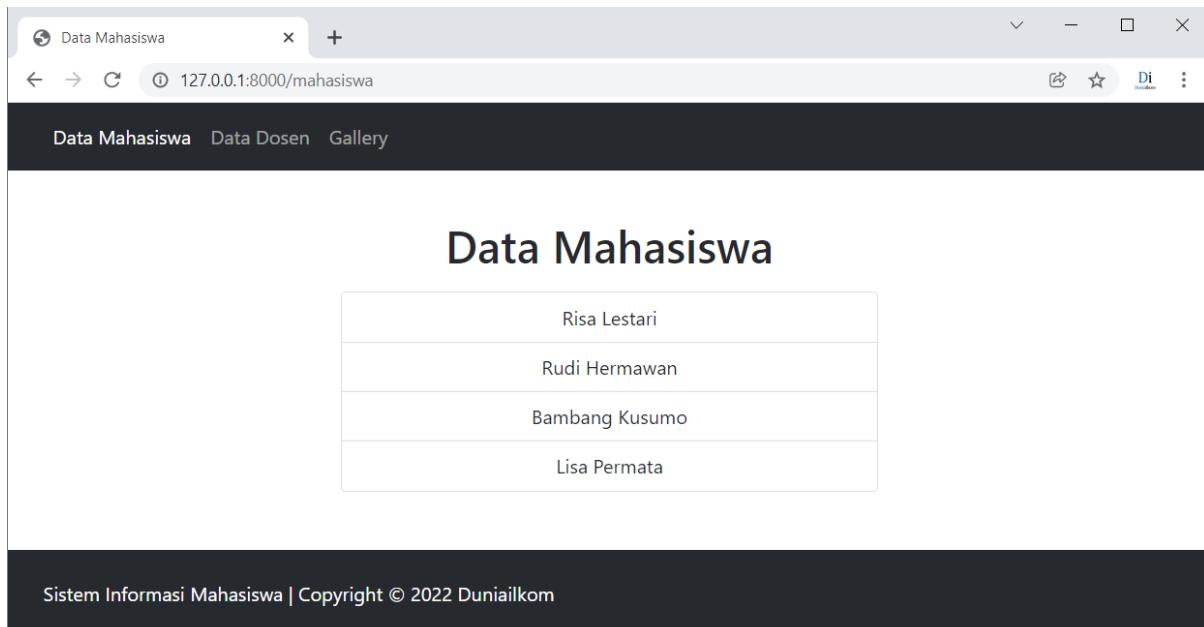
Untuk route `mahasiswa`, saya mengirim array `$mahasiswa` yang berisi 5 nama mahasiswa ke dalam view `mahasiswa`. Begitu pula untuk route `dosen` yang akan mengirim array `$dosen` ke dalam view `dosen`. Terakhir terdapat route `gallery` yang langsung menampilkan view `gallery`.

Berikut kode program untuk view `mahasiswa`:

`resources/views/mahasiswa.blade.php`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Data Mahasiswa</title>
9 </head>
10 <body>
11
12 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13     <div class="container">
14         <ul class="navbar-nav">
```

```
15     <li class="nav-item">
16         <a class="nav-link active" href="/mahasiswa">Data Mahasiswa</a>
17     </li>
18     <li class="nav-item">
19         <a class="nav-link" href="/dosen">Data Dosen</a>
20     </li>
21     <li class="nav-item">
22         <a class="nav-link" href="/gallery">Gallery</a>
23     </li>
24   </ul>
25 </div>
26 </nav>
27
28 <div class="container text-center mt-3 p-4 bg-white">
29   <h1 class="mb-3">Data Mahasiswa</h1>
30   <div class="row">
31     <div class="col-sm-8 col-md-6 m-auto">
32       <ol class="list-group">
33         @forelse ($mahasiswa as $val)
34           <li class="list-group-item">{{$val}}</li>
35         @empty
36           <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
37         @endforelse
38       </ol>
39     </div>
40   </div>
41 </div>
42
43 <footer class="bg-dark py-4 text-white mt-4">
44   <div class="container">
45     Sistem Informasi Mahasiswa | Copyright © {{ date("Y") }} DuniaIlkom
46   </div>
47 </footer>
48
49 </body>
50 </html>
```



Gambar: View mahasiswa

View mahasiswa ini sudah bisa mewakili sebuah halaman web utuh, karena sudah terdapat menu navigasi, konten dan footer.

Kode program untuk view mahasiswa bisa kita pecah menjadi beberapa bagian:

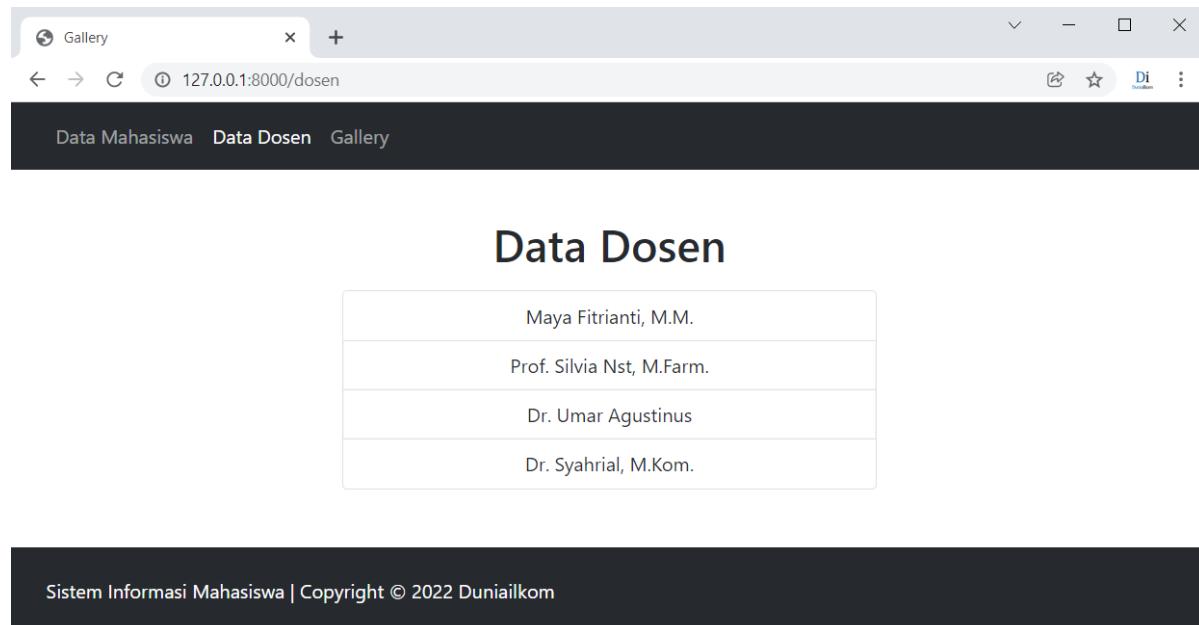
1. Bagian head HTML antara baris 1 – 9. Ini berisi kode pembuka HTML seperti `doctype`, tag `<head>`, `<meta>` dan `<title>`.
2. Bagian menu navigasi antara baris 12 – 26. Di sini saya menggunakan komponen navbar Bootstrap untuk membuat menu navigasi. Terdapat 3 link yang dipakai untuk mengakses halaman mahasiswa, dosen dan gallery.
3. Bagian konten antara baris 28 – 41. Inilah bagian utama dari view mahasiswa, dimana saya menggunakan komponen `list-group` Bootstrap yang di dalamnya terdapat perulangan `foreach` blade untuk menampilkan semua isi variabel `$mahasiswa`.
4. Bagian footer antara baris 43 – 47. Ini hanya sekedar "pemanis" agar tampilan halaman menjadi lebih menarik. Di dalam footer terdapat 1 baris teks `copyright` serta pengaksesan fungsi date PHP.

Lanjut, berikut kode program untuk view dosen:

`resources/views/dosen.blade.php`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <link href="/css/bootstrap.min.css" rel="stylesheet">
```

```
8     <title>Gallery</title>
9     </head>
10    <body>
11
12    <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13        <div class="container">
14            <ul class="navbar-nav">
15                <li class="nav-item">
16                    <a class="nav-link" href="/mahasiswa">Data Mahasiswa</a>
17                </li>
18                <li class="nav-item">
19                    <a class="nav-link active" href="/dosen">Data Dosen</a>
20                </li>
21                <li class="nav-item">
22                    <a class="nav-link" href="/gallery">Gallery</a>
23                </li>
24            </ul>
25        </div>
26    </nav>
27
28    <div class="container text-center mt-3 p-4 bg-white">
29        <h1 class="mb-3">Data Dosen</h1>
30        <div class="row">
31            <div class="col-sm-8 col-md-6 m-auto">
32                <ol class="list-group">
33                    @forelse ($dosen as $val)
34                        <li class="list-group-item">{{$val}}</li>
35                    @empty
36                        <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
37                    @endforelse
38                </ol>
39            </div>
40        </div>
41    </div>
42
43    <footer class="bg-dark py-4 text-white mt-4">
44        <div class="container">
45            Sistem Informasi Mahasiswa | Copyright © {{ date("Y") }} DuniaIlkom
46        </div>
47    </footer>
48
49    </body>
50 </html>
```

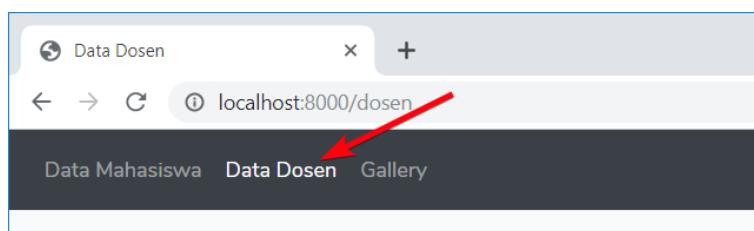


Gambar: View dosen

Tidak banyak perbedaan kode program antara view mahasiswa dengan view dosen:

1. Tag <title> di bagian head sekarang berisi teks "Data Dosen" (baris 8).
2. Class .active di dalam tag <a> menu navigasi pindah dari menu untuk halaman mahasiswa ke halaman dosen (baris 19).
3. Konten utama di baris 28 – 41 sekarang menampilkan isi array \$dosen.
4. Footer tidak ada perubahan.

Untuk point ke-2, class .active di dalam tag <a> berfungsi agar menu navigasi di halaman yang diakses berwarna lebih cerah. Ini biasa dipakai untuk menandakan posisi kita di dalam sebuah web:



Gambar: menu "Data Dosen" yang lebih cerah karena tambahan class .active

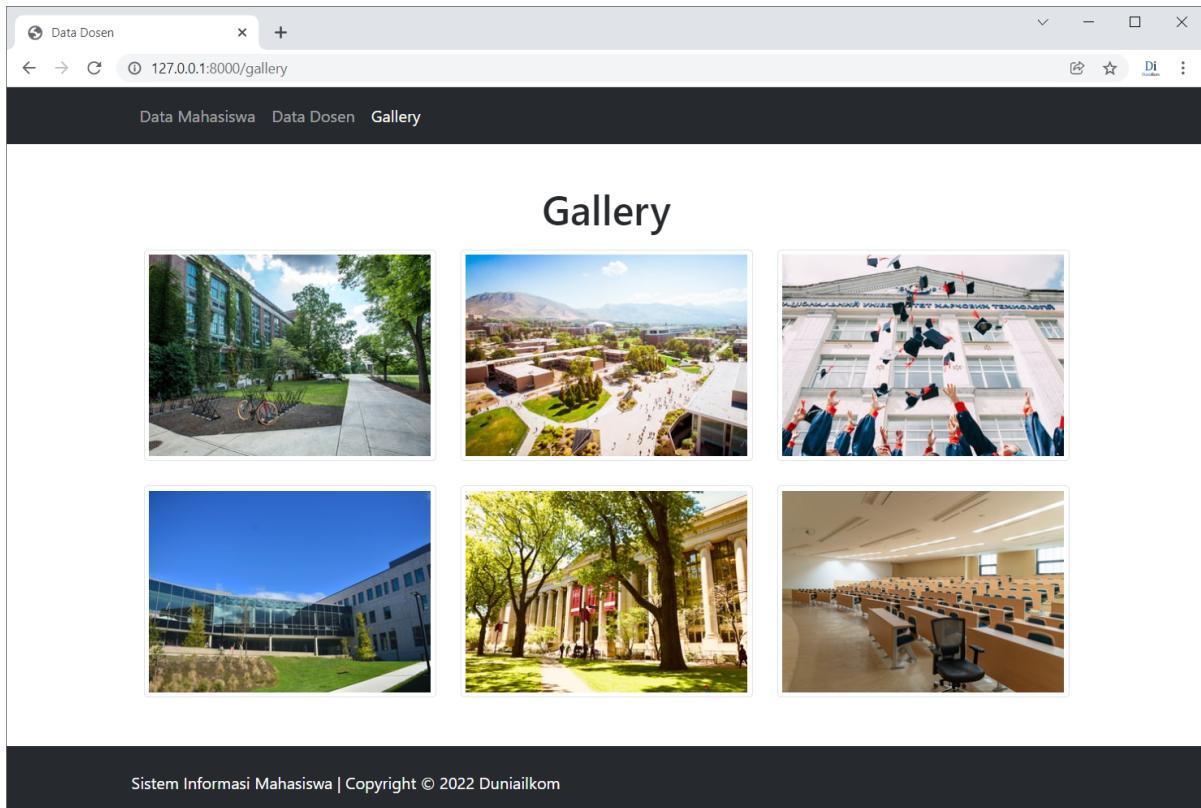
Terakhir, berikut kode program untuk view gallery:

resources/views/gallery.blade.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <link href="/css/bootstrap.min.css" rel="stylesheet">
8   <title>Data Dosen</title>
9   </head>
10  <body>
11
12  <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13    <div class="container">
14      <ul class="navbar-nav">
15        <li class="nav-item">
16          <a class="nav-link" href="/mahasiswa">Data Mahasiswa</a>
17        </li>
18        <li class="nav-item">
19          <a class="nav-link" href="/dosen">Data Dosen</a>
20        </li>
21        <li class="nav-item">
22          <a class="nav-link active" href="/gallery">Gallery</a>
23        </li>
24      </ul>
25    </div>
26  </nav>
27
28  <div class="container text-center mt-3 p-4 bg-white">
29    <h1 class="mb-3">Gallery</h1>
30    <div class="row">
31      <div class="col-4">
32        
34      </div>
35      <div class="col-4">
36        
38      </div>
39      <div class="col-4">
40        
42      </div>
43      <div class="col-4 mt-4">
44        
46      </div>
47      <div class="col-4 mt-4">
48        
50      </div>
51      <div class="col-4 mt-4">
52        
54      </div>
55    </div>
56  </div>
57
58  <footer class="bg-dark py-4 text-white mt-4">
59    <div class="container">
60      Sistem Informasi Mahasiswa | Copyright © {{ date("Y") }} DuniaIlkom
```

```
61 </div>
62 </footer>
63
64 </body>
65 </html>
```



Gambar: View gallery

Untuk view gallery, konten utama sekarang berupa gambar yang saya ambil dari web unsplash.com.

Karena gambar ini diambil langsung dari web online, tampilan di atas baru bisa terlihat jika komputer anda terhubung ke internet, atau bisa juga diganti dengan gambar offline lain. Mengenai cara menginput gambar ke view Laravel sudah kita bahas pada bab sebelumnya.

Selain konten, komponen lain untuk halaman gallery juga mirip seperti view mahasiswa dan dosen, dengan sedikit perubahan:

1. Tag `<title>` di bagian head sekarang berisi teks "Gallery" (baris 8).
2. Class `.active` di dalam tag `<a>` menu navigasi pindah ke tag `<a>` gallery (baris 22).
3. Konten utama di baris 28 – 56 menampilkan 6 gambar dengan tambahan class `.img-thumbnail` dan `.img-fluid` dari Bootstrap.
4. Footer tidak ada perubahan.

Ketiga view kita sudah tampil sempurna. Namun cara pembuatan seperti ini tidak efisien karena banyak komponen yang sama dan selalu berulang di setiap halaman.

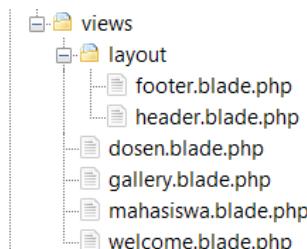
Misalnya jika ingin mengganti teks di bagian footer, maka terpaksa diubah satu per satu ke masing-masing view, padahal ketiganya memiliki kode yang sama persis. Terlebih jika web ini sudah terdiri dari puluhan atau ratusan halaman, maka akan sangat repot mengubahnya satu per satu. Kita perlu cara yang lebih efektif.

8.12. View Include

Cara pertama untuk mempermudah proses pembuatan layout atau template adalah memecah bagian yang berulang menjadi file terpisah, kemudian di satukan kembali di view yang membutuhkan. Blade menyediakan perintah `@include` untuk keperluan ini, yang cara penggunaannya mirip fungsi `include()` bawaan PHP.

Sebagai contoh praktek, saya akan pisahkan bagian header dan footer dari ketiga view menjadi komponen terpisah. File komponen ini di simpan sebagai `header.blade.php` dan `footer.blade.php` ke dalam folder **layout**.

Berikut struktur folder view yang akan kita rancang:



Gambar: Struktur folder view

File `header.blade.php` akan dipakai untuk menampung bagian head HTML serta menu navigasi. Berikut kode program untuk file ini:

`resources/views/layout/header.blade.php`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <link href="/css/bootstrap.min.css" rel="stylesheet">
8   <title>Data Mahasiswa</title>
9 </head>
10 <body>
11
12 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13   <div class="container">
```

```
14 <ul class="navbar-nav">
15   <li class="nav-item">
16     <a class="nav-link active" href="/mahasiswa">Data Mahasiswa</a>
17   </li>
18   <li class="nav-item">
19     <a class="nav-link" href="/dosen">Data Dosen</a>
20   </li>
21   <li class="nav-item">
22     <a class="nav-link" href="/gallery">Gallery</a>
23   </li>
24 </ul>
25 </div>
26 </nav>
```

Serta berikut kode untuk footer.blade.php:

resources/views/layout/footer.blade.php

```
1 <footer class="bg-dark py-4 text-white mt-4">
2   <div class="container">
3     Sistem Informasi Mahasiswa | Copyright © {{ date("Y") }} DuniaIlkom
4   </div>
5 </footer>
6
7 </body>
8 </html>
```

Dengan membuat file header.blade.php dan footer.blade.php, kita sudah memisahkan bagian header, menu navigasi dan footer. Sehingga di 3 view sebelumnya cukup menulis bagian konten saja.

Berikut modifikasi dari view mahasiswa:

resources/views/mahasiswa.blade.php

```
1 @include('layout.header')
2
3 <div class="container text-center mt-3 p-4 bg-white">
4   <h1 class="mb-3">Data Mahasiswa</h1>
5   <div class="row">
6     <div class="col-sm-8 col-md-6 m-auto">
7       <ol class="list-group">
8         @forelse ($mahasiswa as $val)
9           <li class="list-group-item">{{$val}}</li>
10        @empty
11          <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
12        @endforelse
13      </ol>
14    </div>
15  </div>
16 </div>
17
18 @include('layout.footer')
```

Perhatikan baris 1 dan 18, inilah cara penulisan perintah @include blade. Di baris 1 file yang di

include adalah 'layout.header'. Ini artinya saya ingin menginput file header.blade.php yang ada di dalam folder layout. Semoga anda masih ingat aturan bahwa tanda titik di nama view merupakan penanda folder.

Hal yang sama juga dipakai proses include file footer.blade.php yang ada di dalam folder layout, dimana perintah yang dipakai adalah @include('layout.footer').

Begitu juga untuk view dosen, dimana kita tinggal men-include bagian header dan footer saja:

resources/views/dosen.blade.php

```
1 @include('layout.header')
2
3 <div class="container text-center mt-3 p-4 bg-white">
4   <h1 class="mb-3">Data Dosen</h1>
5   <div class="row">
6     <div class="col-sm-8 col-md-6 m-auto">
7       <ol class="list-group">
8         @forelse ($dosen as $val)
9           <li class="list-group-item">{{$val}}</li>
10        @empty
11          <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
12        @endforelse
13        </ol>
14      </div>
15    </div>
16  </div>
17
18 @include('layout.footer')
```

Serta view gallery:

resources/views/gallery.dosen.php

```
1 @include('layout.header')
2
3 <div class="container text-center mt-3 p-4 bg-white">
4   <h1 class="mb-3">Gallery</h1>
5   <div class="row">
6     <div class="col-4">
7       
9     </div>
10    ...
11    ...
12    <div class="col-4 mt-4">
13      
15    </div>
16  </div>
17 </div>
18
19 @include('layout.footer')
```

Dengan pemisahan seperti ini, kode program kita lebih fleksibel. Jika ingin membuat halaman baru, cukup fokus ke konten utama saja karena bagian header dan footer sudah tersedia dan tinggal di include.

Note

Penting untuk dicatat bahwa penulisan alamat file include adalah **relatif ke root folder "view"**, bukan ke masing-masing file blade. Maksudnya untuk merujuk ke file `header.blade.php` yang ada di dalam folder **layout**, tetap ditulis sebagai `@include('layout.header')`, meskipun file yang mengakses ada di folder yang sama.

Sebagai contoh, jika di dalam folder `layout` saya membuat sebuah file `test.blade.php` dan ingin men-include file `header.blade.php`, perintahnya tetap `@include('layout.header')`, bukan `@include('header')` meskipun kedua file ada di folder yang sama.

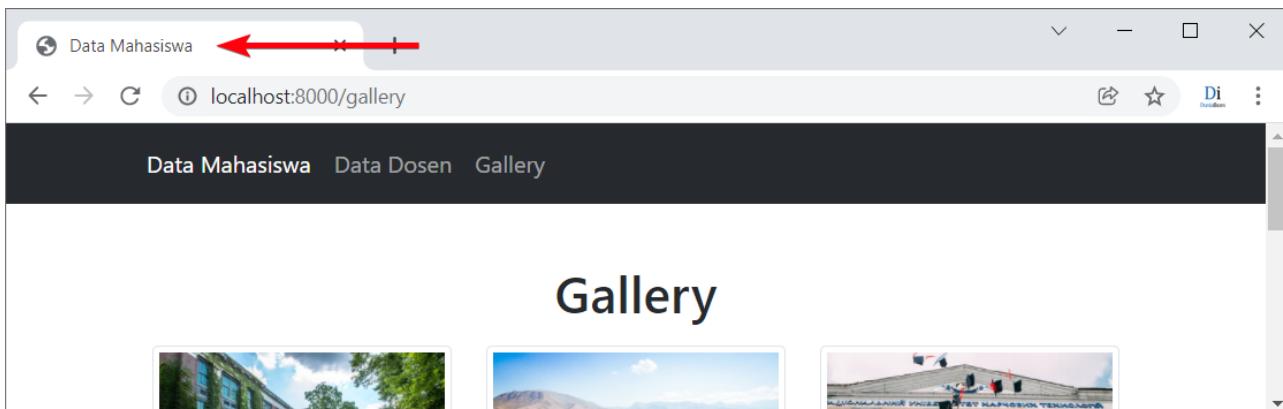
Jika ditulis sebagai `@include('header')`, maka laravel akan mencari file `header.blade.php` langsung di dalam folder `view`, bukan di dalam folder saat ini.

8.13. Mengirim Data ke Include

Perintah `@include` bawaan Laravel tidak hanya menggantikan fungsi `include()` milik PHP saja, tapi juga memiliki fitur tambahan, yakni bisa dipakai untuk mengirim data.

Pada praktik sebelum ini, file yang di-include bersifat "apa adanya" atau tetap. Sebagai contoh, dalam file `header.blade.php` saya menulis tag `<title>Data Mahasiswa</title>`, maka semua halaman yang men-include file header ini memiliki title yang sama pula.

Seharusnya title setiap halaman berbeda-beda tergantung konten-nya. Untuk halaman dosen memakai title `<title>Data Dosen</title>` dan untuk halaman gallery menggunakan `<title>Gallery</title>`.



Gambar: Title di halaman Gallery masih menampilkan teks "Data Mahasiswa"

Dengan PHP native, salah satu solusi yang bisa dipakai adalah membuat kondisi if di dalam file header, yakni jika halaman yang sedang tampil adalah halaman gallery, maka ubah baris tag

<title> menjadi <title>Gallery</title>.

Untuk teknik tersebut kita perlu mengakses konstanta khusus PHP seperti __FILE__, serta beberapa fungsi lain. Namun blade menyediakan solusi yang lebih mudah, yakni dengan cara mengirim data title secara langsung pada saat proses include.

Mari kita bahas dengan contoh praktik. Menggunakan blade, perintah include di view mahasiswa bisa diubah menjadi berikut:

resources/views/mahasiswa.blade.php

```
1 @include('layout.header',[ 'title' => 'Data Mahasiswa'])
2
3 <div class="container text-center mt-3 p-4 bg-white">
4   <h1 class="mb-3">Data Mahasiswa</h1>
5   ...
6   ...
```

Di baris 1, perintah @include mendapat tambahan argument kedua, yakni sebuah associative array. Ini artinya ketika proses include, kirim data 'title' dengan nilai 'Data Mahasiswa' ke dalam layout.header.

Di dalam file header, data ini bisa ditangkap menggunakan format yang sama seperti menampilkan data dari route:

resources/views/layout/header.blade.php

```
1 <!DOCTYPE html>
2 ...
3 <link href="/css/bootstrap.min.css" rel="stylesheet">
4   <title>{{$title}}</title>
5 </head>
6 ...
```

Sekarang isi tag <title> di baris 4 adalah {{\$title}}, yakni perintah blade untuk menampilkan variabel \$title.

Dengan demikian, kita tinggal mengubah penulisan perintah include untuk view dosen dan juga view gallery:

resources/views/dosen.blade.php

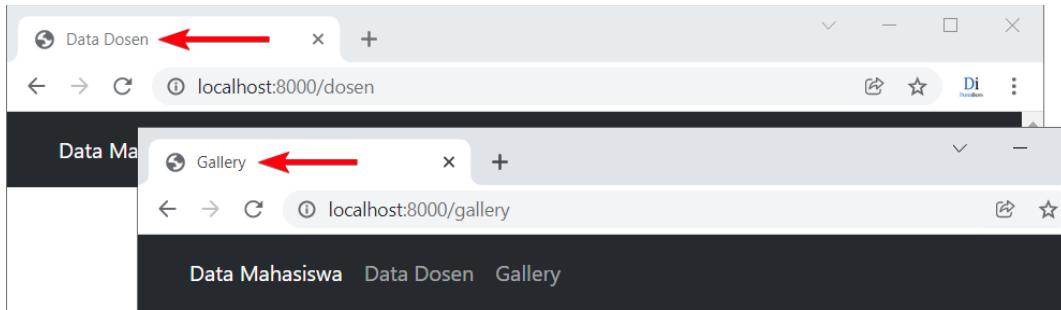
```
1 @include('layout.header',[ 'title' => 'Data Dosen'])
2
3 <div class="container text-center mt-3 p-4 bg-white">
4   <h1 class="mb-3">Data Dosen</h1>
5   ...
```

resources/views/gallery.blade.php

```
1 @include('layout.header',[ 'title' => 'Gallery'])
2
```

```
3 <div class="container text-center mt-3 p-4 bg-white">
4   <h1 class="mb-3">Gallery</h1>
5   ...
```

Dengan penambahan ini, title dari setiap halaman juga akan berubah mengikuti data 'title' yang dikirim.



Gambar: Title di halaman Dosen dan Gallery sudah berubah

Jika diperlukan, kita bisa mengirim lebih dari 1 nilai sewaktu proses include, caranya tinggal menulis element lain di dalam associative array seperti contoh berikut:

```
@include('layout.header', ['title' => 'Data Dosen', 'dataA' = 'NilaiA',
'dataB' => 'NilaiB'])
```

Perintah ini artinya saya mengirim 3 buah data ke file `layout.header`. Nantinya ketiga nilai bisa ditampilkan dengan perintah `{{title}}`, `{{dataA}}` dan `{{dataB}}`.

Sebenarnya ada 1 masalah lagi dengan template kita, yakni class `.active` di menu navigasi masih tetap di dalam tag `<a>` mahasiswa. Seharusnya ketika dibuka halaman dosen, maka class `.active` ini juga pindah ke tag `<a>` dosen. Blade juga menyediakan solusi untuk masalah ini (akan kita bahas sesaat lagi).



Gambar: Warna link tidak pindah ke menu 'Gallery' karena class `.active` ada di menu 'Data Mahasiswa'

8.14. Layout Extends

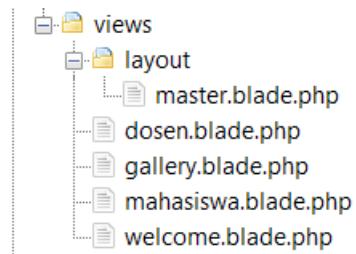
Teknik include yang sudah di pelajari sebelum ini cukup sering dipakai (terutama dalam PHP native). Dengan include, kita meng-import file lain ke file saat ini. Misalnya di view `mahasiswa`

terdapat perintah untuk meng-import file header ke bagian atas file, serta file footer ke bagian bawah file.

Selain itu, blade menyediakan teknik alternatif yang dalam kebanyakan situasi lebih praktis daripada menggunakan teknik include. Caranya adalah dengan "menurunkan" sebuah file view ke view lain, atau bisa juga disebut teknik **men-extends sebuah view**.

Idenya adalah, siapkan sebuah file "master" yang berisi semua komponen HTML seperti header, menu navigasi dan footer. Kemudian beri "tanda" untuk komponen yang bisa di timpa oleh view turunan atau "child view".

Mari kita bahas dengan contoh praktek. Silahkan hapus file header.blade.php dan footer.blade.php di dalam folder layout, kemudian buat file baru dengan nama master.blade.php di dalam folder tersebut:



Gambar: Struktur folder layout

Dan berikut isi kode program untuk file master.blade.php:

resources/views/layout/master.blade.php

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link href="/css/bootstrap.min.css" rel="stylesheet">
8      <title>@yield('title')</title>
9  </head>
10 <body>
11
12 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13     <div class="container">
14         <ul class="navbar-nav">
15             <li class="nav-item">
16                 <a class="nav-link active" href="/mahasiswa">Data Mahasiswa</a>
17             </li>
18             <li class="nav-item">
19                 <a class="nav-link" href="/dosen">Data Dosen</a>
20             </li>
21             <li class="nav-item">
22                 <a class="nav-link" href="/gallery">Gallery</a>
23             </li>
24         </ul>

```

```

25    </div>
26  </nav>
27
28 @yield('content')
29
30 <footer class="bg-dark py-4 text-white mt-4">
31 <div class="container">
32   Sistem Informasi Mahasiswa | Copyright © {{ date("Y") }} DuniaIlkom
33 </div>
34 </footer>
35
36 </body>
37 </html>

```

File ini berisi kode program dari komponen yang sudah kita pakai sebelumnya. Di baris 1 – 9 terdapat kode untuk membuat header, di baris 12 – 24 untuk menu navigasi serta baris 28 – 32 untuk membuat footer.

Kode tambahan ada di baris 8 dan 26, berupa perintah `@yield` bawaan blade. Perintah `@yield` dipakai untuk menandai bagian yang bisa ditimpas oleh view turunan nantinya. Teks di dalam tanda kurung merupakan nama atau id dari yield tersebut. Perintah `@yield('title')` artinya terdapat yield dengan id 'title', serta perintah `@yield('content')` akan membuat yield dengan id 'content'.

File `master.blade.php` ini tidak dirancang bisa diakses langsung, tapi harus diturunkan ke file view lain. Kita akan meng-**extends** file ini ke dalam view `mahasiswa`, `dosen` dan `gallery`.

Untuk file `mahasiswa.blade.php` bisa ditulis sebagai berikut:

`resources/views/mahasiswa.blade.php`

```

1  @extends('layout.master')
2  @section('title', 'Data Mahasiswa')
3
4  @section('content')
5  <div class="container text-center mt-3 p-4 bg-white">
6    <h1 class="mb-3">Data Mahasiswa</h1>
7    <div class="row">
8      <div class="col-sm-8 col-md-6 m-auto">
9        <ol class="list-group">
10       @forelse ($mahasiswa as $val)
11         <li class="list-group-item">{{$val}}</li>
12       @empty
13         <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
14       @endforelse
15     </ol>
16   </div>
17 </div>
18 </div>
19 @endsection

```

Di baris 1 terdapat perintah `@extends('layout.master')`, inilah perintah yang memberitahu

Laravel bahwa view ini merupakan turunan dari file `layout.master`.

Ketika sebuah view di extends, maka semua komponen yield dari view asal harus diisi. Dalam file `layout.master` terdapat 2 buah id yield yakni '`title`' dan '`content`'. Terdapat beberapa cara untuk mengisi yield di child view.

Jika nilai yang ingin kita isi hanya berupa 1 baris teks singkat, bisa menggunakan cara pertama seperti di baris 2. Perintah `@section('title', 'Data Mahasiswa')` artinya isi teks 'Data Mahasiswa' ke dalam yield dengan id '`title`'.

Di dalam file `layout.master`, yield '`title`' ditulis sebagai berikut:

```
<title>@yield('title')</title>
```

Maka dengan pemanggilan perintah `@section('title', 'Data Mahasiswa')`, baris ini akan menjadi:

```
<title>Data Mahasiswa</title>
```

Namun apabila teks yang akan diisi ke dalam yield cukup panjang dan terdiri dari beberapa baris, kita bisa menggunakan cara kedua seperti di baris 4 – 19. Perintah ini diawali dengan `@section` sebagai penanda di mulainya blok dan diakhiri dengan `@endsection`.

Artinya, seluruh kode program yang terdapat di antara `@section('content')` dan `@endsection` akan menggantikan isi dari `@yield('content')` yang ada di `layout.master`. Perulangan `foreach` untuk menampilkan data mahasiswa di baris 10 -14 juga ikut mengisi bagian `@yield('content')` di `layout.master`.

Hasil akhir dari view mahasiswa akan sama seperti contoh kita sebelumnya.

The screenshot shows a web browser window with the following details:

- Title Bar:** Data Mahasiswa
- Address Bar:** localhost:8000/mahasiswa
- Header:** Data Mahasiswa, Data Dosen, Gallery
- Main Content:**
 - Title:** Data Mahasiswa
 - A table structure with four rows:
 - Risa Lestari
 - Rudi Hermawan
 - Bambang Kusumo
 - Lisa Permata
- Footer:** Sistem Informasi Mahasiswa | Copyright © 2022 DuniaIlkom

Gambar: Tampilan halaman mahasiswa hasil extends

Bagaimana dengan view dosen dan gallery? Prinsipnya sama saja, yakni tinggal mengisi bagian @yield('title') dan @yield('content') untuk setiap view.

Berikut kode program dari view dosen:

resources/views/dosen.blade.php

```
1  @extends('layout.master')
2  @section('title','Data Dosen')
3
4  @section('content')
5  <div class="container text-center mt-3 p-4 bg-white">
6      <h1 class="mb-3">Data Dosen</h1>
7      <div class="row">
8          <div class="col-sm-8 col-md-6 m-auto">
9              <ol class="list-group">
10             @forelse ($dosen as $val)
11                 <li class="list-group-item">{{$val}}</li>
12             @empty
13                 <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
14             @endforelse
15             </ol>
16         </div>
17     </div>
18 </div>
19 @endsection
```

Dan berikut kode program untuk view gallery:

resources/views/gallery.blade.php

```
1  @extends('layout.master')
2  @section('title','Gallery')
3
4  @section('content')
5  <div class="container text-center mt-3 p-4 bg-white">
6      <h1 class="mb-3">Gallery</h1>
7      <div class="row">
8          <div class="col-4">
9              
11          </div>
12          ...
13          ...
14          <div class="col-4 mt-4">
15              
17          </div>
18      </div>
19 </div>
20 @endsection
```

Semoga anda bisa memahami cara kerja dari **extends layout** ini. Prinsipnya, kita menurunkan satu view ke view lain dan mengisi komponen tertentu yang ingin diubah.

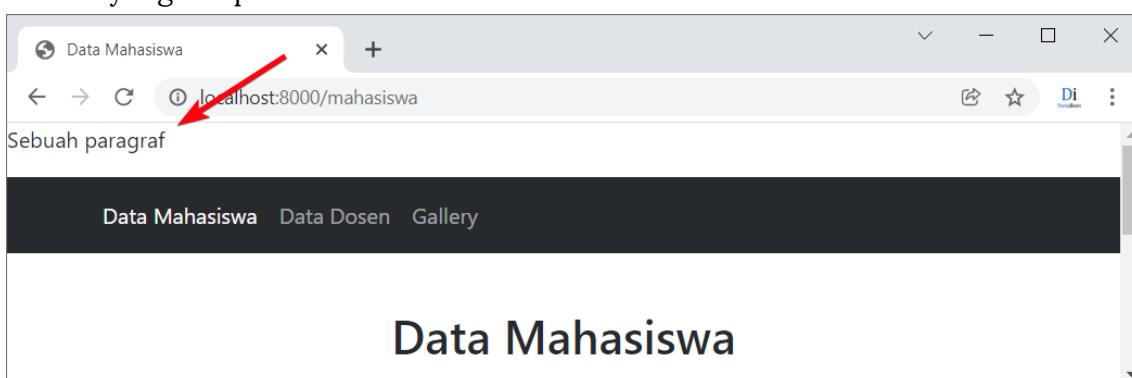
Sedikit catatan tambahan yang mungkin bisa jadi salah satu kekurangan dari extends (dibandingkan dengan teknik include) adalah, kita tidak bisa menulis kode HTML tambahan di luar struktur @section. Jika ini dilakukan, kode HTML tersebut akan "lompat" ke bagian paling atas kode HTML.

Sebagai contoh, saya akan modifikasi view mahasiswa sebagai berikut:

resources/views/mahasiswa.blade.php

```
1  @extends('layout.master')
2  @section('title','Data Mahasiswa')
3
4  @section('content')
5  <div class="container text-center mt-3 p-4 bg-white">
6      <h1 class="mb-3">Data Mahasiswa</h1>
7      <div class="row">
8          <div class="col-sm-8 col-md-6 m-auto">
9              <ol class="list-group">
10             @forelse ($mahasiswa as $val)
11                 <li class="list-group-item">{{$val}}</li>
12             @empty
13                 <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
14             @endforelse
15             </ol>
16         </div>
17     </div>
18 </div>
19 @endsection
20 <p>Sebuah paragraf</p>
```

Perhatikan di baris 20 terdapat tambahan sebuah tag <p> yang berada di luar @section. Berikut hasil yang tampil di web browser:



Gambar: Hasil tambahan paragraf baru ke dalam view mahasiswa

Paragraf tersebut akan tampil di baris paling atas. Dan jika dilihat dari source code, teks ini sebenarnya berada sebelum doctype:

5 <head>
6 <meta charset="UTF-8">
7 <meta name="viewport" content="width=device-width, initial-scale=1.0">
8 <meta http-equiv="X-UA-Compatible" content="ie=edge">
9 <link href="css/app.css" rel="stylesheet">
10 <title>Data Mahasiswa</title>
11 </head>"/>

Gambar: Hasil tambahan paragraf baru ada sebelum doctype

Artinya, ketika sebuah view di extends, selalu tempatkan kode program di dalam blok @section. Jika ingin menambah element baru, itu harus dilakukan di master view, bukan di child view.

Mengatasi Masalah Menu Navigasi

Rancangan template yang kita buat masih memiliki sedikit masalah di menu navigasi, dimana class .active tidak pindah sesuai halaman yang dibuka. Menggunakan teknik layout extends, kita akan coba mengatasi hal ini.

Idenya adalah, buat @yield dengan id yang berbeda-beda untuk setiap tag <a>. Nilai untuk @yield nantinya akan diisi dari child view.

Berikut modifikasi halaman master.blade.php dengan penambahan ini:

resources/views/layout/master.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 ...
4 ...
5 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
6   <div class="container">
7     <ul class="navbar-nav">
8       <li class="nav-item">
9         <a class="nav-link" @yield('menuMahasiswa')> Data Mahasiswa</a>
10      ...
11    </li>
12    <li class="nav-item">
13      <a class="nav-link" @yield('menuDosen')> Data Dosen</a>
14    </li>
15    <li class="nav-item">
16      <a class="nav-link" @yield('menuGallery')> Gallery</a>
17    </li>
18  </ul>
19 </div>
20 </nav>
21 ...
```

Perhatikan baris 9, 13, dan 16, di dalam atribut `class` saya menambah 3 buah `yield` untuk setiap tag `<a>`. Menu yang menuju ke halaman mahasiswa menggunakan `@yield('menuMahasiswa')`, menu ke halaman dosen menggunakan `@yield('menuDosen')`, serta menu ke halaman gallery menggunakan `@yield('menuGallery')`.

Dengan tambahan ini, dalam setiap view yang men-extends `master.blade.php`, kita akan isi menggunakan nilai `@yield` yang berbeda-beda.

Berikut modifikasi dari view `mahasiswa`:

`resources/views/mahasiswa.blade.php`

```
1 @extends('layout.master')
2 @section('title','Data Mahasiswa')
3 @section('menuMahasiswa','active')
4
5 @section('content')
6 <div class="container text-center mt-3 p-4 bg-white">
7   <h1 class="mb-3">Data Mahasiswa</h1>
8   ...
9   ...
10 @endsection
```

Di baris 3 terdapat tambahan perintah `@section('menuMahasiswa','active')`. Artinya, teks '`active`' akan mengisi `@yield('menuMahasiswa')`. Inilah yang membuat menu mahasiswa berwarna lebih cerah.

Begitu juga untuk view `dosen` dan `gallery`:

`resources/views/dosen.blade.php`

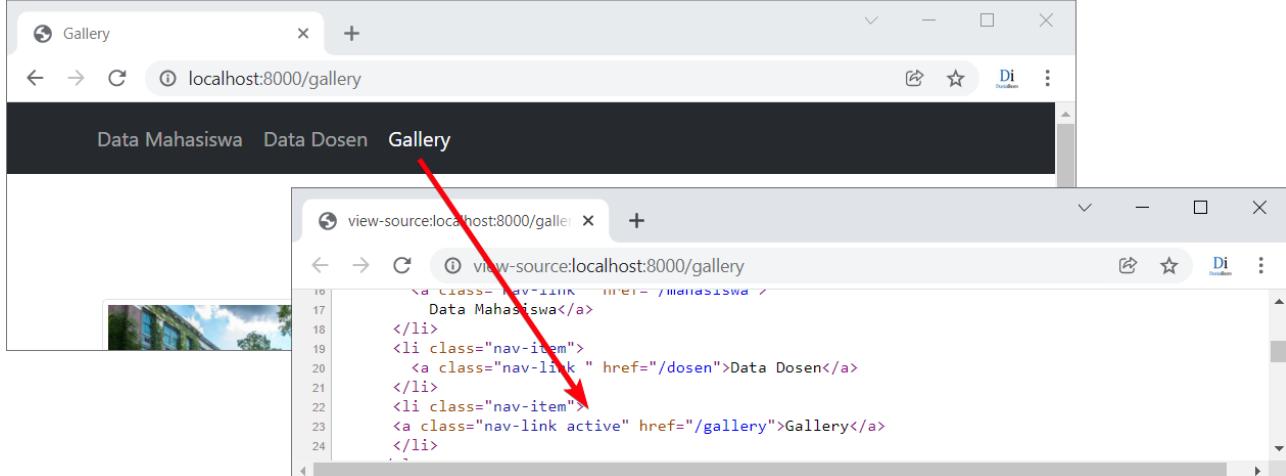
```
1 @extends('layout.master')
2 @section('title','Data Dosen')
3 @section('menuDosen','active')
4
5 @section('content')
6 <div class="container text-center mt-3 p-4 bg-white">
7   <h1 class="mb-3">Data Dosen</h1>
8   ...
9   ...
10 @endsection
```

`resources/views/gallery.blade.php`

```
1 @extends('layout.master')
2 @section('title','Gallery')
3 @section('menuGallery','active')
4
5 @section('content')
6 <div class="container text-center mt-3 p-4 bg-white">
7   <h1 class="mb-3">Gallery</h1>
8   ...
9   ...
```

```
10 @endsection
```

Dengan tambahan ini, ketika kita membuka sebuah halaman, class `.active` juga akan mengikuti:



Gambar: Class `.active` akan menyesuaikan tergantung halaman yang dibuka

8.15. Default Value

Di halaman view master atau *parent view* (file `master.blade.php`), kita juga bisa memberikan nilai default saat menulis perintah `@yield`. Nilai default akan dipakai jika di dalam *child view* tidak ditemukan perintah `@section` untuk mengisi `@yield` tersebut.

Sebagai contoh praktik, saya akan modifikasi sedikit bagian header di file `master.blade.php`:

```
resources/views/layout/master.blade.php
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 ...
5   <title>@yield('title','Sistem Informasi Mahasiswa')</title>
6 </head>
7 <body>
```

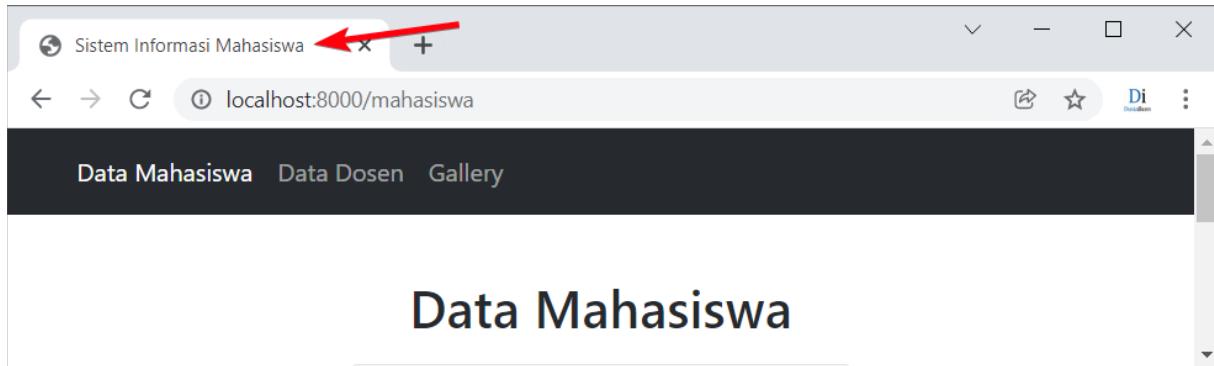
Perintah `@yield` untuk tag `<title>` di baris 5 sekarang memiliki 2 buah argument. Argument pertama yakni 'title' merupakan **id** dari yield, sedangkan argument kedua berupa teks 'Sistem Informasi Mahasiswa' merupakan nilai default.

Jika di view `mahasiswa` kita tidak menulis perintah `@section('title')`, maka tag title akan menjadi `<title>Sistem Informasi Mahasiswa</title>`.

```
resources/views/mahasiswa.blade.php
```

```
1 @extends('layout.master')
2 {{-- @section('title','Data Mahasiswa') --}}
3 @section('menuMahasiswa','active')
```

```
4  
5 @section('content')  
6 <div class="container text-center mt-3 p-4 bg-white">  
7   <h1 class="mb-3">Data Mahasiswa</h1>  
8   ...  
9   ...  
10 @endsection
```



Gambar: Title halaman yang tampil adalah 'Sistem Informasi Mahasiswa'

Di baris 2, perintah `@section('title')` saya non-aktifkan dengan cara menempatkannya ke dalam blok komentar blade. Hasilnya, title halaman mahasiswa akan menampilkan teks 'Sistem Informasi Mahasiswa', sesuai dengan nilai default sewaktu penulisan `@yield`.

8.16. Section Extends

Untuk pembuatan layout yang lebih kompleks, blade juga menyediakan cara untuk meng-**extends section** milik *parent view*. Penjelasannya akan lebih mudah dengan contoh praktek.

Pertama, saya akan modifikasi kembali file `master.blade.php`:

`resources/views/layout/master.blade.php`

```
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4  ...  
5  <title>@yield('title','Sistem Informasi Mahasiswa')</title>  
6  </head>  
7  <body>  
8  
9  <nav class="navbar navbar-expand-sm navbar-dark bg-dark">  
10 ...  
11 </nav>  
12  
13 @section('content')  
14 <div class="alert alert-primary text-center">Sistem Informasi Mahasiswa</div>  
15 @show  
16  
17 <footer class="bg-dark py-4 text-white mt-4">  
18 ...
```

```
19 </footer>
20
21 </body>
22 </html>
```

Kode programnya tidak saya tampilkan lengkap karena sama persis dengan contoh kita sebelumnya. Yang jadi fokus utama ada di baris 13 – 15. Meskipun ini adalah *parent view*, tapi perintah yang digunakan adalah `@section('content')`, yang kemudian diakhiri dengan perintah `@show`.

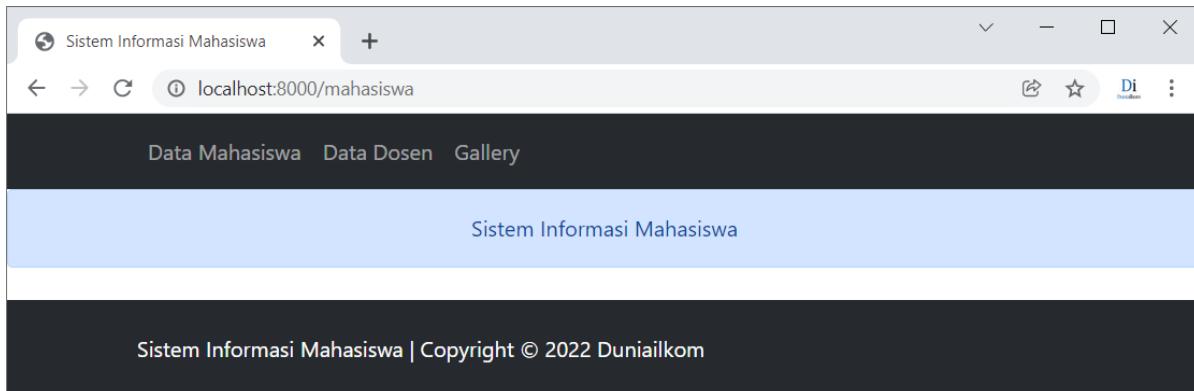
Seluruh kode program di antara `@section('content')` dan `@show` bisa disebut sebagai **opsional section** milik *parent view*. Di dalam *child view*, section ini bisa di sambung atau ditimpak secara keseluruhan.

Selanjutnya saya akan edit view `mahasiswa` dengan menghapus semua kode yang ada dan menyisakan 1 baris saja:

```
resources/views/mahasiswa.blade.php
```

```
1 @extends('layout.master')
```

Betul, hanya 1 baris perintah untuk men-extends file `master.blade.php`. Berikut hasilnya:



Gambar: Halaman mahasiswa dengan hanya extends file master.blade.php

Komponen menu navigasi dan footer tampil secara sempurna karena keduanya diturunkan dari `master.blade.php`.

Di bagian tengah terdapat teks 'Sistem Informasi Mahasiswa' dengan background biru muda. Ini berasal dari `@section('content')` yang ada di `master.blade.php`. Artinya, blok `@section('content')` milik *parent view* tetap tampil di *child view*.

Sekarang, saya akan tambah bagian `@section('content')` di view `mahasiswa`:

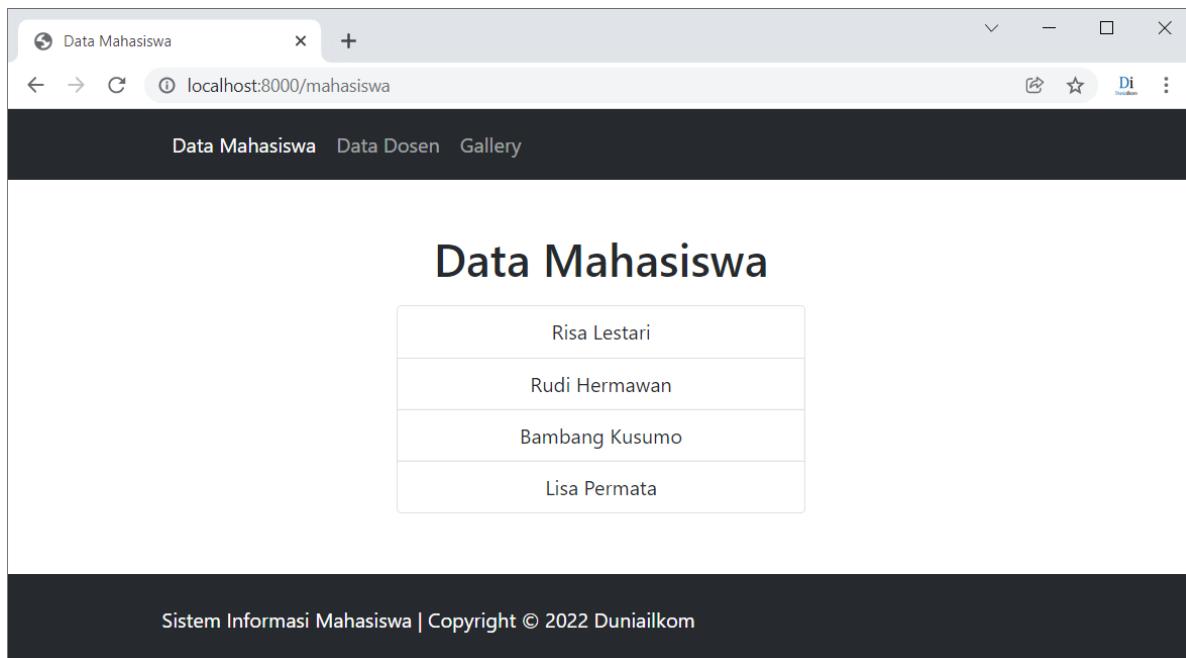
```
resources/views/mahasiswa.blade.php
```

```
1 @extends('layout.master')
2 @section('title','Data Mahasiswa')
3 @section('menuMahasiswa','active')
4
```

```

5  @section('content')
6  <div class="container text-center mt-3 p-4 bg-white">
7    <h1 class="mb-3">Data Mahasiswa</h1>
8    <div class="row">
9      <div class="col-sm-8 col-md-6 m-auto">
10        <ol class="list-group">
11          @forelse ($mahasiswa as $val)
12            <li class="list-group-item">{{$val}}</li>
13          @empty
14            <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
15          @endforelse
16        </ol>
17      </div>
18    </div>
19  </div>
20 @endsection

```



Gambar: Tampilan view mahasiswa

Kode program untuk view mahasiswa ini sama seperti yang kita pakai selama ini, dimana bagian `@section('content')` berisi perintah untuk menampilkan array mahasiswa yang dikirim dari route.

Namun perhatikan, teks 'Sistem Informasi Mahasiswa' dengan background biru tidak lagi tampil. Ini terjadi karena dalam view `mahasiswa`, **kita sudah menimpa** `@section('content')` milik `master.blade.php`. Jika bagian `@section('content')` di view mahasiswa dihapus, maka teks tersebut akan tampil kembali.

Sampai di sini bisa diambil kesimpulan:

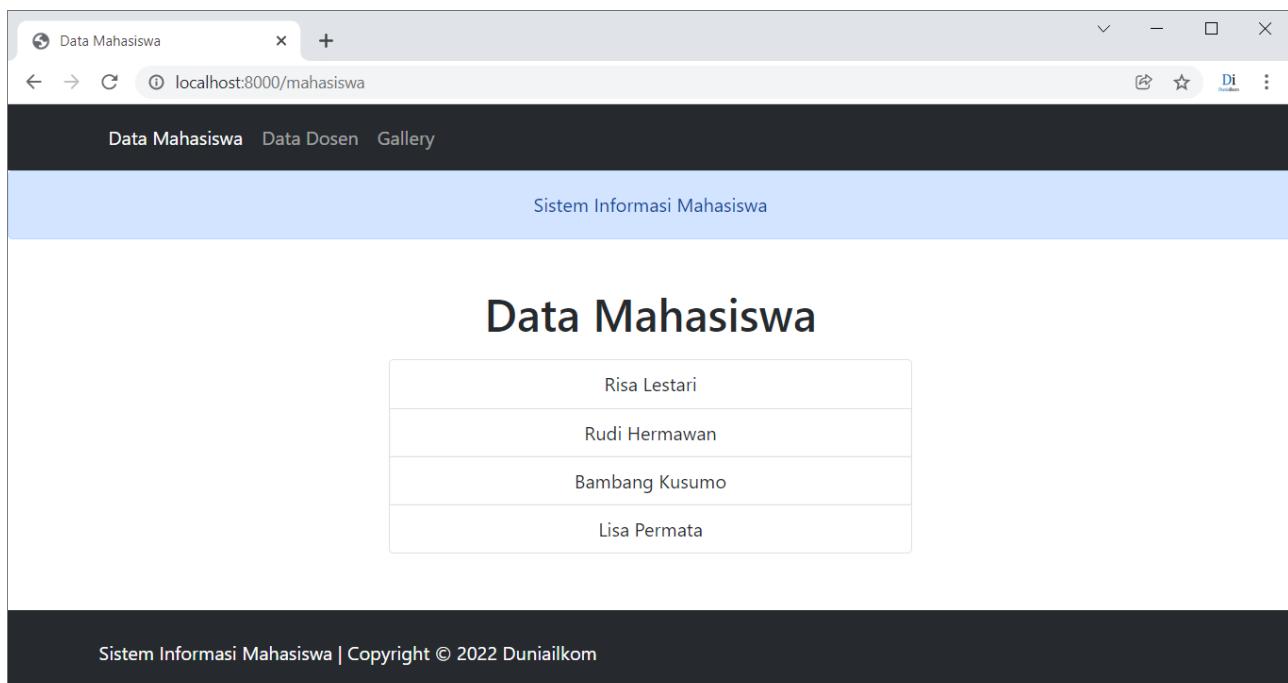
1. Jika di *child view* tidak ada perintah `@section('content')`, maka yang akan tampil adalah `@section('content')` milik *parent view*.

2. Jika di *child view* terdapat perintah `@section('content')`, maka itu akan menimpa `@section('content')` milik *parent view*.

Selain itu terdapat alternatif ketiga, yakni kita bisa menampilkan `@section('content')` milik *parent view*, sekaligus isi `@section('content')` kepunyaan *child view*. Caranya, tambah perintah `@parent`:

resources/views/mahasiswa.blade.php

```
1 @extends('layout.master')
2 @section('title','Data Mahasiswa')
3 @section('menuMahasiswa','active')
4
5 @section('content')
6 @parent
7 <div class="container text-center mt-3 p-4 bg-white">
8   <h1 class="mb-3">Data Mahasiswa</h1>
9   ...
10 </div>
11 @endsection
```



Gambar: Tampilan view mahasiswa yang meng-extends content milik master.blade.php

Di baris 6 terdapat perintah `@parent`. Perintah ini dipakai agar dari `@section('content')` milik *child view*, kita bisa mengakses `@section('content')` milik *parent view*.

Posisi penulisan perintah `@parent` juga berpengaruh. Jika perintah `@parent` di tulis pada akhir section, yakni sebelum perintah penutup `@endsection`, maka teks 'Sistem Informasi Mahasiswa' juga akan muncul di sisi bawah (setelah list nama mahasiswa).

Inilah yang dimaksud dengan **opsional section**, dimana kita bisa mengatur apakah ingin

menampilkan `@section` milik *parent view*, menimpanya dengan nilai baru, atau menyambungnya dengan kode lain.

Sebagai tambahan, perintah `@parent` tidak bisa ditulis terpisah, tapi harus di dalam blok `@section` dengan id yang sama dengan kepunyaan *parent view*. Sebagai contoh, jika di *parent view* terdapat `@section('foo')`, maka hanya bisa di extends dari `@section('foo')` milik *child view*, tidak bisa dari `@section('bar')`.

Selain `@endsection`, untuk menutup `@section` juga bisa memakai perintah `@stop`.

8.17. Components dan Slots

Components dan **slots** adalah fitur blade untuk membuat bagian layout yang "lepas" dan bisa digunakan ulang. Ini sebenarnya mirip seperti men-include view yang sudah kita pelajari, tapi components dan slots memiliki beberapa fitur tambahan.

Sebagai bahan praktik, saya akan membuat route dan view baru:

routes/web.php

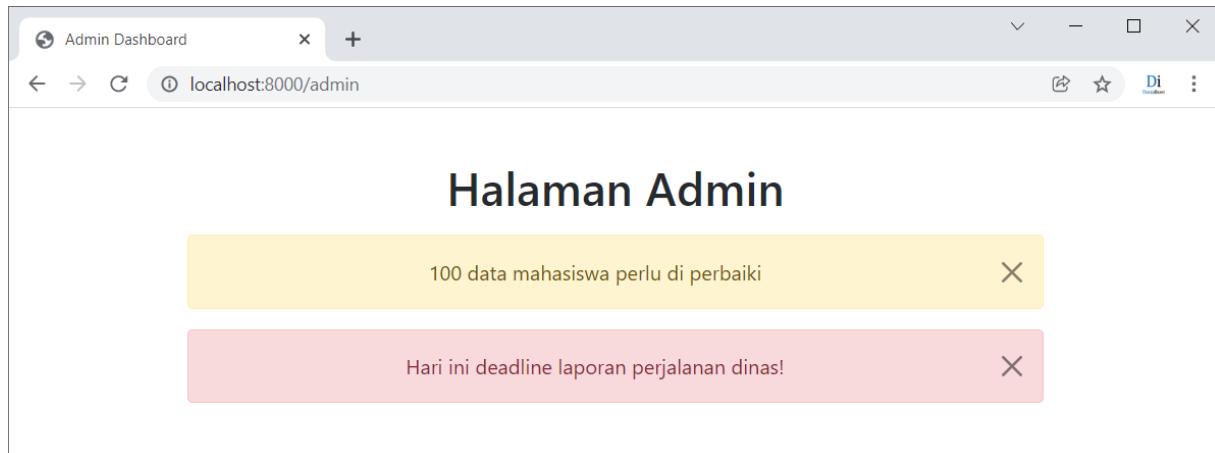
```
1 Route::get('admin', function () {
2     return view('admin');
3 });
```

Route ini hanya memanggil view `admin`. Dan berikut kode program untuk file `admin.blade.php`:

resources/views/admin.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Admin Dashboard</title>
9 </head>
10 <body>
11 <div class="container text-center mt-3 p-4 bg-white">
12     <h1 class="mb-3">Halaman Admin</h1>
13     <div class="row">
14         <div class="col-12">
15
16             <div class="alert alert-warning alert-dismissible fade show">
17                 100 data mahasiswa perlu di perbaiki
18                 <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
19             </div>
20
21             <div class="alert alert-danger alert-dismissible fade show">
22                 Hari ini deadline laporan perjalanan dinas!
```

```
23      <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
24    </div>
25
26  </div>
27</div>
28</div>
29
30<script src="/js/bootstrap.bundle.min.js"></script>
31</body>
32</html>
```



Gambar: Tampilan halaman admin

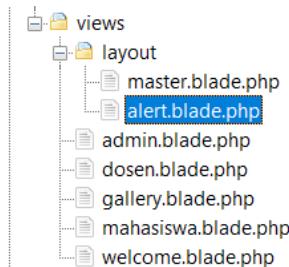
Konten untuk halaman `admin` terdiri dari tag `<h1>` di baris 12, kemudian diikuti dengan 2 buah komponen **alert** Bootstrap. Alert pertama ada di baris 16 – 19, dan alert kedua di baris 21 – 24.

Kedua alert ini memakai struktur kode yang hampir sama. Yang berbeda hanya penggunaan class `.alert-warning` dan `.alert-danger` untuk memberi warna di baris 16 dan 21, serta isi teks alert di baris 17 dan 22.

Dalam praktek kita, komponen alert ini ingin saya pakai pada view lain dengan sedikit perbedaan pada nama class serta pesan alert.

Masalahnya, bagaimana cara membuat alert ini bisa dipakai ulang di tempat lain tanpa harus membuat ulang semua kode HTML? Untuk keperluan inilah blade menyediakan fitur **components** dan **slots**.

Cara kerjanya adalah sebagai berikut, pertama buat potongan view yang berisi hanya komponen alert saja. File ini saya namakan `alert.blade.php` dan di simpan pada folder `layout`:



Gambar: Struktur folder views

```
resources/views/layout/alert.blade.php
```

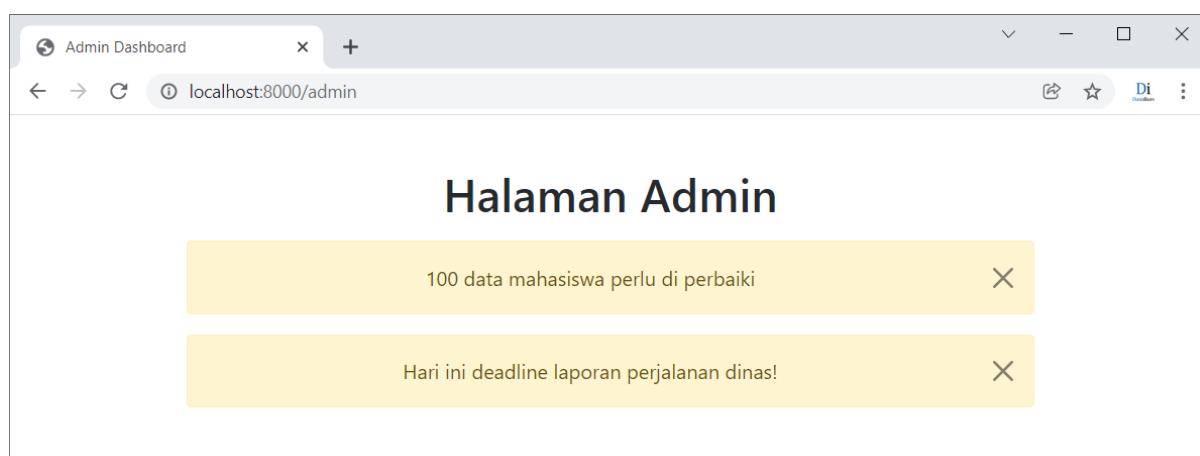
```
1 <div class="alert alert-warning alert-dismissible fade show">
2   {{$slot}}
3   <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
4 </div>
```

Kode untuk alert ini sama persis dengan di halaman admin. Hanya saja sekarang di baris 2 terdapat penulisan perintah `{{$slot}}`. Di baris 2 inilah kita akan menulis pesan alert.

Kembali ke halaman admin, saya akan modifikasi agar kita bisa menggunakan view alert:

```
resources/views/admin.blade.php
```

```
1  <!DOCTYPE html>
2  ...
3  <body>
4  <div class="container text-center mt-3 p-4 bg-white">
5    <h1 class="mb-3">Halaman Admin</h1>
6    <div class="row">
7      <div class="col-12">
8
9        @component('layout.alert')
10          100 data mahasiswa perlu di perbaiki
11        @endcomponent
12
13        @component('layout.alert')
14          Hari ini deadline laporan perjalanan dinas!
15        @endcomponent
16
17      </div>
18    </div>
19  </div>
20  ...
21 </html>
```



Gambar: Tampilan penggunaan komponen alert

Di baris 9 terdapat kode `@component('layout.alert')`, ini adalah instruksi kepada blade bahwa kita ingin menggunakan komponen bernama 'layout.alert', yang merujuk ke file `alert.blade.php` sebelumnya.

Kemudian di baris 10 berisi teks '`100 data mahasiswa perlu di perbaiki`'. Teks inilah yang akan menggantikan baris `{{$slot}}` di file `alert.blade.php`. Dan di baris 11 ditutup dengan perintah `@endcomponent`.

Di baris 13 – 15 saya juga memanggil kembali komponen alert tapi kali ini dengan teks yang berbeda.

Inilah cara penggunaan fitur **components** dan **slots** dari blade. Kita bisa dengan mudah menggunakan ulang (re-use) kode HTML di view lain. Namun tampilan alert masih belum pas karena warna alert seharusnya bisa dibuat berbeda. Blade juga menyediakan solusi untuk masalah ini.

Berikut modifikasi ulang file `alert.blade.php`:

```
resources/views/layout/alert.blade.php
```

```
1 <div class="alert alert-{{$class}} alert-dismissible fade show">
2   {{$slot}}
3   <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
4 </div>
```

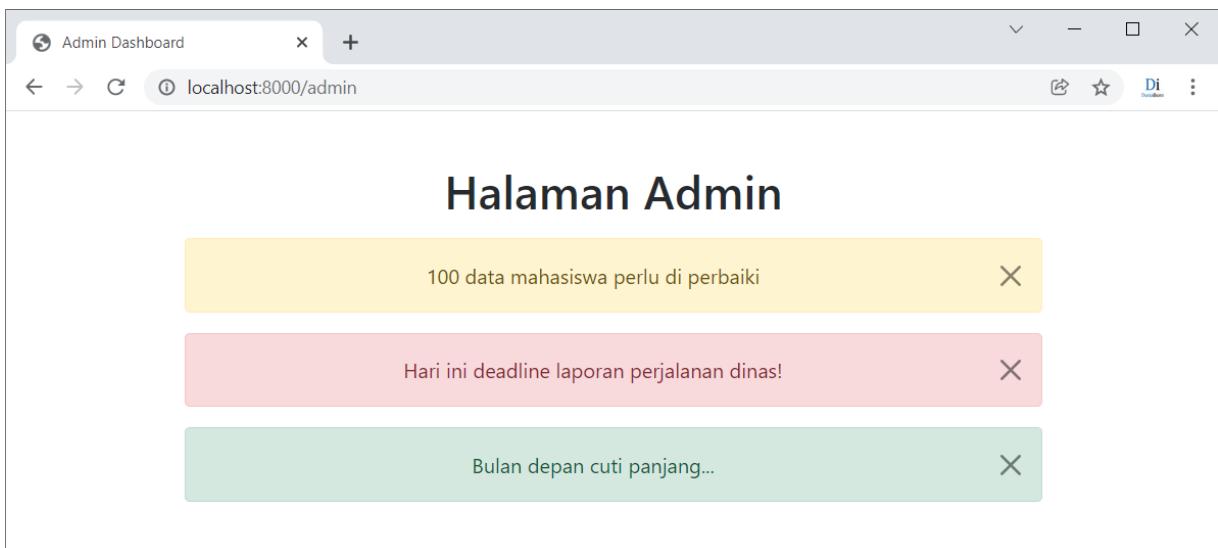
Sekarang di baris 1 terdapat tambahan kode `{{$class}}`. Variabel ini berada di dalam atribut class dari tag `<div>`, sehingga nantinya kita bisa isi dengan berbagai class warna alert Bootstrap, seperti `.alert-success`, `.alert-primary`, `.alert-warning`, dst. Nama variabel `$class` ini juga bisa diganti dengan nama lain sesuai kebutuhan, tidak harus `$class`, tapi bisa juga `$nama_class`, atau `$kodesaya`.

Dengan penambahan ini, saya akan modifikasi kembali cara pemanggilan perintah `@component` di view `admin`:

```
resources/views/admin.blade.php
```

```
1 <div class="container text-center mt-3 p-4 bg-white">
2   <h1 class="mb-3">Halaman Admin</h1>
3   <div class="row">
4     <div class="col-12">
5
6       @component('layout.alert')
7         @slot('class')
8           warning
9         @endslot
10        100 data mahasiswa perlu di perbaiki
11      @endcomponent
12
13      @component('layout.alert')
14        @slot('class')
```

```
15      danger
16      @endslot
17      Hari ini deadline laporan perjalanan dinas!
18  @endcomponent
19
20  @component('layout.alert')
21      @slot('class')
22          success
23  @endslot
24      Bulan depan cuti panjang...
25  @endcomponent
26
27  </div>
28  </div>
29 </div>
```



Gambar: Tampilan alert dengan warna dan teks yang berbeda-beda

Di sini terdapat 3 buah pesan alert menggunakan `@component('layout.alert')`. Perhatikan cara penulisannya, untuk mengisi variabel `{{$class}}` yang terdapat di komponen alert, kita menggunakan format berikut:

```
1  @slot('class')
2      <teks yang dikirim>
3  @endslot
```

Sedangkan untuk mengisi variabel `{{$slot}}`, cukup dengan menulis teks tersebut di dalam blok `@component('layout.alert')` sebelum perintah penutup `@endcomponent`.

Dengan demikian, jika ditulis seperti ini:

```
1  @component('layout.alert')
2      @slot('class')
3          success
4  @endslot
5      Bulan depan cuti panjang...
```

```
6 @endcomponent
```

Akan di proses oleh file `alert.blade.php` menjadi:

```
1 <div class="alert alert-success alert-dismissible fade show">
2   Bulan depan cuti panjang...
3   <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
4 </div>
```

Bagaimana jika ingin membuat variabel lain? Caranya hampir sama. Sebagai contoh, saya akan modifikasi lagi file `alert.blade.php`:

```
resources/views/layout/alert.blade.php
```

```
1 <div class="alert alert-{{$class}} alert-dismissible fade show">
2   <h4 class="alert-heading"><u>{{$judul}}</u></h4>
3   {{$slot}}
4   <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
5 </div>
```

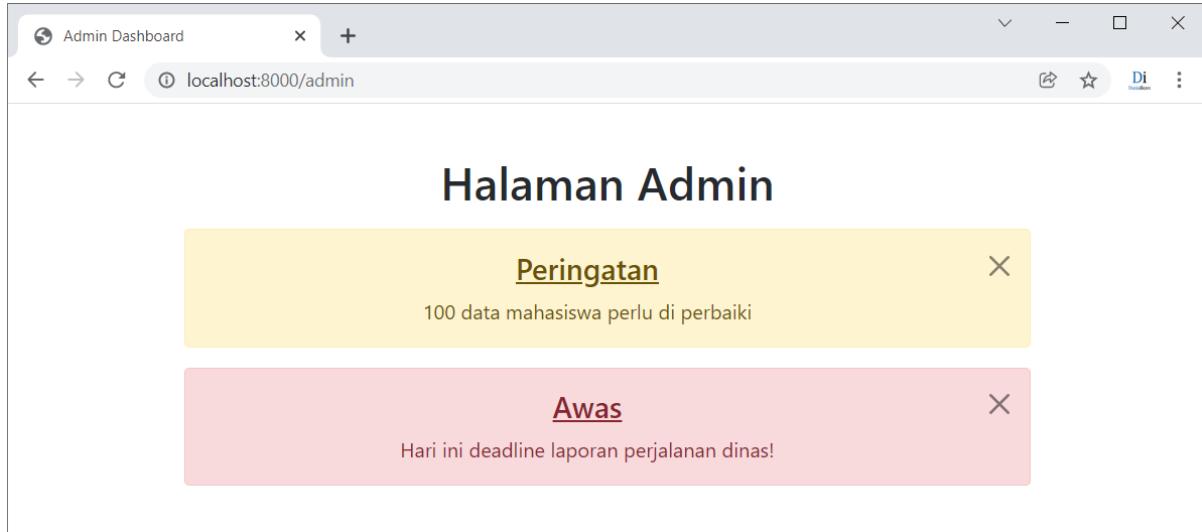
Kali ini terdapat tambahan tag `<h4>` di baris 2. Sebagai teks dari tag ini diisi variabel `>{{$judul}}`, dengan demikian komponen alert berisi 3 variabel: `$class`, `$judul` dan `$slots`.

Berikut cara pemanggilan alert di atas dari view admin:

```
resources/views/admin.blade.php
```

```
1 <div class="container text-center mt-3 p-4 bg-white">
2   <h1 class="mb-3">Halaman Admin</h1>
3   <div class="row">
4     <div class="col-12">
5
6       @component('layout.alert')
7         @slot('class')
8           warning
9         @endslot
10        @slot('judul')
11          Peringatan
12        @endslot
13        100 data mahasiswa perlu di perbaiki
14      @endcomponent
15
16      @component('layout.alert')
17        @slot('class')
18          danger
19        @endslot
20        @slot('judul')
21          Awas
22        @endslot
23        Hari ini deadline laporan perjalanan dinas!
24      @endcomponent
25
26    </div>
27 </div>
```

28 </div>



Gambar: Tampilan alert hasil pengisian 3 variabel slots

Selain teks untuk mengisi variabel `{{$slot}}`, sisanya harus ditulis di antara perintah `@slot('nama_variabel')` dan `@endslot`.

Penulisan dengan cara ini memang cukup panjang, oleh karena itu blade menyediakan cara alternatif dengan menulis semua teks variabel sebagai *associative array*:

resources/views/admin.blade.php

```
1  <div class="container text-center mt-3 p-4 bg-white">
2    <h1 class="mb-3">Halaman Admin</h1>
3    <div class="row">
4      <div class="col-12">
5
6        @component('layout.alert',[ 'class'=>'warning', 'judul'=>'Peringatan'])
7          100 data mahasiswa perlu di perbaiki
8        @endcomponent
9
10       @component('layout.alert',[ 'class'=>'danger', 'judul'=>'Awas'])
11         Hari ini deadline laporan perjalanan dinas!
12       @endcomponent
13
14       @component('layout.alert',[ 'class'=>'success', 'judul'=>'Kabar Baik'])
15         Bulan depan cuti panjang...
16       @endcomponent
17
18     </div>
19   </div>
20 </div>
```

Sekarang nilai setiap variabel (selain `$slots`) ditulis sebagai associative array yang ditempatkan pada argument kedua perintah `@component`. Key array berupa string yang merujuk ke nama variabel yang akan diisi, sedangkan value-nya berupa teks yang akan mengisi variabel

tersebut.

8.18. Named Routes

Materi kita kali ini tidak secara spesifik membahas blade, tapi tentang **route**. Namun karena sangat berhubungan dengan pemrosesan tampilan, saya akan bahas di sini.

Named routes adalah sebutan untuk route yang diberi "nama" atau "alias". Fitur ini diperlukan jika kita sering mengubah alamat route tapi tidak ingin mengganti satu per satu URL dari alamat tersebut di dalam blade. Pengertian ini akan lebih mudah jika dibahas dengan contoh.

Sebagai praktek, kita akan fokus ke bagian menu navigasi saja. Menu navigasi biasanya berisi link menuju halaman yang akan ditampilkan, yang di dalam Laravel alamat tersebut merujuk ke URL yang ditulis di dalam route.

Perhatikan potongan menu navigasi berikut:

resources/views/layout/master.blade.php

```

1 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
2   <div class="container">
3     <ul class="navbar-nav">
4       <li class="nav-item">
5         <a class="nav-link" href="/mahasiswa">Data Mahasiswa</a>
6       </li>
7       <li class="nav-item">
8         <a class="nav-link" href="/dosen">Data Dosen</a>
9       </li>
10      <li class="nav-item">
11        <a class="nav-link" href="/gallery">Gallery</a>
12      </li>
13    </ul>
14  </div>
15 </nav>
```

Alamat URL di dalam atribut href berisi nilai "/mahasiswa", "/dosen" dan "/gallery". Ini bersesuaian dengan route berikut:

routes/web.php

```

1 Route::get('/mahasiswa', function () {
2   $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                     "Lisa Permata"];
4   return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
5 });
6
7 Route::get('/dosen', function () {
8   $arrDosen = ["Maya Fitrianti, M.M.", "Prof. Silvia Nst, M.Farm.",
9                 "Dr. Umar Agustinus", "Dr. Syahrial, M.Kom."];
10    return view('dosen')->with('dosen', $arrDosen);
11});
```

```
12
13 Route::get('/gallery', function () {
14     return view('gallery');
15 });
```

Ketiga route ini sudah kita pakai sepanjang pembahasan materi tentang blade layout. Namun karena sesuatu hal, saya ingin mengubah alamat URL route 'gallery' menjadi sebagai berikut:

routes/web.php

```
1 Route::get('universitas/fmipa/matematika/gallery', function () {
2     return view('gallery');
3 });
```

Dengan perubahan ini, maka isi atribut href di dalam tag <a> menu navigasi juga harus diubah:

resources/views/layout/master.blade.php

```
1 ...
2     <li class="nav-item">
3         <a class="nav-link" href="universitas/fmipa/matematika/gallery">Gallery</a>
4     </li>
5 ...
```

Inilah yang dimaksud bahwa jika alamat URL route diubah, maka semua isi atribut href di dalam tag <a> yang menuju route tersebut juga harus diubah. Bayangkan jika alamat '/gallery' terdapat di berbagai tempat, maka harus diubah satu per satu.

Named route bisa menjadi solusi untuk masalah ini. Caranya, tambahkan pemanggilan method name() di akhir route, seperti contoh berikut:

routes/web.php

```
1 Route::get('universitas/fmipa/matematika/gallery', function () {
2     return view('gallery');
3 })->name('gambar');
```

Perhatikan di baris 3, tambahan ->name('gambar') dipakai untuk membuat alias dari route gallery dengan nama 'gambar'.

Dengan tambahan ini, kita bisa menggunakan function helper route() sebagai nilai dari atribut href:

resources/views/layout/master.blade.php

```
1 ...
2     <li class="nav-item">
3         <a class="nav-link" href="{{route('gambar')}}">Gallery</a>
4     </li>
5 ...
```

Perintah href="{{route('gambar')}}" artinya Laravel akan men-generate alamat URL dari

route yang memiliki *named parameter* 'gambar'.

Sekarang, jika alamat URL route kita ubah lagi menjadi alamat lain, secara otomatis nilai dari atribut href juga akan menyesuaikan.

Relative URL vs Absolute URL

Function `route()` yang baru saja kita pakai akan meng-generate **absolute URL**, yakni alamat URL lengkap dengan nama domain. Dalam praktik kita secara offline, domain yang dimaksud adalah `http://localhost:8000`.

Jika URL ditulis langsung misalnya sebagai `href="/mahasiswa"`, maka itu adalah **relative URL**, yakni alamat URL yang tidak disertai dengan nama domain.

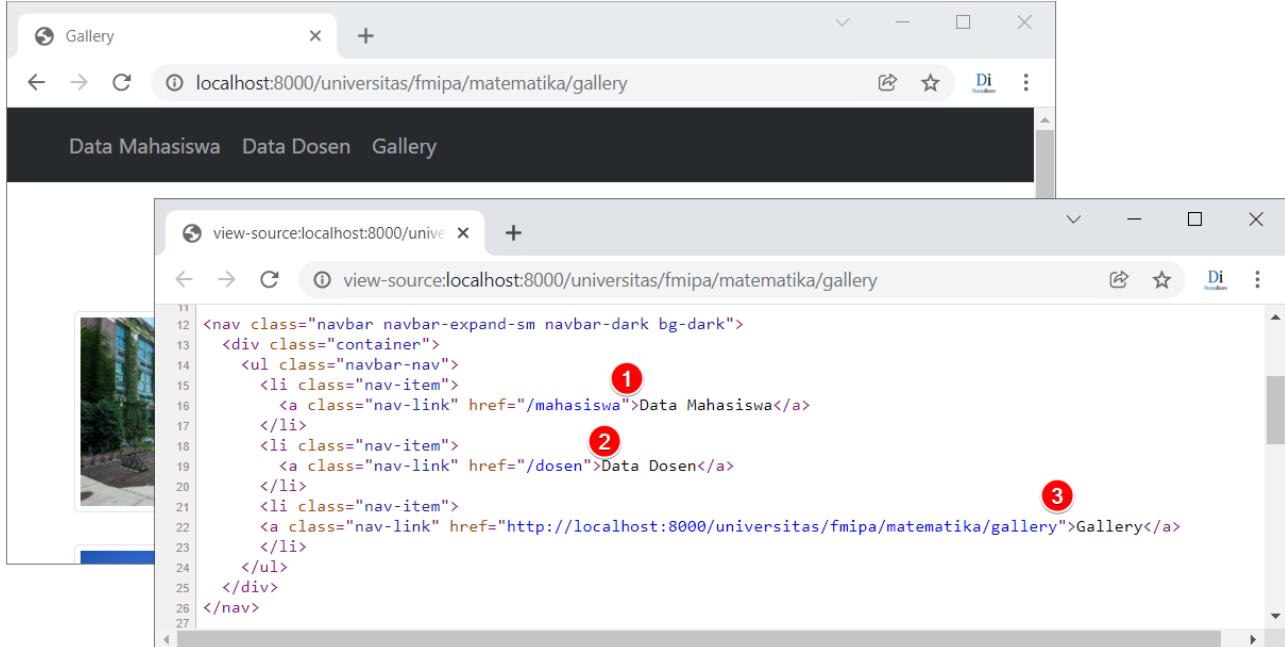
Perhatikan kode menu navigasi berikut:

`resources/views/layout/master.blade.php`

```
1 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
2   <div class="container">
3     <ul class="navbar-nav">
4       <li class="nav-item">
5         <a class="nav-link" href="/mahasiswa">Data Mahasiswa</a>
6       </li>
7       <li class="nav-item">
8         <a class="nav-link" href="/dosen">Data Dosen</a>
9       </li>
10      <li class="nav-item">
11        <a class="nav-link" href="{{route('gambar')}}">Gallery</a>
12      </li>
13    </ul>
14  </div>
15 </nav>
```

Link ke view mahasiswa dan dosen masih menggunakan cara biasa, yakni `href="/mahasiswa"` dan `href="/dosen"`. Sedangkan menu Gallery menggunakan `href="{{route('gambar')}}`.

Berikut hasil source code HTML yang terlihat dari web browser:



Gambar: Source code tampilan menu navigasi

Perhatikan hasil dari atribut href. Untuk menu Data Mahasiswa dan Data Dosen, atribut href berisi "/mahasiswa" (1) dan "/dosen" (2). Ini sesuai dengan yang kita tulis di dalam view.

Sedangkan untuk Gallery, atribut href berisi alamat URL lengkap "http://localhost:8000/universitas/fmipa/matematika/gallery" (3). Inilah yang dimaksud dengan absolute URL.

Baik relative URL dan absolute URL sama-sama bisa dipakai. Biasanya relative URL lebih disukai karena lebih fleksibel dan alamat URL tidak terikat dengan domain tertentu. Namun menggunakan Laravel, kita juga bisa dengan mudah meng-generate absolute URL.

Agar seragam, mari modifikasi semua route menggunakan *named parameter*:

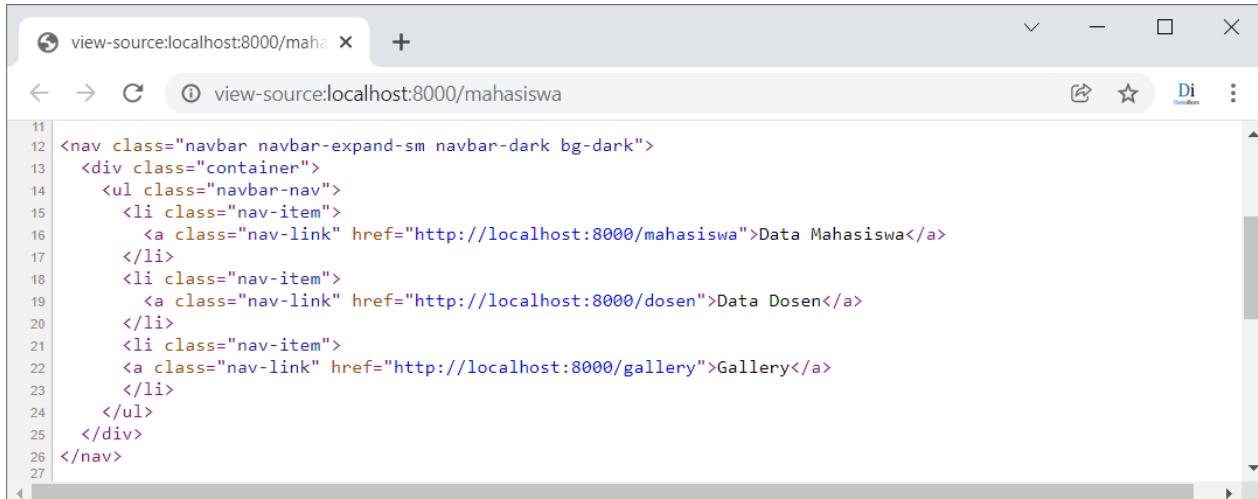
routes/web.php

```
1 Route::get('mahasiswa', function () {
2   $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                     "Lisa Permata"];
4   return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
5 })->name('mahasiswa');
6
7 Route::get('dosen', function () {
8   $arrDosen = ["Maya Fitrianti, M.M.", "Prof. Silvia Nst, M.Farm.",
9                 "Dr. Umar Agustinus", "Dr. Syahrial, M.Kom."];
10  return view('dosen')->with('dosen', $arrDosen);
11 })->name('dosen');
12
13 Route::get('gallery', function () {
14   return view('gallery');
15 })->name('gallery');
```

Dengan perubahan ini, berikut kode view untuk menu navigasi:

resources/views/layout/master.blade.php

```
1 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
2   <div class="container">
3     <ul class="navbar-nav">
4       <li class="nav-item">
5         <a class="nav-link" href="{{route('mahasiswa')}}">Data Mahasiswa</a>
6       </li>
7       <li class="nav-item">
8         <a class="nav-link" href="{{route('dosen')}}">Data Dosen</a>
9       </li>
10      <li class="nav-item">
11        <a class="nav-link" href="{{route('gallery')}}">Gallery</a>
12      </li>
13    </ul>
14  </div>
15 </nav>
```



Gambar: Source code tampilan menu navigasi menggunakan named route

Dengan penulisan seperti ini, tidak masalah jika alamat URL di route berubah, semua link juga akan mengikuti perubahan tersebut.

Named Routes Parameter

Beberapa route bisa memiliki parameter, dan Laravel juga menyediakan solusi untuk membuat *named route* dengan tambahan parameter ini.

Misalkan terdapat route sebagai berikut:

routes/web.php

```
1 Route::get('informasi/{fakultas}/{jurusan}', function ($fakultas, $jurusan) {
2   $data = [$fakultas, $jurusan];
3   return view('informasi')->with('data', $data);
4 })->name('info');
```

Route di atas memiliki 2 buah parameter: `$fakultas` dan `$jurusan` yang akan dikirim ke dalam

view 'informasi'. Di baris 4 terdapat penulisan *named route* dengan nama alias 'info'.

Dan berikut kode program dari view 'informasi':

resources/views/informasi.blade.php

```
1 @extends('layout.master')
2 @section('title','Informasi Fakultas')
3
4 @section('content')
5 <div class="container text-center mt-3 p-4 bg-white">
6   <h1>Informasi Fakultas {{$data[0]}}, Jurusan {{$data[1]}}</h1>
7 </div>
8 @endsection
```

Saya menggunakan struktur yang sama seperti view kita selama ini, yaitu dengan meng-
extends `layout.master`. Isi dari view `informasi` juga hanya sebuah judul yang menampilkan
teks hasil parameter.

Sebagai contoh, jika diakses `http://localhost:8000/informasi/FMIPA/Matematika`, maka akan
tampil teks 'Informasi Fakultas FMIPA, Jurusan Matematika'. Atau jika diakses `http://`
`localhost:8000/informasi/Ekonomi/Manajemen`, maka akan tampil teks 'Informasi Fakultas
Ekonomi, Jurusan Manajemen'.



Gambar: Tampilan halaman informasi

Jika ingin membuat menu navigasi untuk halaman `http://localhost:8000/informasi/`
`Ekonomi/Manajemen`, bisa menggunakan alamat URL relatif sebagai berikut:

resources/views/layout/master.blade.php

```
1 ...
2   <li class="nav-item">
3     <a class="nav-link" href="/informasi/Ekonomi/Manajemen">Info</a>
4   </li>
5 ...
```

Namun bagaimana dengan versi *named route*? Caranya, tulis nama parameter sebagai

associative array pada saat pemanggilan function route().

Misalkan saya ingin membuat menu ke alamat `http://localhost:8000/informasi/FMIPA/Matematika`, maka penulisan atribut href untuk menu navigasi bisa seperti ini:

`resources/views/layout/master.blade.php`

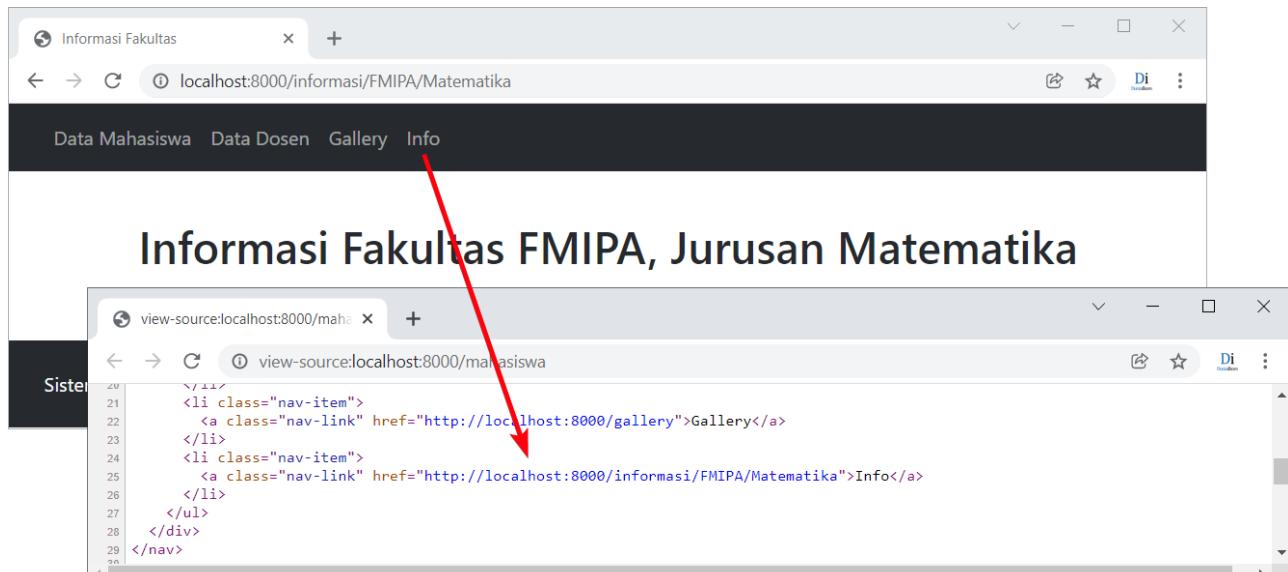
```
1 ...  
2     <li class="nav-item">  
3         <a class="nav-link" href="{{route('info',['fakultas' => 'FMIPA',  
4             'jurusan' => 'Matematika'])}}>Info</a>  
5     </li>  
6 ...
```

Di baris 3, function route() ditulis sebagai berikut:

```
route('info',['fakultas' => 'FMIPA', 'jurusan' => 'Matematika'])
```

Argumen pertama, yakni 'info' merupakan named route yang ditulis di file `routes/web.php`. Dan untuk argumen kedua, berupa associative array untuk parameter `$fakultas` dan `$jurusan`.

Berikut hasil penulisan menu ini di web browser:



Gambar: Hasil URL yang di generate dari named route + parameter

Jika ingin membuat URL ke alamat lain, tinggal mengubah isi associative array ini.

8.19. Asset dan Url Function Helper

Masih terkait dengan penulisan alamat URL, Laravel juga menyediakan function helper `asset()` dan `url()`.

Function `asset()` berguna untuk membuat absolute URL ke folder **public**, yakni folder dimana

file assets berada. Sebagai contoh, untuk mengakses file CSS external dalam file `master.blade.php`, sebelumnya saya menulis sebagai berikut:

```
resources/views/layout/master.blade.php
```

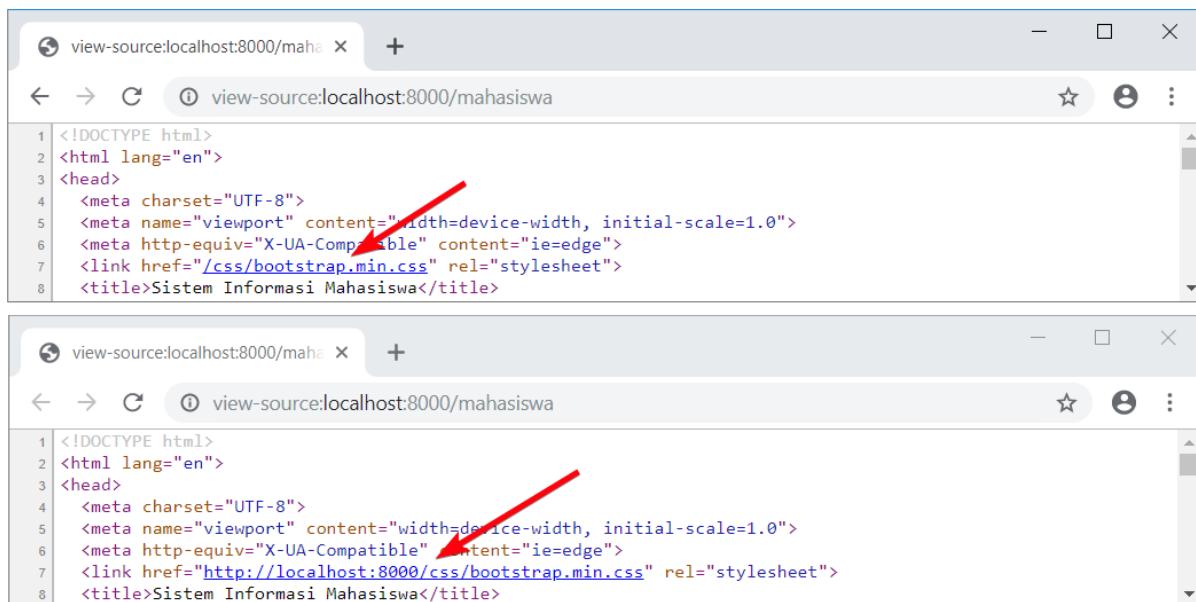
```
1 ...  
2 <link href="/css/bootstrap.min.css" rel="stylesheet">  
3 ...
```

Ini penulisan alamat relatif, dan bisa diubah menjadi alamat absolute dengan function `asset()`:

```
resources/views/layout/master.blade.php
```

```
1 ...  
2 <link href="{{ asset('/css/bootstrap.min.css') }}" rel="stylesheet">  
3 ...
```

Argument untuk function `asset()` berupa path file di dalam folder **public**.



Gambar: Hasil relatif URL (atas) dan absolute URL dari function `asset()` (bawah)

Seandainya saya memiliki gambar di folder `public\img\people1.jpg`, maka penulisan tag `` bisa sebagai berikut:

```

```

Sedangkan function `url()` dipakai untuk membuat alamat URL secara umum. Argument fungsi ini berupa string path yang relatif terhadap domain.

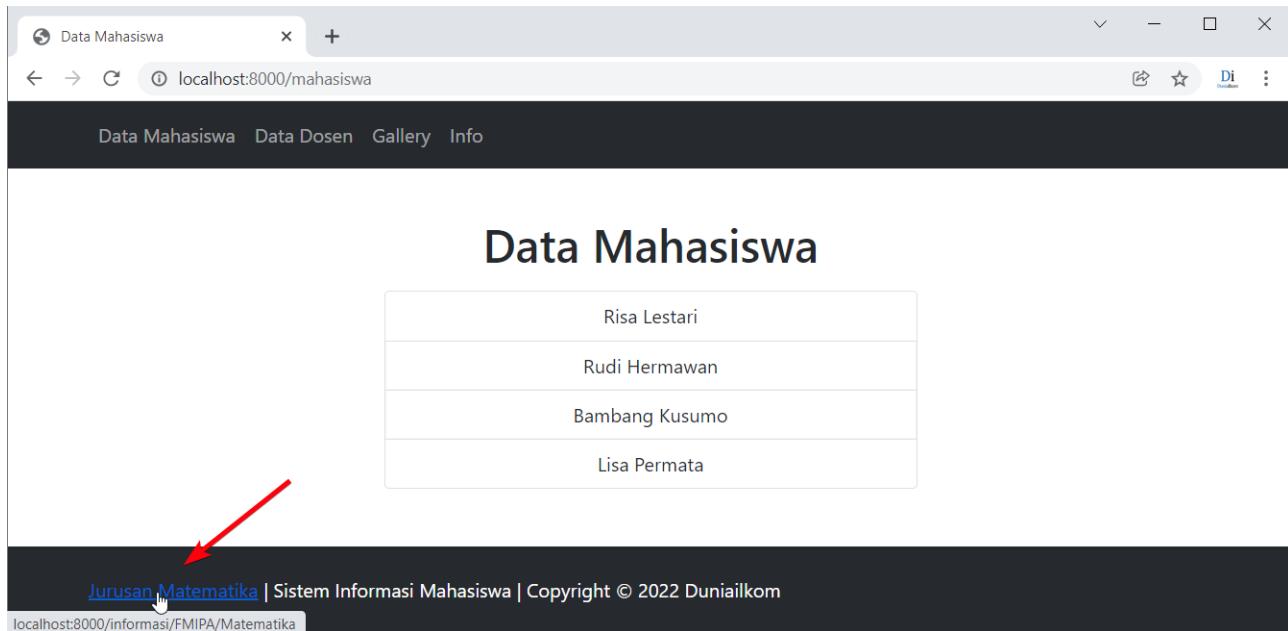
Sebagai contoh, saya akan tambah link berikut ke dalam bagian footer file `master.blade.php`:

```
resources/views/layout/master.blade.php
```

```
1 <footer class="bg-dark py-4 text-white mt-4">  
2 <div class="container">  
3   <a href="{{ url('informasi/ FMIPA/Matematika') }}">Jurusan Matematika</a> |
```

Blade Template Engine

```
4      Sistem Informasi Mahasiswa | Copyright © {{ date("Y") }} DuniaIlkom
5  </div>
6  </footer>
```



Gambar: Link di bagian footer

Di baris 3 terdapat penggunaan function helper `url('informasi/FMIPA/Matematika')` sebagai nilai atribut `href`. Ini akan diproses oleh blade menjadi:

```
<a href="http://localhost:8000/informasi/FMIPA/Matematika">Jurusan Matematika</a>
```

Sebagai rangkuman, function `route()`, `asset()` dan `url()` sama-sama berguna untuk meng-generate URL. Bedanya, `route()` dipakai untuk *named route*, `asset()` untuk link ke file assets, dan `url()` untuk membuat URL secara umum. Ketiga function ini bisa dipakai tergantung kebutuhan.

8.20. VS Code Laravel Extension Pack

Pembahasan tentang blade kita tutup dengan materi opsional. Disebut opsional karena ini lebih ke tambahan saja (tidak wajib).

Di dalam teks editor VS Code, kita bisa menginstall berbagai fitur tambahan yang disebut extension. Mayoritas extension dibuat oleh programmer individu untuk membantu programmer lain. Salah satu extension tersebut adalah **Laravel Extension Pack**.

Laravel Extension Pack sebenarnya merupakan 'paket' yang terdiri dari 10 extension:

- ◆ Laravel Blade Snippets
- ◆ Laravel Snippets

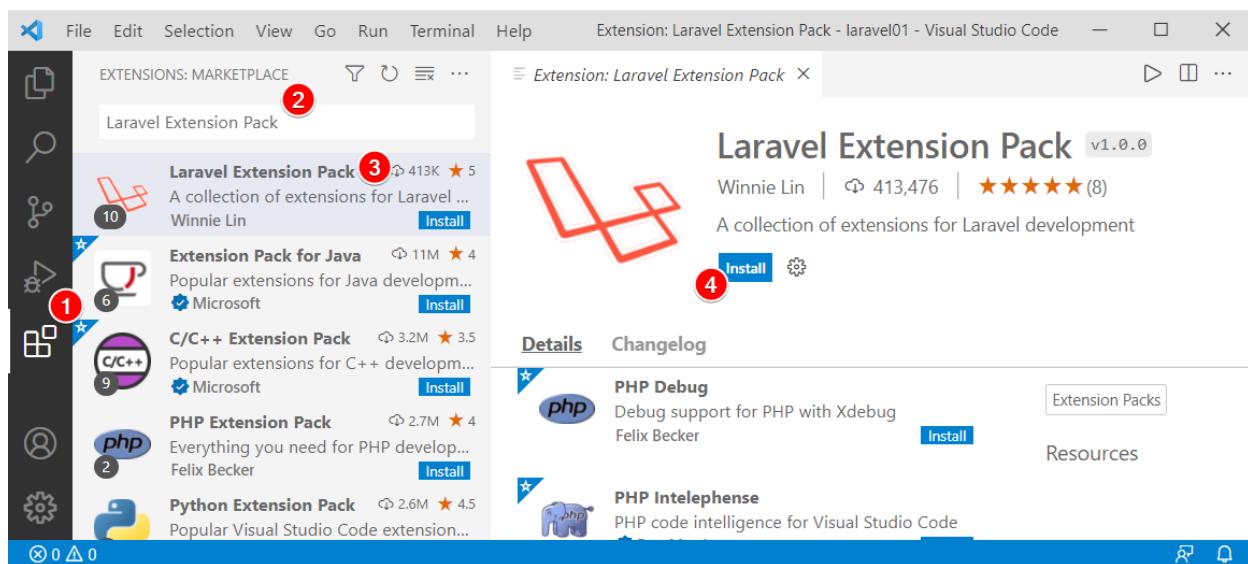
- ◆ Laravel Artisan
- ◆ Laravel goto view
- ◆ laravel-goto-controller
- ◆ Laravel Extra Intellisense
- ◆ DotENV
- ◆ EditorConfig for VS Code
- ◆ PHP Debug
- ◆ PHP Intelephense

Dengan menginstall Laravel Extension Pack, kita sebenarnya menginstall 10 extension ini sekaligus. Berikut penjelasan dari beberapa extension ini:

[Laravel Blade Snippets](#) berfungsi untuk memudahkan penulisan perintah blade. Perintah blade bukan bagian dari web programming seperti HTML atau PHP, sehingga fitur pewarnaan kode program (*syntax highlighting*) dari VS Code tidak aktif. Dengan menginstall extension ini, VS Code jadi bisa memahami perintah blade serta menyediakan shortcut dari kode blade.

[Laravel Snippets](#) berfungsi untuk menambahkan *code completion* dari berbagai perintah Laravel. [Laravel Artisan](#) dipakai untuk menjalankan perintah artisan dari dalam VS Code. [Laravel goto view](#) untuk berpindah antar file laravel dengan mudah, serta DotENV untuk mewarnai file konfigurasi .env.

Karena Laravel Extension Pack merupakan paket kumpulan extension, bisa jadi ke depannya akan ditambah dengan extension lain, atau di update ke versi yang lebih baru. Penjelasan lebih lanjut dari setiap extension, bisa klik link di atas.



Gambar: Cara instalasi Laravel Extension Pack

Untuk menginstall extension ini, silahkan klik icon kotak di kiri tampilan jendela VS Code (1). Ini adalah shortcut untuk ke library extension VS Code. Cara lain bisa juga dengan menekan

kombinasi tombol Ctrl + Shift + X.

Kemudian ketik 'Laravel Extension Pack' di text box (2). Akan keluar daftar berbagai extension, lalu pilih Laravel Extension Pack (3) dan klik tombol install (4). Proses instalasi akan berlangsung beberapa saat, pastikan komputer anda terhubung ke internet.

Setelah proses instalasi selesai, tutup jendela extension ini lalu tes buka salah satu file blade. Di sini akan terlihat bahwa perintah blade sekarang sudah berwarna, yang menandakan fitur syntax highlighting telah aktif.

Selain itu, extension ini juga memiliki shortcut untuk berbagai struktur perintah blade. Caranya, awali perintah yang kita ketik dengan "b:" (huruf b yang diikuti dengan tanda titik dua), akan tampil daftar perintah blade yang bisa dipilih. Jika salah satunya di klik, struktur perintah tersebut akan dibuat.

```
25     @component('layout.alert',[ 'class'=>'success', 'judul'=>'Kabar Baik'])
26         Bulan depan cuti panjang...
27     @endcomponent
28
29     b:|          □ b:echo
30         □ b:echo-raw
31         □ b:empty
32         □ b:extends
33         □ b:for
34         □ b:foreach
35         □ b:forelse
```

Gambar: Tampilan fitur shortcut perintah blade dari Laravel Blade Snippets extension VS Code

Fitur Laravel Blade Snippets akan memudahkan kita dalam menulis view Laravel. Jika anda menggunakan text editor lain seperti Atom atau Sublime Text, sangat mungkin juga tersedia extension serupa. Tapi kembali, ini bukan hal yang wajib dilakukan.

Bab ini memang cukup panjang, mencakup 50 halaman lebih yang hanya membahas tentang blade. Namun saya rasa ini cukup penting karena blade-lah yang bertanggung jawab dalam mengurus tampilan front-end Laravel.

Sebagai pengingat, blade memiliki 2 fitur utama: mempersingkat penulisan perintah PHP serta pemecahan file template (proses pembuatan layout).

Bab berikutnya masih berhubungan dengan front-end, dimana kita akan membahas **Laravel Mix**.

9. Laravel Mix (Compiling Assets)

Laravel berusaha menjadi sebuah *web development framework* yang lengkap. Maksudnya, Laravel tidak hanya memikirkan bagaimana cara membuat kode PHP yang baik, tapi juga bagaimana memproses kode CSS dan JavaScript yang ideal sesuai dengan standar modern.

Di akhir bab tentang view kita pernah bahas cara menginput file CSS dan JavaScript external ke dalam folder **public**. Cara tersebut memang bisa dipakai tapi masih kurang ideal. Dalam bab kali ini kita akan bahas cara yang lebih disarankan, yakni menggunakan **Laravel Mix**.

Laravel Mix sebenarnya bisa di kategorikan sebagai materi level *advanced*, karena sudah gabungan berbagai teknologi web development, terutama **Nodejs** (yup, kita akan menginstall Nodejs sebentar lagi).

Meskipun akan terasa agak sedikit rumit, menurut saya materi ini cukup penting karena sudah menjadi standar di industri *web development*.

Seperti biasa, agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder baru, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

9.1. Pengertian Laravel Mix

Laravel Mix adalah sebuah fitur untuk mengkompilasi file assets (bahasa inggris: *compiling assets*), di antara proses yang bisa dilakukan adalah: penggabungan, memproses kode *pre-processor*, *minification* dan *cache busting*.

Penggabungan Assets

Penggabungan assets yang dimaksud adalah menggabungkan beberapa file CSS dan JavaScript menjadi 1 file saja.

Dalam project yang besar, biasanya kita punya berbagai file CSS dan JavaScript, terutama dari library tambahan seperti Bootstrap, jQuery, dsb. Tidak jarang jumlahnya bisa mencapai puluhan, bahkan ratusan file. Ini tidak efisien karena ketika halaman web dibuka, terjadi

banyak proses request ke server, yakni 1 kali untuk setiap file.

Jika website kita punya 10 file CSS dan 7 file JavaScript, artinya terdapat 17 kali proses request ke server. Memang setiap proses mungkin cuma butuh waktu sepersekian detik, tapi tetap saja bukan cara ideal. Dengan menggunakan Laravel Mix, kita bisa menggabung semua file CSS dan JS ini ke dalam 1 file (yakni 1 file untuk CSS dan 1 file untuk JS), sehingga request ke server menjadi lebih sedikit.

Memproses Kode Pre-processor

Pre-processor adalah istilah untuk menyebut program yang menghasilkan inputan untuk program lain (untuk program utama). Biasanya ini diperlukan karena keterbatasan fitur di program utama.

Sebagai contoh, [Sass](#) adalah sebuah CSS *pre-processor*. Maksudnya, kita menulis dahulu kode dengan "bahasa" Sass, yang nantinya akan menghasilkan kode CSS. Kenapa harus menggunakan Sass? Karena di dalam Sass terdapat banyak fitur programming yang tidak tersedia di CSS biasa, seperti variabel, perulangan, if else, dsb.

Alternatif lain dari CSS pre-processor adalah [Less](#), [CSS-Crush](#), dan [Stylus](#).

Demikian pula, [TypeScript](#) merupakan salah satu JavaScript pre-processor. Fungsinya untuk mengatasi kekurangan di dalam JavaScript. Jika anda pernah mendengar framework JavaScript [Angular](#), di dalamnya menggunakan TypeScript.

Alternatif lain dari JavaScript pre-processor adalah [Coffeescript](#), [LiveScript](#) dan [Babel](#).

HTML pun punya pre-processor, diantaranya [Pug](#) (dulu dikenal sebagai Jade), [Haml](#), serta [Markdown](#). Dan, masih ingat kepanjangan dari PHP? Yup, **PHP: Hypertext Preprocessor**. Sehingga PHP pun merupakan pre-processor dari HTML.

Laravel mix bisa dipakai untuk memproses file asal (kode *pre-processor*), menjadi file CSS dan JavaScript biasa.

Merasa "kewalahan" dengan semua teknologi ini? Saya juga demikian. Semakin masuk kita ke materi advanced, pilihannya juga makin beragam.

Nyaris mustahil untuk bisa menguasai semua ini. Sehingga pada satu titik kita harus membatasi apa yang harus dipelajari. Bagi yang sedang belajar web programming untuk mencari kerja, batasannya adalah: "fokus ke skill yang sering di tulis pada syarat lowongan pekerjaan".

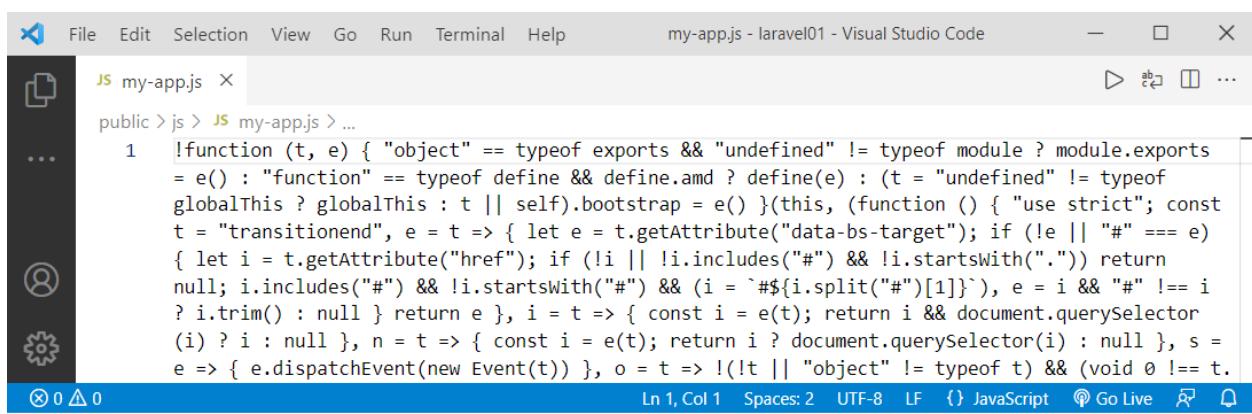
Setelah itu, baru sesuaikan dengan tuntutan di tempat kerja nanti. Jika ternyata client perlu teknologi JavaScript terbaru, maka silahkan perdalam materi advanced JavaScript. Apabila di kantor kekurangan skill CSS, bisa dipertimbangkan lanjut pelajari Sass.

File Minification

Jika anda pernah mempelajari framework CSS seperti **Bootstrap** atau library JavaScript seperti **jQuery**, tentu tidak asing dengan istilah ini.

File minification adalah teknik untuk men-compress ukuran file kode program dengan cara menghilangkan semua baris komentar serta menghapus spasi yang dianggap tidak perlu. Beberapa teknik minification sampai mengubah nama variabel yang panjang menjadi 1 huruf saja. Dengan ukuran yang lebih kecil (dibandingkan sebelum di minified), proses download file juga jadi lebih cepat.

File CSS atau JavaScript yang sudah mengalami proses *minification* nyaris tidak bisa dibaca, setidaknya oleh kita. Tapi komputer tidak mengalami kesulitan memproses kode tersebut.



The screenshot shows a Visual Studio Code interface with a single tab open for 'my-app.js'. The code is highly compressed and unreadable by humans. The file path 'my-app.js - laravel01 - Visual Studio Code' is visible at the top. The status bar at the bottom shows 'Ln 1, Col 1' and other standard editor information.

```
public > js > JS my-app.js > ...
1 !function (t, e) { "object" == typeof exports && "undefined" != typeof module ? module.exports = e() : "function" == typeof define && define.amd ? define(e) : (t = "undefined" != typeof globalThis ? globalThis : t || self).bootstrap = e() }(this, (function () { "use strict"; const t = "transitionend", e = t => { let e = t.getAttribute("data-bs-target"); if (!e || "#" === e) { let i = t.getAttribute("href"); if (!i || !i.includes("#") && !i.startsWith(".")) return null; i.includes("#") && !i.startsWith("#") && (i = `#${i.split("#")[1]}`), e = i && "#" !== i ? i.trim() : null } return e }, i = t => { const i = e(t); return i && document.querySelector(i) ? i : null }, n = t => { const i = e(t); return i ? document.querySelector(i) : null }, s = e => { e.dispatchEvent(new Event(t)) }, o = t => !(t || "object" != typeof t) && (void 0 !== t).
```

Gambar: Contoh isi kode program hasil minification

Terdapat berbagai tools dan aplikasi untuk membuat file *minification*, misalnya dari website layanan online seperti cssminifier.com atau javascript-minifier.com. Caranya, copy-paste kode CSS atau JavaScript ke web tersebut, lalu klik tombol **minify**. Beberapa saat kemudian kode versi minified sudah bisa dipakai.

Laravel Mix juga bisa dipakai untuk menghasilkan file minifier.

Cache Busting (Versioning)

Untuk mempercepat proses tampilan website, sebagian besar web browser menggunakan fitur *cache*. Cara kerjanya adalah, ketika web di buka pertama kali, sebagian file akan disimpan ke dalam cache yang berada di komputer pengunjung.

Ketika web dibuka untuk kali kedua (dan seterusnya), web browser akan memeriksa cache terlebih dahulu. Jika data lama masih tersimpan di cache, maka tinggal ambil file tersebut, tidak perlu mendownload ulang dari server.

Dengan sedikit konfigurasi di bagian HTTP Header, pemilik web bisa menentukan berapa lama sebuah file bisa tersimpan di dalam cache (*expired time*). Untuk mempercepat proses loading, waktu expired ini bisa di-set untuk jangka waktu yang panjang, hingga tahunan. Trik ini juga salah satu cara yang sering dipakai untuk meningkatkan kinerja website.

Namun mengatur expired time yang cukup lama menjadi masalah tersendiri saat kita ingin melakukan update assets (terutama file CSS dan JS). Web browser akan terus me-load file dari cache sampai waktu expired habis. Akibatnya tampilan web kita tidak terupdate di web browser pengunjung.

Salah satu solusi dari masalah ini adalah dengan menambah angka acak di belakang file assets menggunakan *query string*. Tujuannya untuk memaksa web browser mendownload ulang file assets karena menganggap ini adalah file baru, sehingga tidak mengambil apa yang sudah ada di cache.

Contoh penulisannya sebagai berikut:

```
<link rel="stylesheet" href="style.css?v=3.4.1">
```

Tambahan tanda `?v=3.4.1` di akhir file CSS dipakai untuk menghindari efek *cache*. Jika nanti terjadi update lagi di file CSS, tinggal ubah angka versioning ini misalnya menjadi `style.css?v=3.4.2` atau `style.css?v=3.4.3`.

Teknik ini dikenal juga dengan istilah **cache busting** atau **file versioning**. Dengan Laravel mix, kita bisa menggenerate angka versioning untuk file CSS dan JavaScript secara otomatis.

9.2. Instalasi Node.js

Secara teknis, Laravel mix merupakan sebuah *package* dari **Node.js**, yang artinya tidak dibuat menggunakan PHP, melainkan JavaScript. Laravel mix pun bisa dipakai terpisah dari framework Laravel.

Dari halaman resminya di [laravel mix.com](https://laravel-mix.com) tertulis bahwa "*Laravel mix is an elegant wrapper around Webpack for the 80% use case*".

Terjemahan bebas: "*Laravel mix adalah cara praktis untuk menggunakan Webpack yang bisa dipakai dalam 80% keperluan*". [Webpack](#) sendiri merupakan sebuah *module loader*, yang menjadi inti dari Laravel mix.

Huff,.. sampai di sini saya yakin anda semakin bingung, karena banyak istilah baru yang bermunculan. Karena itulah Laravel Mix termasuk materi yang cukup advanced.

Namun untuk bisa menggunakan Laravel dengan baik, mau tidak mau kita harus membahas ini semua. Jika terdapat istilah yang kurang di pahami, dengan berat hati bisa "diterima saja apa adanya", karena memang butuh penjelasan yang cukup panjang untuk bisa memahami semua teknologi yang ada. Alternatif lain bisa juga di googling untuk penjelasan lebih lanjut.

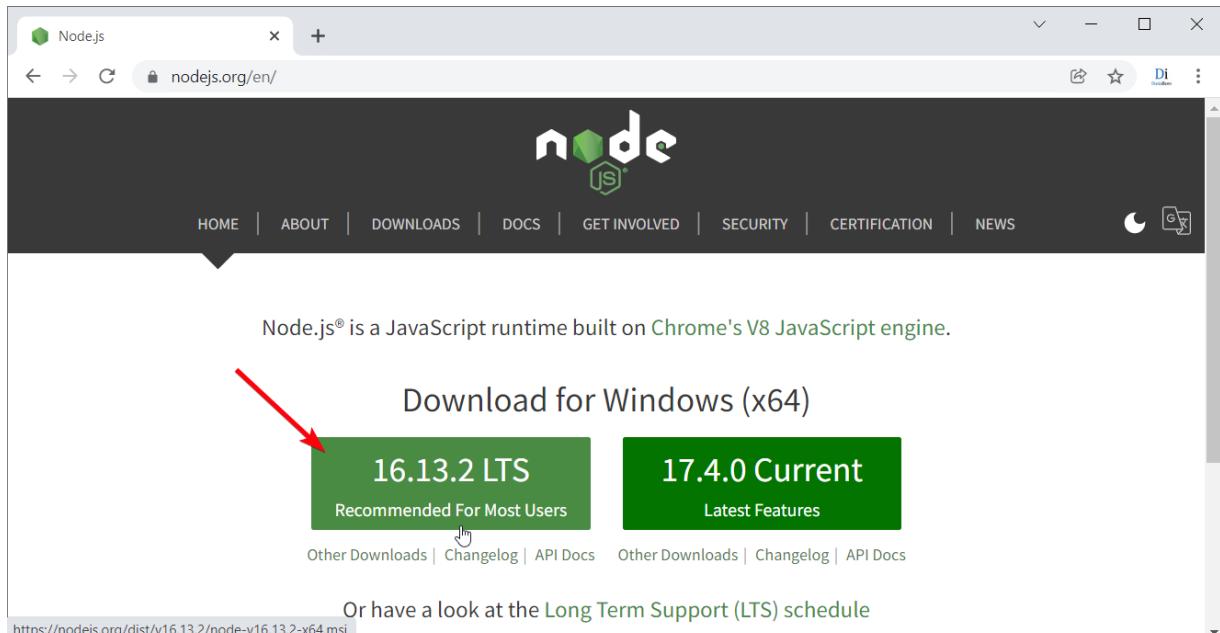
Sebagai kesimpulan sederhana, Laravel mix adalah aplikasi *compiling assets* yang berbasis **Webpack** dan berjalan di **Node.js**. Karena Larevel mix menggunakan Node.js, maka kita harus

install aplikasi ini.

Node.js sendiri merupakan *platform* atau teknologi untuk membuat JavaScript bisa berjalan di web server, kurang lebih bisa menggantikan peran PHP. Namun di sini kita tidak akan membahas Node.js lebih lanjut (yang memang sangat luas dan bisa jadi 1 buku tersendiri).

Kita memerlukan Node.js agar bisa mengakses **npm** (node package manager), yang diperlukan oleh Laravel mix.

Baik, langsung saja kita install Node.js, silahkan buka web resminya di nodejs.org.



Gambar: Tampilan web nodejs.org

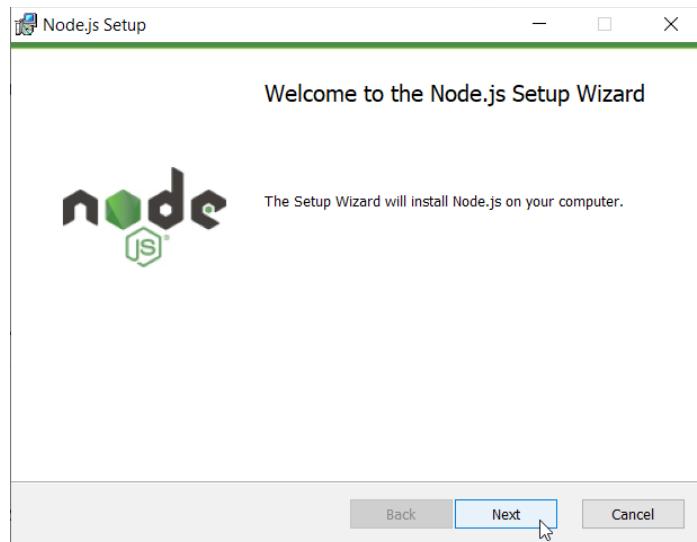
Di halaman awal ini kita bisa langsung download file instalasi. Website Node.js bisa mendekripsi sistem operasi yang digunakan, karena saya memakai Windows 10 64-bit, maka langsung diarahkan untuk memilih salah satu dari versi 16.13.2 atau 17.4.0. Kemungkinan besar versi yang anda dapatkan akan lebih baru dari ini.

Dua jenis versi Node.js berkaitan dengan support bug. Sama seperti Laravel, Node.js juga memiliki versi **LTS (Long Term Support)**, yakni versi stabil dengan dukungan update yang lebih lama, atau versi terbaru. Anda boleh memilih versi yang mana saja, karena yang kita perlukan hanya perintah dasar Node.js.

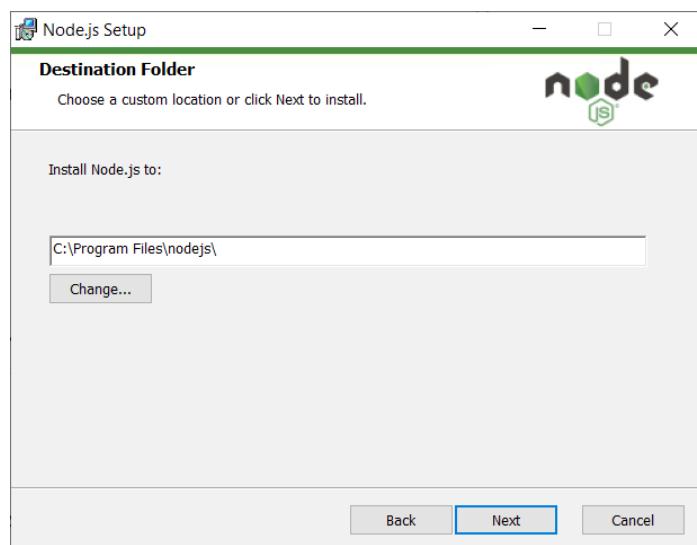
Saya akan memilih versi 16.13. Klik tombol warna hijau, dan proses download file akan berlangsung. File **node-v16.13.2-x64.msi** yang saya download berukuran sekitar 26 MB.

Setelah itu, double klik file instalasi dan ikuti prosesnya. Tidak ada pengaturan yang perlu diubah, cukup klik tombol next beberapa kali sampai selesai.

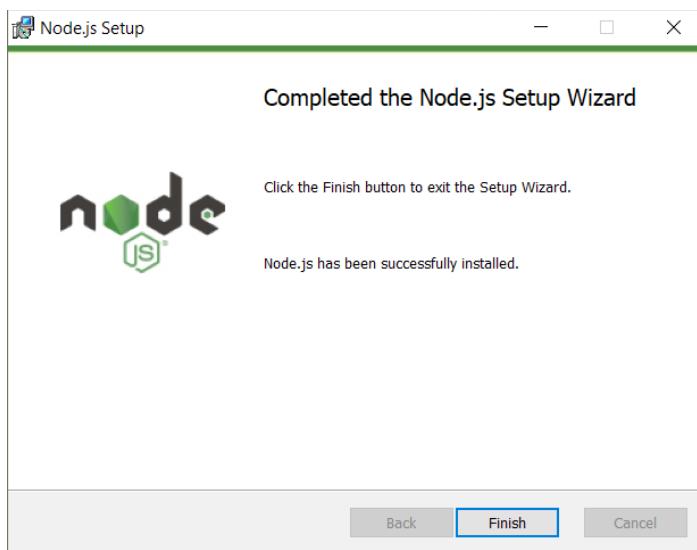
Laravel Mix (Compiling Assets)



Gambar: Jendela awal proses instalasi Node.js



Gambar: Node.js akan diinstall di C:\Program Files\nodejs



Gambar: Proses instalasi Node.js sudah selesai

Setelah proses instalasi selesai, mari kita coba cek apakah Node.js sudah bisa diakses dari cmd atau belum. Silahkan buka cmd, lalu ketik perintah berikut:

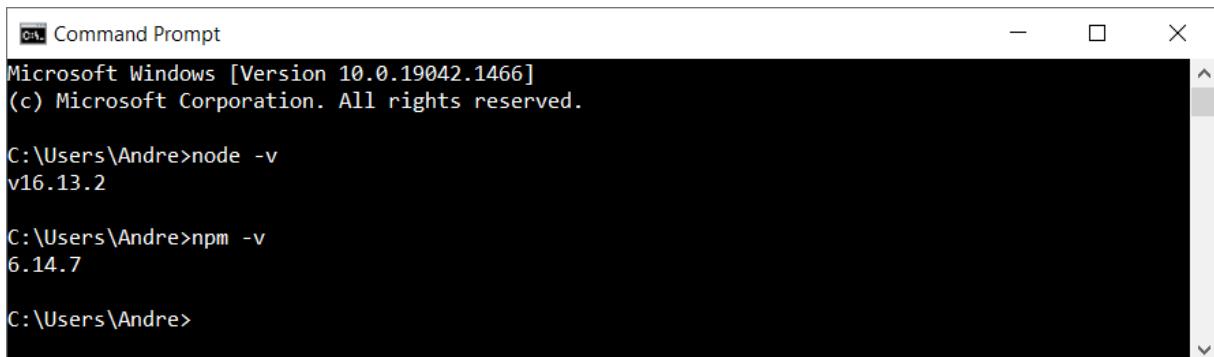
```
node -v
```

Seharusnya akan tampil angka yang berisi versi dari Node.js.

Kemudian ketik lagi

```
npm -v
```

Dan kembali, seharusnya juga tampil angka yang merupakan versi dari npm.



```
Command Prompt
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

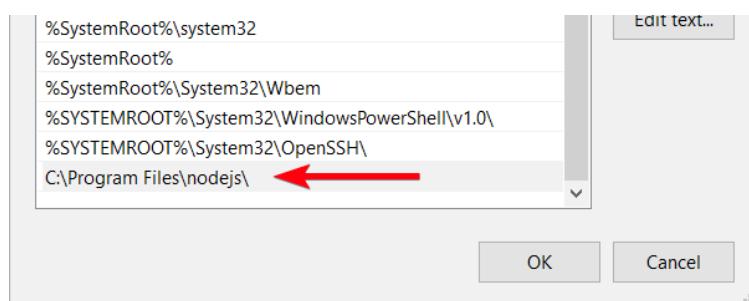
C:\Users\Andre>node -v
v16.13.2

C:\Users\Andre>npm -v
6.14.7

C:\Users\Andre>
```

Gambar: Periksa versi Node.js dan npm yang terinstall

Apabila kedua perintah ini tidak dikenali, cek pengaturan *system environment variable*, pastikan folder C:\Program Files\nodejs\ sudah terdaftar.



Gambar: List folder di *system environment variable* Windows

Mengenal npm

Pada saat kita menginstall Node.js, terdapat aplikasi lain yang juga ikut terinstall, yakni **npm** (node package manager). [Npm](#) adalah sebuah package manager untuk JavaScript yang sangat mirip seperti composer. Malah aplikasi composer dibuat karena terinspirasi dari npm.

Menggunakan npm, kita bisa menginstall berbagai library JavaScript langsung dari cmd. Namun saya tidak akan membahas lebih jauh terkait npm, fokus kita sekedar bisa menjalankan Laravel mix saja.

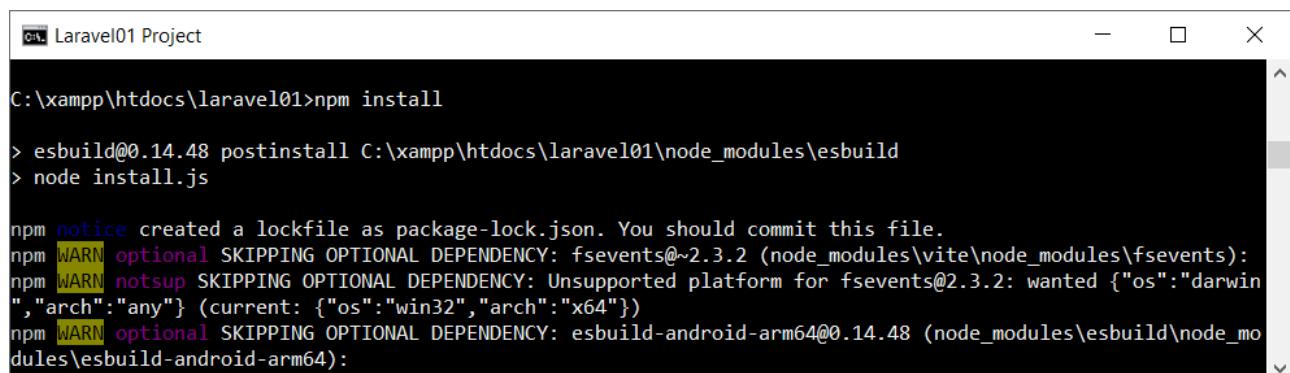
9.3. Instalasi Laravel Mix

Mulai dari versi 9.19, Laravel memperkenalkan [Laravel Vite](#) sebagai tools *compiling assets* default menggantikan Laravel Mix. Akan tetapi karena masih sangat baru dan lebih ditujukan bagi yang ingin membuat front-end JavaScript dengan React atau Vue, kita akan tetap memakai Laravel Mix dalam buku ini.

Untuk memulai proses instalasi Laravel Mix, silahkan buka **cmd** dan masuk ke dalam folder **laravel01** (folder tempat Laravel tersimpan). Pastikan terhubung ke internet karena proses ini akan mendownload file yang cukup besar, sekitar 100 MB.

Di dalam cmd, jalankan perintah berikut:

```
npm install
```



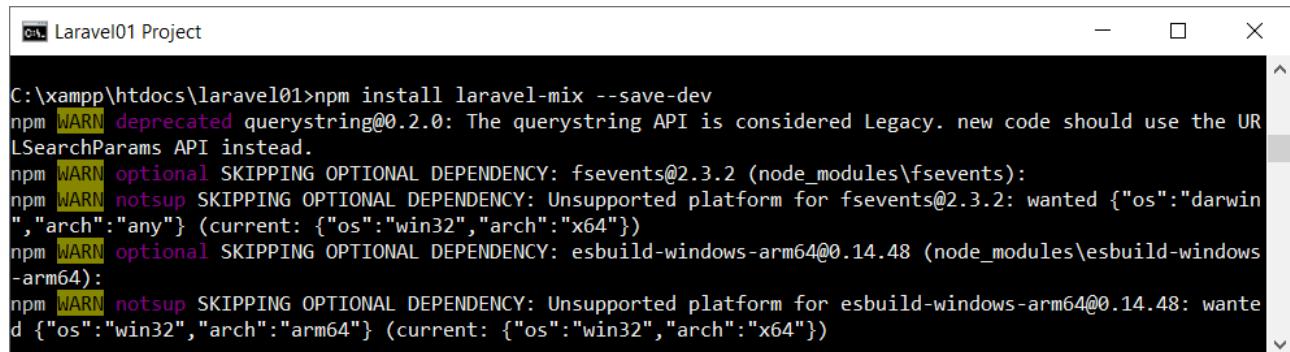
```
C:\xampp\htdocs\laravel01>npm install
> esbuild@0.14.48 postinstall C:\xampp\htdocs\laravel01\node_modules\esbuild
> node install.js

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.2 (node_modules\vite\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: esbuild-android-arm64@0.14.48 (node_modules\esbuild\node_modules\esbuild-android-arm64):
```

Gambar: Menjalankan perintah npm install

Perintah `npm install` akan mendownload berbagai modul JavaScript dan CSS tambahan yang dibutuhkan Laravel untuk memproses file-file front-end. Dulunya, Laravel mix termasuk modul bawaan, akan tetapi sejak Laravel 9.19 harus kita install manual dengan menjalankan perintah berikut:

```
npm install laravel-mix --save-dev
```

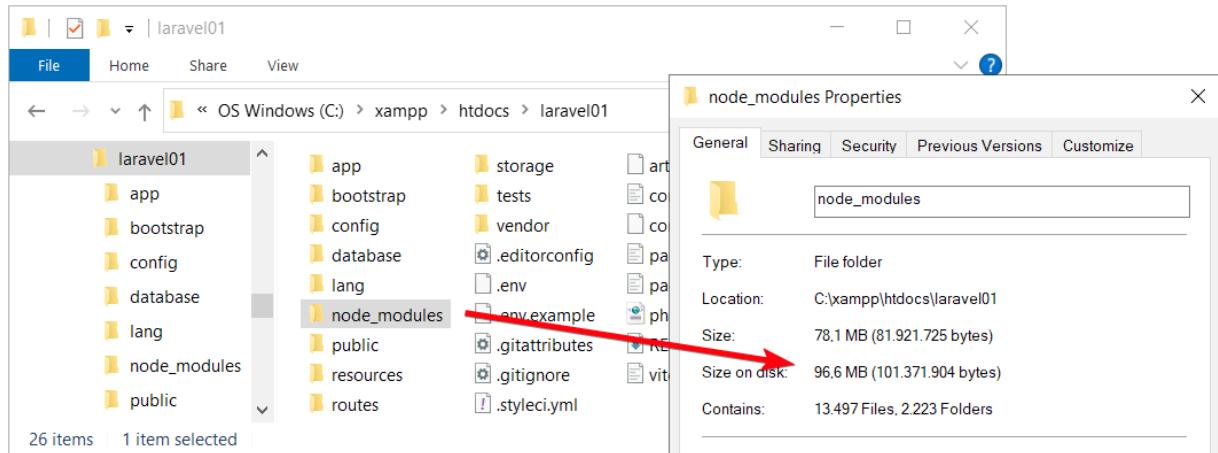


```
C:\xampp\htdocs\laravel01>npm install laravel-mix --save-dev
npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams API instead.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: esbuild-windows-arm64@0.14.48 (node_modules\esbuild-windows-arm64):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for esbuild-windows-arm64@0.14.48: wanted {"os":"win32","arch":"arm64"} (current: {"os":"win32","arch":"x64"})
```

Gambar: Menginstall modul laravel-mix

Setelah proses instalasi selesai, akan muncul folder baru bernama **node_modules**.

Laravel Mix (Compiling Assets)



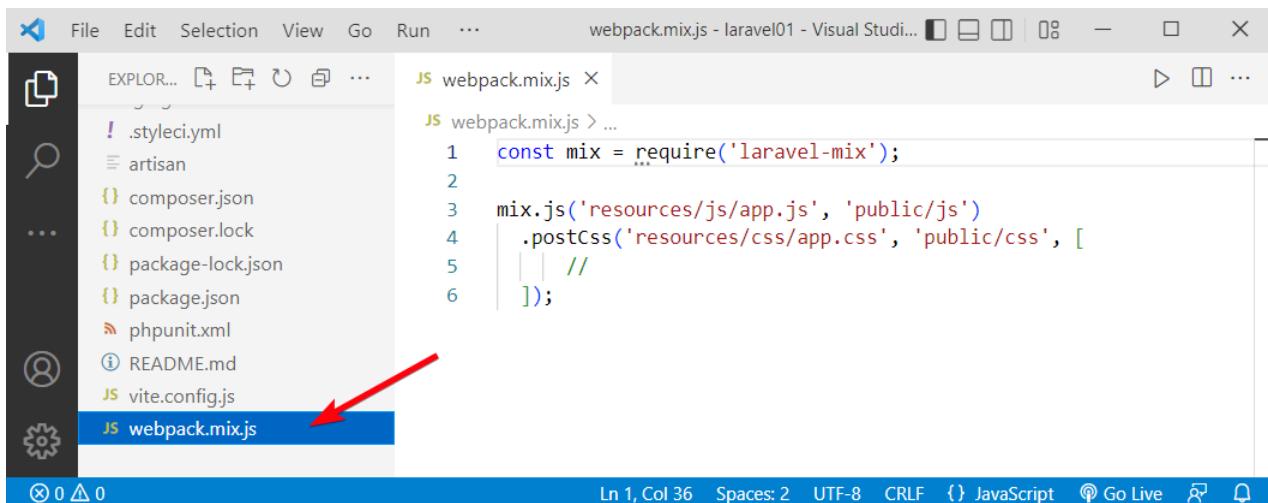
Gambar: Ukuran folder node_modules

Seperti yang terlihat, ukuran folder ini mencapai 101 MB dan terdiri dari hampir 13500 file, jauh lebih banyak dari pada ukuran file instalasi Laravel. Kita juga tidak perlu utak-atik file ini.

Langkah berikutnya, buat file dengan nama `webpack.mix.js` di dalam folder `laravel01` lalu isi dengan kode berikut:

`webpack.mix.js`

```
1 const mix = require('laravel-mix');
2
3 mix.js('resources/js/app.js', 'public/js')
4     .postCss('resources/css/app.css', 'public/css', [
5         //
6     ]);
```



Gambar: Buat file webpack.mix.js di folder instalasi Laravel

Penjelasan dari kode ini akan kita bahas sesaat lagi.

Ketika Laravel Mix masih menjadi tool *compiling assets* default Laravel, file `webpack.mix.js` sudah ada dari bawaan. Tetapi sejak Laravel 9.19, file ini terpaksa kita buat manual.

9.4. Menjalankan Laravel Mix

Pada dasarnya Laravel mix berfungsi untuk memproses file asset asal, yakni kode CSS dan JavaScript yang kita ketik di teks editor, menjadi file asset yang sudah di-optimasi. Proses optimasi tersebut merujuk ke salah satu dari penggabungan file, *pre-processor*, *minification* dan *cache busting*.

Persiapan pertama adalah menentukan lokasi tempat file asset asal disimpan. Meskipun secara teori bisa disimpan di mana saja, tapi Laravel sudah menyediakan lokasi khusus di folder **resources**.

Folder **resources** sudah sering kita akses karena berisi folder **view**, tempat file blade berada. Selain itu terdapat 2 folder lain: **js** dan **css**. Sesuai namanya, folder **js** di sediakan untuk menampung file JavaScript, dan folder **css** sebagai tempat file CSS.

Untuk menampung file hasil optimasi, bisa di simpan ke folder **public** agar bisa langsung diakses dari dalam view.

Tempat kita mengatur folder sumber dan folder hasil ini ada di file **webpack.mix.js**. Berikut saya tampilkan kembali kode yang kita tulis sebelumnya:

`webpack.mix.js`

```
1 const mix = require('laravel-mix');
2
3 mix.js('resources/js/app.js', 'public/js')
4     .postCss('resources/css/app.css', 'public/css', [
5         //
6     ]);
```

Di baris 1 terdapat proses inisialisasi konstanta **mix** dengan hasil pemanggilan fungsi `require('laravel-mix')`. Karena ini adalah file JavaScript, perintahnya sedikit berbeda dengan PHP.

Fokus utama ada di baris 3 - 6. Ini merupakan 1 perintah panjang yang di *chaining* dengan tanda titik. Dalam JavaScript, tanda titik dipakai untuk mengakses property dan method, yang kalau di PHP menggunakan tanda panah " `->` ".

Baris `mix.js('resources/js/app.js', 'public/js')` artinya, Laravel mix akan men-*compile* isi file `resources/js/app.js`, kemudian menyimpan hasilnya ke folder `public/js`.

Begini juga dengan baris `.postCss('resources/css/app.css', 'public/css', [])` yang akan men-*compile* file CSS di `resources/sass/app.css` dan menyimpan hasilnya ke folder `public/css`.

Mari kita tes. Buka kembali jendela cmd, masuk ke folder laravel dan jalankan perintah berikut:

`npx mix`

Laravel Mix (Compiling Assets)

Proses compile akan berlangsung beberapa saat. Jika sudah selesai, akan tampil teks berikut:

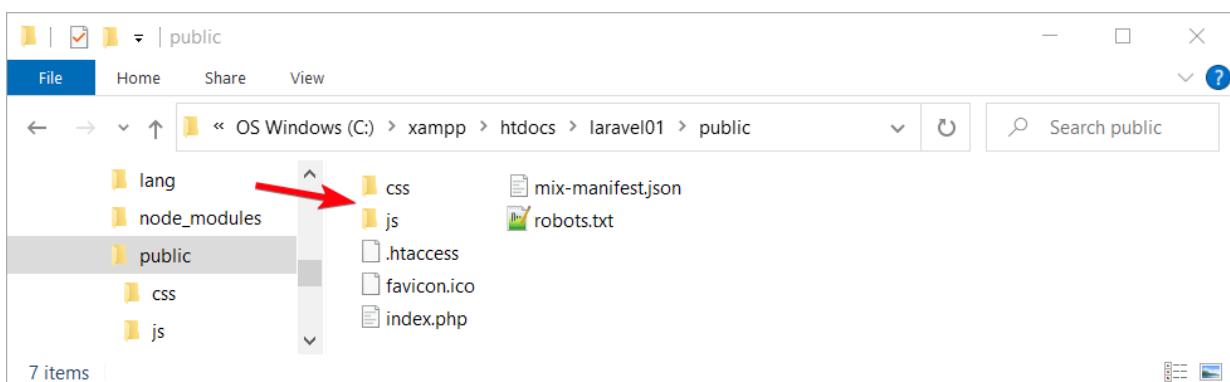
The screenshot shows a terminal window titled "Laravel01 Project" with the command "C:\xampp\htdocs\laravel01>npx mix" entered. The output is as follows:

```
Laravel Mix v6.0.49
Compiled Successfully in 1792ms
File      Size
/js/app.js 608 KiB
/css/app.css 1 bytes
webpack compiled successfully
```

The terminal window has a dark background with light-colored text. The output is in green and white.

Gambar: Hasil proses compile Laravel Mix

Lanjut, periksa folder public/js dan public/css, seharusnya terdapat file app.js dan app.css. Inilah file hasil compile Laravel mix:



Gambar: Folder css dan js dibuat oleh Laravel Mix

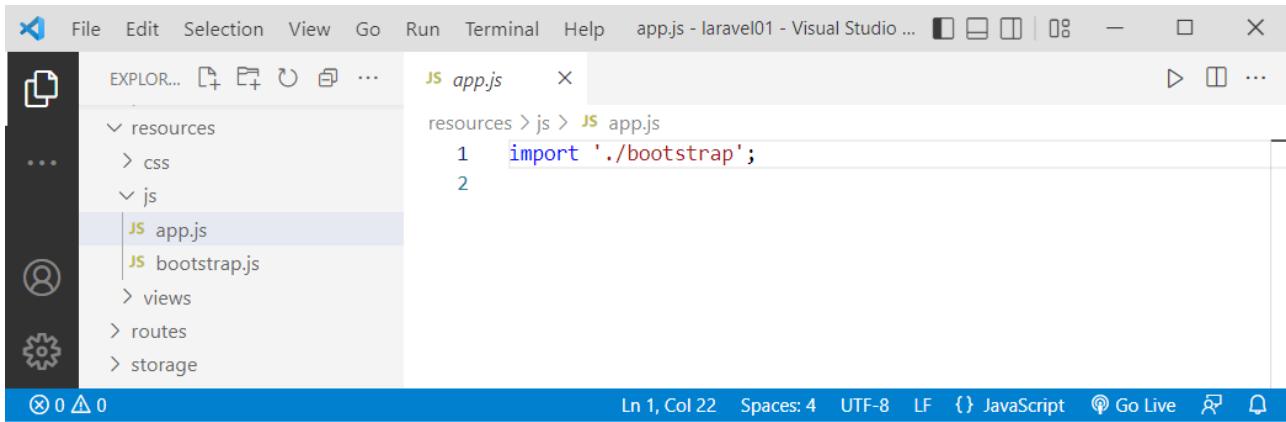
Penggabungan File

Fitur utama dari Laravel mix adalah menggabung beberapa file asset menjadi 1 file akhir. Sebelumnya sudah dibahas kalau file sumber Laravel mix berada di folder **resources**. Untuk file JavaScript, lokasinya ada di **resources/js**. Bawaan Laravel, folder ini sudah berisi 2 file: **app.js** dan **bootstrap.js**.

File **app.js** merupakan file internal Laravel untuk proses penggabungan Laravel mix, isinya berupa 1 baris perintah `import './bootstrap'`. Artinya sudah ada 1 file yang 'antri' untuk digabung. File tersebut merujuk ke **bootstrap.js** yang juga ada di folder ini.

File **bootstrap.js** tidak berhubungan dengan framework CSS Bootstrap, tapi berisi semacam file awal untuk proses internal Laravel. Lebih spesifik lagi, file **bootstrap.js** baru dibutuhkan ketika kita memakai fitur lanjutan seperti **Laravel UI** atau **Vue** framework.

Untuk saat ini, file `app.js` dan `bootstrap.js` bisa diabaikan.



Gambar: isi file app.js

Sebagai bahan percobaan, saya akan buat 2 file JavaScript dengan nama `my-script.js` dan `my-script-console.js`:

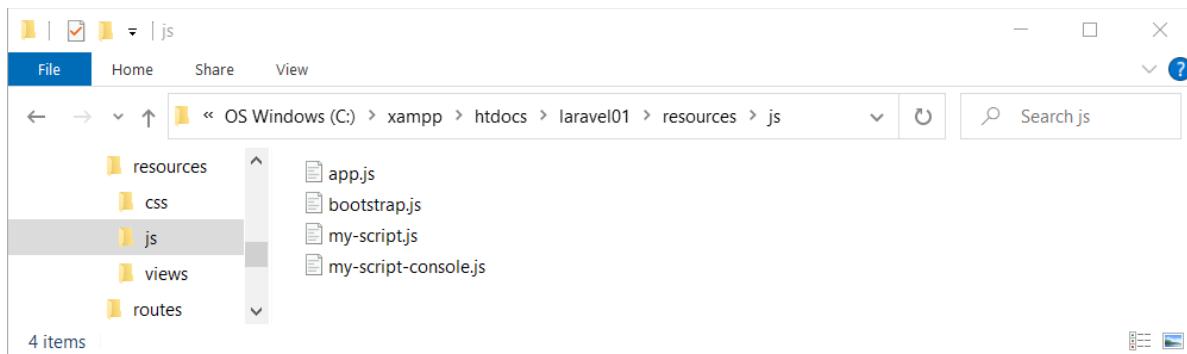
`resources/js/my-script.js`

```
1 // Tampilkan alert ketika halaman di klik
2
3 window.addEventListener('click',function(){
4     alert('Saya di klik');
5});
```

`resources/js/my-script-console.js`

```
1 // Tampilkan pesan console ketika halaman di klik
2
3 window.addEventListener('click',function(){
4     console.log('Saya juga di klik');
5});
```

Kode JavaScript ini di pakai untuk membuat event click ke dalam halaman web. Nantinya jika halaman di klik, akan tampil jendela `alert()` serta sebuah pesan ke tab console. Save kedua file ini ke dalam folder `resources/js`.



Gambar: Isi folder resources/js

Berikutnya saya juga akan buat 2 buah file CSS di folder resources/css.

Saat ini di dalam folder **css** sudah ada satu file **app.css**. File **app.css** tidak berisi kode apapun (kosong) dan untuk sementara boleh diabaikan terlebih dahulu. Silahkan buat file **my-style.css** dan **my-style-h1.css**, lalu simpan ke dalam folder ini:

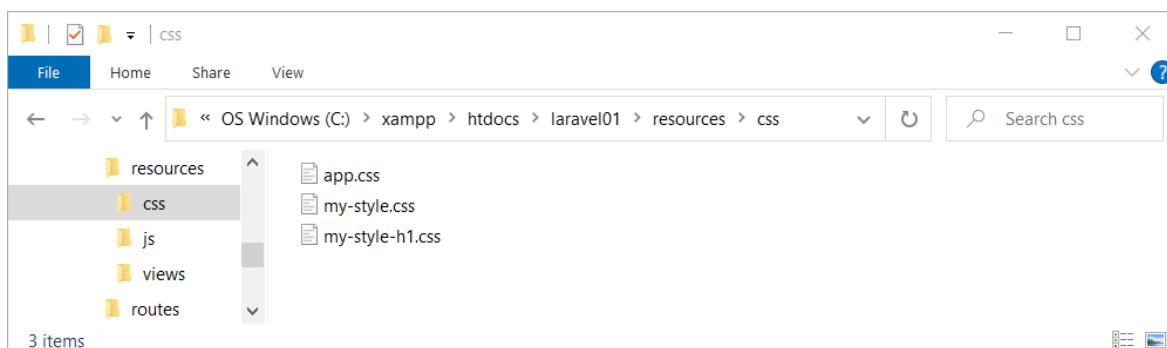
resources/css/my-style.css

```
1 /* Set warna background untuk tag body */
2
3 body {
4     background-color: beige;
5 }
```

resources/css/my-style-h1.css

```
1 /* Set warna background dan teks untuk tag <h1> */
2
3 h1 {
4     background-color: #beaa34;
5     color: #ffffff;
6     padding: 10px;
7     text-align: center;
8 }
```

File **my-style.css** berisi kode CSS sederhana untuk men-set warna background dari tag **<body>** menjadi *beige*, abu-abu muda kekuningan. Sedangkan file **my-style-h1.css** berisi kode untuk men-set style dari tag **<h1>**.



Gambar: Isi folder resources/css

Sekarang kita tinggal menulis instruksi untuk Laravel mix ke dalam file **webpack.mix.js**. Silahkan modifikasi dengan kode berikut:

webpack.mix.js

```
1 const mix = require('laravel-mix');
2
3 mix.scripts([
4     'resources/js/my-script.js',
5     'resources/js/my-script-console.js',
6 ], 'public/js/my-app.js');
```

```
7
8 mix.styles([
9   'resources/css/my-style.css',
10  'resources/css/my-style-h1.css',
11 ], 'public/css/my-app.css');
```

Perintah di baris 1 merupakan proses inisialisasi Laravel mix. Konstanta **mix** akan berisi object tempat kita memanggil berbagai method yang diperlukan.

Baris 3-6 dan baris 8-11 adalah perintah untuk proses penggabungan file. Secara umum, format perintah Laravel mix bisa dibaca sebagai berikut:

```
mix.<nama_proses>(<nama_file_sumber>,<nama_file_hasil>)
```

- <nama_proses> berisi nama proses yang akan dilakukan. Dalam contoh di atas, perintahnya adalah `mix.script()` untuk menggabung file JavaScript, dan `mix.styles()` untuk menggabung file CSS. Nantinya masih banyak perintah lain yang tersedia.
- <nama_file_sumber> adalah nama file yang akan digabung. Untuk perintah `script` dan `style`, nama file ini bisa berisi nama file, atau sebuah array. Karena saya menggabung 2 file, nama file ditulis dalam bentuk array. Isi array berupa alamat *path* dari setiap file.
- <nama_file_hasil> adalah nama file hasil penggabungan. Untuk JavaScript saya tulis sebagai `'public/js/my-app.js'`. Artinya setelah menjalankan Laravel mix, sebuah file bernama `my-app.js` akan dibuat secara otomatis. File ini berada di folder `public/js`. Begitu pula dengan file CSS yang akan disimpan ke dalam file `my-app.css` di folder `public/css`. Nama file ini boleh suka-suka kita, tergantung keinginan. Jika sebelumnya file ini sudah ada, isinya akan ditimpas.

Persiapan sudah selesai. Jalankan ulang proses compile Laravel Mix dengan perintah:

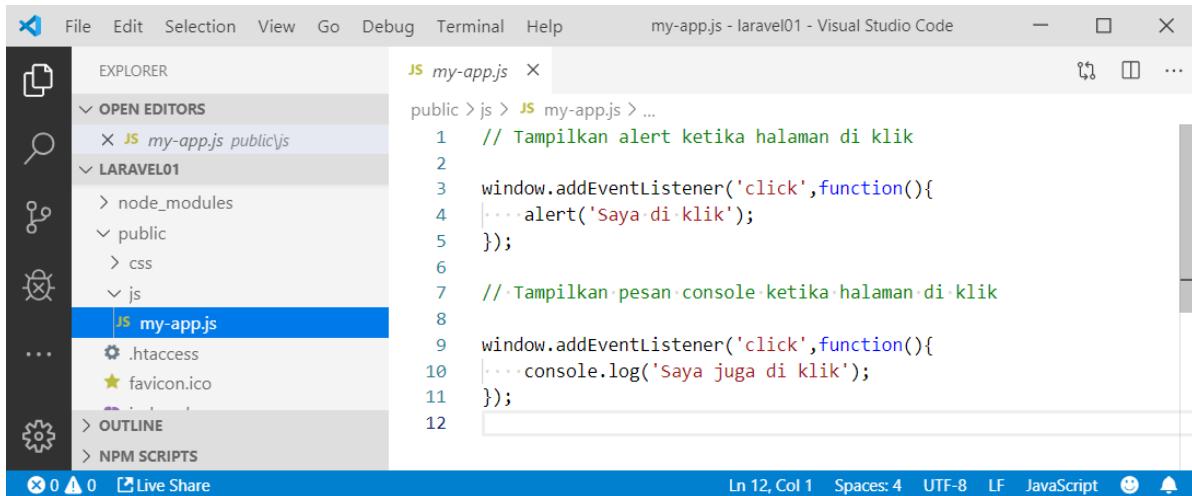
```
npx mix
```

File	Size
\css\my-app.css	235 bytes
\js\my-app.js	265 bytes

Gambar: Menjalankan perintah npx mix

Setelah itu periksa isi file `public/js/my-app.js` dan `public/css/my-app.css`. Di kedua file ini terdapat gabungan kode yang kita tulis sebelumnya:

Laravel Mix (Compiling Assets)

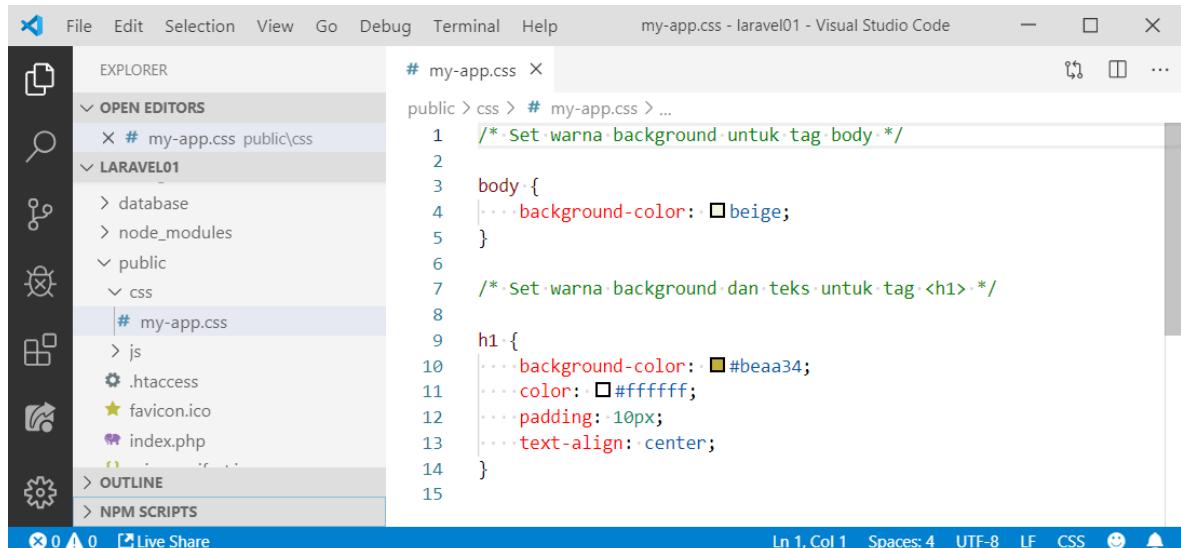


The screenshot shows the Visual Studio Code interface with the file 'my-app.js' open in the editor. The code contains JavaScript code that adds event listeners to the document body and an h1 element, both triggering alerts and console logs.

```
JS my-app.js
public > js > JS my-app.js > ...
1 // Tampilkan alert ketika halaman di klik
2
3 window.addEventListener('click',function(){
4 ....alert('Saya di klik');
5 });
6
7 // Tampilkan pesan console ketika halaman di klik
8
9 window.addEventListener('click',function(){
10 ....console.log('Saya juga di klik');
11 });
12
```

Ln 12, Col 1 Spaces: 4 UTF-8 LF JavaScript

Gambar: Isi file my-app.js



The screenshot shows the Visual Studio Code interface with the file 'my-app.css' open in the editor. The code defines CSS styles for the 'body' and 'h1' elements, setting background colors and text properties.

```
# my-app.css
public > css > # my-app.css > ...
1 /* Set warna background untuk tag body */
2
3 body {
4 ....background-color: beige;
5 }
6
7 /* Set warna background dan teks untuk tag h1 */
8
9 h1 {
10 ....background-color: #beaa34;
11 ....color: white;
12 ....padding: 10px;
13 ....text-align: center;
14 }
15
```

Ln 1, Col 1 Spaces: 4 UTF-8 LF CSS

Gambar: Isi file my-app.css

Di dalam folder public/js dan public/css juga terdapat file app.js dan app.css hasil testing Laravel Mix pertama kali.

Praktek selanjutnya, kita akan coba akses file my-app.js dan my-app.css dari dalam view. Silahkan modifikasi view welcome.blade.php bawaan Laravel dengan kode berikut:

resources/view/welcome.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Document</title>
8   <link rel="stylesheet" href="{{ asset('/css/my-app.css') }}">
9 </head>
```

Laravel Mix (Compiling Assets)

```
10 <body>
11   <h1>Belajar Laravel</h1>
12   <script src="{{ asset('/js/my-app.js') }}"></script>
13 </body>
14 </html>
```

Ini merupakan struktur HTML standar. Perhatikan di baris 8 dan 12 saya memakai *helper function asset()* untuk mengakses file hasil Laravel Mix.

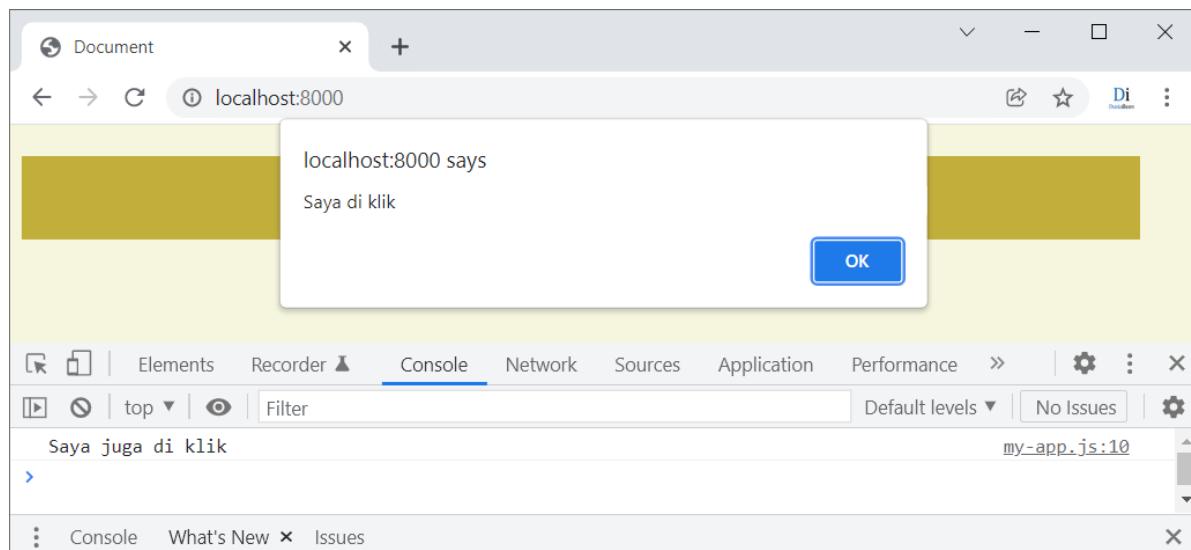
Buka web browser dan berikut hasilnya:



Gambar: Tampilan view welcome.blade.php

Sip, kode CSS kita sudah berjalan.

Bagaimana dengan JavaScript? Silahkan buka tab console dari Web Development Tools (Ctrl + Shift + J) dan klik tempat sembarang di halaman web:



Gambar: Tampilan alert ketika halaman di klik

Hasilnya tampil popup alert 'Saya di klik' serta pesan console 'Saya juga di klik'. Ini menandakan kode JavaScript juga sudah berjalan.

Sampai di sini kita sudah melihat bagaimana step-by-step proses compile assets menggunakan Laravel Mix. Sebagai kesimpulan, terdapat 4 langkah yang harus dilakukan:

1. Tulis kode program ke dalam folder `resources`.
2. Daftarkan semua file yang akan di compile ke file `webpack.mix.js`.
3. Jalankan Laravel Mix dengan perintah `npx mix`.
4. Akses file hasil proses compile dari folder `public/js/` dan `public/css/`.

Kemudian, bagaimana jika kita ingin mengubah sesuatu? Misalnya mengganti warna background atau mengganti isi pesan alert?

Caranya, edit kembali kode CSS dan JS di file asli yang berada di folder `resources`, bukan yang ada di folder `public`. Kemudian jalankan kembali Laravel Mix dengan perintah `npx mix`.

Memproses Pre-processor

Fitur kedua dari Laravel mix adalah memproses kode pre-processor.

Sass merupakan CSS pre-processor yang cukup populer dan sudah menjadi standar tidak resmi dari CSS pre-processor, meskipun ada alternatif lain seperti **Less** dan **Stylus**.

Agar lebih rapi, silahkan buat folder `sass` di dalam folder `resources`. Ini tidak wajib, agar struktur folder kita lebih rapi saja. Ke dalam folder ini, ketik kode berikut dan simpan sebagai `my-style-h1.scss`:

`resources/sass/my-style-h1.scss`

```
1 /* Set warna background dan teks untuk tag <h1> */
2
3 $warna-background: #beaa34;
4 $warna-text: #ffffff;
5
6 h1 {
7     background-color: $warna-background;
8     color: $warna-text;
9     padding: 10px;
10    text-align: center;
11 }
12
```

Materi Sass sangat luas dan di luar pembahasan buku ini. Apa yang akan kita lakukan hanya sekedar melihat bagaimana Laravel Mix bisa dipakai untuk memproses kode Sass menjadi kode CSS biasa. Akhiran `.scss` merupakan extension dari **Sass**. Semua kode program yang menggunakan Sass harus ditulis dalam extension ini.

Kode Sass sebenarnya sangat mirip seperti kode CSS biasa, plus dengan tambahan fitur khusus seperti pembuatan variabel. Di baris 3-4 saya membuat variabel `$warna-background`, dan `$warna-text`. Kedua variabel dipakai sebagai nilai input untuk property `background-color` dan `color` di baris 7-8.

Laravel Mix (Compiling Assets)

Untuk bisa memproses file Sass, Laravel mix menyediakan perintah `mix.sass()`. Berikut kode yang diperlukan:

`webpack.mix.js`

```
1 const mix = require('laravel-mix');
2
3 mix.scripts([
4     'resources/js/my-script.js',
5     'resources/js/my-script-console.js',
6 ], 'public/js/my-app.js');
7
8 mix.sass('resources/sass/my-style-h1.scss','public/css/my-app.css');
```

Kode ini mirip seperti di proses penggabungan file. Di baris 1 terdapat perintah inisialisasi konstanta `mix`, diikuti proses penggabungan file JavaScript di baris 3-6.

Di baris 8 terdapat perintah baru berupa `mix.sass()`. Perintah ini dipakai sebagai instruksi ke Laravel mix bahwa kita ingin memproses file Sass yang ada di `resources/sass/my-style-h1.scss`, menjadi kode CSS biasa yang disimpan ke `public/css/my-app.css`.

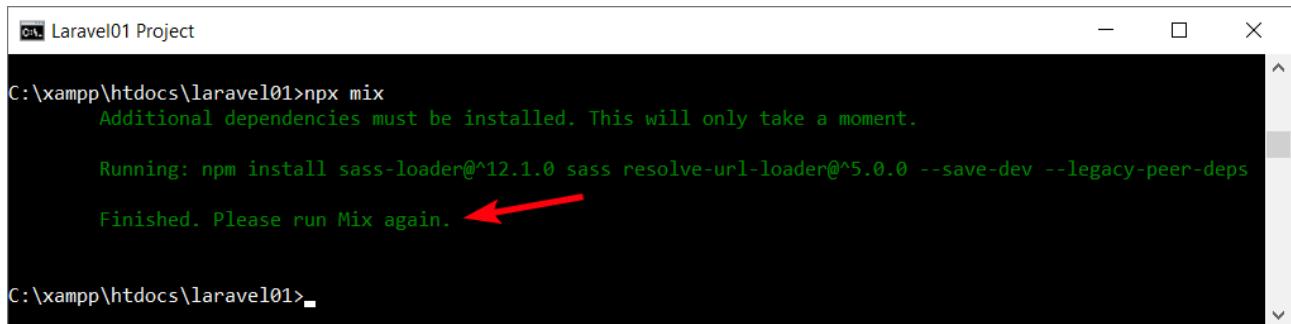
Perhatikan format perintah Laravel mix ini sama seperti sebelumnya, yaitu:

```
mix.<nama_proses>(<nama_file_sumber>,<nama_file_hasil>)
```

Baik, saatnya jalankan Laravel mix dengan perintah:

```
npx mix
```

Pada saat dijalankan pertama kali, perintah di atas belum men-compile file Sass, tapi Laravel Mix akan menginstall `sass-loader` terlebih dahulu:



```
C:\xampp\htdocs\laravel01>npx mix
    Additional dependencies must be installed. This will only take a moment.

    Running: npm install sass-loader@^12.1.0 sass resolve-url-loader@^5.0.0 --save-dev --legacy-peer-deps
    Finished. Please run Mix again.
```

Gambar: Laravel Mix menginstall sass-loader

Maka ketik kembali `npx mix`:

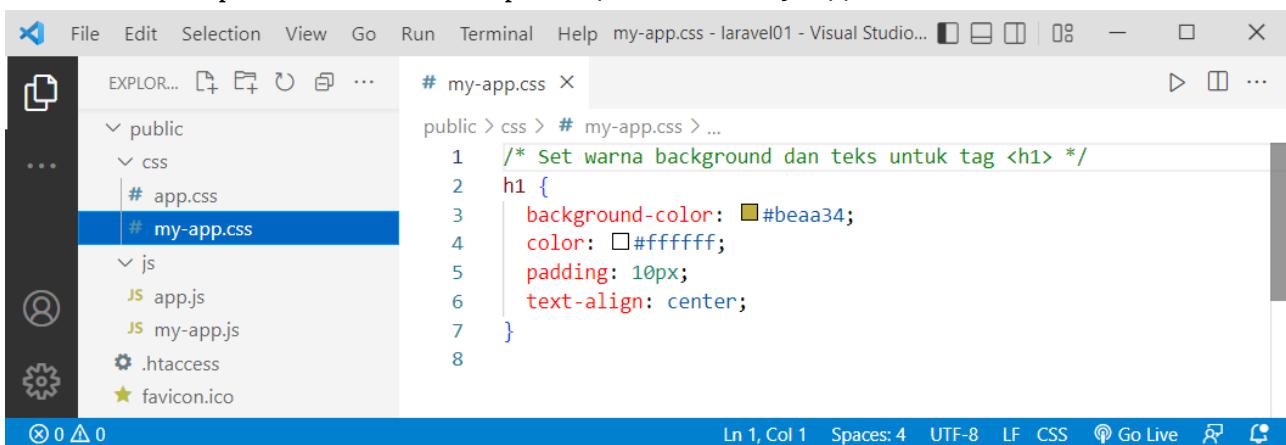
Laravel Mix (Compiling Assets)



The screenshot shows a terminal window titled "Laravel01 Project". It displays the output of the Laravel Mix command, which compiled successfully in 5483ms. The terminal lists two files: "app.js" (265 bytes) and "my-app.css" (144 bytes). Below the terminal, the command "webpack compiled successfully" is shown, along with the path "C:\xampp\htdocs\laravel01>".

Gambar: Proses compile file Sass

Setelah selesai, periksa file hasil compile di public\css\my-app.css:



The screenshot shows a Visual Studio Code editor window. The left sidebar shows a file tree with "public" folder containing "css", "js", ".htaccess", and "favicon.ico". Inside "css", there are "app.css" and "my-app.css". The "my-app.css" file is open in the editor, showing the following CSS code:

```
/* Set warna background dan teks untuk tag <h1> */
h1 {
    background-color: #beaa34;
    color: #ffffff;
    padding: 10px;
    text-align: center;
}
```

The status bar at the bottom indicates "Ln 1, Col 1" and "Spaces: 4".

Gambar: Hasil compile Sass menjadi kode CSS

Terlihat isi dari my-app.css sudah berbentuk kode CSS biasa, inilah hasil proses compile dari my-style-h1.scss. Property background-color dan color yang sebelumnya berisi variabel Sass sudah di konversi menjadi kode warna hexadesimal.

Bagaimana jika ada 2 file Sass yang akan digabung? Perintah `mix.sass()` ternyata tidak mendukung input dalam bentuk array, sehingga tidak bisa ditulis ke `webpack.mix.js`. Namun terdapat cara lain, kumpulkan saja semua file CSS di dalam 1 file menggunakan perintah `@import`, lalu baru input ke dalam `mix.sass()`.

Sebagai file sample kedua, saya akan buat file my-style.scss dengan kode sebagai berikut:

resources/sass/my-style.scss

```
/* Set warna background untuk tag body */
body {
    background-color: beige;
}
```

Betul, isinya hanya kode CSS biasa. Dan ini memang bisa kita lakukan untuk kode Sass. Lalu

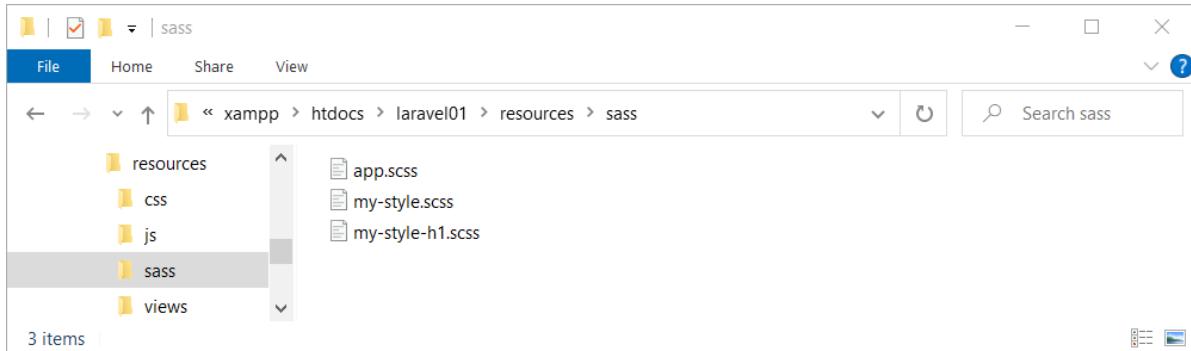
Laravel Mix (Compiling Assets)

buat sebuah file `app.scss` dan ketik perintah berikut:

```
resources/sass/app.scss
```

```
1 @import 'my-style';
2 @import 'my-style-h1';
```

Perintah `@import` juga merupakan kode CSS biasa yang dipakai untuk mengumpulkan 2 file `.scss` dalam 1 file. Berikut isi dari folder `resources/sass` saat ini:



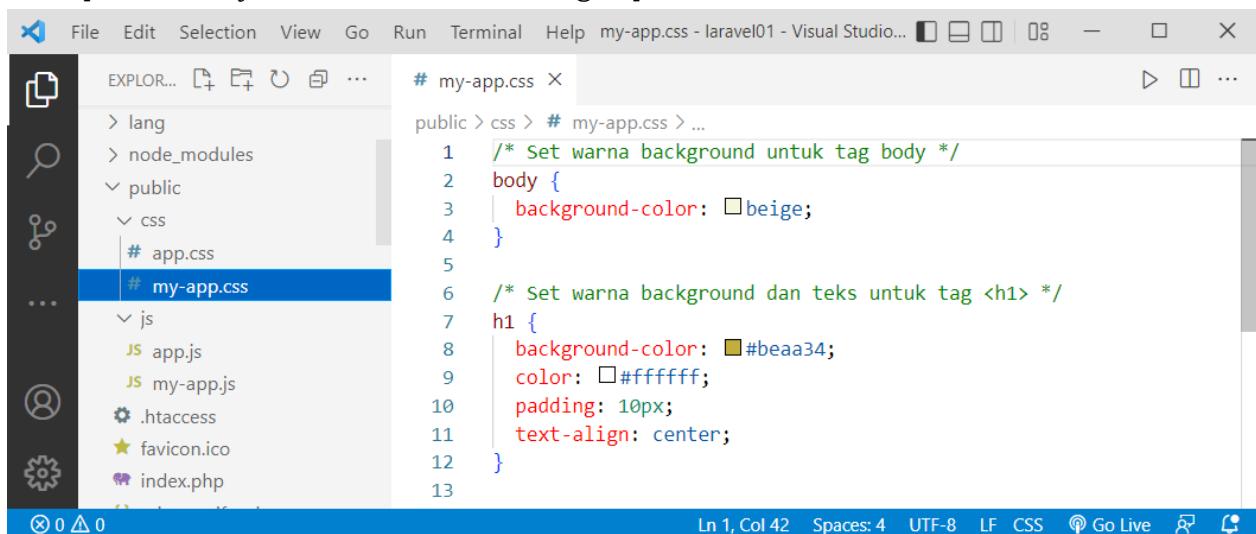
Gambar: Isi folder resources/sass

File `app.scss` inilah yang selanjutnya kita input ke dalam `mix.sass()`:

```
webpack.mix.js
```

```
1 const mix = require('laravel-mix');
2
3 mix.scripts([
4     'resources/js/my-script.js',
5     'resources/js/my-script-console.js',
6 ], 'public/js/my-app.js');
7
8 mix.sass('resources/sass/app.scss', 'public/css/my-app.css');
```

Dan seperti biasa, jalankan Laravel mix dengan perintah `npx mix`.



Gambar: Hasil compile Sass menjadi kode CSS

Sip, kedua file sudah berhasil di proses menjadi `my-app.css`.

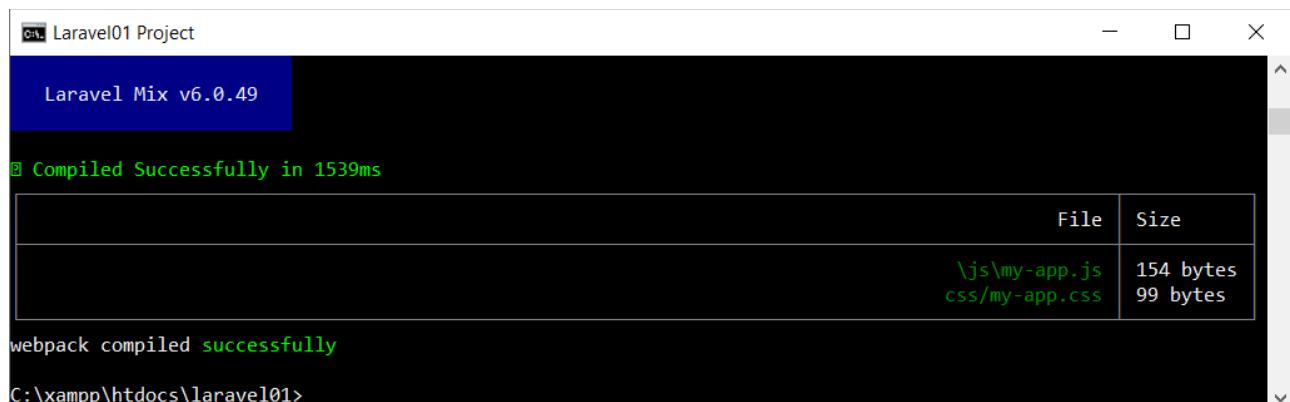
Bagaimana jika kita memiliki beberapa file `.scss` dan `.css` biasa yang ingin digabung? Tidak masalah, kembali 'kumpulkan' semua file ke dalam 1 file menggunakan perintah `@import`, lalu baru input ke `mix.sass()`.

Selain `mix.sass()`, Laravel mix juga menyediakan perintah `mix.less()` dan `mix.stylus()` untuk memproses CSS pre-processor **Less** dan **Stylus**.

Membuat File Minify

Kita sudah berhasil menggabungkan file asset serta memproses kode pre-processor Sass. Namun file itu masih berbentuk kode asli. Untuk mempercepat tampilan web, bisa di-*minify* lebih lanjut. Caranya juga sangat sederhana, cukup jalankan Laravel Mix menggunakan perintah:

```
npx mix -p
```

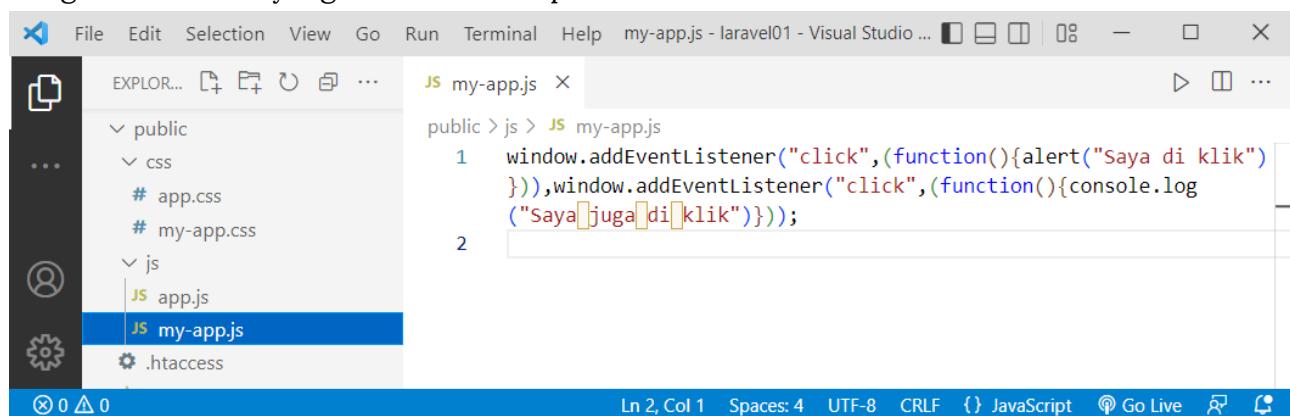


File	Size
\js\my-app.js	154 bytes
css/my-app.css	99 bytes

webpack compiled successfully
C:\xampp\htdocs\laravel01>

Gambar: Hasil menjalankan perintah `npx mix -p`

Hasilnya, file `my-app.js` dan `my-app.css` secara otomatis di *generate* dalam bentuk *minify* dengan ukuran file yang lebih kecil dari pada file asset asli.



```
public > js > JS my-app.js
1 window.addEventListener("click",(function(){alert("Saya di klik")
}),window.addEventListener("click",(function(){console.log
("Saya juga di klik"))));
2
```

Gambar: Isi file `my-app.js` setelah di minify

File `my-app.js` dan `my-app.css` hanya berisi 1 baris panjang, selain itu semua baris komentar

dan mayoritas karakter whitespace juga dihapus. File hasil minify ini sangat pas dipakai ketika aplikasi sudah memasuki tahap *production*, yakni ketika akan di upload ke web hosting.

Perintah "npx mix -p" merupakan singkatan dari "npx mix --production". Kedua perintah ini bisa dipakai untuk membuat versi minify Laravel Mix.

Membuat Cache Busting (File Versioning)

Cache busting termasuk teknik yang agak *advanced*. Jika anda belum pernah membuat project yang dirilis ke hosting lalu di update secara berkala, kemungkinan belum pernah menemui masalah yang butuh *cache busting*.

Seperti yang telah di jelaskan di awal bab, *cache busting* diperlukan untuk memaksa web browser agar mendownload file asset baru, meskipun cache di web browser belum expired. Triknya, tulis *file versioning* memakai *query string* di akhir file. Laravel Mix menyediakan cara yang sangat praktis untuk membuat fitur seperti ini.

Pertama, buka kembali file `webpack.mix.js`, lalu tambah perintah `.version()` di setiap akhir perintah `mix()`:

`webpack.mix.js`

```
1 const mix = require('laravel-mix');
2
3 mix.scripts([
4     'resources/js/my-script.js',
5     'resources/js/my-script-console.js',
6 ], 'public/js/my-app.js').version();
7
8 mix.sass('resources/sass/app.scss','public/css/my-app.css').version();
```

Save file `webpack.mix.js`, lalu compile dengan perintah `npx mix -p` atau bisa juga `npx mix` jika anda tidak ingin file asset langsung di minify.

Sampai di sini, jika halaman web dibuka, tidak ada yang tampak berubah. Link ke file `app.js` dan `app.css` tetap tampil normal tanpa tambahan file versioning. Ini karena kita harus modifikasi sedikit file view:

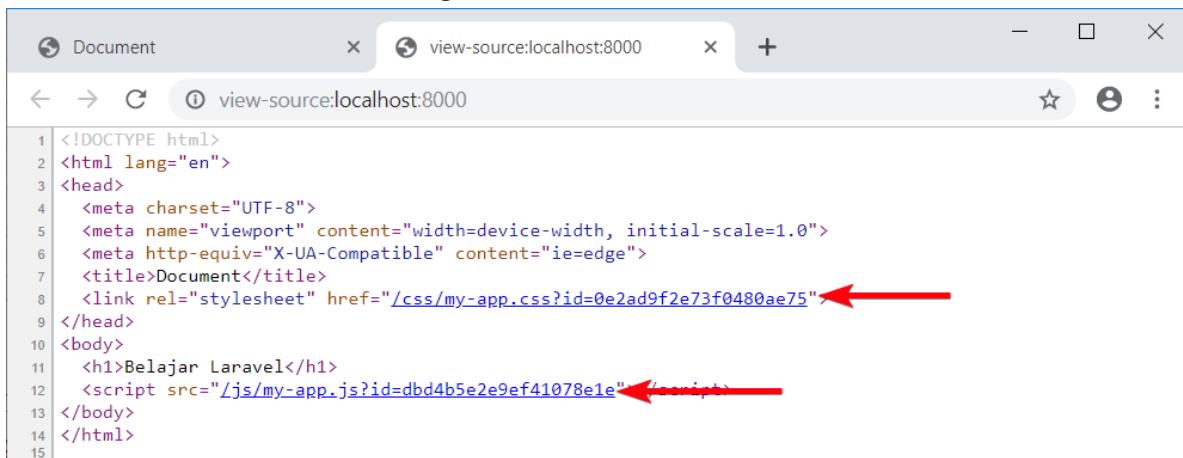
`resources/view/welcome.blade.php`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Document</title>
8     <link rel="stylesheet" href="{{ mix('/css/my-app.css') }}>
9 </head>
```

```
10 <body>
11   <h1>Belajar Laravel</h1>
12   <script src="{{ mix('/js/my-app.js') }}"></script>
13 </body>
14 </html>
```

Perubahannya ada di baris 8 dan 12. Kali ini saya menggunakan *helper function* `mix()`, tidak lagi `asset()`. Ini diperlukan agar Laravel bisa menambah file versioning pada saat view diakses dari web browser. Argumen dari function `mix()` sama seperti `asset()`, yakni alamat URL dari file asset yang ada di folder **public**.

Sekarang buka view di web browser, lalu lihat bagian source code. Sekarang link ke file assets sudah memiliki tambahan versioning:



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Document</title>
8   <link rel="stylesheet" href="/css/my-app.css?id=0e2ad9f2e73f0480ae75">←
9 </head>
10 <body>
11   <h1>Belajar Laravel</h1>
12   <script src="/js/my-app.js?id=dbd4b5e2e9ef41078e1e">←
13 </body>
14 </html>
15
```

Gambar: Tambahan versioning di akhir file app.css dan app.js

File versioning yang dimaksud adalah tambahan *query string* `?id=0e2ad9f2e...` di akhir file `my-app.css` dan `my-app.js`. *Query string* di-generate secara acak setiap kali Laravel Mix dijalankan, dengan catatan harus terdapat perubahan di file asal. Jika tidak ada perubahan, *query string* yang dihasilkan akan tetap sama.

#Exercise

Sebagai latihan, bisa lakukan sedikit percobaan.

Tanpa mengubah isi kode file asset (file js dan sass), jalankan Laravel Mix. Kemudian lihat di source code web browser apakah ada perubahan dari versi teks *query string* atau tidak.

Lalu tambah beberapa kode baru ke file asset, atau bisa juga dengan mengubah urutan daftar file yang digabung. Kemudian jalankan kembali Laravel Mix dan lihat perubahan yang terjadi.

Menjalankan Laravel Mix Secara Otomatis

Dari beberapa percobaan ini saya yakin anda sudah paham bagaimana Laravel Mix bekerja.

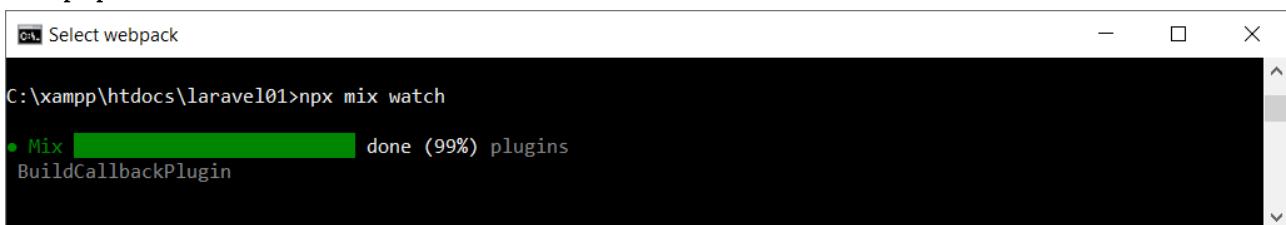
Namun sekarang muncul sedikit masalah yang membuat repot. Ketika kita mengubah file asset, yakni file yang ada di folder `resources`, maka Laravel Mix harus dijalankan ulang agar perubahan bisa terlihat di web browser.

Ini diperlukan karena yang diakses oleh view adalah file asset di folder `public`, sehingga kita harus terus mengetik perintah `npx mix` di cmd secara berkala.

Laravel Mix menyediakan solusi untuk masalah ini agar otomatis berjalan setiap kali file assets di save. Caranya, jalankan Laravel Mix dengan perintah berikut:

```
npx mix watch
```

Perintah ini akan membuat layar cmd tidak bisa dipakai karena proses 'watch' akan terus berjalan di background (mirip seperti efek perintah `php artisan serve`). Jika jendela cmd di tutup, proses `watch` akan berhenti.



```
Select webpack
C:\xampp\htdocs\laravel01>npx mix watch
• Mix [██████████] done (99%) plugins
BuildCallbackPlugin
```

Gambar: Menjalankan Laravel Mix dengan `npx mix watch`

Sekarang silahkan edit file asset. Begitu tombol save ditekan, Laravel Mix langsung meng-compile file asset secara otomatis. Kita tidak perlu repot menjalankan perintah `npx mix` secara manual.

Jika file yang kita compile hanya terdiri dari file JavaScript dan CSS biasa, maka proses compile langsung berjalan secara instan. Begitu di save, file `my-app.css` dan `my-app.js` langsung terupdate (isi cmd memang tidak terlihat perubahan apa-ap).

Tapi jika file yang kita compile menggunakan Sass, maka akan ada jeda beberapa detik menunggu Laravel mix memprosesnya terlebih dahulu (proses ini bisa dilihat di cmd).

Untuk menghentikan `npx mix watch`, bisa dengan menekan tombol Crtl + C, atau tutup saja jendela cmd.

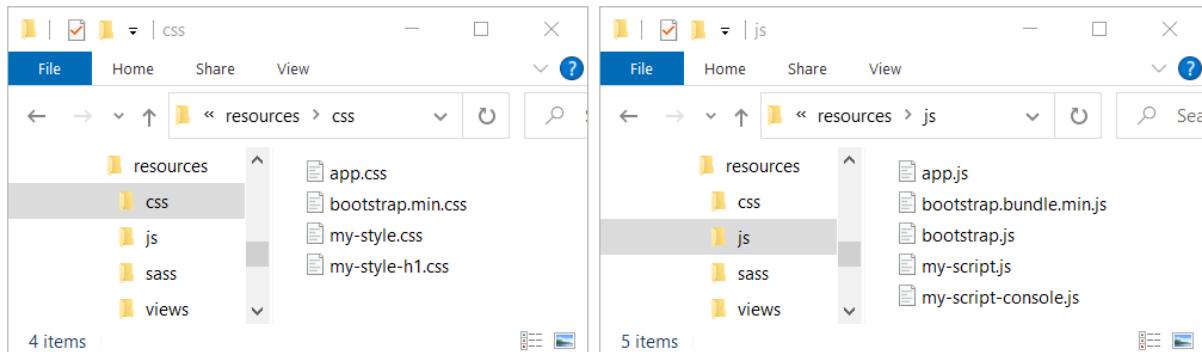
9.5. Compile Bootstrap dengan Laravel Mix

Bahasan kita kali ini sebenarnya lebih ke latihan, dimana saya ingin men-compile file Bootstrap CSS Framework menggunakan Laravel Mix.

Seperi yang sudah kita coba di akhir bab view, terdapat 2 file untuk framework Bootstrap 5,

yakni 1 file CSS (`bootstrap.min.css`), serta 1 file JavaScript (`bootstrap.bundle.min.js`). Jika kita juga memiliki file CSS dan JS sendiri, maka akan lebih efisien jika semua file assets ini bisa digabung.

Caranya juga cukup mudah. Pertama, copy file Bootstrap ke dalam folder **resources**, file `bootstrap.min.css` ke dalam folder `resources/css`, serta file `bootstrap.bundle.min.js` ke dalam folder `resources/js`. Jika anda tidak punya file ini, bisa diakses dari file `belajar_laravel.zip` yang ada di Google Drive.



Gambar: File Boostrap di folder resources/css dan resources/js

Perhatikan bahwa file `bootstrap.js` dan `bootstrap.bundle.min.js` merupakan 2 file yang berbeda. File `bootstrap.js` adalah file bawaan Laravel, sedangkan file `bootstrap.bundle.min.js` adalah file JavaScript milik framework Bootstrap.

Setelah itu saya ingin mengubah sedikit isi file `my-script.js`:

```
resources/js/my-script.js
1 // Tampilkan popover Bootstrap
2
3 new bootstrap.Popover(document.getElementById('myPopover'))
```

Ini merupakan kode JavaScript untuk mengaktifkan komponen *popover* Bootstrap, sekedar membuktikan bahwa kita sudah bisa mengakses file Bootstrap. Selanjutnya daftarkan semua file ke dalam `webpack.mix.js`:

```
webpack.mix.js
1 const mix = require('laravel-mix');
2
3 mix.scripts([
4   'resources/js/bootstrap.bundle.min.js',
5   'resources/js/my-script.js',
6 ], 'public/js/my-app.js').version();
7
8 mix.styles([
9   'resources/css/bootstrap.min.css',
10  'resources/css/my-style.css',
```

```
11 ], 'public/css/my-app.css').version();
```

Satu hal yang harus diperhatikan, urutan dari file-file ini sangat berpengaruh. Misalnya untuk file JavaScript bawaan Bootstrap (`bootstrap.bundle.min.js`) harus ditulis sebelum file `my-script.js`, karena file ini memiliki *dependency* kepada kode JS Bootstrap .

Begitu juga untuk urutan file CSS, file css yang kita tulis harus ditempatkan setelah file CSS bawaan Bootstrap. Jika dibalik, terdapat kemungkinan kode CSS milik Bootstrap akan menimpa kode yang kita tulis (efek cascading CSS).

Baik, semua persiapan sudah selesai, tinggal jalankan Laravel mix dengan perintah `npx mix / npx mix -p / npx mix watch` sesuai keinginan.

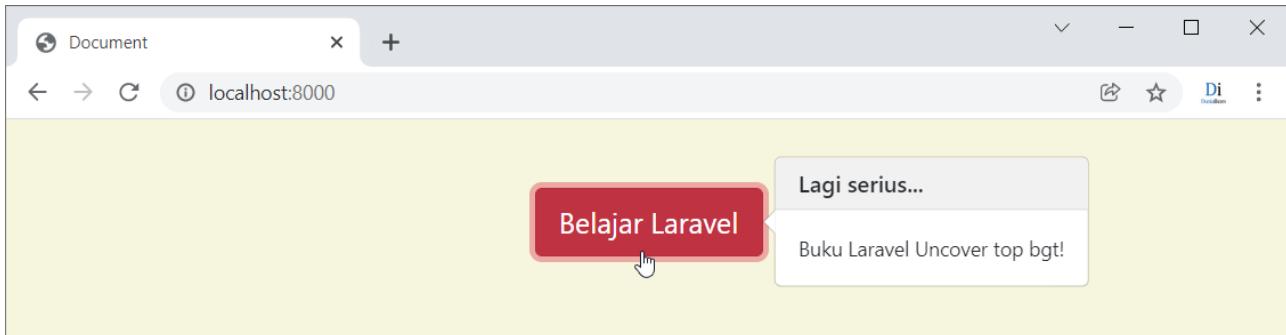
Setelah selesai, silahkan lihat file `my-app.css` dan `my-app.js` di folder public, seharusnya sudah berisi kode Bootstrap serta tambahan file dari `my-script.js` dan `my-style.css`.

Untuk membuktikan, saya akan modifikasi kembali file view welcome:

```
resources/view/welcome.blade.php
```

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Document</title>
8      <link rel="stylesheet" href="{{ mix('/css/my-app.css') }}">
9  </head>
10 <body>
11     <div class="container text-center py-5">
12         <button id="myPopover" type="button" class="btn btn-lg btn-danger"
13             data-bs-toggle="popover" title="Lagi serius..." 
14             data-bs-content="Buku Laravel Uncover top bgt!">
15             Belajar Laravel </button>
16     </div>
17     <script src="{{ mix('/js/my-app.js') }}"></script>
18 </body>
19 </html>
```

Perubahannya hanya di baris 11 – 15, dimana saya membuat sebuah class `.container` yang berisi tag `<button>`. Berikut tampilannya di web browser:



Gambar: Hasil view welcome dengan komponen popover Bootstrap

Silahkan klik tombol "Belajar Laravel". Jika tampil popover di samping kanan tombol, artinya semua file Bootstrap sudah sukses digabung menggunakan Laravel mix.

Sepanjang bab ini kita sudah membahas cara penggunaan Laravel Mix. Cukup banyak pengetahuan tambahan yang kita pelajari, tidak hanya terkait dengan Laravel saja tapi juga cara penanganan asset yang ideal pada sebuah project.

Anda boleh menggunakan Laravel Mix sepanjang sisa buku ini, atau tetap mengakses file JS dan CSS secara manual langsung dari folder `public`.

Saya akan lebih banyak menggunakan cara kedua, yakni mengakses file JS dan CSS langsung dari folder `public`, alasannya agar tidak repot harus terus menginstall Laravel Mix setiap kali Laravel diinstall ulang, terlebih ukuran file yang di download cukup besar.

Namun untuk "project sebenarnya", Laravel mix menjadi salah satu fitur yang wajib dipakai, terlebih jika anda menggunakan Sass dalam pengembangan front-end atau menggunakan Laravel UI yang akan kita bahas sesaat lagi.

10. Laravel UI

Materi kita kali ini masih berhubungan dengan front-end. Laravel sebenarnya tidak mewajibkan stack khusus untuk front-end. Kita bebas apakah ingin memakai framework CSS Bootstrap, Tailwind CSS, React JS, Vue JS, atau framework lain.

Sejak Laravel 6, **Laravel UI** hadir sebagai tambahan fitur untuk mempermudah proses integrasi Laravel dengan Bootstrap, Vue atau React. Setelah itu di Laravel 8 hadir pula **Laravel Breeze** untuk integrasi dengan framework Tailwind CSS.

Laravel UI dan Laravel Breeze nantinya juga dipakai untuk proses pembuatan authentication bawaan Laravel (fitur login/logout). Karena saya pribadi lebih familiar dengan Bootstrap, kita akan bahas tentang Laravel UI saja.

Laravel UI ini bersifat opsional dan tidak harus di install. Namun karena erat kaitannya dengan Laravel mix serta perintah npm, saya ingin membahasnya sekilas. Terlebih nanti ada 1 materi tentang *authentication* yang mengharuskan kita menginstall Laravel UI.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

10.1. Pengertian Laravel UI

Laravel UI adalah komponen khusus yang dipakai untuk menginstall framework front end seperti Bootstrap, Vue atau React ke dalam Laravel. Komponen ini sepenuhnya opsional karena kita tetap bisa menginstall framework tersebut secara manual. Dari halaman dokumentasinya, Laravel UI ini disebut juga sebagai [JavaScript & CSS Scaffolding](#).

Laravel UI mendukung 3 jenis framework front-end (disebut sebagai 'preset'), yakni: **Bootstrap**, **Vue** dan **React**.

10.2. Instalasi Laravel UI

Agar bisa menggunakan Laravel UI, kita harus menginstalnya terlebih dahulu. Seperti banyak hal di Laravel, proses instalasi ini juga menggunakan **composer**.

Silahkan buka **cmd**, masuk ke folder instalasi Laravel, kemudian ketik kode berikut:

```
composer require laravel/ui="^3.4"
```

Proses instalasi akan berlangsung beberapa saat.

```
C:\xampp\htdocs\laravel01>composer require laravel/ui="^3.4"
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking laravel/ui (v3.4.6)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing laravel/ui (v3.4.6): Extracting archive
```

Gambar: Proses instalasi Laravel UI

Setelah selesai, kita bisa memilih akan menggunakan framework yang mana. Secara bawaan, di versi Laravel UI terdapat 3 pilihan: **Bootstrap**, **Vue** atau **React**.

Untuk melihat daftar pilihan ini, jalankan perintah berikut:

```
php artisan ui -h
```

Tambahan *flag* `-h` berarti *help* yang akan menampilkan keterangan dari `php artisan ui`.

```
C:\xampp\htdocs\laravel01>php artisan ui -h
Description:
Swap the front-end scaffolding for the application

Usage:
ui [options] [--] <type>

Arguments:
type           The preset type (bootstrap, vue, react) -----^

Options:
--auth          Install authentication UI scaffolding
--option[=OPTION] Pass an option to the preset command (multiple values allowed)
-h, --help       Display help for the given command. When no command is given display help for the list command
-q, --quiet      Do not output any message
-V, --version    Display this application version
--ansi|--no-ansi Force (or disable --no-ansi) ANSI output
-n, --no-interaction Do not ask any interactive question
--env[=ENV]      The environment the command should run under
-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Gambar: Pilihan preset dari Laravel UI

Fokus utama kita ada di `<type>`, dimana terdapat penjelasan: *The preset type (bootstrap, vue,*

react). Inilah daftar 'preset' atau framework front-end yang bisa di install. Kali ini saya akan menginstall bootstrap, maka perintah yang diperlukan adalah:

```
php artisan ui bootstrap
```

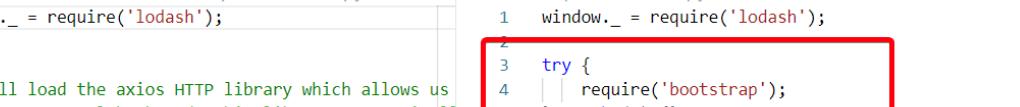
```
C:\xampp\htdocs\laravel01>php artisan ui bootstrap
Bootstrap scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.

C:\xampp\htdocs\laravel01>
```

Gambar: Proses instalasi framework Bootstrap

Setelah menjalankan perintah ini, ada beberapa perubahan yang terjadi.

Pertama, silahkan buka file `resources/js/bootstrap.js`. Dalam bab Laravel Mix, kita tidak menyinggung file ini karena berisi kode internal Laravel, yang salah satunya untuk Laravel UI.



The screenshot shows two tabs in a code editor. The left tab is titled 'bootstrap.js' and contains the original code. The right tab is also titled 'bootstrap.js' and shows the same code with a red box highlighting the try-catch block.

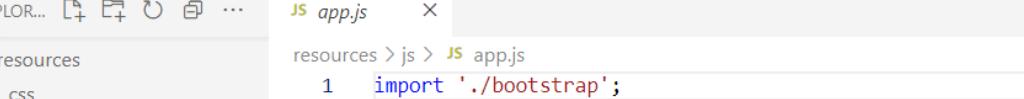
```
C: > xampp > htdocs > laravel02 > resources > js > JS bootstrap.js
JS bootstrap.js X ...
C: > xampp > htdocs > laravel02 > resources > js > JS bootstrap.js
JS bootstrap.js X ...
resources > js > JS bootstrap.js > ...
1 window._ = require('lodash');
2
3 /**
4 * We'll load the axios HTTP library which allows us
5 * to our Laravel back-end. This library automatically
6 * CSRF token as a header based on the value of the 'XSRF-TOKEN' header
7 */
8
9 window.axios = require('axios');
10
11 window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
12
13 /**
14 * Echo exposes an expressive API for subscribing to
15 * events that are broadcast by Laravel. Echo and
16 * allows your team to easily build robust real-time
```

```
1 window._ = require('lodash');
2
3 try {
4   require('bootstrap');
5 } catch (e) {}
6
7 /**
8 * We'll load the axios HTTP library which allows us to easily
9 * to our Laravel back-end. This library automatically handles
10 * CSRF token as a header based on the value of the "XSRF-TOKEN" header
11 */
12
13 window.axios = require('axios');
14
15 window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
```

Gambar: Perbandingan isi file bootstrap.js sebelum (kiri) dan sesudah install Laravel UI Bootstrap (kanan)

Perhatikan terdapat tambahan 3 baris kode program di awal file. Isinya berupa proses import file 'bootstrap' yang diperlukan oleh Bootstrap.

File `bootstrap.js` ini selanjutnya akan di import oleh file `app.js`.



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help, app.js - laravel01 - Visual Studio ...
- Left Sidebar:** Explorer icon, resources folder containing css, js (selected), views, routes, storage.
- Current File:** JS app.js (highlighted in the sidebar). The code content is:

```
resources > js > app.js
1 import './bootstrap';
2
```
- Bottom Status Bar:** Line 1, Column 22, Spaces: 4, UTF-8, LF, {}, JavaScript, Go Live, and several small icons.

Gambar: Isi file app.js

Nantinya, file `app.js` ini akan kita pakai untuk proses kompilasi menggunakan Laravel Mix. Perubahan lain ada di folder `resources/sass`. Dalam bab sebelumnya, folder `sass` kita buat manual. Namun saat instalasi Laravel UI, folder `sass` dibuat otomatis dan sudah berisi 2 file: `_variables.scss` dan `app.scss`.

File `_variables.scss` berisi beberapa variabel Sass:

```
// Body
$body-bg: #f8fafc;
// Typography
$font-family-sans-serif: 'Nunito', sans-serif;
$font-size-base: 0.9rem;
$line-height-base: 1.6;
```

Gambar: Isi file `_variables.scss` dari Laravel UI

Dan berikut isi file `app.scss`:

```
// Fonts
@import url('https://fonts.googleapis.com/css?family=Nunito');
// Variables
@import 'variables';
// Bootstrap
@import '~bootstrap/scss/bootstrap';
```

Gambar: Isi file `app.scss` dari Laravel UI

Terdapat 3 baris `@import` dalam file `app.scss`, yakni:

- `@import url('https://fonts.googleapis.com/css?family=Nunito')`, merupakan perintah untuk proses import font 'Nunito' dari Google Font.
- `@import 'variables'`, adalah perintah untuk proses import file `_variables.scss` yang baru saja kita buka.
- `@import '~bootstrap/scss/bootstrap'`, dipakai untuk proses import file scss bootstrap framework.

Dari ketiga `@import`, yang paling penting hanya perintah ketiga, yakni proses import file Bootstrap. Perintah pertama dan kedua untuk import Font 'Nunito' dan `_variables.scss` bisa

di hapus apabila anda tidak menyukainya style yang dipilih tim Laravel UI. Namun untuk saat ini saya akan biarkan kode tersebut.

Laravel UI juga membuat otomatis file `webpack.mix.js` yang berisi kode berikut:

```
const mix = require('laravel-mix');

/*
| Mix Asset Management
|
| Mix provides a clean, fluent API for defining some Webpack build steps
| for your Laravel application. By default, we are compiling the Sass
| file for the application as well as bundling up all the JS files.
|
*/
mix.js('resources/js/app.js', 'public/js')
    .sass('resources/sass/app.scss', 'public/css')
    .sourceMaps();
```

Gambar: Isi perintah di file webpack.mix.js sebelum (1), dan sesudah instalasi Laravel UI (2)

Perintah `mix.js()` berguna untuk proses compile file JavaScript yang ditulis dalam format EcmaScript 2015 dan juga untuk memproses file **Vue**. Ini berbeda dengan perintah `mix.script()` yang pernah kita gunakan, dimana `mix.script()` lebih dirancang untuk proses compile file JavaScript biasa.

Sedangkan perintah `mix.sass()` berfungsi untuk proses file Sass, yang juga sudah pernah kita praktikkan. Tambahan `mix.sourceMaps()` ikut men-generate file source maps untuk proses debugging.

Selain itu perhatikan file hasil yang merujuk ke alamat folder, yakni 'public/js' dan 'public/css'. Dalam bab sebelumnya, alamat hasil ini kita tulis dalam bentuk file seperti 'public/js/my-app.js'.

Jika ditulis nama folder saja, Laravel mix akan memakai nama file sumber, yakni `app.js` dan `app.scss`, sehingga file akhir Laravel UI akan bernama `public/app.js` dan `public/app.css`.

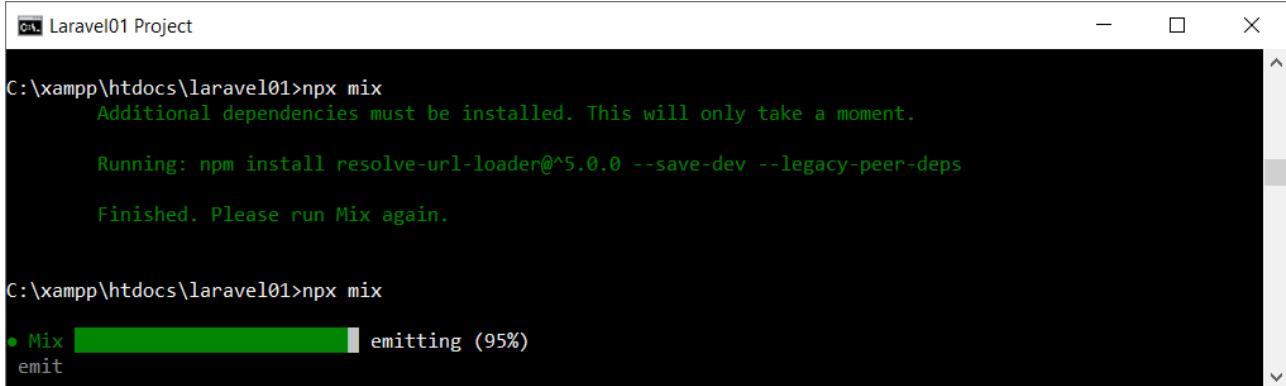
10.3. Menjalankan Laravel UI

Persiapan untuk Laravel UI sudah selesai, langkah selanjutnya menginstall Laravel Mix untuk men-compile file asset. Silahkan buka cmd, masuk ke folder laravel01 dan jalankan 3 perintah berikut secara berurutan:

npm install

```
npm install laravel-mix --save-dev  
npx mix
```

Ketiga perintah ini sudah kita bahas di bab Laravel Mix. Jika terdapat instruksi untuk menjalankan ulang Laravel Mix, ketik sekali lagi `npx mix`:



```
C:\xampp\htdocs\laravel01>npx mix  
Additional dependencies must be installed. This will only take a moment.  
Running: npm install resolve-url-loader@^5.0.0 --save-dev --legacy-peer-deps  
Finished. Please run Mix again.  
  
C:\xampp\htdocs\laravel01>npx mix  
• Mix [██████████] emitting (95%)
```

Gambar: Menjalankan ulang Laravel Mix



```
Laravel Mix v6.0.49  
Compiled Successfully in 7816ms  


| File        | Size     |
|-------------|----------|
| /js/app.js  | 2.23 MiB |
| css/app.css | 200 KiB  |

  
1 WARNING in child compilations (Use 'stats.children: true' resp. '--stats-children' for more details)  
webpack compiled with 1 warning  
  
C:\xampp\htdocs\laravel01>
```

Gambar: Proses compile file asset dengan Larave Mix sudah berhasil

Di akhir proses compile mungkin akan tampil pesan "1 WARNING in child compilations" seperti gambar di atas, pesan ini boleh diabaikan dan tidak berdampak ke kode program kita.

Untuk memastikan, silahkan buka folder **public**. Di dalamnya akan terlihat folder **js** yang berisi file `app.js` serta folder **css** yang berisi file `app.css`. Inilah file hasil compile Laravel mix.

Agar lebih yakin file framework Bootstrap sudah bisa diakses, saya akan pakai kode yang sama seperti bab sebelumnya. Silahkan edit view `welcome.blade.php` dengan kode berikut:

```
resources/view/welcome.blade.php
```

```
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4    <meta charset="UTF-8">  
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
6    <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

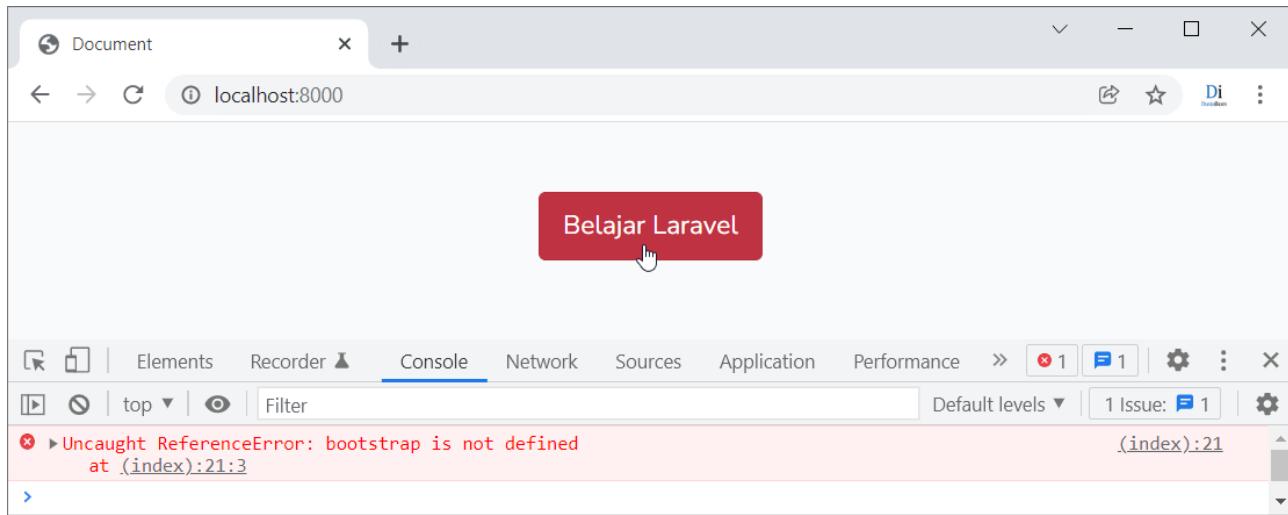
```

7   <title>Document</title>
8   <link rel="stylesheet" href="{{ asset('/css/app.css') }}">
9   </head>
10  <body>
11    <div class="container text-center py-5">
12      <button id="myPopover" type="button" class="btn btn-lg btn-danger"
13          data-bs-toggle="popover" title="Lagi serius..."
14          data-bs-content="Buku Laravel Uncover top bgt!">
15          Belajar Laravel </button>
16    </div>
17  <script src="{{ asset('/js/app.js') }}"></script>
18  <script>
19    // Tampilkan popover Bootstrap
20    new bootstrap.Popover(document.getElementById('myPopover'));
21  </script>
22  </body>
23  </html>

```

Sedikit perubahan ada di baris 8 dan 17. Saya kembali menggunakan function helper `asset()`, serta alamat file sekarang ada di `/css/app.css` dan `/js/app.js`.

Selain itu di baris 18 – 21 terdapat sedikit kode JavaScript untuk uji coba komponen popover Bootstrap. Sebelumnya kode ini berada di file tersendiri (`my-script.js`), namun untuk mempersingkat pembahasan, langsung saja saya tulis ke dalam file view. Berikut hasilnya:



Gambar: Mencoba kode Bootstrap hasil Laravel UI

Tag `<button>` tampil dengan warna merah sebagai bukti kode CSS Bootstrap sudah berhasil diakses. Dibandingkan dengan hasil dari bab Laravel mix, font terlihat sedikit berbeda karena sudah memakai font 'Nunito' bawaan laravel UI.

Akan tetapi ketika tab console di developer tools di buka, tampil pesan error `Uncaught ReferenceError: bootstrap is not defined`. Efek lain, saat tombol "Belajar Laravel" di klik juga tidak tampil popover.

Setelah investigasi lebih lanjut, error ini berasal dari bug Laravel UI karena proses import file

JS Bootstrap tidak menyertakan alias. Akibatnya, object bootstrap yang coba di akses pada baris 20 tidak terdefinisi:

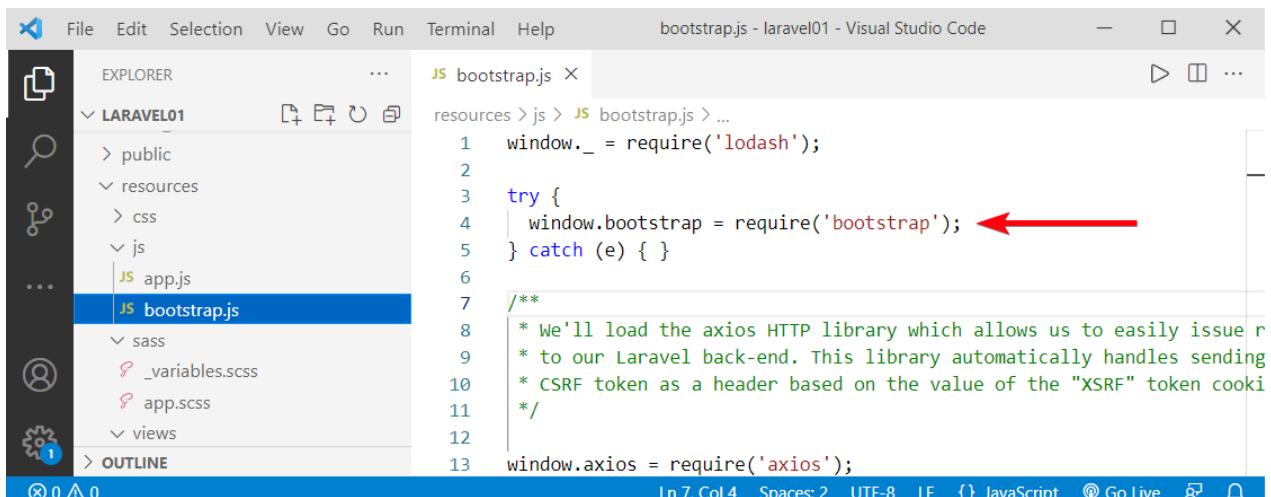
```
new bootstrap.Popover(document.getElementById('myPopover')); // error
```

Solusinya, buka file resources\js\bootstrap.js, lalu cari baris berikut:

```
try {
  require('bootstrap');
} catch (e) { }
```

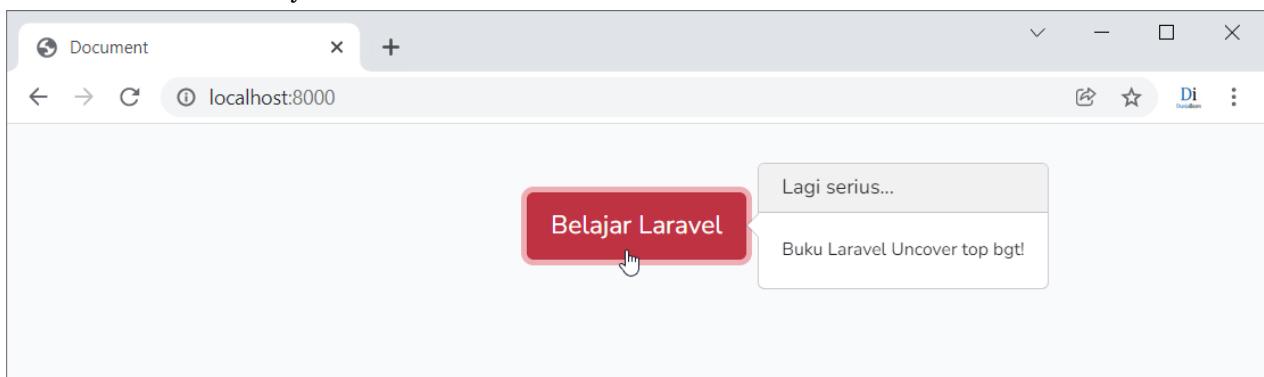
Kemudian modifikasi menjadi:

```
try {
  window.bootstrap = require('bootstrap');
} catch (e) { }
```



Gambar: Modifikasi file resources\js\bootstrap.js

Save file ini dan jalankan ulang Laravel Mix dengan perintah `npx mix`. Setelah selesai, tes kembali file sebelumnya:



Gambar: Tampilan view welcome dengan file Bootstrap yang berasal dari Laravel UI

Sip, sekarang jendela popover sukses tampil ketika tombol "Belajar Laravel" di klik. Mudah-mudahan tim Laravel UI segera memperbaiki masalah ini.

Pentingnya Proses Minify

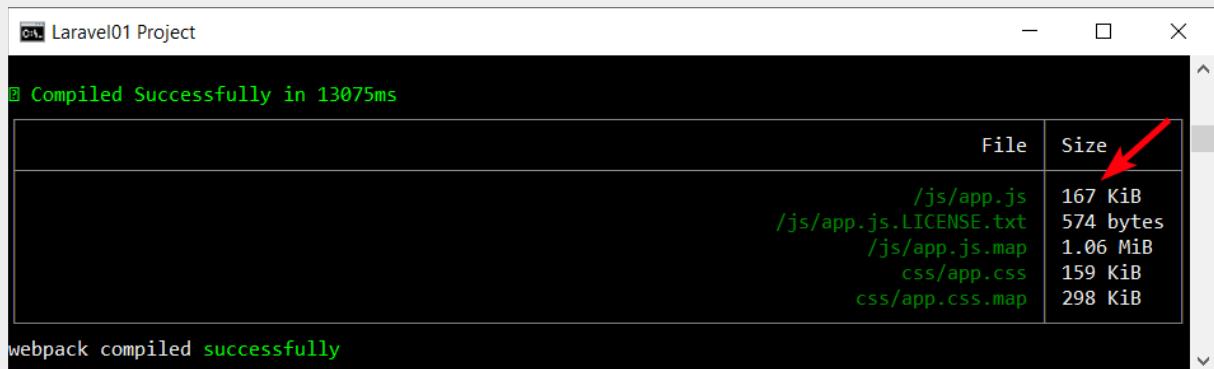
Jika diperhatikan ukuran file `/js/app.js` sangat besar, lebih dari 2 MB yang berasal dari berbagai komponen JavaScript yang dibawa Laravel UI.



File	Size
<code>/js/app.js</code>	2.23 MiB
<code>css/app.css</code>	202 KiB

Gambar: Ukuran file hasil compile dari npx mix

Jika sudah masuk tahap *production*, tentu ini tidak efisien karena proses loading website menjadi lambat. Salah satu solusinya adalah men-minify file tersebut dengan perintah `npx mix -p`.



File	Size
<code>/js/app.js</code>	167 KiB
<code>/js/app.js.LICENSE.txt</code>	574 bytes
<code>/js/app.js.map</code>	1.06 MiB
<code>css/app.css</code>	159 KiB
<code>css/app.css.map</code>	298 KiB

Gambar: Ukuran file hasil compile dari npx mix -p

Sekarang ukuran file mengecil menjadi 166kb. Tambahan file lain seperti `app.js.LICENSE.txt` dan `app.js.map` hanya untuk proses debugging saja dan boleh dihapus.

Cara yang lebih baik bisa juga memangkas file yang tidak diperlukan. Jika kita hanya butuh framework CSS saja, maka tidak perlu pakai komponen tambahan dari Laravel UI, cukup compile langsung seperti yang kita praktekkan di akhir bab Laravel Mix.

Dalam bab ini kita telah melihat cara penggunaan dari Laravel UI. Laravel UI memang tidak harus dipakai, hanya sebagai alternatif untuk menginstall framework Bootstrap, Vue atau React dengan cepat. Kita akan kembali ke Laravel UI ini pada saat masuk materi tentang *authentication*.

Mulai dari bab berikutnya, kita akan lebih banyak membahas sisi back-end dari Laravel, yang dimulai dengan **Controller**.

11. Controller

Kali ini kita masuk ke bagian kedua dari konsep **MVC**, yakni **Controller**. Selain View, Controller adalah tempat dimana kita akan banyak menghabiskan waktu dalam penulisan kode program, karena di sinilah alur logika program dibuat.

Agar seragam dan menghindari error akibat praktik dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

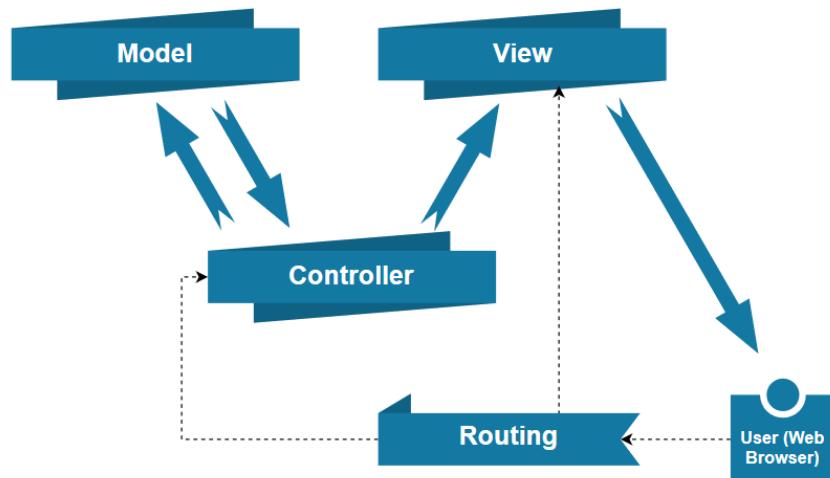
Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css**, karena kita akan memakai sedikit kode Bootstrap.

11.1. Pengertian Controller

Secara sederhana, **controller** adalah bagian kode program yang mengatur logika serta lalu lintas data. Mari lihat kembali diagram alur MVC berikut:



Gambar: Diagram Alur MVC + Routing

Controller berada di pusat MVC. Pada saat user mengetik sebuah alamat di web browser, route akan membaca alamat tersebut dan (idealnya) memanggil controller yang sesuai.

Di dalam controller inilah logika program kita tulis. Apabila butuh mengambil data dari database, controller akan mengakses model. Hasil dari model dikembalikan ke controller yang bisa diolah lebih lanjut untuk kemudian dikirim ke view. Sesampainya di view, data tinggal ditampilkan ke web browser.

Apa yang bisa dilakukan controller sangat banyak. Hampir semua pembahasan hingga akhir buku nanti akan berhubungan dengan controller. Dalam bab ini kita sekedar melihat bagaimana cara menulis struktur dasar serta cara menjalankan controller di Laravel.

11.2. Cara Mengakses Controller

Dalam Laravel, route adalah 'penerima tamu' dari setiap alamat yang kita ketik di web browser. Route-lah yang menentukan proses mana yang akan menangani URL tersebut, apakah langsung ke View, atau ke Controller.

Jika langsung ke View, kita sudah pelajari caranya yakni dengan menulis sebuah **closure (anonymous function)** yang mengembalikan function `view()`:

`routes/web.php`

```
1 Route::get('/', function () {
2     return view('welcome');
3 });
```

Di Laravel 9, untuk memanggil Controller ditulis dengan format berikut:

```
Route::get('<url>', [App\Http\Controllers\Nama_Controller::class, 'nama_method']);
```

Sebagai contoh, untuk memanggil method `index()` di controller bernama **PageController**, penulisan route-nya adalah:

```
Route::get('/', [App\Http\Controllers\PageController::class, 'index']);
```

Dengan penulisan ini, maka ketika alamat `http://localhost:8000` di akses, route akan menjalankan method `index()` milik `PageController`.

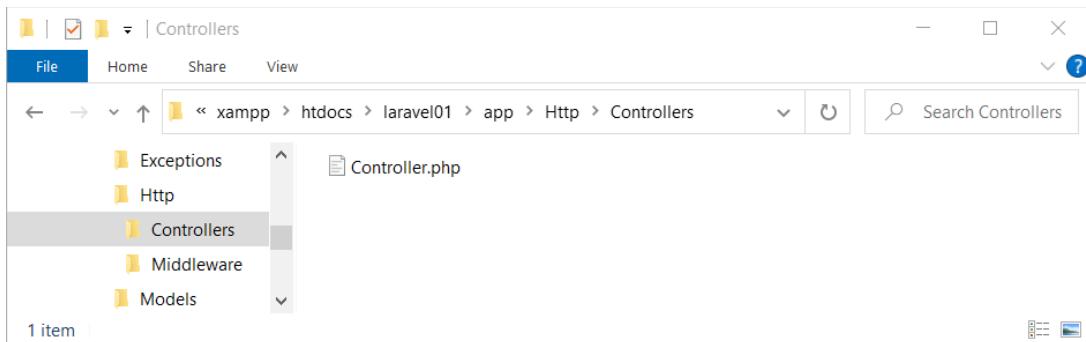
Contoh lain, apabila kita ingin alamat URL `http://localhost:8000/mahasiswa` akan menjalankan method `tampil()` milik `PageController`, penulisan routenya adalah:

```
Route::get('/mahasiswa', [App\Http\Controllers\PageController::class, 'tampil']);
```

Folder app\Http\Controllers

Dalam laravel, controller berbentuk sebuah file yang berisi 1 object dengan berbagai method. Di manakah posisi file controller ini? Bawaan Laravel, semua file controller berada di folder `app\Http\Controllers\`. Silahkan buka folder ini.

Controller



Gambar: Isi folder app\Http\Controllers

Ketika pertama kali mempelajari Laravel, saya agak heran kenapa folder sepenting controller 'terpendam' cukup dalam, yakni di dalam folder `app`, kemudian masuk ke folder `Http`, dan masuk lagi ke folder `Controllers`. Tapi memang struktur folder seperti ini memiliki makna tersendiri karena semua file di Laravel saling berkolaborasi satu sama lain.

Di dalam folder `Controllers` sudah ada 1 file bernama `Controller.php`. File `Controller.php` merupakan file induk dari semua controller yang akan dibuat. Berikut isi dari file ini:

app/Http/Controllers/Controller.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
6 use Illuminate\Foundation\Bus\DispatchesJobs;
7 use Illuminate\Foundation\Validation\ValidatesRequests;
8 use Illuminate\Routing\Controller as BaseController;
9
10 class Controller extends BaseController
11 {
12     use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
13 }
```

Di baris 3 terdapat deklarasi `namespace`, yang artinya file `Controller.php` berada di dalam namespace `App\Http\Controllers`. Ini bersesuaian dengan struktur folder tempat file `Controller.php` berada, yakni di dalam folder `app\Http\Controllers\`.

Di baris 5 – 8 berisi pemanggilan 4 class menggunakan keyword `use`. Pengertian sederhana, keyword `use` dipakai untuk meng-import suatu class ke dalam file saat ini. Artinya terdapat 4 class baru yang diimport, yakni `AuthorizesRequests`, `DispatchesJobs`, `ValidatesRequests` dan `BaseController`.

Lanjut ke baris 10 – 13, terdapat kode untuk membuat class `Controller` yang di `extends` dari class `BaseController`. Isi dari class hanya 1 baris yang dipakai untuk mendeklarasikan penggunaan class `AuthorizesRequests`, `DispatchesJobs`, dan `ValidatesRequests`.

Sepanjang pembahasan nanti, file `Controller.php` tidak perlu di edit sama sekali. Yang akan

kita lakukan adalah meng-extends class Controller ini ke controller yang di buat.

Mungkin anda sedikit kesulitan memahami beberapa istilah yang ada, yakni *class*, *method*, *namespace*, *use*, serta *extends*. Semua ini adalah istilah yang dipakai dalam **object oriented programming**, atau sering disingkat sebagai **OOP**.

Idealnya sebelum masuk ke framework seperti Laravel, harus punya basic yang kuat dulu di OOP PHP. Jalur belajar paling pas adalah: pahami 5 materi dasar web programming yakni HTML, CSS, PHP, MySQL dan JavaScript, lanjut belajar OOP PHP, dan baru sampai ke framework PHP seperti Laravel.

Sebagian ada yang melompati materi OOP dan langsung ke framework. Meskipun tidak dilarang, tapi ini kurang pas karena hampir semua framework menggunakan konsep OOP. Jadi jika anda sedikit kesulitan memahami istilah-istilah yang ada, bisa mundur sejenak untuk belajar dasar OOP PHP.

11.3. Membuat Controller Secara Manual

Terdapat 2 cara pembuatan controller, yakni secara manual dari text editor, atau menggunakan perintah `php artisan`. Kita akan bahas cara manual terlebih dahulu.

Silahkan buat file baru dengan nama `PageController.php`, lalu simpan di folder `app\Http\Controllers`. Sehingga alamat file ada di: `app\Http\Controllers\PageController.php`. Isi file ini dengan kode berikut:

`app\Http\Controllers\PageController.php`

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 class PageController extends Controller
6 {
7     public function index()
8     {
9         return "Halaman Home";
10    }
11
12    public function tampil()
13    {
14        return "Data Mahasiswa";
15    }
16 }
```

Di baris 3 terdapat pendeklarasian namespace `App\Http\Controllers`. Ini sama seperti yang ada di file `Controller.php`. Karena memiliki namespace yang sama, kita bisa langsung mengakses semua kode yang ada di `Controller.php`.

Struktur namespace ini juga bersesuaian dengan alamat **path**, yakni folder `app\Http\Controllers\`. Jika nantinya file controller di pindah ke folder lain, penulisan namespace ini juga harus disesuaikan.

Semua controller di Laravel harus menggunakan namespace, karena ini berfungsi untuk menghindari bentrok jika terdapat nama class yang sama.

Untuk saat ini bisa disebut bahwa jika file controller disimpan dalam folder `app\Http\Controllers\`, maka semua controller **harus** mendeklarasikan perintah `namespace App\Http\Controllers` di baris paling atas.

Selanjutnya di baris 5 saya membuat class `PageController` yang meng-extends `Controller`. Artinya, class `PageController` ini adalah **turunan** atau child class dari class `Controller`. Kembali, class `Controller` berasal dari file `Controller.php` yang sudah kita lihat sebelumnya. Di dalam Laravel, setiap controller **wajib** meng-extends class `Controller`.

Class `PageController` memiliki 2 buah method: `index()` dan `tampil()`. Isi kedua method hanya mengembalikan string sederhana. Tergantung kebutuhan, kita bisa membuat berbagai method lain.

Nama `PageController` sendiri bukan bagian dari Laravel, tapi nama yang saya pilih. Bisa saja nama controller ini diganti dengan `MahasiswaController`, `HalamanController`, dsb. Pemilihan nama `PageController` sendiri cukup sering dipakai karena bersifat umum, yakni controller yang ditujukan untuk membuat *page* (halaman).

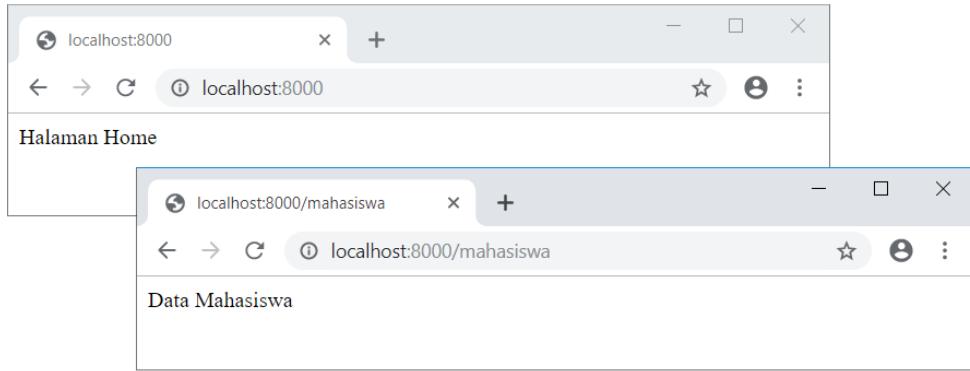
Sekarang mari kita coba akses. Tambah dua routes berikut ke dalam file `routes/web.php`:

`routes/web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', [App\Http\Controllers\PageController::class, 'index']);
6 Route::get('/mahasiswa',[App\Http\Controllers\PageController::class, 'tampil']);
```

Sebagai pengulangan, route di baris 5 artinya jika halaman `http://localhost:8000` diakses, eksekusi method `index()` yang ada di `PageController`.

Sedangkan route di baris 6 berarti jika halaman `http://localhost:8000/mahasiswa` diakses, eksekusi method `tampil()` yang ada di `PageController`. Berikut hasilnya:



Gambar: Hasil pengaksesan controller PageController

Sip, controller sudah berhasil di akses.

Sebagai rangkuman, berikut alur pembuatan controller di Laravel:

1. Tulis route yang akan mengakses sebuah method milik controller.
2. Buat file controller di folder `App\Http\Controllers`.
3. Rancang isi controller beserta methodnya.

Aturan Penamaan Controller

Dalam Laravel terdapat cukup banyak aturan penamaan. Beberapa ada yang bersifat wajib, yakni harus ditulis dengan aturan tertentu, namun ada juga yang lebih ke saran. Meskipun disebut sebagai saran, sebaiknya kita tetap mengikuti penamaan itu karena sudah menjadi 'kesepakatan tidak resmi' dari banyak programmer Laravel.

Kita akan membahas aturan penamaan ini pada saat masuk ke materi yang berhubungan. Namun jika anda tertarik, bisa baca [laravel-best-practices](#). Halaman tersebut memang bukan dokumentasi resmi laravel, tapi lebih 'best practice' dari komunitas.

Sebenarnya kita boleh bebas ingin menulis nama controller dengan apa saja, misalnya `Mahasiswa`, `Mobil`, atau yang lain. Tapi *best practice*-nya adalah menggunakan format **<nama>+Controller**.

Jadi dari pada membuat class `Mobil`, sebaiknya buat class `MobilController`. Yakni dengan tambahan 'ujung' atau suffix 'Controller', contoh lain seperti `MahasiswaController`, `BarangController`, `LoginController`, dst.

Nama controller juga sebaiknya ditulis dengan **CamelCase**, yakni menggunakan huruf kapital di setiap kata dan tidak menggunakan underscore '_'. Karakter pertama dari nama controller juga harus huruf besar, seperti `MahasiswaEkonomiController`, dan bukan `Mahasiswa_Ekonomi_Controller` maupun `mahasiswaEkonomiController`.

Nama class controller harus sama dengan nama file. Misalnya kita ingin membuat class

`MahasiswaController`, maka harus disimpan dalam file `MahasiswaController.php`.

Nama Controller disarankan berbentuk *singular* (tunggal), bukan *plural* (jamak). Ini sebenarnya lebih ke *grammer* bahasa Inggris, misalnya lebih baik menulis `PageController` dan bukan `PagesController`, atau `ItemController` dan bukan `ItemsController`. Namun beberapa programmer ada juga yang lebih menyukai menggunakan plural (jamak).

Bagi kita yang sehari-hari menggunakan Bahasa Indonesia, perbedaan kata singular dan plural ini memang agak aneh, karena kosa kata kita tidak mengenal perbedaan ini.

Namun karena Laravel mayoritas dirancang oleh programmer dari Amerika / Eropa, maka pada materi tertentu kita terpaksa meng-konversi kata-kata Indonesia ke dalam bentuk plural dengan penambahan huruf 's', seperti `Mahasiswas`, `Dosens`, atau `Barangs`. Ini nantinya terpakai ketika sudah masuk ke **Model**.

11.4. Cara Penulisan Route Untuk Controller

Salah satu perubahan penting di Laravel 8 (dan juga Laravel 9) ada di cara penulisan alamat route yang dipakai untuk mengakses controller. Di Laravel 7 ke bawah, format penulis route bisa langsung ditulis dalam bentuk string:

```
Route::get('<alamat_url>', '<nama_controller>@<nama_method>');
```

Misalnya untuk mengakses method `index()` milik `PageController`, routenya adalah:

```
Route::get('/', 'PageController@index');
```

Sedangkan di Laravel 8 ke atas, harus ditulis lengkap beserta namespace file controller, yakni "`App\Http\Controllers\`", kemudian baru nama class controller beserta nama methodnya:

```
Route::get('/', [App\Http\Controllers\PageController::class, 'index']);
```

Saya kurang tau apa alasan tim Laravel dibalik keputusan ini. Karena seperti yang terlihat, kode programnya menjadi lebih panjang.

Alternatif penulisan lain, namespace "`App\Http\Controllers\PageController`" bisa di pindah ke bagian atas file `routes\web.php`. Sebagai contoh, dalam praktek sebelumnya file `routes\web.php` sudah berisi 2 route berikut:

`routes/web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', [App\Http\Controllers\PageController::class, 'index']);
6 Route::get('/mahasiswa',[App\Http\Controllers\PageController::class, 'tampil']);
```

Dengan memindahkan namespace "App\Http\Controllers\PageController" ke bagian atas, file route bisa ditulis seperti ini:

routes/web.php

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\PageController;
5
6 Route::get('/', [PageController::class, 'index']);
7 Route::get('/mahasiswa',[PageController::class, 'tampil']);

```

Di baris 4 terdapat perintah `use App\Http\Controllers\PageController` yang dipakai untuk proses import namespace class `PageController`. Dengan perintah ini maka di baris 6 dan 7 kita tidak perlu lagi menulis lengkap `App\Http\Controllers\PageController::class`, tapi cukup `PageController::class` saja.

Jika nantinya ada file controller kedua, namespace-nya juga harus di definisikan kembali di baris paling atas.

Laravel 9 sebenarnya tetap bisa menerima penulisan route yang singkat seperti di Laravel 7, namun ada file konfigurasi yang harus diubah. Mengenai caranya, akan kita bahas di akhir bab nanti.

11.5. Mengakses View dari Controller

Sebuah controller pada dasarnya terdiri dari kumpulan method. Di dalam method inilah kita bisa melakukan banyak hal, misalnya langsung menampilkan teks dengan perintah `return` seperti contoh kita sebelumnya.

Namun yang paling ideal adalah, mengembalikan sebuah View. Caranya juga sangat mudah karena sama persis seperti yang sering kita lakukan di route. Sebagai contoh, saya akan modifikasi kembali `PageController.php`:

app/Http/Controllers/PageController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 class PageController extends Controller
6 {
7     public function index()
8     {
9         return view('welcome');
10    }
11

```

Controller

```
12     public function tampil()
13     {
14         return "Data Mahasiswa";
15     }
16 }
```

Perhatikan isi method `index()`, yakni perintah `return view('welcome')`. Artinya ketika method ini diakses, tampilkan halaman `welcome.blade.php`.

Bagaimana dengan mengirim data ke view? Caranya juga sama persis di route:

app/Http/Controllers/PageController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 class PageController extends Controller
6 {
7     public function index()
8     {
9         return view('welcome');
10    }
11
12    public function tampil()
13    {
14        $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
15        "Lisa Permata"];
16
17        return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
18    }
19 }
```

Dalam method `tampil()` saya mengirim sebuah array ke view `mahasiswa`. Dapat dilihat caranya sama seperti yang kita pelajari di bab tentang route, yakni menggunakan method `with()`.

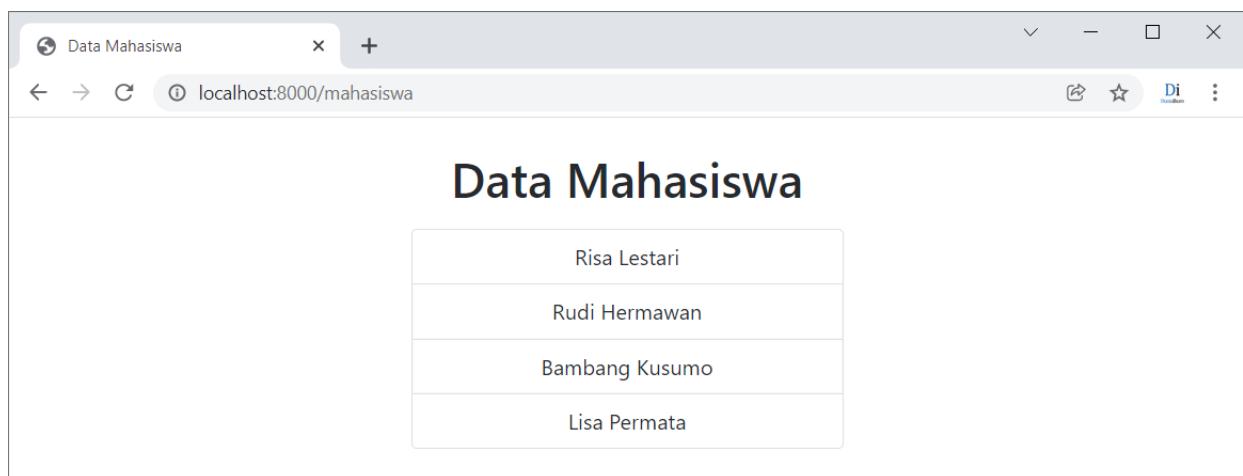
Dan berikut isi view `mahasiswa`:

resources/views/mahasiswa.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="{{asset('/css/bootstrap.min.css')}}" rel="stylesheet">
8     <title>Data Mahasiswa</title>
9 </head>
10 <body>
11
12 <div class="container text-center p-4 bg-white">
13     <h1 class="mb-3">Data Mahasiswa</h1>
14     <div class="row">
```

Controller

```
15 <div class="col-sm-8 col-md-6 m-auto">
16   <ol class="list-group">
17     @forelse ($mahasiswa as $val)
18       <li class="list-group-item">{{$val}}</li>
19     @empty
20       <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
21     @endforelse
22   </ol>
23 </div>
24 </div>
25 </div>
26
27 </body>
28 </html>
```



Gambar: Tampilan view mahasiswa

Semua hal tentang pengiriman data dari route ke view juga bisa diterapkan di controller ke view, sehingga tidak perlu kita bahas lagi.

Lompat Antar File Laravel

Jika anda menginstall **Laravel Extension Pack** di VS Code, terdapat fitur untuk "lompat antar file" langsung dari kode program.

Caranya, tahan tombol **Ctrl** dan arahkan cursor mouse ke alamat path, seperti view. Kode tersebut akan berubah menjadi link yang bisa diklik. Ini sangat memudahkan kita untuk pindah antar file karena tidak perlu mencarinya secara manual.



Gambar: Klik untuk langsung lompat ke view mahasiswa.blade.php

11.6. Memindahkan File Controller

Secara default, semua file controller berada di folder `app\Http\Controllers`. Untuk aplikasi yang besar, bisa saja terdapat 10 atau 100 file controller. Jika sudah seperti ini, akan lebih baik jika file controller diatur ke folder lanjutan agar lebih rapi.

Namun kita tidak bisa sekedar memindahkan file controller karena ada beberapa hal yang mesti disiapkan.

Sebagai bahan praktek, silahkan buat folder '**Admin**' di dalam `app\Http\Controllers`, lalu copy file `PageController.php` ke dalam file ini, sehingga alamat path file berada di `app\Http\Controllers\Admin\PageController.php`. Pembuatan folder admin seolah-olah dipakai untuk menampung semua controller yang berkaitan dengan pemrosesan admin.

Berikut saya tampilkan lagi isi file `PageController.php`:

`app\Http\Controllers\admin\PageController.php`

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 class PageController extends Controller
6 {
7     public function index()
8     {
9         return view('welcome');
10    }
11
12    public function tampil()
13    {
14        $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
15        "Lisa Permata"];
16
17        return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
18    }
19 }
```

Karena file ini sudah di pindah, ada beberapa hal yang harus kita ubah. Pertama, penulisan namespace di baris 3 harus disesuaikan dengan alamat path yang baru, yakni menjadi:

`namespace App\Http\Controllers\Admin;`

Terdapat tambahan **Admin** di akhir namespace. Ini sesuai dengan nama folder tempat file `PageController.php` berada.

Karena namespace sudah berubah, kita juga tidak bisa langsung mengakses class `Controller` untuk di `extends` di baris 5. Alasannya karena class `Controller` tersebut tersimpan di file `Controller.php` yang berada di namespace `App\Http\Controllers`, bukan `App\Http\Controllers\Admin`.

Solusinya, kita harus 'import' file `Controller.php` dengan menggunakan keyword `use` sebagai berikut:

```
use App\Http\Controllers\Controller;
```

Dengan tambahan perintah ini, maka class Controller sudah kembali bisa di extends. Berikut kode program lengkap dari `PageController.php` setelah penambahan:

app/Http/Controllers/Admin/PageController.php

```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6
7 class PageController extends Controller
8 {
9     public function index()
10    {
11        return "Halaman Home Admin";
12    }
13
14    public function tampil()
15    {
16        return "Data Mahasiswa Admin";
17    }
18 }
```

Perubahannya ada di baris 3 dan 5.

Kemudian, bagaimana cara mengakses file ini dari route? Cukup tambah nama folder "Admin" ke dalam penulisan namespace:

routes/web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\Admin\PageController;
5
6 Route::get('/', [PageController::class, 'index']);
7 Route::get('/mahasiswa',[PageController::class, 'tampil']);
```

Di baris 4, perintah import namespace sekarang berubah dari sebelumnya `use App\Http\Controllers\PageController` menjadi `use App\Http\Controllers\Admin\PageController`.

Atau jika kita tidak ingin menggunakan penulisan namespace terpisah, bisa seperti ini:

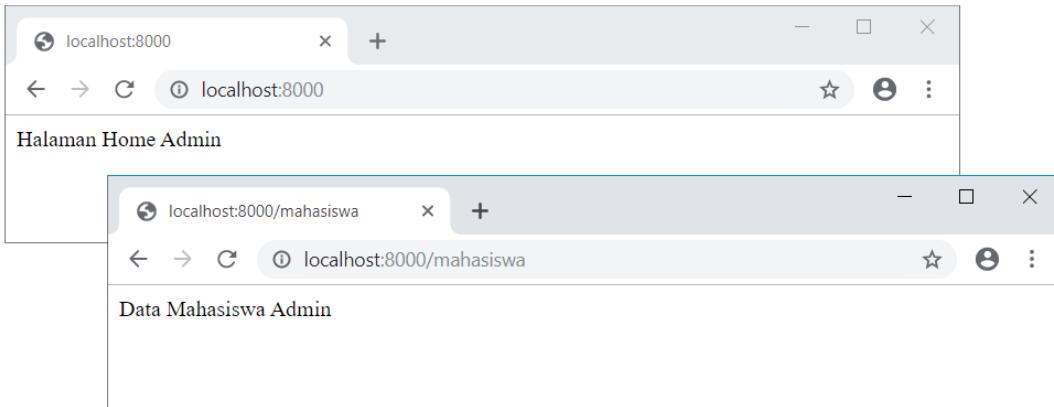
routes/web.php

```
1 <?php
2
```

```

3  use Illuminate\Support\Facades\Route;
4
5  Route::get('/', [App\Http\Controllers\Admin\PageController::class, 'index']);
6  Route::get('/mahasiswa',[App\Http\Controllers\Admin\PageController::class,
7                  'tampil']);

```



Gambar: Hasil tampilan Admin\PageController

11.7. Mengakses Laravel Facade

Di bab View kita sudah membahas sedikit tentang **Facade**, yakni istilah yang dipakai Laravel untuk menyebut suatu class yang berisi berbagai static method. Bisa juga dipahami bahwa facade adalah kumpulan *helper function* yang 'dibungkus' dalam bentuk class.

Sama seperti function biasa di PHP, facade dipakai untuk memudahkan pembuatan kode program. Ini akan sangat membantu proses pengolahan data di controller.

Facade di akses menggunakan *namespace*, sehingga tersedia 2 cara pemanggilan: menulis langkap alamat class facade pada saat dipanggil, atau di deklarasikan secara global menggunakan perintah *use*.

Hampir semua file **facade** di simpan dalam folder **vendor**, yakni folder yang ukurannya mencakup 99% dari seluruh Laravel framework (sebelum meng-install `node_modules`).

Sebagai bahan praktek, saya akan tambah method `cobaFacade()` ke dalam controller `Admin\PageController.php`, lalu membuat route untuk mengaksesnya:

app/Http/Controllers/Admin/PageController.php

```

1  <?php
2
3  namespace App\Http\Controllers\Admin;
4
5  use App\Http\Controllers\Controller;
6
7  class PageController extends Controller

```

Controller

```
8  {
9      public function index()
10     {
11         return "Halaman Home Admin";
12     }
13
14     public function tampil()
15     {
16         return "Data Mahasiswa Admin";
17     }
18
19     public function cobaFacade()
20     {
21         // kode akan kita tulis disini....
22     }
23
24 }
```

routes/web.php

```
1 Route::get('/coba-facade',[PageController::class,'cobaFacade']);
```

Di dalam method `cobaFacade()` inilah kita akan tulis beberapa kode program.

Facade adalah class berisi berbagai kumpulan method bantu. Di manakah bisa melihat daftar method ini?

Daftar facade bisa diakses dari halaman [Laravel API](#) atau bisa juga dari [Facade Class Reference](#). Namun dokumentasi ini memang agak susah ditelusuri terutama bagi pemula. Terlebih tidak ada contoh cara penggunaan (hanya penjelasan singkat saja).

Akses Facade Secara Langsung

Yang dimaksud dengan mengakses facade secara langsung adalah menulis lengkap alamat namespace beserta class pada saat digunakan.

Sebagai contoh, di Laravel terdapat **Str** facade yang berisi berbagai method yang berhubungan dengan pemrosesan string. Class **Str** ini berada di alamat **Illuminate\Support\Str**, sehingga pemanggilan methodnya adalah:

```
\Illuminate\Support\Str::<nama_method>
```

Berikut contoh pemanggilan method dari **Str** facade ini:

app/Http/Controllers/Admin/PageController.php

```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
```

```

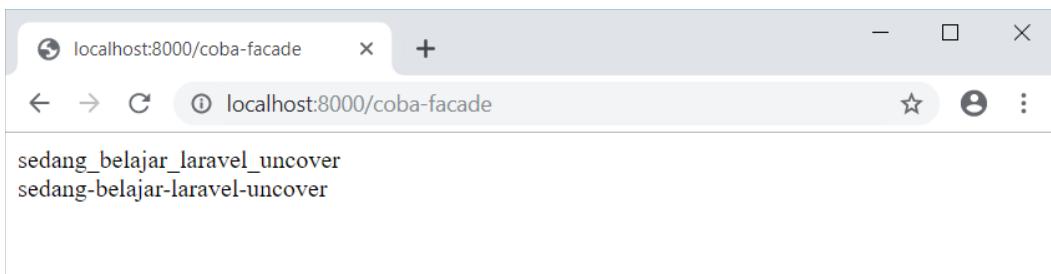
6
7 class PageController extends Controller
8 {
9     //...
10    public function cobaFacade()
11    {
12        echo \Illuminate\Support\Str::snake('SedangBelajarLaravelUncover');
13        echo "<br>";
14        echo \Illuminate\Support\Str::kebab('SedangBelajarLaravelUncover');
15    }
16
17 }

```

Di baris 12 dan 14 saya menjalankan method `snake()` dan `kebab()` milik **Str** facade.

Method `Str::snake()` dipakai untuk mengkonversi string menjadi *snake case*, yakni penulisan string dimana setiap kata dipisah dengan underscore `_`. Sedangkan method `Str::kebab()` dipakai untuk mengkonversi string menjadi *kebab case*, dimana antar kata ditulis dengan tanda hubung `-`.

Berikut hasilnya:



Gambar: Hasil penggunaan `Str::snake()` dan `Str::kebab()`

Akses Facade Dengan Perintah use

Cara kedua untuk menggunakan facade adalah men-deklarasikan namespace dan nama class di awal program menggunakan perintah `use`. Berikut konversi dari percobaan kita sebelumnya:

app/Http/Controllers/Admin/PageController.php

```

1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Support\Str;
7
8 class PageController extends Controller
9 {
10     //...
11     public function cobaFacade()
12     {
13         echo Str::snake('SedangBelajarLaravelUncover');

```

```

14     echo "<br>";
15     echo Str::kebab('SedangBelajarLaravelUncover');
16 }
17
18 }

```

Pendeklarasian facade ada di baris 6, yakni perintah use Illuminate\Support\Str. Dengan perintah ini, maka sepanjang kode program kita bisa langsung menulis nama class dan method, tanpa perlu menulis lengkap alamat facade.

Di dalam Laravel tersedia cukup banyak class facade, beberapa diantaranya akan kita bahas secara bertahap. Yang perlu diperhatikan adalah, jika dalam kode program terdapat pemanggilan class, maka namespace-nya **harus di deklarasikan** terlebih dahulu (jika tidak akan tampil error), kecuali beberapa class bawaan yang bisa dipakai secara langsung.

Sekilas Tentang Penamaan Case

Dalam programming terdapat beberapa istilah untuk menyebut cara penulisan **identifier**, yakni cara penulisan nama class, nama variabel, nama konstanta, dst. Setiap bahasa pemrograman punya 'kebiasaan' masing-masing.

Berikut beberapa istilah yang sering dipakai:

Camel case: dibuat tanpa spasi dengan karakter pertama dari setiap kata ditulis dengan huruf besar, kecuali kata pertama. Penulisan ini banyak dipakai untuk nama method serta variabel. Contoh: `sedangBelajarLaravelUncover`.

Pascal case dibuat tanpa spasi dengan karakter pertama dari setiap kata ditulis dengan huruf besar, termasuk kata pertama. Penulisan ini banyak dipakai untuk nama class. Contoh: `SedangBelajarLaravelUncover`.

Snake case dibuat dengan mengganti spasi antar kata dengan underscore `_`. Penulisan ini banyak dipakai untuk nama variabel di beberapa bahasa pemrograman, seperti PHP procedural, serta untuk penamaan kolom di database. Contoh: `sedang_belajar_laravel_uncover`. Jika snake case ditulis dalam huruf besar semua, sering dipakai sebagai nama konstanta, seperti `PI` atau `KURS_DOLLAR`.

Kebab case dibuat dengan mengganti spasi antar kata dengan karakter minus atau hyphen `-`. Penulisan ini banyak dipakai untuk URL. Contoh: `sedang-belajar-laravel-uncover`.

11.8. Mengakses External Class

Selain mengakses facade, untuk project yang besar kadang kita perlu membuat class sendiri kemudian diakses dari dalam controller. Caranya cukup mudah, hanya perlu mengatur penulisan namespace.

Controller

Sebagai bahan praktek, saya kembali menggunakan `PageController.php` yang ada di folder `Admin`. Kali ini method yang dipakai adalah `cobaClass()`:

app/Http/Controllers/Admin/PageController.php

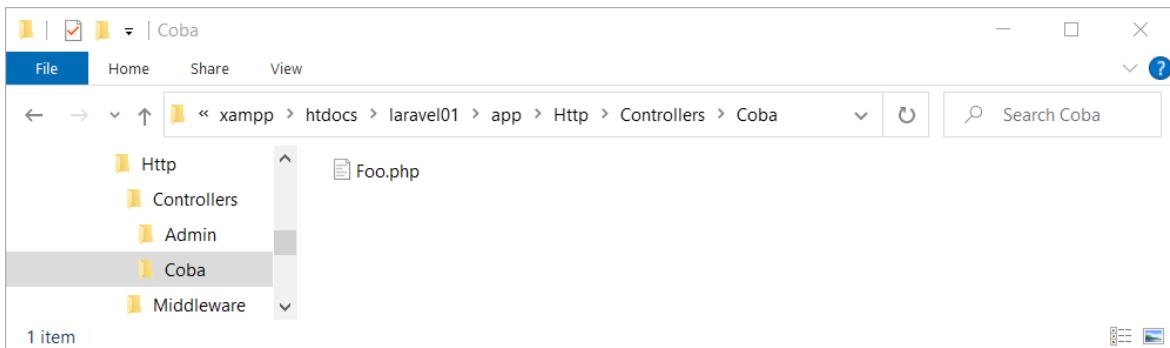
```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6
7 class PageController extends Controller
8 {
9     //...
10    public function cobaClass()
11    {
12        // kode akan kita tulis disini....
13    }
14}
15 }
```

routes/web.php

```
1 Route::get('/coba-class',[PageController::class,'cobaClass']);
```

Di dalam method `cobaClass()` inilah kita akan tulis beberapa kode program.

Selanjutnya, buat folder baru dengan nama **Coba** di dalam `App\Http\Controllers`, kemudian buat file **Foo.php** sehingga alamat path file berada di `App\Http\Controllers\Coba\Foo.php`.



Gambar: Lokasi file Foo.php

Kemudian ketik kode berikut:

app/Http/Controllers/Coba/Foo.php

```
1 <?php
2
3 namespace App\Http\Controllers\Coba;
4
5 class Foo
6 {
7     public function bar()
```

Controller

```
8     {
9         echo "Ini berasal dari Method Bar di dalam class Foo";
10    }
11 }
```

Di baris 3 terdapat perintah pendeklarasian namespace. Nama namespace ini bersesuaian dengan alamat folder tempat file `Foo.php` berada, yakni di `App\Http\Controllers\Coba`. Selanjutnya di baris 5 saya membuat class `Foo` yang berisi 1 method bernama `bar()`. Isi method `bar()` ini hanya perintah `echo` biasa.

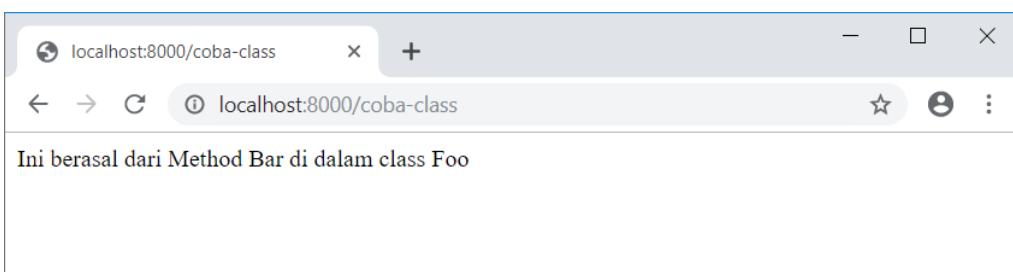
Yang akan kita lakukan adalah, mencoba mengakses class **Foo** dari dalam controller `PageController.php`.

Untuk mengakses class external, caranya hampir sama seperti mengakses class facade, yakni bisa langsung diakses dengan menulis lengkap class namespace, atau menggunakan perintah `use`.

Jika diakses langsung, bisa seperti ini:

`app\Http\Controllers\Admin\PageController.php`

```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6
7 class PageController extends Controller
8 {
9     /**
10     * public function cobaClass()
11     {
12         $foo = new \App\Http\Controllers\Coba\Foo();
13         echo $foo->bar();
14     }
15 }
16 }
```



Gambar: Hasil pemanggilan method `bar()` dari class `Foo`

Di baris 12 saya membuat variabel `$foo` yang dipakai untuk menampung instance atau hasil pembuatan object class **Foo**. Penulisan class ini harus menyertakan nama namespace-nya. Kemudian di baris 13 terdapat perintah untuk mengakses method `bar()` milik object `$foo`. Hasilnya, di web browser tampil teks 'Ini berasal dari Method Bar di dalam class Foo'.

Jika kita menggunakan penulisan langsung seperti ini, jangan lupa untuk selalu menambahkan tanda backslash di awal namespace, yakni \App\Http\Controllers\Coba\Foo(), dan bukan App\Http\Controllers\Coba\Foo().

Atau bisa juga dengan menggunakan perintah `use`:

`app/Http/Controllers/Admin/PageController.php`

```

1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6 use App\Http\Controllers\Coba\Foo;
7
8 class PageController extends Controller
9 {
10     /**
11      * ...
12      *
13      * @return void
14      */
15     public function cobaClass()
16     {
17         $foo = new Foo();
18         echo $foo->bar();
19     }
20 }
```

Dengan deklarasi perintah `use App\Http\Controllers\Coba\Foo` di baris 6, kita bisa langsung mengakses perintah `new Foo()` di baris 13, tidak perlu lagi menulis lengkap namespace-nya.

Pengaksesan class external ini nantinya banyak kita lakukan, terutama jika sudah masuk ke bab tentang Model.

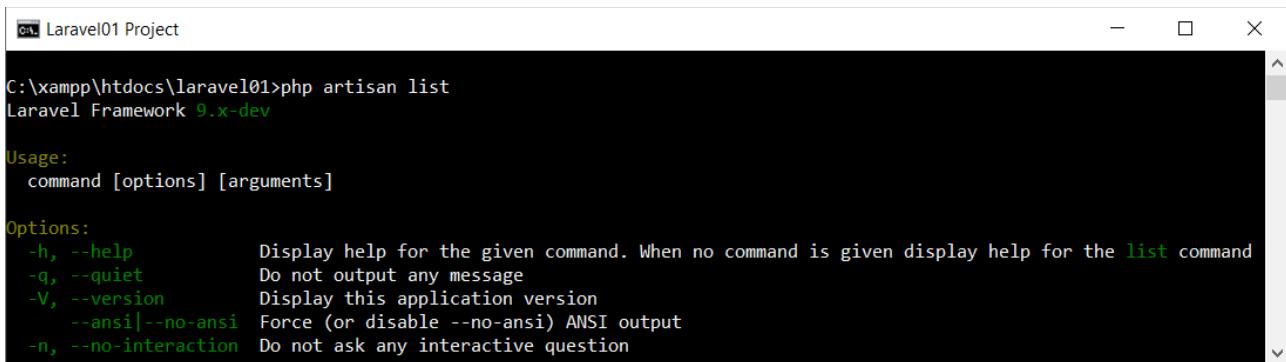
11.9. Membuat Controller dari php artisan

Sampai saat ini kita telah beberapa kali menggunakan perintah **php artisan**, misalnya untuk menjalankan php server dengan perintah `php artisan serve`, melihat daftar route dengan `php artisan route:list`, serta menginstall Laravel UI dengan `php artisan ui`.

Selain itu masih banyak yang bisa kita lakukan dengan perintah `php artisan`, diantaranya membuat file controller, membuat model, membuat migration, dst. Sekilas ini memang terasa ribet karena kita harus bolak balik antara text editor dan cmd, tapi jika sudah terbiasa, perintah `php artisan` ini sangat membantu.

Untuk melihat seluruh daftar perintah yang tersedia, silahkan buka cmd, masuk ke folder laravel dan ketik perintah `php artisan list`:

Controller



```
C:\xampp\htdocs\laravel01>php artisan list
Laravel Framework 9.x-dev

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display help for the given command. When no command is given display help for the list command
  -q, --quiet           Do not output any message
  -V, --version          Display this application version
  --ansi|--no-ansi      Force (or disable --no-ansi) ANSI output
  -n, --no-interaction   Do not ask any interactive question
```

Gambar: Menjalankan perintah php artisan list

Berikut hasil yang saya dapat:

```
C:\xampp\htdocs\laravel01>php artisan list

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display help for the given command. When no command is given display help for the list command
  -q, --quiet           Do not output any message
  -V, --version          Display this application version
  --ansi|--no-ansi      Force (or disable --no-ansi) ANSI output
  -n, --no-interaction   Do not ask any interactive question
  --env[=ENV]             The environment the command should run under
  -v|vv|vvv, --verbose    Increase the verbosity of messages: 1 for normal output,
                           2 for more verbose output and 3 for debug

Available commands:
  clear-compiled          Remove the compiled class file
  completion              Dump the shell completion script
  db                      Start a new database CLI session
  down                    Put the application into maintenance / demo mode
  env                     Display the current framework environment
  help                    Display help for a command
  inspire                 Display an inspiring quote
  list                    List commands
  migrate                 Run the database migrations
  optimize                Cache the framework bootstrap files
  serve                   Serve the application on the PHP development server
  test                    Run the application tests
  tinker                  Interact with your application
  up                      Bring the application out of maintenance mode

auth
  auth:clear-resets       Flush expired password reset tokens
cache
  cache:clear              Flush the application cache
  cache:forget             Remove an item from the cache
  cache:table              Create a migration for the cache database table
config
  config:cache             Create a cache file for faster configuration loading
```

Controller

config:clear	Remove the configuration cache file
db	
db:seed	Seed the database with records
db:wipe	Drop all tables, views, and types
event	
event:cache	Discover and cache the application's events and listeners
event:clear	Clear all cached events and listeners
event:generate	Generate the missing events and listeners based on
registration	
event:list	List the application's events and listeners
key	
key:generate	Set the application key
make	
make:cast	Create a new custom Eloquent cast class
make:channel	Create a new channel class
make:command	Create a new Artisan command
make:component	Create a new view component class
make:controller	Create a new controller class
make:event	Create a new event class
make:exception	Create a new custom exception class
make:factory	Create a new model factory
make:job	Create a new job class
make:listener	Create a new event listener class
make:mail	Create a new email class
make:middleware	Create a new middleware class
make:migration	Create a new migration file
make:model	Create a new Eloquent model class
make:notification	Create a new notification class
make:observer	Create a new observer class
make:policy	Create a new policy class
make:provider	Create a new service provider class
make:request	Create a new form request class
make:resource	Create a new resource
make:rule	Create a new validation rule
make:scope	Create a new scope class
make:seeder	Create a new seeder class
make:test	Create a new test class
migrate	
migrate:fresh	Drop all tables and re-run all migrations
migrate:install	Create the migration repository
migrate:refresh	Reset and re-run all migrations
migrate:reset	Rollback all database migrations
migrate:rollback	Rollback the last database migration
migrate:status	Show the status of each migration
model	
model:prune	Prune models that are no longer needed
notifications	
notifications:table	Create a migration for the notifications table
optimize	
optimize:clear	Remove the cached bootstrap files
package	
package:discover	Rebuild the cached package manifest
queue	
queue:batches-table	Create a migration for the batches database table
queue:clear	Delete all of the jobs from the specified queue

Controller

queue:failed	List all of the failed queue jobs
queue:failed-table	Create a migration for the failed queue jobs database table
queue:flush	Flush all of the failed queue jobs
queue:forget	Delete a failed queue job
queue:listen	Listen to a given queue
queue:monitor	Monitor the size of the specified queues
queue:prune-batches	Prune stale entries from the batches database
queue:prune-failed	Prune stale entries from the failed jobs table
queue:restart	Restart queue worker daemons after their current job
queue:retry	Retry a failed queue job
queue:retry-batch	Retry the failed jobs for a batch
queue:table	Create a migration for the queue jobs database table
queue:work	Start processing jobs on the queue as a daemon
route	
route:cache	Create a route cache file for faster route registration
route:clear	Remove the route cache file
route:list	List all registered routes
sail	
sail:install	Install Laravel Sail's default Docker Compose file
sail:publish	Publish the Laravel Sail Docker files
schedule	
schedule:clear-cache	Delete the cached mutex files created by scheduler
schedule:list	List the scheduled commands
schedule:run	Run the scheduled commands
schedule:test	Run a scheduled command
schedule:work	Start the schedule worker
schema	
schema:dump	Dump the given database schema
session	
session:table	Create a migration for the session database table
storage	
storage:link	Create the symbolic links configured for the application
stub	
stub:publish	Publish all stubs that are available for customization
vendor	
vendor:publish	Publish any publishable assets from vendor packages
view	
view:cache	Compile all of the application's Blade templates
view:clear	Clear all compiled view files

Total terdapat hampir 100 perintah php artisan yang tersedia. Apa yang kita bahas dalam buku ini mungkin baru mencakup 10% saja, karena mayoritas perintah termasuk materi advanced.

Setiap perintah ini dikelompokkan berdasarkan jenis. Yang akan banyak kita pakai adalah kelompok **make**, karena inilah yang dipakai untuk membuat berbagai file Laravel secara otomatis. Kali ini yang jadi fokus utama kita adalah perintah **make:controller**.

Kembali ketik perintah berikut di cmd: `php artisan make:controller -h`

```
C:\xampp\htdocs\laravel01>php artisan make:controller -h
```

Description:

Create a new controller class

Usage:

```
make:controller [options] [--] <name>
```

Arguments:

name	The name of the class
------	-----------------------

Options:

--api	Exclude the create and edit methods from the controller.
--type=TYPE	Manually specify the controller stub file to use.
--force	Create the class even if the controller already exists
-i, --invokable	Generate a single method, invokable controller class.
-m, --model[=MODEL]	Generate a resource controller for the given model.
-p, --parent[=PARENT]	Generate a nested resource controller class.
-r, --resource	Generate a resource controller class.
-R, --requests	Generate FormRequest classes for store and update.
--test	Generate an accompanying PHPUnit test for the Controller
--pest	Generate an accompanying Pest test for the Controller
-h, --help	Display help for the given command. When no command is given display help for the list command
-q, --quiet	Do not output any message
-V, --version	Display this application version
--ansi --no-ansi	Force (or disable --no-ansi) ANSI output
-n, --no-interaction	Do not ask any interactive question
--env[=ENV]	The environment the command should run under
-v vv vvv, --verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Tambahan tanda -h dipakai untuk melihat *help* atau petunjuk penggunaan perintah tersebut. Jika anda ragu mengenai penulisan sebuah perintah php artisan, cukup tambah akhiran -h untuk melihat cara pakainya.

Di bagian *Description* terdapat penjelasan singkat mengenai perintah ini, yaitu '*Create a new controller class*', yang berarti '*Buat sebuah class controller baru*'.

Cara penulisannya ada di bagian *Usage*, yakni `make:controller [options] [--] <name>`. Tanda kurung siku [] menandakan bagian opsional, yang artinya bisa tidak wajib ditulis. Yang wajib hanyalah `<name>`. Isi `<name>` ini dijelaskan pada bagian *Arguments*, yakni '*The name of the class*', atau diisi dengan nama controller yang akan dibuat.

Untuk sementara kita akan abaikan penjelasan di bagian *Options*, karena kita akan bahas di bab terpisah, terutama -m untuk membuat model, dan -r untuk membuat resource.

Dengan demikian, perintah untuk membuat controller adalah:

```
php artisan make:controller <namaController>
```

Sebagai contoh, saya ingin membuat controller `MahasiswaController`. Perintahnya adalah:

```
php artisan make:controller MahasiswaController
```

Controller

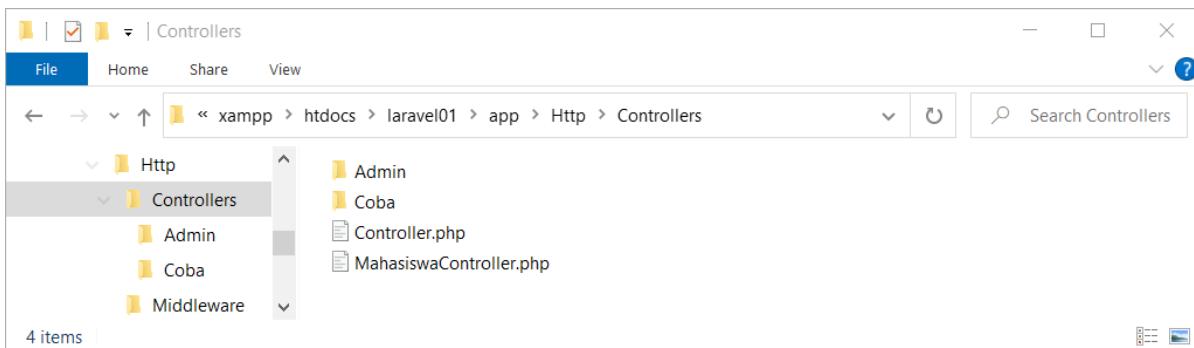
```
C:\xampp\htdocs\laravel01>php artisan make:controller MahasiswaController
Controller created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Hasil dari perintah php artisan make:controller MahasiswaController

Akan tampil teks Controller created successfully, yang artinya controller sudah berhasil dibuat.

Untuk membuktikan, silahkan buka folder app\Http\Controllers, seharusnya akan tampak file baru bernama **MahasiswaController.php**.



Gambar: File MahasiswaController.php

Mari kita lihat isi file ini:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     //
10 }
```

Inilah struktur dasar controller yang dibuat menggunakan perintah `php artisan make:controller`. Terlihat kerangka controller sudah lengkap, yang terdiri dari penulisan `namespace` di baris 3 serta pembuatan class `MahasiswaController` yang sudah langsung meng-`extends` class `Controller` di baris 7.

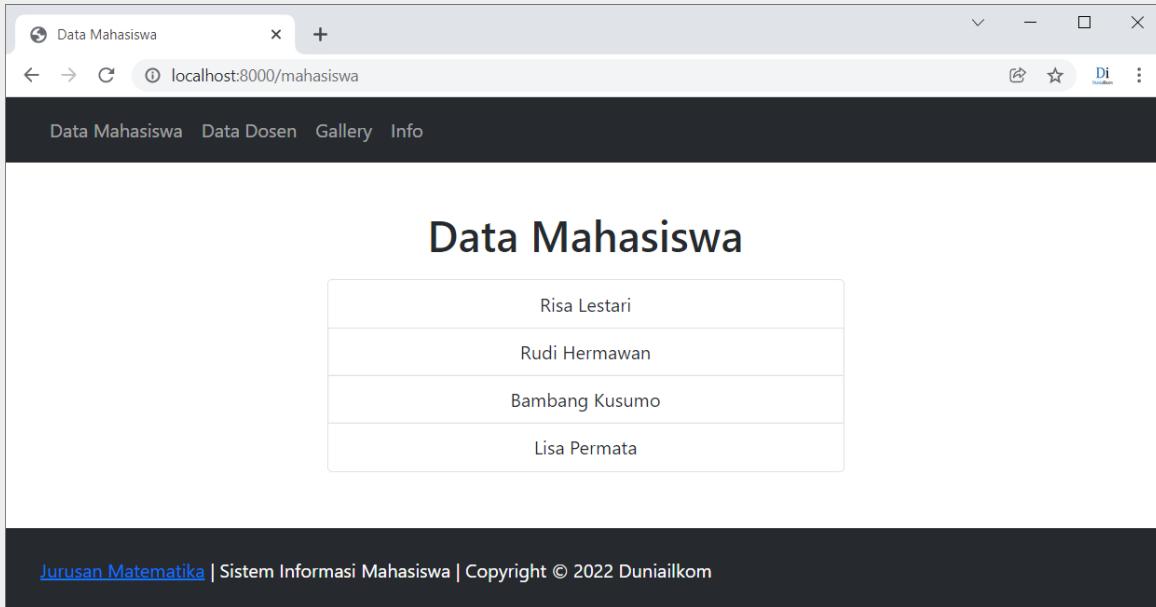
Selain itu terdapat 1 perintah tambahan, yakni `use Illuminate\Http\Request` di baris 5. Jika diperhatikan, ini adalah perintah untuk proses import class bernama **Request**. Sesuai dengan namanya, class `Request` berisi informasi seputar HTTP request yang dikirim dari web browser.

Class `Request` ini tidak wajib (opsional) dan baru terpakai jika kita ingin memproses form. Yang harus ada di sebuah controller adalah penulisan `namespace`, serta pembuatan sebuah class

controller yang meng-extends class Controller.

Exercise

Sebagai latihan tentang controller, saya ingin mengajak anda refreshing sedikit materi tentang route, view dan blade. Di akhir bab blade, kita membuat project sederhana Sistem Informasi Mahasiswa.



Gambar: Tampilan mini project Sistem Informasi Mahasiswa

Di situ saya masih menggunakan alur **route -> view**. Tantangannya, bisakah anda konversi kode program yang dipakai agar alurnya menjadi **route -> controller -> view?**

Kode untuk view tidak perlu kita utak-atik, yang jadi fokus utama hanya mengubah kode di route untuk dipecah ke dalam controller.

Berikut kode program route yang kita gunakan sebelumnya:

routes/web.php

```

1 Route::get('/mahasiswa', function () {
2     $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
3                       "Lisa Permata"];
4
5     return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
6 })->name('mahasiswa');
7
8 Route::get('dosen', function () {
9     $arrDosen = ["Maya Fitrianti, M.M.", "Prof. Silvia Nst, M.Farm.",
10                  "Dr. Umar Agustinus", "Dr. Syahrial, M.Kom."];
11
12    return view('dosen')->with('dosen', $arrDosen);
13 })->name('dosen');
14
15 Route::get('gallery', function () {

```

```

16     return view('gallery');
17 })->name('gallery');
18
19 Route::get('informasi/{fakultas}/{jurusan}', function ($fakultas, $jurusan)
20 {
21     $data = [$fakultas, $jurusan];
22     return view('informasi')->with('data', $data);
23 })->name('info');

```

Saya ingin agar logika program (termasuk pengiriman data ke view), di pindah ke dalam **MahasiswaController**. Di dalam route, cukup berisi pemanggilan method ke controller saja.

Untuk mempermudah percobaan, silahkan copy view dari file **belajar_laravel.zip** yang tersedia di Google Drive, atau anda juga bisa buat ulang view seperti di akhir bab Blade.

Answer

Berikut modifikasi file route:

routes/web.php

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/mahasiswa', [MahasiswaController::class, 'mahasiswa'])
7     ->name('mahasiswa');
8
9 Route::get('/dosen', [MahasiswaController::class, 'dosen'])
10    ->name('dosen');
11
12 Route::get('/gallery', [MahasiswaController::class, 'gallery'])
13    ->name('gallery');
14
15 Route::get('informasi/{fakultas}/{jurusan}',
16           [MahasiswaController::class, 'info'])->name('info');

```

Saya mengganti penulisan closure menjadi `[MahasiswaController::class, 'namaMethod']`. Isi closure akan di pindah ke dalam file **MahasiswaController**:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4

```

```

5  use Illuminate\Http\Request;
6
7  class MahasiswaController extends Controller
8  {
9      public function mahasiswa()
10     {
11         $arrMahasiswa = ["Risa Lestari", "Rudi Hermawan", "Bambang Kusumo",
12                         "Lisa Permata"];
13
14         return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
15     }
16
17     public function dosen()
18     {
19         $arrDosen = ["Maya Fitrianti, M.M.", "Prof. Silvia Nst, M.Farm.",
20                         "Dr. Umar Agustinus", "Dr. Syahrial, M.Kom."];
21
22         return view('dosen')->with('dosen', $arrDosen);
23     }
24
25     public function gallery()
26     {
27         return view('gallery');
28     }
29
30     public function info($fakultas, $jurusan)
31     {
32         $data = [$fakultas, $jurusan];
33         return view('informasi')->with('data', $data);
34     }
35 }

```

Tidak ada kewajiban menulis alamat URL di route agar sesuai dengan nama method di Controller. Maksudnya, URL 'mahasiswa' tidak harus berpasangan dengan method `mahasiswa()` di `MahasiswaController`. Tapi agar mudah dibaca, tidak ada salahnya dibuat sama.

Perhatikan juga penulisan route `info` yang menggunakan *route parameter* `{fakultas}`/`{jurusan}`. Keduanya ditampung ke dalam parameter dari method `info($fakultas, $jurusan)`.

Inilah salah satu keuntungan dari konsep MVC. Meskipun kita mengubah drastis kode program di sisi server (*back-end*), tidak berpengaruh apa-apa ke dalam view. Dengan catatan, data yang dikirim ke view tetap sama seperti sebelumnya.

11.10. Menggunakan Penulisan Route Laravel 7 (Opsional)

Khusus bagi yang pernah mempelajari Laravel 7 ke bawah, mungkin merasa heran kenapa penulisan route di Laravel 8 dan 9 lebih panjang. Untungnya, dengan mengubah sedikit

konfigurasi, penulisan route lama masih bisa kita pakai.

Dalam exercise sebelum ini, untuk mengakses method `mahasiswa()` yang ada di `MahasiswaController`, penulisan routenya adalah sebagai berikut:

```
1 use App\Http\Controllers\MahasiswaController;
2
3 Route::get('/mahasiswa',[MahasiswaController::class,'mahasiswa'])
4     ->name('mahasiswa');
```

Jika menggunakan cara di Laravel 7, bisa ditulis lebih ringkas tanpa namespace:

```
1 Route::get('/mahasiswa','MahasiswaController@mahasiswa')->name('mahasiswa');
```

Agar Laravel 9 mendukung penulisan seperti ini, silahkan buka file `app\Providers\RouteServiceProvider.php`, lalu hapus tanda komentar di baris 29:

Gambar: Hapus komentar di baris 29

Yakni dari sebelumnya:

```
// protected $namespace = 'App\\Http\\Controllers';
```

Menjadi:

```
protected $namespace = 'App\\Http\\Controllers';
```

Atau jika di dalam file `app\Providers\RouteServiceProvider.php` tidak ada baris komentar ini, bisa ditambahkan sendiri.

Selain itu pastikan terdapat juga perintah `->namespace($this->namespace)` di bagian `Route::prefix('api')` dan juga `Route::prefix('web')`. Gambar berikut memperlihatkan 3 bagian kode yang harus ada:

Controller

```
File Edit Selection View Go Run Terminal Help
RouteServiceProvider.php - laravel01 - Visual Studio Code
EXPLORER OPEN EDITORS LARAVEL01
> Admin PageController.php
> Coba Foo.php
> Controller.php
> MahasiswaController.php
> Middleware Kernel.php
> Models
> Providers
  AppServiceProvider.php
  AuthServiceProvider.php
  BroadcastServiceProvider.php
  EventServiceProvider.php
  RouteServiceProvider.php
> bootstrap
> config
> OUTLINE
> NPM SCRIPTS
RouteServiceProvider.php
28 | */
29 | protected $namespace = 'App\\Http\\Controllers'; 1
30 |
31 | /**
32 | * Define your route model bindings, pattern filters, etc.
33 | *
34 | * @return void
35 | */
36 | public function boot()
37 | {
38 |     $this->configureRateLimiting();
39 |
40 |     $this->routes(function () {
41 |         Route::prefix('api')
42 |             ->middleware('api')
43 |             ->namespace($this->namespace) 2
44 |             ->group(base_path('routes/api.php'));
45 |
46 |         Route::middleware('web')
47 |             ->namespace($this->namespace) 3
48 |             ->group(base_path('routes/web.php'));
49 |     });
50 | }
Ln 22, Col 8 Spaces: 4 UTF-8 LF PHP ⚙️ 🔍
```

Gambar: Cara mengaktifkan penulisan route seperti di Laravel 7

Dengan perubahan ini, maka setiap route sudah langsung ditambah namespace secara otomatis. Inilah yang sebenarnya ada di Laravel 7 tapi di-non aktifkan di Laravel 8 dan 9.

Untuk uji coba, route dari file exercise sebelumnya bisa ditulis sebagai berikut:

routes/web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/mahasiswa', 'MahasiswaController@mahasiswa')->name('mahasiswa');
6 Route::get('/dosen', 'MahasiswaController@dosen')->name('dosen');
7 Route::get('/gallery', 'MahasiswaController@gallery')->name('gallery');
8 Route::get('informasi/{fakultas}/{jurusan}', 'MahasiswaController@info')
9         ->name('info');
```

Seharusnya, halaman view tetap bisa diakses seperti sebelumnya.

Meskipun kita sudah mengubah konfigurasi `RouteServiceProvider.php`, cara penulisan route beserta namespace juga tetap bisa dipakai.

Dengan demikian kita tidak perlu mengembalikan tanda komentar ke perintah `protected $namespace = 'App\\Http\\Controllers'.`.

Dalam bab ini kita telah berkenalan dengan Controller. Terdapat 2 cara pembuatan controller di Laravel, yakni cara manual dari teks editor, atau menggunakan perintah `php artisan make:controller`.

Meskipun pada awalnya terasa sedikit repot, membuat controller dari perintah `php artisan` akan terasa lebih praktis, karena file dasar controller sudah tersedia dan bisa langsung dipakai. Terlebih nanti masih banyak perintah `php artisan` lain yang akan kita pelajari.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

12. Collection

Kali ini kita akan membahas tentang **Collection**, yakni sebuah object khusus bawaan Laravel yang nantinya sangat berhubungan dengan cara mengolah data dari dan ke database.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

12.1. Pengertian Collection

Collection adalah object khusus Laravel yang dipakai untuk menampung data. Atau pengertian yang lebih sederhana, collection adalah array dengan *superpower*.

Dalam banyak hal, *collection* sangat mirip seperti array, yakni terdiri dari kumpulan data dan bisa diproses dengan perulangan `foreach`. Namun karena merupakan sebuah object, collection memiliki berbagai method bawaan (total tersedia 116 method yang bisa kita pakai). Jika Laravel meng-generate data dalam jumlah banyak, besar kemungkinan itu disajikan dalam bentuk collection, bukan array biasa.

Sebagai contoh, ketika kita mengambil data dari MySQL menggunakan fungsi `mysqli_query()`, data tersebut sampai di PHP dalam bentuk array. Di Laravel, data yang berasal dari database akan berbentuk collection. Karena alasan inilah saya ingin membahas collection terlebih dahulu sebelum kita masuk ke pemrosesan database.

Contoh lain penggunaan collection adalah **Request** object, yakni sebuah object khusus untuk menampung nilai inputan form. Sehingga pada saat kita masuk ke bab tentang pemrosesan form, tetap akan bertemu dengan collection.

12.2. Cara Pembuatan Collection

Umumnya collection diperoleh sebagai hasil dari proses internal Laravel, misalnya ketika mengakses data dari database. Namun kali ini kita akan membuat collection secara manual.

Sebagai bahan praktek, saya butuh controller bernama `CollectionController`. Silahkan buka **cmd**, masuk ke folder laravel dan ketik perintah berikut:

```
php artisan make:controller CollectionController
```

Dengan perintah ini, Laravel akan meng-generate file bernama `CollectionController.php` di folder `app\Http\Controllers`. Inilah controller yang akan kita pakai.

Untuk route, isi dengan kode berikut:

`routes/web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\CollectionController;
5
6 Route::get('/satu', [CollectionController::class, 'collectionSatu']);
7 Route::get('/dua', [CollectionController::class, 'collectionDua']);
8 Route::get('/tiga', [CollectionController::class, 'collectionTiga']);
9 Route::get('/empat', [CollectionController::class, 'collectionEmpat']);
10 Route::get('/lima', [CollectionController::class, 'collectionLima']);
11 Route::get('/enam', [CollectionController::class, 'collectionEnam']);
```

Saya menyiapkan 6 route untuk mengakses method milik `CollectionController`. Pada saat alamat `localhost:8000/satu` diakses, itu akan menjalankan method `collectionSatu()`. Ketika diakses alamat `localhost:8000/dua`, akan menjalankan method `collectionDua()`, dst. Tidak ada alasan khusus dari pemilihan nama route ini, hanya sebagai route *dummy* untuk kita melakukan berbagai percobaan.

Dan berikut isi dari `CollectionController.php`:

`app/Http/Controllers/CollectionController.php`

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class CollectionController extends Controller
8 {
9     public function collectionSatu()
10    {
11        //... isi method collectionSatu
12    }
13
14    public function collectionDua()
15    {
16        //... isi method collectionDua
17    }
18}
```

Collection

```
19     public function collectionTiga()
20     {
21         //... isi method collectionTiga
22     }
23
24     public function collectionEmpat()
25     {
26         //... isi method collectionEmpat
27     }
28
29     public function collectionLima()
30     {
31         //... isi method collectionLima
32     }
33     public function collectionEnam()
34     {
35         //... isi method collectionEnam
36     }
37 }
```

Kita akan mulai dari method `collectionSatu()` terlebih dahulu.

Terdapat 2 cara untuk membuat collection di Laravel. Cara pertama, dengan langsung mengakses **Collection object** seperti contoh berikut:

app/Http/Controllers/CollectionController.php

```
1 public function collectionSatu()
2 {
3     $myArray = [1, 9, 3, 4, 5, 3, 5, 7];
4     $collection = new \Illuminate\Support\Collection($myArray);
5 }
```

Di baris 3 saya membuat array yang berisi kumpulan angka. Array ini kemudian ditampung ke dalam variabel `$myArray`.

Selanjutnya di baris 4, variabel `$myArray` menjadi inputan untuk class `\Illuminate\Support\Collection()`. Inilah class yang dipakai untuk membuat collection. Hasilnya, variabel `$collection` akan berisi sebuah **collection object**.

Cara kedua untuk membuat collection adalah memakai *helper function* `collect()`. Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```
1 public function collectionSatu()
2 {
3     $myArray = [1, 9, 3, 4, 5, 3, 5, 7];
4     $collection = collect($myArray);
5 }
```

Hampir sama seperti kode pertama, function `collect()` juga menerima sebuah argument yang berbentuk array. Proses pembuatan array di baris 3 ini juga tidak harus terpisah, bisa saja

ditulis langsung sebagai argument:

app/Http/Controllers/CollectionController.php

```
1 public function collectionSatu()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4 }
```

Hasilnya, variabel `$collection` juga sudah berisi sebuah **collection object**. Mari kita lihat struktur detail dari object ini:

app/Http/Controllers/CollectionController.php

```
1 public function collectionSatu()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4
5     echo "<pre>";
6     var_dump($collection);
7     echo "</pre>";
8 }
```



```
object(Illuminate\Support\Collection)#276 (2) {
  ["items":protected]=>
  array(8) {
    [0]=>
    int(1)
    [1]=>
    int(9)
    [2]=>
    int(3)
    [3]=>
    int(4)
    [4]=>
    int(5)
    [5]=>
    int(3)
    [6]=>
    int(5)
    [7]=>
    int(7)
  }
  ["escapeWhenCastingToString":protected]=>
  bool(false)
}
```

Gambar: Hasil var_dump(\$collection)

Saya menggunakan fungsi `var_dump()` bawaan PHP untuk melihat struktur variabel `$collection`. Terlihat bahwa ini adalah sebuah object yang berasal dari class `Illuminate\Support\Collection`.

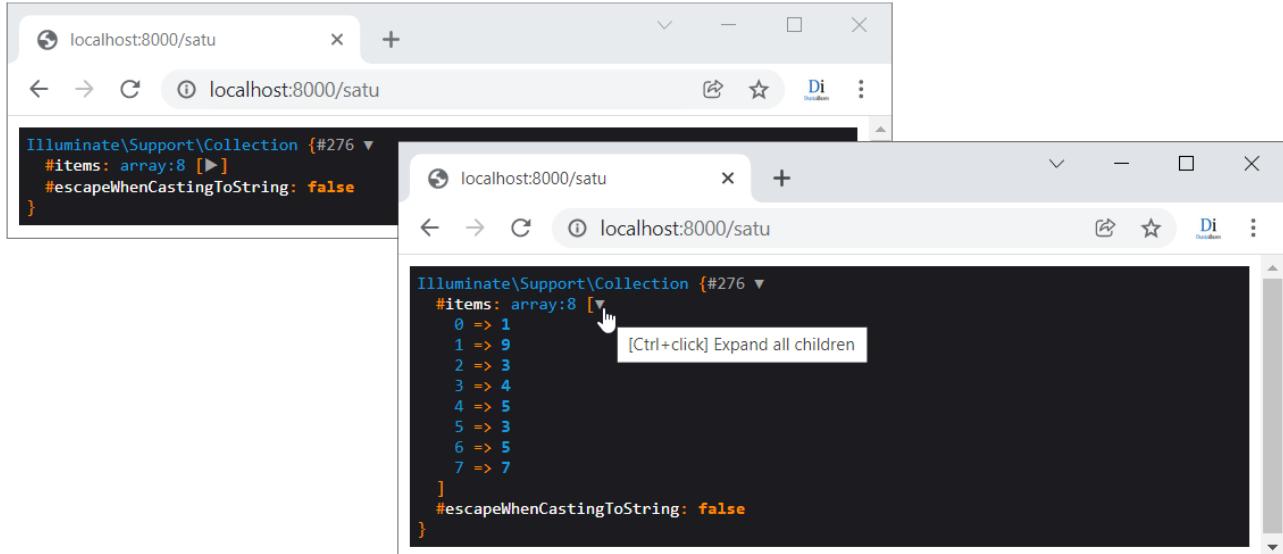
Selain itu semua element array disimpan dalam property **items** yang di-set sebagai **protected**. Karena dibatasi dengan hak akses **protected**, kita tidak bisa langsung mengakses property **items** dari luar object.

Selain menggunakan `var_dump()` untuk melihat detail variabel, Laravel juga menyediakan

fungsi `dump()`:

`app/Http/Controllers/CollectionController.php`

```
1 public function collectionSatu()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     dump($collection);
5 }
```



Gambar: Tampilan hasil fungsi `dump($collection)`

Pada dasarnya hasil yang didapat mirip seperti fungsi `var_dump()`, hanya saja tampilan dari `dump()` lebih rapi dan lebih *colorful*. Di baris bagian atas terdapat tulisan 'Collection', yang menandakan isi variabel yang sedang di-dump adalah sebuah object dari class **Collection**.

12.3. Mengakses Collection

Untuk mengakses isi atau element yang tersimpan di dalam collection, caranya sama seperti mengakses array biasa. Yakni dengan menulis nama variabel kemudian diikuti dengan key atau index dalam kurung siku. Selain itu collection juga bisa diakses menggunakan perulangan `foreach`. Berikut contohnya:

`app/Http/Controllers/CollectionController.php`

```
1 public function collectionDua()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4
5     echo $collection[0]; echo "<br>";
6     echo $collection[2]; echo "<br>";
7
8     foreach($collection as $value) {
9         echo "$value ";
```

Collection

```
10      }
11 }
```

Hasil kode program:

```
1
3
1 9 3 4 5 3 5 7
```

Di baris 5 dan 6 terdapat perintah untuk mengakses element ke-1 dan ke-3, dimana element tersebut berada di index 0 dan index 3 (sebagai pengingat, index array mulai dari 0, bukan 1). Kemudian di baris 8 – 10 saya menulis perulangan foreach untuk menampilkan semua isi `$collection`.

Yang cukup unik, collection juga bisa langsung di echo:

app/Http/Controllers/CollectionController.php

```
1 public function collectionDua()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     echo $collection;
5 }
```

Hasil kode program:

```
[1,9,3,4,5,3,5,7]
```

Di baris 4 saya men-echo `$collection`. Ini tidak masalah karena Laravel secara otomatis mengkonversi isi collection menjadi string ketika diakses seperti ini.

Collection juga bisa berisi array yang lebih kompleks, termasuk associative array:

app/Http/Controllers/CollectionController.php

```
1 public function collectionDua()
2 {
3     // Collection dari berbagai tipe data
4     $collection = collect(["belajar", "laravel", "uncover", 1, 2, 3]);
5     echo $collection;
6
7     echo "<br>";
8
9     // Collection dari associative array
10    $collection = collect([
11        "nama" => "Laura",
12        "sekolah" => "SMA 5 Lampung",
13        "kota" => "Lampung",
14        "jurusan" => "IPA",
15    ]);
16    echo $collection;
17 }
```

Hasil kode program:

```
["belajar", "laravel", "uncover", 1, 2, 3]
{"nama": "Laura", "sekolah": "SMA 5 Lampung", "kota": "Lampung", "jurusan": "IPA"}
```

Ada sedikit yang mesti kita bahas. Perhatikan string hasil perintah echo di baris 16, itu ditampilkan dalam format JSON, bukan bentuk array biasa yang ditulis dalam kurung siku.

JSON adalah singkatan dari JavaScript Object Notation, yakni format penulisan object di bahasa pemrograman JavaScript. JSON sendiri banyak dipakai sebagai sarana pertukaran data menggunakan API.

Ketika collection berisi associative array dengan key yang tidak berurutan atau key non-numeric, maka ketika di echo hasilnya akan berbentuk JSON.

Secara internal, sebenarnya numeric array seperti di baris 4 juga disimpan dalam bentuk associative array, tapi dengan index numerik dan berurutan dari key 0. Kedua kode berikut akan menghasilkan array yang sama:

```
1 public function collectionDua()
2 {
3     $varA = [1, 2, 3];
4     $varB = ["0"=>1, "1"=>2, "2"=>3];
5
6     dump($varA==$varB); // true
7 }
```

Ketika dikonversi menjadi collection dan di-echo, keduanya juga tetap tampil sebagai array:

```
1 public function collectionDua()
2 {
3     $varA = collect([1, 2, 3]);
4     $varB = collect(["0"=>1, "1"=>2, "2"=>3]);
5
6     echo $varA; // [1, 2, 3]
7     echo $varB; // [1, 2, 3]
8 }
```

Namun jika numeric key ini tidak berurutan atau dibuat non-numeric, maka ketika di echo akan menjadi JSON:

```
1 public function collectionDua()
2 {
3     $varA = collect([1, 2, 3]);
4     $varB = collect(["1"=>1, "2"=>2, "3"=>3]);
5
6     echo $varA; // [1, 2, 3]
7     echo $varB; // {"1":1, "2":2, "3":3}
8 }
```

Key dari `$varB` saya modifikasi mulai dari "1", bukan "0". Sehingga akan tampil sebagai JSON.

Perbedaan ini hanya terjadi jika collection di echo, isi dari collection itu sendiri tetap bisa diproses secara normal. Pada prakteknya nanti, kita sangat jarang langsung men-echo collection seperti ini. Biasanya collection diproses terlebih dahulu menggunakan perulangan foreach atau dikirim ke view untuk ditampilkan.

Namun pengetahuan akan perbedaan tampilan ini cukup bermanfaat supaya kita tidak bingung kenapa hasil dari collection tidak tampak seperti array.

12.4. Collection Method

Fitur paling menarik dari collection adalah banyaknya method yang tersedia. Ketika buku ini ditulis, total terdapat 116 method yang bisa kita pakai untuk collection object, dan kemungkinan akan terus bertambah di setiap versi Laravel. Kita akan bahas beberapa di antaranya.

Daftar lengkap method ini bisa anda lihat di [Laravel Collection](#). Mayoritas method bersifat *non-destructive*, yakni akan mengembalikan nilai baru dimana isi collection awal tidak berubah.

Method sum(), avg(), max(), min() dan median()

Sesuai dengan namanya, kelima method ini dipakai untuk mencari total (sum), nilai rata-rata (avg), nilai maksimum (max), nilai minimum (min), serta nilai median dari semua element yang ada di collection.

Karena method ini melihatkan perhitungan matematis, maka semua element collection juga harus bertipe angka. Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```

1  public function collectionTiga()
2  {
3      $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4
5      // Operasi Matematis
6      dump( $collection->sum() );           // 37
7      dump( $collection->avg() );          // 4.625
8      dump( $collection->max() );          // 9
9      dump( $collection->min() );          // 1
10     dump( $collection->median() );        // 4.5
11 }
```

Method random()

Method `random()` berguna untuk mengambil 1 element acak setiap kali method dijalankan.

Collection

Berikut contoh penggunaannya:

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     dump( $collection->random() );           // 5
5 }
```

Method concat()

Berfungsi untuk menambah element ke dalam collection. Hasil akhir dari method ini berupa collection baru yang berisi element asal + element yang baru saja ditambahkan:

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     echo $collection->concat([10,11,12]);
5     // [1,9,3,4,5,3,5,7,10,11,12]
6 }
```

Method contains()

Method `contains()` dipakai untuk memeriksa apakah sebuah nilai ada di dalam collection atau tidak. Hasilnya berupa tipe data boolean `true` atau `false`:

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     dump( $collection->contains(3) );           // true
5     dump( $collection->contains(8) );           // false
6 }
```

Method unique()

Method `unique()` akan men-filter isi collection dan mengembalikan nilai yang unik saja (nilai yang tidak berulang):

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     echo $collection->unique();
5     // {"0":1,"1":9,"2":3,"3":4,"4":5,"7":7}
6 }
```

Di dalam `$collection`, angka 3 dan 5 muncul lebih dari 1 kali, sehingga akan ter-filter oleh method `unique()`. Ketika di echo, hasilnya tampil dalam format JSON karena method `unique()` akan memodifikasi urutan key collection (ada element yang dihapus).

Method all()

Method `all()` berfungsi untuk mengkonversi collection menjadi array. Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     dump( $collection->all() );           // [1, 9, 3, 4, 5, 3, 5, 7]
5 }
```

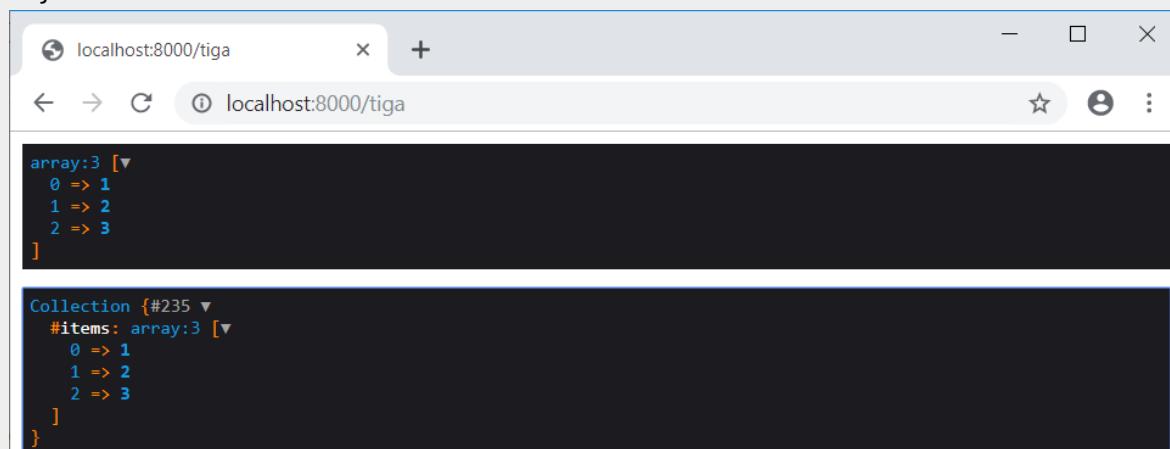
Method `all()` ini akan sering kita pakai ketika memproses data dari database.

Collection vs Array

Kedua tipe data ini serupa tapi tidak sama. Sering terjadi error karena kita menganggap sebuah kode program menghasilkan array tapi memprosesnya sebagai collection, begitu juga sebaliknya.

Fungsi `dump()` bisa dipakai untuk mendeteksi apakah sebuah variabel berisi collection atau array, seperti contoh berikut:

```
1 public function collectionTiga()
2 {
3     $varA = [1,2,3];
4     $varB = collect([1,2,3]);
5
6     dump($varA);
7     dump($varB);
8 }
```



Gambar: Hasil `dump()` collection dan array

Dari kode program sebenarnya kita bisa langsung melihat bahwa \$varA merupakan array PHP, sedangkan \$varB berisi collection. Namun nantinya isi dari \$varA dan \$varB ini bisa berasal dari sumber lain yang tidak terlihat (di-generate oleh Laravel).

Penggunaan fungsi `dump()` akan sangat membantu membedakan kedua tipe data ini. Terlebih keduanya sama-sama bisa diakses menggunakan key seperti `$varA[1]` maupun `$varB[1]`.

Method `first()` dan `last()`

Method `first()` dan `last()` dipakai untuk mengambil 1 element yang berada di urutan pertama dan urutan terakhir dari sebuah collection. Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     dump( $collection->first() );           // 1
5     dump( $collection->last() );           // 7
6 }
```

Method `count()`

Sesuai namanya, method `count()` dipakai untuk menghitung total element di dalam collection:

app/Http/Controllers/CollectionController.php

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     dump( $collection->count() );           // 8
5
6 }
```

Method `sort()`

Method `sort()` berguna untuk mengurutkan isi collection. Proses pengurutan ini tetap mempertahankan key yang sudah ada:

app/Http/Controllers/CollectionController.php

```
1 public function collectionTiga()
2 {
3     $collection = collect([1, 9, 3, 4, 5, 3, 5, 7]);
4     echo $collection->sort();
5     // {"0":1,"2":3,"5":3,"3":4,"4":5,"6":5,"7":7,"1":9}
6 }
```

Terlihat key dari collection tetap dipertahankan (tidak berubah). Hanya saja sekarang urutan

Collection

posisinya sudah tersusun secara menaik.

Method get()

Method `get()` dipakai untuk mengambil 1 element array berdasarkan key yang diinput sebagai argument:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEmpat()
2 {
3     $collection = collect([
4         "nama" => "Laura",
5         "sekolah" => "SMA 5 Lampung",
6         "kota" => "Lampung",
7         "jurusan" => "IPA",
8     ]);
9
10    // Ambil nilai berdasarkan key
11    dump( $collection->get('sekolah') );      // "SMA 5 Lampung"
12 }
```

Method ini juga bisa diisi dengan argument kedua yang berfungsi sebagai nilai default jika key tersebut tidak ditemukan:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEmpat()
2 {
3     $collection = collect([
4         "nama" => "Laura",
5         "sekolah" => "SMA 5 Lampung",
6         "kota" => "Lampung",
7         "jurusan" => "IPA",
8     ]);
9     dump( $collection->get('umur',17) );           // 17"
10 }
```

Hasilnya tampil angka 17 karena key "umur" tidak ada di dalam collection.

Method has()

Method `has()` dipakai untuk memeriksa apakah sebuah key ada di dalam collection atau tidak. Hasil dari method ini berupa boolean `true` atau `false`.

app/Http/Controllers/CollectionController.php

```
1 public function collectionEmpat()
2 {
3     $collection = collect([
4         "nama" => "Laura",
5         "sekolah" => "SMA 5 Lampung",
6         "kota" => "Lampung",
7     ]);
8 }
```

Collection

```
7     "jurusan" => "IPA",
8 );
9
10    dump( $collection->has('jurusan') );           // true
11    dump( $collection->has('umur') );             // false
12 }
```

Method replace()

Method `replace()` berguna untuk mengganti nilai yang ada di dalam collection. Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEmpat()
2 {
3     $collection = collect([
4         "nama" => "Laura",
5         "sekolah" => "SMA 5 Lampung",
6         "kota" => "Lampung",
7         "jurusan" => "IPA",
8     ]);
9
10    // Ganti isi collection
11    $hasil = $collection->replace([
12        'sekolah' => 'SMK 2 Palembang',
13        'kota' => 'Palembang'
14    ]);
15
16    dump( $hasil );
17 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#268 ▼
  #items: array:4 [▼
    "nama" => "Laura"
    "sekolah" => "SMK 2 Palembang"
    "kota" => "Palembang"
    "jurusan" => "IPA"
  ]
}
```

Di baris 11 – 14, saya menggunakan method `replace()` untuk mengganti nilai element 'sekolah' dan 'kota'. Perubahan ini disimpan ke dalam variabel `$hasil` yang selanjutnya di `dump()` pada baris 16.

Method forget()

Method `forget()` bisa dipakai untuk menghapus salah satu element yang ada di dalam collection:

Collection

app/Http/Controllers/CollectionController.php

```
1  public function collectionEmpat()
2  {
3      $collection = collect([
4          "nama" => "Laura",
5          "sekolah" => "SMA 5 Lampung",
6          "kota" => "Lampung",
7          "jurusan" => "IPA",
8      ]);
9
10     dump( $collection->forget('sekolah') );
11 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#236 ▼
  #items: array:3 [▼
    "nama" => "Laura"
    "kota" => "Lampung"
    "jurusan" => "IPA"
  ]
}
```

Di baris 10 saya mengisi string 'sekolah' ke dalam argument method `forget()`, hasilnya element ini dihapus dari dalam collection.

Method `flip()`

Method ini cukup unik, `flip()` bisa dipakai untuk membalik key element menjadi value, dan valuenya menjadi key:

app/Http/Controllers/CollectionController.php

```
1  public function collectionEmpat()
2  {
3      $collection = collect([
4          "nama" => "Laura",
5          "sekolah" => "SMA 5 Lampung",
6          "kota" => "Lampung",
7          "jurusan" => "IPA",
8      ]);
9
10     dump( $collection->flip() );
11 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#271 ▼
  #items: array:4 [▼
    "Laura" => "nama"
    "SMA 5 Lampung" => "sekolah"
    "Lampung" => "kota"
```

Collection

```
    "IPA" => "jurusan"
]
}
```

Method keys() dan values()

Method `keys()` berfungsi untuk mengambil semua nilai key dari collection. Dan method `values()` dipakai untuk mengambil semua nilai value dari collection. Kedua method ini mengembalikan sebuah collection baru:

app/Http/Controllers/CollectionController.php

```
1  public function collectionEmpat()
2  {
3      $collection = collect([
4          "nama" => "Laura",
5          "sekolah" => "SMA 5 Lampung",
6          "kota" => "Lampung",
7          "jurusan" => "IPA",
8      ]);
9
10     // Ambil semua key
11     dump( $collection->keys() );
12
13     // Ambil semua value
14     dump( $collection->values() );
15 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#271 ▼
  #items: array:4 [▼
    0 => "nama"
    1 => "sekolah"
    2 => "kota"
    3 => "jurusan"
  ]
}
Illuminate\Support\Collection {#271 ▼
  #items: array:4 [▼
    0 => "Laura"
    1 => "SMA 5 Lampung"
    2 => "Lampung"
    3 => "IPA"
  ]
}
```

Method search()

Method `search()` berfungsi untuk mencari sebuah nilai di dalam collection, kemudian mengembalikan index dari nilai tersebut. Berikut contoh penggunaannya:

Collection

app/Http/Controllers/CollectionController.php

```
1 public function collectionEmpat()
2 {
3     $collection = collect([
4         "nama" => "Laura",
5         "sekolah" => "SMA 5 Lampung",
6         "kota" => "Lampung",
7         "jurusan" => "IPA",
8     ]);
9
10    dump( $collection->search('IPA') );      // "jurusan"
11 }
```

Di baris 10 saya ingin mencari apa key yang berisi nilai 'IPA', hasilnya berupa string yang merujuk ke index yang berisi nilai tersebut, yakni 'jurusan'.

Method each()

Method each() merupakan cara penulisan lain dari perulangan `foreach`. Argument untuk method ini berbentuk sebuah *closure* atau *anonymous function*. Closure tersebut menerima 2 argument yang merujuk ke key dan value dari element collection.

Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEmpat()
2 {
3     $collection = collect([
4         "nama" => "Laura",
5         "sekolah" => "SMA 5 Lampung",
6         "kota" => "Lampung",
7         "jurusan" => "IPA",
8     ]);
9
10    $collection->each(function ($val, $key) {
11        echo "$key: $val <br>";
12    });
13 }
```

Hasil kode program:

```
nama: Laura
sekolah: SMA 5 Lampung
kota: Lampung
jurusan: IPA
```

Dalam contoh ini saya mengisi argument closure dengan variabel `$val` dan `$key`. Perhatikan urutannya adalah value dulu, baru kemudian key element.

Jika menggunakan perulangan `foreach`, hasil yang sama juga bisa dilakukan dengan kode

berikut:

```
1  foreach($collection as $key => $val) {
2      echo "$key = $val <br>";
3 }
```

12.5. Nested Array Collection Method

Beberapa method bawaan collection ada yang lebih cocok dipakai untuk element yang lebih kompleks, seperti nested array. Dan nantinya kita memang akan lebih banyak bertemu collection seperti ini.

Sebagai bahan praktek, berikut isi collection yang akan kita pakai:

app/Http/Controllers/CollectionController.php

```
1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      dump( $collection );
10 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#237 ▼
  #items: array:3 [▼
    0 => array:2 [▼
      "namaProduk" => "Laptop A"
      "harga" => 59990000
    ]
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
      "harga" => 350000
    ]
  ]
}
```

Saya mengisi variabel `$collection` yang berasal dari nested array, yakni array dengan 3 element, dimana setiap element-nya juga terdiri dari associative array. Struktur seperti ini sebagai simulasi data yang berasal dari tabel MySQL.

Method sortBy() dan sortByDesc()

Sebelumnya kita telah membahas tentang method `sort()`. Method `sortBy()` dan `sortByDesc()` merupakan variasi dari method `sort()`. Di sini kita bisa menentukan nama key yang menjadi penentu urutan:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      // Urutkan berdasarkan key harga
10     dump( $collection->sortBy('harga') );
11
12     // Urutkan berdasarkan key harga
13     dump( $collection->sortByDesc('harga') );
14 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#258 ▼
  #items: array:3 [▼
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
      "harga" => 350000
    ]
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
    0 => array:2 [▼
      "namaProduk" => "Laptop A"
      "harga" => 59990000
    ]
  ]
}
```

```
Illuminate\Support\Collection {#263 ▼
  #items: array:3 [▼
    0 => array:2 [▼
      "namaProduk" => "Laptop A"
      "harga" => 59990000
    ]
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
    ]
}
```

Collection

```
        "harga" => 350000
    ]
}
}
```

Dengan menulis `$collection->sortBy('harga')`, maka isi collection akan di urutkan dari harga terendah ke tertinggi. Sedangkan untuk `$collection->sortByDesc('harga')` akan mengurutkan collection dari harga tertinggi ke harga terendah.

Hasil dari method `sortBy()` ini berbentuk collection, sehingga bisa di chaining dengan method lain seperti contoh berikut:

app/Http/Controllers/CollectionController.php

```
1 public function collectionLima()
2 {
3     $collection = collect([
4         ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5         ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6         ['namaProduk' => 'Speaker C', 'harga' => 350000],
7     ]);
8
9     // Urutkan berdasarkan key harga dan tampilkan sebagai array
10    dump( $collection->sortBy('harga')->all() );
11
12    // Urutkan berdasarkan key harga dan tampilkan menggunakan method each()
13    $collection->sortBy('harga')->each(function ($val, $key) {
14        echo $val['namaProduk']."<br>";
15    });
16 }
```

Hasil kode program:

```
array:3 [▼
  2 => array:2 [▼
    "namaProduk" => "Speaker C"
    "harga" => 350000
  ]
  1 => array:2 [▼
    "namaProduk" => "Smartphone B"
    "harga" => 1599000
  ]
  0 => array:2 [▼
    "namaProduk" => "Laptop A"
    "harga" => 59990000
  ]
]
```

Speaker C
Smartphone B
Laptop A

Di baris 10, method `sortBy()` langsung disambung dengan method `all()`, hasilnya berupa

array yang sudah diurutkan dari harga terendah ke harga tertinggi.

Di baris 13 – 15, saya menjalankan method `each()` dari hasil method `sortBy()` yang dipakai untuk menampilkan daftar nama barang dari harga terendah ke tertinggi.

Teknik *chaining* ini akan sering kita pakai ketika mengolah data collection karena lebih praktis dan lebih singkat. Namun memang perlu sedikit waktu untuk mempelajari apa yang sebenarnya dilakukan kode tersebut.

Method filter()

Method `filter()` mirip seperti `each()`, yakni melakukan perulangan ke setiap element collection. Method ini juga butuh argument berupa closure yang berisi kode untuk menentukan syarat yang harus dipenuhi. Jika hasilnya `true`, maka element tersebut akan diinput sebagai collection baru. Jika hasilnya `false`, element tersebut tidak akan diambil.

Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```

1 public function collectionLima()
2 {
3     $collection = collect([
4         ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5         ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6         ['namaProduk' => 'Speaker C', 'harga' => 350000],
7     ]);
8
9     $hasil = $collection->filter(function ($val, $key) {
10        return $val['harga'] < 2000000;
11    });
12    dump( $hasil );
13 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#271 ▼
  #items: array:2 [▼
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
      "harga" => 350000
    ]
  ]
}
```

Dalam closure di baris 10, saya membuat syarat `$val['harga'] < 2000000`. Artinya, method `filter()` hanya akan menerima element dengan harga kurang dari 2.000.000. Dengan syarat

ini, variabel `$hasil` berisi 2 element saja, karena produk 'Laptop A' memiliki harga di atas 2jt.

Method where() dan firstWhere()

Kedua method ini dipakai untuk mencari element collection dengan syarat tertentu. Hasil dari method `where()` dan `firstWhere()` berupa sebuah collection baru yang memenuhi syarat tersebut.

Cara penggunaan method ini mirip seperti query `WHERE` di dalam bahasa SQL:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      // Cari element yang key harga bernilai 350000
10     dump( $collection->where('harga', 350000) );
11
12     // Tampilkan nama produk yang element key harga lebih dari 1000000
13     dump( $collection->where('harga','>=', 1000000) );
14
15 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#270 ▼
  #items: array:1 [▼
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
      "harga" => 350000
    ]
  ]
}
```

```
Illuminate\Support\Collection {#277 ▼
  #items: array:2 [▼
    0 => array:2 [▼
      "namaProduk" => "Laptop A"
      "harga" => 59990000
    ]
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
  ]
}
```

Method `where()` bisa dipanggil dengan 2 atau 3 argument. Jika dijalankan dengan 2 argument,

maka argument pertama berisi key, dan argument kedua berisi nilai yang akan dicari. Di baris 19 saya menjalankan `$collection->where('harga', 350000)`, yang dipakai untuk mencari element collection dengan harga 350.000. Hasilnya berupa produk "Speaker C".

Jika method `where()` dipanggil dengan 3 argument, maka argument kedua berfungsi menjadi tempat menulis operasi perbandingan yang ingin dipakai. Sebagai contoh di baris 13 terdapat method `$collection->where('harga', '>=', 1000000)`. Perintah ini akan mencari element collection yang memiliki harga lebih atau sama dengan 1.000.000. Hasilnya berupa produk "Laptop A" dan "Smartphone B".

Perhatikan bahwa hasil dari method `where()` adalah sebuah collection, meskipun isinya hanya 1 element saja. Jika kita yakin hasil pencarian hanya akan berisi 1 element, maka bisa disambung dengan method `first()` supaya berubah menjadi array biasa:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      $hasil = $collection->where('harga', 350000)->first();
10     echo $hasil['namaProduk']."<br>";           // Speaker C
11 }
```

Teknik seperti ini akan sering kita pakai ketika mengambil data dari database. Karena itu juga Laravel menyediakan method `firstWhere()`, yang merupakan gabungan dari method `where()` dan `first()` dalam sekali panggil:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      $hasil = $collection->firstWhere('harga', 350000);
10     echo $hasil['namaProduk']."<br>";           // Speaker C
11 }
```

Jika hasil dari method `where()` lebih dari 1 element, bisa juga digabung dengan method `all()` agar hasilnya di konversi menjadi array:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      $hasil = $collection->where('harga', '>=', 1000000)->all();
10
11     echo "<hr>";
12
13     foreach($hasil as $val) {
14         echo $val['namaProduk']."<br>";
15     }
16 }
```

Hasil kode program:

```
Laptop A
Smartphone B
```

Di baris 9, perintah `$collection->where('harga', '>=', 1000000)->all()` dipakai untuk mencari element collection yang memiliki harga di atas 1.000.000, lalu hasilnya di konversi menjadi array dengan method `all()`.

Selanjutnya di baris 13 - 15 saya menggunakan perulangan `foreach` untuk menampilkan nama produk yang memiliki harga di atas 1.000.000. Kembali, teknik seperti ini juga akan banyak kita pakai ketika memproses data dari database.

Method `whereBetween()` dan `whereNotBetween()`

Method `whereBetween()` merupakan variasi dari method `where()` yang bisa dipakai untuk mencari element collection dalam sebuah jangkauan (range), misalnya cari barang dengan harga antara 100.000 sampai 2.000.000. Sedangkan method `whereNotBetween()` dipakai untuk mencari element selain yang ada di dalam jangkauan.

Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      // Cari element dengan harga antara 100000 - 2000000
```

Collection

```
10     dump( $collection->whereBetween('harga', [100000, 2000000]) );
11
12 // Cari element dengan harga bukan di antara 100000 - 2000000
13 dump( $collection->whereNotBetween('harga', [100000, 2000000]) );
14 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#277 ▼
  #items: array:2 [▼
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
      "harga" => 350000
    ]
  ]
}

Illuminate\Support\Collection {#282 ▼
  #items: array:1 [▼
    0 => array:2 [▼
      "namaProduk" => "Laptop A"
      "harga" => 59990000
    ]
  ]
}
```

Kedua method butuh 2 buah argument. Argument pertama berupa key yang akan dicari, dan argument kedua berupa array yang berisi jangkauan.

Method `$collection->whereBetween('harga', [100000, 2000000])` dipakai untuk mencari semua element collection dengan harga antara 100.000 – 2.000.000. Sedangkan method `$collection->whereNotBetween('harga', [100000, 2000000])` berfungsi untuk mencari element collection yang memiliki harga di luar range 100.000 – 2.000.000.

Method `whereIn()` dan `whereNotIn()`

Method `whereIn()` mirip seperti `whereBetween()`, bedanya pencarian key bukan dalam bentuk jangkauan, tapi dari beberapa pilihan nilai yang ditulis sebagai array di argument kedua.

Sedangkan method `whereNotIn()` dipakai untuk mencari element di luar nilai tersebut.

Berikut contoh penggunaannya:

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionLima()
2 {
3     $collection = collect([
4         'namaProduk' => 'Laptop A', 'harga' => 59990000,
```

Collection

```
5      ['namaProduk' => 'Smartphone B', 'harga' => 1599000],
6      ['namaProduk' => 'Speaker C', 'harga' => 350000],
7  ]);
8
9 // Cari element dengan harga 1599000, 2999000 atau 3999000
10 dump( $collection->whereIn('harga', [1599000, 2999000, 3999000]) );
11
12 // Cari element dengan harga selain 1599000, 2999000, 3999000
13 dump( $collection->whereNotIn('harga', [1599000, 2999000, 3999000]) );
14 }
```

Hasil kode program:

```
Illuminate\Support\Collection {#280 ▼
  #items: array:1 [▼
    1 => array:2 [▼
      "namaProduk" => "Smartphone B"
      "harga" => 1599000
    ]
  ]
}
```

```
Illuminate\Support\Collection {#289 ▼
  #items: array:2 [▼
    0 => array:2 [▼
      "namaProduk" => "Laptop A"
      "harga" => 59990000
    ]
    2 => array:2 [▼
      "namaProduk" => "Speaker C"
      "harga" => 350000
    ]
  ]
}
```

Method `$collection->whereIn('harga', [1599000, 2999000, 3999000])` dipakai untuk mencari semua element collection dengan harga 1.599.000, 2.999.000, atau 3.999.000. Nilainya harus sama persis dengan salah satu harga ini.

Sedangkan method `$collection->whereNotIn('harga', [1599000, 2999000, 3999000])` berfungsi untuk mencari element collection dengan harga di luar 1.599.000, 2.999.000, atau 3.999.000.

Method pluck()

Method `pluck()` mirip seperti method `values()`, yakni mengambil seluruh nilai dari sebuah key. Bedanya, method `pluck()` bisa menerima argument untuk menentukan key yang akan diambil.

Berikut contoh penggunaannya:

app/Http/Controllers/CollectionController.php

```

1  public function collectionLima()
2  {
3      $collection = collect([
4          ['namaProduk' => 'Laptop A', 'harga' => 59990000],
5          ['namaProduk' => 'Smartphone B', 'harga' => 15990000],
6          ['namaProduk' => 'Speaker C', 'harga' => 350000],
7      ]);
8
9      // Ambil namaProduk dari semua element
10     dump( $collection->pluck('namaProduk') );
11 }
```

Hasil kode program:

```

Illuminate\Support\Collection {#289 ▼
  #items: array:3 [▼
    0 => "Laptop A"
    1 => "Smartphone B"
    2 => "Speaker C"
  ]
}
```

Method `$collection->pluck('namaProduk')` saya gunakan untuk mengambil nilai 'namaProduk' dari semua element yang ada di dalam collection. Hasilnya berupa collection baru dengan key numeric.

Method `pluck()` ini juga cukup sering dipakai pada saat mengakses database, dimana kita bisa mengambil semua nilai untuk 1 kolom tabel.

12.6. Object Array Collection Method

Materi kali ini membahas collection dalam bentuk yang lebih kompleks lagi, yakni array dari object. Struktur tersebut sangat mirip dengan hasil yang kita terima dari sebuah **Model** dalam Laravel (yang akan kita pelajari dalam bab berikutnya).

Berikut contoh struktur yang dimaksud:

app/Http/Controllers/CollectionController.php

```

1  public function collectionEnam()
2  {
3      $siswa00 = new \stdClass();
4      $siswa00->nama = "Rian";
5      $siswa00->sekolah = "SMK Pelita Ibu";
6      $siswa00->jurusan = "IPA";
7
8      $siswa01 = new \stdClass();
9      $siswa01->nama = "Nova";
10     $siswa01->sekolah = "SMA 2 Kota Baru";
11     $siswa01->jurusan = "IPA";
```

Collection

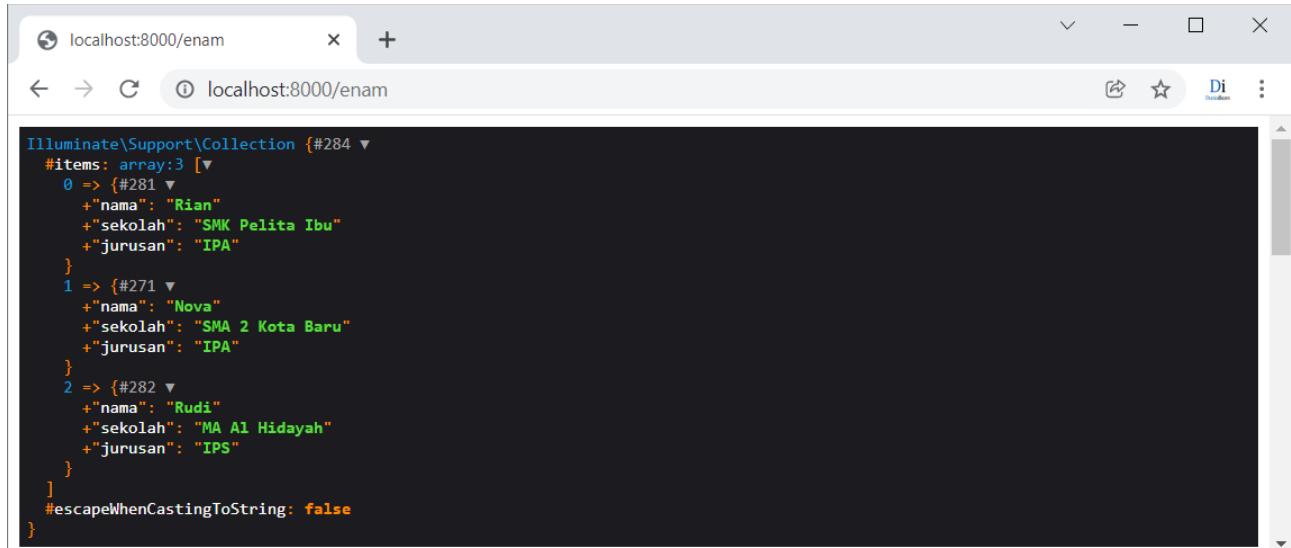
```
12 $siswa02 = new \stdClass();
13 $siswa02->nama = "Rudi";
14 $siswa02->sekolah = "MA Al Hidayah";
15 $siswa02->jurusan = "IPS";
16
17
18 $siswas = collect([
19     0 => $siswa00,
20     1 => $siswa01,
21     2 => $siswa02,
22 ]);
23
24 dump($siswas);
25 }
```

Dari baris 3 – 16 saya membuat 3 object dari class `stdClass()`. `stdClass` merupakan class 'generic' atau class 'kosong' yang tersedia di PHP. Setiap object di-set dengan 3 property yakni `nama`, `sekolah` dan `jurusan`, untuk selanjutnya ditampung ke dalam variabel `$siswa00`, `$siswa01` dan `$siswa02`.

Ketiga object menjadi argument dari function `collect()` di baris 18, yang disusun dalam bentuk associative array. Dengan demikian, variabel `$siswas` akan berisi collection dari array, dimana setiap element array berisi sebuah object.

Pemilihan nama variabel '`$siswas`' merupakan bentuk jamak dari `siswa`. Teknik penamaan ini sesuai dengan yang disarankan Laravel. Jika kita melihat nama variabel dengan tambahan 's' di akhir, bisa disimpulkan bahwa itu terdiri dari banyak data (bisa berbentuk array maupun collection).

Terakhir di baris 24 saya men-dump variabel `$siswas`:



```
Illuminate\Support\Collection {#284 ▶
  #items: array:3 [▼
    0 => {#281 ▼
      +"nama": "Rian"
      +"sekolah": "SMK Pelita Ibu"
      +"jurusan": "IPA"
    }
    1 => {#271 ▼
      +"nama": "Nova"
      +"sekolah": "SMA 2 Kota Baru"
      +"jurusan": "IPA"
    }
    2 => {#282 ▼
      +"nama": "Rudi"
      +"sekolah": "MA Al Hidayah"
      +"jurusan": "IPS"
    }
  ]
  #escapeWhenCastingToString: false
}
```

Gambar: Hasil dump(\$siswas)

Dengan struktur seperti ini, berikut cara pengaksesan nilai collection:

Collection

app/Http/Controllers/CollectionController.php

```
1 public function collectionEnam()
2 {
3     //...
4     //...
5
6     // Cara mengakses nilai collection
7     echo $siswas[1]->nama;
8     echo "<br>";
9     echo $siswas[2]->sekolah;
10
11    echo "<hr>";
12
13    // Perulangan foreach untuk menampilkan data
14    foreach($siswas as $siswa) {
15        echo $siswa->nama; echo "<br>";
16    }
17 }
```

Hasil kode program:

```
Rian
MA Al Hidayah
-----
Rian
Nova
Rudi
```

Yang perlu diperhatikan adalah bagaimana struktur collection ini disusun. Karena setiap element merupakan object, maka untuk mengakses nilainya sekarang menggunakan tanda panah '->', yang biasa kita gunakan untuk mengakses property (dan juga method) dari sebuah object.

Sebagai contoh, perintah `$siswas[0]->nama` dipakai untuk mengakses property `nama` dari element pertama yang ada di dalam collection `$siswas`. Begitu juga dengan perintah `$siswas[2]->sekolah` yang dipakai untuk mengakses property `sekolah` dari element ketiga yang terdapat di dalam collection `$siswas`.

Untuk perulangan `foreach` di baris 14 – 16, saya menulisnya sebagai `foreach($siswas as $siswa)`. Logikanya, variabel `$siswas` terdiri dari banyak `$siswa`. Jadi penamaan `siswas` dan `siswa` menjadi lebih pas, meskipun kita harus teliti untuk membedakan antara `$siswas` dengan `$siswa`.

Semua method collection yang sudah kita pelajari juga bisa dipakai ke dalam struktur ini, misalnya saya ingin mencari nama sekolah untuk siswa bernama Nova:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEnam()
2 {
```

Collection

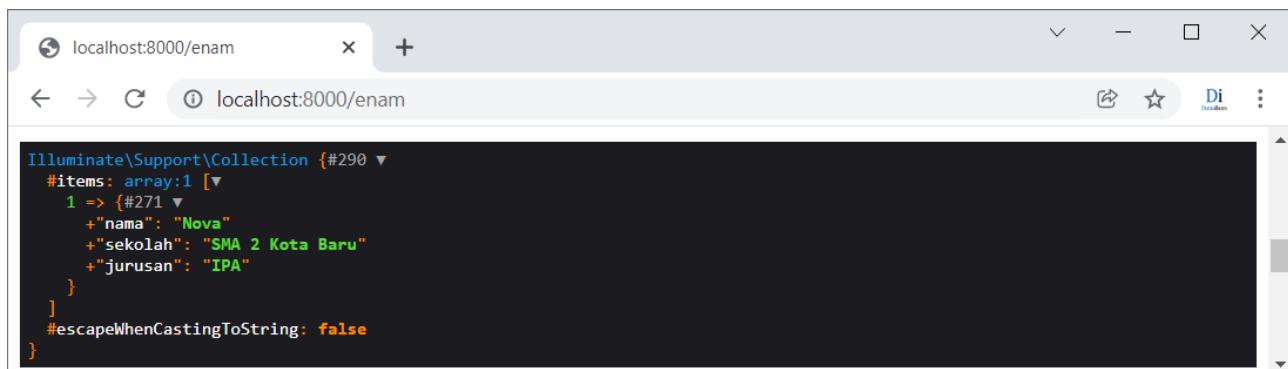
```
3 //...
4 //...
5
6 // Tampilkan nama sekolah dari siswa bernama Nova
7 $siswa = $siswas->where('nama','Nova')->first();
8 echo $siswa->sekolah; // SMA 2 Kota Baru
9 }
```

Di baris 7 setelah pemanggilan method `where()`, langsung di chaining dengan method `first()` agar hasilnya berbentuk 1 object dari `$siswa`.

Pertanyaannya, kenapa harus pakai `first()`? Karena jika hanya menggunakan `where()` saja, hasil akhir yang didapat masih berbentuk array dari object.

app/Http/Controllers/CollectionController.php

```
1 public function collectionEnam()
2 {
3     //...
4     //...
5
6     $siswa = $siswas->where('nama','Nova');
7     // echo $siswa->sekolah; // Ini akan error
8     dump($siswa);
9 }
```



Gambar: Hasil `dump()` dari method `$siswas->where('nama','Nova')`

Jika hanya menggunakan `where()`, maka untuk menampilkan nama sekolah harus menulis nama key dari array tersebut, yakni `echo $siswa[1]->sekolah`. Jika tidak melihat hasil `dump()`, kita juga tidak tau bahwa key yang diperlukan adalah 1.

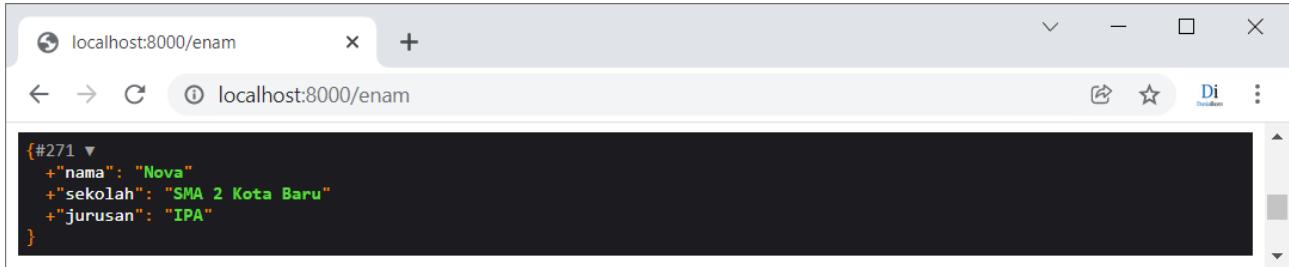
Dengan tambahan `first()`, hasil akhir sudah berbentuk 1 object saja:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEnam()
2 {
3     //...
4     //...
5     $siswa = $siswas->where('nama','Nova')->first();
6     dump($siswa);
```

Collection

```
7 }
```



Gambar: Hasil dump() dari method \$siswas->where('nama','Nova')->first();

Konsep seperti ini sangat penting dipahami karena akan banyak kita pakai dalam pemrosesan database nanti.

Method get() yang pernah kita bahas juga bisa dipakai dalam struktur ini. Berikut contoh penggunaannya:

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionEnam()
2 {
3     //...
4     //...
5     $siswa = $siswas->get(2);
6     echo "$siswa->nama, $siswa->sekolah, $siswa->jurusan";
7     // Rudi, MA Al Hidayah, IPS
8 }
```

Di baris 5, method \$siswas->get(2) dipakai untuk mengambil 1 element collection yang memiliki key 2. Karena hasilnya sudah berbentuk 1 object, bisa langsung di akses dengan \$siswa->nama, \$siswa->sekolah, dan \$siswa->jurusan.

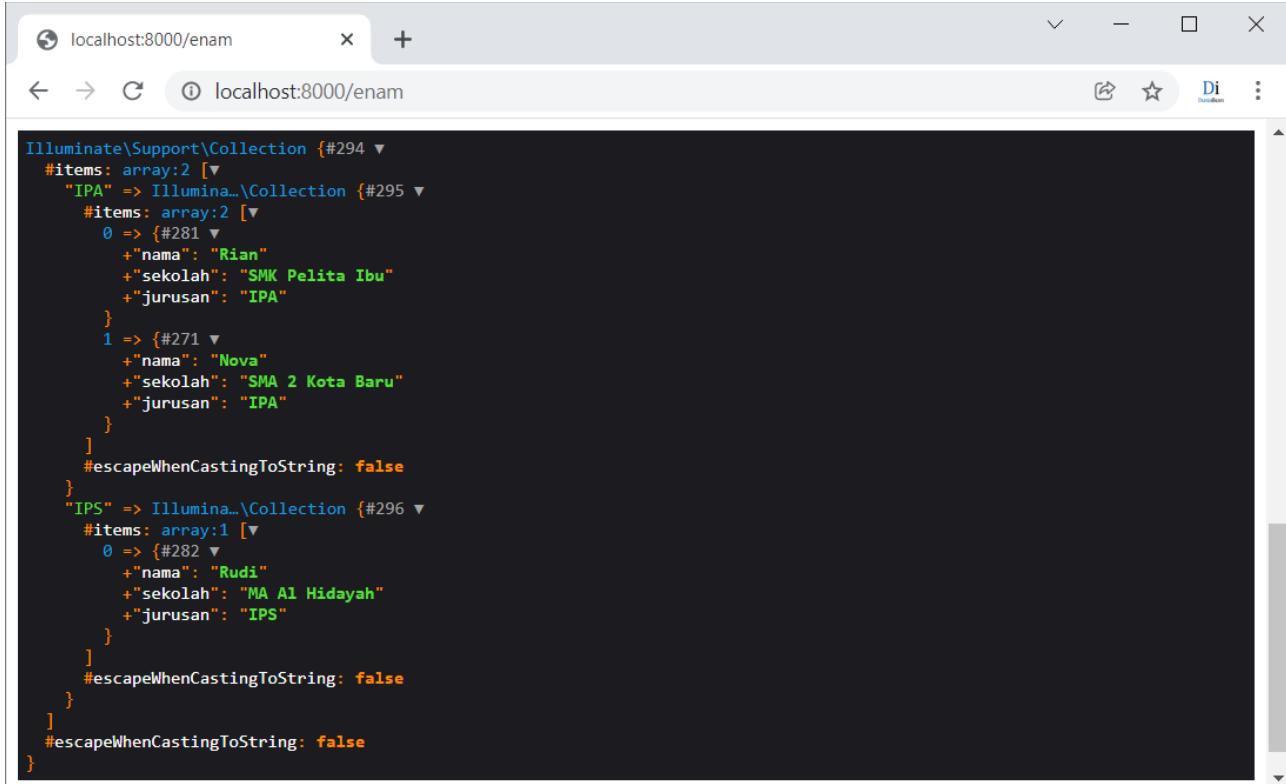
Method groupBy()

Method groupBy() mirip seperti query GROUP BY di dalam bahasa SQL, yakni dipakai untuk mengelompokkan array berdasarkan key yang diinput sebagai argument. Berikut contoh penggunaannya:

```
app/Http/Controllers/CollectionController.php
```

```
1 public function collectionEnam()
2 {
3     //...
4     //...
5     $hasil = $siswas->groupBy('jurusan');
6     dump($hasil);
7 }
```

Collection



The screenshot shows a browser window with the URL `localhost:8000/enam`. The page content displays the output of a PHP `dump()` function on a collection object. The output is a nested array structure representing the grouped data:

```
Illuminate\Support\Collection {#294 ▶
  #items: array:2 [▼
    "IPA" => Illuminate\Support\Collection {#295 ▶
      #items: array:2 [▼
        0 => {#281 ▶
          +"nama": "Rian"
          +"sekolah": "SMK Pelita Ibu"
          +"jurusan": "IPA"
        }
        1 => {#271 ▶
          +"nama": "Nova"
          +"sekolah": "SMA 2 Kota Baru"
          +"jurusan": "IPA"
        }
      ]
      #escapeWhenCastingToString: false
    }
    "IPS" => Illuminate\Support\Collection {#296 ▶
      #items: array:1 [▼
        0 => {#282 ▶
          +"nama": "Rudi"
          +"sekolah": "MA Al Hidayah"
          +"jurusan": "IPS"
        }
      ]
      #escapeWhenCastingToString: false
    }
  ]
  #escapeWhenCastingToString: false
}
```

Gambar: Hasil perintah `dump($siswas->groupBy('jurusan'))`

Meskipun kita hanya menjalankan 1 perintah sederhana, yakni `$siswas->groupBy('jurusan')`, namun struktur collection yang dihasilkan menjadi lebih kompleks lagi.

Variabel `$hasil` sekarang berisi 2 array dengan key "IPA" dan "IPS", inilah 2 kelompok yang dihasilkan dari method `groupBy('jurusan')`. Jika nantinya ada siswa dengan jurusan lain, itu akan ada di kelompok key berikutnya. Di dalam setiap key, ada lagi array dengan key numeric sesuai urutan element.

Menggunakan hasil yang didapat dari method `$siswas->groupBy('jurusan')` ini, bisakah anda menampilkan semua nama siswa yang mengambil jurusan IPA?

Berikut salah satu kode yang bisa dipakai:

app/Http/Controllers/CollectionController.php

```
1 public function collectionEnam()
2 {
3     //...
4     //...
5     $namaJurusanIpa = $siswas->groupBy('jurusan')->get('IPA')
6             ->pluck('nama')->all();
7     echo 'Nama siswa jurusan IPA: '.implode(', ', $namaJurusanIpa);
8 }
```

Di baris 5 saya menjalankan 4 method yang saling di chaining satu sama lain, yakni `$siswas->groupBy('jurusan')->get('IPA')->pluck('nama')->all()`.

Perintah ini bisa dibaca: "Kelompokkan collection `$siswas` berdasarkan jurusan. Kemudian ambil jurusan 'IPA'. Lalu ambil isi property `nama` dari semua siswa tersebut. Terakhir, kumpulan nama yang didapat dikonversi dari collection menjadi array".

Pada saat akan ditampilkan, saya memanggil method `implode()` bawaan PHP untuk mengkonversi kembali array `$namaJurusanIpa` menjadi string. Hasilnya berupa teks berikut:

`Nama siswa jurusan IPA: Rian, Nova`

Sebagai penutup bab collection, kita akhiri dengan sebuah exercise.

#Exercise

Silahkan pelajari sejenak isi collection berikut:

`app/Http/Controllers/CollectionController.php`

```

1  public function exercise()
2  {
3      $matkul00 = new \stdClass();
4      $matkul00->namaMatkul = "Sistem Operasi";
5      $matkul00->jumlahSks = 3;
6      $matkul00->semester = 3;
7
8      $matkul01 = new \stdClass();
9      $matkul01->namaMatkul = "Algoritma dan Pemrograman";
10     $matkul01->jumlahSks = 4;
11     $matkul01->semester = 1;
12
13     $matkul02 = new \stdClass();
14     $matkul02->namaMatkul = "Kalkulus Dasar";
15     $matkul02->jumlahSks = 2;
16     $matkul02->semester = 1;
17
18     $matkul03 = new \stdClass();
19     $matkul03->namaMatkul = "Basis Data";
20     $matkul03->jumlahSks = 2;
21     $matkul03->semester = 3;
22
23     $matkuls = collect([$matkul00, $matkul01, $matkul02, $matkul03]);
24 }
```

Rancang pemanggilan collection method untuk:

1. Tampilkan semua mata kuliah di semester 3. Hasil akhir yang diinginkan adalah:

`Nama mata kuliah di semester 3: Sistem Operasi, Basis Data`

2. Urutkan mata kuliah berdasarkan jumlah sks (dari sks terbesar ke sks terkecil), lalu tampilkan sebagai string. Hasil akhir yang diinginkan adalah:

`Nama mata kuliah: Algoritma dan Pemrograman (4), Sistem Operasi (3), Kalkulus`

Dasar (2), Basis Data (2)**Tips:**

- * Silahkan buat route untuk method `exercise()` ini, bisa menggunakan Collection Controller atau boleh juga dari controller baru.
- * Sebelum mulai menulis kode program, sebaiknya jalankan perintah `dump($matkuls)` terlebih dahulu dan pelajari struktur collection tersebut.
- * Untuk soal nomor 1, bisa memakai teknik yang sama seperti menampilkan nama siswa jurusan IPA sebelumnya. Cara lain, bisa juga menggunakan method `where()` + perulangan `foreach`.
- * Untuk soal nomor 2, bisa menggunakan method `sortByDesc()`.

#Answer

1. Cara pertama bisa dibuat dengan gabungan method `groupBy()`, `get()`, `pluck()` dan `all()` seperti yang kita coba untuk data `$siswas`:

```

1 public function exercise()
2 {
3     //...
4     //...
5     $matkulsSem3 = $matkuls->groupBy('semester')->get(3)
6         ->pluck('namaMatkul')->all();
7     echo 'Nama mata kuliah di semester 3: '.implode(', ', $matkulsSem3);
8 }
```

Alternatif lain, bisa juga menggunakan method `where()` yang digabung dengan perulangan `foreach`:

```

1 public function exercise()
2 {
3     //...
4     //...
5     $matkulsSem3 = $matkuls->where('semester', 3);
6     $stringMatkul = "";
7     foreach($matkulsSem3 as $matkul) {
8         $stringMatkul .= $matkul->namaMatkul.", ";
9     }
10    echo 'Nama mata kuliah di semester 3: '.substr($stringMatkul, 0, -2);
11 }
```

Hasil dari method `$matkuls->where('semester', 3)` berbentuk collection yang terdiri dari array semua object mata kuliah di semester 3. Untuk mengambil nama mata kuliah saja, saya menggunakan perulangan `foreach`. Dalam setiap perulangan, nama mata kuliah

akan disambung menjadi sebuah string.

Pada saat akan ditampilkan, fungsi `substr()` dipakai untuk menghapus tambahan koma di akhir string.

2. Untuk mengurutkan mata kuliah dari tertinggi ke terendah, bisa memakai method `$matkuls->sortByDesc('jumlahSks')`. Kemudian untuk menampilkannya menjadi string, bisa pakai perulangan `foreach`:

```

1 public function exercise()
2 {
3     //...
4     //...
5     $matkulsSort = $matkuls->sortByDesc('jumlahSks');
6     $stringMatkul = "";
7     foreach($matkulsSort as $matkul) {
8         $stringMatkul .= "$matkul->namaMatkul ($matkul->jumlahSks), ";
9     }
10    echo 'Nama mata kuliah: '.substr($stringMatkul, 0, -2);
11 }
```

Dalam setiap perulangan, akses nama mata kuliah dan jumlah sks dengan perintah `$matkul->namaMatkul` dan `$matkul->jumlahSks`. Kemudian sama seperti soal no 1, ketika akan ditampilkan kita butuh method `substr()` untuk menghapus tambahan koma di dalam `$stringMatkul` hasil perulangan `foreach`.

Dalam bab ini kita telah membahas collection serta berbagai method yang tersedia. Collection menjadi salah satu object terpenting di Laravel, sehingga tidak heran ada begitu banyak method yang bisa kita pakai.

Berikutnya kita akan masuk ke materi database, yang dimulai dengan **Migration**.

13. Migration

Pengaksesan database menjadi salah satu fitur terpenting dari setiap *web application*. Dan konsep MVC sendiri sudah mengakomodasi komponen bernama **Model** untuk memprosesnya. Laravel menyediakan cukup banyak fitur pengelolaan database seperti *migration*, *DB facade*, *query builder*, serta *eloquent*. Materi inilah yang membentuk Model di Laravel. Dalam bab ini kita akan bahas tentang **migration** terlebih dahulu. Namun sebelum itu mari lihat sejenak terkait pengaturan serta konfigurasi database.

13.1. Pengaturan Database Laravel

Sebagaimana yang sudah kita pahami, database merupakan aplikasi yang terpisah dari PHP. Aplikasi database ini juga banyak jenis-jenisnya. Untuk aplikasi web berbasis PHP, yang paling sering dipakai adalah **MySQL/MariaDB**, **SQLite**, **PostgreSQL** serta **SQL Server**.

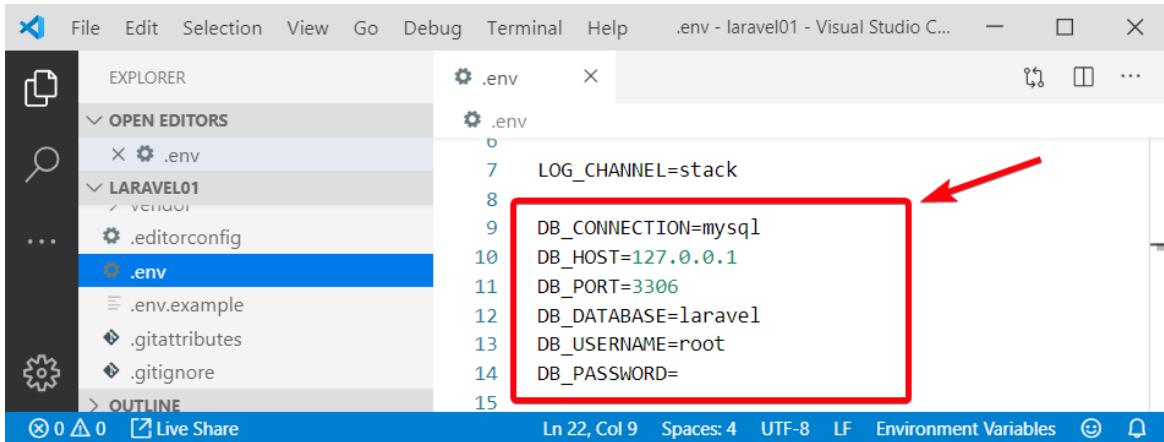
Keempat jenis database ini sudah didukung Laravel secara bawaan. Jika anda menggunakan aplikasi database lain, kemungkinan besar tetap bisa hanya saja butuh konfigurasi lebih lanjut.

Dalam buku ini kita akan memakai database MySQL (atau tepatnya MariaDB) bawaan XAMPP. Agar bisa berkomunikasi dengan MySQL, perlu mengatur beberapa konfigurasi Laravel.

File Konfigurasi .env dan config\database.php

Ketika membahas bab *Error Display & Proses Debugging*, kita sudah pelajari bahwa Laravel punya 2 lokasi pengaturan: konfigurasi lokal di file **.env** dan pengaturan global di folder **config**.

Kita akan mulai dari file **.env** terlebih dahulu. Silahkan buka file ini dan cari baris berikut:



```

LOG_CHANNEL=stack
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=

```

Gambar: Pengaturan database di file **.env**

Terdapat 6 baris pengaturan terkait database:

- ◆ **DB_CONNECTION:** Meskipun secara bawaan berisi 'mysql', ini bukanlah pengaturan tentang jenis database, tapi lebih ke *profile* yang berisi kumpulan konfigurasi yang ada di file `config/database.php` (kita akan lihat sesaat lagi). Namun memang secara bawaan profile 'mysql' berisi konfigurasi ke database MySQL.
Nilai lain yang bisa diisi adalah 'sqlite', 'pgsql' dan 'sqlsrv', yang merujuk ke jenis database SQLite, PostgreSQL dan SQL Server.
- ◆ **DB_HOST:** Dipakai untuk mengatur alamat server database. Secara bawaan sudah berisi `127.0.0.1`, yakni alamat localhost.
- ◆ **DB_PORT:** Dipakai untuk mengatur port dari aplikasi server database. Secara bawaan berisi angka `3306`, yakni port default bawaan MySQL.
- ◆ **DB_DATABASE:** Nama database yang akan dipakai. Secara bawaan sudah berisi '`laravel`', yang artinya Laravel akan langsung mencari sebuah database bernama '`laravel`'.
- ◆ **DB_USERNAME** dan **DB_PASSWORD:** Dipakai untuk membuat nama user dan password login ke database server. Secara bawaan nama user adalah `root`, dan password tidak diisi.

Selain `DB_DATABASE`, pengaturan lain sudah sesuai dengan konfigurasi MySQL bawaan XAMPP sehingga tidak perlu kita atur lebih lanjut. Namun jika anda mengubah konfigurasi di MySQL Server, bisa input dengan nilai yang sesuai.

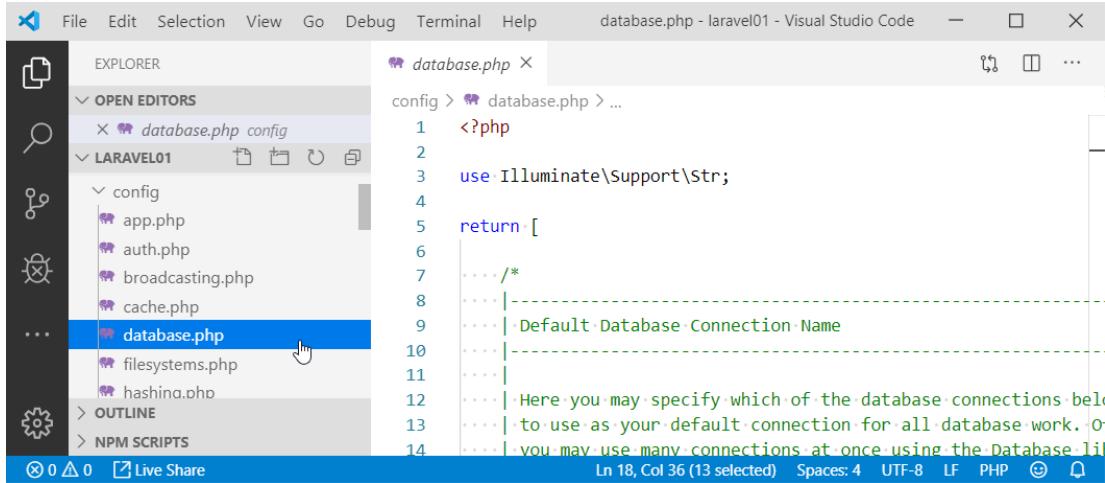
Jika anda menginstall laravel melalui laravel installer, yakni dari perintah `laravel new <nama_folder>`, maka pengaturan `DB_DATABASE` akan berisi `<nama_folder>`.

Misalnya jika mengetik `laravel new laravel01`, maka dalam file `.env` akan berisi `DB_DATABASE=laravel01`. Namun jika menginstall laravel dari perintah `composer create-project...`, pengaturan `.env` akan berisi `DB_DATABASE=laravel`.

Pengaturan default ini bisa berubah setiap saat, yang terpenting pengaturan `DB_DATABASE` harus diisi dengan nama database yang ada di MySQL.

Sekarang mari kita lihat file konfigurasi kedua, yakni di `config\database.php`:

Migration



```
database.php - laravel01 - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
config > database.php > ...
1 <?php
2
3 use Illuminate\Support\Str;
4
5 return [
6
7     /*
8     |--------------------------------------------------------------------------
9     | Default Database Connection Name
10    |--------------------------------------------------------------------------
11
12    |
13    | Here you may specify which of the database connections below
14    | you wish to use as your default connection for all database work. Of
15    | course, you may use many connections at once using the Database l...
Ln 18, Col 36 (13 selected) Spaces: 4 UTF-8 LF PHP ⚡
```

Gambar: Pengaturan database di file config\database.php

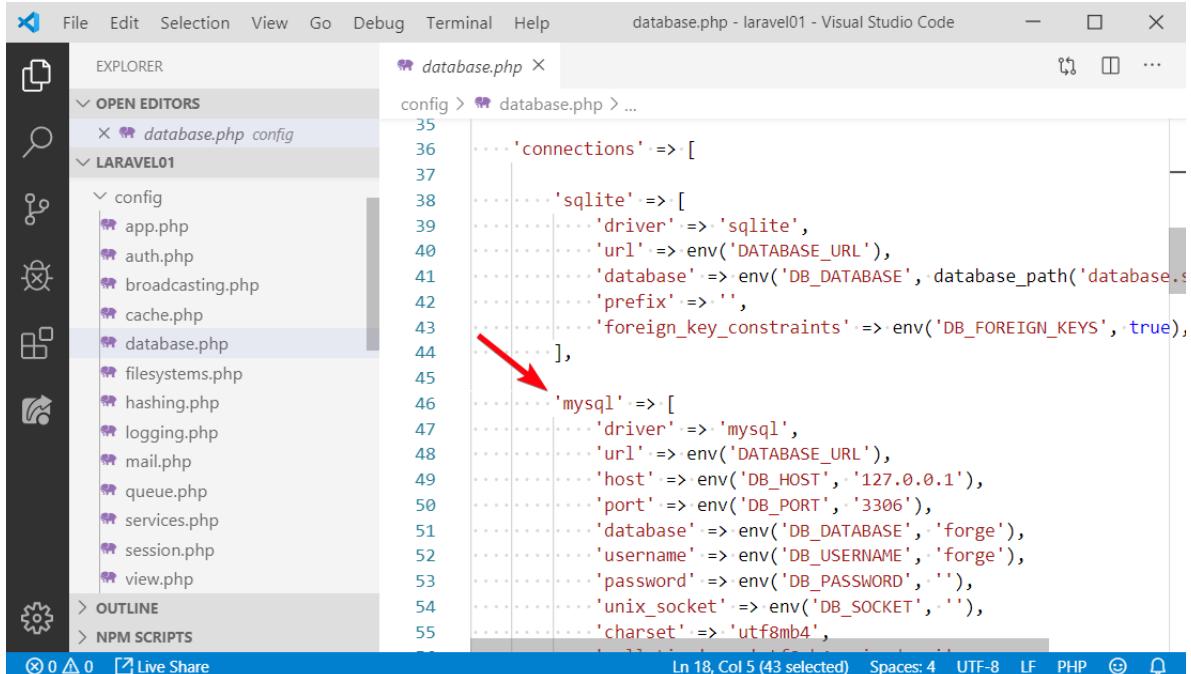
Pengaturan di file ini cukup panjang karena disertai cukup banyak baris komentar. Kita tidak akan bahas semuanya, hanya yang dirasa penting saja.

Pertama, di awal terdapat pengaturan berikut:

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Ini adalah pengaturan *profile* default yang akan dipakai. Kita juga pernah bahas bahwa *helper function* `env()` berguna untuk mengambil nilai konfigurasi yang ada di file `.env`. Jika di file `.env` tidak ditemukan konfigurasi `DB_CONNECTION`, maka secara default akan diisi nilai '`mysql`'.

Profile '`mysql`' ini merujuk ke pengaturan lanjutan yang ada di bagian '`connection`':



```
database.php - laravel01 - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
config > database.php > ...
35
36     'connections' => [
37
38         'sqlite' => [
39             'driver' => 'sqlite',
40             'url' => env('DATABASE_URL'),
41             'database' => env('DB_DATABASE', database_path('database.sqlite')),
42             'prefix' => '',
43             'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
44         ],
45
46         'mysql' => [
47             'driver' => 'mysql',
48             'url' => env('DATABASE_URL'),
49             'host' => env('DB_HOST', '127.0.0.1'),
50             'port' => env('DB_PORT', '3306'),
51             'database' => env('DB_DATABASE', 'forge'),
52             'username' => env('DB_USERNAME', 'forge'),
53             'password' => env('DB_PASSWORD', ''),
54             'unix_socket' => env('DB_SOCKET', ''),
55             'charset' => 'utf8mb4',
Ln 18, Col 5 (43 selected) Spaces: 4 UTF-8 LF PHP ⚡
```

Gambar: Pengaturan profile database di file config\database.php

Maksudnya, ketika nilai '`mysql`' diisi ke dalam pengaturan `DB_CONNECTION` di file `.env`, profile

konfigurasi 'mysql' inilah yang dipakai. Terdapat cukup banyak pengaturan yang bisa ubah, termasuk pilihan *charset*, *collation*, *engine*, dll.

Selain itu tersedia 4 profile untuk database lain, yakni 'sqlite', 'pgsql' dan 'sqlsrv'. Setiap profile ini bisa diatur lebih lanjut, dan untuk menggunakannya cukup ganti pengaturan `DB_CONNECTION` di file `.env`.

Jika anda bingung dengan banyaknya pengaturan yang ada, tidak perlu khawatir, karena apa yang ada di file `.env` sudah mencukupi untuk keperluan kita. File `config\database.php` ini lebih ditujukan untuk pengaturan yang lebih detail.

Dalam versi Laravel 9 yang saya gunakan, secara default pengaturan database di file `.env` adalah sebagai berikut:

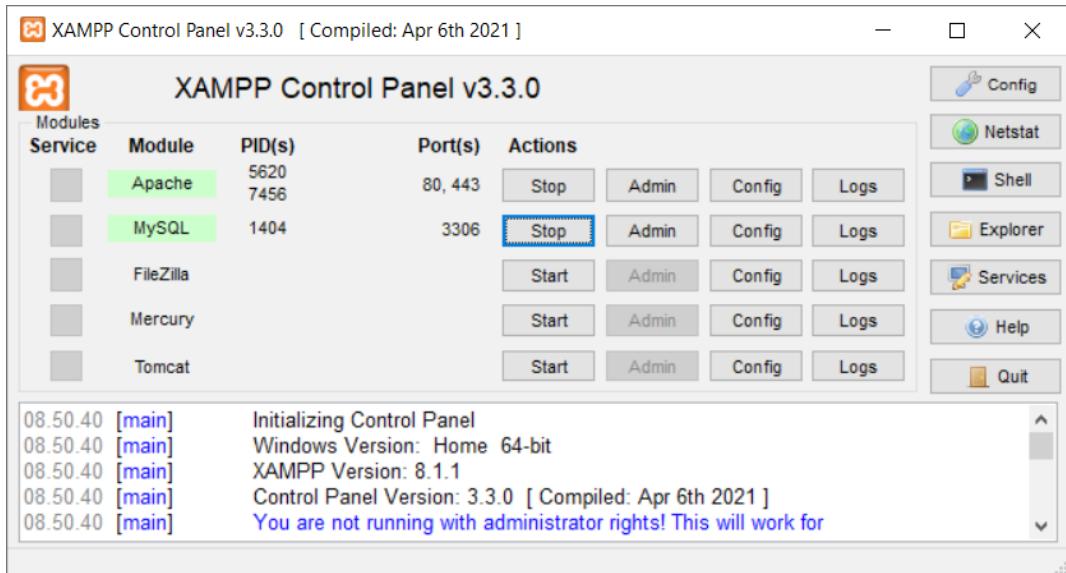
```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Semua pengaturan ini sudah sesuai dengan MySQL bawaan XAMPP. Yang kurang hanyalah database '`laravel`', yang akan kita buat sesaat lagi.

Jalankan Apache dan MySQL dari XAMPP

Kita bisa saja menggunakan aplikasi MySQL atau MariaDB yang diinstall terpisah, tapi akan lebih praktis menggunakan XAMPP karena semuanya sudah tersedia, termasuk phpMyAdmin.

Silahkan buka **XAMPP Control Panel** dan jalankan Apache web server dan MySQL server.



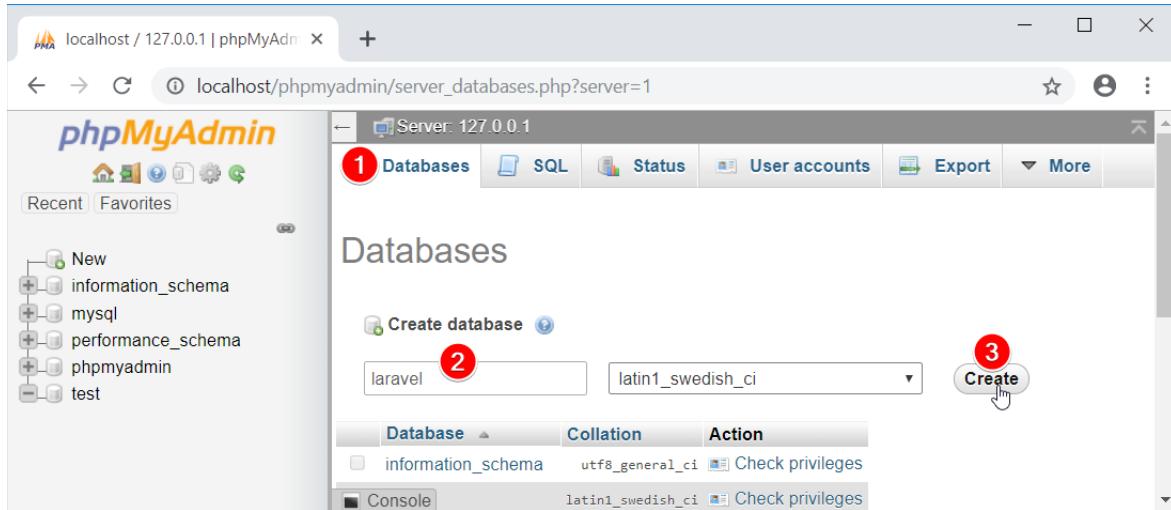
Gambar: Tampilan XAMPP Control Panel

Untuk membuat database, bisa dari cmd MySQL Client atau dari phpMyAdmin. Agar lebih

Migration

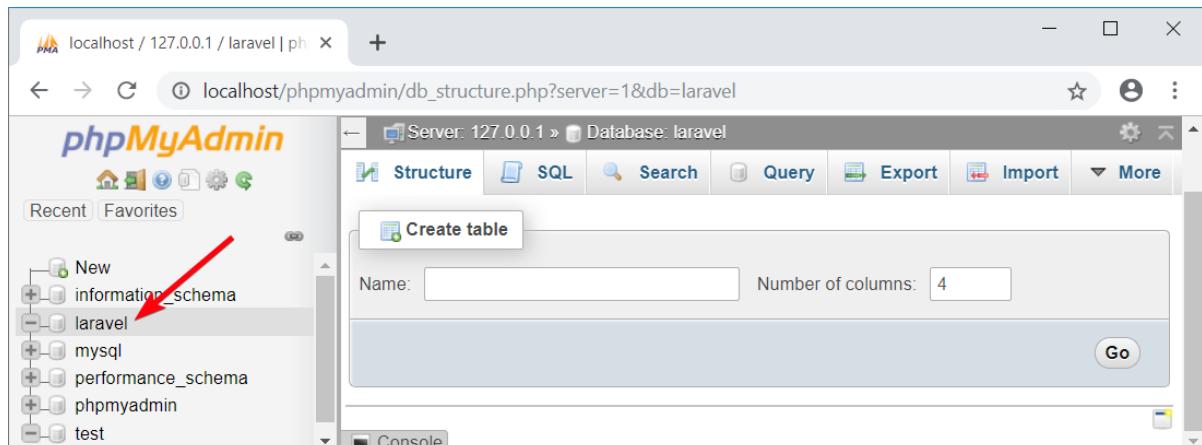
praktis, saya akan pakai phpMyAdmin. Silahkan buka web browser dan ketik alamat <http://localhost/phpmyadmin>.

Kemudian klik menu **Database** (1), ketik 'laravel' di inputan *Database name* (2), lalu klik tombol **Create** (3).



Gambar: Buat database 'laravel' dari phpMyAdmin

Jika tidak masalah, di sisi kiri akan tampil database **laravel**.



Gambar: Database 'laravel' berhasil dibuat

Persiapan kita sudah selesai, saatnya membahas **Laravel Migration**.

Nama database sebenarnya boleh suka-suka, tidak harus 'laravel'. Kita bisa saja membuat database 'belajar' atau 'universitas', kemudian input nama database ini ke dalam pengaturan DB_DATABASE di file .env.

13.2. Pengertian Migration

Secara sederhana, **migration** adalah fitur Laravel untuk men-generate tabel dengan cepat.

Hanya dengan 1 perintah `php artisan`, satu hingga puluhan tabel bisa langsung tersedia. Yang perlu kita lakukan hanyalah menyiapkan struktur tabel tersebut.

Jika anda pernah membaca buku **PHP Uncover** atau **OOP PHP Uncover**, di studi kasus pada bab terakhir saya selalu menyiapkan file `generate.php`. File ini terdiri dari perintah PHP serta query MySQL, yang ketika dijalankan secara otomatis akan men-generate semua tabel yang diperlukan. Migration kurang lebih seperti itu, plus berbagai fitur tambahan.

Migration sangat berguna untuk project yang dibuat dengan tim. Jika ada anggota tim ingin mencoba aplikasi, tidak perlu membuat tabel secara manual dari phpMyAdmin, tapi cukup jalankan file migration.

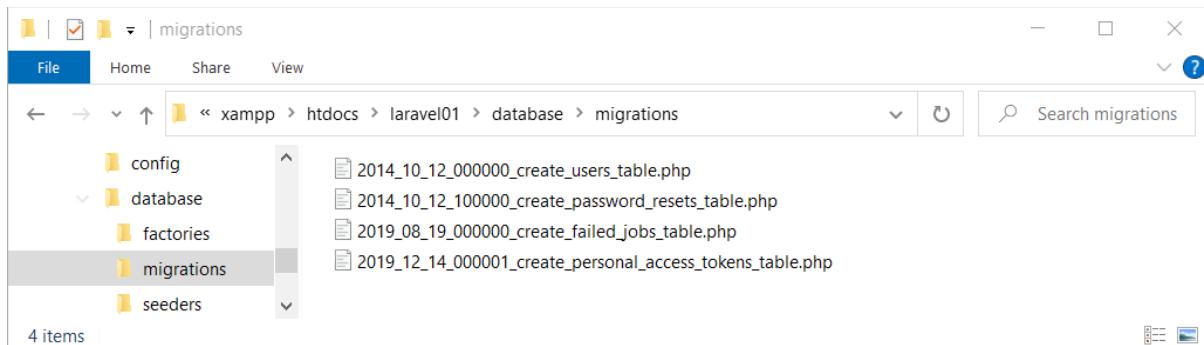
Dalam istilah yang lebih teknis, migration disebut sebagai **version control untuk database**. Selain dipakai membuat tabel, kita juga bisa membuat step-by-step perubahan struktur tabel.

Misalnya hari ini programmer A membuat tabel `mahasiswa` dengan 6 kolom, kemudian keesokan harinya programmer B memutuskan untuk memodifikasi menjadi 8 kolom. Namun tambahan kolom ini membuat program menjadi bermasalah, maka programmer C bisa memundurkan migration kembali ke posisi 6 kolom.

Kita akan bahas dengan detail cara penggunaan Migration.

13.3. File Migration Bawaan Laravel

Di Laravel 9, sudah terdapat beberapa file migration bawaan. File ini berada di folder `database\migrations`.



Gambar: Isi folder database\migrations

Dalam folder ini terdapat 4 file migration, yakni:

- `2014_10_12_000000_create_users_table.php`
- `2014_10_12_100000_create_password_resets_table.php`
- `2019_08_19_000000_create_failed_jobs_table.php`
- `2019_12_14_000001_create_personal_access_tokens_table.php`

Setiap file migration dipakai untuk membuat 1 tabel. Artinya 4 file migration ini akan membuat

4 buah tabel ke dalam database.

Nama file migration diawali dengan *timestamp*, yakni waktu file tersebut dibuat. Dalam contoh di atas, file migration `2014_10_12_000000_create_users_table.php` dibuat pada tanggal 12 Oktober 2014.

Setelah *timestamp*, diikuti dengan nama migration. Nama ini boleh bebas, misalnya `2019_10_20_000000_halo_sedang_belajar.php`. Namun format penulisan yang disarankan adalah sebagai berikut:

```
<timestamp>_<proses>_<nama_tabel(s)>_table.php
```

Jika misalkan kita ingin membuat file migration untuk tabel mahasiswa, lebih baik ditulis sebagai:

```
2019_10_20_000000_create_mahasiswas_table.php
```

Perhatikan bahwa ada tambahan huruf 's' di akhir. Ini adalah bentuk plural atau jamak dari sebuah kata dalam bahasa inggris. Yup, dalam laravel kita **di sarankan membuat nama tabel dalam bentuk jamak**, caranya tambahkan huruf s di akhir kata, seperti `mahasiswas`, `dosens`, `barangs`, dst.

Keperluan untuk membuat nama tabel dalam bentuk jamak (dengan tambahan 's' di akhir) baru akan terpakai saat kita masuk ke materi tentang **Eloquent**, yakni perintah singkat untuk penulisan query SQL.

Jika ingin membuat nama tabel tanpa akhiran 's' juga tidak masalah, tapi nanti perlu sedikit modifikasi tambahan. Agar tidak ada kendala, saya akan mengikuti aturan yang umum dipakai, yakni dengan membuat nama tabel dengan tambahan 's' di akhir.

Menulis kata dalam bentuk jamak ini juga memudahkan kita dalam menebak jenis data. Misalkan ada variabel bernama `$mahasiswas`, maka bisa di tebak kalau itu merupakan array yang terdiri dari banyak mahasiswa. Sedangkan variabel `$mahasiswa` hanya berisi data dari 1 mahasiswa saja.

Dengan aturan penulisan seperti ini, bisa kita lihat bahwa 4 file migration bawaan Laravel dipakai untuk membuat tabel **users**, **password_resets**, **failed_jobs**, dan **personal_access_tokens**.

Tabel `users` dan `password_resets` nantinya dipakai untuk proses *authentication*, yakni fitur register, login dan logout user (akan kita bahas dalam bab terpisah).

Tabel `failed_jobs` dipakai untuk fitur **Queues**, sedangkan tabel `personal_access_tokens` dipakai untuk **Laravel Sanctum**. Kedua materi ini masuk ke level advanced dan belum kita bahas dalam buku ini.

Mari lihat isi salah satu file, yakni `2014_10_12_000000_create_users_table.php`:

Migration

database\migrations\2014_10_12_000000_create_users_table.php

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateUsersTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('users', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->string('email')->unique();
20             $table->timestamp('email_verified_at')->nullable();
21             $table->string('password');
22             $table->rememberToken();
23             $table->timestamps();
24         });
25     }
26
27     /**
28      * Reverse the migrations.
29      *
30      * @return void
31      */
32     public function down()
33     {
34         Schema::dropIfExists('users');
35     }
36 }
```

Di awal terdapat kode program untuk proses import beberapa class internal Laravel, yakni `Migration`, `Blueprint` dan `Schema` (baris 3 – 5).

Kemudian di baris 7 masuk ke kode pembuatan class `CreateUsersTable` yang di extends dari class `Migration`. Class ini memiliki 2 buah method: `up()` dan `down()`.

Di dalam method `up()` inilah kode program untuk membuat tabel kita tulis, yakni menggunakan static method `Schema::create()`. Argument pertama dari method ini berupa nama tabel, yakni '`users`'. Sedangkan argument kedua berupa sebuah closure tempat kita menulis struktur kolom dari tabel '`users`'.

Mulai dari baris 17 – 23 adalah tempat untuk menulis nama dan tipe kolom. Mengenai apa saja tipe kolom untuk migration akan kita bahas sesaat lagi, tapi bisa dilihat sekilas bahwa nantinya

untuk tabel `users` akan terdapat kolom '`id`', '`name`', '`email`', '`email_verified_at`' serta '`password`'.

Method selanjutnya adalah `down()` di baris 32. Method ini berisi perintah kebalikan dari method `up()`. Jika di method `up()` kita membuat tabel '`users`', maka di method `down()` ini ditulis perintah untuk menghapus tabel '`users`' menggunakan perintah `Schema::dropIfExists('users')`. Inilah teknik yang dipakai agar migration bisa berfungsi sebagai *version control*.

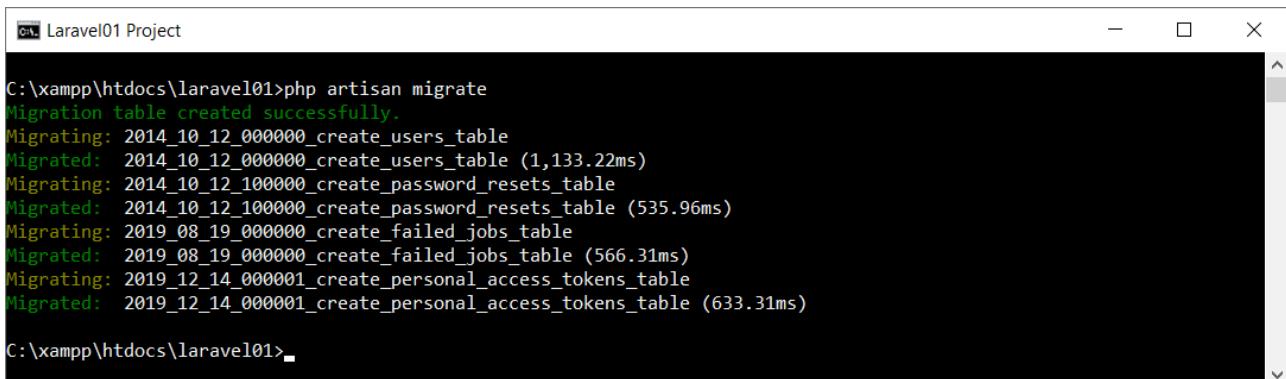
Silahkan lihat juga kode program untuk kedua file migration lainnya, anda bisa perhatikan bahwa semua file identik satu sama lain, yang berubah hanyalah isi dari method `up()` dan `down()`.

13.4. Menjalankan Migration

Kita akan coba jalankan file migration bawaan Laravel. Ketiga file migration ini akan membuat tabel `users`, `password_resets` dan `failed_jobs` ke dalam database **laravel** yang sudah kita siapkan.

Untuk menjalankan migration, silahkan buka **cmd**, masuk ke folder laravel dan jalankan perintah berikut:

```
php artisan migrate
```



```
C:\xampp\htdocs\laravel01>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (1,133.22ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (535.96ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (566.31ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (633.31ms)

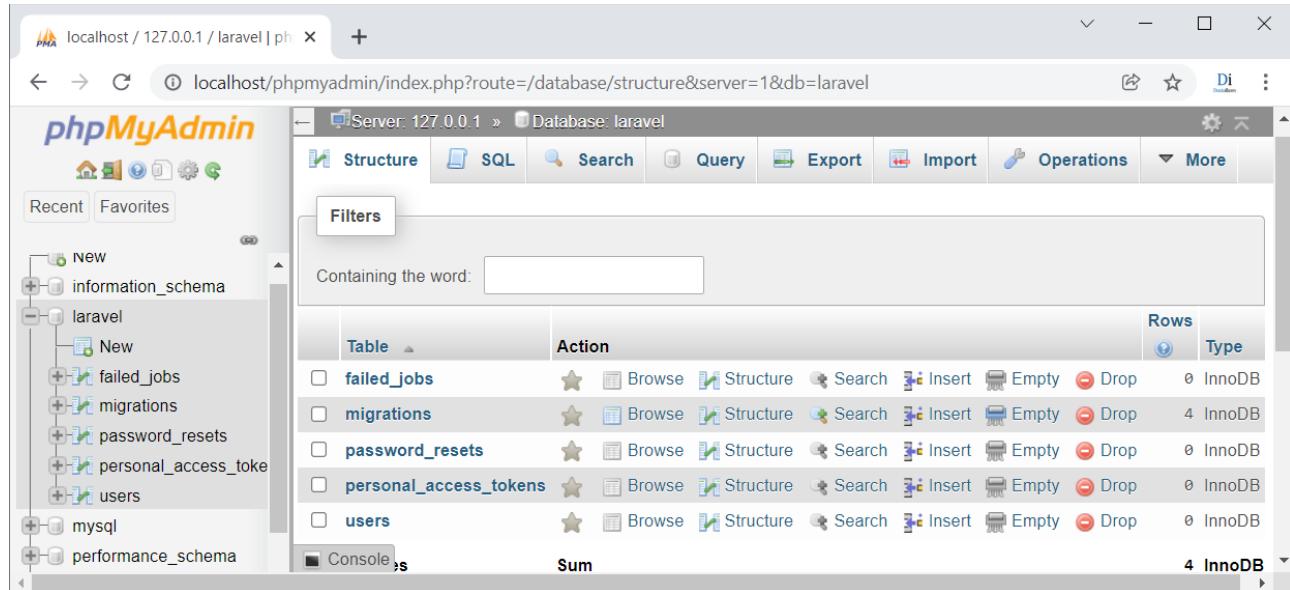
C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan perintah `php artisan migrate`

Apabila tampil pesan error, silahkan cek kembali apakah MySQL Server bawaan XAMPP sudah berjalan atau belum.

Jika tampil hasil seperti di atas, maka proses migration sudah sukses berjalan. Silahkan buka phpMyAdmin dan periksa isi database **laravel**.

Migration

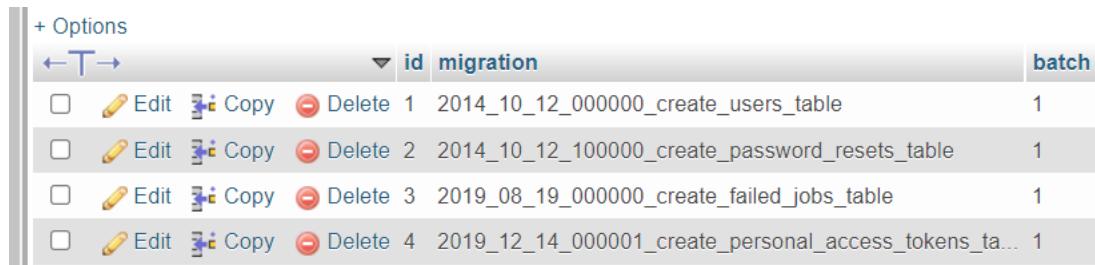


The screenshot shows the phpMyAdmin interface for the 'laravel' database. On the left, the database tree shows 'information_schema', 'laravel' (which contains 'failed_jobs', 'migrations', 'password_resets', 'personal_access_tokens', and 'users'), 'mysql', and 'performance_schema'. The 'migrations' table is currently selected. The main area displays the table structure with columns: id, migration, and batch. There are four rows in the table.

Table	Action	Rows	Type
failed_jobs	Browse Structure Search Insert Empty Drop	0	InnoDB
migrations	Browse Structure Search Insert Empty Drop	4	InnoDB
password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB
personal_access_tokens	Browse Structure Search Insert Empty Drop	0	InnoDB
users	Browse Structure Search Insert Empty Drop	0	InnoDB
Console	Sum	4	InnoDB

Gambar: Isi database 'laravel'

Ternyata terdapat 5 tabel, yakni tabel users, password_resets, failed_jobs, personal_access_tokens serta tabel migrations. Tabel migrations secara khusus dipakai untuk mencatat proses migration yang terjadi. Mari kita lihat isi tabel ini:



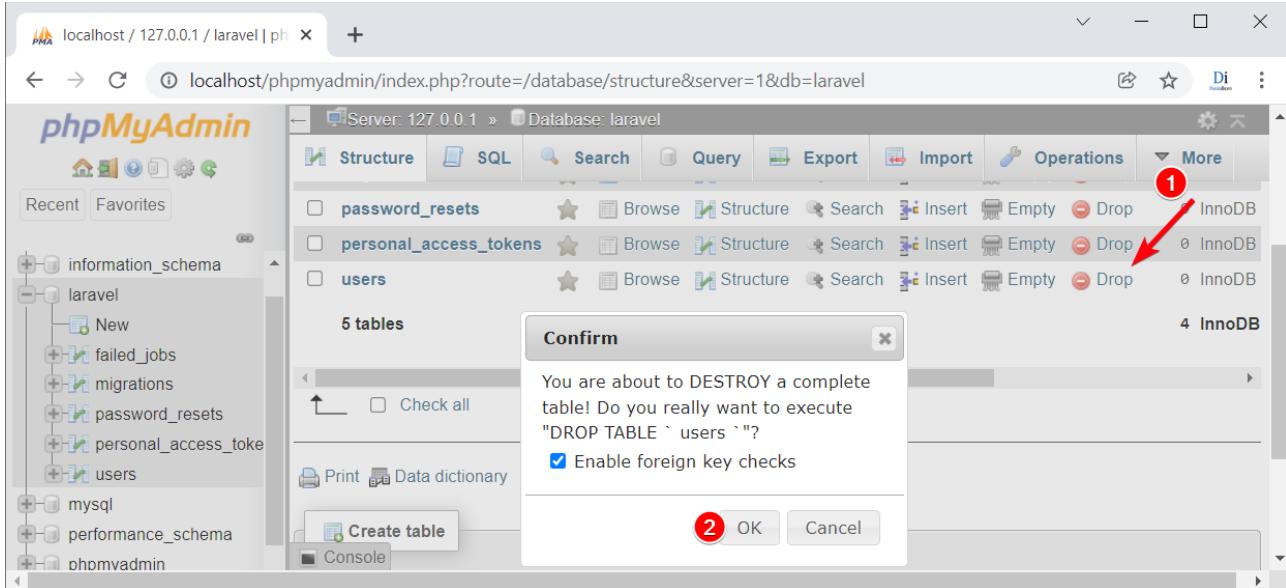
	+ Options	← →	id	migration	batch
<input type="checkbox"/>				1 2014_10_12_000000_create_users_table	1
<input type="checkbox"/>				2 2014_10_12_100000_create_password_resets_table	1
<input type="checkbox"/>				3 2019_08_19_000000_create_failed_jobs_table	1
<input type="checkbox"/>				4 2019_12_14_000001_create_personal_access_tokens_ta...	1

Gambar: Isi tabel migrations

Tabel migration terdiri dari 3 kolom: `id`, `migration` dan `batch`. Kolom `migration` berisi semua file migration yang sudah dijalankan, sedangkan kolom `batch` dipakai untuk mencatat urutan proses migration. Semua file migration memiliki batch 1, karena kita memang baru menjalankan 1 kali perintah `php artisan migrate`.

Mari lakukan sedikit percobaan dengan menghapus tabel `users`. Balik ke menu database, klik link **Drop** di baris tabel `users` (1), lalu klik tombol **OK** (2) pada jendela konfirmasi.

Migration



Gambar: Hapus tabel users

Situasi ini sering terjadi dimana kita tidak sengaja menghapus sebuah tabel. Namun tidak masalah, karena selama ada file migration, tabel bisa dibuat lagi dengan mudah.

Jalankan kembali perintah `php artisan migrate`:

```
C:\xampp\htdocs\laravel01>php artisan migrate
Nothing to migrate.

C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan kembali perintah php artisan migrate

Hasilnya tampil pesan '*Nothing to migrate.*', dimana Laravel mendeteksi tidak ada file migration yang belum dijalankan. Ini terjadi karena file migration untuk tabel `users` sudah terdaftar di tabel `migrations`.

Solusinya, kita bisa me-*rollback* atau menghapus semua migration yang terdaftar dengan perintah `php artisan migrate:reset`

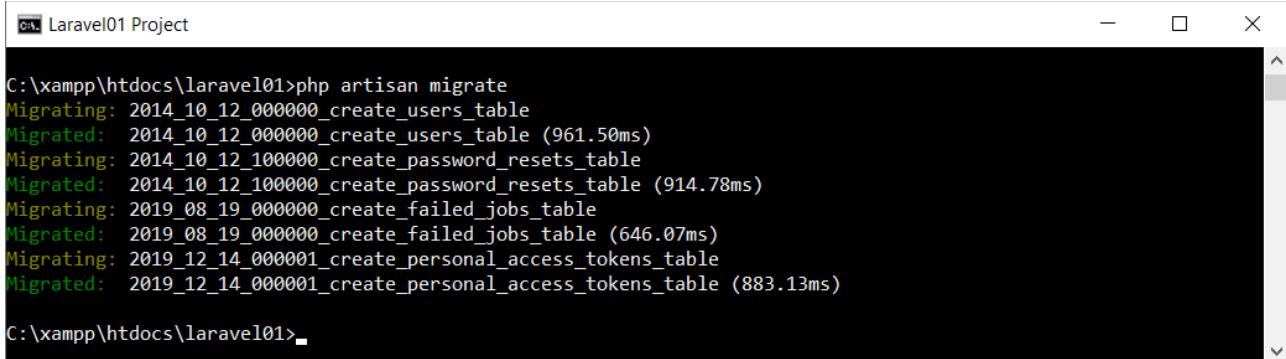
```
C:\xampp\htdocs\laravel01>php artisan migrate:reset
Rolling back: 2019_12_14_000001_create_personal_access_tokens_table
Rolled back: 2019_12_14_000001_create_personal_access_tokens_table (315.58ms)
Rolling back: 2019_08_19_000000_create_failed_jobs_table
Rolled back: 2019_08_19_000000_create_failed_jobs_table (109.21ms)
Rolling back: 2014_10_12_100000_create_password_resets_table
Rolled back: 2014_10_12_100000_create_password_resets_table (177.02ms)
Rolling back: 2014_10_12_000000_create_users_table
Rolled back: 2014_10_12_000000_create_users_table (2.81ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Rollback semua tabel dengan perintah php artisan migrate:reset

Migration

Perintah `php artisan migrate:reset` akan menghapus isi tabel `migration` dan juga menghapus semua tabel yang ada. Dengan demikian, kita bisa jalankan kembali `php artisan migrate`.



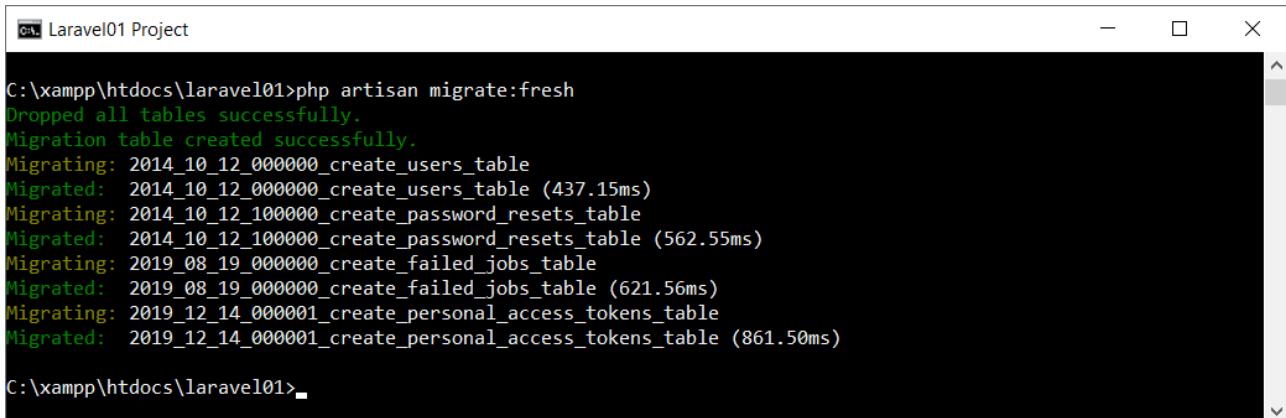
```
C:\xampp\htdocs\laravel01>php artisan migrate
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (961.50ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (914.78ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (646.07ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (883.13ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Jalankan kembali migration

Dengan perintah ini, semua tabel kembali ke kondisi semula. Kita tidak perlu khawatir untuk memodifikasi tabel, karena jika ada sesuatu yang salah, tinggal jalankan ulang migration.

Laravel juga menyediakan perintah `php artisan migrate:fresh` sebagai cara singkat untuk menghapus semua tabel dan langsung membuatnya kembali. Perintah ini ibarat perpaduan dari `php artisan migrate:reset` dan `php artisan migrate` dalam sekali jalan.



```
C:\xampp\htdocs\laravel01>php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (437.15ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (562.55ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (621.56ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (861.50ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Me-rollback dan membuat ulang semua tabel

Fitur **migration** hanya dipakai untuk mengelola tabel dan struktur tabel, tapi tidak dengan isi tabel itu sendiri. Jika anda juga ingin meng-generate isi tabel secara otomatis, tersedia fitur lanjutan yang dinamakan **Seeding**. Untuk saat ini kita akan fokus ke migration saja. Materi seeding akan menjadi jatah buku [Laravel In Depth #1](#).

13.5. Migration Rollback

Pada saat menjalankan perintah `php artisan migrate`, file migration akan di eksekusi secara berurutan sesuai tanggal **timestamp**. File dengan timestamp terbaru akan dijalankan paling

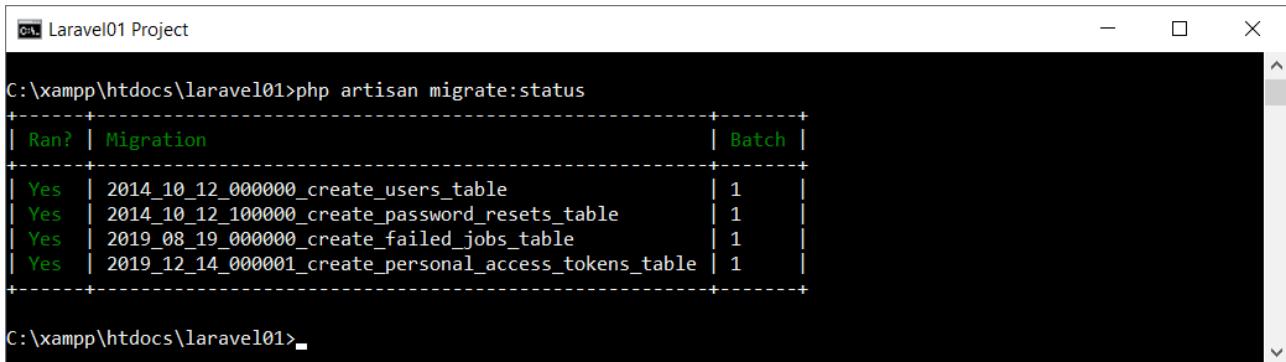
Migration

akhir.

Pada prakteknya, bisa saja file migration ini ditambah secara bertahap. Misalkan programmer A sedang membuat fitur baru dan butuh tambahan sebuah tabel, maka dia bisa membuat dan menjalankan file migration untuk `tabel_a`. Esoknya programmer B juga bisa membuat file migration lain untuk `tabel_b`, dst.

Jika tiba-tiba terjadi masalah akibat penambahan tabel ini, kita bisa melakukan proses *rollback* sebagian terhadap file migration. Jadi seolah-olah kembali ke struktur database di hari sebelumnya.

Untuk melihat daftar urutan migration, bisa dari perintah `php artisan migrate:status`.



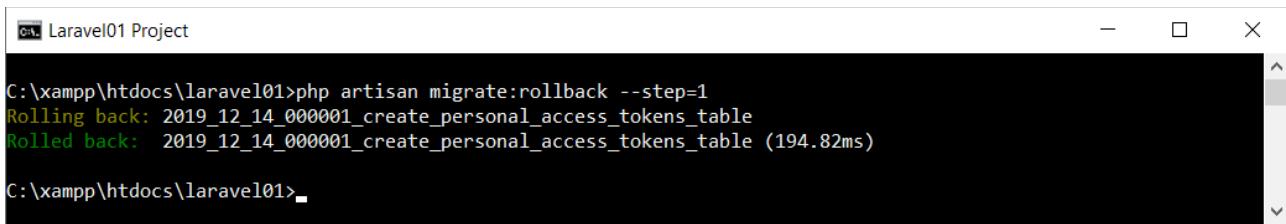
Ran?	Batch
Yes	1

Gambar: Hasil dari perintah `php artisan migrate:status`

Di sini terlihat 4 file migration yang di urutkan berdasarkan timestamp, mulai dari tabel `users`, tabel `password_resets`, dan tabel `failed_jobs`.

Jika saya ingin mengembalikan posisi sebelum file migration terakhir, bisa menggunakan perintah berikut:

```
php artisan migrate:rollback --step=1
```



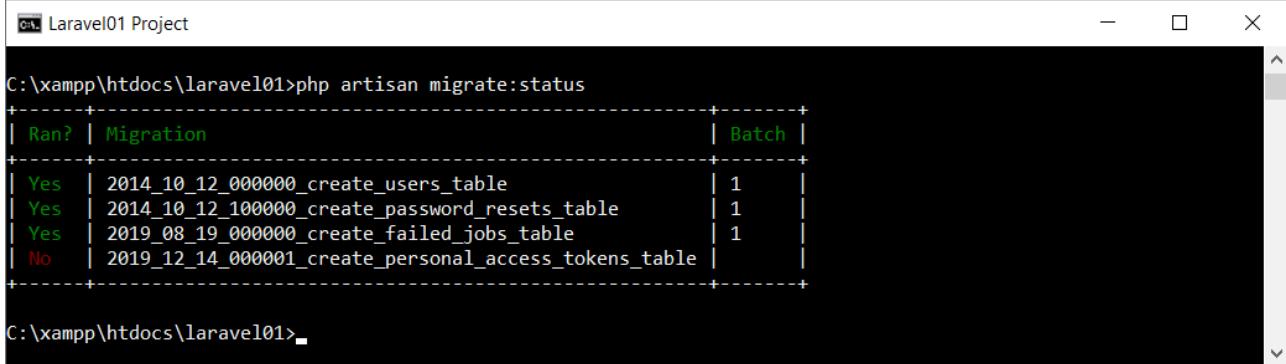
Gambar: Hasil dari perintah `php artisan migrate:rollback --step=1`

Perintah diatas akan me-*rollback* 1 file migration terakhir dan juga menghapus tabelnya. Akibat perintah ini, tabel `personal_access_tokens` sudah tidak ada lagi di database.

Dalam project besar, bisa saja perlu mundur 6 atau 10 step jika diperlukan. Caranya, cukup mengatur nilai `--step`. Misalnya untuk mundur sebanyak 7 file migration, perintahnya adalah `php artisan migrate:rollback --step=7`.

Jika perintah `php artisan migrate:status` dijalankan kembali, akan terlihat sedikit perubahan.

Migration

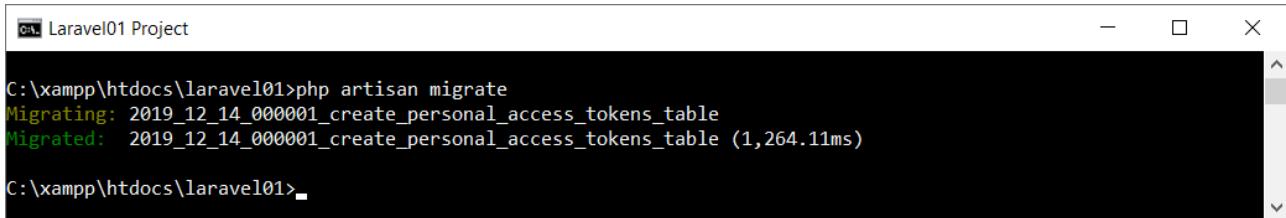


Ran?	Migration	Batch
Yes	2014_10_12_00000_create_users_table	1
Yes	2014_10_12_100000_create_password_resets_table	1
Yes	2019_08_19_000000_create_failed_jobs_table	1
No	2019_12_14_000001_create_personal_access_tokens_table	

Gambar: Hasil dari perintah `php artisan migrate:status`

Sekarang isi kolom "Ran?" untuk file migration `personal_access_tokens` berisi No, yang menandakan file migration untuk tabel tersebut belum dijalankan atau sudah di rollback.

Jika kita berubah pikiran dan ingin menjalankan kembali file migration untuk tabel `failed_jobs`, cukup dengan perintah `php artisan migrate`.



```
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (1,264.11ms)
```

Gambar: Jalankan kembali proses migrate

Hasilnya, tabel `personal_access_tokens` sudah bisa digunakan kembali. Inilah fitur migration sebagai *version control*, dimana kita bisa 'maju-mundur' untuk semua file migration. Jika ternyata tabel baru yang ditambahkan menjadi masalah, tinggal di-rollback saja. Atau jika ada tabel yang tidak sengaja terhapus, tinggal jalankan ulang file migration.

#Exercize

Anda bisa lakukan sebuah percobaan. Jalankan perintah `php artisan migrate:rollback --step=1`, lalu cek dengan `php artisan migrate:status`. Kemudian jalankan `php artisan migrate`. Perhatikan isi kolom `batch`, apakah ada perubahan?

Test kembali `php artisan migrate:rollback --step=2`, kemudian `php artisan migrate`, dan cek kembali di `php artisan migrate:status`.

13.6. Membuat Migration

Kita sudah pelajari bagaimana menjalankan file migration bawaan Laravel. Sekarang saatnya untuk membuat sebuah file migration sendiri.

Sama seperti controller, file migration bisa dibuat dari teks editor. Caranya, copy kode dari file

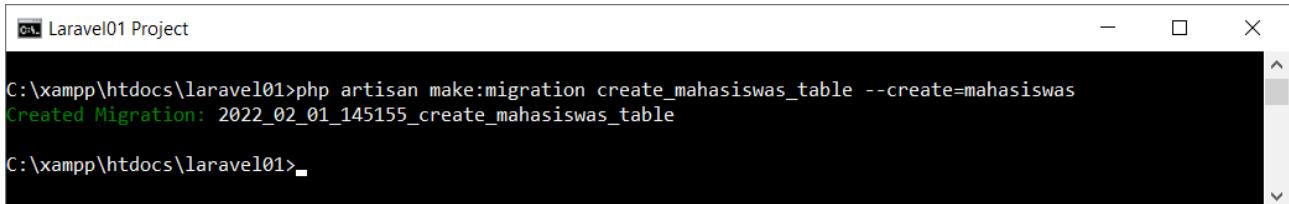
Migration

migration yang sudah ada, lalu edit di bagian yang diperlukan. Atau cara yang lebih praktis adalah dari perintah **php artisan** dengan format berikut:

```
php artisan make:migration <nama_migration> --create=<nama_tabel>
```

Untuk **nama_migration** bisa di isi suka-suka, namun penulisan terbaik adalah dengan format: **<nama_proses>_<nama_tabel(s)>_table**. Sebagai contoh, untuk membuat file migration tabel **mahasiswa**, bisa dengan menjalankan perintah berikut:

```
php artisan make:migration create_mahasiswa_table --create=mahasiswa
```



```
C:\xampp\htdocs\laravel01>php artisan make:migration create_mahasiswa_table --create=mahasiswa
Created Migration: 2022_02_01_145155_create_mahasiswa_table
C:\xampp\htdocs\laravel01>
```

Gambar: Membuat file migration

Hasilnya, di dalam folder **database\migrations** akan tampil file **2022_02_01_145155_create_mahasiswa_table.php**. Nama file ini bisa berbeda karena tanggal timestamp di generate secara otomatis dari sistem komputer.

Tambahan flag **--create=mahasiswa** boleh saja tidak ditulis dan Laravel akan membuat tabel berdasarkan nama file. Misal jika dijalankan **php artisan make:migration create_dosens_table**, maka Laravel akan men-generate nama tabel **dosens**.

Penamaan tabel dengan akhiran 's' ini sudah kita bahas sebelumnya, yakni untuk mengikuti aturan Laravel dimana nama tabel sebaiknya dibuat plural.

Ini bukan kewajiban karena Laravel tetap mengizinkan kita membuat nama tabel dengan kata biasa, namun nanti perlu sedikit konfigurasi jika ingin menggunakan fitur **Eloquent**.

Agar lebih mudah dan terbiasa dengan penamaan umum di aplikasi Laravel, saya akan buat semua nama tabel dengan akhiran 's', seperti **mahasiswa**, **gurus**, **barang**, dst.

Alternatif lain yang juga banyak di pilih programmer indonesia adalah memakai kata bahasa inggris seperti **students**, **teachers**, **items**, dst.

Berikut isi dari file migration tersebut:

```
database/migrations/2022_02_01_145155_create_mahasiswa_table.php

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
```

Migration

```
8
9  {
10     /**
11      * Run the migrations.
12      *
13      * @return void
14      */
15     public function up()
16     {
17         Schema::create('mahasiswa', function (Blueprint $table) {
18             $table->id();
19             $table->timestamps();
20         });
21     }
22
23     /**
24      * Reverse the migrations.
25      *
26      * @return void
27      */
28     public function down()
29     {
30         Schema::dropIfExists('mahasiswa');
31     }
32 }
```

Kode ini sangat mirip seperti yang ada di file migration bawaan Laravel. Kita akan langsung fokus ke method `up()` saja, karena di sinilah tempat untuk menulis struktur tabel.

Format penulisan nama kolom tabel adalah sebagai berikut:

```
$table-><tipe_data_kolom>('<nama_kolom>')
```

Untuk `tipe_data_kolom`, tidak selalu sesuai dengan bawaan MySQL. Laravel menyediakan berbagai tipe data baru yang kadang merupakan gabungan dari tipe data kolom MySQL.

Sebagai contoh, di baris 17 terdapat pemanggilan method `$table->id()`, ini dipakai untuk membuat kolom bertipe **big integer** yang langsung di set dengan **auto increment**.

Tabel berikut merangkum jenis tipe data kolom yang bisa dipakai:

Command	Description
<code>\$table->id();</code>	Alias of <code>\$table->bigIncrements('id');</code> .
<code>\$table->foreignKey('user_id');</code>	Alias of <code>\$table->unsignedBigInteger('user_id')</code>
<code>\$table->bigIncrements('id');</code>	Auto-incrementing UNSIGNED BIGINT (primary key) equivalent column.
<code>\$table->bigInteger('votes');</code>	BIGINT equivalent column.
<code>\$table->binary('data');</code>	BLOB equivalent column.
<code>\$table->boolean('confirmed');</code>	BOOLEAN equivalent column.

Command	Description
<code>\$table->char('name', 100);</code>	CHAR equivalent column with an optional length.
<code>\$table->date('created_at');</code>	DATE equivalent column.
<code>\$table->dateTime('created_at');</code>	DATETIME equivalent column.
<code>\$table->decimal('amount', 8, 2);</code>	DECIMAL equivalent column with a precision (total digits) and scale (decimal digits).
<code>\$table->double('amount', 8, 2);</code>	DOUBLE equivalent column with a precision (total digits) and scale (decimal digits).
<code>\$table->enum('level', ['easy', 'hard']);</code>	ENUM equivalent column.
<code>\$table->float('amount', 8, 2);</code>	FLOAT equivalent column with a precision (total digits) and scale (decimal digits).
<code>\$table->increments('id');</code>	Auto-incrementing UNSIGNED INTEGER (primary key) equivalent column.
<code>\$table->integer('votes');</code>	INTEGER equivalent column.
<code>\$table->ipAddress('visitor');</code>	IP address equivalent column.
<code>\$table->longText('description');</code>	LONGTEXT equivalent column.
<code>\$table->mediumIncrements('id');</code>	Auto-incrementing UNSIGNED MEDIUMINT (primary key) equivalent column.
<code>\$table->mediumInteger('votes');</code>	MEDIUMINT equivalent column.
<code>\$table->mediumText('description');</code>	MEDIUMTEXT equivalent column.
<code>\$table->rememberToken();</code>	Adds a nullable <code>remember_token</code> VARCHAR(100) equivalent column.
<code>\$table->set('flavors', 'strawberry');</code>	SET equivalent column.
<code>\$table->smallIncrements('id');</code>	Auto-incrementing UNSIGNED SMALLINT (primary key) equivalent column.
<code>\$table->smallInteger('votes');</code>	SMALLINT equivalent column.
<code>\$table->softDeletes();</code>	Adds a nullable <code>deleted_at</code> TIMESTAMP equivalent column for soft deletes.
<code>\$table->string('name', 100);</code>	VARCHAR equivalent column with a optional length.
<code>\$table->text('description');</code>	TEXT equivalent column.
<code>\$table->time('sunrise');</code>	TIME equivalent column.
<code>\$table->timestamp('added_on');</code>	TIMESTAMP equivalent column.
<code>\$table->timestamps();</code>	Adds nullable <code>created_at</code> and <code>updated_at</code> TIMESTAMP equivalent columns.
<code>\$table->tinyIncrements('id');</code>	Auto-incrementing UNSIGNED TINYINT (primary key) equivalent column.
<code>\$table->tinyInteger('votes');</code>	TINYINT equivalent column.

Command	Description
<code>\$table->unsignedBigInteger('votes');</code>	UNSIGNED BIGINT equivalent column.
<code>\$table->unsignedDecimal('amount',8,2);</code>	UNSIGNED DECIMAL equivalent column with a precision (total digits) and scale (decimal digits).
<code>\$table->unsignedInteger('votes');</code>	UNSIGNED INTEGER equivalent column.
<code>\$table->unsignedMediumInteger('votes');</code>	UNSIGNED MEDIUMINT equivalent column.
<code>\$table->unsignedSmallInteger('votes');</code>	UNSIGNED SMALLINT equivalent column.
<code>\$table->unsignedTinyInteger('votes');</code>	UNSIGNED TINYINT equivalent column.
<code>\$table->uuid('id');</code>	UUID equivalent column.
<code>\$table->year('birth_year');</code>	YEAR equivalent column.

Sebagian besar jenis tipe data sudah ada di tabel di atas. Untuk daftar lengkapnya bisa ke <https://laravel.com/docs/9.x/migrations>.

Dari daftar ini ada beberapa tipe data yang bisa men-generate kolom lain. Sebagai contoh `$table->timestamps()` akan menambah 2 buah kolom, yakni `created_at` and `updated_at` ke dalam tabel. Kedua kolom bertipe `timestamp` yang akan terisi otomatis pada saat data tabel ditambah / dimodifikasi (tidak perlu diinput manual).

Kolom `created_at` and `updated_at` ini sangat bermanfaat proses mengurutkan data berdasarkan tanggal dibuat dan tanggal modifikasi. Disarankan untuk selalu menambah `$table->timestamps()` untuk setiap tabel yang dibuat.

Selain itu beberapa method punya pengaturan tambahan yang di input sebagai argument kedua atau ketiga. Sebagai contoh, `$table->char('name', 100)` dipakai untuk membuat kolom 'name' dengan tipe data `CHAR(100)`. Begitu juga dengan method `$table->double('amount', 8, 2)` yang dipakai untuk membuat kolom 'amount' dengan tipe data `DOUBLE(8,2)`.

Bahasan tentang jenis-jenis tipe data untuk kolom tabel MySQL merupakan materi di luar buku ini. Jika anda tertarik, bisa membacanya dari buku [MySQL Uncover](#). Di sana saya membahas tipe data tabel MySQL dengan cukup detail (termasuk fitur MySQL lain).

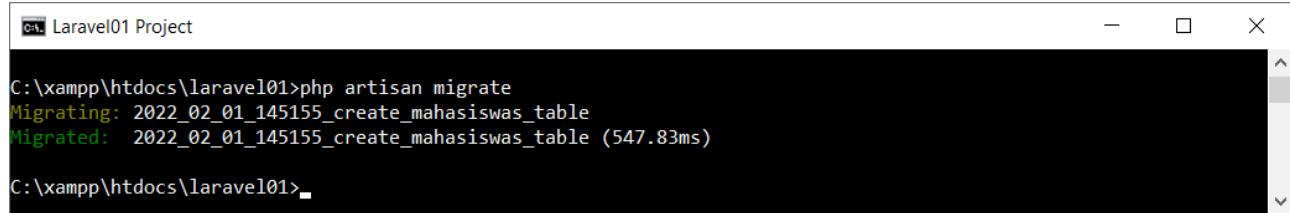
Dari file migration yang di generate oleh perintah php artisan, tabel **mahasiswa** sudah memiliki 2 method pembuatan kolom:

```

1  public function up()
2  {
3      Schema::create('mahasiswa', function (Blueprint $table) {
4          $table->id();
5          $table->timestamps();
6      });
7 }
```

Migration

Langsung saja kita coba jalankan dengan perintah `php artisan migrate`.

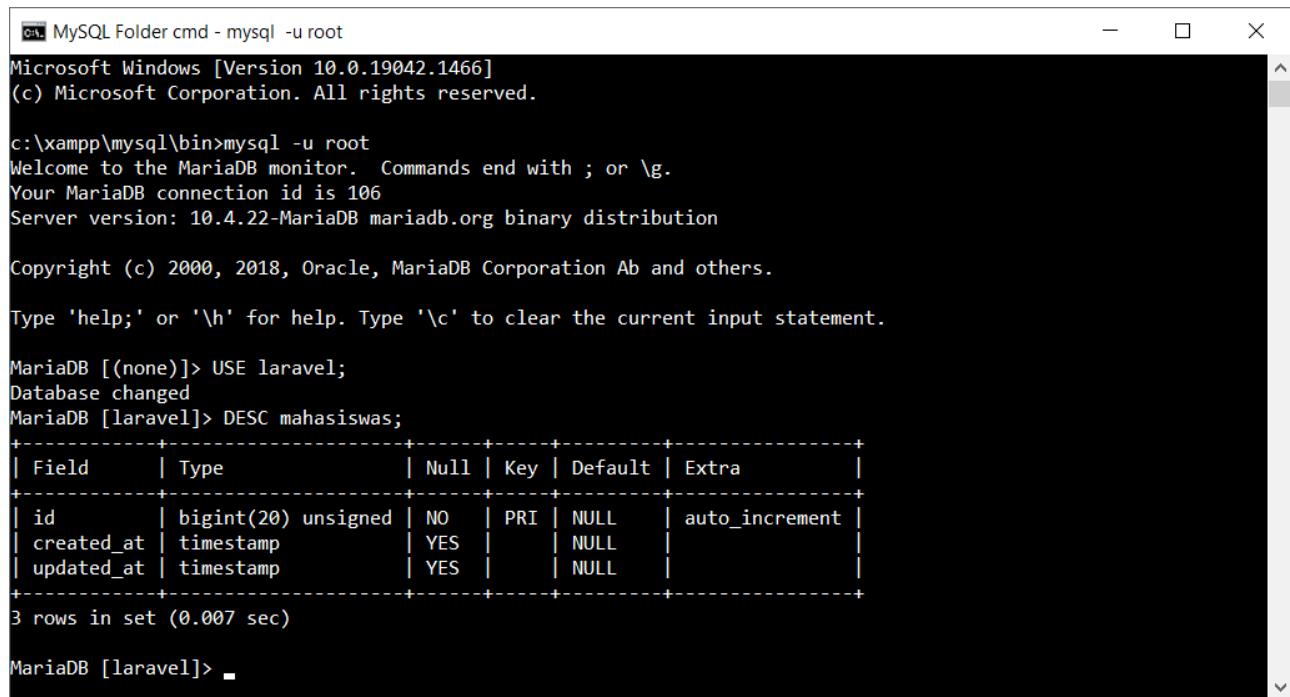


```
C:\xampp\htdocs\laravel01>php artisan migrate
Migrating: 2022_02_01_145155_create_mahasiswa_table
Migrated: 2022_02_01_145155_create_mahasiswa_table (547.83ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan file migration untuk tabel mahasiswa

Untuk melihat struktur tabel, bisa dengan menjalankan perintah query `DESC mahasiswa` dari cmd MySQL Client:



```
c:\xampp\mysql\bin>mysql -u root
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

c:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 106
Server version: 10.4.22-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE laravel;
Database changed
MariaDB [laravel]> DESC mahasiswa;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| id         | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| created_at | timestamp   | YES  |     | NULL    |              |
| updated_at | timestamp   | YES  |     | NULL    |              |
+-----+-----+-----+-----+-----+
3 rows in set (0.007 sec)

MariaDB [laravel]>
```

Gambar: Struktur detail dari tabel mahasiswa

Hasilnya, terlihat tabel mahasiswa memiliki struktur kolom sebagai berikut:

- Kolom 'id' bertipe `BIGINT(20)` dengan atribut `unsigned`, `primary key` dan `auto increment`.
- Kolom 'created_at' bertipe `TIMESTAMP` dan bisa diisi nilai null.
- Kolom 'updated_at' bertipe `TIMESTAMP` dan bisa diisi nilai null.

Saya akan tambah beberapa kolom lain ke dalam file migrate `mahasiswa`:

`database/migrations/2022_02_01_145155_create_mahasiswa_table.php`

```
1 public function up()
2 {
3     Schema::create('mahasiswa', function (Blueprint $table) {
4         $table->id();
5         $table->char('nim',8);
```

Migration

```
6     $table->string('nama');
7     $table->string('tempat_lahir');
8     $table->date('tanggal_lahir');
9     $table->string('fakultas');
10    $table->string('jurusan');
11    $table->decimal('ipk',3,2);
12    $table->timestamps();
13  });
14 }
```

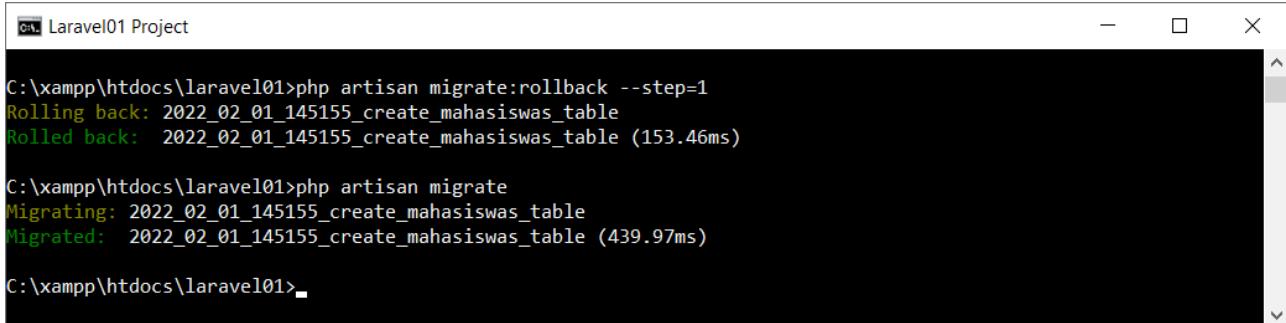
Terdapat tambahan 7 kolom baru (baris 5 - 11). Di baris 5, method `$table->char('nim',8)` dipakai untuk membuat kolom 'nim' dengan tipe CHAR(8).

Kemudian di baris 6, 7, 9 dan 10 saya menggunakan `$table->string()` untuk membuat kolom 'nama', 'tempat_lahir', 'fakultas', dan 'jurusan'. Keempat kolom ini akan dikonversi menjadi tipe data VARCHAR(255).

Di baris 8, method `$table->date('tanggal_lahir')` dipakai untuk membuat kolom 'tanggal_lahir' dengan tipe data DATE.

Terakhir di baris 11 terdapat method `$table->decimal('ipk',3,2)` untuk membuat kolom 'ipk' dengan tipe data DECIMAL(3,2).

Save file migrate, rollback 1 step dengan method `php artisan migrate:rollback --step=1`, lalu jalankan kembali `php artisan migrate`:



```
Laravel01 Project
C:\xampp\htdocs\laravel01>php artisan migrate:rollback --step=1
Rolling back: 2022_02_01_145155_create_mahasiswa_table
Rolled back: 2022_02_01_145155_create_mahasiswa_table (153.46ms)

C:\xampp\htdocs\laravel01>php artisan migrate
Migrating: 2022_02_01_145155_create_mahasiswa_table
Migrated: 2022_02_01_145155_create_mahasiswa_table (439.97ms)

C:\xampp\htdocs\laravel01>
```

Gambar: rollback dan jalankan kembali file migrate mahasiswa

Dan berikut hasil dari query DESC mahasiswa:

```
MariaDB [laravel]> DESC mahasiswa;
+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+
| id         | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| nim        | char(8)         | NO   |     | NULL    |                 |
| nama       | varchar(255)    | NO   |     | NULL    |                 |
| tempat_lahir | varchar(255)    | NO   |     | NULL    |                 |
| tanggal_lahir | date           | NO   |     | NULL    |                 |
| fakultas   | varchar(255)    | NO   |     | NULL    |                 |
| jurusan    | varchar(255)    | NO   |     | NULL    |                 |
| ipk        | decimal(3,2)     | NO   |     | NULL    |                 |
| created_at | timestamp       | YES  |     | NULL    |                 |
```

Migration

updated_at timestamp	YES	NULL	
------------------------	-----	------	--

Inilah struktur tabel yang dihasilkan dari migration. Tergantung kebutuhan, kita tinggal menambah kolom yang sesuai berdasarkan daftar method yang ada di tabel sebelumnya.

Laravel juga menyediakan method lanjutan untuk menambah atribut khusus atau *modifier* ke dalam kolom. Misalnya untuk membuat nilai default, men-set kolom agar bisa menerima nilai null, mengatur charset, dll. Penulisan method ini di-*chaining* dari method yang dipakai untuk membuat kolom.

Sebagai contoh, untuk men-set kolom 'nim' agar tidak bisa diinput dengan nilai yang sama, bisa ditulis sebagai:

```
$table->char('nim',8)->unique();
```

Atau agar kolom 'ipk' memiliki nilai default 1.00 bisa ditulis dengan perintah:

```
$table->decimal('ipk',3,2)->default(1.00);
```

Berikut daftar tambahan atribut untuk Migration:

Command	Description
->after('column')	Place the column "after" another column (MySQL)
->autoIncrement()	Set INTEGER columns as auto-increment (primary key)
->charset('utf8')	Specify a character set for the column (MySQL)
->collation('utf8_unicode_ci')	Specify a collation for the column (MySQL/PostgreSQL/SQL Server)
->comment('my comment')	Add a comment to a column (MySQL/PostgreSQL)
->default(\$value)	Specify a "default" value for the column
->first()	Place the column "first" in the table (MySQL)
->nullable(\$value = true)	Allows (by default) NULL values to be inserted into the column
->storedAs(\$expression)	Create a stored generated column (MySQL)
->unsigned()	Set INTEGER columns as UNSIGNED (MySQL)
->useCurrent()	Set TIMESTAMP columns to use CURRENT_TIMESTAMP as default value
->virtualAs(\$expression)	Create a virtual generated column (MySQL)
->generatedAs(\$expression)	Create an identity column with specified sequence options (PostgreSQL)
->always()	Defines the precedence of sequence values over input for an identity column (PostgreSQL)

Tidak semua tambahan perintah ini bisa dipakai di setiap aplikasi database, beberapa ada yang hanya tersedia untuk database MySQL, dan ada juga yang hanya bisa dipakai untuk database PostgreSQL.

Sebagai praktek, saya akan modifikasi ulang file migration untuk tabel mahasiswa:

```
database/migrations/2022_02_01_145155_create_mahasiswa_table.php
```

```

1 public function up()
2 {
3     Schema::create('mahasiswa', function (Blueprint $table) {
4         $table->id();
5         $table->char('nim',8)->unique();
6         $table->string('nama');
7         $table->string('tempat_lahir');
8         $table->date('tanggal_lahir');
9         $table->string('fakultas');
10        $table->string('jurusan');
11        $table->decimal('ipk',3,2)->default(1.00);
12        $table->timestamps();
13    });
14 }
```

Perubahannya ada di baris 5 dan 11, dimana saya menambahkan method `unique()` dan `default()` ke kolom 'nim' dan 'ipk'.

Agar perubahan ini bisa diterapkan ke tabel `mahasiswa`, silahkan rollback, lalu migrate kembali. Dan berikut hasil dari query DESC `mahasiswa`:

```
MariaDB [laravel]> DESC mahasiswa;
```

Field	Type	Null	Key	Default	Extra
<code>id</code>	<code>bigint(20) unsigned</code>	NO	PRI	<code>NULL</code>	<code>auto_increment</code>
<code>nim</code>	<code>char(8)</code>	NO	UNI	<code>NULL</code>	
<code>nama</code>	<code>varchar(255)</code>	NO		<code>NULL</code>	
<code>tempat_lahir</code>	<code>varchar(255)</code>	NO		<code>NULL</code>	
<code>tanggal_lahir</code>	<code>date</code>	NO		<code>NULL</code>	
<code>fakultas</code>	<code>varchar(255)</code>	NO		<code>NULL</code>	
<code>jurusan</code>	<code>varchar(255)</code>	NO		<code>NULL</code>	
<code>ipk</code>	<code>decimal(3,2)</code>	NO		<code>1.00</code>	
<code>created_at</code>	<code>timestamp</code>	YES		<code>NULL</code>	
<code>updated_at</code>	<code>timestamp</code>	YES		<code>NULL</code>	

Hasilnya, kolom `nim` memiliki index `UNI`, yang merupakan singkatan dari `UNIQUE`. Dimana MySQL akan menampilkan pesan error jika kita menginput nomor `nim` yang berulang. Atau dengan kata lain inputan data `nim` harus unik.

Untuk kolom IPK sekarang sudah memiliki nilai default `1.00`, yang bisa dilihat dari kolom `Default`.

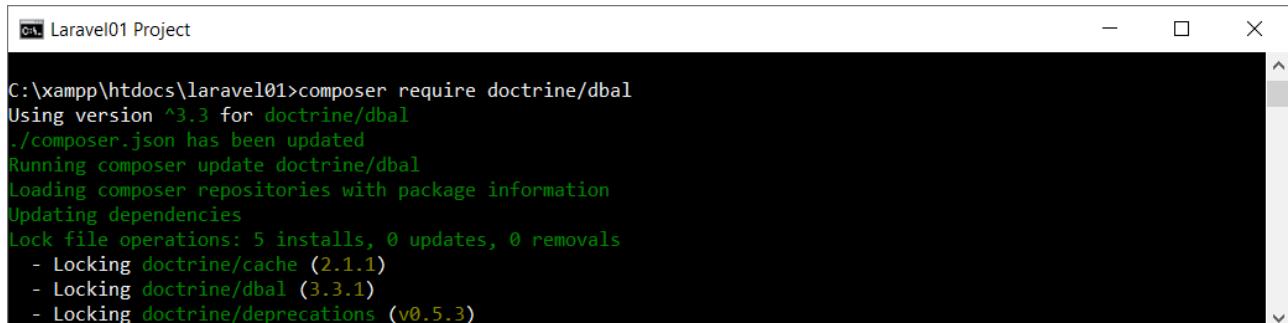
13.7. Alter Table Migration

Migration tidak hanya dipakai untuk membuat dan menghapus tabel, tapi juga untuk modifikasi struktur tabel (menjalankan query ALTER). Ini cukup sering kita lakukan sepanjang pembuatan project.

Misalnya tiba-tiba client memberitahu ada perubahan data yang butuh modifikasi struktur tabel. Sebagai antisipasi, kita bisa membuat file migration baru yang berisi perintah modifikasi tersebut. Jika ternyata ada masalah, dengan 1 perintah rollback, tabel bisa kembali ke bentuk semula.

Agar bisa melakukan modifikasi tabel ke dalam migration, Laravel butuh sebuah library tambahan bernama [Doctrine DBAL](#). Proses instalasi library ini sangat mudah, cukup jalankan perintah berikut di folder laravel:

```
composer require doctrine/dbal
```

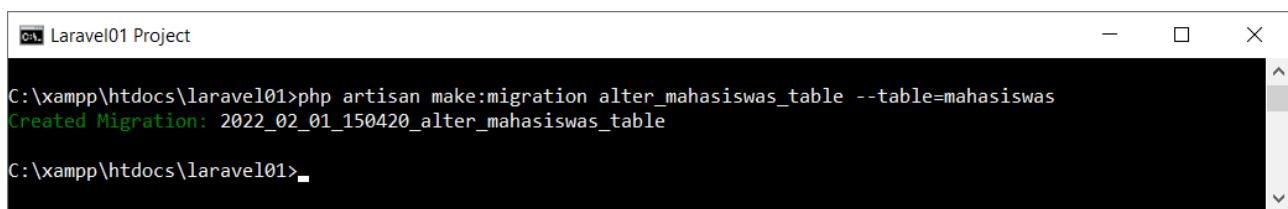


```
C:\xampp\htdocs\laravel01>composer require doctrine/dbal
Using version ^3.3 for doctrine/dbal
./composer.json has been updated
Running composer update doctrine/dbal
Loading composer repositories with package information
Updating dependencies
Lock file operations: 5 installs, 0 updates, 0 removals
- Locking doctrine/cache (2.1.1)
- Locking doctrine/dbal (3.3.1)
- Locking doctrine/deprecations (v0.5.3)
```

Gambar: Instalasi doctrine/dbal library

Berikutnya, saya ingin memodifikasi struktur tabel `mahasiswa`. Silahkan buat file migration baru dengan nama `alter_mahasiswa_table`:

```
php artisan make:migration alter_mahasiswa_table --table=mahasiswa
```



```
C:\xampp\htdocs\laravel01>php artisan make:migration alter_mahasiswa_table --table=mahasiswa
Created Migration: 2022_02_01_150420_alter_mahasiswa_table
C:\xampp\htdocs\laravel01>
```

Gambar: Buat file migration alter_mahasiswa_table

Kemudian buka file migration ini di teks editor. Terlihat method `up()` dan `down()` tidak berisi kode apapun karena Laravel bisa mendeteksi kalau tabel `mahasiswa` sudah ada, sehingga tidak mungkin file migration juga dipakai pembuatan tabel lagi.

Modifikasi file migration sebagai berikut:

Migration

database/migrations/2020_03_04_151623_alter_mahasiswa_table.php

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::table('mahasiswa', function (Blueprint $table) {
17             $table->renameColumn('nama', 'nama_lengkap');
18             $table->text('alamat')->after('tanggal_lahir');
19             $table->dropColumn('ipk');
20         });
21     }
22
23     /**
24      * Reverse the migrations.
25      *
26      * @return void
27      */
28     public function down()
29     {
30         Schema::table('mahasiswa', function (Blueprint $table) {
31             $table->renameColumn('nama_lengkap', 'nama');
32             $table->dropColumn('alamat');
33             $table->decimal('ipk', 3, 2)->default(1.00);
34         });
35     }
36 }
```

Di dalam method `up()` saya membuat 3 buah perintah modifikasi:

- Method `$table->renameColumn('nama', 'nama_lengkap')` dipakai untuk mengubah nama kolom 'nama' menjadi 'nama_lengkap'.
- Method `$table->text('alamat')->after('tanggal_lahir')` dipakai untuk menambah kolom 'alamat' dengan tipe data TEXT, yang posisinya ditempatkan setelah kolom 'tanggal_lahir'.
- Method `$table->dropColumn('ipk')` dipakai untuk menghapus kolom 'ipk'.

Ketiga perubahan di method `up()` ini harus kita balik di method `down()`:

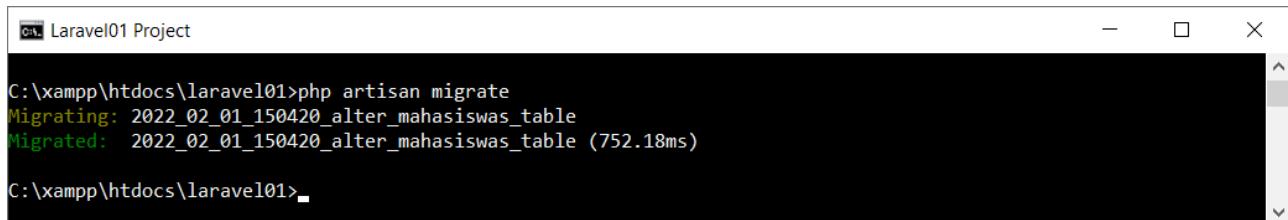
- Method `$table->renameColumn('nama_lengkap', 'nama')` dipakai untuk mengubah kembali nama kolom dari 'nama_lengkap' menjadi 'nama'.

Migration

- Method `$table->dropColumn('alamat')` dipakai untuk menghapus kolom 'alamat'.
- Method `$table->decimal('ipk', 3, 2)->default(1.00)` dipakai untuk membuat kembali kolom 'ipk' dengan tipe data DECIMAL(3,2) dan nilai default 1.00.

Kode program di dalam method `up()` dan `down()` harus berpasangan agar proses *rollback* bisa berlangsung dengan baik.

Sekarang mari kita jalankan perubahan ini dengan perintah `php artisan migrate`:



```
C:\xampp\htdocs\laravel01>php artisan migrate
Migrating: 2022_02_01_150420_alter_mahasiswa_table
Migrated: 2022_02_01_150420_alter_mahasiswa_table (752.18ms)

C:\xampp\htdocs\laravel01>
```

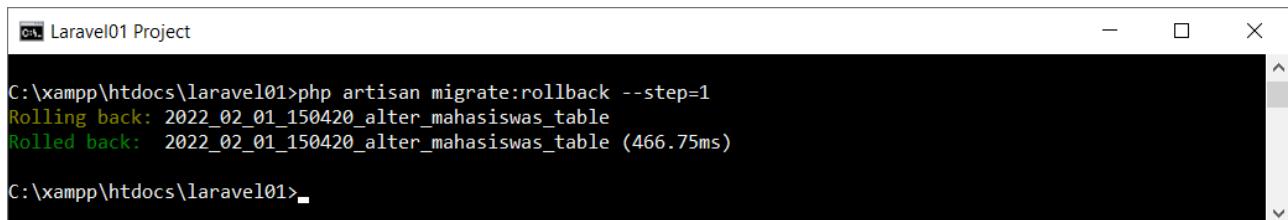
Gambar: Jalankan migration alter_mahasiswa_table

Kemudian cek struktur tabel `mahasiswa` dari MySQL Client:

```
MariaDB [laravel]> DESC mahasiswa;
+-----+-----+-----+-----+-----+-----+
| Field      | Type           | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+-----+
| id         | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| nim        | char(8)          | NO   | UNI | NULL    |               |
| nama_lengkap | varchar(255)     | NO   |     | NULL    |               |
| tempat_lahir | varchar(255)     | NO   |     | NULL    |               |
| tanggal_lahir | date            | NO   |     | NULL    |               |
| alamat      | text             | NO   |     | NULL    |               |
| fakultas    | varchar(255)     | NO   |     | NULL    |               |
| jurusan     | varchar(255)     | NO   |     | NULL    |               |
| created_at  | timestamp        | YES  |     | NULL    |               |
| updated_at  | timestamp        | YES  |     | NULL    |               |
+-----+-----+-----+-----+-----+-----+
```

Sip, tabel `mahasiswa` telah berhasil di modifikasi. Kolom 'nama' sudah berubah jadi 'nama_lengkap', kolom 'alamat' juga telah ditambahkan dan berada setelah kolom 'tanggal_lahir', serta kolom 'ipk' juga tidak lagi ditemukan.

Apabila setelah beberapa saat terjadi masalah atau kita berubah pikiran, perubahan ini bisa di *rollback* dengan `php artisan migrate:rollback --step=1`:



```
C:\xampp\htdocs\laravel01>php artisan migrate:rollback --step=1
Rolling back: 2022_02_01_150420_alter_mahasiswa_table
Rolled back: 2022_02_01_150420_alter_mahasiswa_table (466.75ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Rollback migration alter_mahasiswa_table

Hasilnya, struktur tabel `mahasiswa` akan kembali ke bentuk semula. Kunci dari proses ini, isi method `down()` harus bisa membalik perubahan yang dilakukan oleh method `up()`.

Selain apa yang kita bahas di sini, masih terdapat beberapa fitur migration lain seperti modifikasi `index`, update atribut kolom, serta menambah `key`. Agar materi kita tidak terlalu panjang, jika tertarik anda bisa lanjut baca ke dokumentasi Laravel di [Database: Migrations](#).

Migration merupakan fitur yang sangat berguna. Selain memudahkan kerja tim, migration juga sangat praktis ketika kita ingin men-deploy aplikasi di komputer client atau di web hosting karena tidak perlu repot-repot buat tabel secara manual. Bayangkan jika ada 10 tabel yang harus dibuat satu per satu menggunakan phpMyAdmin.

Alternatif lain untuk membuat tabel secara massal juga bisa menggunakan fitur export-import MySQL (perintah `mysqldump`).

Selain itu, fitur `version control` dari migration juga bisa meminimalisir dampak kesalahan. Jika anda ragu saat mengubah struktur tabel yang mungkin bisa jadi masalah, buatlah sebuah file migration baru. Jika terjadi error atau hal lain, tinggal di rollback ke kondisi semula.

Lanjut, kita akan membahas **DB Facade**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

=====

14. DB Facade (Raw SQL Queries)

Untuk mengelola data tabel, terdapat beberapa cara yang disediakan Laravel, diantaranya: **DB Facade (Raw SQL Queries)**, **Query Builder** dan **Eloquent ORM**. Ketiga materi ini akan kita bahas dalam bab terpisah, yang dimulai dari DB Facade terlebih dahulu.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css**, karena kita akan memakai sedikit kode Bootstrap.

Karena dalam bab migration kita telah membuat beberapa tabel di database **laravel** (hasil dari perintah `php artisan migration`), silahkan hapus tabel-tabel ini. Dengan demikian database **laravel** tidak berisi tabel apapun.

14.1. Pengertian DB Facade (Raw SQL Queries)

DB facade adalah *facade* class bawaan Laravel yang berisi berbagai method untuk menjalankan query SQL.

Apa yang akan kita pelajari kali ini adalah menjalankan *raw SQL*, atau 'perintah query mentah'. Disebut sebagai raw (mentah), karena query langsung ditulis sebagaimana yang biasa diinput ke dalam mysqli extension atau PDO. Query yang dimaksud adalah perintah SQL seperti '`SELECT * FROM mahasiswa`', '`INSERT INTO mahasiswa...`', '`UPDATE mahasiswa SET...`', dst.

Raw query merupakan cara paling dasar dan 'paling tradisional' di dalam Laravel, terutama jika dibandingkan dengan *query builder* dan *eloquent ORM*. Raw query juga sangat familiar karena sudah biasa kita pakai ketika membuat aplikasi PHP tanpa framework. Selain itu untuk query yang kompleks, kadang hanya bisa dijalankan dengan raw query.

Membuat Migration dan Controller

Sebelum membahas materi DB facade lebih lanjut, kita perlu siapkan file migration dan controller sebagai bahan praktek.

Pastikan MySQL Server sudah *running* dan database dengan nama '`laravel`' telah tersedia. Jika terdapat tabel hasil praktek dari bab sebelumnya, silahkan hapus terlebih dahulu (kosongkan database `laravel`).

Untuk migration, jalankan perintah berikut:

```
php artisan make:migration create_mahasiswa_table --create=mahasiswa
```

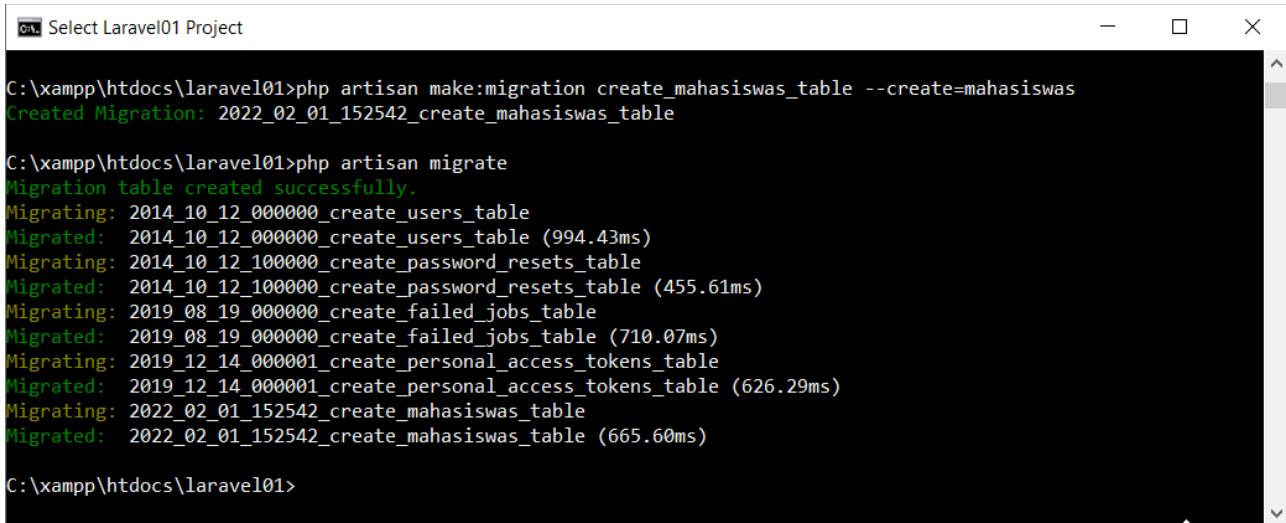
Perintah ini akan membuat file migration untuk tabel **mahasiswa**. Setelah dijalankan, buka file migration di `database\migrations\<timestamp>_create_mahasiswa_table.php`, dan isi method `up()` sebagai berikut:

```
database/migrations/2022_02_01_152542_create_mahasiswa_table.php
```

```
1 <?php
2 // ...
3 // ...
4     public function up()
5     {
6         Schema::create('mahasiswa', function (Blueprint $table) {
7             $table->id();
8             $table->char('nim',8)->unique();
9             $table->string('nama');
10            $table->date('tanggal_lahir');
11            $table->decimal('ipk',3,2)->default(1.00);
12            $table->timestamps();
13        });
14    }
15 // ...
16 // ...
```

Dengan perintah ini, tabel `mahasiswa` akan memiliki 7 kolom, yakni `id`, `nim`, `nama`, `tanggal_lahir`, `ipk`, `created_at` dan `updated_at`. Inilah tabel praktek yang kita pakai.

Kemudian jalankan migration: `php artisan migrate`



```
C:\xampp\htdocs\laravel01>php artisan make:migration create_mahasiswa_table --create=mahasiswa
Created Migration: 2022_02_01_152542_create_mahasiswa_table

C:\xampp\htdocs\laravel01>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (994.43ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (455.61ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (710.07ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (626.29ms)
Migrating: 2022_02_01_152542_create_mahasiswa_table
Migrated: 2022_02_01_152542_create_mahasiswa_table (665.60ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Buat dan jalankan migration

Ketika menjalankan migration, ada beberapa tabel bawaan yang di-generate oleh Laravel, tapi yang akan kita pakai dalam bab ini hanya tabel `mahasiswa` saja.

Lanjut, buat controller bernama **MahasiswaController**:

```
php artisan make:controller MahasiswaController
```

Setelah dijalankan, buka file `app\Http\Controllers\MahasiswaController.php` dan buat sebuah method `index()`:

```
app/Http/Controllers/MahasiswaController.php
```

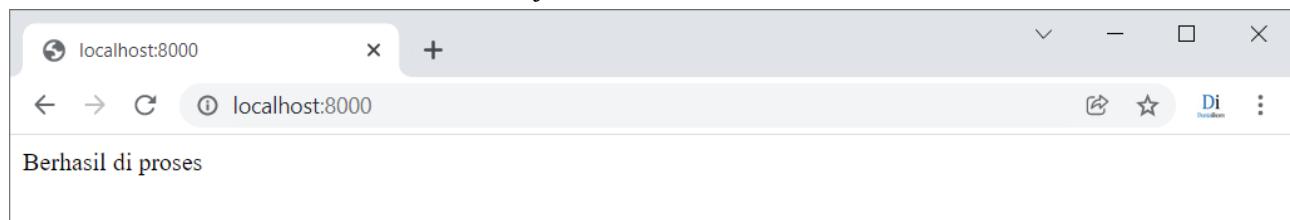
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     public function index(){
10         return "Berhasil di proses";
11     }
12 }
```

Method `index()` ini hanya sebagai *dummy method*, sekedar uji coba apakah controller berhasil diakses atau tidak. Untuk mengakses controller ini, tulis route ke dalam file `routes\web.php`:

```
routes/web.php
```

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/', [MahasiswaController::class, 'index']);
```

Pastikan php artisan serve sudah berjalan, dan akses alamat <http://localhost:8000>



Gambar: Controller berhasil di proses

Sip, method index() milik MahasiswaController sudah berhasil di akses.

Kembali ke file routes/web.php, tambah beberapa route lain:

routes/web.php

```

1 ...
2 Route::get('/insert-sql',      [MahasiswaController::class, 'insertSql']);
3 Route::get('/insert-timestamp', [MahasiswaController::class, 'insertTimestamp']);
4 Route::get('/insert-prepared',  [MahasiswaController::class, 'insertPrepared']);
5 Route::get('/insert-binding',   [MahasiswaController::class, 'insertBinding']);
6 Route::get('/update',          [MahasiswaController::class, 'update']);
7 Route::get('/delete',          [MahasiswaController::class, 'delete']);
8 Route::get('/select',          [MahasiswaController::class, 'select']);
9 Route::get('/select-tampil',   [MahasiswaController::class, 'selectTampil']);
10 Route::get('/select-view',    [MahasiswaController::class, 'selectView']);
11 Route::get('/select-where',   [MahasiswaController::class, 'selectWhere']);
12 Route::get('/statement',     [MahasiswaController::class, 'statement']);

```

Inilah daftar route dan method yang akan kita bahas sepanjang bab ini.

14.2. Menginput Data (DB::insert)

Perintah raw query pertama yang akan kita pelajari adalah proses input data menggunakan DB::insert() facade. Langsung saja masuk ke contoh kode program:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\DB;
7
8 class MahasiswaController extends Controller
9 {
10     public function index(){
11         return "Berhasil di proses";
12     }
13
14     public function insertSql(){

```

```

15     $result = DB::insert("INSERT INTO mahasiswa(nim,nama,tanggal_lahir,ipk)
16                         VALUES ('19003036','Sari Citra Lestari','2001-12-31',3.5)");
17     dump($result);
18 }
19 }
```

Dalam controller ini kita memerlukan **DB facade**, sehingga butuh perintah untuk meng-import-nya di baris 6, yakni kode `use Illuminate\Support\Facades\DB;`

Di baris 14 – 19 saya membuat method `insertSql()`. Method ini berisi pemanggilan perintah `DB::insert()` di baris 15 yang dipakai untuk menjalankan query INSERT SQL. Query tersebut di tulis sebagai argument, yakni:

```
"INSERT INTO mahasiswa(nim,nama,tanggal_lahir,ipk) VALUES ('19003036','Sari Citra Lestari','2001-12-31',3.5)"
```

Perintah SQL ini akan menambah 1 data baru ke dalam tabel `mahasiswa`. Hasil dari `DB::insert()` selanjutnya di tampung ke dalam variabel `$result` yang kemudian di `dump()`.

Untuk menjalankan method `insertSql()`, akses alamat `localhost:8000/insert-sql`:



Gambar: Tampilan halaman `http://localhost:8000/insert-sql`

Hasil ini memperlihatkan bahwa method `DB::insert()` akan mengembalikan nilai `true` jika query berhasil dijalankan. Sebagai pembuktian, silahkan buka phpMyAdmin atau cmd MySQL client, kemudian lihat isi tabel `mahasiswa`:

```

MySQL Folder cmd - mysql -u root
c:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 38
Server version: 10.4.22-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE laravel;
Database changed
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+
| id | nim   | nama      | tanggal_lahir | ipk   | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2001-12-31    | 3.50 | NULL       | NULL       |
+----+-----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]> 
```

Gambar: Isi tabel `mahasiswa` setelah proses `INSERT`

Terlihat data mahasiswa 'Sari Citra Lestari' sudah berhasil diinput ke dalam tabel `mahasiswas`. Namun perhatikan kolom `created_at` dan `update_at` berisi nilai `NULL`, ini karena pada saat menjalankan query `INSERT`, tidak ada perintah untuk menginput nilai ke dalam kedua kolom ini.

Nantinya jika kita sudah sampai ke bab **Eloquent ORM**, kolom `created_at` dan `update_at` bisa diisi Laravel secara otomatis.

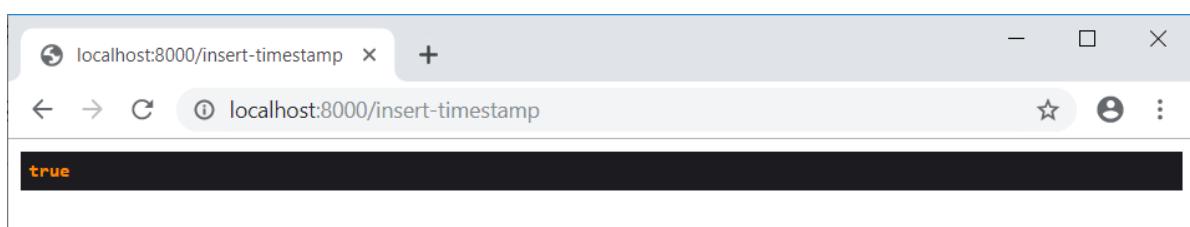
Saya akan buat query `INSERT` kedua yang menyertakan nilai untuk kolom `created_at` dan `update_at`:

app/Http/Controllers/MahasiswaController.php

```

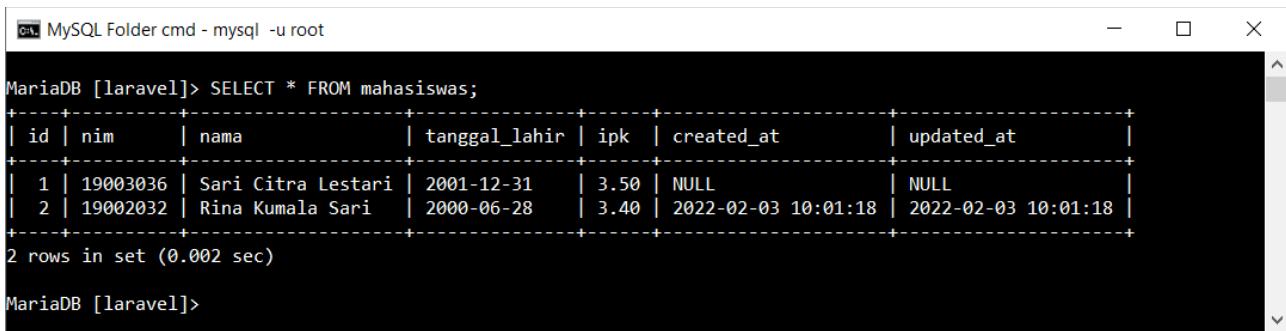
1 <?php
2 //...
3 //...
4 class MahasiswaController extends Controller
5 {
6     public function insertTimestamp(){
7         $result = DB::insert(
8             "INSERT INTO mahasiswas
9                 (nim, nama, tanggal_lahir, ipk, created_at, updated_at)
10            VALUES
11              ('19002032','Rina Kumala Sari','2000-06-28',3.4,now(),now())
12        );
13        dump($result);
14    }
15 }
```

Sekarang function `now()` bawaan MySQL akan mengisi nilai kolom `created_at` dan `update_at`. Jalankan method di atas dengan membuka halaman `localhost:8000/insert-timestamp`.



Gambar: Tampilan halaman `http://localhost:8000/insert-timestamp`

Dan berikut isi tabel `mahasiswas`:



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | NULL | NULL |
| 2 | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.40 | 2022-02-03 10:01:18 | 2022-02-03 10:01:18 |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi tabel mahasiswa

Sip, kolom `created_at` dan `update_at` untuk mahasiswa Rina Kumala Sari sudah berisi tanggal timestamp.

Membuat Prepared Statement

Apa yang baru saja kita lakukan adalah menjalankan query "apa adanya". Supaya lebih aman, query ini lebih baik ditulis dengan teknik **prepared statement**, dimana terdapat pemisahan antara perintah SQL dengan data yang akan diinput. Terlebih data ini nantinya banyak berasal dari inputan form yang sangat rawan diisi nilai-nilai aneh, termasuk resiko dari SQL Injection.

Untuk membuat prepared statement, caranya cukup mudah. Kita tetap tulis query seperti biasa ke dalam method `DB::insert()`, tapi kali ini mengganti nilai data menjadi *placeholder* dengan karakter tanda tanya " ? ". Kemudian input sebuah array di argument kedua method `DB::insert()` untuk menggantikan *placeholder* tadi. Berikut contoh penulisannya:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 //...
3 //...
4 class MahasiswaController extends Controller
5 {
6     public function insertPrepared(){
7         $result = DB::insert(
8             'INSERT INTO mahasiswa
9                 (nim, nama, tanggal_lahir, ipk, created_at, updated_at)
10                VALUES (?, ?, ?, ?, ?, ?)',
11                ['18012012', 'James Situmorang', '1999-04-02', 2.7, now(), now()]
12        );
13        dump($result);
14    }
15 }
```

Jika anda sudah pernah mempelajari prepared statement (misalnya dari buku **PHP Uncover** dan **OOP PHP Uncover**), tentu sudah tidak asing dengan penulisan seperti ini. Setiap tanda tanya akan berpasangan dengan data di dalam array. Jalankan method `insertPrepared()` dengan mengakses alamat: `localhost:8000/insert-prepared`.

Selain menggunakan placeholder berupa tanda tanya, method `DB::insert()` juga bisa memakai teknik **named parameter** seperti contoh berikut:

app/Http/Controllers/MahasiswaController.php

```

1  <?php
2  //...
3  //...
4  class MahasiswaController extends Controller
5  {
6      //...
7      //...
8
9      public function insertBinding(){
10         $result = DB::insert(
11             'INSERT INTO mahasiswas
12             (nim, nama, tanggal_lahir, ipk, created_at, updated_at)
13             VALUES (:nim,:nama,:tgl,:ipk,:created,:updated)',
14             [
15                 'nim' => '19005011',
16                 'nama' => 'Riana Putria',
17                 'tgl' => '2000-11-23',
18                 'ipk' => 2.7,
19                 'created' => now(),
20                 'updated' => now(),
21             ]
22         );
23         dump($result);
24     }
25 }
```

Di baris 13 saya menulis query `VALUES` berbentuk *named parameter* yang akan berpasangan dengan associative array di baris 15 – 20. Kembali, materi named parameter ini juga sudah di bahas detail dalam buku **OOP PHP Uncover** (bab tentang PDO).

Jalankan method `insertNamedBinding()` dengan mengakses alamat `localhost:8000/insert-binding`.

Sampai di sini kita sudah menjalankan 4 kali query `INSERT`. Berikut isi dari tabel `mahasiswas`:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.50	NULL	NULL
2	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 10:01:18	2022-02-03 10:01:18
3	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 03:04:01	2022-02-03 03:04:01
4	19005011	Riana Putria	2000-11-23	2.70	2022-02-03 03:05:11	2022-02-03 03:05:11

Gambar: Isi tabel mahasiswas setelah menjalankan 4 kali `DB::insert()`

14.3. Mengupdate Data (DB::update)

Untuk proses update data, DB facade menyediakan method `DB::update()`. Cara penggunaan method ini sama seperti `DB::insert()`, dimana kita menulis query SQL sebagai argument pertama, kemudian sebuah array nilai di argument kedua jika menggunakan prepared statement. Berikut contoh penggunaannya:

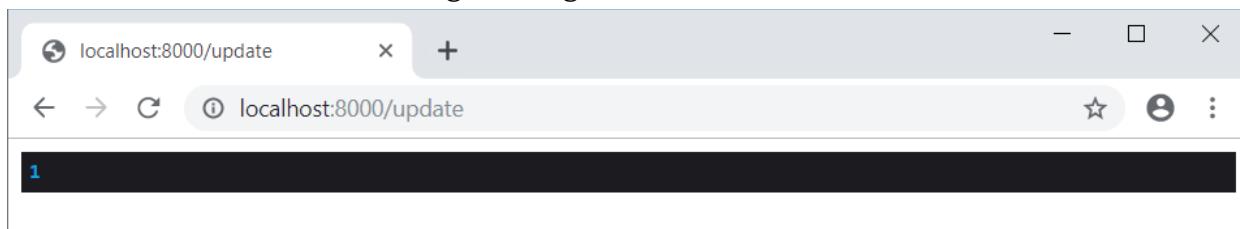
app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     //...
6     //...
7     public function update(){
8         $result = DB::update(
9             'UPDATE mahasiswas SET created_at = now(), updated_at = now()
10            WHERE nim = ?', ['19003036']);
11         dump($result);
12     }
13 }
```

Di sini saya menjalankan query UPDATE untuk mengubah nilai kolom `created_at` dan `updated_at` dari siswa yang memiliki nim '19003036'.

Jalankan method `update()` ini dengan mengakses halaman `localhost:8000/update`.



Gambar: Tampilan halaman `http://localhost:8000/update`

Angka 1 memperlihatkan bahwa ada 1 data yang berhasil di update, kurang lebih sama dengan hasil fungsi `mysqli_affected_rows()` di PHP native. Dan berikut isi tabel mahasiswa:

MariaDB [laravel]> SELECT * FROM mahasiswas;						
id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.50	2022-02-03 10:05:55	2022-02-03 10:05:55
2	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 10:01:18	2022-02-03 10:01:18
3	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 03:04:01	2022-02-03 03:04:01
4	19005011	Riana Putria	2000-11-23	2.70	2022-02-03 03:05:11	2022-02-03 03:05:11

Gambar: Isi tabel mahasiswa setelah menjalankan `DB::update()`

14.4. Menghapus Data (DB::delete)

Untuk menghapus data tabel, DB facade menyediakan method `DB::delete()`. Penggunaannya juga sama seperti method `DB::insert()` dan `DB::update()`. Yang berbeda hanya di penulisan perintah SQL saja:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     //...
6     //...
7
8     public function delete(){
9         $result = DB::delete(
10            'DELETE FROM mahasiswas WHERE nama = ?' , ['James Situmorang']
11        );
12        dump($result);
13    }
14 }
```

Method `DB::delete()` diatas saya pakai untuk menghapus data dari tabel mahasiswa dengan kondisi dimana `nama = 'James Situmorang'`. Jalankan dengan mengakses `localhost:8000/delete`.

Hasilnya, tabel mahasiswa hanya tinggal 3 baris:

```

MySQL [laravel] > SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+-----+
| id | nim   | nama           | tanggal_lahir | ipk  | created_at      | updated_at      |
+----+-----+-----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | 2022-02-03 10:05:55 | 2022-02-03 10:05:55 |
| 2  | 19002032 | Rina Kumala Sari  | 2000-06-28 | 3.40 | 2022-02-03 10:01:18 | 2022-02-03 10:01:18 |
| 4  | 19005011 | Riana Putria     | 2000-11-23 | 2.70 | 2022-02-03 03:05:11 | 2022-02-03 03:05:11 |
+----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.002 sec)

```

Gambar: Isi tabel mahasiswas setelah menjalankan `DB::delete()`

14.5. Menampilkan Data (DB::select)

Seperti yang bisa di tebak, untuk menampilkan data DB facade menyediakan method `DB::select()`. Cara penggunaannya juga sama seperti method sebelumnya:

app/Http/Controllers/MahasiswaController.php

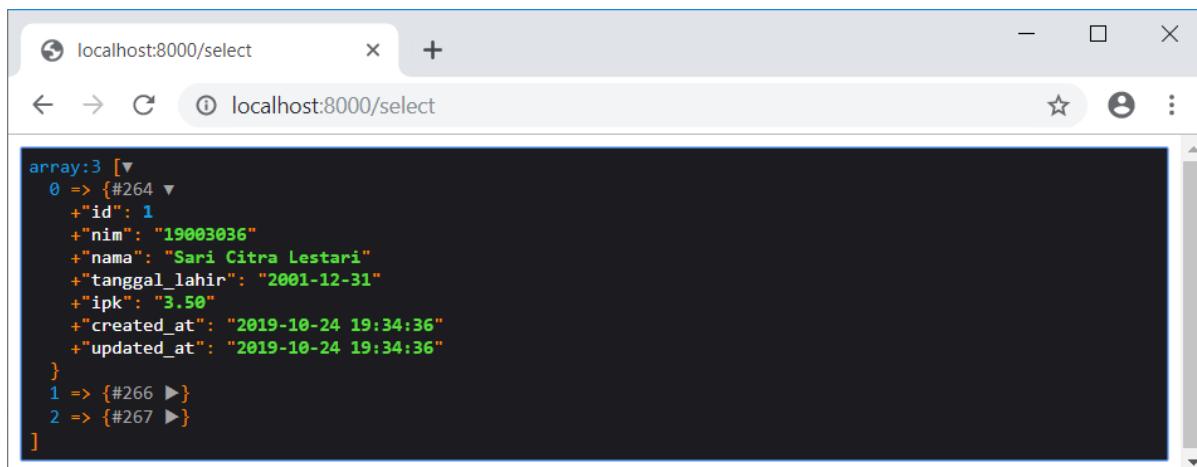
```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
```

```

5 //...
6 //...
7
8     public function select(){
9         $result = DB::select('SELECT * FROM mahasiswa');
10        dump($result);
11    }
12 }
```

Di sini saya menulis query 'SELECT * FROM mahasiswa' yang hasilnya disimpan ke dalam variabel \$result untuk kemudian di dump(). Untuk menjalankan method ini, silahkan akses localhost:8000/select:



Gambar: Tampilan halaman http://localhost:8000/select

Terlihat hasil dari method DB::select() adalah sebuah array. Dimana setiap element array berbentuk object yang berisi data mahasiswa. Karena di tabel mahasiswa terdiri dari 3 baris, maka dalam array \$hasil juga ada 3 object. Struktur seperti ini sudah pernah kita pelajari ketika membahas bab collection.

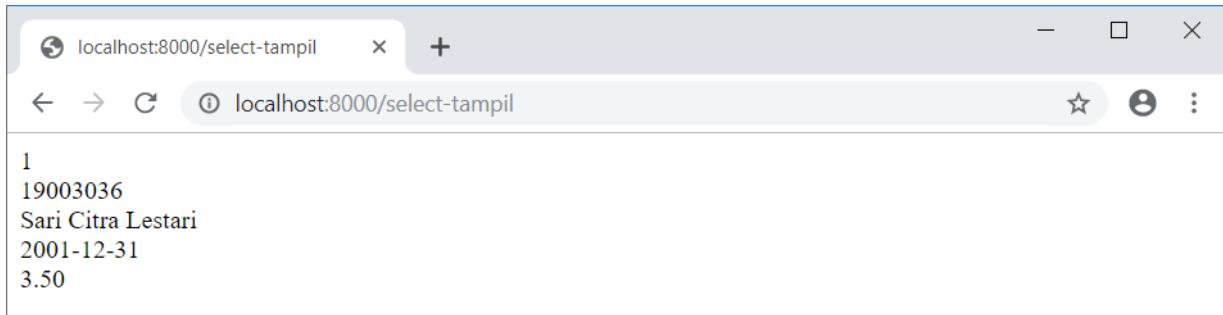
Berikut cara untuk menampilkan semua data dari object pertama:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     //...
6     //...
7
8     public function selectTampil(){
9         $result = DB::select('SELECT * FROM mahasiswa');
10
11        echo($result[0]->id). '<br>';
12        echo($result[0]->nim). '<br>';
13        echo($result[0]->nama). '<br>';
14        echo($result[0]->tanggal_lahir). '<br>';
15        echo($result[0]->ipk);
```

```
16     }
17 }
```



Gambar: Tampilan halaman http://localhost:8000/select-tampil

Dengan latihan yang sudah kita coba di bab collection, saya yakin anda sudah bisa memahami maksud dari perintah `echo($result[0]->id)`. Yup, perintah ini dipakai untuk mengakses property `id` milik object pertama yang ada di dalam array `$result`.

Karena kita sudah bisa mengakses semua isi tabel mahasiswa, saya ingin menampilkan data ini ke dalam view. Berikut method yang diperlukan:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     //...
6     public function selectView(){
7         $result = DB::select('SELECT * FROM mahasiswa');
8         return view('tampil-mahasiswa',['mahasiswa' => $result]);
9     }
10 }
11 }
```

Perintah di baris 8 akan mengirim data `['mahasiswa' => $result]` ke dalam view `tampil-mahasiswa.blade.php`. Dengan demikian, di dalam view kita bisa mengakses semua array yang tersimpan di dalam `$result` melalui variabel `$mahasiswa`.

Berikutnya, silahkan buat file `tampil-mahasiswa.blade.php` di folder `resources\views`. Kemudian isi dengan kode berikut:

resources/views/tampil-mahasiswa.blade.php

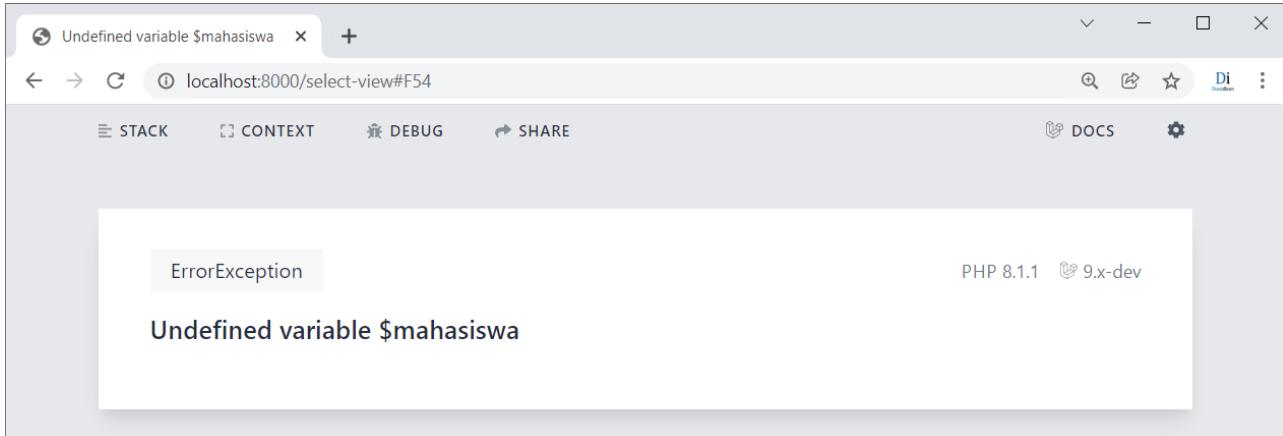
```
1 <?php dump($mahasiswa) ?>
```

Hanya butuh 1 baris saja, yakni `<?php dump($mahasiswa) ?>`. Tujuannya untuk memastikan bahwa data memang sudah sampai di view atau tidak.

Teknik seperti ini sangat sangat disarankan, atau boleh dibilang **WAJIB** dilakukan karena sering kejadian kita menghabiskan waktu berjam-jam mencari sumber error yang ternyata

disebabkan data dari controller belum sampai ke view.

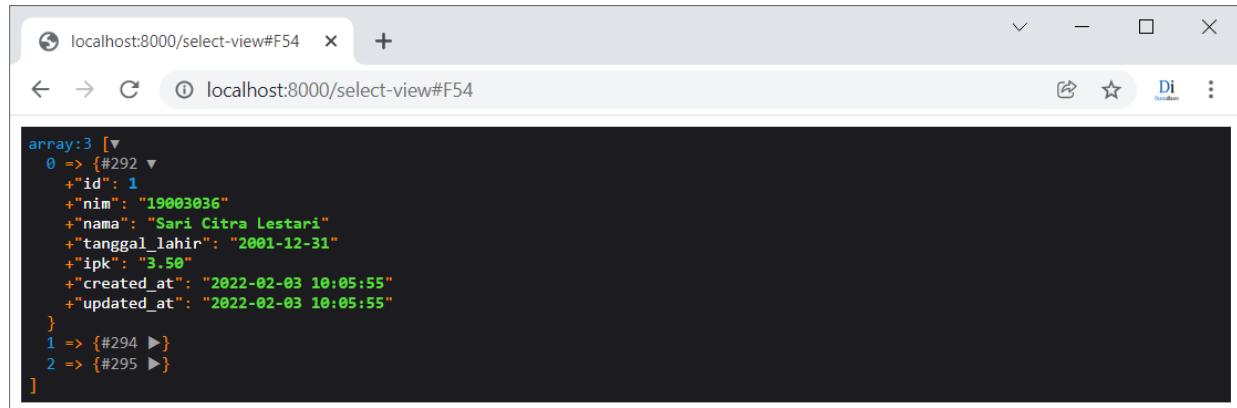
Jadi sebelum menulis kode program lain, pastikan data itu sudah bisa diakses dan isinya sesuai dengan keinginan. Mari coba akses dari `localhost:8000/select-view`:



Gambar: Error di variabel \$mahasiswa

Ternyata memang terjadi error karena saya salah tulis variabel. Seharusnya yang diakses adalah `$mahasiswa`, bukan `$mahasiswa`.

Silahkan perbaiki sumber error dengan mengganti `<?php dump($mahasiswa) ?>` menjadi `<?php dump($mahasiswa) ?>` di file view, kemudian refresh halaman:



Gambar: Data tabel mahasiswa sudah bisa diakses dari view

Sip, data array tabel `mahasiswa` sudah berhasil diakses. Selanjutnya tinggal membuat tampilan design dengan HTML dan sedikit kode Bootstrap:

`resources/views/tampil-mahasiswa.blade.php`

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <meta http-equiv="X-UA-Compatible" content="ie=edge">
7    <link href="{{ asset('/css/bootstrap.min.css') }}" rel="stylesheet">
8    <title>Data Mahasiswa</title>

```

```

9  </head>
10 <body>
11
12 <div class="container text-center p-4">
13   <h1 class="mb-3">Data Mahasiswa</h1>
14   <div class="row">
15     <div class="m-auto">
16       <ol class="list-group">
17         @forelse ($mahasiswas as $mahasiswa)
18           <li class="list-group-item">
19             {{$mahasiswa->nama}} ( {{$mahasiswa->nim}} ), 
20             Tanggal Lahir: {{$mahasiswa->tanggal_lahir}}, 
21             IPK: {{$mahasiswa->ipk}}
22         </li>
23         @empty
24           <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
25         @endforelse
26       </ol>
27     </div>
28   </div>
29 </div>
30
31 </body>
32 </html>

```



Gambar: Tampilan view untuk semua data tabel mahasiswas

Mayoritas kode ini merupakan perintah HTML dan sedikit class Bootstrap.

Proses menampilkan isi tabel berada di baris 17 – 25. Di sini saya menggunakan perintah `@forelse ($mahasiswas as $mahasiswa)` untuk melakukan perulangan. Dalam setiap perulangan, tinggal mengakses `$mahasiswa->nama`, `$mahasiswa->nim`, `$mahasiswa->tanggal_lahir`, dan `$mahasiswa->ipk`.

Pemilihan nama `$mahasiswas` dan `$mahasiswa` membuat kode program kita menjadi lebih mudah dibaca. Jika itu adalah **mahasiswas** (jamak) maka isinya berupa array yang terdiri dari banyak mahasiswa, sedangkan jika itu adalah **mahasiswa** (tunggal) maka isinya berupa data dari 1 mahasiswa saja.

View `tampil-mahasiswa.blade.php` ini bisa dipakai untuk menampilkan data mahasiswa apa saja, selama variabel `$mahasiswa`s yang dikirim berbentuk array dari object mahasiswa, yakni hasil dari `DB::select()`.

Sebagai contoh, saya ingin membuat method baru untuk menampilkan data mahasiswa yang memiliki IPK di atas 3 dan di urutkan berdasarkan nama:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     //...
6     public function selectWhere(){
7         $result = DB::select(
8             'SELECT * FROM mahasiswa WHERE ipk > ? ORDER BY nama ASC ',[3]
9         );
10        return view('tampil-mahasiswa',[ 'mahasiswa' =>$result]);
11    }
12 }
```

Kali ini query yang dijalankan adalah:

```
SELECT * FROM mahasiswa WHERE ipk > 3 ORDER BY nama ASC'
```

Sama seperti method `selectView()`, method `selectWhere()` ini juga meneruskan data `$mahasiswa`s ke view `tampil-mahasiswa`. Akses alamat <http://localhost:8000/select-where> untuk melihat hasilnya:



Gambar: Tampilan view untuk mahasiswa dengan IPK > 3

Karena pembatasan query, sekarang hanya mahasiswa dengan IPK di atas 3 saja yang tampil. Jika anda ingin hasil yang lain, tinggal menyesuaikan perintah query saja.

14.6. Menjalankan Query (DB::statement)

Method raw query yang kita bahas sejauh ini sudah mencakup semua keperluan **CRUD**, yakni

DB::insert() untuk proses **create**, DB::select() untuk proses **read**, DB::update() untuk proses **update**, dan DB::delete() untuk proses **delete**.

Selain itu, DB facade juga menyediakan method DB::statement() sebagai method 'generik' untuk menjalankan query lain yang tidak termasuk 4 kategori CRUD tadi. Cara penggunaannya juga sama, kita tinggal menulis query SQL sebagai argument ke dalam method ini.

Sebagai contoh, saya ingin menjalankan query TRUNCATE mahasiswa untuk menghapus semua data di tabel mahasiswa:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     //...
6     public function statement(){
7         $result = DB::statement('TRUNCATE mahasiswa');
8         return('Tabel mahasiswa sudah dikosongkan');
9     }
10 }
```

Jalankan method ini dengan mengakses localhost:8000/statement:



Gambar: Isi tabel mahasiswa sudah dikosongkan

Hasilnya, tabel mahasiswa kembali kosong. Method DB::statement() ini juga bisa dipakai untuk menjalankan query-query lain.

Dalam bab ini kita telah membahas 5 method milik **DB facade** yang bisa dipakai untuk menjalankan raw query. Raw query tersebut juga bisa ditulis dalam bentuk *prepared statement* agar lebih aman dari SQL Injection.

Bagi yang baru pertama kali menggunakan framework, raw query terasa mudah dipahami karena kita sudah familiar dengan perintah query SQL biasa. Namun Laravel menyediakan metode lain yang lebih praktis, diantaranya **Query Builder** yang akan kita bahas berikutnya.

15. Query Builder

Menjalankan raw query seperti yang kita bahas dalam bab sebelumnya tidak umum dilakukan dalam sebuah framework PHP. Biasanya, framework menyediakan semacam interface khusus yang penulisannya lebih rapi, bahkan kita tidak perlu menulis query SQL sama sekali. Salah satu interface tersebut adalah **Query Builder**.

15.1. Pengertian Query Builder

Query builder adalah interface khusus yang disediakan Laravel untuk mengakses database. Berbeda dengan raw query dimana kita menulis langsung perintah query SQL, di dalam query builder perintah SQL ini diakses menggunakan method. Artinya, kita tidak menulis langsung perintah SQL, tapi hanya memanggil method-method saja.

Keunggulan dari cara ini adalah kode program kita menjadi lebih rapi dan lebih mudah dibaca. Karena tidak menulis query secara langsung, program kita tidak terikat ke satu jenis database saja. Artinya jika ingin beralih dari database MySQL ke SQLite, tidak akan banyak kendala.

Namun kelemahan dari query builder, kita terpaksa harus mempelajari nama-nama method yang ada.

Jika anda pernah membahas buku [OOP PHP Uncover](#), di sana ada satu bab khusus tentang pembuatan 'plugin' **DB class**. DB class tersebut terinspirasi dari query builder seperti ini, sehingga akan terasa sangat mirip.

Membuat Migration dan Controller

Sebagai bahan praktek, saya akan kembali memakai tabel `mahasiswa` dan controller `MahasiswaController` seperti yang gunakan dalam bab **DB facade**. Jika anda mengikuti semua praktek dari bab tersebut, maka tabel `mahasiswa` sudah kembali kosong. Jika belum, silahkan jalankan query `TRUNCATE mahasiswa`.

Untuk route, tulis ulang dengan kode berikut:

```
routes/web.php  
1 <?php  
2  
3 use Illuminate\Support\Facades\Route;
```

```

4  use App\Http\Controllers\MahasiswaController;
5
6  Route::get('/insert', [MahasiswaController::class, 'insert']);
7  Route::get('/insert-banyak', [MahasiswaController::class, 'insertBanyak']);
8  Route::get('/update', [MahasiswaController::class, 'update']);
9  Route::get('/update-where', [MahasiswaController::class, 'updateWhere']);
10 Route::get('/update-or-insert', [MahasiswaController::class, 'updateOrInsert']);
11 Route::get('/delete', [MahasiswaController::class, 'delete']);
12 Route::get('/get', [MahasiswaController::class, 'get']);
13 Route::get('/get-tampil', [MahasiswaController::class, 'getTampil']);
14 Route::get('/get-view', [MahasiswaController::class, 'getView']);
15 Route::get('/get-where', [MahasiswaController::class, 'getWhere']);
16 Route::get('/select', [MahasiswaController::class, 'select']);
17 Route::get('/take', [MahasiswaController::class, 'take']);
18 Route::get('/first', [MahasiswaController::class, 'first']);
19 Route::get('/find', [MahasiswaController::class, 'find']);
20 Route::get('/raw', [MahasiswaController::class, 'raw']);

```

Inilah daftar route yang akan kita bahas sepanjang bab nantinya. Saya memakai penamaan **kebab case** untuk nama route yang diambil dari nama method. Sebagai contoh, jika nama method controller yang diakses adalah `getWhere()`, maka routenya menjadi `get-where`. Jika method yang akan dijalankan adalah `insertBanyak()`, maka routenya menjadi `insert-banyak`, dst.

Untuk `MahasiswaController`, silahkan hapus semua kode yang ada dan sisakan kode berikut:

`app/Http/Controllers/MahasiswaController.php`

```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB;
7
8  class MahasiswaController extends Controller
9  {
10     // kode program akan kita tulis disini
11 }

```

Yang perlu menjadi perhatian adalah, query builder juga butuh DB class, sehingga kita tetap harus menulis perintah import `use Illuminate\Support\Facades\DB` seperti di baris 6.

Alur pembahasan yang akan saya pakai juga sama seperti di bab DB facade, yang dimulai dari proses input data, update data, delete, dan terakhir baru masuk ke cara menampilkan data.

15.2. Menginput Data

Query builder menyediakan method `insert()` untuk proses input data ke dalam tabel. Method ini butuh sebuah argument dalam bentuk associative array dengan key berupa nama kolom

tabel. Berikut contoh penggunaannya:

app/Http/Controllers/MahasiswaController.php

```

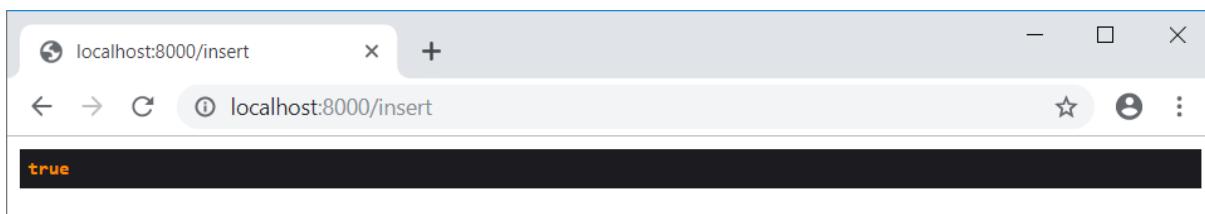
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB;
7
8  class MahasiswaController extends Controller
9  {
10     public function insert(){
11         $result = DB::table('mahasiswas')->insert(
12             [
13                 'nim' => '19003036',
14                 'nama' => 'Sari Citra Lestari',
15                 'tanggal_lahir' => '2001-12-31',
16                 'ipk' => 3.5,
17                 'created_at' => now(),
18                 'updated_at' => now(),
19             ]
20         );
21
22         dump($result);
23     }
24 }
```

Hampir semua perintah query builder diawali dengan memilih tabel yang akan dipakai. Caranya, jalankan method `DB::table(<nama_tabel>)`, baru kemudian diikuti dengan nama method.

Di baris 11, saya menjalankan method `DB::table('mahasiswas')->insert()`. Teknik *method chaining* inilah yang menjadi cara ciri khas dari query builder, artinya method `insert()` dipakai untuk mengakses `DB::table('mahasiswas')`.

Sebagai argument dari method `insert()`, saya mengisinya dengan sebuah associative array yang terdiri dari pasangan nama kolom tabel sebagai key, dan nilai yang ingin diisi sebagai value. Hasil pemanggilan method ini disimpan ke dalam variabel / property `$result` untuk kemudian di `dump()`.

Jalankan method `insert()` dengan mengakses alamat `localhost:8000/insert`:



Gambar: Hasil tampilan halaman `http://localhost:8000/insert`

Sama seperti di DB facade, hasil dari query `insert()` juga sebuah nilai boolean `true`. Silahkan buka phpMyAdmin atau cmd MySQL client untuk melihat data ini:

```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | 2022-02-03 03:34:33 | 2022-02-03 03:34:33 |
+----+-----+-----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi tabel mahasiswa

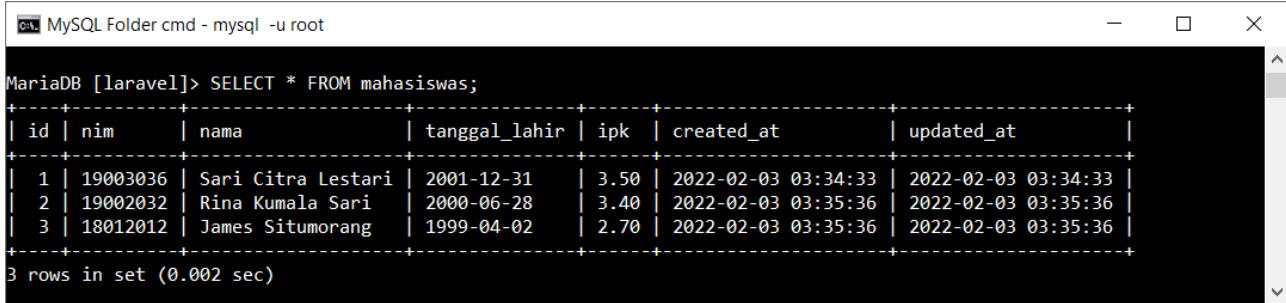
Method `insert()` juga bisa melakukan banyak input sekaligus. Caranya, isi argument dengan nested array atau array 2 dimensi, seperti contoh berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function insertBanyak(){
6         $result = DB::table('mahasiswa')->insert(
7             [
8                 [
9                     'nim' => '19002032',
10                    'nama' => 'Rina Kumala Sari',
11                    'tanggal_lahir' => '2000-06-28',
12                    'ipk' => 3.4,
13                    'created_at' => now(),
14                    'updated_at' => now(),
15                ],
16                [
17                    'nim' => '18012012',
18                    'nama' => 'James Situmorang',
19                    'tanggal_lahir' => '1999-04-02',
20                    'ipk' => 2.7,
21                    'created_at' => now(),
22                    'updated_at' => now(),
23                ],
24            ],
25        );
26    }
27
28    dump($result);
29 }
30 }
```

Di sini saya membuat 2 buah array mahasiswa yang ditulis di dalam tanda kurung siku, artinya setiap array mahasiswa ini adalah element dari array lain. Jika kita butuh menambah data mahasiswa ketiga, tinggal ditambah sebagai array baru.

Jalankan method `insertBanyak()` dari halaman `localhost:8000/insert-banyak` dan berikut isi tabel `mahasiswa`:



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | 2022-02-03 03:34:33 | 2022-02-03 03:34:33 |
| 2 | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.40 | 2022-02-03 03:35:36 | 2022-02-03 03:35:36 |
| 3 | 18012012 | James Situmorang | 1999-04-02 | 2.70 | 2022-02-03 03:35:36 | 2022-02-03 03:35:36 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa

15.3. Mengupdate Data

Untuk proses update data tabel, query builder menyediakan method `update`. Method ini butuh sebuah argument berbentuk array yang berisi data yang akan di update. Namun untuk proses update, kita butuh bantuan dari method `where()` untuk menentukan kondisi yang dicari.

Sebagai contoh, saya ingin mengupdate data tanggal lahir dan nilai ipk dari mahasiswa bernama 'Sari Citra Lestari', berikut kode yang dibutuhkan:

`app/Http/Controllers/MahasiswaController.php`

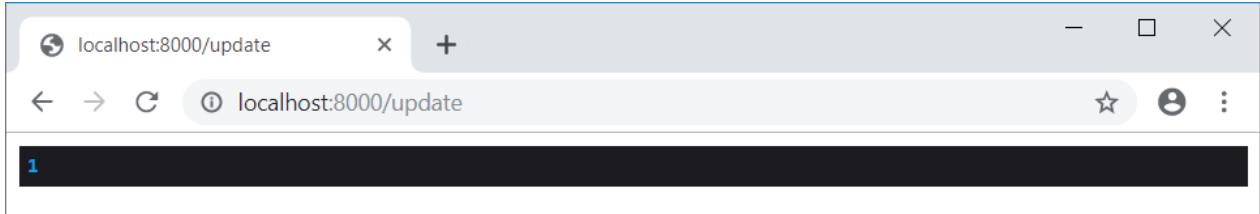
```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function update(){
6         $result = DB::table('mahasiswa')
7             ->where('nama', 'Sari Citra Lestari')
8             ->update(
9                 [
10                     'tanggal_lahir' => '2002-01-01',
11                     'ipk' => 3.19,
12                     'updated_at' => now(),
13                 ]
14             );
15
16         dump($result);
17     }
18 }
```

Di baris 6, diawali dengan pemanggilan method `DB::table('mahasiswa')`, yang berguna untuk menentukan tabel yang akan kita pakai. Kemudian disambung dengan method `where('nama', 'Sari Citra Lestari')` di baris 7, inilah kondisi yang dipakai untuk mencari baris yang ingin di update. Terakhir baru terdapat pemanggilan method `update()` di baris 8.

Argument dari method `update()` ini berbentuk associative array dengan data yang ingin di

update. Terdapat 3 kolom tabel yang ingin saya ubah, yakni `tanggal_lahir`, `ipk` dan `updated_at`.

Jalankan kode program ini dengan mengakses `localhost:8000/update`:



Gambar: Hasil tampilan halaman `http://localhost:8000/update`

```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim      | nama          | tanggal_lahir | ipk    | created_at        | updated_at      |
+----+-----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2002-01-01   | 3.19   | 2022-02-03 03:34:33 | 2022-02-03 03:36:27 |
| 2  | 19002032 | Rina Kumala Sari  | 2000-06-28   | 3.40   | 2022-02-03 03:35:36 | 2022-02-03 03:35:36 |
| 3  | 18012012 | James Situmorang | 1999-04-02   | 2.70   | 2022-02-03 03:35:36 | 2022-02-03 03:35:36 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa

Hasilnya, data kita sudah berhasil di update, serta kolom `update_at` juga berisi tanggal pada saat proses update dilakukan.

Jika kita butuh membuat 2 atau lebih kondisi update, misalnya untuk mahasiswa dengan ipk kurang dari 3 dan nama selain 'alex', bisa men-chaining method `where()` seperti contoh berikut:

```
app/Http/Controllers/MahasiswaController.php

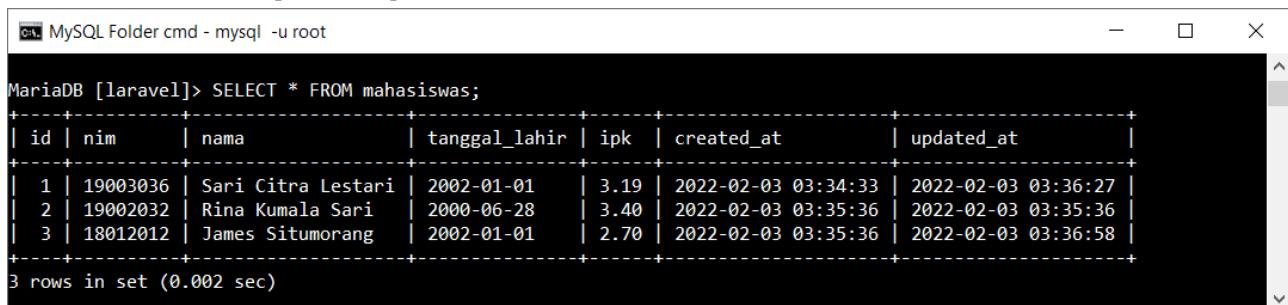
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function updateWhere(){
6         $result = DB::table('mahasiswa')
7             ->where('ipk', '<', 3)
8             ->where('nama', '!=', 'alex')
9             ->update(
10                 [
11                     'tanggal_lahir' => '2002-01-01',
12                     'updated_at' => now(),
13                 ]
14             );
15
16         dump($result);
17     }
18 }
```

Di baris 7 dan 8 saya membuat 2 buah method `where()` yang saling di-chaining satu sama lain.

Cara penggunaan method `where()` ini sama seperti yang pernah kita pakai di bab collection, yakni bisa di isi dengan 2 atau 3 argument. Jika diisi dengan 2 argument, maka operasi perbandingan yang dipakai adalah 'sama dengan', misalnya `where('nama', 'Sari Citra Lestari')`, akan mencari isi kolom `nama` yang sama dengan 'Sari Citra Lestari'.

Jika method `where()` dipanggil dengan 3 argument, maka argument kedua akan menjadi operator perbandingan. Sebagai contoh, `where('ipk', '<', 3)` akan mencari kolom `ipk` dengan nilai kurang dari 3, sedangkan `where('nama', '<>', 'alex')` akan mencari kolom `nama` yang tidak sama dengan 'alex'.

Hasilnya, hanya baris yang memenuhi kedua kondisi inilah yang akan di update. Jalankan method `updateWhere()` ini dari alamat `localhost:8000/update-where`, dan berikut isi tabel `mahasiswa` setelah proses update:



```
MySQL [laravel] > SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 19003036 | Sari Citra Lestari | 2002-01-01 | 3.19 | 2022-02-03 03:34:33 | 2022-02-03 03:36:27 |
| 2 | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.40 | 2022-02-03 03:35:36 | 2022-02-03 03:35:36 |
| 3 | 18012012 | James Situmorang | 2002-01-01 | 2.70 | 2022-02-03 03:35:36 | 2022-02-03 03:36:58 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa

Terlihat bahwa tanggal lahir dari 'James Situmorang' sudah berubah menjadi `2002-01-01`, karena memenuhi syarat `ipk < 3` dan bukan bernama alex.

Query builder juga menyediakan method lain bernama `updateOrInsert()`, yang merupakan gabungan dari proses `update` dan `insert`. Jika data yang ingin di update belum ada di tabel, maka jalankan proses insert. Berikut contoh penggunaannya:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function updateOrInsert(){
6         $result = DB::table('mahasiswa')->updateOrInsert(
7             [
8                 'nim' => '19005011',
9             ],
10            [
11                'nama' => 'Riana Putria',
12                'tanggal_lahir' => '2000-11-23',
13                'ipk' => 2.7,
14                'created_at' => now(),
15                'updated_at' => now(),
```

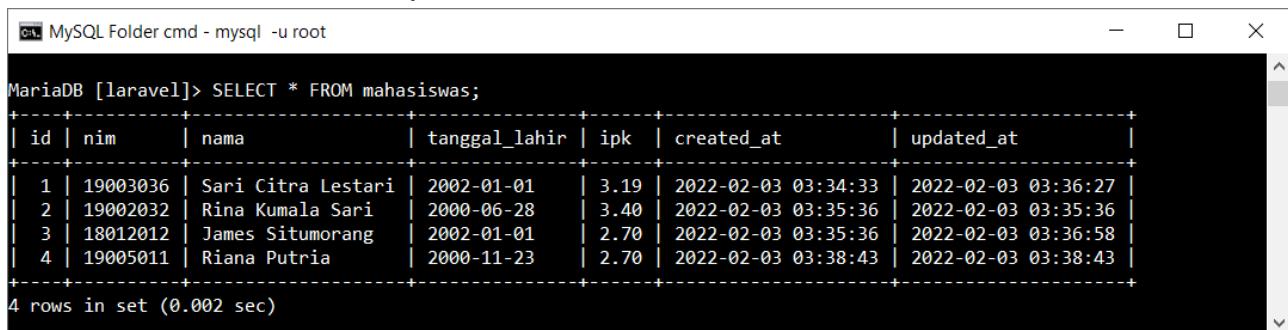
```

16         ]
17     );
18
19     dump($result);
20 }
21 }
```

Method `updateOrInsert()` butuh 2 buah argument. Argument pertama berupa associative array yang berisi kondisi yang dicari. Dalam contoh di atas, saya ingin mencari mahasiswa dengan nim '19005011'. Kemudian sebagai argument kedua adalah data yang akan di update, yang juga ditulis dalam bentuk associative array.

Jika dalam tabel terdapat mahasiswa dengan nim 19005011, maka data mahasiswa tersebut akan di update dengan isi dari argument kedua. Namun jika tidak ditemukan mahasiswa dengan nim tersebut, maka lakukan proses insert.

Jalankan method `updateOrInsert()` ini dengan mengakses halaman `localhost:8000/update-or-insert`. Dan berikut hasilnya:



```

MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+-----+
| id | nim   | nama        | tanggal_lahir | ipk    | created_at  | updated_at  |
+----+-----+-----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2002-01-01 | 3.19  | 2022-02-03 03:34:33 | 2022-02-03 03:36:27 |
| 2  | 19002032 | Rina Kumala Sari   | 2000-06-28 | 3.40  | 2022-02-03 03:35:36 | 2022-02-03 03:35:36 |
| 3  | 18012012 | James Situmorang  | 2002-01-01 | 2.70  | 2022-02-03 03:35:36 | 2022-02-03 03:36:58 |
| 4  | 19005011 | Riana Putria       | 2000-11-23 | 2.70  | 2022-02-03 03:38:43 | 2022-02-03 03:38:43 |
+----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswas

Karena mahasiswa dengan nim 19005011 tidak ditemukan, maka data tersebut akan ditambah ke dalam tabel `mahasiswas`.

15.4. Menghapus Data

Query builder menyediakan method `delete()` untuk menghapus data. Method ini tidak butuh argument apapun, dimana untuk menentukan kolom mana yang akan dihapus kita butuh method `where()`.

Sebagai contoh, saya ingin menghapus data mahasiswa yang memiliki ipk ≥ 3.4 , berikut kode yang bisa dipakai:

app/Http/Controllers/MahasiswaController.php

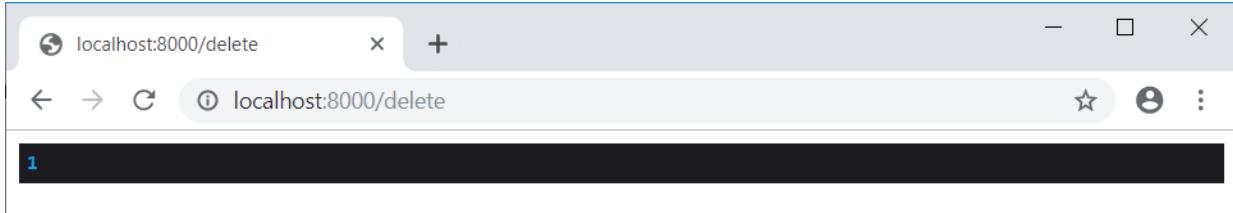
```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function delete(){
```

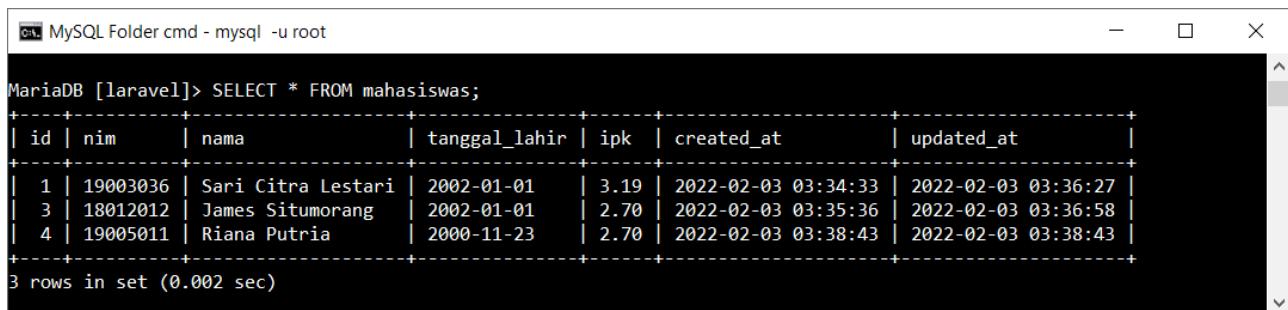
Query Builder

```
6     $result = DB::table('mahasiswa')
7         ->where('ipk','>=',3.4)
8         ->delete();
9
10    dump($result);
11 }
12 }
```

Jalankan method `delete()` ini dengan mengakses halaman `localhost:8000/delete`:



Gambar: Hasil tampilan halaman `http://localhost:8000/delete`



id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2002-01-01	3.19	2022-02-03 03:34:33	2022-02-03 03:36:27
3	18012012	James Situmorang	2002-01-01	2.70	2022-02-03 03:35:36	2022-02-03 03:36:58
4	19005011	Riana Putria	2000-11-23	2.70	2022-02-03 03:38:43	2022-02-03 03:38:43

Gambar: Isi tabel mahasiswa

Terlihat ada 1 data mahasiswa yang sudah dihapus, yakni 'Rina Kumala Sari' yang syarat ipk di atas 4.3.

15.5. Menampilkan Data

Bagian paling menarik dari query builder adalah proses menampilkan data, karena inilah hal yang sangat sering kita pakai sepanjang pembuatan kode program. Atas alasan itu pula terdapat cukup banyak method yang tersedia.

Method `get()`

Pertama, untuk menampilkan semua data dari tabel mahasiswa, bisa menggunakan method `get()` yang langsung diakses dari `DB::table('mahasiswa')`:

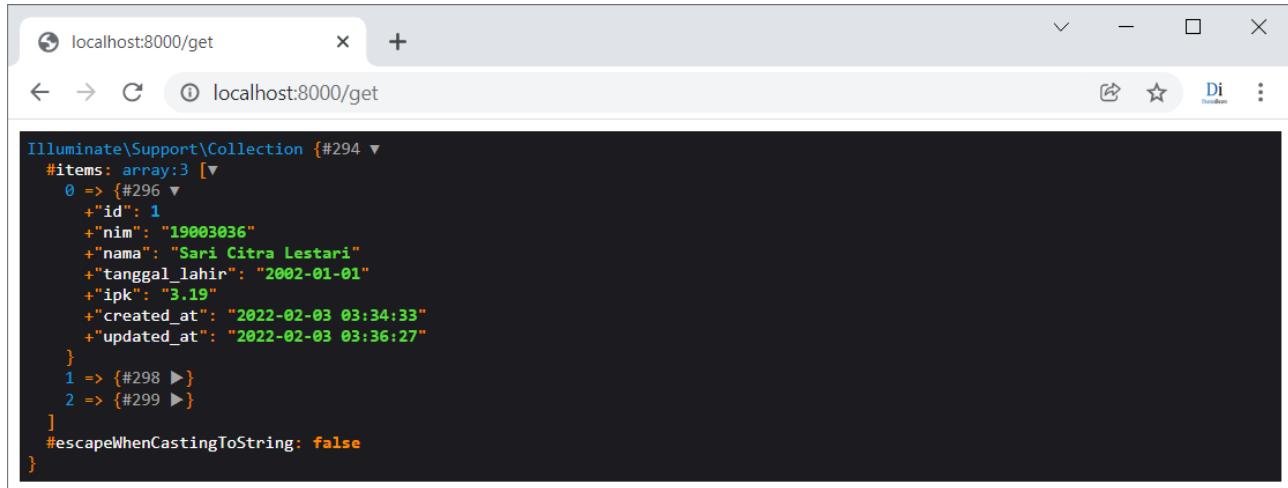
app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function get(){
6 }
```

Query Builder

```
6     $result = DB::table('mahasiswa')->get();
7
8     dump($result);
9 }
10}
11}
```

Cara penggunaannya sangat sederhana, cukup chaining method `get()` dengan hasil dari `DB::table('mahasiswa')`. Jalankan method ini dari halaman `localhost:8000/get`:



```
Illuminate\Support\Collection {#294 ▶
  #items: array:3 [▶
    0 => {#296 ▶
      +"id": 1
      +"nim": "19003036"
      +"nama": "Sari Citra Lestari"
      +"tanggal_lahir": "2002-01-01"
      +"ipk": "3.19"
      +"created_at": "2022-02-03 03:34:33"
      +"updated_at": "2022-02-03 03:36:27"
    }
    1 => {#298 ▶}
    2 => {#299 ▶}
  ]
  #escapeWhenCastingToString: false
}
```

Gambar: Hasil tampilan halaman `http://localhost:8000/get`

Hasil dari method `get()` adalah sebuah **collection**. Setiap baris tabel `mahasiswa` menjadi object dari array yang tersimpan dalam bentuk collection. Karena kita sudah membahas collection sebelumnya, struktur seperti ini seharusnya tidak asing lagi.

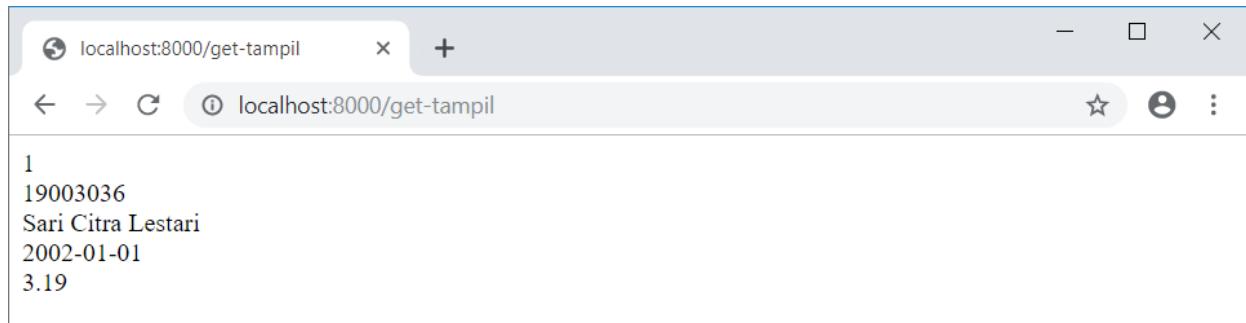
Sebagai latihan, bisakah anda menampilkan isi baris pertama menggunakan perintah `echo?`

Caranya, akses kolom tabel sebagai property dari object `$result[0]`. Ini sudah pernah kita lakukan dalam bab collection:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function getTampil(){
6         $result = DB::table('mahasiswa')->get();
7
8         echo($result[0]->id). '<br>';
9         echo($result[0]->nim). '<br>';
10        echo($result[0]->nama). '<br>';
11        echo($result[0]->tanggal_lahir). '<br>';
12        echo($result[0]->ipk);
13    }
14 }
```

Jalankan method `getTampil()` dari halaman `localhost:8000/get-tampil`:



Gambar: Hasil tampilan halaman `http://localhost:8000/get-tampil`

Bagaimana jika data ini kita teruskan ke view? Tidak masalah. Pastikan view `tampil-mahasiswa.blade.php` masih tersedia, dan jalankan method berikut:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function getView(){
6         $result = DB::table('mahasiswas')->get();
7         return view('tampil-mahasiswa',['mahasiswas' => $result]);
8     }
9 }

```

Isi variabel `$result` yang berisi collection langsung saya input ke dalam argument kedua dari method `view()`. Artinya collection tidak perlu di konversi menjadi array terlebih dahulu, cukup langsung ditulis sebagai `['mahasiswas' => $result]`. Sesampainya di view, variabel `$mahasiswas` sudah bisa diakses.

Jalankan method ini dari alamat `localhost:8000/get-view`:



Gambar: Hasil tampilan halaman `http://localhost:8000/get-view`

Sebagai pengingat, berikut bagian dari view `tampil-mahasiswa.blade.php` yang dipakai untuk

mengakses setiap isi object mahasiswa:

resources/views/tampil-mahasiswa.blade.php

```

1 ...
2   <h1 class="mb-3">Data Mahasiswa</h1>
3   <div class="row">
4     <div class="m-auto">
5       <ol class="list-group">
6         @forelse ($mahasiswas as $mahasiswa)
7           <li class="list-group-item">
8             {{$mahasiswa->nama}} ( {{$mahasiswa->nim}} ), 
9             Tanggal Lahir: {{$mahasiswa->tanggal_lahir}}, 
10            IPK: {{$mahasiswa->ipk}}
11          </li>
12        @empty
13          <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
14        @endforelse
15      </ol>
16    </div>
17  </div>
18 </div>
19 ...

```

Kode yang dipakai sama persis seperti pada pembahasan raw query. Di dalam view, kita bisa mengakses `$mahasiswas` yang berisi array dari object mahasiswa.

Method where() dan orderBy()

Query builder juga menyediakan berbagai method lain untuk proses pencarian yang lebih detail, misalnya digabung dengan method `where()` dan `orderBy()` seperti contoh berikut:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5   public function getWhere(){
6     $result = DB::table('mahasiswas')
7       ->where('ipk', '<', '3')
8       ->orderBy('nama', 'desc')
9       ->get();
10
11     return view('tampil-mahasiswa', ['mahasiswas' => $result]);
12   }
13 }

```

Perintah `DB::table('mahasiswas')->where('ipk', '<', '3')->orderBy('nama', 'desc')->get()` bisa dibaca: 'Ambil semua isi tabel mahasiswas dengan kondisi ipk < 3, lalu urutkan berdasarkan nama secara menurun'.

Atau jika menggunakan query SQL, sama artinya dengan:

Query Builder

```
SELECT * FROM mahasiswa WHERE ipk < 3 ORDER BY nama DESC
```

Berikut hasilnya:

Riana Putria (19005011), Tanggal Lahir: 2000-11-23, IPK: 2.70
James Situmorang (18012012), Tanggal Lahir: 2002-01-01, IPK: 2.70

Gambar: Hasil tampilan halaman http://localhost:8000/get-where

Method select()

Method lain yang tersedia adalah `select()`, yang bisa dipakai untuk membatasi nama kolom. Berikut contoh penggunaannya:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function select(){
6         $result = DB::table('mahasiswa')
7             ->select('nim', 'nama as nama_mahasiswa')
8             ->get();
9
10        dump($result);
11    }
12 }
```

```
Illuminate\Support\Collection {#298 ▶
  #items: array:3 [▼
    0 => {#299 ▼
      +"nim": "19003036"
      +"nama_mahasiswa": "Sari Citra Lestari"
    }
    1 => {#302 ▶}
    2 => {#303 ▶}
  ]
  #escapeWhenCastingToString: false
}
```

Gambar: Hasil tampilan halaman http://localhost:8000/select

Di baris 7 terdapat pemanggilan method `select('nim', 'nama as nama_mahasiswa')`. Kedua

kolom inilah yang akan menjadi nilai collection. Khusus untuk kolom nama, saya tukar dengan alias 'nama_mahasiswa'.

Method skip() dan take()

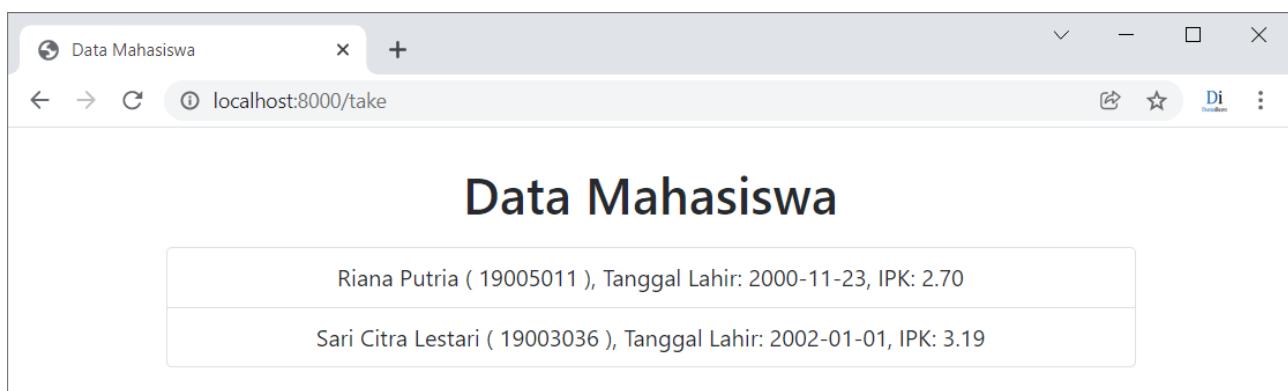
Method pembatasan lain yang bisa kita pakai adalah `skip()` dan `take()`. Keduanya mirip seperti fungsi `LIMIT` di query MySQL, yakni untuk membatasi jumlah baris yang akan diambil.

Kedua method butuh sebuah argument berupa angka yang berisi jumlah baris. Method `skip()` dipakai untuk melompati baris, sedangkan `take()` untuk mengambil baris. Berikut contoh penggunaannya:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function take(){
6         $result = DB::table('mahasiswas')
7             ->orderBy('nama', 'asc')->skip(1)->take(2)->get();
8
9         return view('tampil-mahasiswa', ['mahasiswas' => $result]);
10    }
11 }
```



Gambar: Hasil tampilan halaman <http://localhost:8000/take>

Perintah `DB::table('mahasiswas')->orderBy('nama', 'asc')->skip(1)->take(2)->get()` bisa dibaca: 'Urutkan isi tabel mahasiswas berdasarkan nama, lewatkan 1 baris pertama dan ambil 2 baris berikutnya'.

Jika di dalam tabel `mahasiswas` terdapat 10 baris data, hasil dari query di atas tetap 2 baris saja karena pembatasan dari `take(2)`.

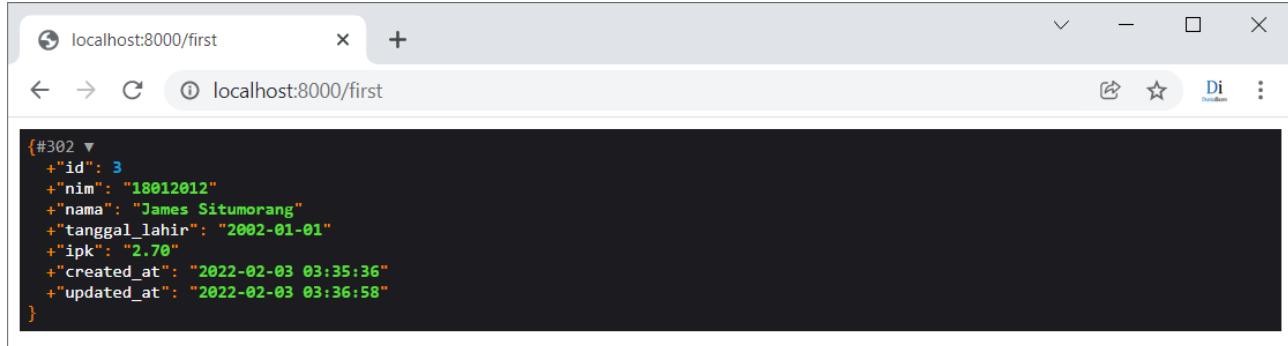
Method first()

Selanjutnya terdapat juga method `first()` yang bisa dipakai untuk mengambil 1 baris data pertama saja. Ini biasa digabung dengan kondisi `where()` seperti contoh berikut:

Query Builder

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function first(){
6         $result = DB::table('mahasiswas')
7             ->where('nama','James Situmorang')->first();
8         dump ($result);
9 }
```



Gambar: Hasil tampilan halaman http://localhost:8000/first

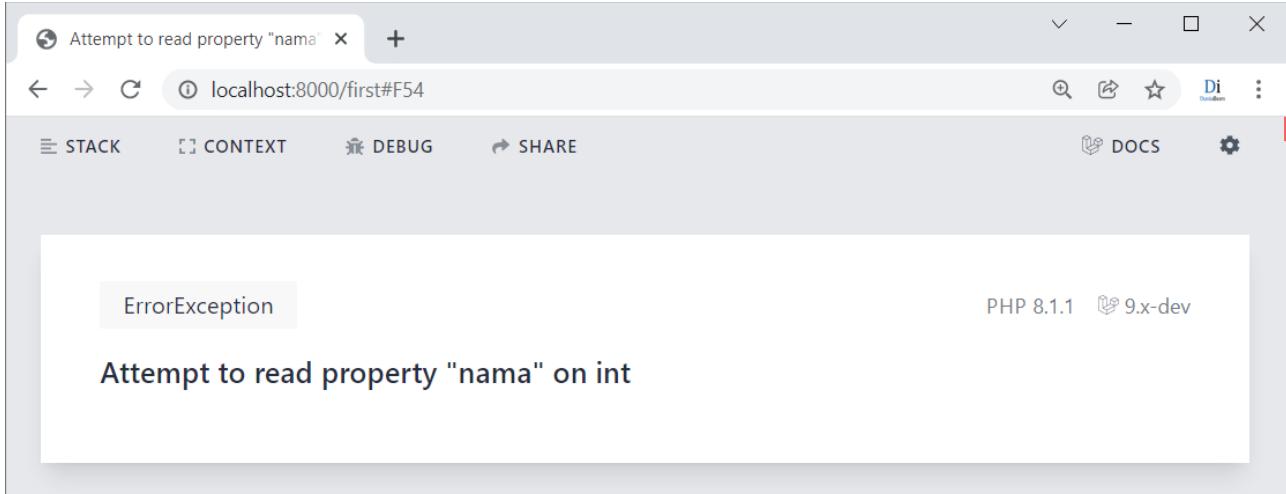
Di baris 6 saya ingin mencari 1 nama mahasiswa yang bernama 'James Situmorang'. Perhatikan bahwa variabel \$result berisi object, bukan collection. Inilah perbedaan mendasar antara `get()` dengan `first()`.

Ini bisa jadi masalah jika kita tidak sadar dan langsung meneruskannya ke dalam view dengan perintah berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function first(){
6         $result = DB::table('mahasiswas')
7             ->where('nama','James Situmorang')->first();
8         return view('tampil-mahasiswa',['mahasiswas' => $result]); // error
9     }
10 }
```

Query Builder



Gambar: Hasil error halaman http://localhost:8000/first

Error terjadi karena sesampainya di view, variabel \$mahasiswa bukan berisi array dari object, tapi sudah langsung object saja. Sehingga tidak bisa diproses dengan perulangan @forelse.

Solusinya kita bisa edit kode di view `tampil-mahasiswa.php` agar langsung mengakses object. Atau alternatif lain, 'bungkus' isi variabel \$result agar tetap sampai sebagai array ke dalam view:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function first(){
6         $result = DB::table('mahasiswa')
7             ->where('nama', 'James Situmorang')->first();
8         return view('tampil-mahasiswa', ['mahasiswa' => [$result]]);
9     }
10 }
```



Gambar: Hasil halaman http://localhost:8000/first

Perhatikan cara pengiriman data ke view di baris 8, saya menulisnya sebagai `['mahasiswa' => [$result]]`. Karena variabel \$result dikirim dalam tanda kurung siku array, maka sesampainya di view tetap berbentuk array dari object.

Method find()

Method `find()` dipakai untuk mencari baris tabel berdasarkan kolom `id`. Sampai saat ini saya memang tidak menampilkan isi kolom `id`, tapi kolom inilah yang berfungsi sebagai primary key di tabel `mahasiswa`. Pada saat membuat migration, kolom `id` di set dengan atribut `auto increment`, sehingga nilainya akan naik secara otomatis.

Method `find()` bisa dipakai secara cara singkat untuk mengambil 1 data tabel yang memiliki `id` tertentu. `Id` tersebut diinput sebagai argument ke dalam method `find()`. Berikut contoh penggunaannya:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function find(){
6         $result = DB::table('mahasiswa')->find(4);
7         return view('tampil-mahasiswa', ['mahasiswa' => [$result]]);
8     }
9 }
```



Gambar: Hasil halaman `http://localhost:8000/find`

Sama seperti `first()`, hasil dari method `id` juga berbentuk 1 object saja, bukan array atau collection.

Perintah `DB::table('mahasiswa')->find(4)` di baris 6 sama artinya dengan `DB::table('mahasiswa')->where('id',4)->first()`.

Method selectRaw()

Method `selectRaw()` bisa dipakai untuk menjalankan query `select` dalam bentuk perintah raw SQL. Berikut contoh penggunaannya:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
```

```

5     public function raw(){
6         $result = DB::table('mahasiswa')
7             ->selectRaw('count(*) as total_mahasiswa')
8             ->get();
9
10        echo($result[0]->total_mahasiswa). '<br>'; // 3
11    }
12 }

```

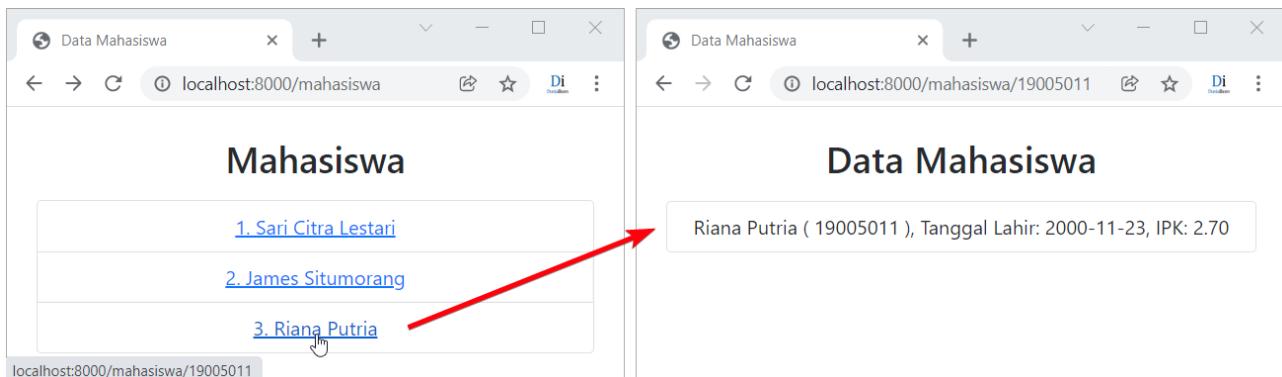
Di sini saya menjalankan query `count(*) as total_mahasiswa` untuk menghitung jumlah baris yang terdapat di dalam tabel `mahasiswa`. Meskipun query ini hanya menghasilkan angka integer, namun karena diakses menggunakan method `get()`, hasilnya berupa collection. Dengan demikian cara mengakses nilainya adalah dari `$result[0]->total_mahasiswa`.

Jika anda perlu menjalankan query yang cukup kompleks, tidak ada salahnya menggunakan DB facade atau raw query seperti yang kita bahas dalam bab sebelum ini.

15.6. Mini Case Study

Sebagai materi penutup bab query builder, saya ingin membuat sebuah studi kasus sederhana. Idenya adalah, rancang halaman yang berisi daftar nama mahasiswa dalam bentuk link. Ketika link di klik, akan terbuka halaman baru yang menampilkan data detail dari mahasiswa tersebut.

Berikut hasil akhir yang di inginkan:



Gambar: Daftar mahasiswa yang ketika di klik akan menampilkan detail mahasiswa

Untuk keperluan ini, kita butuh 2 buah route, satu untuk menampilkan daftar nama mahasiswa, dan satu untuk detail data mahasiswa.

Perhatikan alamat URL data mahasiswa (kanan), yakni `localhost:8000/mahasiswa/19005011`. Di segmen terakhir terdapat nomor nim dari mahasiswa tersebut. Artinya, ini merupakan sebuah **route parameter**.

Ketika di ketik alamat `localhost:8000/mahasiswa/19005011`, akan tampil detail data mahasiswa yang memiliki nomor nim `19005011`. Jika diakses halaman `localhost:8000/`

`mahasiswa/18012012`, akan tampil detail data mahasiswa dengan nomor nim `18012012`, dst.

Kita mulai dari route terlebih dahulu:

routes/web.php

```

1 <?php
2
3 //...
4 Route::get('/mahasiswa', [MahasiswaController::class, 'index']);
5 Route::get('/mahasiswa/{nim}', [MahasiswaController::class, 'mahasiswa']);

```

Route pertama, `'/mahasiswa'` dipakai untuk mengakses method `index()` yang ada di controller `MahasiswaController`. Inilah halaman yang menampilkan daftar nama mahasiswa.

Route kedua, `'/mahasiswa/{nim}'` dipakai untuk mengakses method `mahasiswa()` di controller `MahasiswaController`. Tanda kurung kurawal menandakan **nim** adalah sebuah route parameter. Data ini akan menjadi kunci untuk menampilkan detail data dari setiap mahasiswa.

Berikut kode program untuk method `index()`:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function index(){
6         $result = DB::table('mahasiswas')->select('nim', 'nama')->get();
7         return view('index-mahasiswa',[ 'mahasiswas' => $result]);
8     }
9 }

```

Tidak ada yang baru di sini, di baris 6 saya menjalankan perintah untuk mengambil nilai kolom `nim` dan `nama` tabel `mahasiswas`, yang kemudian dikirim ke view `index-mahasiswa`.

Dan berikut kode untuk view `index-mahasiswa.blade.php`:

resources/views/index-mahasiswa.blade.php

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="{{ asset('/css/bootstrap.min.css') }}" rel="stylesheet">
8     <title>Data Mahasiswa</title>
9 </head>
10 <body>
11
12 <div class="container text-center p-4">
13     <h1 class="mb-3">Mahasiswa</h1>
14     <div class="row">

```

```

15   <div class="col-sm-8 col-md-6 m-auto">
16     <ol class="list-group">
17       @forelse ($mahasiswa as $mahasiswa)
18         <li class="list-group-item">
19           <a href="{{ url('mahasiswa/'.$mahasiswa->nim)}}">
20             {{ $loop->iteration }}. {{$mahasiswa->nama}}
21           </a>
22         </li>
23       @empty
24         <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
25       @endforelse
26     </ol>
27   </div>
28 </div>
29 </div>
30
31 </body>
32 </html>

```

Kode program ini sangat mirip seperti view `tampil-mahasiswa.blade.php`, yang berbeda hanya di baris perulangan `@forelse`.

Di baris 19 saya membuat tag `<a>` dengan atribut `href` yang berisi `{{ url('mahasiswa/'.$mahasiswa->nim)}}`. Karena berada di dalam perulangan `@forelse`, maka nilai `href` ini akan berbeda-beda untuk setiap mahasiswa.

Di baris 20 terdapat kode baru, yakni `{{ $loop->iteration }}`. Ini adalah kode khusus bawaan blade yang hanya bisa diakses dari dalam perulangan. Fungsinya, untuk meng-generate angka increment dalam setiap iterasi. Kemudian diikuti dengan nama mahasiswa yang diakses dari `>{{$mahasiswa->nama}}`.

Saat ini terdapat 3 data di tabel `mahasiswa`, sehingga perintah di baris 17 – 22 akan di proses menjadi kode HTML berikut:

```

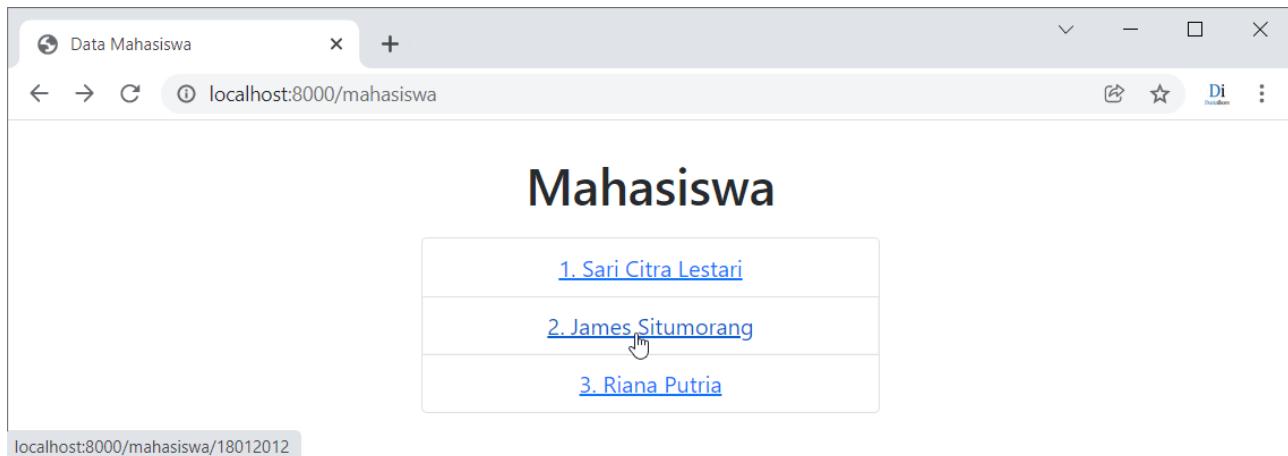
<li class="list-group-item">
  <a href="http://localhost:8000/mahasiswa/19003036">
    1. Sari Citra Lestari
  </a>
</li>

<li class="list-group-item">
  <a href="http://localhost:8000/mahasiswa/18012012">
    2. James Situmorang
  </a>
</li>

<li class="list-group-item">
  <a href="http://localhost:8000/mahasiswa/19005011">
    3. Riana Putria
  </a>
</li>

```

Inilah yang ditampilkan sebagai link ketika mengakses halaman localhost:8000/mahasiswa:



Gambar: Hasil halaman http://localhost:8000/mahasiswa

Selanjutnya, kita masuk ke method `mahasiswa()`:

```
app/Http/Controllers/MahasiswaController.php

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function mahasiswa($nim){
6         return $nim;
7     }
8 }
```

Ini adalah method yang akan dijalankan ketika route '/mahasiswa/{nim}' diakses, yang berarti kita memiliki sebuah route parameter. Route parameter ini ditangkap oleh variabel `$nim` di baris 5.

Di baris 6 saya langsung me-return varibel `$nim` ini, tujuannya sekedar menguji apakah route parameter sudah sampai ke method `mahasiswa()` atau belum. Untuk mengujinya, buka kembali halaman `localhost:8000/mahasiswa`, lalu klik salah satu link, seharusnya akan tampil hasil sebagai berikut:



Gambar: Testing route parameter \$nim

Jika tampil angka `nim`, maka route parameter sudah bisa diakses. Proses testing seperti ini sangat penting sebelum menginput `$nim` ke dalam query builder. Tujuannya untuk menghindari error yang lebih kompleks.

Dan berikut kode program sebenarnya untuk method `mahasiswa()`:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function mahasiswa($nim){
6         $result = DB::table('mahasiswas')->where('nim',$nim)->get();
7         return view('tampil-mahasiswa',[ 'mahasiswas' => $result]);
8     }
9 }
```

Perintah `DB::table('mahasiswas')->where('nim',$nim)->get()` dipakai untuk mengambil data mahasiswa dengan kondisi `WHERE 'nim' = $nim`. Perhatikan bahwa nilai `$nim` akan berubah-ubah sesuai nilai yang berasal dari URL.

Terakhir variabel `$result` yang berisi data mahasiswa diteruskan ke dalam view `tampil-mahasiswa`. Isi dari view ini sama seperti yang kita pakai selama ini.



Gambar: Detail data mahasiswa ditampilkan berdasarkan angka nim dari URL

Sip, studi kasus kita sudah selesai. Kurang lebih anda bisa mendapat gambaran bagaimana cara menampilkan data dari database dan memprosesnya dalam bentuk aplikasi sederhana.

Dalam bab ini kita telah membahas tentang query builder, yang dalam banyak hal relatif lebih ringkas untuk ditulis dibandingkan raw query.

Selanjutnya, kita akan masuk ke 'bintang' dari pengaksesan database di laravel, yakni **Eloquent ORM**.

16. Eloquent ORM

Pada bab kali ini kita akan membahas **Eloquent ORM**, yakni sebuah cara modern untuk mengakses database dari Laravel. Selain menyediakan fitur untuk proses CRUD biasa, eloquent juga memiliki fitur **Soft Delete**, dimana kita bisa mengembalikan data yang sudah terhapus.

16.1. Pengertian Eloquent ORM

Eloquent ORM adalah cara pengaksesan database dimana setiap baris tabel dianggap sebagai sebuah object. Kata **ORM** sendiri merupakan singkatan dari **Object-relational mapping**, yakni sebuah teknik programming untuk mengkonversi data ke dalam bentuk object.

Sebagaimana yang sudah kita pahami, database terdiri dari kumpulan tabel yang saling terhubung. Di dalam setiap tabel, data disimpan dalam bentuk baris dan kolom. ORM dipakai untuk mengubah baris dan kolom ini menjadi sebuah object. Nantinya, setiap kolom akan menjadi property dari object tersebut.

Sebagai perbandingan, berikut cara menginput data ke tabel `mahasiswa` menggunakan **query builder**:

```

1  public function insert(){
2      $result = DB::table('mahasiswa')->insert(
3          [
4              'nim' => '19003036',
5              'nama' => 'Sari Citra Lestari',
6              'tanggal_lahir' => '2001-12-31',
7              'ipk' => 3.5,
8              'created_at' => now(),
9              'updated_at' => now(),
10         ]
11     );
12 }
```

Dengan query builder, data yang akan diinput kita tulis dalam bentuk associative array (baris 4 – 9). Array ini merupakan argument dari method `insert()` yang diakses dari `DB::table('mahasiswa')`.

Jika menggunakan **Eloquent ORM**, data tabel ditulis sebagai property dari object **Mahasiswa**:

```

1  public function insert(){
2      $mahasiswa = new Mahasiswa;
```

```

3     $mahasiswa->nim = '19003036';
4     $mahasiswa->nama = 'Sari Citra Lestari';
5     $mahasiswa->tanggal_lahir = '2001-12-31';
6     $mahasiswa->ipk = 3.5;
7     $mahasiswa->save();
8 }
```

Di baris 2 terdapat perintah untuk proses instansiasi (pembuatan object) class **Mahasiswa** yang disimpan ke dalam variabel `$mahasiswa`. Kemudian di baris 3 – 6 saya mengisi beberapa property ke dalam object `$mahasiswa`. Nama property ini harus sesuai dengan nama kolom yang ada di tabel `mahasiswas`.

Terakhir, di baris 7 terdapat pemanggilan method `$mahasiswa->save()` yang akan menyimpan object `$mahasiswa` sebagai baris baru ke dalam tabel `mahasiswas`.

Inilah cara kerja dari ORM, yakni mengkonversi baris tabel menjadi sebuah **object**. Teknik penulisan seperti ini sangat cocok di pakai dalam framework seperti Laravel, yang semuanya sudah menerapkan konsep pemrograman berbasis object.

Bagi yang sudah terbiasa menggunakan perintah query SQL, ORM mungkin terasa sedikit membingungkan, karena cukup berbeda dengan cara yang kita pakai selama ini (menulis perintah query). Namun lambat laun anda akan terbiasa, selain itu ORM membuat kode program kita menjadi lebih rapi dan 'seragam' dengan fitur OOP lain di Laravel.

Eloquent adalah "nama" dari implementasi ORM yang digunakan Laravel. Selain eloquent, masih banyak ORM yang cukup terkenal, diantaranya **Doctrine ORM** yang dipakai framework **Symfony**.

Istilah lain yang cukup sering di jumpai adalah **Active record pattern**, yakni pola arsitektur yang dipakai Eloquent ORM. Selain Active record pattern, pola arsitektur ORM lain adalah **Data mapper pattern**. Perbedaan antara keduanya tidak akan kita bahas karena sangat teknis.

Yang cukup kita pahami adalah, *active record pattern* merupakan konsep pola arsitektur yang dipakai oleh Eloquent ORM. Karena itulah dari [dokumentasi Laravel](#), ditulis bahwa:

The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database.

Yang bisa diterjemahkan sebagai:

Eloquent ORM yang ada di Laravel menyediakan sebuah implementasi cantik dari ActiveRecord sederhana yang bisa dipakai untuk mengelola database anda.

Jika tertarik dengan perbedaan antara **active record pattern** yang dipakai oleh Eloquent ORM dengan **data mapper pattern**, bisa ke: [What's the difference between Active Record and Data Mapper?](#)

Membuat Migration dan Controller

Sebagai bahan praktek, saya kembali menggunakan tabel `mahasiswa` dan controller `MahasiswaController` seperti yang kita pakai dalam bab **DB Facade** dan **Query builder**.

Jika anda mengikuti semua praktek dari bab tersebut, tabel `mahasiswa` mungkin masih berisi sesuatu. Silahkan kosongkan terlebih dahulu menggunakan query `TRUNCATE mahasiswa`, atau bisa juga jalankan perintah `php artisan migration:fresh`.

Sebagai pengingat, berikut kode program `migration` untuk membuat tabel `mahasiswa`:

```
1 Schema::create('mahasiswa', function (Blueprint $table) {
2     $table->id();
3     $table->char('nim',8)->unique();
4     $table->string('nama');
5     $table->date('tanggal_lahir');
6     $table->decimal('ipk',3,2)->default(1.00);
7     $table->timestamps();
8});
```

Untuk route, tulis ulang dengan kode berikut:

`routes/web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/cek-object', [MahasiswaController::class, 'cekObject']);
7
8 Route::get('/insert', [MahasiswaController::class, 'insert']);
9 Route::get('/mass-assignment', [MahasiswaController::class, 'massAssignment']);
10 Route::get('/mass-assignment2', [MahasiswaController::class, 'massAssignment2']);
11
12 Route::get('/update', [MahasiswaController::class, 'update']);
13 Route::get('/update-where', [MahasiswaController::class, 'updateWhere']);
14 Route::get('/mass-update', [MahasiswaController::class, 'massUpdate']);
15
16 Route::get('/delete', [MahasiswaController::class, 'delete']);
17 Route::get('/destroy', [MahasiswaController::class, 'destroy']);
18 Route::get('/mass-delete', [MahasiswaController::class, 'massDelete']);
19
20 Route::get('/all', [MahasiswaController::class, 'all']);
21 Route::get('/all-view', [MahasiswaController::class, 'allView']);
22 Route::get('/get-where', [MahasiswaController::class, 'getWhere']);
23 Route::get('/test-where', [MahasiswaController::class, 'testWhere']);
24 Route::get('/first', [MahasiswaController::class, 'first']);
25 Route::get('/find', [MahasiswaController::class, 'find']);
26 Route::get('/latest', [MahasiswaController::class, 'latest']);
27 Route::get('/limit', [MahasiswaController::class, 'limit']);
28 Route::get('/skip-take', [MahasiswaController::class, 'skipTake']);
29
30 Route::get('/soft-delete', [MahasiswaController::class, 'softDelete']);
```

```

31 Route::get('/with-trashed',      [MahasiswaController::class, 'withTrashed']);
32 Route::get('/restore',          [MahasiswaController::class, 'restore']);
33 Route::get('/force-delete',     [MahasiswaController::class, 'forceDelete']);

```

Inilah daftar route yang akan kita bahas sepanjang bab nanti. Kembali, saya juga memakai **kebab case** untuk nama route yang diambil dari nama method. Sebagai contoh, jika nama method controller yang diakses adalah `getWhere()`, maka routenya menjadi `get-where`, dst.

Untuk `MahasiswaController`, silahkan hapus semua perintah yang ada dan sisakan kode berikut:

`app/Http/Controllers/MahasiswaController.php`

```

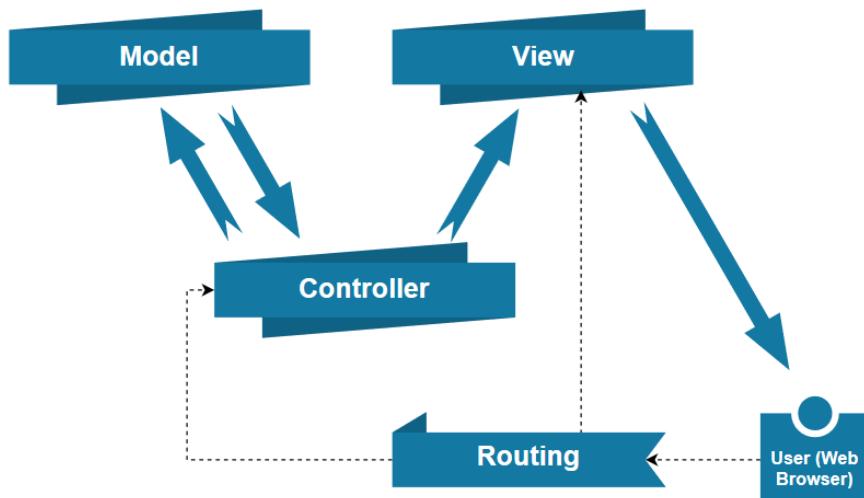
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     // kode program akan kita tulis disini
10 }

```

Salah satu perbedaan dengan `MahasiswaController` di bab DB Facade atau bab Query Builder adalah, Eloquent ORM tidak butuh class **DB** sehingga perintah import `use Illuminate\Support\Facades\DB` tidak perlu ditulis.

16.2. Pembuatan Model

Pada bab DB Facade dan Query Builder kita sudah mengakses database tapi belum dalam bentuk **Model**, semua kode program di tulis di dalam controller. Melihat diagram konsep MVC, seharusnya terdapat komponen Model yang akan berkomunikasi dengan database:



Gambar: Diagram Alur MVC + Routing

Eloquent ORM memerlukan Model untuk proses konversi data tabel menjadi object. Object inilah yang nantinya akan kita akses dari dalam controller. Sama seperti pembuatan berbagai file di Laravel, tersedia perintah `php artisan` untuk membuat Model.

Berikut format dasar perintah yang digunakan:

```
php artisan make:model <namaModel>
```

Nama model harus berpasangan dengan nama tabel namun dalam bentuk tunggal (*singular*). Jika kita membuat tabel `mahasiswa`, maka nama modelnya adalah **Mahasiswa**, atau jika memiliki tabel `matakuliah` maka nama modelnya adalah **Matakuliah**.

Karena kita sudah memiliki tabel `mahasiswa`, maka perintah yang diperlukan adalah:

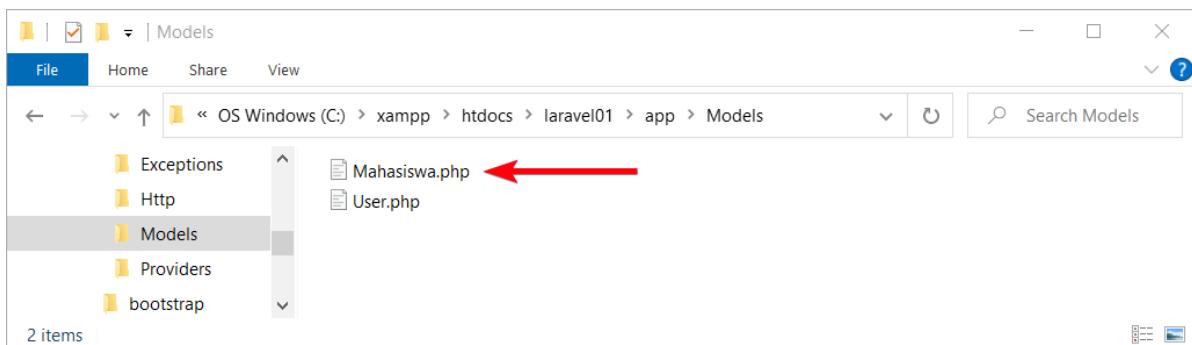
```
php artisan make:model Mahasiswa
```

```
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa
Model created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: pembuatan model Mahasiswa

Setelah dijalankan, file baru bernama `Mahasiswa.php` akan muncul di folder `app\Models`.



Gambar: file model Mahasiswa.php

Berikut isi dari file model `Mahasiswa.php` yang di generate Laravel:

app/Models/Mahasiswa.php

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Mahasiswa extends Model
9  {
10      use HasFactory;
11 }
```

File model terdiri dari beberapa baris perintah. Di bagian awal terdapat namespace App\Models, kemudian diikuti perintah import class HasFactory dan class Model di baris 5 - 6.

Isi class Mahasiswa dimulai pada baris 8 – 11, dimana class Mahasiswa meng-extends class Model serta memiliki satu perintah use HasFactory di baris 10.

Factory adalah fitur Laravel untuk membuat "object Model". Dalam buku ini kita belum membahas cara penggunaan factory dan akan menjadi jatah buku [Laravel In Depth #1](#).

Factory juga bukan fitur wajib untuk bisa menggunakan model, serta tidak semua model butuh factory. Oleh sebab itu perintah import class HasFactory di baris 5 dan use HasFactory di baris 10 sebenarnya bisa dihapus dan tidak akan berpengaruh apa-apa dalam praktek kita sepanjang buku ini.

Kembali ke file model, yang cukup unik kita tidak akan banyak menambah kode program ke dalamnya. Semua 'kerja keras' sudah dilakukan oleh class Illuminate\Database\Eloquent\Model yang di import pada baris 6. Jika anda iseng membuka file ini, isinya terdiri dari lebih 1650 baris kode program!, dan itu pun juga banyak meng-import berbagai class lain.

Untuk bisa menggunakan Model (dan **Eloquent ORM**), kode yang ada di class Illuminate\Database\Eloquent\Model tidak perlu dipahami. Sebagai pengguna framework, kita cukup belajar cara pakainya saja. Inilah keunggulan (dan juga kekurangan) framework, dimana kita sering tidak tau bagaimana sebuah fitur bekerja, karena *it's just work!*

16.3. Pengaksesan Model

Pembuatan kode program dengan Eloquent ORM tetap di lakukan dari controller. Hanya saja sekarang kita butuh mengakses Model, yakni class Mahasiswa yang ada di file Mahasiswa.php.

Sama seperti mengakses Facade, class Mahasiswa ini harus di import terlebih dahulu ke dalam controller. Caranya, tulis kode berikut di bagian atas file MahasiswaController:

```
use App\Models\Mahasiswa;
```

Dengan demikian, kita sudah bisa mengakses model Mahasiswa sepanjang controller. Berikut percobaannya:

```
app/Http/Controllers/MahasiswaController.php
```

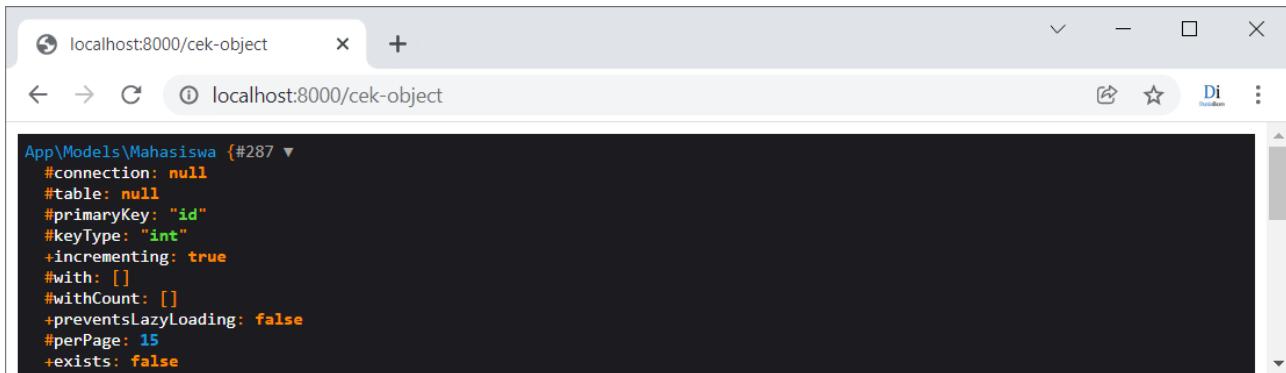
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Mahasiswa;
7
8 class MahasiswaController extends Controller
```

```

9  {
10     public function cekObject(){
11         $mahasiswa = new Mahasiswa;
12
13         dump($mahasiswa);
14     }
15 }
```

Di baris 6 terdapat tambahan perintah `use App\Models\Mahasiswa` yang dipakai untuk proses import file model `Mahasiswa`.

Kemudian di dalam method `cekObject()`, saya meng-instansiasi class `Mahasiswa` yang disimpan ke variabel `$mahasiswa`. Variabel `$mahasiswa` ini selanjutnya di `dump()` dengan hasil sebagai berikut:



```

App\Models\Mahasiswa {#287 ▾
  #connection: null
  #table: null
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  +preventsLazyLoading: false
  #perPage: 15
  +exists: false
}
```

Gambar: Hasil `dump($mahasiswa)`

Terlihat isi dari object `Mahasiswa` dengan berbagai property. Dari object inilah kita akan melakukan banyak hal untuk memproses data ke tabel `mahasiswas`.

Alternatif lain untuk mengakses class `Mahasiswa` adalah dengan menulis namespace langsung pada saat proses instansiasi seperti kode berikut:

```
$mahasiswa = new \App\Models\Mahasiswa;
```

Ini bisa dilakukan jika kita tidak ingin mengimport class `Mahasiswa` secara global.

16.4. Menginput Data

Proses menginput data menggunakan Eloquent ORM cukup sederhana. Caranya, buat object dari model `Mahasiswa`, input data ke dalam atribut yang bersesuaian dengan nama kolom tabel, dan terakhir jalankan method `save()`. Berikut kode program yang dimaksud:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
```

```

5     public function insert(){
6         $mahasiswa = new Mahasiswa;
7         $mahasiswa->nim = '19003036';
8         $mahasiswa->nama = 'Sari Citra Lestari';
9         $mahasiswa->tanggal_lahir = '2001-12-31';
10        $mahasiswa->ipk = 3.5;
11        $mahasiswa->save();
12
13        dump($mahasiswa);
14    }
15 }
```

Kode program ini sudah pernah kita bahas di awal bab. Di baris 6 terdapat proses instansiasi class `Mahasiswa` ke dalam object `$mahasiswa`. Kemudian di baris 7-10 saya mengisi beberapa property yang sama dengan nama kolom tabel. Tidak semua kolom harus diisi, tergantung kebutuhan dan selama sesuai dengan syarat di tabel.

Pada baris 11, method `$mahasiswa->save()` dipakai untuk menyimpan data-data ini ke dalam database. Dan terakhir object `$mahasiswa` saya `dump()` untuk melihat isinya.

Jalankan method di atas dengan membuka alamat `localhost:8000/insert`. Dari hasil tampilan `dump()`, silahkan buka `#attributes`:

```
+wasRecentlyCreated: true
#escapeWhenCastingToString: false
#attributes: array:7 [▼
  "nim" => "19003036"
  "nama" => "Sari Citra Lestari"
  "tanggal_lahir" => "2001-12-31"
  "ipk" => 3.5
  "updated_at" => "2022-02-03 10:59:43"
  "created_at" => "2022-02-03 10:59:43"
  "id" => 1
]
#original: array:7 [▶]
```

Gambar: Hasil localhost:8000/insert

Di dalam bagian `#attributes` bisa terlihat semua property yang baru saja kita input, ditambah kolom khusus yang di-generate secara otomatis oleh Laravel, seperti kolom `id`, serta kolom `updated_at` dan kolom `created_at` yang juga otomatis berisi data `timestamp`.

Untuk memastikan, silahkan cek dari phpMyAdmin atau cmd MySQL Client:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.50	2022-02-03 10:59:43	2022-02-03 10:59:43

1 row in set (0.002 sec)

Gambar: Isi tabel mahasiswa

Sip, data sudah masuk ke dalam tabel `mahasiswa`. Perhatikan kolom `updated_at` dan `created_at` yang juga otomatis berisi nilai.

Mass Assignment

Alternatif cara lain untuk proses input menggunakan Eloquent adalah dengan teknik **mass assignment**. Disebut seperti ini karena kita bisa mengisi banyak property untuk object `Mahasiswa` dalam sekali proses.

Cara input *mass assignment* dilakukan dengan mengakses static method `create()` dari Model object. Karena kita menggunakan model `Mahasiswa`, maka pemanggilan method ini adalah dengan perintah `Mahasiswa::create()`. Data yang akan diinput ditulis sebagai associative array dan menjadi argument dari method `Mahasiswa::create()`.

Berikut contoh penggunaannya:

`app/Http/Controllers/MahasiswaController.php`

```

1  <?php
2
3  class MahasiswaController extends Controller
4  {
5      public function massAssignment(){
6          Mahasiswa::create(
7              [
8                  'nim' => '19021044',
9                  'nama' => 'Rudi Permana',
10                 'tanggal_lahir' => '2000-08-22',
11                 'ipk' => 2.5,
12             ]
13         );
14
15         return "Berhasil di proses";
16     }
17 }
```

Associative array yang ada di baris 8 – 11 inilah yang menjadi alasan kenapa disebut sebagai *mass assignment*. Karena seharusnya kita men-set satu per satu nilai ini ke dalam property object `Mahasiswa`:

```

$mahasiswa->nim = '19021044';
$mahasiswa->nama = 'Rudi Permana';
$mahasiswa->tanggal_lahir = '2000-08-22';
$mahasiswa->ipk = 2.5;
```

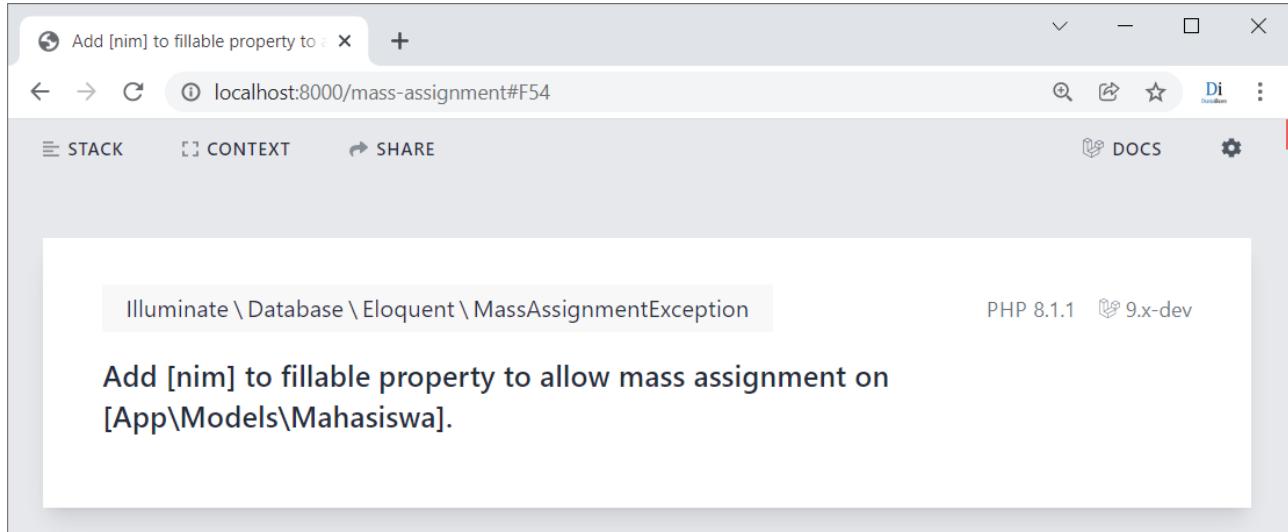
Dengan method `Mahasiswa::create()`, kode di atas diganti menjadi associative array.

Teknik seperti ini akan berguna untuk pemrosesan form karena bawaan Laravel inputan form sudah dalam bentuk associative array. Sehingga jika kita ingin langsung menginput hasil form ke dalam tabel di database, bisa menjalankan kode seperti berikut:

```
Mahasiswa::create([hasil_form]);
```

Lebih lanjut tentang pemrosesan form akan kita bahas dalam bab berikutnya.

Kembali ke proses input menggunakan *mass assignment*, jalankan dengan membuka halaman `localhost:8000/mass-assignment`:



Gambar: Hasil error http://localhost:8000/mass-assignment

Hasilnya tampil pesan error dengan keterangan `Add [nim] to fillable property to allow mass assignment on [App\Models\Mahasiswa]`. Error ini terjadi karena Laravel membatasi akses ke sebuah tabel ketika di proses menggunakan mass assignment.

Pembatasan ini dibuat karena mass assignment sering dipakai dengan nilai yang langsung berasal dari form. Sehingga untuk mencegah kemungkinan terjadi 'manipulasi data', Laravel mewajibkan programmer untuk menulis nama kolom apa saja yang boleh diinput.

Dalam contoh ini, saya harus mendaftarkan kolom 'nim', 'nama', 'tanggal_lahir' dan 'ipk' ke dalam property **\$fillable** yang ada di model `Mahasiswa`. Property ini memang tidak ada sebelumnya dan harus kita tambah manual.

Silahkan buka file model `Mahasiswa.php`, lalu tambah 1 baris berikut ke dalam class `Mahasiswa`:

```
protected $fillable = ['nim', 'nama', 'tanggal_lahir', 'ipk'];
```

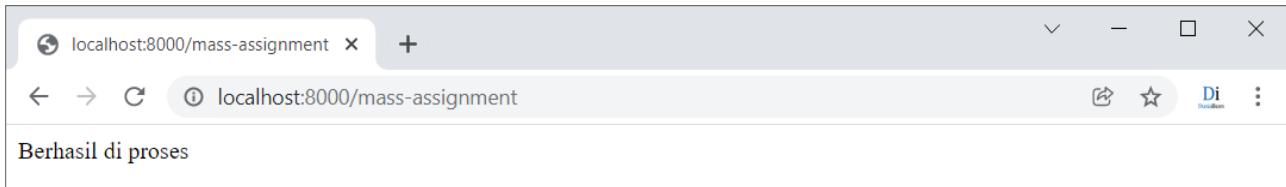
Berikut kode akhir dari file model `Mahasiswa.php`:

app/Models/Mahasiswa.php

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
```

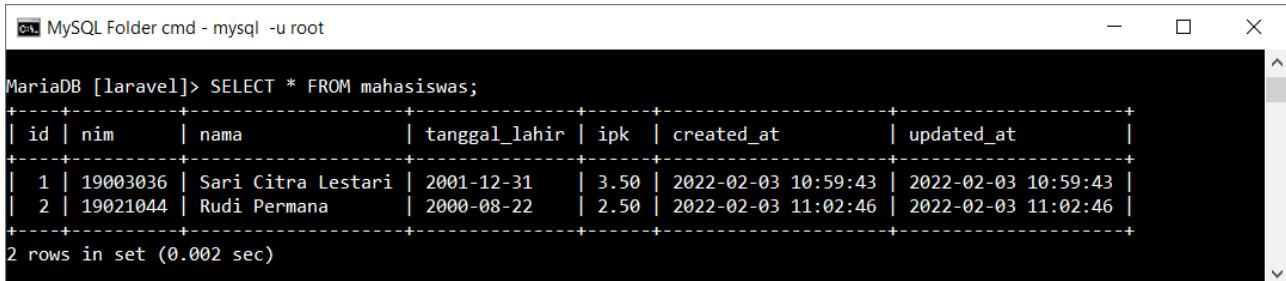
```
8 class Mahasiswa extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['nim', 'nama', 'tanggal_lahir', 'ipk'];
12 }
```

Dengan penambahan ini, jalankan ulang method `massAssignment()` dari halaman `localhost:8000/mass-assignment`:



Gambar: Hasil `http://localhost:8000/mass-assignment`

Tidak ada error. Mari kita cek ke database:



Gambar: Isi tabel mahasiswas

Terlihat data Rudi Permana sudah berhasil masuk ke tabel `mahasiswas`.

Cara lain untuk mengizinkan nama kolom bisa diakses dari `mass assignment` adalah menggunakan property `$guarded`.

Berbeda dengan `$fillable`, `$guarded` dipakai untuk menulis nama kolom apa saja yang **tidak boleh diisi**. Sebagai contoh, jika saya memutuskan kolom `ipk` tidak boleh diisi menggunakan mass assignment, bisa menulis kode berikut:

```
protected $guarded = ['ipk'];
```

Akibatnya, pada saat proses input dilakukan, nilai untuk kolom `ipk` akan terhalang dan tidak bisa sampai ke database meskipun sudah tertulis dalam controller.

Salah satu teknik yang sering dipakai adalah mengisi array kosong ke dalam `$guarded`:

```
protected $guarded = [];
```

Jika ditulis seperti ini, maka tidak ada pembatasan apapun ke dalam tabel. Artinya, semua kolom bisa di proses dengan `mass assignment`, termasuk seandainya kita mengubah struktur tabel (mengubah atau menambah kolom baru). Ini terdengar sedikit berbahaya, tapi dengan teknik tertentu pada saat pemrosesan form nanti, masalah ini bisa diatasi.

Catatan lain, property `$fillable` dan `$guarded` tidak bisa dipakai bersamaan, tapi harus pilih salah satu.

Sebagai contoh praktek dari `$guarded`, saya akan modifikasi kembali isi model `Mahasiswa` sebagai berikut:

app/Models/Mahasiswa.php

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Mahasiswa extends Model
9 {
10     use HasFactory;
11     protected $guarded = [];
12 }
```

Sekarang di dalam class `Mahasiswa` terdapat property `protected $guarded = []`, yang berarti semua kolom bisa diproses menggunakan *mass assignment* (tidak ada pembatasan). Kode ini akan kita pakai sampai akhir bab nanti.

Fungsi dari pembatasan *mass assignment* mungkin belum terlalu jelas fungsinya, karena tujuan dari pengaturan ini adalah untuk pembatasan inputan form.

Kita akan kembali membahas masalah pada bab berikutnya (tentang pemrosesan form). Untuk sementara waktu boleh dipahami bahwa property `$fillable` atau `$guarded` harus ditulis jika ingin menggunakan *mass assignment*.

Kembali ke proses input, saya ingin tambah beberapa data lain ke dalam tabel mahasiswa:

app/Http/Controllers/MahasiswaController.php

```

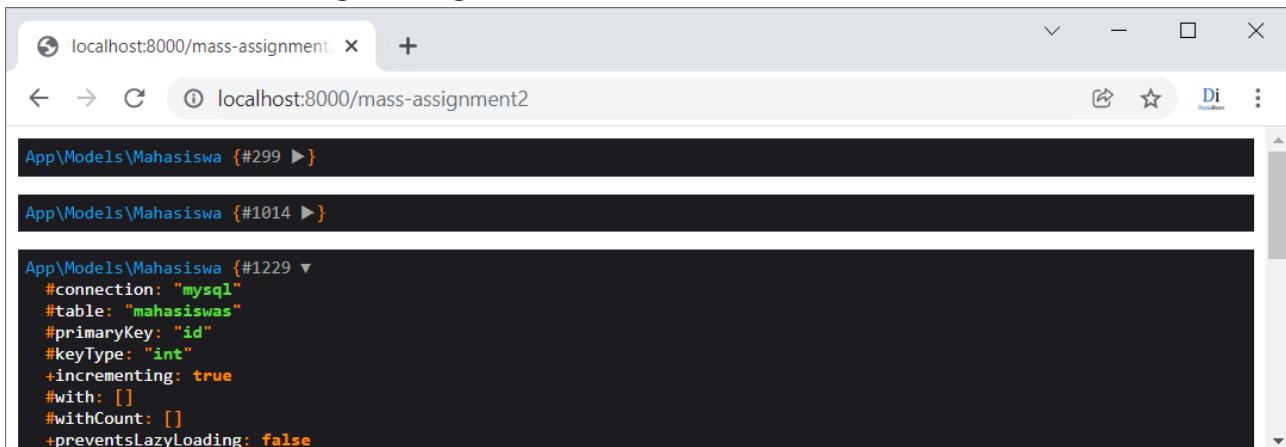
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function massAssignment2(){
6         $mahasiswa1 = Mahasiswa::create(
7             [
8                 'nim' => '19002032',
9                 'nama' => 'Rina Kumala Sari',
10                'tanggal_lahir' => '2000-06-28',
11                'ipk' => 3.4,
12            ]
13        );
14        dump($mahasiswa1);
15    }
}
```

```

16     $mahasiswa2 = Mahasiswa::create(
17         [
18             'nim' => '18012012',
19             'nama' => 'James Situmorang',
20             'tanggal_lahir' => '1999-04-02',
21             'ipk' => 2.7,
22         ]
23     );
24     dump($mahasiswa2);
25
26     $mahasiswa3 = Mahasiswa::create(
27         [
28             'nim' => '19005011',
29             'nama' => 'Riana Putria',
30             'tanggal_lahir' => '2000-11-23',
31             'ipk' => 2.9,
32         ]
33     );
34     dump($mahasiswa3);
35 }
36 }
```

Di sini terdapat pemanggilan 3 buah method `Mahasiswa::create()`, yang artinya ada 3 kali proses input menggunakan *mass assignment*. Method `Mahasiswa::create()` juga mengembalikan object `Mahasiswa`, yang saya simpan ke dalam variabel `$mahasiswa1`, `$mahasiswa2`, dan `$mahasiswa3`.

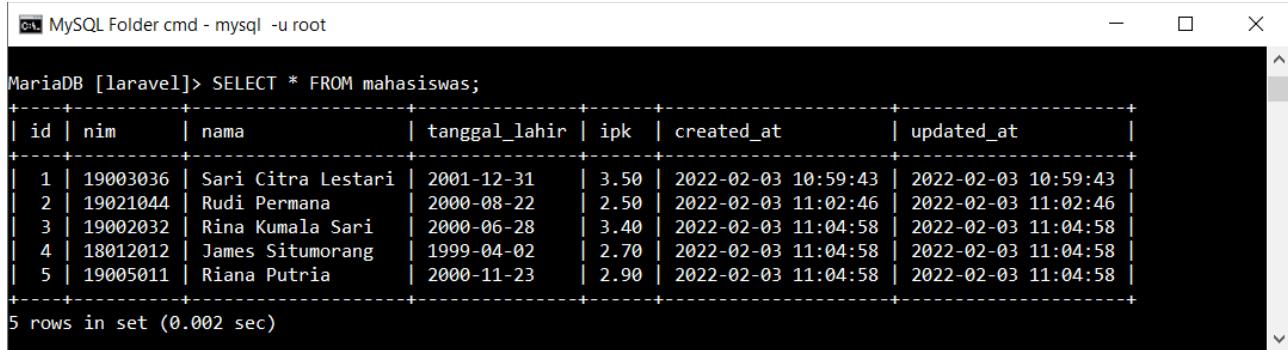
Jalankan method ini dengan mengakses `localhost:8000/mass-assignment2`:



Gambar: Hasil http://localhost:8000/mass-assignment2

Terlihat tampilan detail dari setiap object `Mahasiswa`, yang berasal dari perintah `dump($mahasiswa1)`, `dump($mahasiswa2)` dan `dump($mahasiswa3)`.

Dengan tambahan ini, isi tabel `mahasiswas` sudah bertambah jadi 5 buah data:



The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM mahasiswa;". The output is a table with the following data:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.50	2022-02-03 10:59:43	2022-02-03 10:59:43
2	19021044	Rudi Permana	2000-08-22	2.50	2022-02-03 11:02:46	2022-02-03 11:02:46
3	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 11:04:58	2022-02-03 11:04:58
4	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 11:04:58	2022-02-03 11:04:58
5	19005011	Riana Putria	2000-11-23	2.90	2022-02-03 11:04:58	2022-02-03 11:04:58

5 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswa

16.5. Mengupdate Data

Eloquent ORM memproses data tabel menggunakan object. Untuk proses update, caranya adalah kita harus cari Model object dari tabel, lalu ubah beberapa property dan simpan kembali. Berikut contoh prakteknya:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function update(){
6         $mahasiswa = Mahasiswa::find(1);
7         $mahasiswa->tanggal_lahir = '2001-01-01';
8         $mahasiswa->ipk = 2.9;
9         $mahasiswa->save();
10
11         dump($mahasiswa);
12     }
13 }
```

Di baris 6 terdapat perintah `Mahasiswa::find(1)`. Method `find()` dipakai untuk mencari data tabel berdasarkan kolom `id`. Artinya, perintah `Mahasiswa::find(1)` akan mengambil data di tabel `mahasiswa` yang memiliki `id = 1`, lalu membuat object dari data tersebut. Object ini selanjutnya disimpan ke dalam variabel `$mahasiswa`.

Kemudian di baris 7 dan 8, saya mengisi data '2001-01-01' ke dalam `$mahasiswa->tanggal_lahir`, dan data 2.9 ke dalam `$mahasiswa->ipk`. Kedua nilai ini akan menimpa data yang ada sebelumnya, yakni data dari mahasiswa dengan `id = 1`.

Agar perubahan tersimpan ke database, akhiri dengan perintah `$mahasiswa->save()` seperti di baris 9.

Jalankan kode ini dengan mengakses halaman `localhost:8000/update`:

```
#attributes: array:7 [▶]
#original: array:7 [▶]
#changes: array:3 [▼
    "tanggal_lahir" => "2001-01-01"
    "ipk" => 2.9
    "updated_at" => "2022-02-03 11:06:54"
]
#casts: []
#classCastCache: []
```

Gambar: Hasil http://localhost:8000/update

Dari hasil `dump($mahasiswa)`, silahkan buka tab `#changes`, di dalamnya berisi 3 data yang berubah, yakni `tanggal_lahir`, `ipk`, dan `update_at`. Perhatikan bahwa Eloquent langsung mengupdate kolom `update_at` ketika proses update terjadi, kita tidak perlu melakukan hal ini secara manual.

Agar lebih yakin, mari cek langsung ke tabel `mahasiswas`:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-01-01	2.90	2022-02-03 10:59:43	2022-02-03 11:06:54
2	19021044	Rudi Permana	2000-08-22	2.50	2022-02-03 11:02:46	2022-02-03 11:02:46
3	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 11:04:58	2022-02-03 11:04:58
4	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 11:04:58	2022-02-03 11:04:58
5	19005011	Riana Putria	2000-11-23	2.90	2022-02-03 11:04:58	2022-02-03 11:04:58

Gambar: Isi tabel mahasiswas

Ok, perubahan dari proses update sudah sampai ke database, dimana mahasiswa yang memiliki `id = 1` adalah 'Sari Citra Lestari'.

Selain menggunakan method `find()`, terdapat berbagai method lain yang bisa dipakai untuk mencari data tabel. Mayoritas method ini mirip seperti yang kita pakai di bab **QueryBuilder** dan **Collection**. Salah satunya adalah method `where()`:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function updateWhere(){
6         $mahasiswa = Mahasiswa::where('nim', '19003036')->first();
7         $mahasiswa->tanggal_lahir = '2001-12-31';
8         $mahasiswa->ipk = 4.0;
9         $mahasiswa->save();
10
11         dump($mahasiswa);
12     }
}
```

```
13 }
```

Di baris 6 saya menjalankan method `Mahasiswa::where('nim', '19003036')->first()`. Perintah ini akan mencari mahasiswa dengan kolom `nim` yang berisi nilai `19003036`. Tambahan method `first()` perlu ditulis karena method `where()` akan menghasilkan sebuah collection. Setelah object di dapat, update `tanggal_lahir` dan `ipk` dari mahasiswa tersebut dan simpan kembali dengan method `save()`. Jalankan dengan mengakses halaman `localhost:8000/update-where`:

```
#attributes: array:7 [▶]
#original: array:7 [▶]
#changes: array:3 [▼
    "tanggal_lahir" => "2001-12-31"
    "ipk" => 4.0
    "updated_at" => "2022-02-03 11:08:37"
]
#casts: []
#classCastCache: []
```

Gambar: Hasil http://localhost:8000/update-where

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	4.00	2022-02-03 10:59:43	2022-02-03 11:08:37
2	19021044	Rudi Permana	2000-08-22	2.50	2022-02-03 11:02:46	2022-02-03 11:02:46
3	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 11:04:58	2022-02-03 11:04:58
4	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 11:04:58	2022-02-03 11:04:58
5	19005011	Riana Putria	2000-11-23	2.90	2022-02-03 11:04:58	2022-02-03 11:04:58

Gambar: Isi tabel mahasiswa

Ternyata mahasiswa yang memiliki nim `19003036` masih 'Sari Citra Lestari', yang sekarang nilai `tanggal_lahir` sudah berubah menjadi `2001-12-31` dan `ipk` menjadi `4.00`.

Mass Update

Proses update juga bisa dilakukan menggunakan teknik *mass update*. Caranya mirip seperti *mass assignment*, tapi kali ini kita menggunakan method `update()`:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function massUpdate(){
6         Mahasiswa::where('nim', '19003036')->first()->update([
7             'tanggal_lahir' =>'2000-04-20',
8             'ipk' => 2.1
```

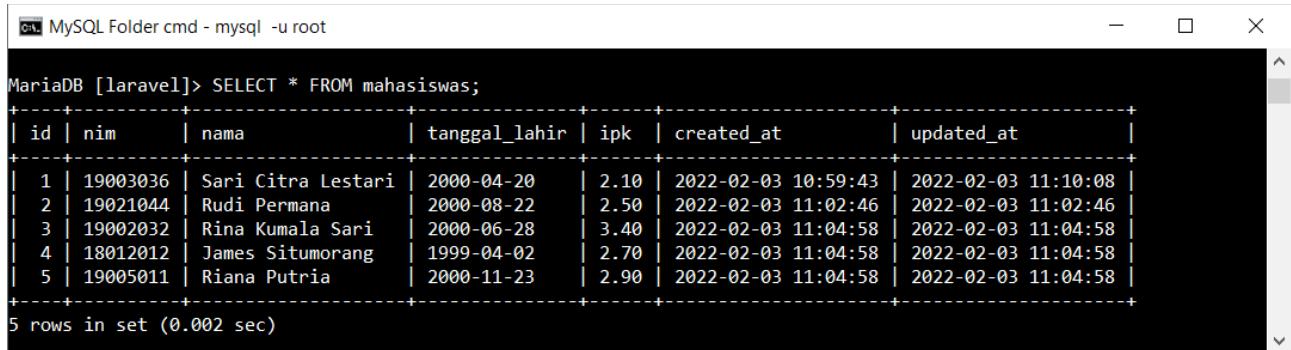
```

9         ]);
10        return "Berhasil di proses";
11    }
12 }

```

Di sini saya kembali menggunakan method `Mahasiswa::where('nim','19003036')->first()` untuk mencari mahasiswa yang ingin di update, kemudian langsung di-chaining dengan method `update()` yang berisi associative array dari data yang akan di ubah.

Jalankan method `massUpdate()` ini dengan membuka localhost:8000/mass-update. Hasilnya, kolom `tanggal_lahir`, `ipk` dan `update_at` untuk 'Sari Citra Lestari' sudah kembali berubah:



The screenshot shows a terminal window titled 'MySQL Folder cmd - mysql -u root'. The command run is 'SELECT * FROM mahasiswa;'. The output is a table with columns: id, nim, nama, tanggal_lahir, ipk, created_at, updated_at. The data includes rows for Sari Citra Lestari, Rudi Permana, Rina Kumala Sari, James Situmorang, and Riana Putria. The 'updated_at' column shows the timestamp '2022-02-03 11:04:58' for all rows except the first one, which has a different timestamp.

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2000-04-20	2.10	2022-02-03 10:59:43	2022-02-03 11:10:08
2	19021044	Rudi Permana	2000-08-22	2.50	2022-02-03 11:02:46	2022-02-03 11:02:46
3	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 11:04:58	2022-02-03 11:04:58
4	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 11:04:58	2022-02-03 11:04:58
5	19005011	Riana Putria	2000-11-23	2.90	2022-02-03 11:04:58	2022-02-03 11:04:58

5 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswas

Teknik *mass update* juga butuh pengaturan property `$fillable` atau `$guarded` di file model `Mahasiswa.php`. Jika anda menemukan error, cek apakah salah satu dari property ini sudah ditambahkan atau belum.

16.6. Menghapus Data

Proses menghapus data dengan Eloquent sangat *simple*, cukup dengan mengakses method `delete()` dari Model object. Namun sebelum itu kita harus cari terlebih dahulu Model object yang ingin dihapus.

Berikut contoh menghapus data menggunakan Eloquent ORM:

app/Http/Controllers/MahasiswaController.php

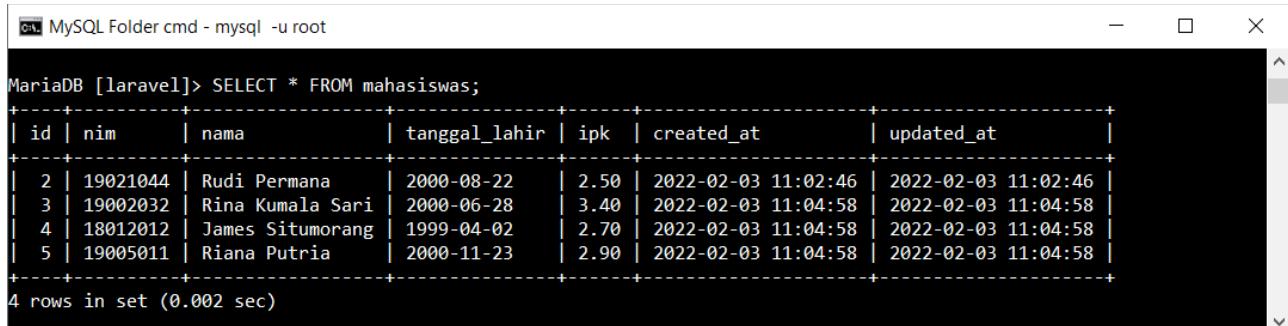
```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function delete(){
6         $mahasiswa = Mahasiswa::find(1);
7         $mahasiswa->delete();
8
9         dump($mahasiswa);
10    }
11 }

```

Di baris 5 saya menggunakan method `find(1)` untuk mencari data mahasiswa yang memiliki `id = 1`. Setelah object di dapat, tinggal jalankan method `delete()` seperti di baris 7.

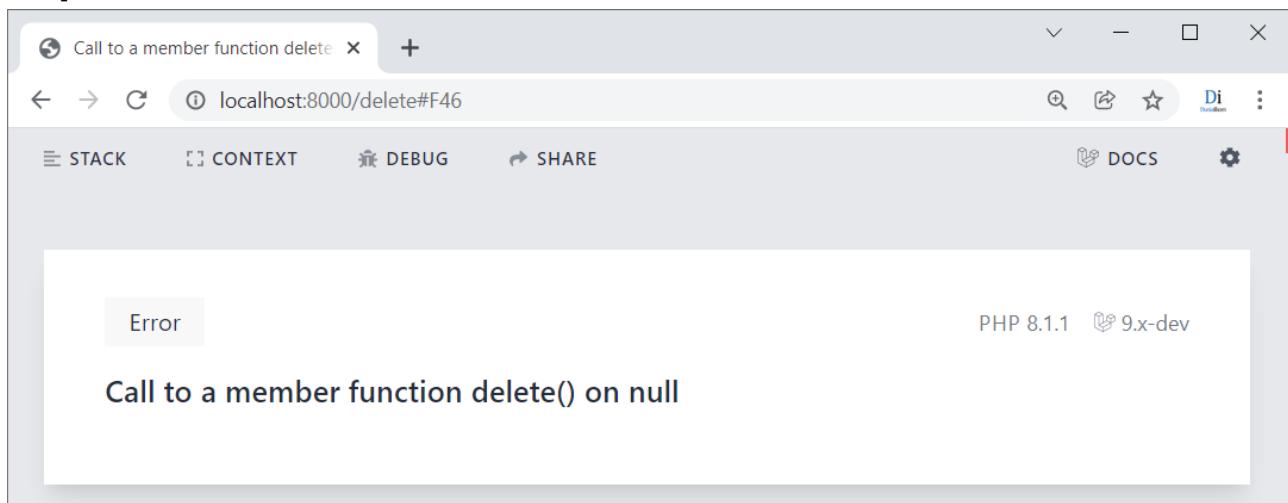
Jalankan kode di atas dengan membuka halaman `localhost:8000/delete`. Hasilnya, data mahasiswa 'Sari Citra Lestari' sudah tidak ada lagi.



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim   | nama        | tanggal_lahir | ipk    | created_at      | updated_at      |
+----+-----+-----+-----+-----+-----+
| 2  | 19021044 | Rudi Permana | 2000-08-22   | 2.50   | 2022-02-03 11:02:46 | 2022-02-03 11:02:46 |
| 3  | 19002032 | Rina Kumala Sari | 2000-06-28   | 3.40   | 2022-02-03 11:04:58 | 2022-02-03 11:04:58 |
| 4  | 18012012 | James Situmorang | 1999-04-02   | 2.70   | 2022-02-03 11:04:58 | 2022-02-03 11:04:58 |
| 5  | 19005011 | Riana Putria   | 2000-11-23   | 2.90   | 2022-02-03 11:04:58 | 2022-02-03 11:04:58 |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa

Sedikit percobaan, silahkan akses kembali halaman `localhost:8000/delete`. Ternyata akan tampil error berikut:



Gambar: Hasil error dari halaman `localhost:8000/delete`

Error ini terjadi karena method `Mahasiswa::find(1)` tidak bisa menemukan data mahasiswa dengan `id = 1`, yang memang sudah kita hapus sebelumnya.

Karena tidak ditemukan, method `Mahasiswa::find(1)` akan mengembalikan nilai `null` ke dalam variabel `$mahasiswa`. Pengaksesan method `delete()` dari nilai `null` inilah yang menyebabkan error di atas. Idealnya, sebelum menghapus data kita bisa cek terlebih dahulu apakah data tersebut memang ada atau tidak.

Dalam beberapa kasus, error ini bisa dibiarkan saja karena ketika sudah masuk ke production, kita bisa set pengaturan Laravel agar setiap halaman error menampilkan pesan **500 | Server Error**, seperti yang dibahas dalam bab **Error Display**.

Atau jika tidak ingin tampil error, bisa menggunakan static method `Model::destroy()` untuk

menghapus data tabel. Method `destroy()` ini butuh sebuah argument berupa `id` dari data yang ingin dihapus:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function destroy(){
6         $mahasiswa = Mahasiswa::destroy(1);
7         dump($mahasiswa);
8     }
9 }
```



Gambar: Hasil halaman localhost:8000/destroy

Perintah `Mahasiswa::destroy(1)` akan menghapus data dari tabel `mahasiswas` yang memiliki kolom `id = 1`. Jika tidak ditemukan, method ini mengembalikan nilai 0 dan tidak terjadi error.

Method `destroy()` juga bisa diisi dengan array atau `collection` yang terdiri dari kumpulan `id` seperti contoh berikut:

```

Mahasiswa::destroy([3, 9, 10]);
Mahasiswa::destroy(collect([3, 9, 10]));
```

Jika method di atas dijalankan, maka data id 3, 9 dan 10 akan dihapus dari tabel `mahasiswas`.

Mass Delete

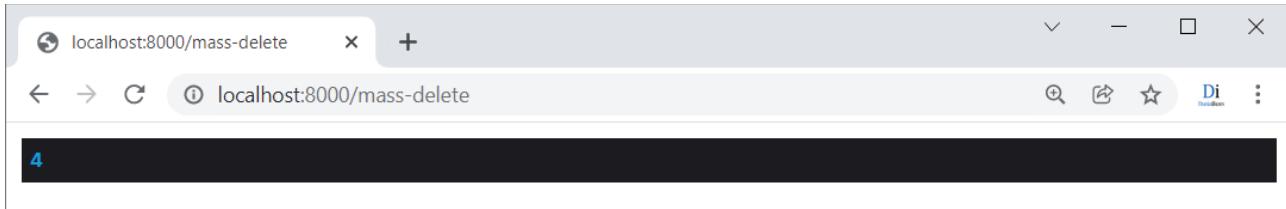
Mass delete adalah sebutan untuk cara menghapus data dari kumpulan Model object.

Kumpulan model object ini didapat dari hasil pencarian method `where()` seperti contoh berikut:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function massDelete(){
6         $mahasiswa = Mahasiswa::where('ipk', '>', 2)->delete();
7         dump($mahasiswa);
8     }
9 }
```



Gambar: Hasil halaman localhost:8000/mass-delete

Di baris 6 saya menjalankan method `Mahasiswa::where('ipk', '>', 2)` yang dipakai untuk mencari semua mahasiswa dengan ipk lebih dari 2. Untuk semua data ini, langsung di chaining dengan method `delete()`.

Hasilnya, setelah halaman `localhost:8000/mass-delete` di akses, isi tabel `mahasiswas` akan kosong karena semua mahasiswa memenuhi syarat `ipk > 2`.

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswas;
Empty set (0.928 sec)
MariaDB [laravel]> -
```

Gambar: Isi tabel mahasiswas

16.7. Menampilkan Data

Menampilkan data dengan **Eloquent ORM** artinya kita akan banyak berurusan dengan `object` dan `collection`. Beberapa method mengembalikan 1 Model object saja, namun sebagian method lain akan mengembalikan `collection` yang terdiri dari banyak Model object. Kuncinya adalah, periksa hasil dari setiap method sebelum mengakses isinya.

Sebelum masuk ke praktek, saya ingin mengisi kembali tabel `mahasiswas` karena sudah kosong dari praktek tentang method `delete()` dan `destroy()`. Caranya, akses halaman `localhost:8000/insert`, `localhost:8000/mass-assignment` dan `localhost:8000/mass-assignment2`. Semua halaman ini sudah kita buat ketika membahas cara input data di awal bab.

Berikut isi tabel `mahasiswas` setelah menjalankan 3 halaman di atas:

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+
| id | nim   | nama        | tanggal_lahir | ipk  | created_at      | updated_at    |
+----+-----+-----+-----+-----+-----+
| 6  | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | 2022-02-03 11:14:40 | 2022-02-03 11:14:40 |
| 7  | 19021044 | Rudi Permana   | 2000-08-22 | 2.50 | 2022-02-03 11:14:43 | 2022-02-03 11:14:43 |
| 8  | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.40 | 2022-02-03 11:14:46 | 2022-02-03 11:14:46 |
| 9  | 18012012 | James Situmorang | 1999-04-02 | 2.70 | 2022-02-03 11:14:46 | 2022-02-03 11:14:46 |
| 10 | 19005011 | Riana Putria   | 2000-11-23 | 2.90 | 2022-02-03 11:14:46 | 2022-02-03 11:14:46 |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswas

Perhatikan kolom `id` yang dimulai dari angka 6. Di MySQL, kolom auto increment akan terus bertambah 1 angka meskipun baris data sebelumnya sudah dihapus.

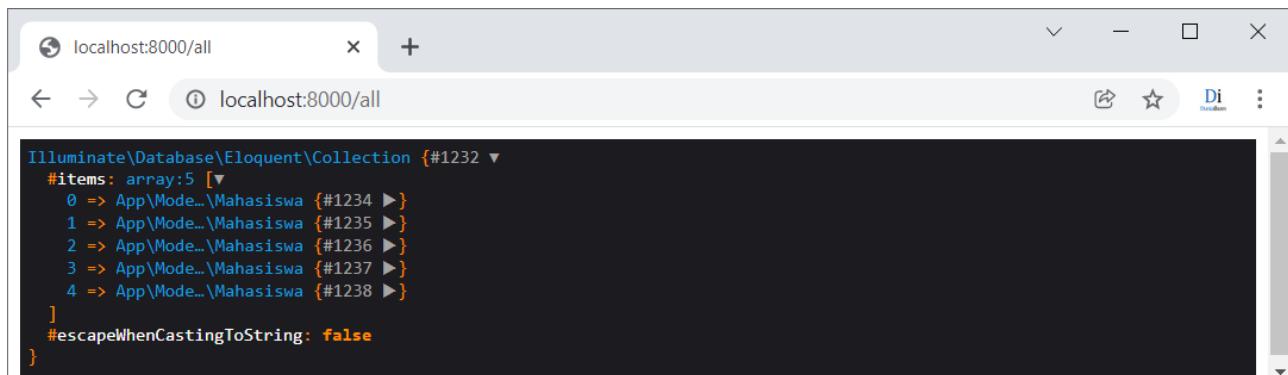
Method All()

Method pertama yang akan kita bahas adalah `all()`. Method ini berfungsi untuk mengambil semua data dari dalam tabel:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function all(){
6         $result = Mahasiswa::all();
7         dump($result);
8     }
9 }
```



Gambar: Hasil halaman localhost:8000/all

Di baris 6, hasil pemanggilan method `Mahasiswa::all()` saya simpan ke variabel `$result`, untuk kemudian di `dump()`. Terlihat bahwa `$result` ini berisi sebuah **collection** dari kumpulan Model object `Mahasiswa`. Karena di tabel `mahasiswas` ada 5 data, maka collection `$result` juga berisi 5 buah object.

Pemilihan nama variabel `$result` memperlihatkan kita bebas ingin menampung hasil Eloquent dengan nama variabel apa saja. Namun agar mengikuti konsep *plural* dan *singular*, biasanya ini ditampung ke dalam variabel `$mahasiswas` jika isinya terdiri dari collection, atau `$mahasiswa` jika isinya terdiri dari 1 object saja.

Kita sudah membahas tentang collection sebelumnya, karena itu bisakah anda membuat kode program untuk menampilkan data dari 1 mahasiswa hasil dari method `all()`? Silahkan coba rancang sebentar.

Baik, berikut kode yang saya pakai:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function all(){
6         $result = Mahasiswa::all();
7
8         echo($result[0]->id). "<br>";
9         echo($result[0]->nim). "<br>";
10        echo($result[0]->nama). "<br>";
11        echo($result[0]->tanggal_lahir). "<br>";
12        echo($result[0]->ipk);
13    }
14 }
```



Gambar: Menampilkan 1 data dengan Eloquent

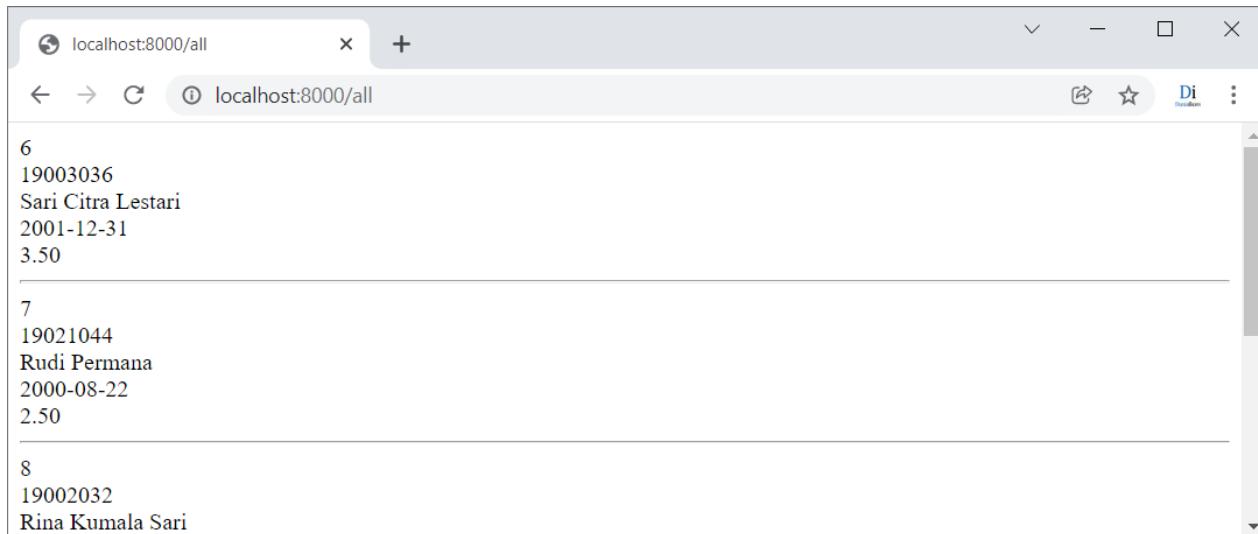
Variabel `$result` berisi collection dari object `Mahasiswa`, maka perintah `$result[0]->id` akan mengakses nilai kolom `id` milik mahasiswa yang berada di urutan paling awal collection (key ke-0). Jika ingin mengakses data berikutnya, bisa menggunakan `$result[1]`, `$result[2]`, dst.

Lanjut, bisakah anda membuat perulangan `foreach` untuk menampilkan semua data mahasiswa? Materi ini sangat mirip seperti yang pernah kita bahas di bab tentang collection. Berikut kode yang saya pakai:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function all(){
6         $result = Mahasiswa::all();
7
8         foreach ($result as $mahasiswa) {
9             echo($mahasiswa->id). "<br>";
10            echo($mahasiswa->nim). "<br>";
11            echo($mahasiswa->nama). "<br>";
12            echo($mahasiswa->tanggal_lahir). "<br>";
13            echo($mahasiswa->ipk). "<br>";
14            echo "<hr>";
15        }
16    }
17 }
```



Gambar: Menampilkan semua data dengan perulangan foreach

Untuk menampilkan semua data mahasiswa, saya menulis perulangan `foreach ($result as $mahasiswa)`. Di dalam perulangan, variabel `$mahasiswa` akan berisi object dari setiap model Mahasiswa yang ada di dalam collection. Jika anda merasa bingung dengan kode program ini, bisa buka sejenak bab tentang collection.

Mengirim Data ke View

Menampilkan data langsung di controller terasa kurang pas, karena seharusnya data ini dikirim ke view terlebih dahulu. Kita juga sudah memiliki view `tampil-mahasiswa` yang berasal dari bab Query Builder. Jika belum, silahkan buat view `tampil-mahasiswa.blade.php` dengan kode berikut:

`resources/views/tampil-mahasiswa.blade.php`

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link href="{{ asset('/css/bootstrap.min.css') }}" rel="stylesheet">
8      <title>Data Mahasiswa</title>
9  </head>
10 <body>
11
12 <div class="container text-center p-4">
13     <h1 class="mb-3">Data Mahasiswa</h1>
14     <div class="row">
15         <div class="m-auto">
16             <ol class="list-group">
17                 @forelse ($mahasiswa as $mahasiswa)
18                     <li class="list-group-item">
19                         {{$mahasiswa->nama}} ( {{$mahasiswa->nim}} ),
20                         Tanggal Lahir: {{$mahasiswa->tanggal_lahir}},

```

```

21         IPK: {{$mahasiswa->ipk}}
22     </li>
23     @empty
24         <div class="alert alert-dark d-inline-block">Tidak ada data...</div>
25     @endforelse
26     </ol>
27   </div>
28 </div>
29 </div>
30
31 </body>
32 </html>

```

Inti dari view ini ada di baris 17 – 22, yakni perulangan `@forelse` untuk menampilkan isi dari variabel `$mahasiswas`. Perhatikan bahwa isi variabel `$mahasiswas` harus berbentuk array atau collection, jika berbentuk tipe data lain, maka perulangan `@forelse` akan menghasilkan error.

Berikut contoh kode program untuk mengirim data Eloquent ke dalam view ini:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function allView(){
6         $mahasiswas = Mahasiswa::all();
7         return view('tampil_mahasiswa',['mahasiswas' => $mahasiswas]);
8     }
9 }

```

Hasil dari `Mahasiswa::all()` sudah berbentuk collection, sehingga bisa langsung di teruskan ke dalam view `tampil_mahasiswa` dengan cara menulis `['mahasiswas' => $mahasiswas]`.

Sari Citra Lestari (19003036), Tanggal Lahir: 2001-12-31, IPK: 3.50
Rudi Permana (19021044), Tanggal Lahir: 2000-08-22, IPK: 2.50
Rina Kumala Sari (19002032), Tanggal Lahir: 2000-06-28, IPK: 3.40
James Situmorang (18012012), Tanggal Lahir: 1999-04-02, IPK: 2.70
Riana Putria (19005011), Tanggal Lahir: 2000-11-23, IPK: 2.90

Gambar: Data dari Eloquent ditampilkan di view

Sip, kita sudah bisa menampilkan data tabel MySQL yang diakses menggunakan Eloquent ORM ke dalam view.

Method where()

Untuk mengambil sebagian data dari tabel, kita bisa memakai method `where()`. Cara penggunaannya sama seperti method `where()` di materi Collection dan Query Builder. Hasil dari method ini bisa di *chaining* untuk kombinasi pembatasan lebih lanjut.

Secara internal, eloquent juga menggunakan *query builder*, sehingga hampir semua method yang ada di *query builder* juga bisa diterapkan ke eloquent.

Sebagai contoh, saya ingin menampilkan semua mahasiswa dengan nilai `ipk < 3`, dan di urutkan berdasarkan `nama` secara menurun. Berikut kode programnya:

`app/Http/Controllers/MahasiswaController.php`

```

1  <?php
2
3  class MahasiswaController extends Controller
4  {
5      public function getWhere(){
6          $mahasiswas = Mahasiswa::where('ipk', '<', '3')
7              ->orderBy('nama', 'desc')
8              ->get();
9          return view('tampil-mahasiswa',[ 'mahasiswas' => $mahasiswas]);
10     }
11 }
```

Di sini saya men-chaining beberapa method sekaligus. Pertama, method `Mahasiswa::where('ipk', '<', '3')` untuk mencari data mahasiswa dengan `ipk < 3`, lalu method `orderBy('nama', 'desc')` untuk menguratkannya berdasarkan kolom `nama` secara menurun, serta method `get()` untuk mengambil hasil data dalam bentuk collection.

Collection ini kemudian disimpan ke dalam variabel `$mahasiswas` untuk dikirim ke view:



Gambar: Hasil pembatasan data menggunakan method `where()`, `orderBy()` dan `get()`

Sedikit catatan, jika kita memakai method `where()` di dalam Eloquent, maka harus ada method lain untuk mengaksesnya, seperti `get()`. Jika hanya menulis sampai `where()` saja, maka itu akan menghasilkan `Builder` class, bukan `collection` dari class `Model`:

```
$mahasiswa = Mahasiswa::where('ipk', '<', '3');
```

Jika ditulis seperti ini, variabel `$mahasiswa` akan berisi `Builder` class, yakni class internal Laravel untuk memproses Eloquent. Yang seharusnya kita tulis adalah:

```
$mahasiswa = Mahasiswa::where('ipk', '<', '3')->get();
```

Tambahan method `get()` di akhir ini kadang sering lupa ditulis, sehingga bisa terjadi error.

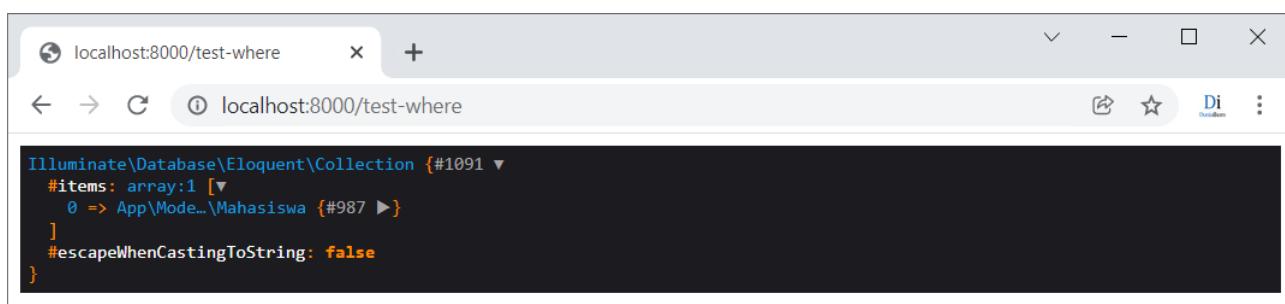
Method `first()`

Ketika kita membuat batasan menggunakan method `where()->get()`, hasilnya adalah sebuah `collection`, meskipun itu hanya terdiri dari 1 object saja. Method `first()` bisa dipakai sebagai pengganti method `get()` untuk mengambil element pertama dari hasil batasan `where()` ini.

Perhatikan contoh berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function testWhere(){
6         $mahasiswa = Mahasiswa::where('nim', '18012012')->get();
7         dump($mahasiswa);
8     }
9 }
```



Gambar: Hasil halaman localhost:8000/test-where

Ketika merancang migration, kolom `nim` saya set dengan key `unique`, sehingga isi kolom tidak bisa diisi data yang berulang (tidak boleh ada `nim` yang sama). Ketika menjalankan method `Mahasiswa::where('nim', '18012012')->get()`, seharusnya hasil yang didapat hanya 1 mahasiswa saja.

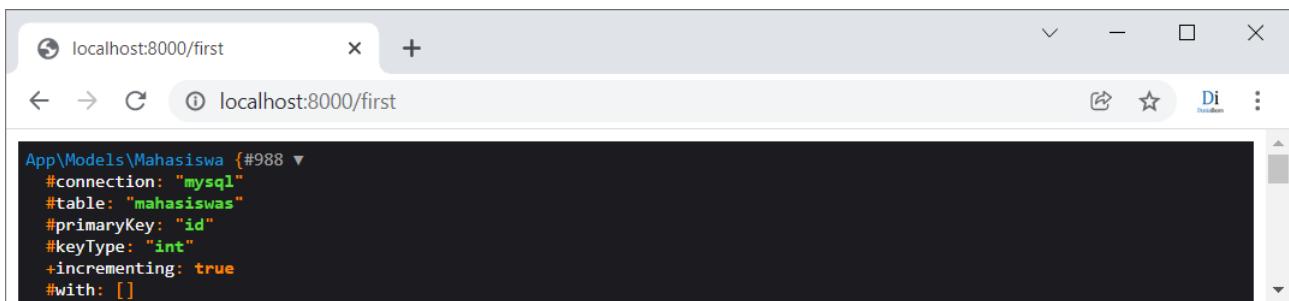
Namun bawaan dari Eloquent, ketika method `where()` disambung dengan `get()`, hasilnya tetap

berbentuk collection, bukan 1 object model Mahasiswa. Jika yang kita inginkan adalah 1 object saja, bisa memakai method `first()` sebagai penganti `get()`:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function first(){
6         $mahasiswa = Mahasiswa::where('nim','18012012')->first();
7         dump($mahasiswa);
8     }
9 }
```



Gambar: Hasil halaman localhost:8000/first

Di sini, variabel `$mahasiswa` akan berisi 1 object dari model `Mahasiswa`, bukan lagi dalam bentuk collection.

Method `first()` juga bisa dipakai untuk kondisi `where()` yang menghasilkan banyak object, misalnya seperti contoh berikut:

```
$mahasiswa = Mahasiswa::where('ipk','<','3')->first();
```

Variabel `$mahasiswa` tetap berisi 1 object, meskipun kondisi `Mahasiswa::where('ipk','<','3')` bisa saja cocok dengan banyak data mahasiswa. Method `first()` hanya mengambil object pertama yang ada di dalam hasil `Mahasiswa::where('ipk','<','3')`.

Yang juga harus diperhatikan, terdapat perbedaan cara memproses data jika hasil Eloquent berbentuk collection atau 1 object saja. Jika itu adalah collection, maka kita harus 'buka' terlebih dahulu. Berikut perbandingan antara keduanya:

```

1 $mahasiswa = Mahasiswa::where('nim','18012012')->get();
2 echo $mahasiswa[0]->nama;      // James Situmorang
3
4 $mahasiswa = Mahasiswa::where('nim','18012012')->first();
5 echo $mahasiswa->nama;        // James Situmorang
```

Di baris 1 saya memakai method `get()`, sehingga variabel `$mahasiswa` berisi collection. Maka cara mengakses nama mahasiswa adalah dengan perintah `$mahasiswa[0]->nama`. Tambahan `[0]` harus ditulis karena yang kita akses adalah element pertama dari collection (index ke-0).

Di baris 4, saya menggunakan method `first()`, maka variabel `$mahasiswa` langsung berisi 1 object model `Mahasiswa`. Sehingga untuk mengakses kolom nama, tinggal langsung menulis perintah `$mahasiswa->nama`.

Hal sebaliknya juga bisa terjadi, dimana kode program kita malah minta sebuah collection. Misalnya untuk mengirim hasil `first()` ke dalam view `tampil-mahasiswa`, saya harus menggunakan sedikit trik:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function first(){
6         $mahasiswa = Mahasiswa::where('ipk', '<', '3')->first();
7         return view('tampil-mahasiswa',['mahasiswas' => [$mahasiswa]]);
8     }
9 }
```

Di dalam view `tampil-mahasiswa`, terdapat perulangan `@forelse ($mahasiswas as $mahasiswa)` yang dipakai untuk menampilkan semua data mahasiswa. Ini berarti view **harus** menerima data dalam bentuk collection atau array, tidak bisa dalam bentuk 1 object langsung seperti hasil dari method `first()`.

Untuk mengakalinya, pada saat mengirim variabel `$mahasiswas` ke view, hasil dari method `first()` saya tempatkan di dalam tanda kurung siku, supaya variabel `$mahasiswas` dikirim dalam bentuk array, yakni perintah `['mahasiswas' => [$mahasiswa]]` di baris 7.

Tanpa trik seperti ini, view akan mengalami error karena melakukan perulangan `@forelse` ke sebuah object. Alternatif lain, kita bisa membuat 2 buah view terpisah, satu untuk menangani data berbentuk collection, dan satu view lagi untuk data dalam bentuk object langsung.

Method `find()`

Method `find()` bisa dipakai untuk mencari data Model berdasarkan kolom `id`. Hasil dari method ini langsung berbentuk object, yang sama seperti method `first()`. Berikut contoh penggunaannya:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function find(){
6         $mahasiswa = Mahasiswa::find(8);
7         return view('tampil-mahasiswa',['mahasiswas' => [$mahasiswa]]);
8     }
9 }
```



Gambar: Hasil halaman http://localhost:8000/find

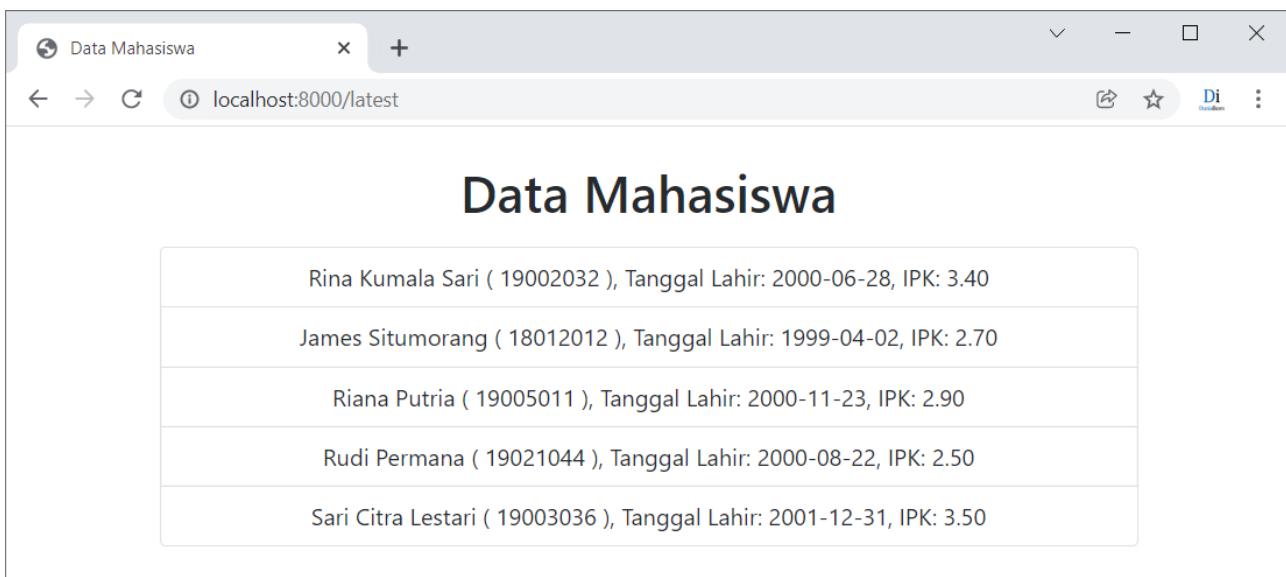
Perintah `Mahasiswa::find(8)` akan mencari mahasiswa yang memiliki `id = 8`. Karena ini adalah 1 object (bukan collection) proses pengiriman data ke view `tampil-mahasiswa` juga harus saya tulis dalam tanda kurung siku agar menjadi array.

Method latest()

Method `latest()` berguna untuk mengambil collection yang sudah di urutkan berdasarkan tanggal pembuatan secara menurun, data paling akhir yang diinput akan ada di urutan pertama. Berikut contoh penggunaannya:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function latest(){
6         $mahasiswa = Mahasiswa::latest()->get();
7         return view('tampil-mahasiswa',['mahasiswas' => $mahasiswa]);
8     }
9 }
```



Gambar: Hasil halaman http://localhost:8000/latest

Apa yang dijalankan oleh method `latest()` ini sama dengan query `SELECT * FROM mahasiswa ORDER BY created_at DESC;`

```
C:\ MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswa ORDER BY created_at DESC;
+----+-----+-----+-----+-----+-----+
| id | nim  | nama        | tanggal_lahir | ipk   | created_at      | updated_at    |
+----+-----+-----+-----+-----+-----+
| 8  | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.40 | 2022-02-03 11:14:46 | 2022-02-03 11:14:46 |
| 9  | 18012012 | James Situmorang | 1999-04-02 | 2.70 | 2022-02-03 11:14:46 | 2022-02-03 11:14:46 |
| 10 | 19005011 | Riana Putria    | 2000-11-23 | 2.90 | 2022-02-03 11:14:46 | 2022-02-03 11:14:46 |
| 7  | 19021044 | Rudi Permana    | 2000-08-22 | 2.50 | 2022-02-03 11:14:43 | 2022-02-03 11:14:43 |
| 6  | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | 2022-02-03 11:14:40 | 2022-02-03 11:14:40 |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa yang diurutkan berdasarkan kolom `created_at`

Method `limit()`

Method `limit()` bisa dipakai untuk membatasi hasil collection. Method ini butuh sebuah argument berupa angka. Sebagai contoh, `limit(1)` hanya akan mengambil 1 object, sedangkan `limit(3)` akan mengambil 3 object teratas dari collection.

Berikut contoh penggunaannya:

`app/Http/Controllers/MahasiswaController.php`

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function limit(){
6         $mahasiswa = Mahasiswa::latest()->limit(2)->get();
7         return view('tampil-mahasiswa',['mahasiswa' => $mahasiswa]);
8     }
9 }
```



Gambar: Hasil halaman `localhost:8000/limit`

Perintah `Mahasiswa::latest()->limit(2)->get()` dipakai untuk menampilkan 2 nama mahasiswa yang terakhir diinput.

Sama seperti perintah SQL biasa, terdapat banyak cara untuk menampilkan sebuah data. Hasil yang sama juga bisa didapat menggunakan perintah berikut:

```
$mahasiswa = Mahasiswa::orderBy('created_at', 'desc')->limit(2)->get();
```

Kali ini saya mengganti method `latest()` dengan `orderBy('created_at', 'desc')`.

Method skip() dan take()

Kedua method ini sudah pernah kita bahas pada materi menampilkan data menggunakan query builder. Method `skip()` berfungsi untuk melompati beberapa data, sedangkan method `take()` dipakai untuk mengambil sejumlah data tabel. Kedua method ini butuh argument berupa angka. Berikut contoh penggunaan dari `skip()` dan `take()`:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function skipTake(){
6         $mahasiswa = Mahasiswa::orderBy('ipk')->skip(1)->take(3)->get();
7         return view('tampil-mahasiswa',['mahasiswas' => $mahasiswa]);
8     }
9 }
```



Gambar: Hasil halaman <http://localhost:8000/skip-take>

Perintah di baris 6 bisa dibaca: Urutkan tabel `mahasiswas` berdasarkan `ipk` (dari nilai terendah), lewatkan 1 data paling atas, lalu ambil 3 data setelahnya. Jika menggunakan perintah SQL, ini sama dengan perintah `SELECT * FROM mahasiswas ORDER BY ipk LIMIT 1,3`.

16.8. Soft Delete

Bawaan dari MySQL, sebuah data yang dihapus dengan query `DELETE` tidak bisa dikembalikan lagi (tidak ada fitur *undo*). Ini kadang jadi masalah jika user pengguna aplikasi kita tidak

sengaja menghapus data penting.

Eloquent memberikan solusi untuk hal ini, yang disebut sebagai **Soft Delete**. Teknik yang digunakan adalah, eloquent tidak benar-benar menghapus data dari dalam tabel, tapi menambah satu kolom khusus sebagai penanda data yang telah dihapus.

Ketika ditampilkan, misalnya dengan method `all()`, data ini tidak akan terlihat. Jika di lain waktu kita berubah pikiran, data tersebut bisa dikembalikan ke kondisi semula, yakni dengan cara menghapus kolom penanda tadi.

Ada beberapa syarat agar bisa menggunakan **soft delete**. Pertama, sebuah tabel harus memiliki kolom `delete_at`. Untuk membuatnya silahkan buka kembali file migration dari tabel `mahasiswa` lalu tambah perintah `$table->softDeletes()` ke dalam struktur tabel:

`database/migrations/<timestampl>_create_mahasiswa_table.php`

```

1 public function up()
2 {
3     Schema::create('mahasiswa', function (Blueprint $table) {
4         $table->id();
5         $table->char('nim', 8)->unique();
6         $table->string('nama');
7         $table->date('tanggal_lahir');
8         $table->decimal('ipk', 3, 2)->default(1.00);
9         $table->timestamps();
10        $table->softDeletes();
11    });
12 }
```

Method `$table->softDeletes()` adalah method khusus yang disediakan Laravel untuk membuat kolom `delete_at`. Pada kode di atas, perintah ini ada di baris 10.

Supaya struktur tabel `mahasiswa` berubah, kita harus *rollback* dan jalankan kembali proses **migration**. Kedua langkah ini bisa dilakukan dengan perintah `php artisan migrate:fresh`.

```

C:\xampp\htdocs\laravel01>php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (716.48ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (450.99ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (642.94ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (883.23ms)
Migrating: 2022_02_01_152542_create_mahasiswa_table
Migrated: 2022_02_01_152542_create_mahasiswa_table (1,099.30ms)

C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan ulang proses migration

Selanjutnya silahkan akses halaman `localhost:8000/insert`, `localhost:8000/mass-`

assignment dan localhost:8000/mass-assignment2 untuk mengisi kembali tabel mahasiswa. Setelah itu, cek isi tabel:

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at | deleted_at |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.50 | 2022-02-03 12:10:27 | 2022-02-03 12:10:27 | NULL |
| 2 | 19021044 | Rudi Permana | 2000-08-22 | 2.50 | 2022-02-03 12:10:34 | 2022-02-03 12:10:34 | NULL |
| 3 | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.40 | 2022-02-03 12:10:38 | 2022-02-03 12:10:38 | NULL |
| 4 | 18012012 | James Situmorang | 1999-04-02 | 2.70 | 2022-02-03 12:10:38 | 2022-02-03 12:10:38 | NULL |
| 5 | 19005011 | Riana Putria | 2000-11-23 | 2.90 | 2022-02-03 12:10:38 | 2022-02-03 12:10:38 | NULL |
+----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa dengan tambahan kolom delete_at

Di sisi paling kanan, terdapat kolom `delete_at`. Inilah kolom yang di-generate oleh method `$table->softDeletes()` dari file migration. Saat ini nilainya `NULL` untuk semua baris tabel.

Syarat kedua agar bisa menggunakan **soft delete** adalah, import class `Illuminate\Database\Eloquent\SoftDeletes` ke dalam Model. Berikut isi model `Mahasiswa` dengan penambahan perintah ini:

App\Models\Mahasiswa.php

```
1 <?php
2 namespace App\Models;
3
4 use Illuminate\Database\Eloquent\Factories\HasFactory;
5 use Illuminate\Database\Eloquent\Model;
6 use Illuminate\Database\Eloquent\SoftDeletes;
7
8 class Mahasiswa extends Model
9 {
10     use HasFactory;
11     protected $guarded = [];
12     use SoftDeletes;
13 }
```

Terdapat 2 kode tambahan, yakni di baris 6 untuk import class `SoftDeletes`, dan di baris 12 untuk menggunakan class ini.

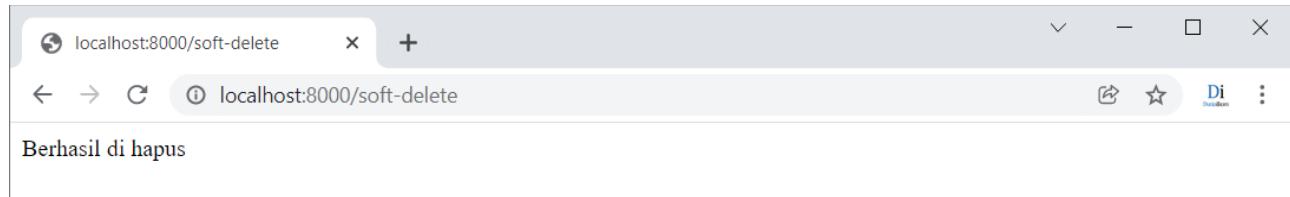
Persiapan kita sudah selesai, sekarang saya akan coba menghapus salah satu data mahasiswa:

app\Http\Controllers\MahasiswaController.php

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function softDelete(){
6         Mahasiswa::where('nim','18012012')->delete();
7         return "Berhasil di hapus";
}
```

```
8     }
9 }
```

Data yang akan saya hapus adalah mahasiswa dengan nim = 18012012. Jalankan method ini dengan mengakses route localhost:8000/soft-delete.



Gambar: Hasil halaman http://localhost:8000/soft-delete

Proses penghapusan berhasil. Mari kita cek ke database:

Select MySQL Folder cmd - mysql -u root							
MariaDB [laravel]> SELECT * FROM mahasiswas;							
+-----+-----+-----+-----+-----+-----+-----+-----+	id nim nama tanggal_lahir ipk created_at updated_at deleted_at	+-----+-----+-----+-----+-----+-----+-----+-----+					
1 19003036 Sari Citra Lestari 2001-12-31 3.50 2022-02-03 12:10:27 2022-02-03 12:10:27 NULL							
2 19021044 Rudi Permana 2000-08-22 2.50 2022-02-03 12:10:34 2022-02-03 12:10:34 NULL							
3 19002032 Rina Kumala Sari 2000-06-28 3.40 2022-02-03 12:10:38 2022-02-03 12:10:38 NULL							
4 18012012 James Situmorang 1999-04-02 2.70 2022-02-03 12:10:38 2022-02-03 12:39:31 2022-02-03 12:39:31							
5 19005011 Riana Putria 2000-11-23 2.90 2022-02-03 12:10:38 2022-02-03 12:10:38 NULL							

Gambar: Isi tabel mahasiswas setelah di delete

Mahasiswa yang memiliki nim 18012012 adalah James Situmorang, setelah dilakukan proses delete menggunakan Eloquent, data ini tetap masih ada di database, namun kolom `deleted_at` berisi nilai timestamp yang berisi tanggal dan waktu saat proses delete di lakukan.

Sekarang saya akan coba akses isi tabel `mahasiswas` menggunakan method `all()`. Kode program untuk proses ini sudah kita buat sebelumnya dan bisa diakses dari halaman `localhost:8000/all-view`:

Data Mahasiswa							
Sari Citra Lestari (19003036), Tanggal Lahir: 2001-12-31, IPK: 3.50							
Rudi Permana (19021044), Tanggal Lahir: 2000-08-22, IPK: 2.50							
Rina Kumala Sari (19002032), Tanggal Lahir: 2000-06-28, IPK: 3.40							
Riana Putria (19005011), Tanggal Lahir: 2000-11-23, IPK: 2.90							

Gambar: Hasil halaman localhost:8000/all-view

Terlihat data mahasiswa James Situmorang tidak tampil, padahal method `all()` dipakai untuk menampilkan seluruh isi tabel. Inilah hasil dari proses **soft delete**, dimana data seolah-olah telah terhapus dan tidak bisa diakses lagi, termasuk dengan berbagai method eloquent yang sudah kita bahas.

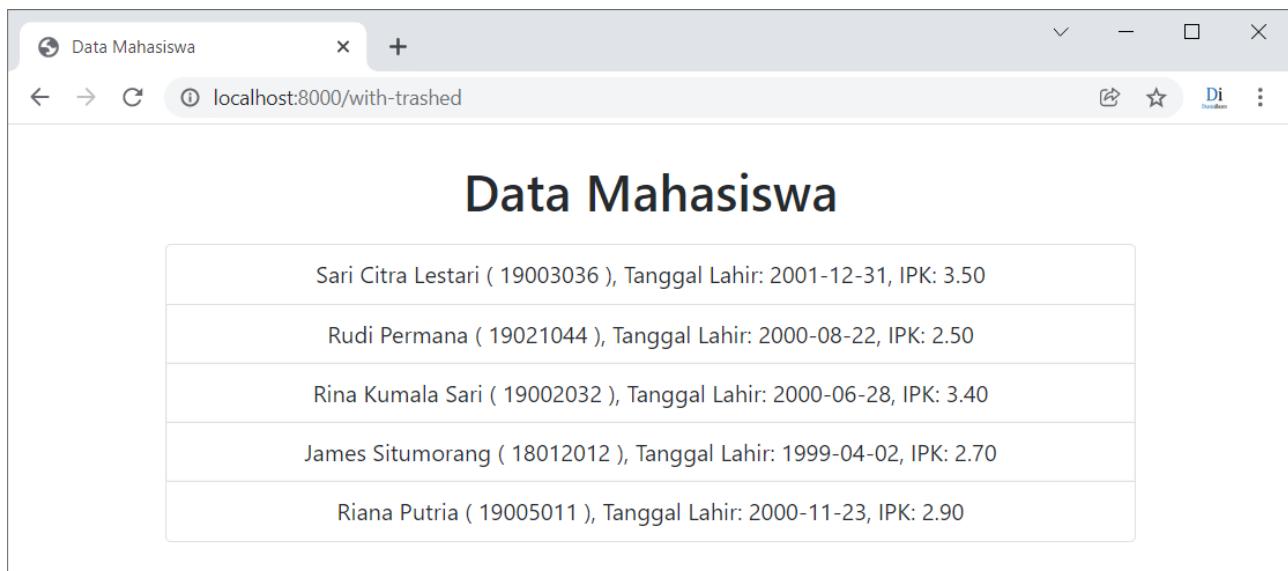
Syaratnya tentu saja data tabel harus diakses menggunakan Eloquent. Jika isi tabel diakses menggunakan *raw query* maupun *query builder*, data yang sudah di soft-delete tetap akan tampil.

Namun bagaimana jika kita tetap ingin menampilkan data yang sudah di soft delete dari Eloquent? Caranya, gunakan method `Model::withTrashed()`, seperti contoh berikut:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function withTrashed(){
6         $mahasiswas = Mahasiswa::withTrashed()->get();
7         return view('tampil-mahasiswa',['mahasiswas' => $mahasiswas]);
8     }
9 }
```



Gambar: Hasil halaman `localhost:8000/with-trashed`

Dengan perintah `Mahasiswa::withTrashed()->get()`, kita bisa mengakses semua data termasuk yang sudah di soft delete.

Restore Soft Delete

Tujuan dari soft delete adalah agar data yang sudah terhapus bisa diakses kembali. Untuk keperluan ini, Eloquent menyediakan method `restore()`. Berikut contoh penggunaannya:

```
app/Http/Controllers/MahasiswaController.php
```

```
1 <?php
2
3 class MahasiswaController extends Controller
4 {
5     public function restore(){
6         Mahasiswa::withTrashed()->where('nim', '18012012')->restore();
7         return "Berhasil di restore";
8     }
9 }
```

Kode program di baris 6 bisa dibaca: Ambil semua data mahasiswa termasuk yang sudah dihapus dengan soft delete, lalu cari mahasiswa dengan nim 18012012, kemudian restore data tersebut.

Silahkan akses route `localhost:8000/restore`, lalu cek ke database.



Gambar: Hasil halaman `http://localhost:8000/restore`

The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". It displays the output of the SQL query `SELECT * FROM mahasiswas;`. The results are shown in a table:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at	deleted_at
1	19003036	Sari Citra Lestari	2001-12-31	3.50	2022-02-03 12:10:27	2022-02-03 12:10:27	NULL
2	19021044	Rudi Permana	2000-08-22	2.50	2022-02-03 12:10:34	2022-02-03 12:10:34	NULL
3	19002032	Rina Kumala Sari	2000-06-28	3.40	2022-02-03 12:10:38	2022-02-03 12:10:38	NULL
4	18012012	James Situmorang	1999-04-02	2.70	2022-02-03 12:10:38	2022-02-03 13:55:25	NULL
5	19005011	Riana Putria	2000-11-23	2.90	2022-02-03 12:10:38	2022-02-03 12:10:38	NULL

5 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswas setelah di delete

Hasilnya, kolom `delete_at` untuk James Situmorang kembali menjadi `NULL`, yang berarti sudah bisa diakses kembali seperti biasa.

Menghapus Data Permanen

Dalam beberapa situasi, kita ingin menghapus data secara permanen dari database, bukan lagi sekedar soft delete. Namun method `delete()` sudah tidak bisa dipakai karena itu hanya menghapus data sebagai soft delete. Untuk keperluan ini, Eloquent menyediakan method `forceDelete()`.

Berikut cara penggunaan dari method `forceDelete()`:

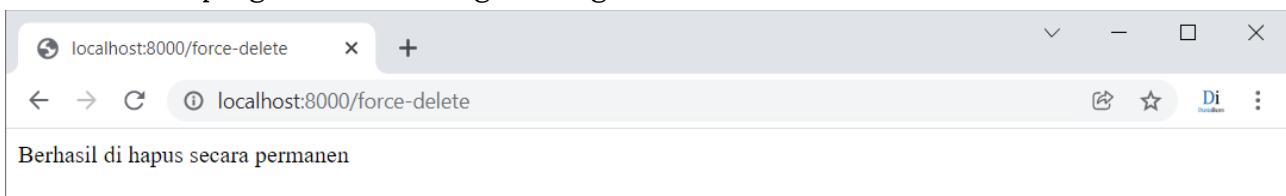
```
app/Http/Controllers/MahasiswaController.php
```

```
1 <?php
```

```
2
3 class MahasiswaController extends Controller
4 {
5     public function forceDelete(){
6         Mahasiswa::where('nim','19005011')->forceDelete();
7         return "Berhasil di hapus secara permanen";
8     }
9 }
```

Dengan perintah `Mahasiswa::where('nim','19005011')->forceDelete()`, maka data mahasiswa dengan nim `19005011` akan dihapus secara permanen, yakni sama seperti method `delete()` jika tanpa menggunakan soft delete.

Jalankan kode program di atas dengan mengakses halaman `localhost:8000/force-delete`:



Gambar: Hasil halaman `localhost:8000/force-delete`

Data yang sudah dihapus secara permanen tidak bisa dikembalikan lagi, karena memang sudah tidak ada di database.

Fitur **soft delete** ini sangat menarik dan bisa dipertimbangkan untuk dipakai pada project yang akan anda kerjakan.

Sepanjang bab ini kita telah membahas **Eloquent**, yang juga menjadi penutup 3 bab bahasan kita tentang database, yakni **raw query**, **query builder**, dan **eloquent ORM**. Ketiga cara akses ini bisa dipakai tergantung kebutuhan, dan juga bisa digabung di dalam satu controller (tidak harus memilih salah satu).

Dalam kebanyakan situasi, cara yang disarankan adalah memakai **Eloquent ORM**, karena penulisannya terkesan lebih modern, singkat, dan mudah digabung dengan materi lain di Laravel.

Eloquent juga memiliki banyak fitur tambahan, misalnya kita bisa membuat relasi antar tabel (*relationship*), yakni menggunakan eloquent untuk memproses query `JOIN` antar tabel. Lebih jauh mengenai eloquent *relationship* ada di buku [Laravel In Depth #1](#) karena materi tersebut butuh penjelasan yang cukup panjang (total akan dibahas 8 jenis relationship).

Berikutnya, kita akan masuk ke materi tentang **Form Processing** dan **Form Validation**.

17. Form Processing dan Form Validation

Form merupakan salah satu fitur terpenting di sebuah web, yang sekaligus paling kompleks karena melibatkan banyak hal, mulai dari perancangan form, design tampilan, proses penerimaan data, validasi inputan form, hingga menyimpan data hasil form ke dalam database. Untungnya, Laravel sudah menyiapkan berbagai komponen untuk memudahkan kita dalam mengelola form.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css**, karena kita akan memakai sedikit kode Bootstrap.

17.1. Pembuatan Form

Pembahasan tentang form kita mulai dengan membahas (yup), cara membuat form. Proses pembuatan form di Laravel tidak terlalu sulit, cukup buat sebuah route untuk mengakses view, dan di dalam view inilah form ditulis. Form bisa dibuat dengan kode HTML dan CSS biasa.

Dulunya Laravel memiliki sebuah fitur form builder (**Form & HTML Helpers**), yakni kode khusus untuk meng-generate form. Namun sejak versi 5, form builder berstatus *deprecated* dan sudah dihapus dari core Laravel. Komponen form builder saat ini dipisah dan dikelola oleh komunitas yang bernama [Laravel Collective](#). Namun saat ini library tersebut juga sudah tidak di maintenance lagi (berhenti di Laravel 6).

Banyak perdebatan apakah form sebaiknya tetap dibuat menggunakan form builder atau cukup dengan kode HTML biasa. Namun karena sudah tidak menjadi bawaan Laravel, dalam bab ini kita akan memakai kode HTML biasa untuk membuat form.

Untuk memulai praktek, silahkan buat controller baru bernama **MahasiswaController** dengan perintah berikut:

```
php artisan make:controller MahasiswaController
```

Lalu buka file route dan tambahkan kode berikut:

```
routes/web.php
```

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/', [MahasiswaController::class, 'index']);
```

Artinya ketika halaman root atau localhost:8000 diakses, jalankan method index() milik MahasiswaController. Dan berikut kode program untuk method index() ini:

```
app/Http/Controllers/MahasiswaController.php
```

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     public function index()
10    {
11        return view('form-pendaftaran');
12    }
13 }
```

Isi dari method index() hanya 1 baris saja, yakni perintah untuk menampilkan view form-pendaftaran. View ini belum ada sebelumnya, sehingga silahkan buat file form-pendaftaran.blade.php ke dalam folder resources\views dengan kode sebagai berikut:

```
resources/views/form-pendaftaran.blade.php
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Form Registrasi</title>
9 </head>
10 <body>
11
12 <div class="container pt-4 bg-white">
13     <div class="row">
14         <div class="col-md-8 col-xl-6">
15             <h1>Pendaftaran Mahasiswa</h1>
16             <hr>
```

```
17      <form action="{{url('/proses-form')}}" method="GET">
18          <div class="mb-3">
19              <label class="form-label" for="nim">NIM</label>
20              <input type="text" class="form-control" id="nim" name="nim">
21          </div>
22
23
24          <div class="mb-3">
25              <label class="form-label" for="nama">Nama Lengkap</label>
26              <input type="text" class="form-control" id="nama" name="nama">
27          </div>
28
29          <div class="mb-3">
30              <label class="form-label" for="email">Email</label>
31              <input type="text" class="form-control" id="email" name="email">
32          </div>
33
34          <div class="mb-3">
35              <label class="form-label">Jenis Kelamin</label>
36              <div class="d-flex">
37                  <div class="form-check me-3">
38                      <input class="form-check-input" type="radio" name="jenis_kelamin"
39                          id="laki_laki" value="L">
40                      <label class="form-check-label" for="laki_laki">Laki-laki</label>
41                  </div>
42                  <div class="form-check">
43                      <input class="form-check-input" type="radio" name="jenis_kelamin"
44                          id="perempuan" value="P">
45                      <label class="form-check-label" for="perempuan">Perempuan</label>
46                  </div>
47              </div>
48          </div>
49
50          <div class="mb-3">
51              <label class="form-label" for="jurusan">Jurusan</label>
52              <select class="form-select" name="jurusan" id="jurusan">
53                  <option value="Teknik Informatika">Teknik Informatika</option>
54                  <option value="Sistem Informasi">Sistem Informasi</option>
55                  <option value="Ilmu Komputer">Ilmu Komputer</option>
56                  <option value="Teknik Komputer">Teknik Komputer</option>
57                  <option value="Teknik Telekomunikasi">Teknik Telekomunikasi</option>
58              </select>
59          </div>
60
61          <div class="mb-3">
62              <label class="form-label" for="alamat">Alamat</label>
63              <textarea class="form-control" id="alamat" rows="3"
64                  name="alamat"></textarea>
65          </div>
66
67          <button type="submit" class="btn btn-primary mb-2">Daftar</button>
68      </form>
69
70      </div>
71  </div>
```

```
72 </div>
73
74 </body>
75 </html>
```

The screenshot shows a web browser window titled "Form Registrasi" at the URL "localhost:8000". The page has a title "Pendaftaran Mahasiswa". It contains several input fields: "NIM" (text box), "Nama Lengkap" (text box), "Email" (text box), "Jenis Kelamin" (radio buttons for "Laki-laki" and "Perempuan"), "Jurusan" (dropdown menu currently set to "Teknik Informatika"), "Alamat" (text area), and a blue "Daftar" button.

Gambar: Tampilan view form-pendaftaran.blade.php

Kode programnya cukup panjang karena terdapat 6 jenis inputan form. Di sini saya memakai beberapa class CSS bawaan Bootstrap agar tampilan form lebih menarik.

Ada beberapa hal yang perlu kita bahas, Pertama, di baris 18 terdapat tag `<form>` dengan atribut `action="{{url('/proses-form')}}`. Ini berarti ketika di submit, nilai form akan dikirim ke alamat `localhost:8000/proses-form`. Route untuk URL ini akan kita tulis sesaat lagi. Selain itu juga terdapat atribut `method="GET"`, sehingga nilai form akan dikirim menggunakan GET request.

Tiga inputan form paling atas merupakan text box untuk **Nim**, **Nama Lengkap** dan **Email**. Ketiganya menggunakan atribut `name="nim"`, `name="nama"` dan `name="email"`. Atribut `name` inilah yang nantinya menjadi nama variabel untuk mengakses nilai form.

Inputan form keempat berupa radio button untuk **Jenis Kelamin**. Di sini saya memakai atribut `name="jenis_kelamin"`. Selain itu terdapat atribut `value="L"` untuk pilihan laki-laki dan atribut

`value="P"` untuk pilihan perempuan. Jika salah satu radio button ini dipilih, nilai yang akan terkirim merupakan salah satu dari L atau P.

Inputan form kelima adalah sebuah dropdown `<select>` untuk memilih **Jurusan**. Atribut yang dipakai adalah `name="jurusan"`. Nilai yang bisa dipilih berada di dalam tag `<option>` dengan atribut value yang sesuai dengan pilihan jurusan.

Inputan form keenam berupa sebuah `<textarea>` untuk menginput **Alamat**. Atribut yang dipakai adalah `name="alamat"`. Dan terakhir terdapat sebuah tombol **Daftar** untuk proses submit sebagai penutup form.

Jika semua inputan form ini diisi, kita akan memiliki 6 inputan yang tersimpan ke dalam atribut `name` sebagai berikut:

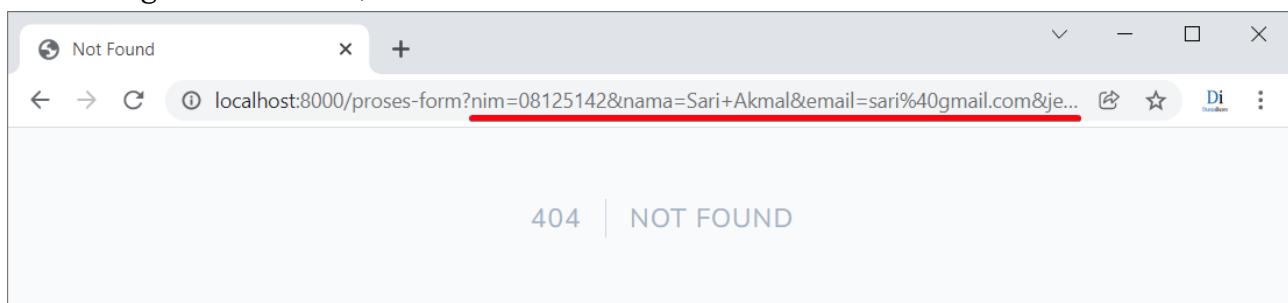
- ◆ `nim`
- ◆ `nama`
- ◆ `email`
- ◆ `jenis_kelamin`
- ◆ `jurusran`
- ◆ `alamat`

Atribut `name` inilah yang nantinya menjadi kunci pada saat pemrosesan nilai form.

17.2. Request Object

Form sudah tersedia, kita bisa lanjut membahas bagaimana cara mengirim data inputan form ini agar bisa di proses oleh Laravel.

Di dalam tag `<form>`, saya menulis atribut `action="{{url('/proses-form')}}`, inilah alamat pengiriman nilai form ketika tombol submit di tekan. Mari kita coba, silahkan input data sembarang ke dalam form, lalu klik tombol Submit.



Gambar: Testing proses submit form

Betul, akan tampil halaman 404 karena route untuk alamat `/proses-form` belum tersedia. Namun perhatikan bagian URL, terdapat tambahan *query string* karena data form dikirim menggunakan `method="GET"`. Data inilah yang harus kita tangkap untuk bisa memproses nilai form.

Lanjut, saya akan buat route untuk alamat '/proses-form':

routes/web.php

```
1 Route::get('/', [MahasiswaController::class, 'index']);  
2 Route::get('/proses-form', [MahasiswaController::class, 'prosesForm']);
```

Kode route di baris 2 artinya, jika alamat /proses-form diakses, panggil method prosesForm() yang ada di MahasiswaController. Di method prosesForm() inilah kita akan membuat kode program untuk mengakses nilai form tadi.

Jika menggunakan PHP native, terdapat global variabel `$_GET` dan `$_POST` yang bisa dipakai untuk mengakses inputan form. Sebagai contoh, jika inputan form ditulis dengan kode `<input type="text" name="nim">`, kita bisa akses nilainya dari variabel `$_GET['nim']` atau `$_POST['nim']`, tergantung metode apa yang dipakai untuk mengirim data form.

Laravel menggunakan cara yang sedikit berbeda, dimana semua inputan form disimpan ke dalam **Request** object. Object ini sebenarnya sudah sering kita lihat karena ada di setiap file controller yang di-generate Laravel, yakni baris kode berikut:

```
use Illuminate\Http\Request;
```

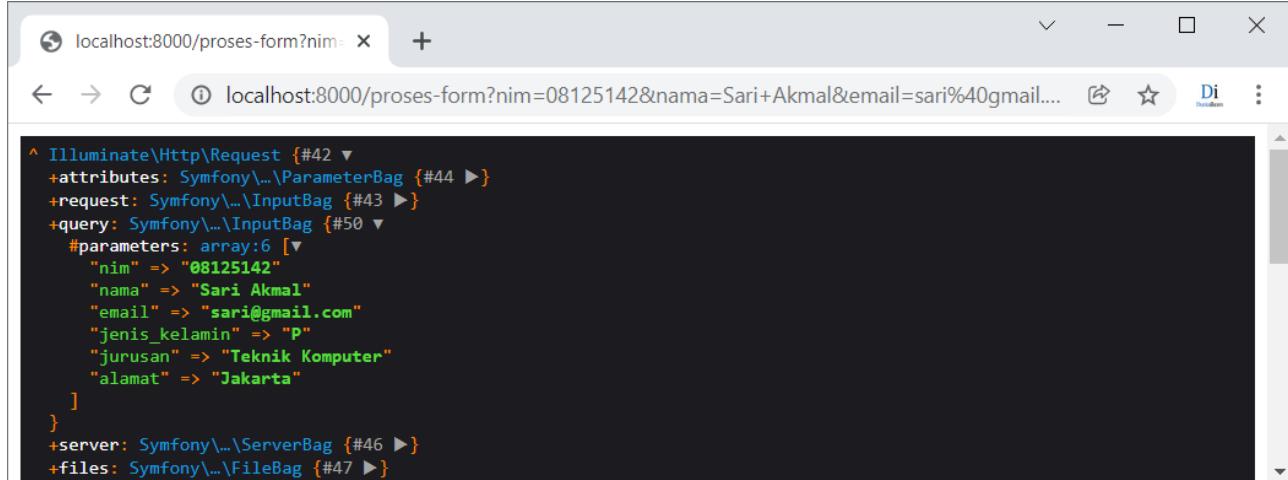
Kode di atas dipakai untuk meng-import Request class ke dalam controller, namun selama ini belum pernah kita akses. Sekarang modifikasi file `MahasiswaController` menjadi berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4  
5 use Illuminate\Http\Request;  
6  
7 class MahasiswaController extends Controller  
8 {  
9     public function index()  
10    {  
11        return view('form-pendaftaran');  
12    }  
13  
14    public function prosesForm(Request $request)  
15    {  
16        dump($request);  
17    }  
18 }
```

Perhatikan tambahan kode `Request $request` di dalam argument method `prosesForm()`. Ini merupakan type hint khusus yang dipakai Laravel untuk menerapkan teknik yang disebut sebagai **dependency injection**. Dengan penulisan seperti ini, di dalam method `prosesForm()` kita bisa mengakses variabel `$request` yang sudah berisi **Request** object.

Variabel `$request` selanjutnya saya `dump()` di baris 16 untuk sekedar melihat isinya. Mari kita akses method ini dengan cara mengisi form dengan data sembarang, lalu klik tombol submit.



```

^ Illuminate\Http\Request {#42 ▶
+attributes: Symfony\...\\ParameterBag {#44 ▶}
+request: Symfony\...\\InputBag {#43 ▶}
+query: Symfony\...\\InputBag {#50 ▶
    #parameters: array:6 [▼
        "nim" => "08125142"
        "nama" => "Sari Akmal"
        "email" => "sari@gmail.com"
        "jenis_kelamin" => "P"
        "jurusan" => "Teknik Komputer"
        "alamat" => "Jakarta"
    ]
}
+server: Symfony\...\\ServerBag {#46 ▶}
+files: Symfony\...\\FileBag {#47 ▶}

```

Gambar: Tampilan isi Request object

Inilah isi dari **Request** object. Cukup banyak data yang tersimpan, salah satunya tab `+query` yang berisi `#parameters`. Di dalamnya bisa terlihat semua nilai inputan form. Proses `dump($request)` ini sebaiknya menjadi hal pertama yang dilakukan, sekedar memastikan bahwa nilai form memang sudah diterima Laravel.

Di dalam controller, variabel `$request` inilah yang akan kita pakai untuk mengambil nilai inputan form. Laravel menyediakan berbagai cara, salah satunya dengan menulis nama inputan form sebagai property dari variabel `$request`. Berikut format dasar penulisannya:

```
$request-><nama_inputan_form>
```

Sebagai contoh, untuk mengambil nilai dari inputan `nim`, bisa diakses dengan kode `$request->nim`, atau untuk inputan `tanggal_lahir`, bisa diakses dari `$request->tanggal_lahir`.

Saya akan modifikasi isi method `prosesForm()` menjadi sebagai berikut:

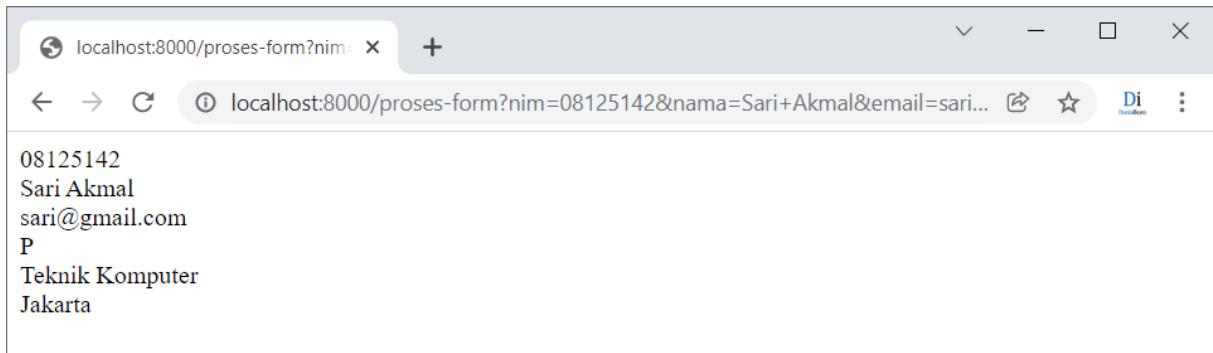
app/Http/Controllers/MahasiswaController.php

```

1  <?php
2  // ...
3  // ...
4
5  public function prosesForm(Request $request)
6  {
7      echo $request->nim;           echo "<br>";
8      echo $request->nama;          echo "<br>";
9      echo $request->email;         echo "<br>";
10     echo $request->jenis_kelamin; echo "<br>";
11     echo $request->jurusan;       echo "<br>";
12     echo $request->alamat;
13 }

```

Untuk uji coba, silahkan isi kembali semua nilai form lalu klik tombol submit. Atau bisa juga refresh halaman /proses-form jika nilai inputan form masih ada di dalam query string. Berikut hasilnya:



Gambar: Inputan form sudah bisa diakses

Sip, semua inputan form sudah bisa diakses.

Laravel juga menyediakan cara lain untuk mengakses nilai form ini. Sebagai contoh, selain `$request->nim`, nilai **nim** juga bisa diakses menggunakan kode berikut:

```
1 echo $request->input('nim');
2 echo $request->get('nim');
3 echo request('nim');
4 echo data_get($request, 'nim');
```

Banyaknya cara akses ini memiliki kelebihan dan kekurangan tersendiri. Sisi plusnya, kode program Laravel menjadi lebih fleksibel dan mendukung beragam cara penulisan. Sisi minusnya, kita kadang pusing harus memilih yang mana, dan bagi yang tidak tau bisa menganggap cara yang satu merupakan pengganti dari cara yang lain. Dalam buku ini saya akan memakai `$request->nim` saja.

17.3. Post Request dan CSRF Token

Kita sudah bisa mengakses nilai form, namun inputan form tersebut dikirim menggunakan method **GET**. Method GET sebaiknya dipakai untuk form yang mengambil sesuatu (get) dari database, seperti form pencarian.

Untuk form yang mengirim data dengan tujuan diinput ke dalam database atau memiliki data sensitif seperti password, sebaiknya memakai method **POST**. Untuk mengubah form-pendaftaran agar diproses dengan method POST, kita perlu modifikasi beberapa baris kode program.

Pertama, nilai atribut `method` dari tag `<form>` yang ada di dalam `form-pendaftaran.blade.php` harus diubah menjadi POST:

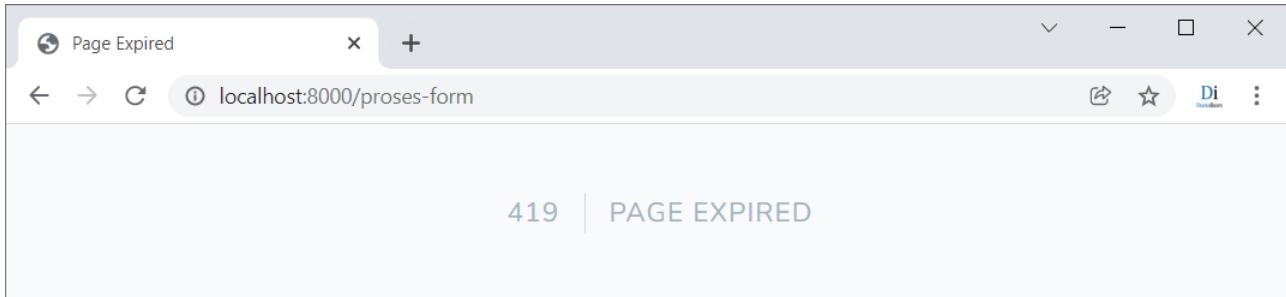
```
<form action="{{url('/proses-form')}}" method="POST">
```

Karena sekarang form di submit dengan post request, maka untuk route juga harus dimodifikasi dari Route::get menjadi Route::post:

```
Route::post('/proses-form', [MahasiswaController::class, 'prosesForm']);
```

Dalam kode ini, kita juga berkenalan dengan method baru dari route, yakni Route::post(). Ini adalah static method khusus untuk memproses post request, yakni proses pengiriman form yang menggunakan method **post**.

Mari kita coba. Silahkan buka halaman localhost:8000, input beberapa data sembarang dan klik tombol submit.



Gambar: Halaman error 419, Page Expired

Akan tampil halaman error 419 | Page Expired. Error ini disebabkan karena jika form dikirim menggunakan method POST, Laravel butuh sebuah **CSRF token**.

CSRF adalah singkatan dari **Cross-Site Request Forgery**, yakni sebuah celah keamanan dengan cara mengirim form yang nilainya sudah di manipulasi.

Sebagai contoh, misalkan rudi ingin mentransfer uang ke lisa menggunakan e-banking yang tidak terproteksi. Pada saat mengirim uang, tentunya menggunakan form dimana rudi harus menginput nama, nomor rekening dan nominal yang akan dikirim.

Untuk memudahkan ilustrasi, kita anggap form tersebut menggunakan method GET. Ketika form di submit, halaman web akan membaca nilai inputan tadi yang diambil dari query string, misalnya:

```
www.bank.com/proses.php?user=lisa&no_rek=12345&nominal=1000000
```

Di halaman proses.php, query string ini bisa diambil dengan kode `$_GET['user']`, `$_GET['no_rek']`, dan `$_GET['nominal']`, yang kemudian di proses lebih lanjut.

Halaman di atas hanya bisa diakses saat rudi login ke dalam aplikasi bank tersebut. Jika anda sering menggunakan e-banking, biasanya website bank memberikan waktu 5 menit sebelum otomatis di logout.

Selama jangka waktu tersebut, tiba-tiba rudi menerima email yang berisi link berikut:

```
1 <a href="www.bank.com/proses.php?user=alex&no_rek=57890&nominal=999999">
2 Anda mendapat hadiah 1 mobil, klik untuk klaim
```

3

Karena penasaran, rudi men-klik link tersebut. Maka halaman proses.php milik bank akan memproses transfer dana ke user alex sebesar 9.999.999!

Inilah ilustrasi dari teknik CSRF. Jika form dikirim menggunakan method POST, prosesnya menjadi sedikit rumit tapi tetap bisa dilakukan.

Untuk mencegah hal inilah, Laravel mengharuskan memakai **CSRF token**. Idenya, setiap form request yang dikirim untuk diproses, harus memiliki angka acak tertentu. Jika form tersebut tidak memiliki angka acak yang sesuai, maka form dianggap tidak valid dan tidak akan diproses.

Cara untuk menggunakan CSRF token ini juga sangat sederhana, cukup tambah perintah `@csrf` ke dalam tag form di dalam view, seperti contoh berikut:

resources/views/form-pendaftaran.blade.php

```

1 // ...
2 <form action="{{url('/proses-form')}}" method="POST">
3     @csrf
4     <div class="mb-3">
5         <label class="form-label" for="nim">NIM</label>
6         <input type="text" class="form-control" id="nim" name="nim">
7     </div>
8 // ...

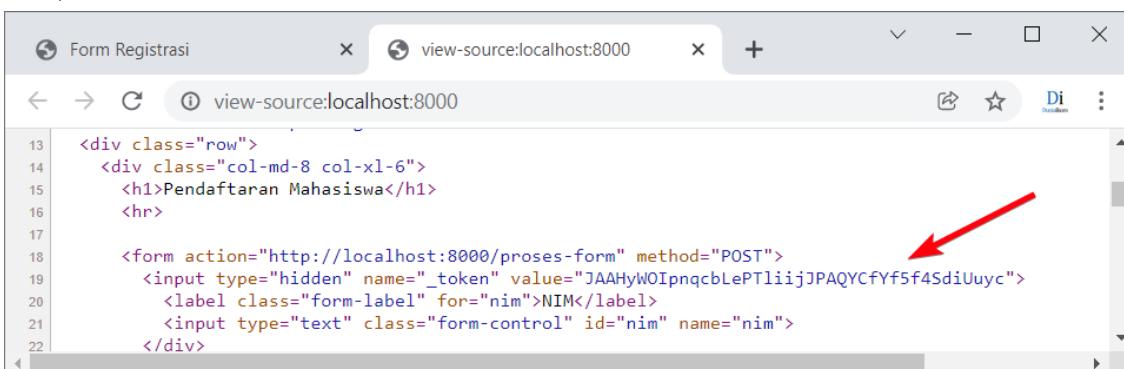
```

Dalam contoh di atas, kode `@csrf` saya tempatkan di baris 3, yakni tepat setelah tag `<form>`.

Posisi peletakan CSRF token ini bisa dimana saja, sepanjang masih di dalam tag `<form>`.

Beberapa programmer ada yang lebih suka menempatkannya di akhir form, yakni sebelum tag penutup `</form>`.

Save view, lalu buka halaman form ini. Cek source code form ini dari web browser:



The screenshot shows the browser's developer tools with the "view-source" tab selected. The page content is displayed with line numbers on the left. A red arrow points to the line containing the generated CSRF token:

```

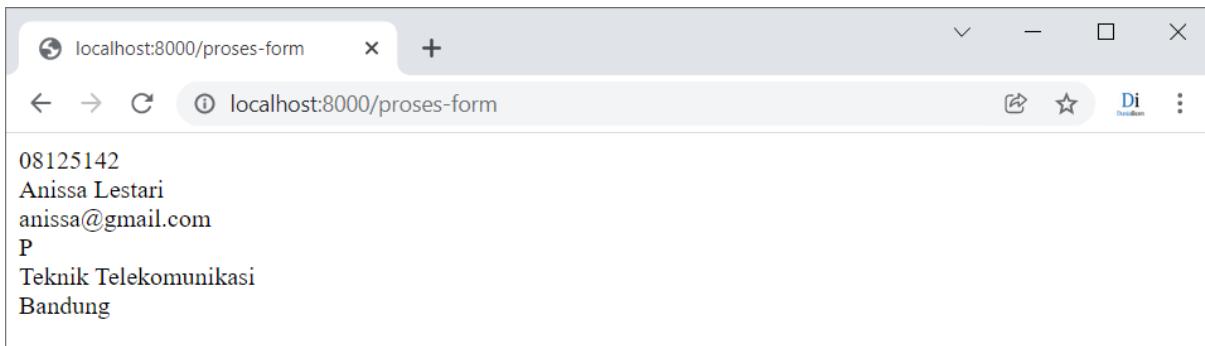
13 <div class="row">
14     <div class="col-md-8 col-xl-6">
15         <h1>Pendaftaran Mahasiswa</h1>
16         <hr>
17
18     <form action="http://localhost:8000/proses-form" method="POST">
19         <input type="hidden" name="_token" value="JAAHyWOIpncbLePTljjJPAQYcfYf5f4SdiUuyc">
20         <label class="form-label" for="nim">NIM</label>
21         <input type="text" class="form-control" id="nim" name="nim">
22     </div>

```

Gambar: CSRF token yang di generate Laravel

Di baris tempat kita menulis perintah `@csrf` akan berganti dengan sebuah tag `<input type="hidden" name="_token" value="...>` bernama `_token` yang berisi value berupa karakter acak. Inilah cara yang dipakai Laravel untuk menghindari celah keamanan CSRF.

Silahkan input beberapa data ke dalam form dan klik tombol submit.



Gambar: Hasil inputan form

Jika anda masih memiliki kode program untuk mengakses nilai form di dalam method `prosesForm()`, maka nilai form bisa terlihat di web browser. Artinya, untuk mengakses nilai form yang dikirim dari method GET maupun POST, kita tetap menggunakan perintah yang sama, yakni `$request-><nama_inputan_form>`.

Untuk CSRF sendiri sudah diproses otomatis oleh Laravel, kita tidak perlu menambah kode apapun ke dalam controller.

17.4. Validasi Form dari Request Object

Selanjutnya adalah proses yang tidak kalah penting, yakni **Validasi**. Validasi adalah proses pemeriksaan inputan form agar sesuai dengan data yang kita inginkan, misalnya nim harus terdiri dari 8 digit, nama tidak boleh kosong, alamat email harus sesuai dengan format sebuah email, dst.

Syarat untuk proses validasi ini sangat beragam, mulai dari yang sederhana seperti contoh di atas, sampai yang rumit seperti pemeriksaan apakah nomor nim yang sama sudah ada di database atau belum. Untungnya, Laravel sudah menyediakan sebagian besar metode validasi ini. Yang kita perlukan hanya menulis *keyword* untuk syarat yang sesuai.

Proses validasi di Laravel juga bisa dilakukan dengan banyak cara, salah satu yang paling mudah adalah memanggil method `validate()` dari **Request** object.

Berikut modifikasi method `prosesForm()` dengan tambahan proses validasi:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2 // ...
3 // ...
4
5 public function prosesForm(Request $request)
6 {
7     $validateData = $request->validate([
8         'nim'          => 'required|size:8',

```

```

9      'nama'          => 'required|min:3|max:50',
10     'email'          => 'required|email',
11     'jenis_kelamin'  => 'required|in:P,L',
12     'jurusan'        => 'required',
13     'alamat'         => '',
14   ]);
15
16 dump ($validateData);
17
18 echo $validateData['nim'];           echo "<br>";
19 echo $validateData['nama'];          echo "<br>";
20 echo $validateData['email'];         echo "<br>";
21 echo $validateData['jenis_kelamin']; echo "<br>";
22 echo $validateData['jurusan'];       echo "<br>";
23 echo $validateData['alamat'];
24 }

```

Di baris 7 saya menjalankan method `$request->validate()`. Sebagai argument dari method ini adalah sebuah associative array yang berisi syarat validasi. Hasil proses validasi kemudian disimpan ke dalam variabel `$validateData`.

Format penulisan syarat validasi adalah sebagai berikut:

```
<nama_inputan_form> => syarat1 | syarat2 | syarat3 | , dst...
```

Karakter pipe " | " dipakai untuk memisahkan syarat satu dengan syarat lain.

Dalam contoh di atas, semua inputan form kecuali `alamat` memiliki syarat `required` yang artinya tidak boleh kosong. Syarat `size:8` untuk `nim` berarti nilai inputan `nim` harus berisi persis 8 digit karakter. Syarat `min:3` dan `max:50` untuk `nama` artinya inputan `nama` harus berisi minimal 3 karakter dan maksimal 50 karakter.

Syarat `email` berarti inputan `email` harus sesuai dengan format email seperti memiliki karakter '@' dan nama domain. Serta syarat `in:P,L` untuk inputan `jenis_kelamin` artinya nilai yang dikirim haruslah salah satu dari karakter 'P' atau 'L'.

Untuk Laravel 9, terdapat setidaknya **66 syarat validasi**. Karena terlalu banyak, saya sarankan untuk membaca langsung dari dokumentasi Laravel di [Available Validation Rules](#). Beberapa di antara syarat validasi ini akan kita bahas secara bertahap.

Setelah menulis syarat validasi, di baris 18 – 23 saya kembali menampilkan isi inputan form. Namun perhatikan kali ini bukan lagi dari **Request** object, tapi dari array `$validateData` yang dihasilkan oleh method `$request->validate()`. Karena berbentuk array, maka untuk mengakses nilai input ditulis sebagai `$validateData['nim']`.

Inputan `alamat` sebenarnya tidak memiliki syarat validasi apapun, namun terpaksa tetap saya tulis ke dalam method `$request->validate()` agar nilai alamat ini juga dikembalikan ke dalam array `$validateData`. Jika tidak di sertakan, maka array `$validateData` tidak berisi nilai alamat.

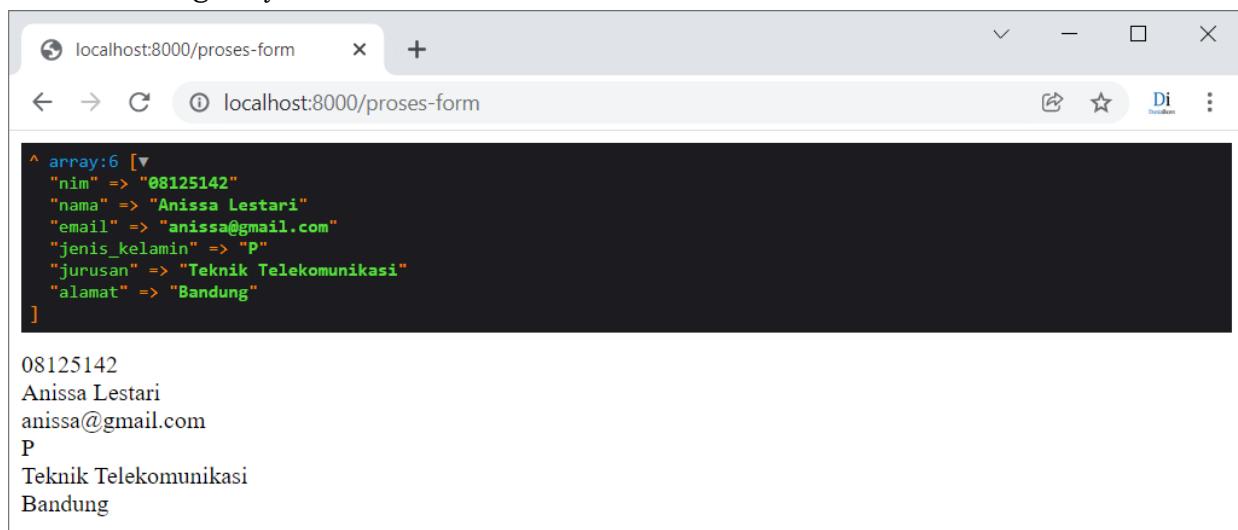
Bisa juga disebut bahwa data yang tersimpan di dalam `$validateData` adalah nilai inputan

form yang sudah bersih, yakni telah melewati semua proses validasi.

Selain menggunakan format pipe " | ", syarat validasi juga bisa ditulis dalam bentuk nested array seperti contoh berikut:

```
1 $validateData = $request->validate([
2     'nim'          => ['required','size:8'],
3     'nama'         => ['required','min:3','max:50'],
4     'email'        => ['required','email'],
5     'jenis_kelamin' => ['required','in:P,L'],
6     'jurusan'      => ['required'],
7     'alamat'       => [],
8 ]);
```

Mari kita coba syarat validasi di atas. Silahkan buka kembali form pendaftaran dan isi semua data sesuai dengan syarat di atas, lalu klik tombol submit:



Gambar: Hasil proses Validasi

Tidak ada masalah, semua data yang terdapat di array `$validateData` sudah bisa di akses.

Sekarang input ulang form dengan data yang salah (yang tidak memenuhi syarat validasi), atau bisa juga tanpa mengisi inputan apapun, langsung saja klik tombol **Daftar**.

Hasilnya, halaman form seolah-olah *refresh*, tidak ada hasil inputan yang tampil dan juga tidak ada pesan error. Apa yang terjadi?

Sebenarnya halaman form ini sudah mengalami proses *redirect*. Ketika form di submit, semua nilai inputan form akan dikirim dan di proses oleh halaman `localhost:8000/proses-form`. Namun karena ada data yang tidak lolos validasi, Laravel secara otomatis akan me-*redirect* kembali ke halaman asal form **beserta pesan error**. Namun pesan error ini tidak tampil karena harus di akses terlebih dahulu.

17.5. Mengakses Pesan Error

Ketika hasil inputan form tidak lolos syarat validasi, Laravel akan membuat sebuah **session** yang berisi pesan error, kemudian me-redirect halaman ke alamat asal form, yang dalam contoh kita adalah halaman root.

Pesan error ini disimpan dalam *instance* dari class `Illuminate\Support\MessageBag`, yang bisa di akses dari dalam view menggunakan variabel `$errors`. Artinya, untuk bisa menampilkan pesan error dari dalam view, kita harus akses variabel `$errors` ini.

Silahkan buka kembali file `form-pendaftaran.blade.php`, lalu tambah kode berikut di baris paling atas:

`resources/views/form-pendaftaran.blade.php`

```

1 <?php
2     dump($errors);
3 ?>
4
5 <html lang="en">
6 <head>
7     <meta charset="UTF-8">
8     // ...
9     // ...

```

Terdapat tambahan 3 baris program, yang sebenarnya hanya berisi 1 perintah untuk proses `dump($errors)`. Save file ini, lalu buka kembali halaman root yang berisi form pendaftaran.

Di bagian paling atas akan terlihat hasil tampilan dump, namun saat ini belum berisi hasil apapun karena form belum di submit. Dengan kondisi inputan kosong, langsung saja klik tombol **Daftar**.

```

^ Illuminate\Support\ViewErrorBag {#257 ▾
  #bags: array:1 [▼
    "default" => Illuminate\Support\MessageBag {#258 ▾
      #messages: array:4 [▼
        "nim" => array:1 [▼
          0 => "The nim field is required."
        ]
        "nama" => array:1 [▼
          0 => "The nama field is required."
        ]
        "email" => array:1 [▼
          0 => "The email field is required."
        ]
        "jenis_kelamin" => array:1 [▼
          0 => "The jenis_kelamin field is required."
        ]
      ]
    }
  ]
}

```

Gambar: Hasil dump dump(\$errors)

Hasil dump di bagian atas sudah berisi sesuatu, inilah pesan error yang di generate Laravel berdasarkan syarat validasi yang kita tulis di MahasiswaController. Pesan error ini akan berganti-ganti tergantung jenis validasi yang gagal.

Agar tampilannya lebih mudah dibaca, saya akan modifikasi kembali view form-pendaftaran dengan tambahan perulangan foreach untuk menampilkan isi variabel \$errors. Berikut kode yang diperlukan:

resources/views/form-pendaftaran.blade.php

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      // ....
5  </head>
6  <body>
7
8  <div class="container pt-4 bg-white">
9      <div class="row">
10         <div class="col-md-8 col-xl-6">
11             <h1>Pendaftaran Mahasiswa</h1>
12             <hr>
13
14             @if ($errors->any())
15                 <div class="alert alert-danger">
16                     <ul class="mb-0">
17                         @foreach ($errors->all() as $error)
18                             <li>{{ $error }}</li>
19                         @endforeach
20                     </ul>
21                 </div>
22             @endif
23
24             <form action="{{url('/proses-form')}}" method="POST">
25                 @csrf
26                 <div class="form-group">
27                     // ...

```

Saya menghapus dump() dibagian atas dan menambah kode program baru di baris 14 – 22.

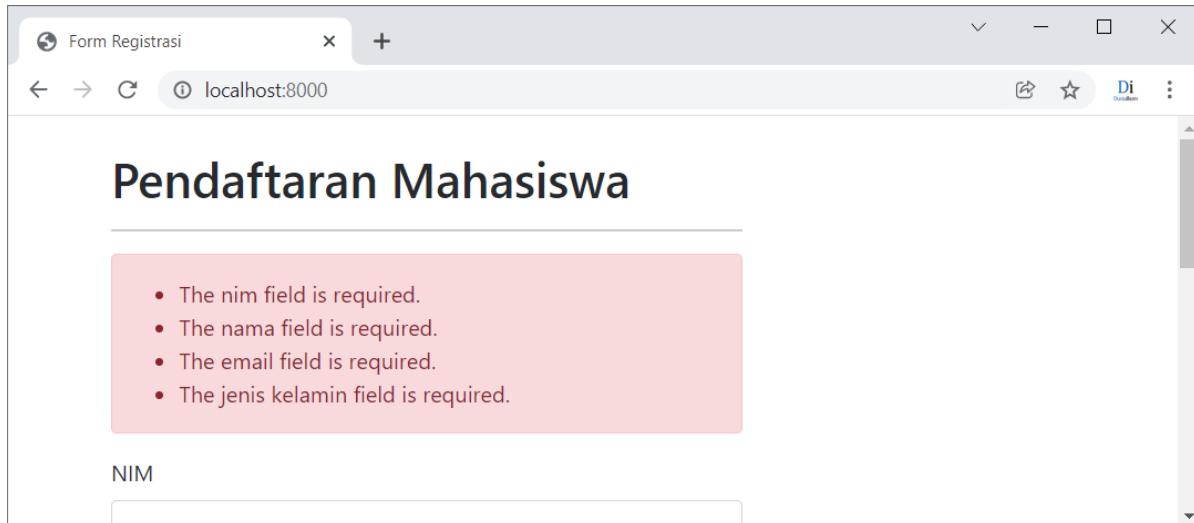
Di baris 14, terdapat perintah `@if ($errors->any())`, ini adalah proses pemeriksaan apakah ada pesan error atau tidak. Method `$errors->any()` akan menghasilkan boolean **true** jika terdapat error dari proses validasi, atau bernilai **false** jika tidak ada error, misalnya pada saat form pertama kali ditampilkan.

Jika ternyata terdapat error, tampilkan sebuah *unordered list* dari tag `` yang berada di dalam class `.alert` dan `.alert-danger` bawaan Bootstrap. Tambahan class ini saya pakai agar tampilan pesan error menjadi lebih menarik dengan warna background merah.

Kemudian terdapat perulangan `@foreach` di baris 17 – 19 yang dipakai untuk mengakses hasil dari pemanggilan method `$errors->all()`. Method `$errors->all()` akan mengembalikan

semua pesan error dalam bentuk array, yang langsung di loop menggunakan @foreach. Setiap pesan error berada di dalam tag supaya tampil sebagai list.

Buka ulang halaman form pendaftaran, dan langsung klik tombol **Daftar**.

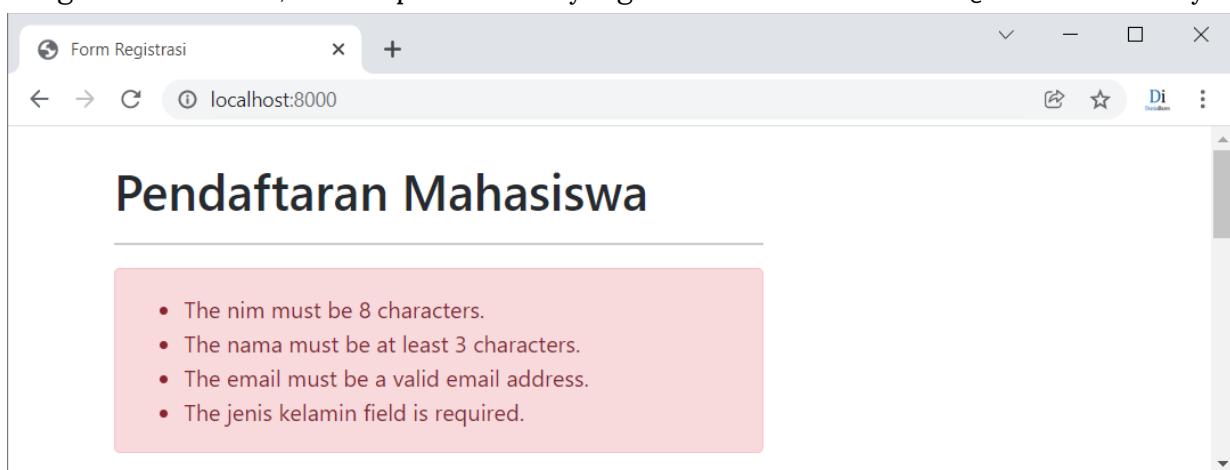


Gambar: Tampilan pesan error dalam bentuk list

Hasilnya, di bagian atas terlihat pesan error dalam bentuk *unordered list*. Pesan error ini di generate otomatis oleh Laravel dengan format tertentu dan dalam bahasa inggris. Pada materi berikutnya, kita akan bahas cara membuat teks pesan error ini secara manual.

Laravel men-format pesan error berdasarkan nama inputan. Sebagai contoh, karena inputan pertama saya tulis sebagai <input type="text" name="nim">, maka ketika form ini tidak berisi nilai, pesan yang tampil adalah 'The **nim** field is required'. Apabila di nama inputan terdapat karakter underscore (_), akan dikonversi menjadi spasi seperti pesan error terakhir yang berasal dari inputan **jenis_kelamin**.

Isi pesan error juga berbeda untuk setiap syarat validasi yang gagal. Sebagai contoh, silahkan isi inputan **nim** dengan teks yang kurang dari 8 karakter, inputan **nama** dengan teks yang kurang dari 2 karakter, serta inputan **email** yang tidak memiliki karakter '@'. Berikut hasilnya:



Gambar: Tampilan pesan error berbeda-beda

Terlihat pesan error yang tampil juga sudah berubah, sesuai dengan syarat validasi yang gagal.

17.6. Pesan Error Terpisah

Laravel juga menyediakan cara jika kita ingin menampilkan pesan error untuk inputan tertentu saja, yakni terpisah untuk setiap inputan form. Sebagai contoh, untuk menampilkan pesan error untuk **nim**, bisa menggunakan kode berikut:

```
@error('nim')
    {{ $message }}
@enderror
```

Perintah `@error()` merupakan perintah khusus blade yang dipakai untuk memeriksa apakah terdapat error untuk inputan yang ditulis sebagai argument. Jika ada error, blok kode program antara `@error()` sampai dengan `@enderror` akan dijalankan. Di dalam blok kode program ini kita bisa mengakses variabel `$message` yang hanya berisi pesan error untuk inputan itu saja.

Sebagai contoh lain, jika saya ingin menampilkan pesan error untuk inputan **nama**, kode yang dipakai adalah sebagai berikut:

```
@error('nama')
    {{ $message }}
enderror
```

Dengan tambahan ini, saya akan modifikasi ulang file `form-pendaftaran.blade.php` agar pesan error tampil secara terpisah di bagian bawah setiap inputan, tidak lagi secara global di atas form. Berikut kode program lengkap dari hasil modifikasi ini:

`resources/views/form-pendaftaran.blade.php`

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link href="/css/bootstrap.min.css" rel="stylesheet">
8      <title>Form Registrasi</title>
9  </head>
10 <body>
11
12 <div class="container pt-4 bg-white">
13     <div class="row">
14         <div class="col-md-8 col-xl-6">
15             <h1>Pendaftaran Mahasiswa</h1>
16             <hr>
17
18             <form action="{{url('/proses-form')}}" method="POST">
19                 @csrf
20                 <div class="mb-3">
```

```
21      <label class="form-label" for="nim">NIM</label>
22      <input type="text" class="form-control" id="nim" name="nim">
23      @error('nim')
24          <div class="text-danger">{{ $message }}</div>
25      @enderror
26  </div>
27
28  <div class="mb-3">
29      <label class="form-label" for="nama">Nama Lengkap</label>
30      <input type="text" class="form-control" id="nama" name="nama">
31      @error('nama')
32          <div class="text-danger">{{ $message }}</div>
33      @enderror
34  </div>
35
36  <div class="mb-3">
37      <label class="form-label" for="email">Email</label>
38      <input type="text" class="form-control" id="email" name="email">
39      @error('email')
40          <div class="text-danger">{{ $message }}</div>
41      @enderror
42  </div>
43
44  <div class="mb-3">
45      <label class="form-label">Jenis Kelamin</label>
46      <div class="d-flex">
47          <div class="form-check me-3">
48              <input class="form-check-input" type="radio" name="jenis_kelamin"
49                  id="laki_laki" value="L">
50              <label class="form-check-label" for="laki_laki">Laki-laki</label>
51          </div>
52          <div class="form-check">
53              <input class="form-check-input" type="radio" name="jenis_kelamin"
54                  id="perempuan" value="P">
55              <label class="form-check-label" for="perempuan">Perempuan</label>
56          </div>
57      </div>
58      @error('jenis_kelamin')
59          <div class="text-danger">{{ $message }}</div>
60      @enderror
61  </div>
62
63  <div class="mb-3">
64      <label class="form-label" for="jurusan">Jurusan</label>
65      <select class="form-select" name="jurusan" id="jurusan">
66          <option value="Teknik Informatika">Teknik Informatika</option>
67          <option value="Sistem Informasi">Sistem Informasi</option>
68          <option value="Ilmu Komputer">Ilmu Komputer</option>
69          <option value="Teknik Komputer">Teknik Komputer</option>
70          <option value="Teknik Telekomunikasi">Teknik Telekomunikasi</option>
71      </select>
72      @error('jurusan')
73          <div class="text-danger">{{ $message }}</div>
74      @enderror
75  </div>
```

```

76
77      <div class="mb-3">
78          <label class="form-label" for="alamat">Alamat</label>
79          <textarea class="form-control" id="alamat" rows="3"
80              name="alamat"></textarea>
81      </div>
82
83          <button type="submit" class="btn btn-primary mb-2">Daftar</button>
84      </form>
85
86  </div>
87 </div>
88 </div>
89
90 </body>
91 </html>
```

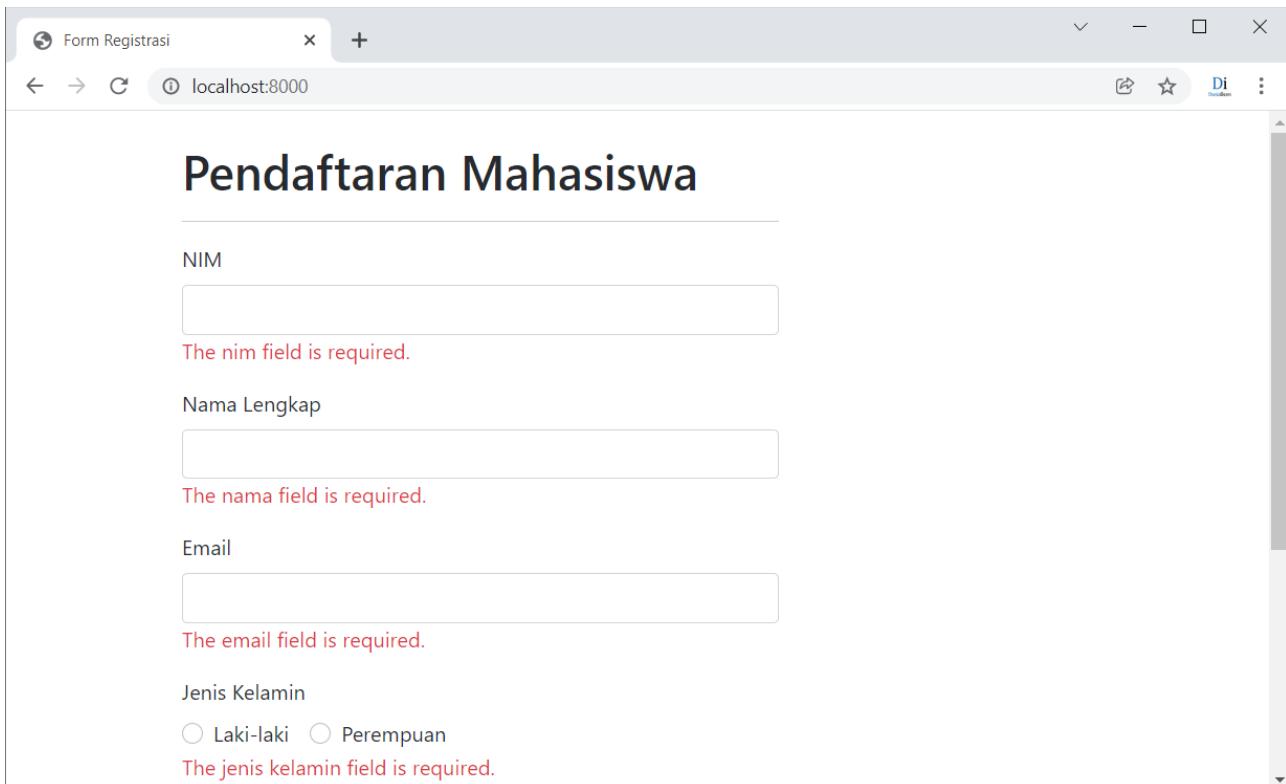
Modifikasi yang saya lakukan adalah, menghapus perulangan @foreach di bagian atas (yang sebelumnya kita pakai untuk menampilkan semua pesan error), lalu menambah perintah @error di bagian bawah setiap inputan form. Sebagai contoh, di baris 23 – 25 terdapat kode berikut:

```

@error('nim')
    <div class="text-danger">{{ $message }}</div>
@enderror
```

Kode ini dipakai untuk menampilkan pesan error dari inputan **nim** saja. Saya memakai tambahan tag `<div class="text-danger">` untuk mempercantik tampilan teks error, yakni memakai class `.text-danger` bawaan Bootstrap agar teks error tampil berwarna merah.

Hal yang sama juga ditambah untuk semua inputan lain, kecuali **alamat** yang memang tidak memiliki pesan error karena tidak memiliki syarat validasi. Berikut tampilan error:



Gambar: Tampilan pesan error dibagian bawah setiap inputan

Untuk lebih mempercantik, Bootstrap menyediakan class `.is-invalid` yang bisa di tambah ke dalam tag `<input type="text">`. Ini dipakai untuk membuat efek border warna merah ke kotak inputan serta tanda seru di sisi paling kanan. Berikut contoh penulisannya:

```
<input type="text" class="form-control is-invalid" id="nim" name="nim">
```

Namun class `.is-invalid` ini seharusnya hanya akan tampil jika inputan form berisi pesan error. Jika tidak, maka class ini tidak perlu ditulis.

Agar logika ini bisa dipakai, kita bisa tempatkan penulisan class `.is-invalid` di dalam blok `@error()`, kurang lebih mirip seperti cara menampilkan pesan error sebelumnya:

```
<input type="text" class="form-control @error('nim') is-invalid @enderror" id="nim" name="nim">
```

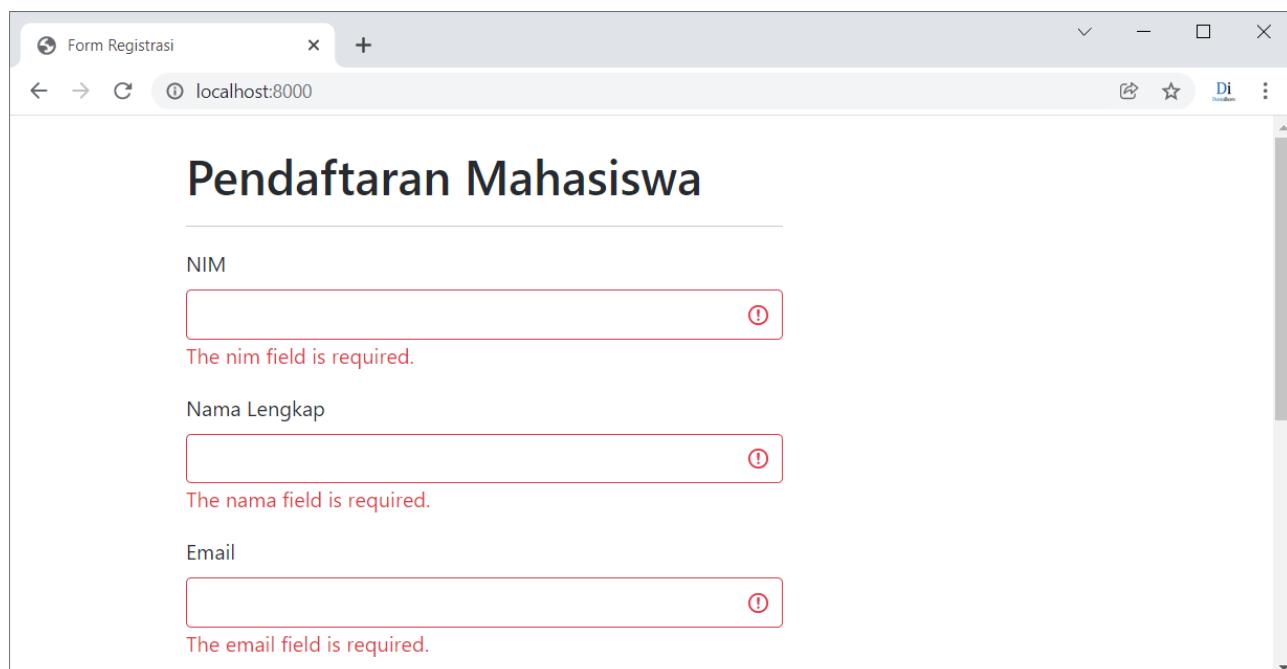
Dengan penulisan seperti ini, maka class `.is-invalid` hanya ditambah jika terdapat error di inputan **nim**. Jika tidak ada error, teks `.is-invalid` tidak akan tampil.

Berikut potongan kode program untuk view `form-pendaftaran.blade.php` dengan class `.is-invalid` ke inputan **nim**, **nama** dan **email**.

`resources/views/form-pendaftaran.blade.php`

```
1 <div class="mb-3">
2   <label class="form-label" for="nim">NIM</label>
3   <input type="text" class="form-control @error('nim') is-invalid @enderror"
```

```
4   id="nim" name="nim">
5     @error('nim')
6       <div class="text-danger">{{ $message }}</div>
7     @enderror
8   </div>
9
10  <div class="mb-3">
11    <label class="form-label" for="nama">Nama Lengkap</label>
12    <input type="text" class="form-control @error('nama') is-invalid @enderror"
13      id="nama" name="nama">
14    @error('nama')
15      <div class="text-danger">{{ $message }}</div>
16    @enderror
17  </div>
18
19  <div class="mb-3">
20    <label class="form-label" for="email">Email</label>
21    <input type="text" class="form-control @error('email') is-invalid @enderror"
22      id="email" name="email">
23    @error('email')
24      <div class="text-danger">{{ $message }}</div>
25    @enderror
26  </div>
```



Gambar: Tampilan error dengan tambahan class .is-invalid

Hasilnya, tampilan error menjadi lebih menarik dan hanya tampil jika terdapat error.

17.7. Repopulate Form

Saat ini, jika terdapat inputan form yang tidak lolos validasi, halaman akan *redirect* dan teks yang sudah diisi akan kembali kosong. Seharusnya, teks tersebut tetap ada dan user tinggal

perbaiki inputan yang salah. Inilah yang dimaksud dengan proses **repopulate form**. Laravel sudah menyediakan cara praktis untuk membuatnya.

Pada saat proses *redirect* (jika ada inputan yang gagal validasi), selain pesan error, inputan form sebelumnya juga disimpan ke dalam **session** dan bisa kita akses menggunakan method **old()** dari dalam dalam view.

Method **old()** butuh sebuah argument berupa nama inputan form. Sebagai contoh, agar ketika proses *redirect* inputan **nim** tidak kembali kosong, saya bisa tulis sebagai berikut:

```
<input type="text" name="nim" value="{{ old('nim') }}">
```

Untuk inputan bertipe text, atribut **value** berfungsi memberikan nilai awal. Inilah yang bisa kita isi dengan hasil pemanggilan method **old()**.

Jika pada saat form ditampilkan tidak ada session, maka method **old('nim')** tidak akan mengembalikan nilai apa-apa. Namun apabila form ditampilkan sebagai hasil *redirect*, maka method **old('nim')** akan mengambil nilai inputan **nim** yang tersimpan di session.

Bagaimana dengan inputan **nama** dan **email**? Tidak masalah, cukup ganti nilai argument dari method **old()** sebagai berikut:

```
<input type="text" name="nama" value="{{ old('nama') }}">
<input type="text" name="email" value="{{ old('email') }}">
```

Begitu juga untuk inputan **alamat** yang berbentuk textarea:

```
<textarea name="alamat">{{ old('alamat') }}</textarea>
```

Untuk inputan textarea, isi dari teks berada di antara tag **<textarea>** dan **</textarea>**.

Yang butuh sedikit trik adalah untuk inputan *checkbox*, *radio*, dan *select*. Kita harus rancang sebuah kode program agar ketika form di-*redirect*, inputan yang sudah dipilih tidak bertukar.

Untuk inputan *checkbox* dan *radio*, agar bisa terpilih kita harus tambah atribut **checked** ke dalam tag **<input>**. Sebagai contoh, supaya pilihan *checkbox* "laki-laki" langsung terpilih pada saat form tampil, bisa ditulis sebagai berikut:

```
<input type="radio" name="jenis_kelamin" value="L" checked>
```

Masalahnya, yang disimpan di dalam **old('jenis_kelamin')** bukanlah nilai 'checked', tapi hasil inputan form yang ada di dalam atribut **value**, yakni "L". Jadi, bagaimana mengakali masalah ini?

Salah satu solusi bisa menggunakan kondisi **if**. Yakni jika isi dari **old('jenis_kelamin')** adalah "L", maka tampilkan teks **checked** ke dalam tag **<input>**. Kita bisa saja menggunakan perintah **if** bawaan PHP maupun perintah **@if** dari blade, namun sebagai alternatif yang lebih singkat, bisa memakai operator conditional " ? : " seperti contoh berikut:

```
<input type="radio" name="jenis_kelamin" value="L"
{{ old('jenis_kelamin')=='L' ? 'checked': '' }} >
```

Perintah `old('jenis_kelamin')=='L' ? 'checked': ''` bisa dibaca: Apakah hasil dari `old('jenis_kelamin')` adalah 'L'? jika iya (**true**) tampilkan teks 'checked', jika tidak (**false**) tampilkan string kosong ''.

Hal yang sama jika saya tulis untuk inputan `jenis_kelamin` perempuan:

```
<input type="radio" name="jenis_kelamin" value="P"
{{ old('jenis_kelamin')=='P' ? 'checked': '' }} >
```

Untuk inputan **jurusang** yang menggunakan tag `<select>`, teknik yang dipakai sama saja, tapi kali ini teks yang kita gunakan adalah 'selected':

```
<select name="jurusan" id="jurusan">
<option value="Teknik Informatika"
{{ old('jurusan')=='Teknik Informatika' ? 'selected': '' }} >Teknik Informatika
</option>
<option value="Sistem Informasi"
{{ old('jurusan')=='Sistem Informasi' ? 'selected': '' }} >Sistem Informasi
</option>
<option value="Ilmu Komputer"
{{ old('jurusan')=='Ilmu Komputer' ? 'selected': '' }} >Ilmu Komputer
</option>
<option value="Teknik Komputer"
{{ old('jurusan')=='Teknik Komputer' ? 'selected': '' }} >Teknik Komputer
</option>
<option value="Teknik Telekomunikasi"
{{ old('jurusan')=='Teknik Telekomunikasi' ? 'selected': '' }} >
Teknik Telekomunikasi
</option>
</select>
```

Untuk inputan **jurusang**, nilainya adalah salah satu dari nama jurusan yang ada. Setelah proses *redirect*, dari 5 nama jurusan ini hanya akan ada 1 tag `<option>` yang memiliki atribut `selected`.

Silahkan modifikasi view form-pendaftaran dengan menambah kode-kode ini, seharusnya inputan form tidak akan kosong setelah proses *redirect*.

Checked / Selected Blade Directives

Laravel 9 membawa fitur baru untuk mempermudah proses re-populate inputan form, yakni dengan perintah `@checked` untuk inputan checkbox dan radio, serta perintah `@selected` untuk tag `<option>`.

Dalam contoh sebelum ini, saya memakai operator conditional " ? : " untuk menampilkan atribut `checked` di inputan radio button:

```
{{ old('jenis_kelamin')=='L' ? 'checked': '' }}
```

Dalam Laravel 9, kode di atas bisa diganti menjadi:

```
@checked(old('jenis_kelamin') == 'L')
```

Makna dari kode ini sebenarnya sama saja, dimana jika `old('jenis_kelamin')` berisi nilai "L", maka tampilkan teks `checked`.

Untuk perintah `@selected`, cara penulisannya juga mirip. Hanya saja sekarang dipakai ke dalam tag `<option>` di dalam tag `<select>`. Sebagai contoh, sebelumnya saya menggunakan perintah berikut:

```
{{ old('jurusan')=='Teknik Informatika' ? 'selected': '' }}
```

Sekarang bisa ditulis menjadi:

```
@selected(old('jurusan') == 'Teknik Informatika')
```

Berikut kode program yang diperlukan untuk proses re-populate inputan jenis kelamin dan pilihan jurusan:

```
1 <input type="radio" name="jenis_kelamin" value="L"
2 @checked(old('jenis_kelamin') == 'L')>Laki-laki
3
4 <input type="radio" name="jenis_kelamin" value="P"
5 @checked(old('jenis_kelamin') == 'P')>Perempuan
6
7 <select name="jurusan">
8   <option value="Teknik Informatika"
9     @selected(old('jurusan') == 'Teknik Informatika')>Teknik Informatika
10    </option>
11    <option value="Sistem Informasi"
12      @selected(old('jurusan') == 'Sistem Informasi') >Sistem Informasi
13      </option>
14    <option value="Ilmu Komputer"
15      @selected(old('jurusan') == 'Ilmu Komputer') >Ilmu Komputer
16      </option>
17    <option value="Teknik Komputer"
18      @selected(old('jurusan') == 'Teknik Komputer') >Teknik Komputer
19      </option>
20    <option value="Teknik Telekomunikasi"
21      @selected(old('jurusan') == 'Teknik Telekomunikasi') >Teknik Telekomunikasi
22      </option>
23 </select>
```

Penggunaan perintah `@checked` dan `@selected` bisa sedikit mempersingkat penulisan kode program. Tapi menggunakan operator conditional seperti sebelumnya juga tidak salah.

17.8. Validator Class

Alur bahasan form processing dan form validation sebenarnya sudah selesai, dimana kita telah

mempelajari cara mengirim data form, membuat validasi, menampilkan pesan error dan repopulate form. Semua proses ini dilakukan secara langsung dari method `$request->validate()`.

Sebagai alternatif, proses validasi bisa juga dibuat menggunakan **Validator** class, yakni sebuah facade yang berada di `Illuminate\Support\Facades\Validator`. Class **Validator** dipakai jika kita ingin memiliki kontrol lebih terhadap setiap proses yang terjadi, salah satunya membuat pesan error sendiri.

Sebagai bahan praktik, saya akan menambah route baru:

`routes/web.php`

```
Route::post('/proses-form-validator', [MahasiswaController::class,
    'prosesFormValidator']);
```

Route ini disiapkan untuk mengakses method `prosesFormValidator()` milik `Mahasiswa Controller`. Tujuannya, saya ingin pemrosesan form dilakukan oleh method ini. Oleh karena itu, nilai atribut `action` dari tag `<form>` di dalam view `form-pendaftaran` juga harus di modifikasi:

`resources/views/form-pendaftaran.blade.php`

```
1 //...
2 <form action="{{url('/proses-form-validator')}}" method="POST">
3     @csrf
4     <div class="mb-3">
5         //...
```

Dengan penambahan ini, maka ketika form di submit data inputan form akan diproses oleh method `prosesFormValidator()`. Berikut struktur dasar dari method ini:

`app/Http/Controllers/MahasiswaController.php`

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Validator;
7
8 class MahasiswaController extends Controller
9 {
10     public function index()
11     {
12         return view('form-pendaftaran');
13     }
14
15     public function prosesForm(Request $request)
16     {
17         //..
18     }
}
```

```

19
20     public function prosesFormValidator(Request $request)
21     {
22         // Kode program akan kita tulis disini
23     }
24 }
```

Karena **Validator** class adalah sebuah *facade*, maka kita perlu tambahan perintah di baris 6 untuk proses import, yakni `use Illuminate\Support\Facades\Validator;`

Kode program untuk method `prosesFormValidator()` masih belum saya isi, namun perhatikan di bagian argument terdapat perintah type *hint* `Request $request`. Ini diperlukan karena di dalam method ini kita juga butuh **Request** object untuk mengambil nilai inputan form.

Proses instansiasi dari class **Validator** nantinya dilakukan dengan static method `Validator::make()`. Method ini butuh 3 buah argument:

- Argument 1 berupa associative array yang berisi hasil inputan form. Ini bisa kita akses dari **Request** object menggunakan method `$request->all()`.
- Argument 2 berupa associative array yang berisi syarat validasi, ini kurang lebih sama seperti yang kita buat sebelumnya.
- Argument 3 berupa associative array yang berisi pesan error jika tidak lolos validasi. Pesan error ini nantinya ditampilkan dari dalam view dengan fungsi `old()`.

Penggunaan **Validator** class juga mengharuskan kita membuat kode program untuk proses *redirect* serta pengiriman data session secara manual.

Berikut kode lengkap dari cara pembuatan validasi menggunakan **Validator** class:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Validator;
7
8 class MahasiswaController extends Controller
9 {
10     //...
11     //...
12
13     public function prosesFormValidator(Request $request)
14     {
15         $validator = Validator::make($request->all(), [
16             'nim'          => 'required|size:8',
17             'nama'         => 'required|min:3|max:50',
18             'email'        => 'required|email',
19             'jenis_kelamin' => 'required|in:P,L',
20             'jurusan'      => 'required',
21         ]);
22
23         if ($validator->fails())
24             return redirect('mahasiswa/tambah')
25                 ->withErrors($validator)
26                 ->withInput();
27
28         $data = $validator->validated();
29
30         // Simpan data ke database
31
32         return redirect('mahasiswa/tambah')
33             ->with('success', 'Data berhasil ditambahkan');
34     }
35 }
```

```

21     'alamat'      => '',
22 ],
23 [
24     'required'   => ':attribute wajib diisi.',
25     'size'        => ':attribute harus berukuran :size karakter.',
26     'max'         => ':attribute maksimal berisi :max karakter.',
27     'min'         => ':attribute minimal berisi :min karakter.',
28     'email'       => ':harus diisi dengan alamat email yang valid.',
29     'in'          => ':attribute yang dipilih tidak valid.',
30 ]
31 );
32
33 if ($validator->fails()) {
34     return redirect('/')->withErrors($validator)->withInput();
35 }
36 else {
37     echo $request->nim;           echo "<br>";
38     echo $request->nama;          echo "<br>";
39     echo $request->email;         echo "<br>";
40     echo $request->jenis_kelamin; echo "<br>";
41     echo $request->jurusan;       echo "<br>";
42     echo $request->alamat;
43 }
44 }
45
46 }

```

Di baris 15 saya menjalankan method `Validator::make()` yang hasilnya ditampung ke dalam variabel `$validator`. Variabel `$validator` di sini akan berisi sebuah object dari **Validator** class.

Sebagai argument pertama dari method `Validator::make()` adalah hasil dari pemanggilan `$request->all()`. Method `$request->all()` mengembalikan sebuah associative array yang berisi pasangan nama inputan form beserta nilainya. Nilai inilah yang akan di validasi oleh **Validator** class.

Sebagai argument kedua, diisi associative array dari syarat validasi. Penulisan array ini sama persis seperti syarat validasi pada contoh kita yang menggunakan `$request->validate()`.

Argument ketiga berupa associative array yang berisi pesan error. Sebagai key dari array ini adalah nama syarat validasi, dan nilainya berupa teks dari pesan error yang ingin ditampilkan.

Di dalam teks ini, kita bisa mengakses beberapa keyword khusus seperti `:attribute` yang nantinya akan diganti dengan nama inputan form, `:size` yang akan diganti dengan syarat jumlah karakter, serta `:max` dan `:min` yang nantinya akan berganti dengan angka maksimum dan minimum dari syarat validasi.

Sebagai contoh, jika syarat `required` untuk inputan **nim** tidak terpenuhi, maka pesan error yang tampil adalah: "nim wajib diisi". Atau jika syarat `max:50` dari inputan **nama** terlewati, maka pesan error yang tampil adalah: "nama maksimal berisi 50 karakter.", dst. Anda bisa menambah pesan error lain sesuai keinginan.

Mengenai format penulisan pesan error, akan kita bahas dalam bab berikutnya, yakni tentang **Localization**, sekaligus nanti juga di pelajari cara mengganti semua pesan error ke dalam Bahasa Indonesia (tidak harus dibuat satu per satu seperti ini).

Setelah mengisi ketiga argument untuk `Validator::make()`, pekerjaan belum selesai. Kita harus tulis kode program untuk menjalankan proses *redirect* secara manual, yakni dengan perintah di baris 33 – 35:

```
if ($validator->fails()) {  
    return redirect('/')->withErrors($validator)->withInput();  
}
```

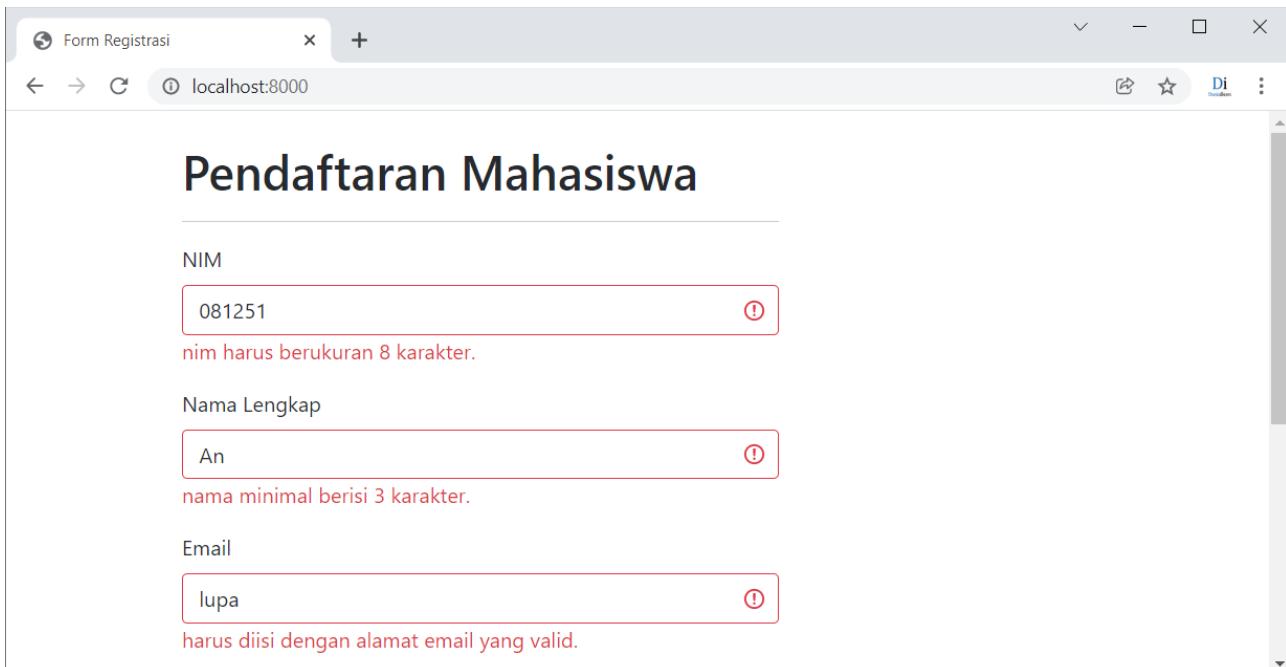
Method `$validator->fails()` akan mengembalikan nilai **true** jika terdapat inputan form yang tidak lolos syarat validasi. Jika ini terjadi, jalankan proses *redirect* ke halaman root dengan perintah `return redirect('/')`.

Selain itu, perintah `return redirect('/')` harus di *chaining* dengan method `withErrors($validator)` yang dipakai untuk membuat session berisi pesan error. Tanpa perintah ini, di dalam view nanti variabel `$errors` tidak berisi pesan apapun.

Selain itu saya juga men-chainning method `withInput()` yang dipakai untuk membuat session yang berisi nilai inputan form sebelumnya. Tanpa ini, method `old()` di dalam view juga akan berisi teks kosong, yang berarti kita tidak bisa melakukan *repopulate form*.

Apabila seluruh inputan form lolos validasi, maka method `$validator->fails()` mengembalikan nilai **false** dan blok else di baris 37 – 42 akan dijalankan. Blok else ini berisi kode program untuk menampilkan semua inputan form yang diakses dari **Request** object.

Silahkan coba input beberapa data yang salah ke dalam form, seharusnya pesan error yang tampil sekarang sudah menggunakan Bahasa Indonesia.



Gambar: Pesan error form tampil dengan bahasa indonesia

Agar lebih rapi, penulisan array yang berisi syarat validasi dan pesan error ini bisa kita pisah dalam variabel tersendiri, seperti contoh berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Validator;
7
8 class MahasiswaController extends Controller
9 {
10     //...
11     //...
12
13     public function prosesFormValidator(Request $request)
14     {
15         $rules = [
16             'nim'          => 'required|size:8',
17             'nama'         => 'required|min:3|max:50',
18             'email'        => 'required|email',
19             'jenis_kelamin' => 'required|in:P,L',
20             'jurusan'      => 'required',
21             'alamat'       => '',
22         ];
23
24         $error_message = [
25             'required'   => ':attribute wajib diisi.',
26             'size'        => ':attribute harus berukuran :size karakter.',
27             'max'         => ':attribute maksimal berisi :max karakter.',
28         ];
29
30         $validator = Validator::make($request->all(), $rules);
31
32         if ($validator->fails())
33         {
34             return redirect('register')
35                 ->withErrors($validator)
36                 ->withInput();
37         }
38
39         $mahasiswa = new Mahasiswa();
40         $mahasiswa->nim = $request->nim;
41         $mahasiswa->nama = $request->nama;
42         $mahasiswa->email = $request->email;
43         $mahasiswa->jenis_kelamin = $request->jenis_kelamin;
44         $mahasiswa->jurusan = $request->jurusan;
45         $mahasiswa->alamat = $request->alamat;
46
47         $mahasiswa->save();
48
49         return redirect('register')
50             ->with('success', 'Data berhasil disimpan');
51     }
52 }
```

```

28      'min'          => ':attribute minimal berisi :min karakter.',  

29      'email'        => 'harus diisi dengan alamat email yang valid.',  

30      'in'           => ':attribute yang dipilih tidak valid.',  

31  ];  

32  

33  $validator = Validator::make($request->all(), $rules, $error_message);  

34  

35  if ($validator->fails()) {  

36      return redirect('/')->withErrors($validator)->withInput();  

37  }  

38  else {  

39      echo $request->nim;                echo "<br>";  

40      echo $request->nama;               echo "<br>";  

41      echo $request->email;              echo "<br>";  

42      echo $request->jenis_kelamin;    echo "<br>";  

43      echo $request->jurusan;           echo "<br>";  

44      echo $request->alamat;  

45  }  

46 }  

47  

48 }

```

Dalam kode program kali ini, saya membuat variabel \$rules yang berisi associative array dari syarat validasi, kemudian membuat variabel \$error_message yang berisi associative array dari pesan error. Kedua variabel ini kemudian diinput ke dalam argument method Validator::make() di baris 33, yakni perintah:

```
$validator = Validator::make($request->all(), $rules, $error_message);
```

Pemisahan ini hanya agar kode program kita menjadi lebih rapi saja. Untuk sisa kode program sama seperti contoh sebelumnya.

17.9. Membuat Custom FormRequest Class

Untuk keperluan yang lebih advanced lagi, Laravel memiliki fitur yang dinamakan **custom FormRequest class**. Tujuannya untuk memindahkan proses validasi dari dalam controller.

Salah satu kegunaan dari fitur ini adalah, kita bisa memakai ulang kode validasi di banyak tempat (di berbagai file controller), selama struktur form yang dipakai juga sama. Misalnya untuk project sistem informasi sekolah dimana form pendaftaran untuk siswa SD, SMP dan harus dibuat terpisah, padahal inputannya sama saja. Daripada membuat 3 buah kode validasi, kita bisa rancang sebuah **custom FormRequest class** dan membuat validasi di dalam class ini.

Custom Request class merupakan sebuah file terpisah. Dan seperti biasa, jika kita perlu membuat sebuah file baru, itu adalah tugas dari `php artisan`. Berikut format dasar perintahnya:

```
php artisan make:request <nama_request_class>
```

Dalam praktek kali ini saya akan membuat sebuah class bernama `DaftarMahasiswa`, sehingga perintah yang diperlukan adalah:

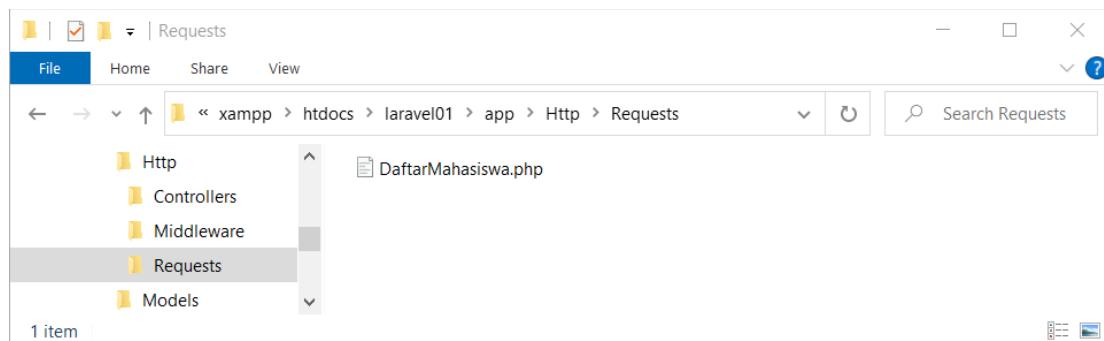
```
php artisan make:request DaftarMahasiswa
```

```
C:\xampp\htdocs\laravel01>php artisan make:request DaftarMahasiswa
Request created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Perintah untuk membuat request class

Perintah ini akan membuat sebuah folder bernama `Requests` di `app\Http\`, yang di dalamnya terdapat file `DaftarMahasiswa.php`.



Gambar: Isi folder app\Http\Requests\

Berikut isi dari file ini:

app/Http/Requests/DaftarMahasiswa.php

```
1 <?php
2
3 namespace App\Http\Requests;
4
5 use Illuminate\Foundation\Http\FormRequest;
6
7 class DaftarMahasiswa extends FormRequest
8 {
9     /**
10      * Determine if the user is authorized to make this request.
11      *
12      * @return bool
13      */
14     public function authorize()
15     {
16         return false;
17     }
18
19     /**
20      * Get the validation rules that apply to the request.
21      *
```

```
22     * @return array
23     */
24     public function rules()
25     {
26         return [
27             //
28         ];
29     }
30 }
```

Di baris 3 terdapat perintah untuk pembuatan namespace `App\Http\Requests`, yang diikuti dengan proses import class `Illuminate\Foundation\Http\FormRequest` di baris 5.

Kemudian di baris 7 adalah kode untuk pembuatan class `DaftarMahasiswa` yang di extends dari class `FormRequest`. Secara bawaan, sudah ada 2 method dalam class ini yaitu `authorize()` dan `rules()`.

Method `authorize()` di pakai untuk menangani hak akses, yakni apakah user berhak melakukan input form atau tidak. Untuk saat ini kita tidak akan membahas tentang hak akses, namun kode di dalam method `authorize()` ini harus di ubah menjadi `return true` agar tidak bermasalah di hak akses:

```
public function authorize()
{
    return true;
}
```

Method `rules()` adalah tempat untuk menulis syarat validasi. Method ini harus mengembalikan associative array seperti contoh berikut:

```
public function rules()
{
    return [
        'nim'          => 'required|size:8',
        'nama'         => 'required|min:3|max:50',
        'email'        => 'required|email',
        'jenis_kelamin' => 'required|in:P,L',
        'jurusan'      => 'required',
        'alamat'       => '',
    ];
}
```

Kode ini sama persis seperti syarat validasi yang kita pakai sebelumnya. Save perubahan untuk method `authorize()` dan `rules()` ini.

Agar kita tidak perlu menimpa kode dari praktik sebelumnya, saya akan membuat route baru untuk tujuan pengiriman form, yakni:

```
routes/web.php
Route::post('/proses-form-request', [MahasiswaController::class,
```

```
'prosesFormRequest']);
```

Route ini disiapkan untuk mengakses method prosesFormRequest() di dalam MahasiswaController. Kemudian nilai atribut action dari tag <form> juga harus di modifikasi:

resources/views/form-pendaftaran.blade.php

```
1 //...
2 <form action="{{url('/proses-form-request')}}" method="POST">
3     @csrf
4     <div class="mb-3">
5 //...
```

Dengan perubahan ini, maka ketika form di submit, data inputan form akan diproses oleh method prosesFormRequest() di dalam MahasiswaController.

Berikut kode program untuk prosesFormRequest():

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Validator;
7 use App\Http\Requests\DaftarMahasiswa;
8
9 class MahasiswaController extends Controller
10 {
11     //...
12     //...
13
14     public function prosesFormRequest(DaftarMahasiswa $request)
15     {
16         $validateData = $request->validated();
17         dump($validateData);
18     }
19 }
20 }
```

Di bagian atas controller, kita harus menambah 1 baris perintah untuk proses import **FormRequest** class, yakni kode use App\Http\Requests\DaftarMahasiswa di baris 7.

Kemudian, type hint untuk argument method prosesFormRequest() ditulis sebagai prosesFormRequest(DaftarMahasiswa \$request). Inilah cara dari penggunaan custom FormRequest class, dimana kita men-inject variabel \$request sebagai instance dari object DaftarMahasiswa.

Untuk menjalankan proses validasi, cukup menulis \$request->validated() seperti di baris 16. Perintah ini akan mengakses syarat validasi yang kita tulis di class DaftarMahasiswa dan langsung melakukan proses redirect jika ada syarat validasi yang gagal. Terakhir saya men-

dump variabel `$validateData` sekedar menampilkan hasil inputan form jika validasi berhasil dilewati.

Dengan membuat custom **FormRequest** class seperti ini, kode di controller menjadi lebih bersih, bahkan hanya ada 1 baris saja.

Dalam bab kali ini kita telah pelajari cara memproses nilai inputan form beserta proses validasi. Meskipun terlihat agak kompleks, tapi proses ini jauh lebih sederhana dibandingkan jika dibuat sendiri menggunakan PHP native.

Materi yang kelihatannya agak menggantung ada di syarat validasi yang belum banyak di bahas, karena jumlahnya memang sangat banyak (lebih dari 60 syarat), selain itu setiap syarat mesti dengan contoh kode program agar cara penggunaannya lebih jelas. Namun hingga akhir buku nanti, kita akan bahas berapa syarat lanjutan, seperti pemeriksaan inputan ke database (agar tidak ada data yang double).

Bab berikutnya, kita akan masuk ke materi tentang **Localization**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

18. Localization

Dalam beberapa project, kadang kita harus membuat website dengan berbagai macam bahasa, atau setidaknya untuk bahasa inggris dan juga bahasa indonesia. Laravel menyediakan fitur **localization** sebagai solusi.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css** karena kita akan memakai sedikit kode Bootstrap.

18.1. Pengertian Localization

Localization adalah istilah programming yang merujuk ke cara men-translate sebuah aplikasi ke dalam bahasa tertentu. Dalam materi kita, aplikasi yang dimaksud tentunya berbentuk web.

Sebagai contoh, web **Traveloka** memiliki halaman dengan bahasa indonesia di www.traveloka.com/id-id/, serta halaman dalam bahasa inggris di www.traveloka.com/en/. Kedua web ini bisa saja dibuat terpisah, 1 tim membuat website bahasa indonesia dan tim lain membuat website bahasa inggris.

Namun yang jadi masalah, jika tim bahasa indonesia ingin menambah menu baru, maka tim bahasa inggris juga harus menyamakan struktur kode program untuk menu tersebut (mencakup kode HTML, CSS, PHP, dll). Hal seperti ini tentu tidak efisien, akan lebih baik jika struktur web bisa di *sharing* untuk kedua website. Inilah yang bisa kita lakukan dari fitur **Localization** Laravel.

Prinsip kerja dari proses localization di Laravel adalah sebagai berikut:

Pertama, buat semacam "kamus" yang berisi kata atau kalimat untuk setiap bahasa. Kamus ini disimpan dalam file terpisah namun menggunakan nama key yang sama (berbentuk associative array). Misalnya kita ingin merancang tombol "**Daftar**" / "**Register**", maka bisa buat sebuah file bernama **indonesia.php** yang berisi "**tombol**" => "**Daftar**", dan file **english.php**

yang berisi array "tombol" => "Register".

Kemudian, di dalam view tulis tag <button> yang merujuk ke nama key yang sudah disiapkan (bukan langsung menulis teksnya). Seperti contoh berikut:

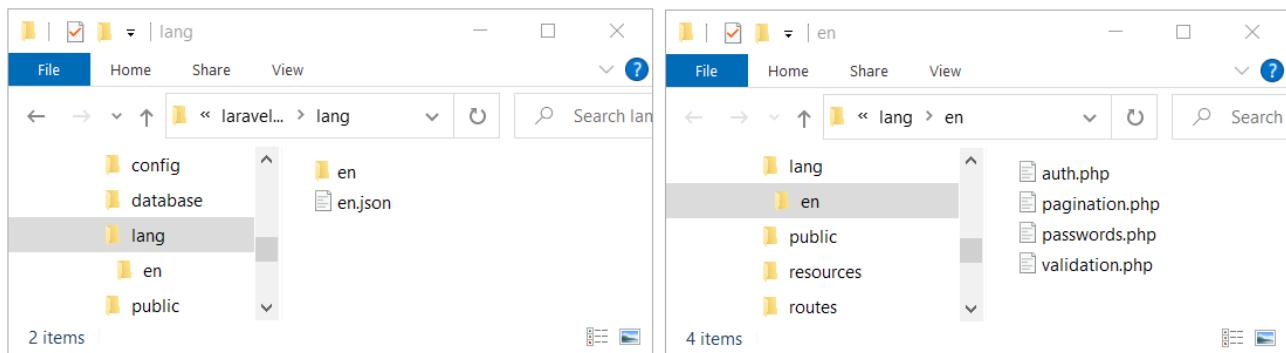
```
<button>{{ __('tombol') }}</button>
```

Tanda __(), yakni karakter underscore dua kali adalah function khusus di blade yang berfungsi untuk menampilkan teks hasil localization.

Langkah terakhir, rancang kode di route atau controller agar ketika diakses, tombol tersebut tampil dalam bahasa yang sesuai. Kita akan bahas contoh prakteknya secara bertahap.

18.2. Folder lang

Lokasi "kamus" localization ada di folder lang. Folder ini berisi 1 file en.json dan 1 folder en. Di dalam folder en terdapat 4 file: auth.php, pagination.php, passwords.php dan validation.php.



Gambar: Isi folder lang (kiri), dan isi folder lang\en (kanan)

Di Laravel versi 8 ke bawah, folder lang berada di dalam folder resources. Namun di Laravel 9 dibawa keluar dan langsung berada di bawah folder utama.

Nama folder en merupakan kode bahasa default yang dipakai Laravel. Jika ingin membuat localization untuk bahasa lain, tempatkan dalam folder terpisah dengan kode bahasa yang sesuai. Misalnya untuk bahasa indonesia, kodennya adalah id (akan kita buat sesaat lagi).

Kembali ke folder en, mari buka salah satu file, yakni auth.php:

lang/en/auth.php

```
1  <?php
2
3  return [
4
5      /*
6      |-----|
7      | Authentication Language Lines |
8      |-----|
```

```

9     | The following language lines are used during authentication for various
10    | messages that we need to display to the user. You are free to modify
11    | these language lines according to your application's requirements.
12    |
13    */
14
15
16    'failed' => 'These credentials do not match our records.',
17    'password' => 'The provided password is incorrect.',
18    'throttle' => 'Too many login attempts. Please try again in :seconds ...',
19
20 ];

```

File ini pada dasarnya berisi sebuah *associative array* dengan format sebagai berikut:

```

return [
    'key1' => 'value1',
    'key2' => 'value2',
    'key3' => 'value3',
    //...
];

```

Inilah cara penulisan kamus untuk localization. Nantinya di dalam view, kita bisa mengakses key array untuk proses translasi.

18.3. Function __() dan Perintah @lang

Sebagai praktek, silahkan buat file baru bernama `test.php` di dalam folder `lang\en`, lalu isi dengan kode berikut:

```

lang/en/test.php

1 <?php
2
3 return [
4     'judul' => 'Belajar Laravel Uncover',
5 ];
6

```

Di sini saya membuat kamus dengan key 'judul' yang ketika diakses akan menampilkan teks 'Belajar Laravel Uncover'.

Kemudian silahkan buka view `welcome.blade.php`, yakni view bawaan Laravel. Scroll hingga tag pembuka `<body>`, lalu tambahkan baris di bawah ini:

```

<h1 style="text-align: center"> {{ __('test.judul') }}</h1>

```

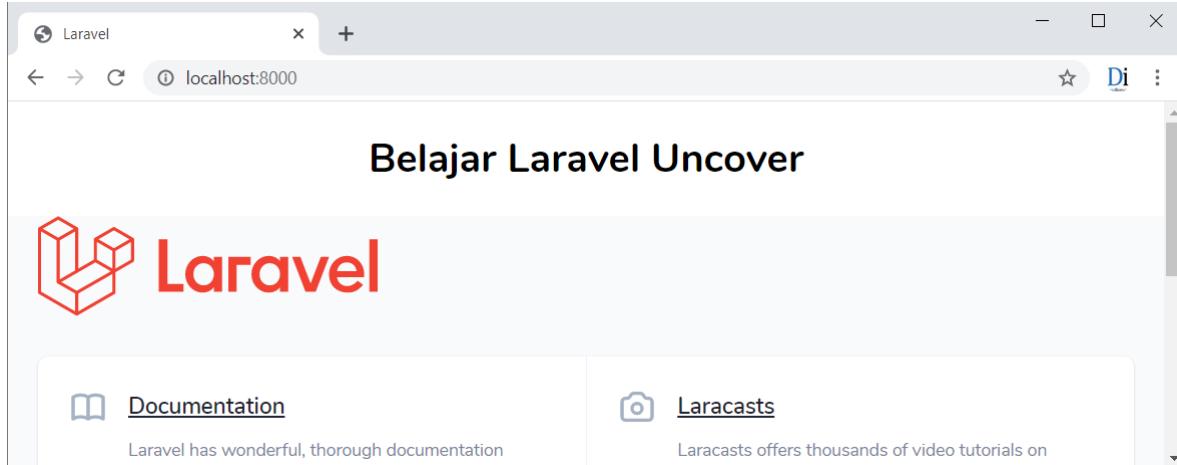
Berikut posisi penambahannya:

Localization

```
22 </head>
23 <body class="antialiased">
24     <h1 style="text-align: center"> {{ __('test.judul') }}</h1> ←
25     <div class="relative flex items-top justify-center min-h-screen bg-gray-100">
26         @if (Route::has('login'))
27             <div class="hidden fixed top-0 right-0 px-6 py-4 sm:block">
```

Gambar: Potongan kode program di welcome.blade.php

Save, dan jalankan di web browser:



Gambar: Tampilan halaman localhost:8000

Hasilnya, tampil teks 'Belajar Laravel Uncover' seperti yang tertulis di dalam kamus.

Function `__()` adalah fungsi khusus blade untuk menampilkan teks localization. Argument dari fungsi ini berupa gabungan nama file localization tanpa akhiran `.php`, serta nama key kamus yang ingin diakses.

Karena nama file kita adalah `test.php` dan nama key-nya '`judul`', maka ditulis menjadi `{{ __('test.judul') }}`. Alternatif cara lain bisa dengan perintah `@lang` seperti berikut:

```
@lang('test.judul')
```

Jika menggunakan perintah `@lang`, kita tidak perlu menambah tanda kurung kurawal dua kali sebagaimana fungsi `__()`.

Untuk keperluan yang lebih kompleks, struktur kamus localization juga bisa dibuat dalam bentuk nested array atau array di dalam array. Misalnya file `test.php` berisi kode berikut:

lang/en/test.php

```
1 <?php
2
3 return [
4     'home' => [
5         'judul' => 'Belajar Laravel Uncover',
6     ]
7];
```

Maka untuk mengaksesnya, bisa menggunakan perintah `@lang('test.home.judul')`. Dimana tanda titik dipakai sebagai pemisah key array.

18.4. Membuat 2 Bahasa Localization

Contoh sebelum ini baru sekedar mengakses kamus localization untuk 1 bahasa saja. Idealnya, terdapat 2 bahasa atau lebih untuk praktik localization. Sebagai bahan latihan, kita akan translate form pendaftaran hasil dari bab sebelumnya ke dalam 2 bahasa bahasa: bahasa inggris dan bahasa indonesia.

Pertama, buat file `form.php` di folder `lang\en` dan isi dengan kode berikut:

`lang/en/form.php`

```

1  <?php
2
3  return [
4      'judul' => 'Student Registration',
5      'input' => [
6          'nim' => 'NIM',
7          'nama_lengkap' => 'Full Name',
8          'email' => 'Email',
9          'jenis_kelamin' => 'Gender',
10         'pilihan_jenis_kelamin' => [
11             'laki_laki' => 'Male',
12             'perempuan' => 'Female',
13         ],
14         'jurusan' => 'Major',
15         'alamat' => 'Address',
16         'tombol' => 'Register',
17     ]
18 ];

```

Di sini saya membuat kamus localization versi bahasa inggris dalam bentuk nested array. Baris 4 merupakan kode untuk teks 'judul' yang nantinya berada di bagian atas form. Kemudian antara baris 6 – 16 berupa teks inputan form yang semuanya ada di dalam sub-array 'input'. Khusus pilihan jenis kelamin, disimpan lagi dalam nested array 'pilihan_jenis_kelamin'.

Sebagai latihan, anda bisa coba modifikasi view `form-pendaftaran.blade.php` dari bab sebelumnya, lalu tambah kode localization untuk bagian judul form serta inputan nama form berdasarkan key dari file `form.php` di atas.

Baik, berikut kode lengkap dari `form-pendaftaran.blade.php`:

`resources/views/form-pendaftaran.blade.php`

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">

```

```
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <link href="/css/bootstrap.min.css" rel="stylesheet">
8   <title>Form Registrasi</title>
9   </head>
10  <body>
11
12  <div class="container pt-4 bg-white">
13    <div class="row">
14      <div class="col-md-8 col-xl-6">
15        <h1>{{ __('form.judul') }}</h1>
16        <hr>
17
18        <form action="{{url('/proses-form')}}" method="POST">
19          @csrf
20          <div class="mb-3">
21            <label class="form-label" for="nim">
22              {{ __('form.input.nim') }}
23            </label>
24            <input type="text" id="nim" name="nim" value="{{ old('nim') }}"
25              class="form-control @error('nim') is-invalid @enderror">
26            @error('nim')
27              <div class="text-danger">{{ $message }}</div>
28            @enderror
29          </div>
30
31          <div class="mb-3">
32            <label class="form-label" for="nama">
33              {{ __('form.input.nama_lengkap') }}
34            </label>
35            <input type="text" id="nama" name="nama" value="{{ old('nama') }}"
36              class="form-control @error('nama') is-invalid @enderror">
37            @error('nama')
38              <div class="text-danger">{{ $message }}</div>
39            @enderror
40          </div>
41
42          <div class="mb-3">
43            <label class="form-label" for="email">
44              {{ __('form.input.email') }}
45            </label>
46            <input type="text" id="email" name="email" value="{{ old('email') }}"
47              class="form-control @error('email') is-invalid @enderror">
48            @error('email')
49              <div class="text-danger">{{ $message }}</div>
50            @enderror
51          </div>
52
53          <div class="mb-3">
54            <label class="form-label">
55              {{ __('form.input.jenis_kelamin') }}
56            </label>
57            <div class="d-flex">
58              <div class="form-check me-3">
59                <input class="form-check-input" type="radio" name="jenis_kelamin"
```

```

60      id="laki_laki" value="L"
61      {{ old('jenis_kelamin')=='L' ? 'checked': '' }}>
62      <label class="form-check-label" for="laki_laki">
63          {{ __('form.input.pilihan_jenis_kelamin.laki_laki') }}
64      </label>
65  </div>
66  <div class="form-check">
67      <input class="form-check-input" type="radio" name="jenis_kelamin"
68      id="perempuan" value="P"
69      {{ old('jenis_kelamin')=='P' ? 'checked': '' }}>
70      <label class="form-check-label" for="perempuan">
71          {{ __('form.input.pilihan_jenis_kelamin.perempuan') }}
72      </label>
73  </div>
74  </div>
75  @error('jenis_kelamin')
76      <div class="text-danger">{{ $message }}</div>
77  @enderror
78</div>
79
80<div class="mb-3">
81    <label class="form-label" for="jurusan">
82        {{ __('form.input.jurusan') }}
83    </label>
84    <select class="form-select" name="jurusan" id="jurusan"
85    value="{{ old('jurusan') }}>
86        <option value="Teknik Informatika"
87        {{ old('jurusan')=='Teknik Informatika' ? 'selected': '' }} >
88        Teknik Informatika
89        </option>
90        <option value="Sistem Informasi"
91        {{ old('jurusan')=='Sistem Informasi' ? 'selected': '' }} >
92        Sistem Informasi
93        </option>
94        <option value="Ilmu Komputer"
95        {{ old('jurusan')=='Ilmu Komputer' ? 'selected': '' }} >
96        Ilmu Komputer
97        </option>
98        <option value="Teknik Komputer"
99        {{ old('jurusan')=='Teknik Komputer' ? 'selected': '' }} >
100       Teknik Komputer
101       </option>
102       <option value="Teknik Telekomunikasi"
103       {{ old('jurusan')=='Teknik Telekomunikasi' ? 'selected': '' }} >
104       Teknik Telekomunikasi
105       </option>
106   </select>
107   @error('jurusan')
108      <div class="text-danger">{{ $message }}</div>
109  @enderror
110</div>
111
112<div class="mb-3">
113    <label class="form-label" for="alamat">
114        {{ __('form.input.alamat') }}
```

Localization

```
115      </label>
116      <textarea class="form-control" id="alamat" rows="3"
117          name="alamat">{{ old('alamat') }}</textarea>
118    </div>
119
120    <button type="submit" class="btn btn-primary mb-2">
121        {{ __('form.input.tombol') }}
122    </button>
123  </form>
124
125  </div>
126 </div>
127 </div>
128
129 </body>
130 </html>
```

Kode yang ada memang cukup panjang. Penjelasan lengkap dari setiap baris sudah kita bahas pada bab sebelumnya. Yang berubah hanya bagian judul form, nama inputan, serta pilihan jenis kelamin.

Untuk setiap kata, saya mengakses kamus localization dengan kode {{ __('form.judul') }} untuk judul, {{ __('form.input.nama_lengkap') }} untuk inputan nama lengkap, hingga {{ __('form.input.pilihan_jenis_kelamin.perempuan') }} untuk jenis kelamin perempuan.

Langkah selanjutnya, buat route untuk mengakses view dengan kode berikut:

routes/web.php

```
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\MahasiswaController;
5
6  Route::get('/form-pendaftaran',[MahasiswaController::class,'formPendaftaran']);
7  Route::post('/proses-form', [MahasiswaController::class,'prosesForm']);
```

Route pertama untuk menampilkan form, dan route kedua untuk memproses inputan form. Karena kita belum memiliki `MahasiswaController`, silahkan buat controller ini dari perintah:

```
php artisan make:controller MahasiswaController
```

Dan isi `MahasiswaController` dengan kode berikut:

app/Http/Controllers/MahasiswaController.php

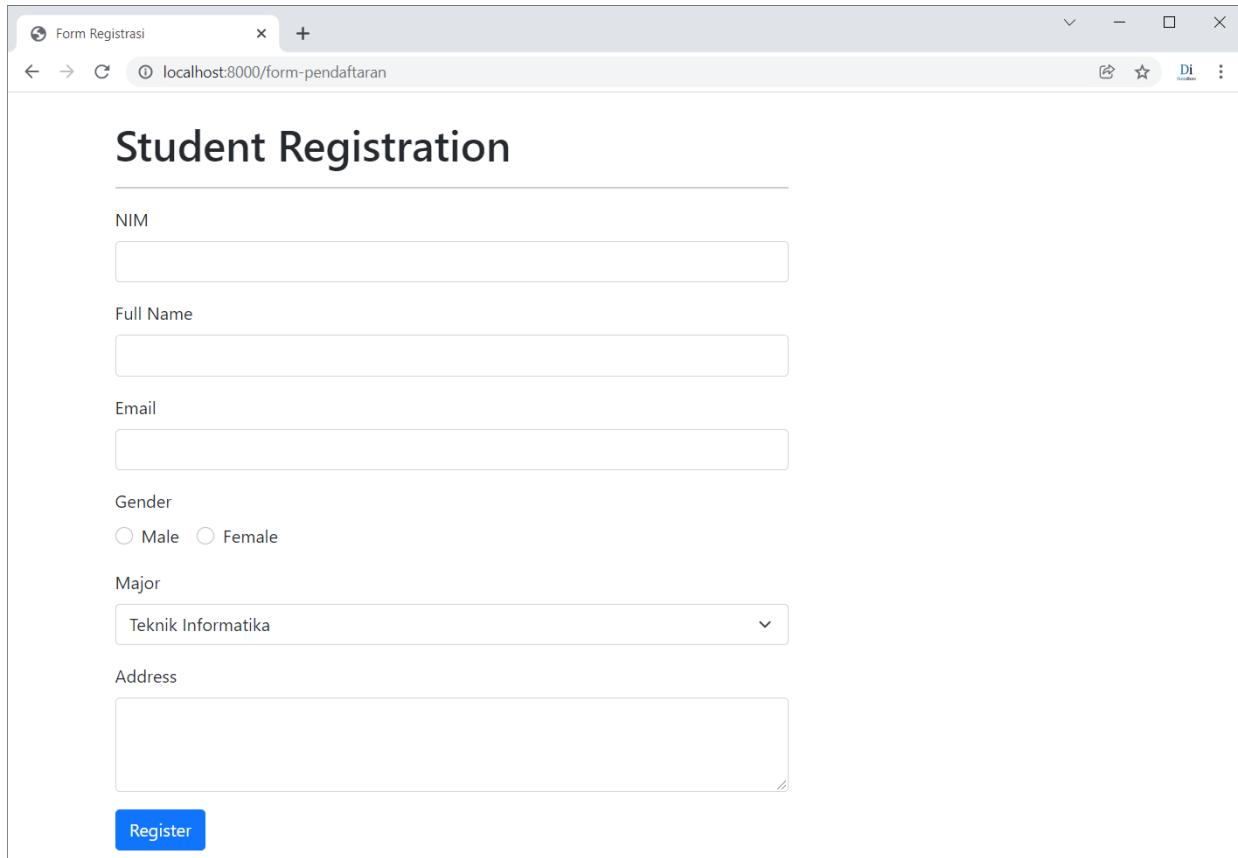
```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  class MahasiswaController extends Controller
```

Localization

```
7  {
8      public function formPendaftaran()
9      {
10         return view('form-pendaftaran');
11     }
12
13    public function prosesForm(Request $request)
14    {
15        $validateData = $request->validate([
16            'nim'          => 'required|size:8',
17            'nama'         => 'required|min:3|max:50',
18            'email'        => 'required|email',
19            'jenis_kelamin' => 'required|in:P,L',
20            'jurusan'      => 'required',
21            'alamat'       => '',
22        ]);
23
24        dump($validateData);
25    }
26 }
```

Method `formPendaftaran()` dipakai untuk menampilkan view `form-pendaftaran`, sedangkan method `prosesForm()` dipakai untuk proses validasi. Syarat validasi ini juga sama seperti di bab sebelumnya.

Persiapan kita sudah selesai, silahkan akses halaman localhost:8000/form-pendaftaran/.



Gambar: Form pendaftaran dalam bahasa inggris

Terlihat judul form serta nama inputan sudah tampil dalam bahasa inggris. Lalu bagaimana untuk versi bahasa indonesia?

Silahkan buat folder baru dengan nama **id** di dalam folder **lang**, lalu buat juga file **form.php** di dalamnya dan isi dengan kode berikut:

lang/id/form.php

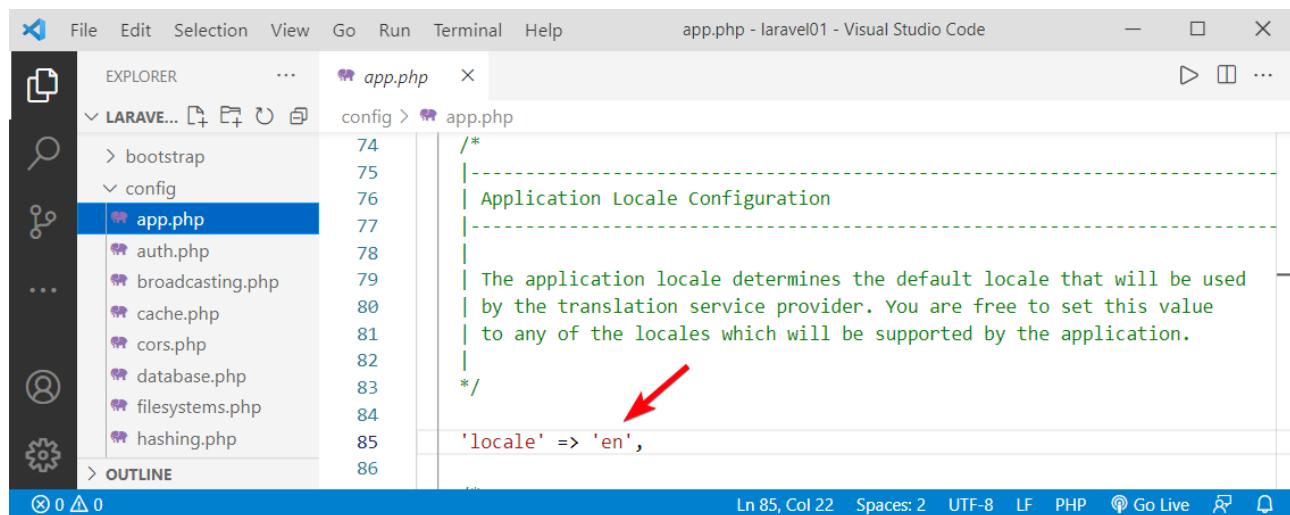
```

1  <?php
2
3  return [
4      'judul' => 'Pendaftaran Mahasiswa',
5      'input' => [
6          'nim' => 'NIM',
7          'nama_lengkap' => 'Nama Lengkap',
8          'email' => 'Email',
9          'jenis_kelamin' => 'Jenis Kelamin',
10         'pilihan_jenis_kelamin' => [
11             'laki_laki' => 'Laki-Laki',
12             'perempuan' => 'Perempuan',
13         ],
14         'jurusan' => 'Jurusan',
15         'alamat' => 'Alamat',
16         'tombol' => 'Daftar',
17     ],
18 ];

```

Struktur array yang digunakan sama persis seperti file **form.php** di folder **en**. Hanya saja sekarang semuanya dalam bahasa Indonesia.

Kemudian kita harus tukar pengaturan localization default Laravel. Caranya, buka file **config\app.php**, lalu cari baris: `'locale' => 'en'`.



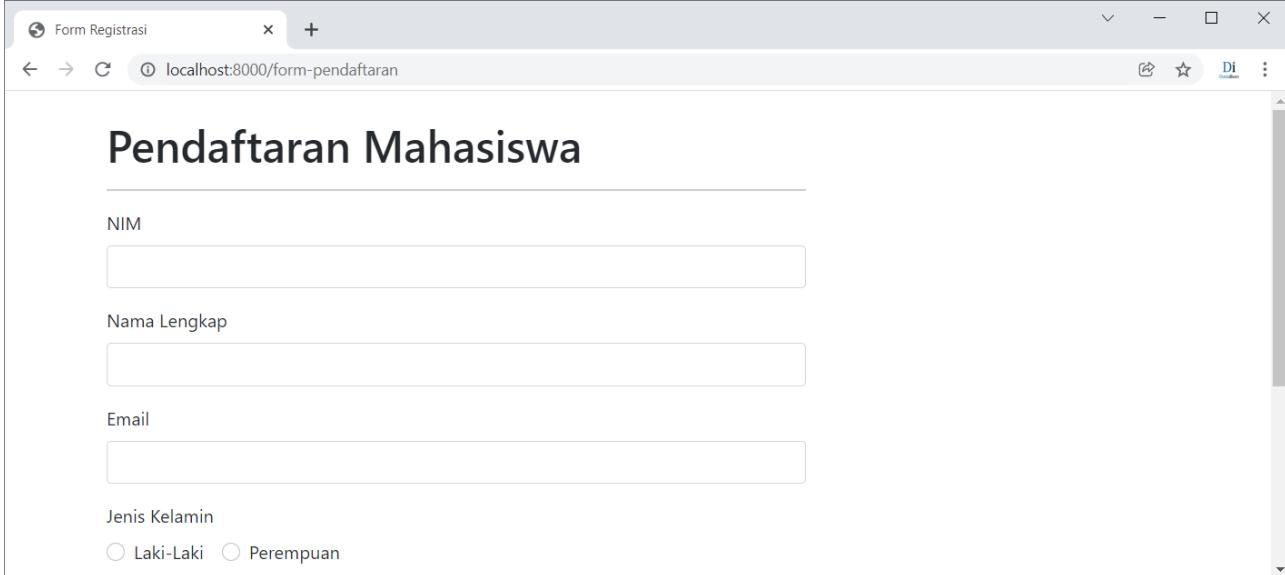
Gambar: Pengaturan locale di app.php

Inilah pengaturan global localization di dalam Laravel. Nilai 'en' merupakan kode bahasa yang dipakai saat ini, yaitu *english*. Agar kita bisa menggunakan localization dalam bahasa

indonesia, tukar nilainya dengan 'id':

```
'locale' => 'id',
```

Save file app.php, lalu buka refresh localhost:8000/form-pendaftaran:



Gambar: Form pendaftaran tampil dalam bahasa indonesia

Sekarang form pendaftaran sudah tampil dengan bahasa indonesia. Semua nilai nama form akan diambil dari "kamus" yang ada di file lang\id\form.php. Anda bisa coba kembalikan pengaturan 'locale' menjadi 'en' dan buka kembali halaman form.

Sampai di sini bisa kita ambil kesimpulan bahwa pengaturan locale akan mencari file localization berdasarkan nama folder. Jika diisi dengan 'locale' => 'id', maka Laravel akan mencari kamus ke folder lang\id. Jika misalnya diisi 'locale' => 'es' (untuk bahasa spanyol), maka Laravel akan mencari di lang\es.

Jika ternyata folder tersebut tidak ditemukan, secara bawaan akan kembali ke 'en' atau english. Ini diatur dengan settingan lain di file config\app.php, yakni baris:

```
'fallback_locale' => 'en'
```

18.5. Mengatur Localization secara Dinamis

Pengaturan localization dari file config\app.php bersifat global dan harus dilakukan secara manual. Namun Laravel juga membolehkan kita mengatur localization secara dinamis, bahkan menampilkan bahasa yang berbeda-beda untuk setiap view. Untuk keperluan ini, kita butuh mengakses static method App::setLocale() dari dalam controller.

Sebelum itu, saya akan buat 2 buah route baru untuk mengakses form pendaftaran:

routes/web.php

```

1 ...
2 Route::get('/form-pendaftaran{id}', [MahasiswaController::class,
3 'formPendaftaranId']);
4
5 Route::get('/form-pendaftaran/en', [MahasiswaController::class,
6 'formPendaftaranEn']);

```

Tujuannya, ketika diakses halaman /form-pendaftaran/id, form tampil dalam bahasa Indonesia, sedangkan ketika diakses halaman /form-pendaftaran/en, form akan tampil dalam bahasa Inggris. Kedua route ini juga akan memanggil method yang berbeda, yakni formPendaftaranId() dan formPendaftaranEn().

Berikut kode program untuk MahasiswaController:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App;
7
8 class MahasiswaController extends Controller
9 {
10 //...
11
12     public function formPendaftaranId()
13     {
14         App::setLocale('id');
15         return view('form-pendaftaran');
16     }
17
18     public function formPendaftaranEn()
19     {
20         App::setLocale('en');
21         return view('form-pendaftaran');
22     }
23
24 //...
25 //...
26 }

```

Tambahan kode pertama ada di baris 6, yakni proses import App class.

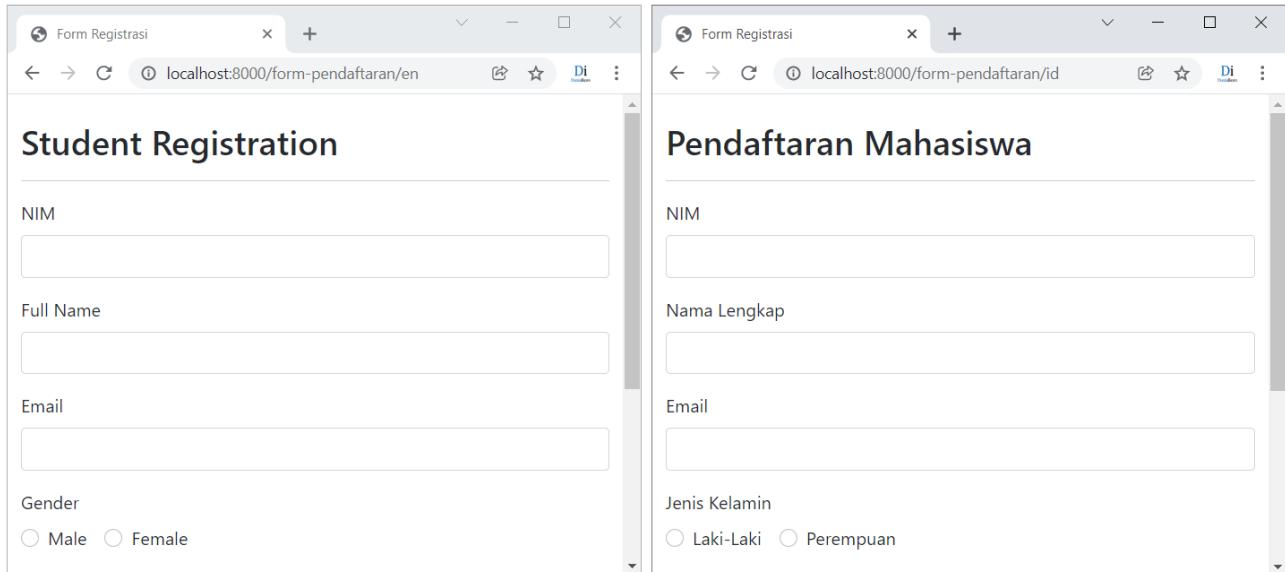
Kemudian sebelum proses pemanggilan view 'form-pendaftaran', di dalam method formPendaftaranId() dan formPendaftaranEn() terdapat perintah App::setLocale('id') dan App::setLocale('en'). Inilah cara untuk mengatur settingan locale secara dinamis.

Ketika perintah App::setLocale('id') dijalankan, maka semua kode program di dalam method tersebut akan memakai pengaturan 'locale' => 'id'. Inilah yang membuat form

tampil dalam bahasa indonesia.

Perintah `App::setLocale('id')` juga akan menimpa pengaturan `locale` di file `config/app.php`, namun hanya sebatas kode program di dalam method tersebut saja. Jika kita membuka halaman lain yang tidak memiliki perintah `App::setLocale()`, maka tetap kembali memakai pengaturan dari file `config/app.php`.

Silahkan coba buka halaman localhost:8000/form-pendaftaran/id dan localhost:8000/form-pendaftaran/en, form yang sama akan tampil dalam 2 jenis bahasa.



Gambar: Form pendaftaran dalam bahasa inggris (kiri) dan dalam bahasa indonesia (kanan)

18.6. Localization dengan Route Parameter

Kita sudah berhasil menampilkan form dalam 2 bahasa, namun sebenarnya masih kurang efisien karena terdapat duplikasi kode program di method `formPendaftaranId()` dan `formPendaftaranEn()`. Akan lebih baik jika kode untuk menampilkan view form pendaftaran cukup satu saja. Sebagai solusi, kita bisa memanfaatkan **route parameter**.

Pertama, saya akan modifikasi penulisan 2 route sebelumnya menjadi 1 route saja:

```
routes/web.php
Route::get('/form-pendaftaran/{locale?}', [MahasiswaController::class,
    'formPendaftaran']);
```

Route `form-pendaftaran` sekarang bisa menerima *optional route parameter* yang ditampung oleh variabel `locale`. Jika anda sedikit bingung dengan penulisan ini, bisa balik sejenak ke materi tentang route di awal buku.

Selanjutnya, di dalam controller bisa di buat dengan kode berikut:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App;
7
8 class MahasiswaController extends Controller
9 {
10     public function formPendaftaran($locale = 'id')
11     {
12         App::setLocale($locale);
13         return view('form-pendaftaran');
14     }
15
16     //...
17     //...
18 }
```

Nilai yang diinput dari route parameter akan ditangkap oleh argument \$locale yang juga diisi dengan nilai default 'id'. Tujuannya, agar ketika halaman form pendaftaran diakses tanpa argument, nilai 'id' inilah yang akan dipakai.

Argument \$locale kemudian menjadi nilai input ke dalam method App::setLocale(), seperti di baris 12 yang saya tulis sebagai App::setLocale(\$locale). Hasilnya, form pendaftaran menjadi lebih fleksibel dan bisa ditampilkan dalam berbagai bahasa.

Silahkan anda coba akses halaman localhost:8000/form-pendaftaran/id, dan halaman localhost:8000/form-pendaftaran/en. Tampilan form akan tetap seperti sebelumnya, yakni dalam bahasa indonesia dan bahasa inggris, tergantung alamat yang ditulis.

Selain itu form juga tetap bisa diakses dari halaman localhost:8000/form-pendaftaran yang akan menampilkan form dalam bahasa indonesia, tidak peduli pengaturan yang ada di dalam app\config.php. Lebih jauh lagi, halaman localhost:8000/form-pendaftaran/fr juga tersedia jika kita ingin menampilkan form dalam bahasa Prancis, namun tentunya harus buat dulu kamus bahasa prancis di lang\fr\form.php.

18.7. Localization Untuk Validasi Form

Salah satu penggunaan paling umum dari localization adalah untuk menampilkan pesan kesalahan form. Secara default, pesan kesalahan form tampil dalam bahasa inggris. File asalnya terdapat di lang\en\validation.php.

Berikut potongan dari file ini:

lang\en\validation.php

```
1 <?php
```

```

2
3     return [
4
5         /*
6         |--------------------------------------------------------------------------
7         | Validation Language Lines
8         |--------------------------------------------------------------------------
9
10        |
11        | The following language lines contain the default error messages used by
12        | the validator class. Some of these rules have multiple versions such
13        | as the size rules. Feel free to tweak each of these messages here.
14    */
15
16    'accepted' => 'The :attribute must be accepted.',
17    'accepted_if' => 'The :attribute must be accepted when :other is :value.',
18    'active_url' => 'The :attribute is not a valid URL.',
19    ...

```

File lang\en\validation.php cukup panjang, hingga ratusan baris. Ini semua sesuai dengan syarat validasi yang ada di Laravel. Sebagai contoh, jika syarat required tidak terpenuhi (inputan form tidak diisi), pesan kesalahan yang tampil berasal dari isi kamus berikut:

```
'required' => 'The :attribute field is required.',
```

Begitu juga dengan pesan kesalahan lain, semuanya ada di file ini. Jika berminat, anda bisa pelajari sekilas. Beberapa pesan error juga berbentuk nested array tergantung kompleksitas syarat validasi.

Sekarang, bagaimana membuat pesan validasi tampil dalam bahasa indonesia? Caranya, copy file validation.php ini ke dalam folder id, lalu translate satu persatu teks yang ada di dalam array.

Namun mengingat jumlah array yang sangat banyak, tentu ini bukanlah pekerjaan yang mudah. Untungnya, terdapat library localization Laravel yang siap pakai, salah satunya [laravel-lang.com](#). Di library ini, tersedia file translasi Laravel untuk puluhan bahasa, termasuk bahasa Indonesia.

Untuk menggunakannya, silahkan buka cmd, masuk ke folder laravel dan ketik perintah:

```
composer require laravel-lang/lang="^10.0"
```

```

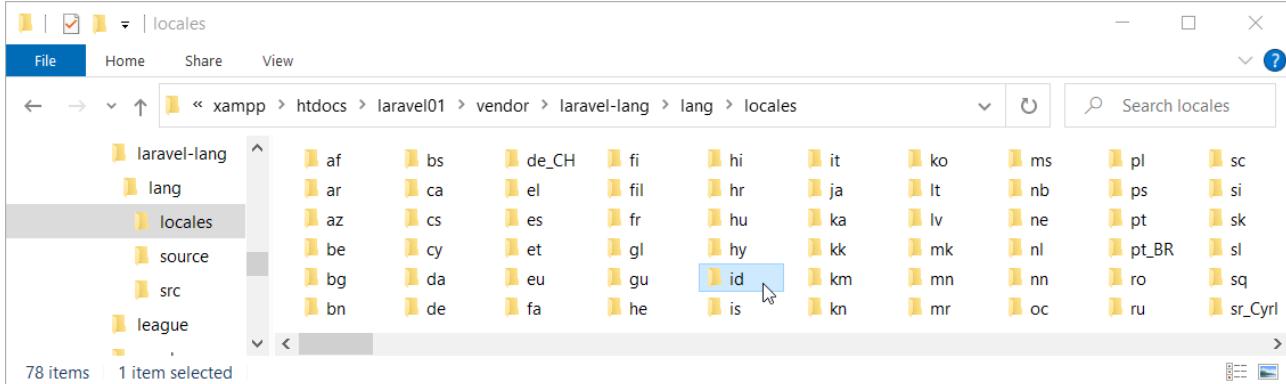
C:\xampp\htdocs\laravel01>composer require laravel-lang/lang="^10.0"
Info from https://repo.packagist.org: #StandWithUkraine
./composer.json has been updated
Running composer update laravel-lang/lang
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking laravel-lang/lang (10.9.5)

```

Gambar: Proses instalasi library laravel-lang

Localization

Setelah selesai, silahkan buka folder `vendor\laravel-lang\lang\locales`. Di dalamnya terdapat puluhan folder dengan kode bahasa:



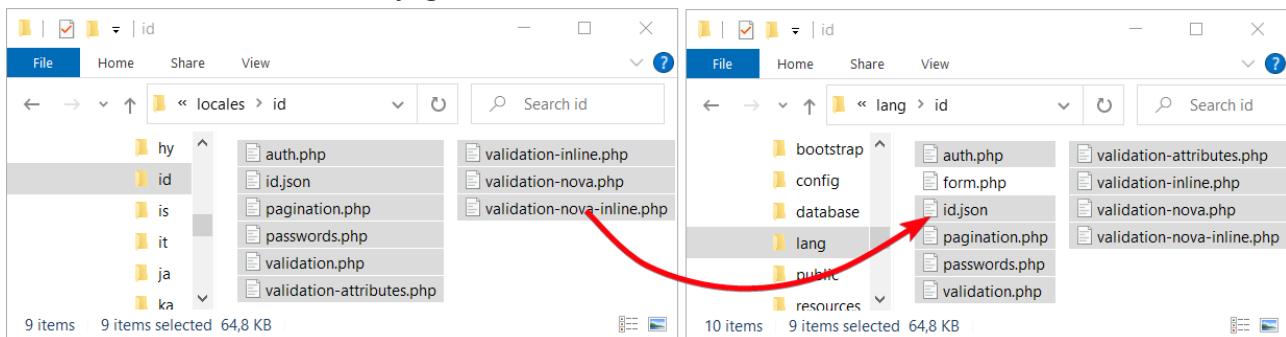
Gambar: Berbagai folder bahasa dari library laravel-lang

Setiap folder berisi beberapa file, termasuk 4 file localization utama bawaan Laravel:

- **auth.php**: Berisi pesan teks untuk proses authentication (login user).
- **pagination.php**: Berisi teks untuk tampilan pagination (kata "sebelum" dan "sesudah").
- **passwords.php**: Berisi pesan teks untuk proses reset password.
- **validation.php**: Berisi pesan teks untuk proses validasi form.

Selain 4 file di atas, juga terdapat beberapa file tambahan lain sesuai kebutuhan tim pengembang library Laravel Lang.

Silahkan buka folder **id**, copy saja semua file yang ada di dalamnya, lalu paste ke dalam folder `lang\id`. Jika berminat, anda juga bisa lihat sekilas isi dari file-file ini.



Gambar: Copy isi folder `vendor\laravel-lang\lang\src\id` dan paste ke `lang\id`

Persiapan kita sudah selesai. Sekarang silahkan cek pengaturan locale di file `config/app.php`, pastikan nilainya adalah **id**. Cek juga di controller apakah sudah ada method `prosesForm()` yang akan memproses validasi form pendaftaran. Terakhir tes jalankan form pendaftaran dan langsung saja klik tombol **Daftar**:

Localization

NIM
Nim wajib diisi.

Nama Lengkap
Nama wajib diisi.

Email
Email wajib diisi.

Jenis Kelamin
 Laki-Laki Perempuan
Jenis kelamin wajib diisi.

Gambar: Tampilan error validasi

Hasilnya, pesan error form tampil dalam bahasa Indonesia. Jika dirasa kurang pas, silahkan modifikasi sendiri file `lang\id\validation.php`.

Namun ada sedikit masalah terkait proses validasi dinamis. Sebelumnya, kita sudah set route agar menampilkan form sesuai alamat URL (menggunakan route parameter). Jika yang diakses `localhost:8000/form-pendaftaran/en`, maka form sudah tampil dalam bahasa Inggris.

Akan tetapi ketika form disubmit, pesan error muncul dalam bahasa Indonesia. Ini karena error validasi tetap merujuk ke settingan 'locale' di file `config/app.php`. Bagaimana cara agar pesan error validasi tampil sesuai dengan bahasa yang diakses?

Jika anda masih ingat, proses pengaturan validasi global bisa ditimpakan oleh perintah `App::setLocale()` dari dalam controller. Inilah yang kita pakai untuk menampilkan form pendaftaran dinamis:

`app/Http/Controllers/MahasiswaController.php`

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App;
7
8 class MahasiswaController extends Controller
9 {
```

```

10     public function formPendaftaran($locale = 'id')
11     {
12         App::setLocale($locale);
13         return view('form-pendaftaran');
14     }
15
16     public function prosesForm(Request $request)
17     {
18         $validateData = $request->validate([
19             'nim'          => 'required|size:8',
20             'nama'         => 'required|min:3|max:50',
21             'email'        => 'required|email',
22             'jenis_kelamin' => 'required|in:P,L',
23             'jurusan'      => 'required',
24             'alamat'       => '',
25         ]);
26
27         dump($validateData);
28     }
29 }
```

Di baris 12 terdapat perintah `App::setLocale($locale)`, sehingga sepanjang method `formPendaftaran()`, bahasa yang dipakai sesuai dengan isi dari parameter `$locale`.

Jika kita ingin pesan error validasi juga tampil sesuai bahasanya, maka perintah `App::setLocale($locale)` juga perlu ditulis ke dalam method `prosesForm()`, karena di sinilah validasi form dilakukan. Masalahnya, bagaimana cara mengirim data `$locale` ke method ini?

Sebagai pengingat, data `$locale` berasal dari route parameter. Ketika user membuka halaman `localhost:8000/form-pendaftaran/en`, route akan menangkap string "en" untuk dikirim ke method `formPendaftaran()`. Di dalam method `formPendaftaran()`, route parameter ini disimpan ke dalam variabel `$locale`.

Agar bisa sampai ke method `prosesForm()`, data `$locale` bisa kita kirim ke dalam view `form-pendaftaran` terlebih dahulu. Silahkan modifikasi isi `formPendaftaran()` di `MahasiswaController` menjadi berikut:

app/Http/Controllers/MahasiswaController.php

```

1 ...
2     public function formPendaftaran($locale = 'id')
3     {
4         App::setLocale($locale);
5         return view('form-pendaftaran', ["locale" => $locale]);
6     }
7 ...
```

Terdapat tambahan array `["locale" => $locale]` di akhir baris 5. Sesampainya di view `form-pendaftaran`, tambah 1 inputan form dengan tipe hidden di dalam form:

```
resources/views/form-pendaftaran.blade.php
```

```
1 ...  
2 <form action="{{url('/proses-form')}}" method="POST">  
3 @csrf  
4 <input type="hidden" name="locale" value="{{ $locale }}" >  
5 <div class="mb-3">  
6 <label class="form-label" for="nim">  
7 ...
```

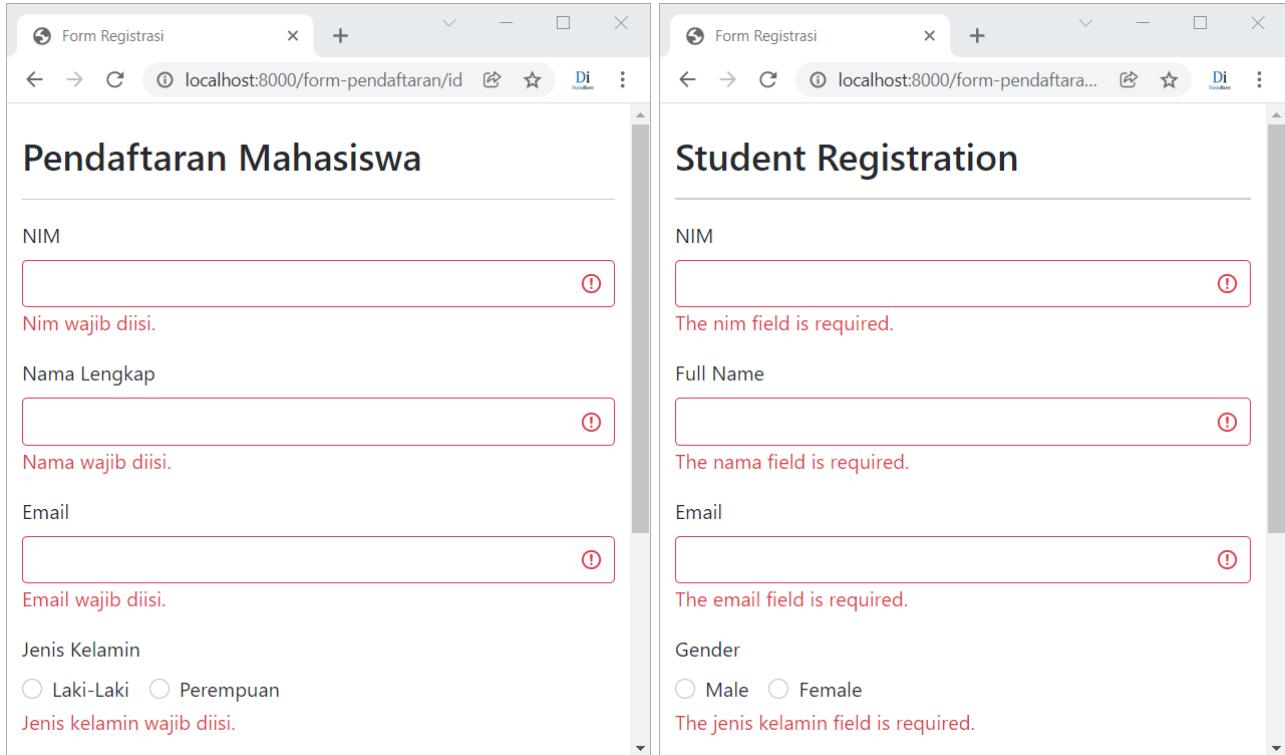
Tag <input> di baris 4 akan mengirim data 'locale' pada saat form di submit. Dengan demikian, di dalam method prosesForm(), kita bisa mengaksesnya dari \$request->locale. Berikut modifikasinya:

```
app/Http/Controllers/MahasiswaController.php
```

```
1 ...  
2 public function prosesForm(Request $request)  
3 {  
4     App::setLocale($request->locale);  
5     $validateData = $request->validate([  
6         'nim'          => 'required|size:8',  
7         'nama'         => 'required|min:3|max:50',  
8         'email'        => 'required|email',  
9         'jenis_kelamin' => 'required|in:P,L',  
10        'jurusan'      => 'required',  
11        'alamat'       => '',  
12    ]);  
13  
14    dump($validateData);  
15 }  
16 ...
```

Di baris 4, perintah App::setLocale(\$request->locale) akan men-set bahasa locale sepanjang method prosesForm(), termasuk tampilan bahasa error validasi. Sekarang pesan error akan tampil sesuai jenis bahasa form:

Localization



Gambar: Tampilan error validasi dalam bahasa indonesia (kiri), dan dalam bahasa inggris (kanan)

18.8. Translation Strings

Di Laravel 9, folder `lang` memiliki 1 file JSON bawaan, yakni `en.json`. File ini dipakai untuk membuat proses translasi menggunakan fitur **translation strings** sebagai alternatif cara mentranslate kata/kalimat.

Penggunaan *translation strings* sebenarnya mirip seperti kamus localization yang sudah kita praktekkan, hanya saja sekarang key kamus bisa berbentuk kalimat utuh. Sebagai contoh, jika di dalam file blade terdapat perintah berikut:

```
{{ __('form.nama_kampus') }}
```

Maka Laravel akan mencari padanan translasi ke file `form.php`, lalu ke key kamus '`nama_kampus`'. Jika ternyata key tersebut tidak ditemukan, maka teks '`form.nama_kampus`' akan tampil di web browser. Laravel menyebut cara ini sebagai **short keys**.

Dengan *translation strings*, kita bisa membuat perintah berikut:

```
{{ __('Universitas Ilkoom') }}
```

Sekarang Laravel akan coba mencari padanan translasi ke file `[locale].json` seperti `en.json`, atau `id.json` sesuai dengan bahasa locale yang dipilih. Jika ternyata tidak ditemukan, teks '`Universitas Ilkoom`' akan tampil di web browser. Ini membuat cara translasi dengan *translation strings* terasa lebih natural.

Sebagai bahan praktek, silahkan modif bagian atas view form-pendaftaran sebagai berikut:

resources/views/form-pendaftaran.blade.php

```

1 ...
2 <div class="container pt-4 bg-white">
3   <div class="row">
4     <div class="col-md-8 col-xl-6">
5       <h1>{{ __('form.judul') }}</h1>
6       <p class="lead">{{ __('form.nama_kampus') }}</p>
7       <p class="lead">{{ __('Universitas Ilkoom') }}</p>
8       <hr>

```

Di baris 6 saya menggunakan key kamus atau *short keys*, sedangkan di baris 7 memakai teknik *translation strings*. Tambahan `class="lead"` ke dalam tag `<p>` hanya sekedar class CSS bawaan Bootstrap untuk mempercantik tampilan teks. Berikut hasilnya di web browser:



Gambar: Hasil translasi dengan short keys (1) dan translation strings (2)

Hasil translasi dengan short keys (1) menampilkan teks `form.nama_kampus` karena key tersebut belum ada di dalam kamus. Agar bisa menampilkan teks yang sesuai, harus ditambah key '`'nama_kampus'`' ke file `lang\id\form.php` serta file `lang\en\form.php`:

lang/en/form.php

```

1 <?php
2
3 return [
4   'judul' => 'Student Registration',
5   'input' => [
6     ...
7   ],
8   'nama_kampus' => 'Ilkoom University',
9 ];

```

lang/id/form.php

```

1 <?php
2
3 return [

```

Localization

```
4     'judul' => 'Pendaftaran Mahasiswa',
5     'input' => [
6         ...
7     ],
8     'nama_kampus' => 'Universitas Ilkoom',
9 ];
```

Bagaimana dengan versi *translation strings*? Cukup dengan mengisi kamus translasi ke setiap file JSON.

Silahkan buka file `en.json`, secara bawaan di dalamnya sudah ada beberapa teks translasi. Teks ini boleh saja dihapus atau dibiarkan. Kemudian tambah 1 baris di bagian paling bawah:

`lang/en.json`

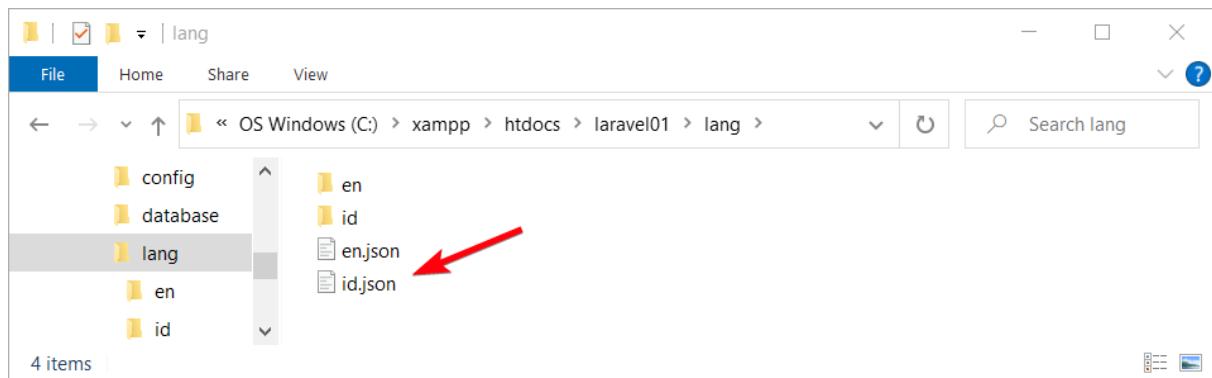
```
1 {
2     "The :attribute must contain at least one letter.": "The :attribute ...",
3     "The :attribute must contain at least one number.": "The :attribute ...",
4     "The :attribute must contain at least one symbol.": "The :attribute ...",
5     "The :attribute must contain at least ... letter.": "The :attribute ...",
6     "The given :attribute has appeared in ... :attribute.": "The.... ",
7     "Universitas Ilkoom": "Ilkoom University"
8 }
```

Meskipun mirip seperti array PHP, tapi ini adalah file JSON (JavaScript Object Notation), sehingga kita harus menggunakan format JSON dalam menambah kamus.

Untuk versi bahasa indonesia, silahkan buat file baru dengan nama `id.json` ke dalam folder `lang`, lalu isi dengan kode berikut:

`lang/id.json`

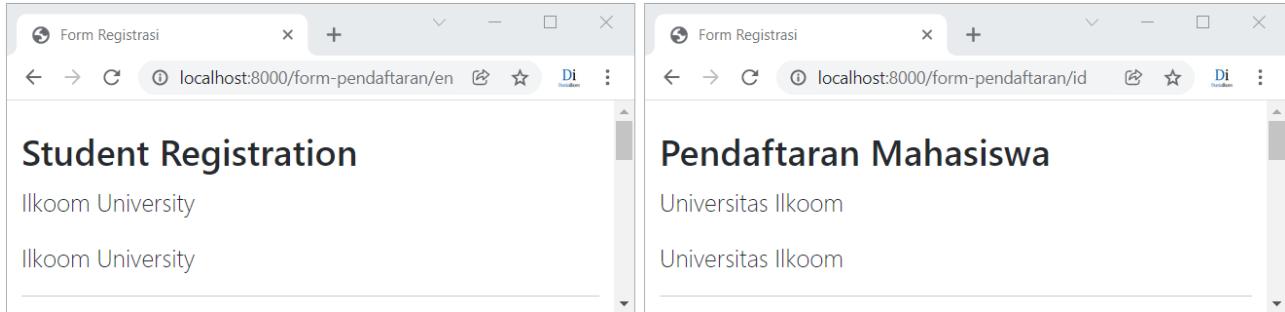
```
1 {
2     "Universitas Ilkoom": "Universitas Ilkoom"
3 }
```



Gambar: File `en.json` dan `id.json` di dalam folder `lang`

Sekarang proses translasi dengan *translation strings* sudah bisa berjalan. Silahkan tes akses halaman `localhost:8000/form-pendaftaran/en` dan `localhost:8000/form-pendaftaran/id`:

Localization



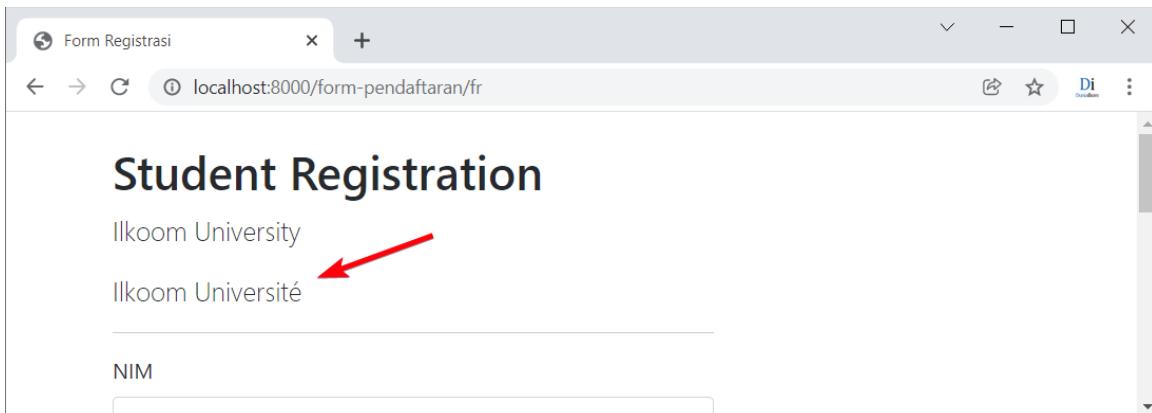
Gambar: Proses translasi sudah berjalan untuk bahasa inggris (en) dan bahasa indonesia (id)

Sedikit tambahan latihan, silahkan buat juga file `fr.json` ke dalam folder `lang`, dan isi dengan kode berikut:

`lang/fr.json`

```
1  {
2      "Universitas Ilkoom": "Ilkoom Université"
3 }
```

Akses alamat `localhost:8000/form-pendaftaran/fr`, dan teks versi translation strings akan tampil dalam bahasa Prancis:



Gambar: Hasil tampilan alamat `localhost:8000/form-pendaftaran/fr`

Fitur `translation strings` ini menawarkan cara translasi yang lebih mudah dan cukup di 1 file saja.

Localization menjadi fitur pelengkap yang sangat berguna. Meskipun tidak semua project butuh website multi bahasa, setidaknya menampilkan pesan error form dalam bahasa indonesia akan sering kita pakai.

Berikutnya, kita akan masuk ke pembahasan tentang **CRUD**.

19. CRUD

CRUD merupakan singkatan dari **Create**, **Read**, **Update** dan **Delete**, yang menjadi fondasi dasar dari sebuah aplikasi web. CRUD juga bisa disebut sebagai materi yang mempelajari cara pengelolaan data di database. Di sini kita akan membuat kode program untuk proses input menggunakan form, menampilkan data tabel, update data, dan penghapusan data.

Apa yang akan kita pelajari juga merangkum berbagai materi yang sudah di bahas sejak awal buku, terutama tentang konsep **MVC**, **Eloquent ORM** dan **Form Processing**.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css**, karena kita akan memakai sedikit kode Bootstrap. Hapus juga tabel di database **laravel** (jika ada).

19.1. Persiapan Awal

Sebagai materi praktek sepanjang bab ini, saya kembali menggunakan tabel **mahasiswa**. Untuk membuat tabel tentunya butuh file migration, dan nantinya kita juga perlu file model dan controller. Daripada membuat file ini satu per satu, terdapat perintah **php artisan** untuk men-generate ketiga file ini sekaligus. Caranya, jalankan perintah berikut:

```
php artisan make:model Mahasiswa -mc
```

```
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa -mc
Model created successfully.
Created Migration: 2022_01_23_074817_create_mahasiswa_table
Controller created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Pembuatan model Mahasiswa dan file migration

Pada dasarnya ini merupakan perintah untuk membuat model `Mahasiswa` dengan tambahan option `-mc` di bagian akhir. Option `-mc` merupakan singkatan dari *migration* dan *controller*. Alternatif perintah lain adalah:

```
php artisan make:model Mahasiswa --migration --controller
```

Perhatikan bahwa Laravel langsung memberikan nama tabel migration dalam bentuk plural, yakni `mahasiswas`.

Silahkan buka file migration `mahasiswas` dan rancang struktur tabel berikut:

`database/migrations/<timestamp>_create_mahasiswas_table.php`

```
1 public function up()
2 {
3     Schema::create('mahasiswas', function (Blueprint $table) {
4         $table->id();
5         $table->char('nim',8)->unique();
6         $table->string('nama');
7         $table->char('jenis_kelamin',1);
8         $table->string('jurusan');
9         $table->text('alamat')->nullable();
10        $table->timestamps();
11    });
12 }
```

Struktur tabel `mahasiswas` nyaris sama seperti praktik kita sebelumnya, hanya saja saya mengurangi kolom email agar inputan form tidak terlalu banyak. Simpan file di atas dan jalankan migration dengan perintah `php artisan migrate`.

Selanjutnya adalah untuk **route**. Laravel sudah memiliki aturan tersendiri (atau lebih tepatnya "saran") terkait pembuatan route untuk CRUD, yakni sebagai berikut:

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Gambar: Tabel penulisan route untuk CRUD (sumber: [Resource Controllers](#))

Ini adalah daftar route yang mengimplementasikan konsep **REST** (REpresentational State Transfer). Meskipun bukan keharusan, namun dengan menggunakan REST, route yang kita buat akan lebih rapi dan mengikuti standar umum di dunia web programming. Struktur penulisan seperti ini juga akan memudahkan programmer lain dalam memahami sistem CRUD.

Istilah lain yang juga sering anda temui adalah **RESTful** dan **Restful API**.

RESTful dipakai untuk menyebut implementasi konsep REST ke dalam web service. Sedangkan **RESTful API** adalah penerapan konsep REST ke dalam API (Application Programming Interface), yakni metode komunikasi antar web yang biasanya dilakukan dengan format JSON.

Laravel sendiri menyediakan fitur pembuatan RESTful API yang cukup lengkap dan sering dipakai digabung dengan teknologi frontend JavaScript seperti **Vue** atau **React**. Selain itu, RESTful API juga bisa untuk data input ke aplikasi android. Dalam buku ini, materi tentang RESTful API belum akan kita bahas.

Berikut penjelasan dari setiap kolom tabel di atas:

- Kolom **Verb** berisi jenis HTTP method yang akan digunakan, di antaranya method get dan juga post, selain itu terdapat juga method put, patch dan delete.
- Kolom **URI** adalah alamat route yang dipakai untuk mengakses proses ini. Perhatikan bahwa alamatnya menggunakan kata jamak (plural), yakni /photos, bukan /photo. Karena CRUD yang akan kita rancang dipakai untuk mengelola data mahasiswa, maka alamat URInya adalah /mahasiswas, bukan /mahasiswa.
- Kolom **Action** adalah jenis tindakan yang dilakukan, apakah itu membuat data (*create*), menampilkan data (*show*), dsb. Nantinya, ini akan menjadi nama method di controller.
- Kolom **Route Name** berisi cara penulisan *named routes*. Terkait dengan materi named route ini sudah pernah kita bahas di akhir bab tentang blade.

Penjelasan lebih detail dari isi tabel ini akan saya bahas secara bertahap dengan contoh praktek. Untuk saat ini, silahkan buat 3 route berikut:

routes/web.php

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/mahasiswas', [MahasiswaController::class, 'index'])
7 ->name('mahasiswas.index');
8
9 Route::get('/mahasiswas/create', [MahasiswaController::class, 'create'])
10->name('mahasiswas.create');
```

```

11
12 Route::post('/mahasiswa', [MahasiswaController::class, 'store'])
13 ->name('mahasiswa.store');

```

Ketiga route ini mengikuti aturan yang ada di tabel REST sebelumnya.

Route pertama, yakni '/mahasiswa' akan menampilkan semua isi tabel `mahasiswa`. Kode program untuk route ini nantinya di proses oleh method `index()` di dalam `MahasiswaController`. Route ini memiliki *named routes* berupa '`mahasiswa.index`'.

Route kedua, '/mahasiswa/create' dipakai untuk menampilkan form input untuk tabel `mahasiswa`. Kode programnya akan di proses oleh method `create()` di dalam `MahasiswaController`. Route ini memiliki *named routes* berupa '`mahasiswa.create`'.

Route ketiga, '/mahasiswa/' ditujukan untuk memproses hasil form, termasuk menginput hasil form ke dalam tabel `mahasiswa`. Kode program untuk route ini akan di proses oleh method `store()` di dalam `MahasiswaController`. Route ini memiliki *named routes* berupa '`mahasiswa.store`'.

Perhatikan bahwa alamat URL untuk route pertama dan ketiga sama-sama berisi '/mahasiswa', bedanya ada di method yang digunakan. Route pertama memakai method `get`, sedangkan route ketiga menggunakan method `post`.

Lanjut, buka file `MahasiswaController` dan isi dengan kode program berikut:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     public function index()
10    {
11        return "Tabel mahasiswa di sini";
12    }
13
14    public function create()
15    {
16        return view('form-pendaftaran');
17    }
18
19    public function store(Request $request)
20    {
21        $validateData = $request->validate([
22            'nim'          => 'required|size:8',
23            'nama'         => 'required|min:3|max:50',
24            'jenis_kelamin' => 'required|in:P,L',
25            'jurusan'      => 'required',

```

CRUD

```
26           'alamat'      => '',
27           ]);          dump($validateData);
28       }
29   }
30 }
```

Semua kode ini sudah kita pelajari sebelumnya. Untuk method `index()`, sementara ini hanya akan menampilkan teks 'Tabel mahasiswa disini...', yang nantinya akan kita ganti dengan kode program untuk menampilkan semua data mahasiswa.

Isi method `create()` juga hanya 1 baris, yakni menampilkan view form-pendaftaran. Untuk method `store()`, berisi syarat validasi inputan form. Jika semua validasi sesuai, hasil form akan ditampilkan oleh perintah `dump($validateData)`.

Terakhir, buat file view form-pendaftaran.blade.php dengan kode program berikut:

resources/views/form-pendaftaran.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Pendaftaran Mahasiswa</title>
9 </head>
10 <body>
11
12 <div class="container pt-4 bg-white">
13     <div class="row">
14         <div class="col-md-8 col-xl-6">
15             <h1>Pendaftaran Mahasiswa</h1>
16             <hr>
17
18             <form action="{{ route('mahasiswas.store') }}" method="POST">
19                 @csrf
20                 <div class="mb-3">
21                     <label class="form-label" for="nim">NIM</label>
22                     <input type="text" id="nim" name="nim" value="{{ old('nim') }}"
23                         class="form-control" @error('nim') is-invalid @enderror">
24                     @error('nim')
25                         <div class="text-danger">{{ $message }}</div>
26                     @enderror
27                 </div>
28
29                 <div class="mb-3">
30                     <label class="form-label" for="nama">Nama Lengkap</label>
31                     <input type="text" id="nama" name="nama" value="{{ old('nama') }}"
32                         class="form-control" @error('nama') is-invalid @enderror">
33                     @error('nama')
34                         <div class="text-danger">{{ $message }}</div>
35                     @enderror
36                 </div>
37             </form>
38
39             <div class="d-grid gap-2 d-md-block">
40                 <button type="button" class="btn btn-primary" @click="cancel"><i>Batal</i></button>
41                 <button type="button" class="btn btn-primary" @click="store">Simpan</button>
42             </div>
43         </div>
44     </div>
45 </div>
```

```
36 </div>
37
38 <div class="mb-3">
39   <label class="form-label">Jenis Kelamin</label>
40   <div class="d-flex">
41     <div class="form-check me-3">
42       <input class="form-check-input" type="radio" name="jenis_kelamin"
43         id="laki_laki" value="L"
44         {{ old('jenis_kelamin')=='L' ? 'checked': '' }}>
45       <label class="form-check-label" for="laki_laki">Laki-laki</label>
46     </div>
47     <div class="form-check">
48       <input class="form-check-input" type="radio" name="jenis_kelamin"
49         id="perempuan" value="P"
50         {{ old('jenis_kelamin')=='P' ? 'checked': '' }}>
51       <label class="form-check-label" for="perempuan">Perempuan</label>
52     </div>
53   </div>
54   @error('jenis_kelamin')
55     <div class="text-danger">{{ $message }}</div>
56   @enderror
57 </div>
58
59 <div class="mb-3">
60   <label class="form-label" for="jurusan">Jurusan</label>
61   <select class="form-select" name="jurusan" id="jurusan"
62     value="{{ old('jurusan') }}">
63     <option value="Teknik Informatika"
64       {{ old('jurusan')=='Teknik Informatika' ? 'selected': '' }} >
65       Teknik Informatika
66     </option>
67     <option value="Sistem Informasi"
68       {{ old('jurusan')=='Sistem Informasi' ? 'selected': '' }} >
69       Sistem Informasi
70     </option>
71     <option value="Ilmu Komputer"
72       {{ old('jurusan')=='Ilmu Komputer' ? 'selected': '' }} >
73       Ilmu Komputer
74     </option>
75     <option value="Teknik Komputer"
76       {{ old('jurusan')=='Teknik Komputer' ? 'selected': '' }} >
77       Teknik Komputer
78     </option>
79     <option value="Teknik Telekomunikasi"
80       {{ old('jurusan')=='Teknik Telekomunikasi' ? 'selected': '' }} >
81       Teknik Telekomunikasi
82     </option>
83   </select>
84   @error('jurusan')
85     <div class="text-danger">{{ $message }}</div>
86   @enderror
87 </div>
88
89 <div class="mb-3">
90   <label class="form-label" for="alamat">Alamat</label>
```

```
91      <textarea class="form-control" id="alamat" rows="3"
92          name="alamat">{{ old('alamat') }}</textarea>
93      </div>
94
95      <button type="submit" class="btn btn-primary mb-2">Daftar</button>
96      </form>
97
98  </div>
99  </div>
100 </div>
101
102 </body>
103 </html>
```

View form-pendaftaran ini nyaris identik seperti yang kita gunakan pada bab Form Processing, sehingga tidak perlu saya bahas lagi. Termasuk diantaranya kode program untuk menampilkan pesan kesalahan serta proses repopulate form.

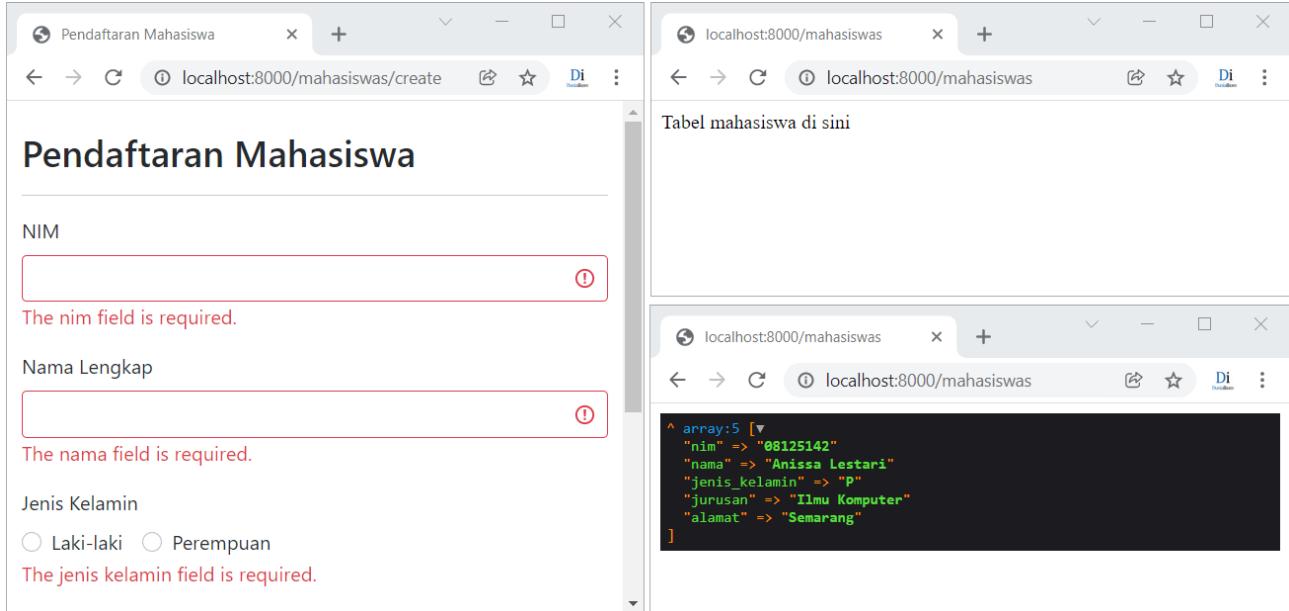
Sedikit modifikasi ada di baris 18, yakni alamat pengiriman inputan form:

```
<form action="{{ route('mahasiswa.store') }}" method="POST">
```

Artinya ketika form di submit, nilainya akan dikirim ke *named routes* `mahasiswa.store`, yang tidak lain merujuk ke alamat `Route::post('/mahasiswa')`. Selain itu inputan email juga saya hapus karena tidak kita gunakan dalam database nanti.

Persiapan kita sudah selesai, silahkan cek dengan melakukan hal berikut:

1. Buka localhost:8000/mahasiswa, cek apakah tampil teks "Tabel mahasiswa di sini".
2. Buka localhost:8000/mahasiswa/create, pastikan form pendaftaran sudah tampil.
3. Isi form pendaftaran dengan data sembarang, cek apakah tampil pesan error validasi.
4. Isi form pendaftaran dengan data yang benar, seharusnya akan tampil hasil `dump()` dari inputan form.



Gambar: Hasil percobaan pengisian form

Sip, semuanya sudah berjalan, saatnya kita masuk ke materi tentang CRUD.

19.2. Create

Konsep CRUD pertama yang akan di bahas adalah **Create**, yakni proses input data baru ke dalam tabel. Terdapat 3 cara yang bisa digunakan, yakni raw query, query builder dan eloquent ORM. Untuk bab ini kita akan memakai **Eloquent ORM**.

Jika dilihat dari tabel REST, proses create ditangani oleh 2 route, yakni **get /mahasiswa/** untuk menampilkan form pendaftaran, serta **post /mahasiswa** untuk memproses hasil form. Kedua route ini juga sudah kita buat, termasuk proses validasi inputan form.

Yang perlu di tambah adalah kode program Eloquent untuk menginput data dari form ke dalam tabel `mahasiswa`. Berikut kode yang bisa digunakan:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Mahasiswa;
7
8 class MahasiswaController extends Controller
9 {
10     public function index()
11     {
12         return "Tabel mahasiswa di sini";
13     }

```

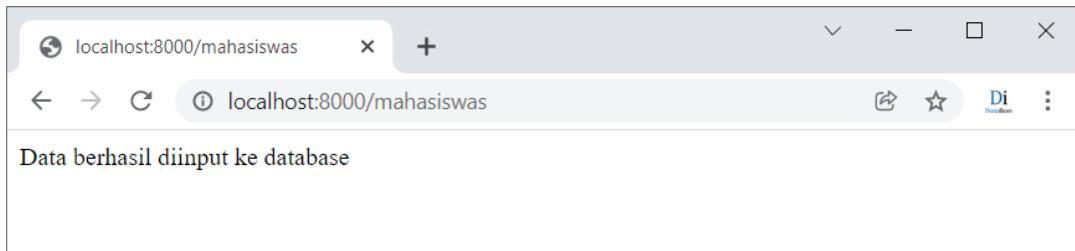
```

14
15     public function create()
16     {
17         return view('form-pendaftaran');
18     }
19
20     public function store(Request $request)
21     {
22         $validateData = $request->validate([
23             'nim'          => 'required|size:8',
24             'nama'         => 'required|min:3|max:50',
25             'jenis_kelamin' => 'required|in:P,L',
26             'jurusan'      => 'required',
27             'alamat'       => '',
28         ]);
29
30         $mahasiswa = new Mahasiswa();
31         $mahasiswa->nim = $validateData['nim'];
32         $mahasiswa->nama = $validateData['nama'];
33         $mahasiswa->jenis_kelamin = $validateData['jenis_kelamin'];
34         $mahasiswa->jurusan = $validateData['jurusan'];
35         $mahasiswa->alamat = $validateData['alamat'];
36         $mahasiswa->save();
37
38         return "Data berhasil diinput ke database";
39     }
40 }
```

Karena kita akan memakai model `Mahasiswa`, maka perlu proses import `use App\Models\Mahasiswa` di baris 6.

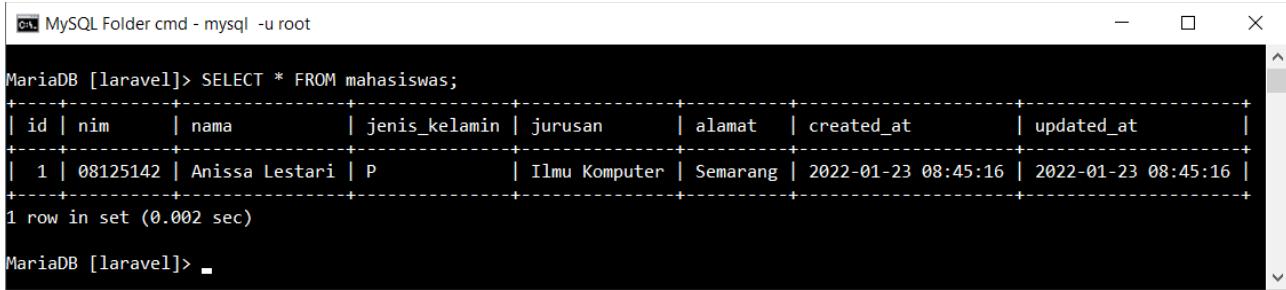
Tambahan kode program ada di baris 30 – 38, dimana saya membuat object `$mahasiswa` yang di instansiasi dari class `Mahasiswa`. Setiap kolom kemudian diinput sebagai property dari `$mahasiswa` seperti yang pernah kita bahas pada bab tentang Eloquent ORM.

Silahkan input data ke form pendaftaran, lalu cek ke database.



Gambar: Tampilan pesan teks setelah form pendaftaran di isi

CRUD



The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". It displays the result of a SQL query: "SELECT * FROM mahasiswa;". The output is a table with columns: id, nim, nama, jenis_kelamin, jurusan, alamat, created_at, and updated_at. There is one row of data: id=1, nim=08125142, nama=Anissa Lestari, jenis_kelamin=P, jurusan=Ilmu Komputer, alamat=Semarang, created_at=2022-01-23 08:45:16, and updated_at=2022-01-23 08:45:16. A message at the bottom says "1 row in set (0.002 sec)".

Gambar: Data berhasil masuk ke database

Jika semuanya sesuai, data baru mahasiswa sudah sampai ke tabel `mahasiswa`.

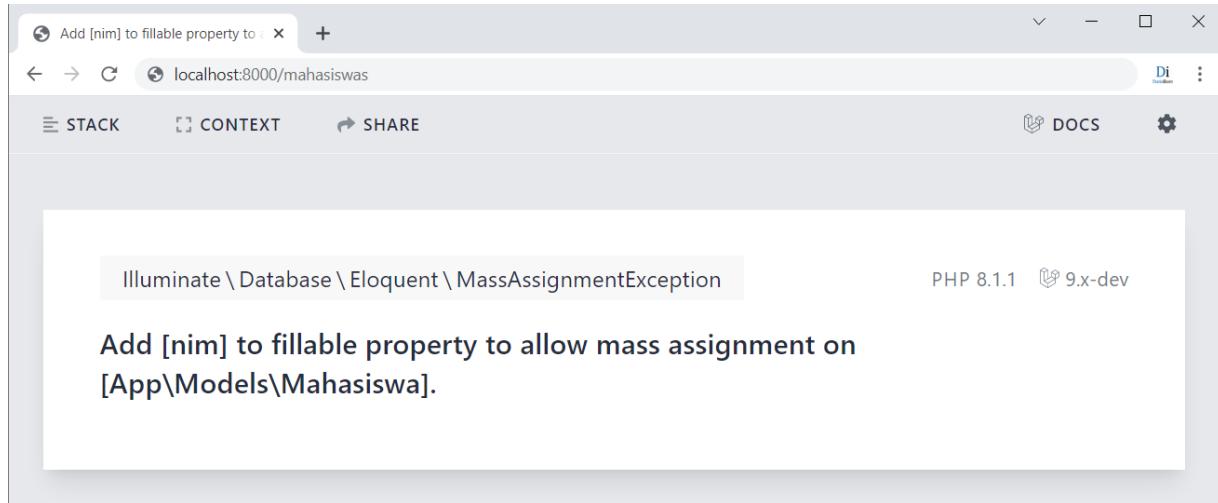
Cara yang kita pakai sudah pas, tapi masih bisa lebih singkat lagi jika menggunakan teknik *mass assignment*:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 //...
3     public function store(Request $request)
4     {
5         $validateData = $request->validate([
6             'nim'          => 'required|size:8',
7             'nama'         => 'required|min:3|max:50',
8             'jenis_kelamin' => 'required|in:P,L',
9             'jurusan'      => 'required',
10            'alamat'       => '',
11        ]);
12
13        Mahasiswa::create($validateData);
14
15        return "Data berhasil diinput ke database";
16    }
```

Sekarang proses input ke database dilakukan oleh 1 perintah saja, yakni `Mahasiswa::create ($validateData)` seperti di baris 13. Jika anda masih ingat, *mass assignment* butuh sebuah argument berbentuk associative array, dan itu sudah tersedia di variabel `$validateData`. Sangat praktis.

Mari kita coba, silahkan isi kembali form pendaftaran dan submit:



Gambar: Error proses input data

Bisakah anda menebak apa yang salah? Error ini sudah pernah kita hadapi sebelumnya.

Yup, ini berhubungan dengan pembatasan *mass assignment* di model `Mahasiswa`, dimana kita harus menulis property `$fillable` atau `$guarded` ke dalam class Model.

Silahkan buka file model `Mahasiswa.php`, lalu tambah property `protected $guarded = []`.

`App/Models/Mahasiswa.php`

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Mahasiswa extends Model
9 {
10     use HasFactory;
11     protected $guarded = [];
12 }
```

Dengan perintah `$guarded = []` artinya kita mengizinkan semua kolom tabel untuk diisi dari *mass assignment*.

Silahkan input ulang form pendaftaran dan klik tombol submit. Sekarang sudah ada 2 data di dalam tabel `mahasiswa`.

```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | nim  | nama           | jenis_kelamin | jurusan        | alamat          | created_at     | updated_at    |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1  | 08125142 | Anissa Lestari   | P              | Ilmu Komputer  | Semarang        | 2022-01-23 08:45:16 | 2022-01-23 08:45:16 |
| 2  | 19003036 | Sari Citra Lestari | P              | Teknik Informatika | Jakarta        | 2022-01-23 10:33:33 | 2022-01-23 10:33:33 |
+----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi tabel mahasiswa

Kenapa harus ada \$fillable dan \$guarded?

Alasannya untuk menghindari celah keamanan jika ada programmer yang menjalankan kode *mass assignment* sebagai berikut:

```
1 public function store(Request $request)
2 {
3     Mahasiswa::create($request->all());
4 }
```

Di sini, hasil dari method `$request->all()` yang berisi semua nilai form langsung diinput ke dalam argument method `Mahasiswa::create()`. Ini bisa jalan namun memiliki celah keamanan. Bisa saja seseorang memodifikasi form dan menambahkan inputan baru, misalnya `role = 'admin'`.

Inputan tersebut otomatis langsung masuk ke database jika tidak ada proteksi. Dengan menulis nama kolom secara manual di dalam property `$fillable`, maka kita bisa membatasi apa saja kolom yang boleh diisi. Laravel akan menolak kolom tambahan yang tidak terdaftar.

Tapi bagaimana jika menggunakan `$guarded = []`? bukankah itu sama saja dengan menghapus pembatasan ini?

Betul, apabila property `$guarded = []` dipakai pada kode program di atas, maka celah keamanan tadi tetap ada. Namun akan berbeda jika kode controller ditulis sebagai berikut:

```
1 public function store(Request $request)
2 {
3     $validateData = $request->validate([
4         'nim'          => 'required|size:8',
5         'nama'         => 'required|min:3|max:50',
6         'jenis_kelamin' => 'required|in:P,L',
7         'jurusan'      => 'required',
8         'alamat'       => '',
9     ]);
10
11     Mahasiswa::create($validateData);
```

```
12 }
```

Kali ini variabel yang diinput ke dalam method `Mahasiswa::create()` adalah `$validateData`, yang berisi array hasil proses validasi. Array yang tersimpan di `$validateData` hanya berisi key yang sudah di validasi. Jadi meskipun di dalam form ada yang menambah inputan lain, itu tidak akan sampai ke dalam variabel `$validateData`.

Jika kode untuk mass assignment ditulis seperti ini, tidak masalah kita menggunakan `$guarded = []`, karena proses filter sudah dilakukan di dalam controller.

Namun tentu saja tidak ada salahnya jika anda tetap ingin menggunakan property `$fillable` dan menulis daftar kolom yang boleh diisi. Proteksinya sekarang ada di dua tempat, yakni dari controller dan juga di model.

19.3. Read

Materi CRUD berikutnya adalah **Read**, yakni proses pembacaan data dari database.

Dari tabel REST, terdapat 2 route yang berhubungan dengan proses read, yakni `get /mahasiswas` yang dipakai untuk menampilkan semua isi tabel `mahasiswas`. Dan route `get /mahasiswas/{mahasiswa}` untuk menampilkan 1 data mahasiswa.

Menampilkan Semua Isi Tabel

Proses menampilkan semua tabel mahasiswa dilakukan oleh route `get /mahasiswas`. Route ini sudah kita buat sebelumnya:

```
Route::get('/mahasiswas', [MahasiswaController::class, 'index'])
->name('mahasiswas.index');
```

Namun sampai saat ini, isi dari method `index()` di `MahasiswaController` hanya sebatas menampilkan teks "*Tabel mahasiswa di sini*". Kita akan ubah dengan proses pembacaan data tabel `mahasiswas` menggunakan Eloquent.

Perintah Eloquent untuk mengambil semua data tabel sangat *simple*, yakni:

```
$mahasiswas = Mahasiswa::all();
```

Hasilnya, variabel `$mahasiswas` akan berisi collection dari semua data yang ada di tabel `mahasiswas`. Agar lebih menarik, saya ingin data tabel ini di proses oleh view dengan sedikit kode Bootstrap. Namun sebelum itu, mari bahas sedikit tentang struktur file view.

Selama proses pembuatan CRUD, kita perlu membuat beberapa file view. Agar lebih rapi, semua file ini akan saya tempatkan ke dalam folder `mahasiswa` di `resources\views\`.

Nama dari file view sebaiknya mengikuti nama controller yang memprosesnya. Sebagai

contoh, kode program untuk menampilkan semua tabel `mahasiswa` di tangani oleh method `index()` di `MahasiswaController`. Maka nama file blade yang memproses tampilan ini menjadi `index.blade.php`.

Struktur penamaan ini akan memudahkan kita (dan juga programmer lain) untuk mencari file view, karena terdapat sebuah standar nama file.

Sebelum ke view, saya akan siapkan proses pengiriman data di method `index()` terlebih dahulu:

`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2 //...
3     public function index()
4     {
5         $mahasiswa = Mahasiswa::all();
6         return view('mahasiswa.index',[ 'mahasiswa' => $mahasiswa]);
7     }

```

Ketika method `index()` di akses, ambil semua data di tabel `mahasiswa` dengan method `Mahasiswa::all()`, lalu tampung hasilnya ke dalam variabel `$mahasiswa`. Selanjutnya kirim variabel `$mahasiswa` ini ke dalam view `mahasiswa.index`.

Semoga anda juga masih ingat bahwa tanda titik di dalam penulisan nama view merujuk ke struktur folder. Artinya, view `mahasiswa.index` ini berada di `mahasiswa\index.blade.php`. Berikut kode program untuk view ini:

`resources/views/mahasiswa/index.blade.php`

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Data Mahasiswa</title>
9 </head>
10 <body>
11
12 <div class="container mt-3">
13     <div class="row">
14         <div class="col-12">
15
16             <div class="py-4">
17                 <h2>Tabel Mahasiswa</h2>
18             </div>
19
20             <table class="table table-striped">
21                 <thead>
22                     <tr>

```

```

23     <th>#</th>
24     <th>Nim</th>
25     <th>Nama</th>
26     <th>Jenis Kelamin</th>
27     <th>Jurusan</th>
28     <th>Alamat</th>
29   </tr>
30 </thead>
31 <tbody>
32 @forelse ($mahasiswa as $mahasiswa)
33   <tr>
34     <th>{{$loop->iteration}}</th>
35     <td>{{$mahasiswa->nim}}</td>
36     <td>{{$mahasiswa->nama}}</td>
37     <td>{{$mahasiswa->jenis_kelamin == 'P'?'Perempuan':'Laki-laki'}}</td>
38     <td>{{$mahasiswa->jurusan}}</td>
39     <td>{{$mahasiswa->alamat == '' ? 'N/A' : $mahasiswa->alamat}}</td>
40   </tr>
41 @empty
42   <td colspan="6" class="text-center">Tidak ada data...</td>
43 @endforelse
44 </tbody>
45 </table>
46 </div>
47 </div>
48 </div>
49
50 </body>
51 </html>
```

Semua data mahasiswa saya tempatkan ke dalam tabel HTML. Isi tabel akan di generate dengan perulangan `@forelse` di baris 33 – 40. Untuk setiap perulangan, tampilkan semua kolom mahasiswa yang diakses dari variabel collection `$mahasiswa`. Proses menampilkan data seperti ini sudah kita bahas di bab **Eloquent**.

Khusus untuk kolom jenis kelamin dan alamat, terdapat sedikit modifikasi tambahan. Di dalam tabel `mahasiswa`, jenis kelamin di simpan dengan karakter **P** atau **L**. Agar lebih rapi, saya ingin ketika ditampilkan nilainya tidak lagi berbentuk inisial, tapi teks lengkap 'Perempuan' atau 'Laki-laki'.

Salah satu solusi yang bisa dipakai adalah menggunakan operator kondisi ":" ?" seperti di baris 37. Yakni jika nilai `$mahasiswa->jenis_kelamin` sama dengan 'P' maka tampilkan teks 'Perempuan', selain itu, tampilkan teks 'Laki-laki'.

Begini juga dengan nilai dari `$mahasiswa->alamat`. Di tabel `mahasiswa`, kolom alamat bersifat opsional (boleh diisi atau tidak). Namun saya tidak ingin kolom ini terlihat kosong pada saat ditampilkan. Solusinya juga mirip seperti kolom jenis kelamin, yakni dengan operator kondisi seperti di baris 39. Jika ada mahasiswa yang tidak mengisi data alamat, tampilkan dengan teks 'N/A'.

Alternatif cara lain jika ingin mengubah nilai yang berasal dari tabel database adalah menggunakan **accessors** and **mutators** dari dalam Model. Tapi karena kita belum mempelajari teknik ini, untuk sementara tidak masalah jika ingin diubah langsung dari dalam view.

Silahkan buka halaman localhost:8000/mahasiswa, seharusnya akan tampil tabel berikut:



The screenshot shows a browser window with the title "Data Mahasiswa". The address bar displays "localhost:8000/mahasiswa". The main content area is titled "Tabel Mahasiswa" and contains a table with the following data:

#	Nim	Nama	Jenis Kelamin	Jurusan	Alamat
1	08125142	Anissa Lestari	Perempuan	Ilmu Komputer	Semarang
2	19003036	Sari Citra Lestari	Perempuan	Teknik Informatika	Jakarta

Gambar: Tampilan halaman localhost:8000/mahasiswa

Menampilkan Satu Data Tabel

Route kedua yang juga masuk kategori **Read** adalah proses menampilkan 1 data tabel. Dalam contoh kita, yang dimaksud adalah untuk menampilkan 1 data mahasiswa.

Jika merujuk ke tabel REST di awal bab, route yang diperlukan adalah sebagai berikut:

routes/web.php

```
Route::get('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'show'])
->name('mahasiswa.show');
```

Silahkan tambah ke dalam file routes\web.php. Route ini merupakan sebuah *route parameter*, sehingga di dalam method show() nantinya kita harus tulis variabel \$mahasiswa untuk menangkap nilai {mahasiswa}.

Dari konsep REST, parameter {mahasiswa} akan menampung **id** dari mahasiswa. Jika alamat URL yang diakses adalah /mahasiswa/1, maka tampilkan data mahasiswa yang memiliki **id = 1**. Jika yang diakses adalah /mahasiswa/257, maka tampilkan data mahasiswa yang memiliki **id = 257**, dst.

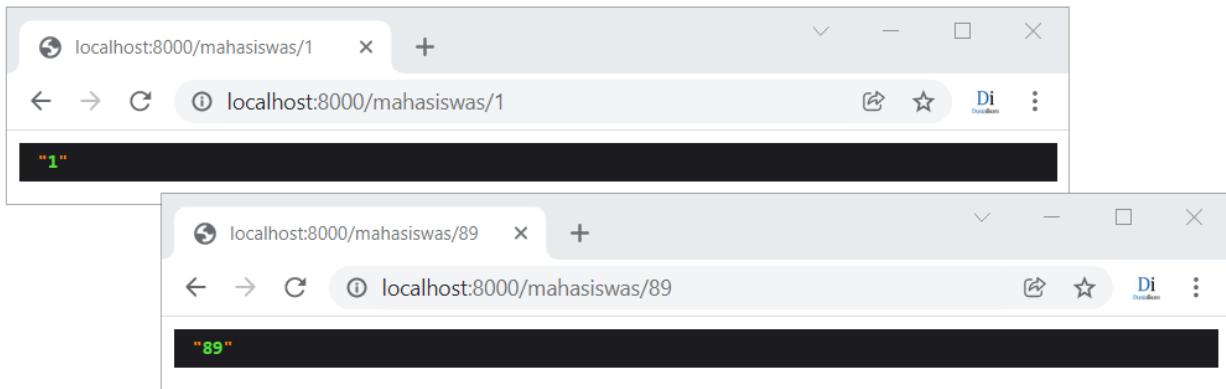
Untuk uji coba, tambah kode berikut ke dalam controller:

CRUD

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 /**
3  * public function show($mahasiswa)
4  {
5      dd($mahasiswa);
6 }
```

Di sini saya langsung men-dump() variabel \$mahasiswa yang berasal dari route. Silahkan coba akses beberapa alamat seperti <localhost:8000/mahasiswas/1> atau <localhost:8000/mahasiswas/89>.



Gambar: Percobaan akses method show()

Jika tampil angka yang sesuai dengan parameter alamat, maka tidak ada masalah. Silahkan modifikasi method show() dengan kode berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 /**
3  * public function show($mahasiswa)
4  {
5      $result = Mahasiswa::find($mahasiswa);
6      return view('mahasiswa.show', ['mahasiswa' => $result]);
7 }
```

Untuk mengambil data 1 mahasiswa ke dalam database, saya menggunakan method `find()` dari Eloquent. Hasilnya, variabel \$result akan berisi 1 object mahasiswa. Isi variabel \$result kemudian dikirim ke view `mahasiswa.show`.

Berikut kode program untuk view `mahasiswa\show.blade.php`:

resources/views/mahasiswa/show.blade.php

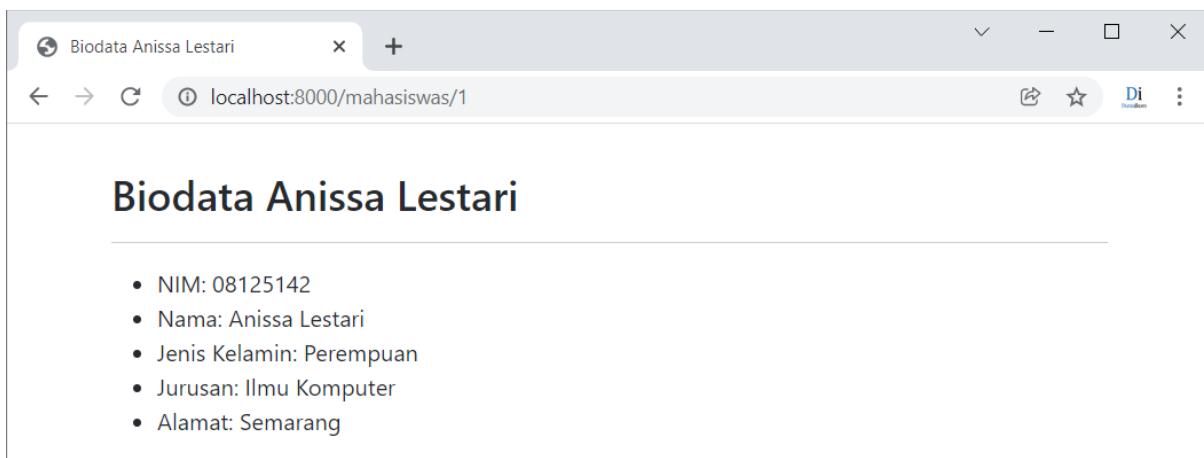
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```

6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <link href="/css/bootstrap.min.css" rel="stylesheet">
8   <title>Biodata {{$mahasiswa->nama}}</title>
9   </head>
10  <body>
11
12  <div class="container mt-3">
13    <div class="row">
14      <div class="col-12">
15
16        <div class="pt-3">
17          <h1 class="h2">Biodata {{$mahasiswa->nama}}</h1>
18        </div>
19        <hr>
20
21        <ul>
22          <li>NIM: {{$mahasiswa->nim}} </li>
23          <li>Nama: {{$mahasiswa->nama}} </li>
24          <li>Jenis Kelamin:
25            {{$mahasiswa->jenis_kelamin == 'P' ? 'Perempuan' : 'Laki-laki'}}</li>
26          <li>Jurusan: {{$mahasiswa->jurusan}} </li>
27          <li>Alamat:
28            {{$mahasiswa->alamat == '' ? 'N/A' : $mahasiswa->alamat}}</li>
29        </ul>
30
31      </div>
32    </div>
33  </div>
34
35 </div>
36
37 </body>
38 </html>

```

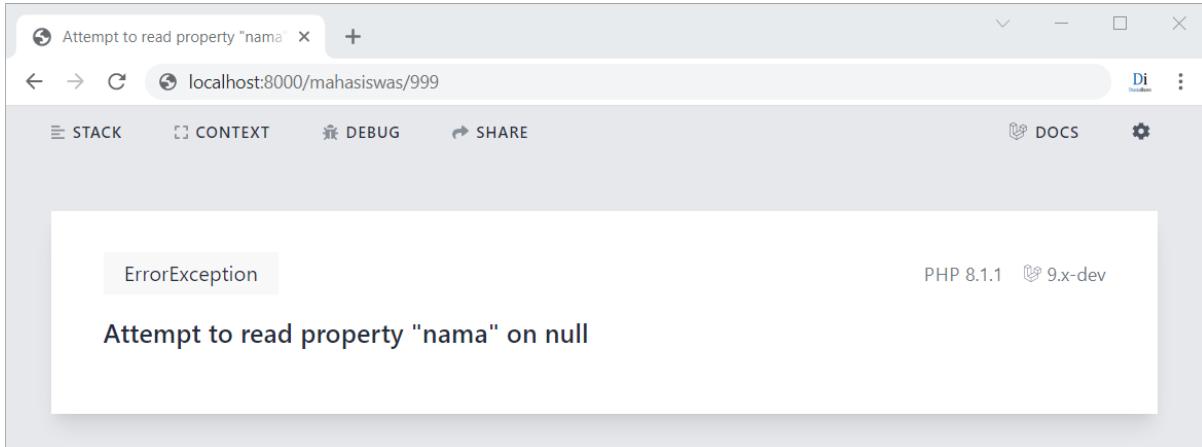
Dalam view ini, data yang dikirim dari method `show()` adalah variabel `$mahasiswa` yang berisi data 1 object mahasiswa. Dengan demikian, kita bisa memakai perintah `$mahasiswa->nim`, `$mahasiswa->nama`, dst untuk menampilkan data mahasiswa. Hasil ini akan saya buat dalam bentuk list:



Gambar: Hasil tampilan 1 data mahasiswa

Anda bisa coba dengan mengubah alamat URL, misalnya <localhost:8000/mahasiswa/2> untuk menampilkan data mahasiswa yang memiliki id 2, dst.

Bagaimana jika yang dibuka adalah <localhost:8000/mahasiswa/999>?



Gambar: Error pada saat menampilkan data

Error ini terjadi karena di dalam tabel `mahasiswa` tidak terdapat mahasiswa yang memiliki id 999. Untuk mengatasi masalah ini, ganti penggunaan method `Mahasiswa::find($mahasiswa)` di dalam controller menjadi `Mahasiswa::findOrFail($mahasiswa)`:

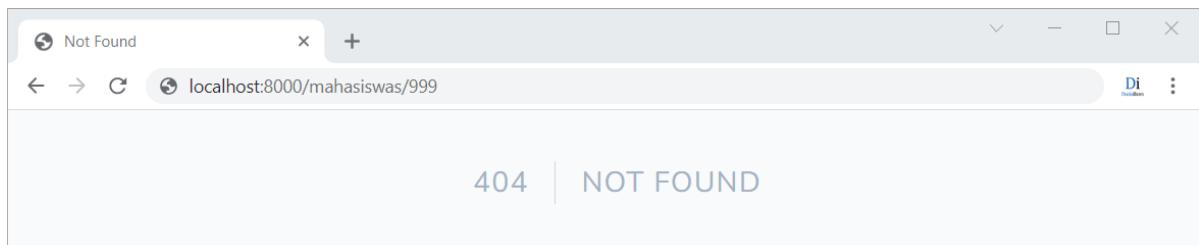
`app/Http/Controllers/MahasiswaController.php`

```

1 <?php
2 //...
3     public function show($mahasiswa)
4     {
5         $result = Mahasiswa::findOrFail($mahasiswa);
6         return view('mahasiswa.show',[ 'mahasiswa' => $result]);
7     }

```

Method `findOrFail()` akan menghasilkan halaman 404 jika data id mahasiswa tidak ditemukan di dalam tabel.



Gambar: Halaman 404 jika id mahasiswa tidak ditemukan

Route Model Binding

Alternatif lain yang lebih "sakti" untuk menampilkan 1 data mahasiswa adalah menggunakan teknik yang disebut sebagai **route model binding**. Di sebut seperti ini karena kita melakukan dependency injection class Model ke dalam parameter dari method `show()`. Berikut kode

program yang dimaksud:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2 //...
3     public function show(Mahasiswa $mahasiswa)
4     {
5         return view('mahasiswa.show',[ 'mahasiswa' => $mahasiswa]);
6     }

```

Perhatikan penulisan parameter method `show()` di baris 3, yakni `show(Mahasiswa $mahasiswa)`. Inilah yang dimaksud dengan **route model binding**, dimana kita melakukan type hint parameter `$mahasiswa` yang berisi id (berasal dari route), dengan class `Mahasiswa` yang berasal dari Model.

Hasilnya, parameter `$mahasiswa` langsung berisi object mahasiswa sesuai dengan id yang dikirim route. Karena sudah berisi 1 object mahasiswa, maka bisa langsung dikirim ke dalam view `mahasiswa.show`.

Route model binding ini kadang terasa seperti salah satu "magic" yang ada di Laravel. Yang jelas, it's just work!, meskipun kita tidak tau seperti apa Laravel melakukannya.

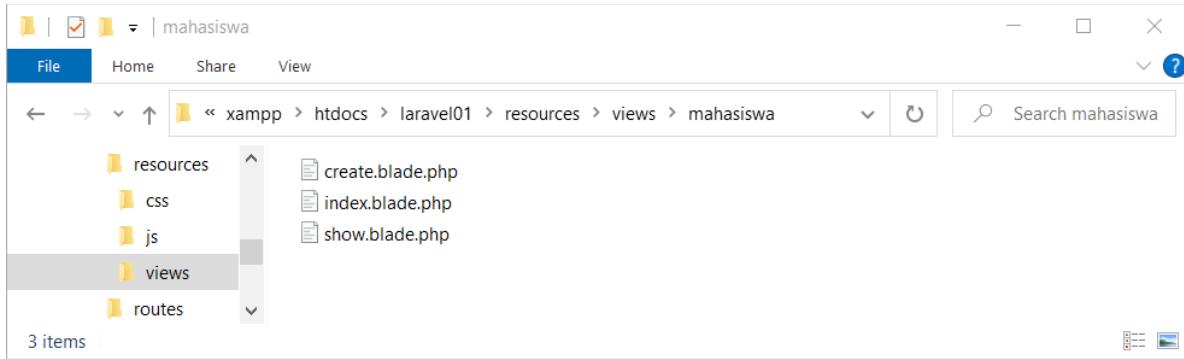
19.4. Refactoring File

Sebelum masuk ke materi CRUD selanjutnya, saya ingin merapikan sedikit kode program saat ini serta menambah beberapa fitur lengkap. **Refactoring** sendiri adalah istilah programming untuk menata ulang kode program agar lebih rapi tanpa mempengaruhi hasil akhir.

Refactoring Form Pendaftaran

Dalam materi tentang **Read**, saya menempatkan semua view ke dalam folder `mahasiswa`. Namun sebelumnya juga sudah ada view `form-pendaftaran.blade.php`.

Saya ingin memindahkan view ini ke dalam folder `mahasiswa` dan mengubah nama file menjadi `create.index.php` agar sesuai dengan nama method `create()` yang ada di dalam `MahasiswaController`, yakni method yang dipakai untuk menjalankan view ini.



Gambar: Isi folder views\mahasiswa

Setelah memindahkan file view, isi method `create()` juga harus di modifikasi agar mengakses alamat baru dari view form-pendaftaran:

app/Http/Controllers/MahasiswaController.php

```

1  <?php
2  //...
3  public function create()
4  {
5      return view('mahasiswa.create');
6  }

```

Sekarang semua file view sudah berada di dalam folder `mahasiswa`.

Refactoring View Index

Proses refactoring selanjutnya adalah untuk halaman index, yakni halaman yang menampilkan isi tabel semua mahasiswa. Saya ingin menambah 2 fitur baru:

1. Link untuk menampilkan 1 data mahasiswa (menuju route '`mahasiswas.show`')
2. Tombol untuk menambah data mahasiswa baru (menuju route '`mahasiswas.create`')

Fitur pertama sebenarnya cukup mudah, karena itu akan kita jadikan sebuah **exercise**.

#Excercise

Kita sudah bisa menampilkan data mahasiswa secara individu. Dan akan lebih praktis jika terdapat link untuk mengakses setiap data dari view `index.blade.php`, yakni dari halaman yang menampilkan semua data mahasiswa.

Berikut hasil akhir yang saya inginkan:

#	Nim	Nama	Jenis Kelamin	Jurusan	Alamat
1	08125142	Anissa Lestari	Perempuan	Ilmu Komputer	Semarang
2	19003036	Sari Citra Lestari	Perempuan	Teknik Informatika	Jakarta

Gambar: Teks Nim menjadi link

Kolom **Nim** di halaman `localhost:8000/mahasiswa` harus berfungsi sebagai link. Ketika nim di klik, itu akan membuka halaman `localhost:8000/mahasiswa/{id-mahasiswa}`. Tentunya id setiap mahasiswa akan berbeda-beda sesuai data yang ada di database.

Silahkan rancang sebentar seperti apa perubahan di view `index.blade.php`. Kode yang butuh modifikasi cukup 1 baris saja.

Answer

Untuk membuat kolom nim menjadi link, tambah tag `<a>` ke dalam tag `<td>` seperti contoh berikut:

```
resources/views/mahasiswa/index.blade.php
```

```

1 ...
2 <th>{{$loop->iteration}}</th>
3 <td><a href="#">{{$mahasiswa->nim}}</a></td>
4 <td>{{$mahasiswa->nama}}</td>
5 ...

```

Yang jadi masalah, apa kode yang harus ditulis ke dalam atribut `href`?

Di dalam halaman `index.blade.php`, terdapat variabel `$mahasiswa` yang berasal dari perulangan `@forelse ($mahasiswa as $mahasiswa)`. Variabel `$mahasiswa` ini berisi semua data dari 1 mahasiswa, termasuk kolom `id` yang bisa di akses dari `$mahasiswa->id`.

Kemudian alamat route yang dituju adalah `localhost:8000/mahasiswa/{mahasiswa}`, yang juga sudah di set dengan `named route 'mahasiswa.show'`.

Dari gabungan kedua nilai ini, atribut `href` bisa diisi dengan kode berikut:

```
href="{{ route('mahasiswa.show', ['mahasiswa' => $mahasiswa->id]) }}"
```

Ini adalah cara pengisian *route parameter* ke dalam *named route*. Bahasannya pernah kita pelajari di bagian akhir bab blade, yakni sub materi **Named Routes Parameter**. Anda bisa

buka sejenak sekedar menyegarkan ingatan.

Atau jika anda tidak ingin menggunakan *named parameter*, atribut href juga bisa diisi dengan kode berikut:

```
href="{{ url('/mahasiswa/' . $mahasiswa->id) }}"
```

Di sini saya langsung menyambung string "/mahasiswa/" dengan \$mahasiswa->id. Hasilnya akan di proses menjadi URL seperti localhost:8000/mahasiswa/1 atau localhost:8000/mahasiswa/2.

Modifikasi kedua adalah menambah sebuah tombol "**Tambah Mahasiswa**" di sisi kanan atas. Kode yang dibutuhkan juga tidak banyak, cukup membuat tag <a> dengan atribut href="{{ route('mahasiswa.create') }}".

Namun perlu beberapa class Bootstrap agar tombol ini tampil rapi dan sejajar dengan tag <h2>Tabel Mahasiswa</h2>. Berikut modifikasi untuk file index.blade.php:

Sebelum:

```
resources/views/mahasiswa/index.blade.php
```

```
1 ...
2 <div class="py-4">
3   <h2>Tabel Mahasiswa</h2>
4 </div>
5 ...
```

Sesudah:

```
resources/views/mahasiswa/index.blade.php
```

```
1 ...
2 <div class="py-4 d-flex justify-content-between align-items-center">
3   <h2>Tabel Mahasiswa</h2>
4   <a href="{{ route('mahasiswa.create') }}" class="btn btn-primary">
5     Tambah Mahasiswa
6   </a>
7 </div>
8 ...
```



The screenshot shows a web browser window titled "Data Mahasiswa". The URL in the address bar is "localhost:8000/mahasiswas". The page displays a table titled "Tabel Mahasiswa" with columns: #, Nim, Nama, Jenis Kelamin, Jurusan, and Alamat. There are two rows of data: row 1 has Nim 08125142, Name Anissa Lestari, Gender Perempuan, Major Ilmu Komputer, and Address Semarang; row 2 has Nim 19003036, Name Sari Citra Lestari, Gender Perempuan, Major Teknik Informatika, and Address Jakarta. A blue button labeled "Tambah Mahasiswa" is visible in the top right corner of the table area. A cursor arrow is pointing at the "Tambah Mahasiswa" button.

#	Nim	Nama	Jenis Kelamin	Jurusan	Alamat
1	08125142	Anissa Lestari	Perempuan	Ilmu Komputer	Semarang
2	19003036	Sari Citra Lestari	Perempuan	Teknik Informatika	Jakarta

Gambar: Penambahan tombol "Tambah Mahasiswa"

Saya menggunakan beberapa class `flexbox` Bootstrap untuk memisahkan judul `<h2>` di sisi kiri halaman, dan tag `<a>` tombol di sisi kanan halaman. Ketika tombol ini di klik, akan terbuka form pendaftaran mahasiswa.

19.5. Flash Data

Halaman index sudah cukup lengkap, dimana terdapat link untuk menampilkan data mahasiswa secara terpisah serta tombol untuk menambah mahasiswa baru. Namun saat ini ketika sebuah data mahasiswa berhasil di input, pesan yang tampil hanya berupa teks "Data berhasil diinput ke database".

Agar lebih user friendly, saya ingin begitu data mahasiswa berhasil di input ke dalam tabel, langsung di redirect ke halaman index beserta pesan "Penambahan data berhasil".

Ada 2 hal yang perlu kita rancang:

1. Membuat mekanisme redirect.
2. Menampilkan pesan sukses.

Untuk membuat redirect ke URL '/mahasiswa', bisa menggunakan perintah berikut:

```
return redirect('/mahasiswa');
```

Sebagai argument untuk function `redirect()` berupa route untuk alamat yang dituju. Namun karena kita menggunakan *named route*, bisa ditulis sebagai berikut:

```
return redirect()->route('mahasiswas.index');
```

Proses berikutnya adalah menampilkan pesan "Penambahan data berhasil". Untuk keperluan ini kita bisa memanfaatkan fitur Laravel yang dinamakan **Flash Data**.

Secara teknis, *flash data* akan menyimpan sebuah data atau pesan teks ke dalam session, namun pesan tersebut hanya aktif untuk 1 kali proses *redirect* saja. Setelah itu, *flash data*

otomatis dihapus dari session. Fitur seperti ini sangat cocok untuk menampilkan pesan teks yang memang hanya perlu tampil 1 kali saja.

Untuk membuat flash data, tersedia function **flash()**. Function ini butuh 2 buah argument, yakni **key** yang dipakai untuk mengakses flash data dan isi data tersebut. Berikut contoh penulisannya:

```
$request->session()->flash('pesan', 'Penambahan data berhasil');
```

Function **flash()** harus di-*chaining* dari **\$request->session()**, dimana variabel **\$request** berisi **Request** object hasil dari proses type hint. Setelah menjalankan function ini, pesan "Penambahan data berhasil" bisa diakses dari dalam view menggunakan function **session()->get('pesan')**.

Mari kita terapkan konsep ini ke dalam method **store()** yang sebelumnya dipakai untuk proses input data:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 //...
3     public function store(Request $request)
4     {
5         // $validateData = $request->validate([
6         //     'nim'          => 'required|size:8',
7         //     'nama'         => 'required|min:3|max:50',
8         //     'jenis_kelamin' => 'required|in:P,L',
9         //     'jurusan'       => 'required',
10        //     'alamat'        => '',
11        // ]);
12
13        // Mahasiswa::create($validateData);
14
15        $request->session()->flash('pesan', 'Penambahan data berhasil');
16        return redirect()->route('mahasiswas.index');
17    }
```

Untuk sementara, saya me-nonaktifkan proses validasi serta perintah untuk input data ke database (baris 5 - 13). Di baris 15 terdapat tambahan perintah untuk membuat pesan flash, yang kemudian diikuti proses *redirect* di baris 16. Simpan perubahan ini.

Kemudian tambah kode `{{ session()->get('pesan') }}` di baris paling atas **index.blade.php**:

resources/views/mahasiswa/index.blade.php

```
1 {{ session()->get('pesan') }}
2 <!DOCTYPE html>
3 <html lang="en">
4 ...
```

Sekarang saatnya percobaan. Silahkan buka form pendaftaran dari halaman

<localhost:8000/mahasiswa/create>, kemudian langsung saja klik tombol "Daftar". Halaman akan redirect ke <localhost:8000/mahasiswa> dan dibagian atas akan tampil teks 'Penambahan data berhasil':



The screenshot shows a browser window titled 'Data Mahasiswa' with the URL 'localhost:8000/mahasiswa'. A red arrow points to the status message 'Penambahan data berhasil' (Data addition successful) displayed above the table. The table below has columns: #, Nim, Nama, Jenis Kelamin, Jurusan, and Alamat. It contains two rows of data.

#	Nim	Nama	Jenis Kelamin	Jurusan	Alamat
1	08125142	Anissa Lestari	Perempuan	Ilmu Komputer	Semarang
2	19003036	Sari Citra Lestari	Perempuan	Teknik Informatika	Jakarta

Gambar: Tampilan pesan flash

Kemudian silahkan refresh halaman index ini, maka pesan flash tadi akan langsung hilang, padahal perintah {{ session()->get('pesan') }} tetap ada di bagian atas halaman. Inilah cara kerja dari flash data, yang langsung terhapus setelah 1 kali akses.

Kembali ke method `store()`, saya akan aktifkan kembali proses validasi, serta menambah nama mahasiswa ke dalam pesan flash:

app/Http/Controllers/MahasiswaController.php

```

1  <?php
2  //...
3      public function store(Request $request)
4      {
5          $validateData = $request->validate([
6              'nim'           => 'required|size:8',
7              'nama'          => 'required|min:3|max:50',
8              'jenis_kelamin' => 'required|in:P,L',
9              'jurusan'       => 'required',
10             'alamat'        => '',
11         ]);
12
13         Mahasiswa::create($validateData);
14
15         $request->session()->flash('pesan', "Penambahan data
16             {$validateData['nama']} berhasil");
17         return redirect()->route('mahasiswa.index');
18     }

```

Isi pesan flash data saya modifikasi sedikit dengan penambahan `[$validateData['nama']]`, nantinya ini akan berganti dengan nama mahasiswa yang baru saja di daftarkan.

Sebagai contoh, jika yang diinput ke dalam form adalah "Andi Permana", maka pesan flash akan

berisi: 'Penambahan data Andi Permana berhasil'. Perhatikan juga karena array `[$validateData['nama']]` di akses dari dalam string, maka penulisan pesan flash ada di dalam tanda kutip dua, bukan tanda kutip satu.

Untuk view `index.blade.php`, saya ingin pesan flash tampil di dalam class `.alert` Bootstrap dan posisinya berada setelah judul:

`resources/views/mahasiswa/index.blade.php`

```

1 ...
2 <div class="py-4 d-flex justify-content-between align-items-center">
3   <h2>Tabel Mahasiswa</h2>
4   <a href="{{ route('mahasiswa.create') }}" class="btn btn-primary">
5     Tambah Mahasiswa
6   </a>
7 </div>
8
9 @if(session()->has('pesan'))
10 <div class="alert alert-success">
11   {{ session()->get('pesan') }}
12 </div>
13 @endif
14
15 <table class="table table-striped">
16   <thead>
17 ...

```

Pesan flash ditampilkan oleh perintah `{{ session()->get('pesan') }}` di baris 11, yang berada di dalam tag `<div>` dengan class `.alert` dan `.alert-success` bawaan Bootstrap.

Namun tag `<div>` ini juga berada di dalam kondisi `@if(session()->has('pesan'))`. Kondisi if ini saya buat agar tag `<div>` hanya tampil jika terdeteksi ada 'pesan' di dalam session. Jika tidak ditemukan flash data dengan key 'pesan', maka tag `<div>` ini tidak akan diproses.

Mari kita coba, silahkan isi data baru ke dalam form, klik tombol Daftar, dan jika semuanya sesuai akan tampil pesan alert di bagian atas tabel halaman index:

The screenshot shows a browser window titled "Data Mahasiswa". The address bar says "localhost:8000/mahasiswas". The main content area has a title "Tabel Mahasiswa" and a button "Tambah Mahasiswa". A green flash message box contains the text "Penambahan data Rudi Satria berhasil". Below it is a table with columns: #, Nim, Nama, Jenis Kelamin, Jurusan, and Alamat. The table has 4 rows, including the header row.

#	Nim	Nama	Jenis Kelamin	Jurusan	Alamat
1	08125142	Anissa Lestari	Perempuan	Ilmu Komputer	Semarang
2	19003036	Sari Citra Lestari	Perempuan	Teknik Informatika	Jakarta
3	19005011	Rudi Satria	Laki-laki	Teknik Telekomunikasi	N/A

Gambar: Proses penambahan data berhasil dan tampil pesan flash

Alternatif cara lain untuk membuat pesan flash adalah menggunakan function `with()` yang langsung di chaining dengan pemanggilan fungsi `redirect()`:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2 //...
3     public function store(Request $request)
4     {
5         $validateData = $request->validate([
6             'nim'          => 'required|size:8',
7             'nama'         => 'required|min:3|max:50',
8             'jenis_kelamin' => 'required|in:P,L',
9             'jurusan'      => 'required',
10            'alamat'       => '',
11        ]);
12
13        Mahasiswa::create($validateData);
14
15        return redirect()->route('mahasiswas.index')->with('pesan',
16                                "Penambahan data {$validateData['nama']} berhasil");
17    }

```

Hasilnya akan sama seperti sebelumnya, dimana pesan flash tetap sampai ke view `index.blade.php`.

19.6. Validasi Duplikasi Data

Proses input data mahasiswa sudah berhasil dijalankan dan juga tampil dengan rapi. Tapi masih ada satu masalah terkait validasi. Silahkan anda coba input nomor nim yang sama dengan yang sudah ada di tabel mahasiswas:



Gambar: Error saat menginput data yang sama

Error SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry terjadi karena pada saat perancangan tabel mahasiswa, kolom nim saya set dengan atribut unique. Di MySQL, kolom dengan atribut unique tidak bisa diisi dengan data yang sama.

Bagaimana cara mengatasi error ini? Tentunya sebelum proses insert kita harus cek dulu ke tabel `mahasiswa`, apakah sudah ada nim yang sama atau belum. Jika sudah ada, tampilkan pesan error dan minta user untuk menginput nomor nim lain.

Jika menggunakan PHP native, kode yang dibutuhkan akan cukup panjang. Tapi dengan Laravel, cukup tambah 1 syarat validasi lagi ke dalam inputan nim:

```
1 $validateData = $request->validate([
2     'nim'          => 'required|size:8|unique:mahasiswa',
3     //...
4 ]);
```

Syarat tambahan adalah '`unique:mahasiswa`'. Jika ditulis seperti ini, maka ketika Laravel melakukan proses validasi, akan di periksa sampai ke tabel `mahasiswa`. Jika terdapat nilai nim yang sama, maka tampilkan pesan error.

Secara default, Laravel mengasumsikan nama kolom tabel sama dengan nama inputan form, yang dalam contoh kita memang sama-sama menggunakan nama **nim**. Namun seandainya nama kolom tabel tidak sama dengan nama inputan form, bisa ditulis sebagai berikut:

```
1 $validateData = $request->validate([
2     'nim'          => 'required|size:8|unique:mahasiswa,nim_mhs',
3     //...
4 ]);
```

Syarat validasi '`unique:mahasiswa,nim_mhs`' berarti Laravel akan memeriksa kolom `nim_mhs` di tabel `mahasiswa`.

Dengan tambahan syarat ini, berikut modifikasi dari method `store()`:

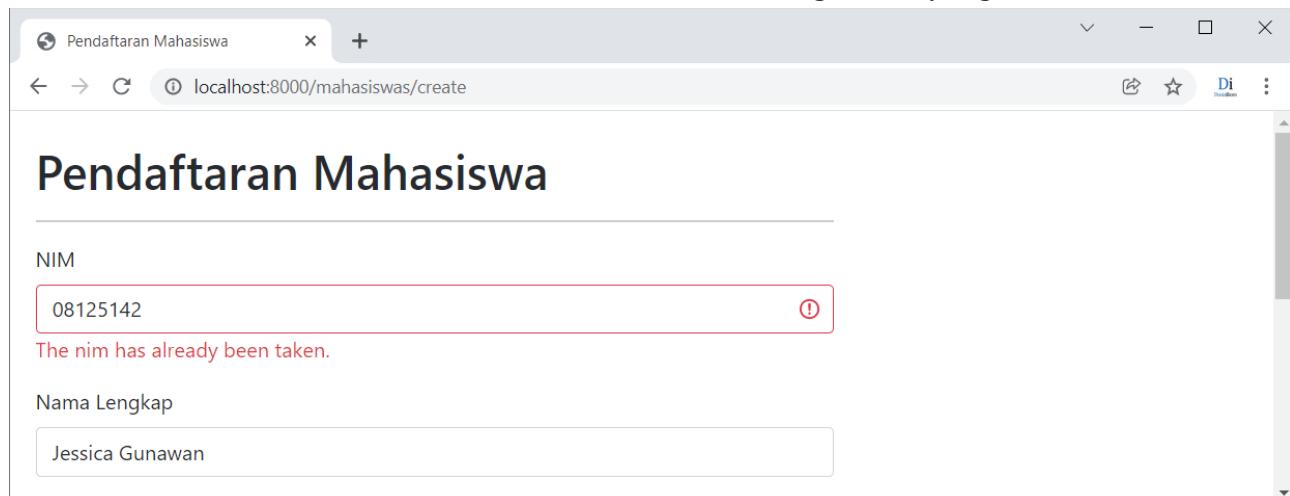
`app/Http/Controllers/MahasiswaController.php`

```

1  <?php
2  //...
3      public function store(Request $request)
4      {
5          $validateData = $request->validate([
6              'nim'          => 'required|size:8|unique:mahasiswas',
7              'nama'         => 'required|min:3|max:50',
8              'jenis_kelamin' => 'required|in:P,L',
9              'jurusan'       => 'required',
10             'alamat'        => '',
11         ]);
12
13         Mahasiswa::create($validateData);
14
15         return redirect()->route('mahasiswas.index')->with('pesan',
16             "Penambahan data {$validateData['nama']} berhasil");
17     }

```

Save controller dan coba daftarkan kembali mahasiswa dengan nim yang sudah ada:



Gambar: Error nim sudah ada

Hasilnya, tampil pesan error 'The nim has already been taken' di bawah inputan form, sehingga user bisa memilih nomor nim lain.

Dalam bab ini saya tidak memodifikasi folder `lang`, sehingga pesan validasi tampil dalam bahasa inggris. Jika ingin pesan error tampil dalam bahasa Indonesia, silahkan ikuti cara yang kita bahas dalam bab **Localization**.

19.7. Update

Materi CRUD berikutnya adalah **Update**, yakni proses perubahan data yang sudah tersimpan di dalam database. Proses update ini terdiri dari 2 bagian: menampilkan form yang sudah berisi data dari database, serta menjalankan query UPDATE menggunakan Eloquent.

Berdasarkan tabel REST, kita perlu membuat 2 buah route, yakni **get** /mahasiswa/{mahasiswa}/edit untuk menampilkan form dan **patch** /mahasiswa/{mahasiswa} untuk melakukan proses update.

Menampilkan Form Update

Proses menampilkan form update akan dilakukan oleh route berikut:

routes/web.php

```
Route::get('/mahasiswa/{mahasiswa}/edit', [MahasiswaController::class, 'edit'])
->name('mahasiswa.edit');
```

Silahkan tambah route di atas ke dalam file routes\web.php. Struktur URL route ini terdiri dari 3 segmen, yakni /mahasiswa/, {mahasiswa} dan /edit. Sama seperti route untuk proses **read**, route parameter {mahasiswa} dipakai untuk menampung **id** dari mahasiswa yang ingin di edit.

Sebagai contoh, jika saya ingin mengedit mahasiswa yang memiliki **id = 2**, maka alamat URL yang harus ditulis adalah /mahasiswa/2/edit. Atau jika saya ingin mengedit mahasiswa dengan **id = 674**, dilakukan dari alamat /mahasiswa/674/edit.

Method yang akan memproses route ini adalah method **edit()** di dalam **MahasiswaController**. Berikut kode program untuk method ini:

app\Http\Controllers\MahasiswaController.php

```
1 <?php
2 //...
3     public function edit(Mahasiswa $mahasiswa)
4     {
5         return view('mahasiswa.edit',[ 'mahasiswa' => $mahasiswa]);
6     }
```

Di sini saya kembali menggunakan teknik **route model binding**, dimana variabel \$mahasiswa akan langsung berisi object **Mahasiswa** yang sedang diakses dari route.

Maksudnya, jika yang diakses saat ini adalah halaman /mahasiswa/2/edit, angka 2 akan diteruskan ke method **edit()**. Sesampainya di method **edit()**, angka ini 'ditangkap' oleh variabel \$mahasiswa yang ditulis sebagai parameter. Kemudian dengan sedikit 'magic' dari Laravel, secara otomatis akan mengambil semua data mahasiswa dengan id = 2 dari database dan menginputnya kembali ke dalam variabel \$mahasiswa,

Karena variabel \$mahasiswa sudah berisi object dari 1 mahasiswa yang ingin di edit, kita tinggal

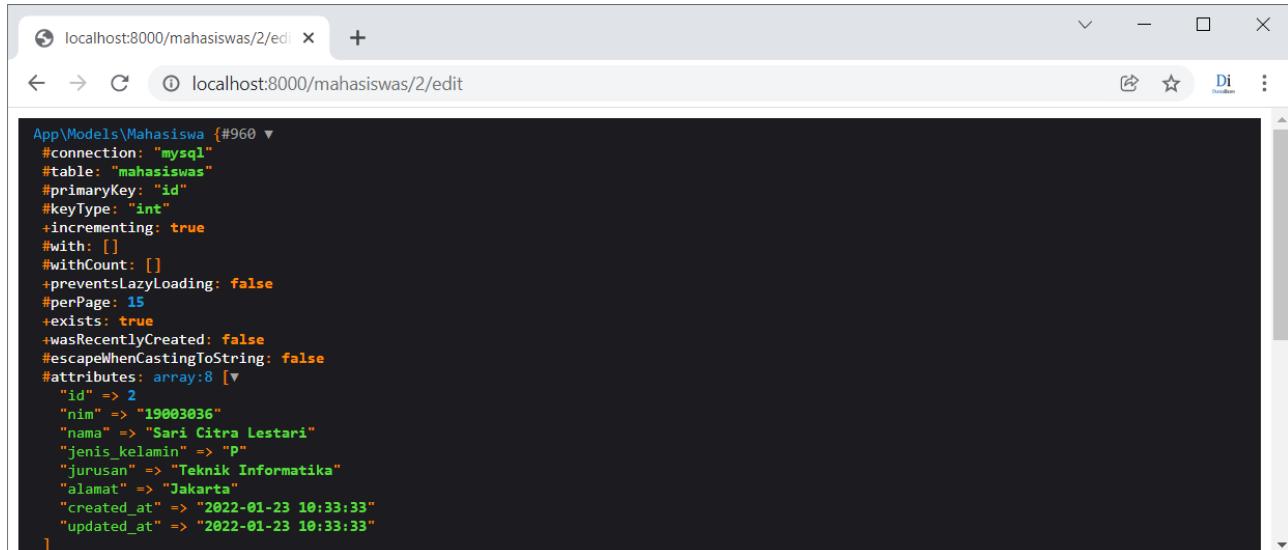
meneruskannya ke view `mahasiswa.edit` untuk di proses lebih lanjut.

View `mahasiswa.edit` ini berisi kode HTML yang sama persis seperti form `create` (form pendaftaran mahasiswa). Namun ada beberapa perbedaan terkait cara menampilkan data. Silahkan buat file `edit.blade.php` di folder `resources\views\mahasiswa`, lalu isi dengan kode berikut:

```
resources/views/mahasiswa/edit.blade.php
```

```
1 {{ dump($mahasiswa) }}
```

Betul, hanya 1 baris saja. Ini untuk memastikan kode program di route dan controller sudah benar. Setelah itu akses halaman localhost:8000/mahasiswa/1/edit dan localhost:8000/mahasiswa/2/edit untuk uji coba. Jika tidak ada kendala, akan tampil hasil dump yang berisi object dari 1 data mahasiswa sesuai nilai id yang ada di dalam URL:



```
App\Models\Mahasiswa {#960 ▾
  #connection: "mysql"
  #table: "mahasiswa"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  +preventsLazyLoading: false
  #perPage: 15
  +exists: true
  +wasRecentlyCreated: false
  #escapesWhenCastingToString: false
  #attributes: array:8 [▼
    "id" => 2
    "nim" => "19003036"
    "nama" => "Sari Citra Lestari"
    "jenis_kelamin" => "P"
    "jurusan" => "Teknik Informatika"
    "alamat" => "Jakarta"
    "created_at" => "2022-01-23 10:33:33"
    "updated_at" => "2022-01-23 10:33:33"
  ]
}
```

Gambar: Hasil dump dari data mahasiswa yang akan di edit

Dalam gambar ini, alamat di dalam URL adalah `localhost:8000/mahasiswa/2/edit`, sehingga yang akan tampil adalah object dari mahasiswa dengan `id = 2`, yakni 'Sari Citra Lestari'. Tugas kita sekarang adalah, mencari cara untuk menampilkan data ini ke dalam bentuk form.

Untuk memulai proses edit, data mahasiswa sudah harus tersedia di dalam form. Ini bisa dilakukan dengan cara mengisi atribut `value` inputan form dengan data yang berasal dari variabel `$mahasiswa`, kira-kira sebagai berikut:

```
<input type="text" name="nim" value="{{ $mahasiswa->nim }}">
```

Dengan menulis seperti ini, maka pada saat form tampil, inputan form sudah langsung berisi data `nim` mahasiswa yang akan di edit.

Tapi jika anda masih ingat, atribut `value` ini juga harus diisi dengan method `old()` untuk proses repopulate form:

```
<input type="text" name="nim" value="{{ old('nim') }}">
```

Jadi, bagaimana solusi dari masalah ini?

Idenya adalah, kita bisa buat sebuah pemeriksaan kondisi. Pada saat form ditampilkan pertama kali, fungsi `old('nim')` tidak berisi apa-apa. Jika ini yang terjadi, ambil data dari `$mahasiswa->nim`. Namun jika form tampil dari hasil `redirect` (karena tidak lolos validasi), tampilkan hasil dari `old('nim')`.

Berikut kode program yang bisa dipakai:

```
1 <input type="text" name="nim"
2 value=
3 @if(empty(old('nim')))
4     {{ $mahasiswa->nim }}
5 @else
6     {{ old('nim') }}
7 @endif
8 >
```

Perintah `@if` di baris 3 – 7 adalah penerapan dari penjelasan sebelumnya. Yakni jika `old('nim')` kosong, tampilkan `$mahasiswa->nim`. Namun jika `old('nim')` berisi sesuatu, tampilkan nilai `old('nim')` tersebut.

Logika program kita sudah sesuai, tapi kode di atas kurang rapi dan terlalu panjang.

Untungnya, terdapat operator baru milik PHP 7 yang sangat pas untuk keperluan ini, yaitu *null coalescing operator*.

Null coalescing operator adalah operator khusus untuk proses perbandingan dengan memeriksa ada atau tidaknya nilai di sebuah variabel. Operator ini ditulis dengan tanda tanya dua kali " ?? ". Dengan *null coalescing operator*, proses perbandingan sebelumnya bisa ditulis dalam 1 baris saja:

```
<input type="text" name="nim" value="{{ old('nim') ?? $mahasiswa->nim }}">
```

Perintah `{{ old('nim') ?? $mahasiswa->nim }}` akan menampilkan `old('nim')` jika berisi sesuatu, atau `$mahasiswa->nim` jika ternyata `old('nim')` kosong. Cara penggunaan *null coalescing operator* juga pernah kita bahas di buku **PHP Uncover**.

Kemudian, bagaimana untuk inputan **nama** mahasiswa? Tidak masalah, tinggal mengubah nama variabelnya saja:

```
<input type="text" name="nama" value="{{ old('nama') ?? $mahasiswa->nama }}">
```

Yang sedikit ribet ada di inputan **jenis_kelamin** yang dibuat dari radio button, serta inputan jurusan yang dibuat dari tag `<select>`. Tapi kita bisa menerapkan ide yang sama.

Saat ini, inputan jenis kelamin di proses sebagai berikut:

```
<input type="radio" name="jenis_kelamin" value="L"
{{ old('jenis_kelamin')=='L' ? 'checked': '' }} >
```

Kode di baris 2 berfungsi untuk repopulate form, yakni jika `old('jenis_kelamin')` berisi "L", tampilkan teks 'checked'. Kita bisa tambah `null coalescing operator` ke dalam perintah ini:

```
<input type="radio" name="jenis_kelamin" value="L"
{{ (old('jenis_kelamin') ?? $mahasiswa->jenis_kelamin) == 'L' ? 'checked': '' }} >
```

Sekarang, sebelum dibandingkan dengan "L", akan dijalankan terlebih dahulu proses pemeriksaan nilai inputan apakah akan diambil dari `old('jenis_kelamin')` atau `$mahasiswa->jenis_kelamin`. Baru kemudian salah satu nilai yang didapat akan dibandingkan dengan "L".

Tulis hal yang sama untuk inputan **jenis_kelamin** perempuan:

```
<input type="radio" name="jenis_kelamin" value="P"
{{ (old('jenis_kelamin') ?? $mahasiswa->jenis_kelamin) == 'P' ? 'checked': '' }} >
```

Untuk inputan **jurusank**, kodennya sedikit panjang tetapi masih menggunakan teknik yang sama:

```
<select name="jurusan" id="jurusan">
  <option value="Teknik Informatika"
    {{(old('jurusan') ?? $mahasiswa->jurusan)=='Teknik Informatika'? 'selected': '')}}>
    Teknik Informatika
  </option>
  <option value="Sistem Informasi"
    {{(old('jurusan') ?? $mahasiswa->jurusan)=='Sistem Informasi' ? 'selected': '')}}>
    Sistem Informasi
  </option>
  <option value="Ilmu Komputer"
    {{(old('jurusan') ?? $mahasiswa->jurusan)=='Ilmu Komputer' ? 'selected': '')}}>
    Ilmu Komputer
  </option>
  <option value="Teknik Komputer"
    {{(old('jurusan') ?? $mahasiswa->jurusan)=='Teknik Komputer' ? 'selected': '')}}>
    Teknik Komputer
  </option>
  <option value="Teknik Telekomunikasi"
    {{(old('jurusan') ?? $mahasiswa->jurusan)=='Teknik Telekomunikasi'? 'selected': '')}}>
    Teknik Telekomunikasi
  </option>
</select>
```

Sebagai contoh, kita akan bahas pilihan jurusan **Teknik Informatika**.

Pertama, akan di periksa terlebih dahulu apakah nilainya diambil dari `old('jurusan')` atau `$mahasiswa->jurusan`. Setelah di dapat, periksa apakah berisi string 'Teknik Informatika' atau tidak. Jika iya (**true**), maka tambah atribut `selected` ke dalam tag `<option>`. Jika tidak (**false**), tampilkan string kosong. Lakukan hal yang sama untuk kelima pilihan jurusan.

Terakhir untuk inputan **alamat**, tulis `null coalescing operator` di antara tag `<textarea>`:

```
<textarea name="alamat">&{{ old('alamat') ?? $mahasiswa->alamat }}</textarea>
```

Dengan penulisan ini, alamat akan diambil dari nilai `old('alamat')` jika form terbuka karena redirect validasi, atau dari `$mahasiswa->alamat` jika form dibuka untuk proses edit.

Form kita sudah selesai, berikut kode program lengkap dari **Form Edit Mahasiswa**, yakni file `edit.blade.php`:

`resources/views/mahasiswa/edit.blade.php`

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link href="/css/bootstrap.min.css" rel="stylesheet">
8      <title>Edit Mahasiswa</title>
9  </head>
10 <body>
11
12 <div class="container pt-4 bg-white">
13     <div class="row">
14         <div class="col-md-8 col-xl-6">
15             <h1>Edit Mahasiswa</h1>
16             <hr>
17
18             <form action="" method="POST">
19                 @csrf
20                 <div class="mb-3">
21                     <label class="form-label" for="nim">NIM</label>
22                     <input type="text" id="nim" name="nim"
23                         value="{{ old('nim') ?? $mahasiswa->nim }}"
24                         class="form-control @error('nim') is-invalid @enderror">
25                     @error('nim')
26                         <div class="text-danger">{{ $message }}</div>
27                     @enderror
28                 </div>
29
30                 <div class="mb-3">
31                     <label class="form-label" for="nama">Nama Lengkap</label>
32                     <input type="text" id="nama" name="nama"
33                         value="{{ old('nama') ?? $mahasiswa->nama }}"
34                         class="form-control @error('nama') is-invalid @enderror">
35                     @error('nama')
36                         <div class="text-danger">{{ $message }}</div>
37                     @enderror
38                 </div>
39
40                 <div class="mb-3">
41                     <label class="form-label">Jenis Kelamin</label>
42                     <div class="d-flex">
43                         <div class="form-check me-3">
44                             <input class="form-check-input" type="radio" name="jenis_kelamin">
```

```

45      id="laki_laki" value="L"
46      {{ (old('jenis_kelamin') ?? $mahasiswa->jenis_kelamin)
47      == 'L' ? 'checked': '' }}>
48      <label class="form-check-label" for="laki_laki">Laki-laki</label>
49      </div>
50      <div class="form-check">
51          <input class="form-check-input" type="radio" name="jenis_kelamin"
52          id="perempuan" value="P"
53          {{ (old('jenis_kelamin') ?? $mahasiswa->jenis_kelamin)
54          == 'P' ? 'checked': '' }}>
55          <label class="form-check-label" for="perempuan">Perempuan</label>
56          </div>
57      </div>
58      @error('jenis_kelamin')
59          <div class="text-danger">{{ $message }}</div>
60      @enderror
61  </div>
62
63  <div class="mb-3">
64      <label class="form-label" for="jurusan">Jurusan</label>
65      <select class="form-select" name="jurusan" id="jurusan"
66      value="{{ old('jurusan') }}>
67          <option value="Teknik Informatika"
68          {{ (old('jurusan') ?? $mahasiswa->jurusan)==
69          'Teknik Informatika' ? 'selected': '' }} >
70          Teknik Informatika
71          </option>
72          <option value="Sistem Informasi"
73          {{ (old('jurusan') ?? $mahasiswa->jurusan)==
74          'Sistem Informasi' ? 'selected': '' }} >
75          Sistem Informasi
76          </option>
77          <option value="Ilmu Komputer"
78          {{ (old('jurusan') ?? $mahasiswa->jurusan)==
79          'Ilmu Komputer' ? 'selected': '' }} >
80          Ilmu Komputer
81          </option>
82          <option value="Teknik Komputer"
83          {{ (old('jurusan') ?? $mahasiswa->jurusan)==
84          'Teknik Komputer' ? 'selected': '' }} >
85          Teknik Komputer
86          </option>
87          <option value="Teknik Telekomunikasi"
88          {{(old('jurusan'))??$mahasiswa->jurusan)==
89          'Teknik Telekomunikasi'? 'selected': ''}} >
90          Teknik Telekomunikasi
91          </option>
92      </select>
93      @error('jurusan')
94          <div class="text-danger">{{ $message }}</div>
95      @enderror
96  </div>
97
98  <div class="mb-3">
99      <label class="form-label" for="alamat">Alamat</label>

```

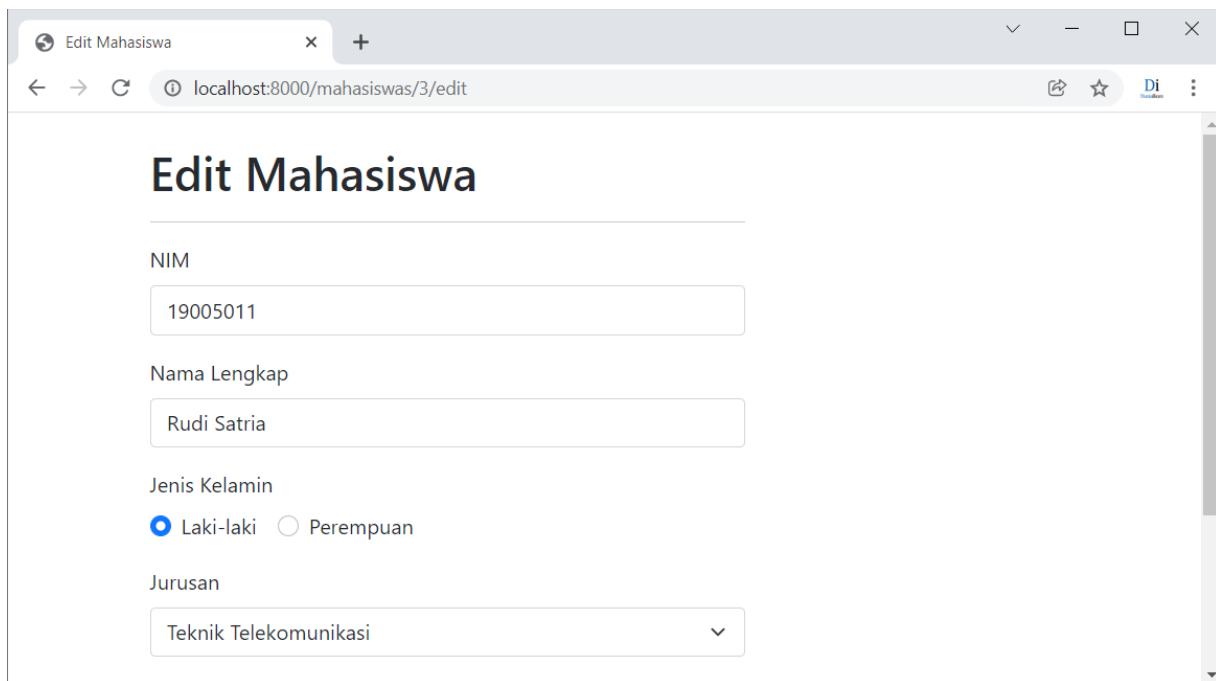
```

100      <textarea class="form-control" id="alamat" rows="3"
101          name="alamat">{{ old('alamat') ?? $mahasiswa->alamat}}</textarea>
102      </div>
103
104      <button type="submit" class="btn btn-primary mb-2">Update</button>
105      </form>
106
107  </div>
108 </div>
109 </div>
110
111 </body>
112 </html>

```

Kode programnya memang agak panjang, tapi semuanya sudah kita bahas. Khusus untuk atribut `action` di dalam tag `<form>` saat ini saya kosongkan (baris 18) karena akan diproses oleh route yang kita bahas sesaat lagi. Tombol submit form juga saya tujar menjadi tombol "**Update**" (baris 104).

Silahkan tes form ini. Buka alamat localhost:8000/mahasiswa/1/edit atau localhost:8000/mahasiswa/2/edit, maka inputan form sudah langsung berisi data yang diambil dari database.



Gambar: Form Edit Mahasiswa

Proses Update Tabel

Ketika form edit mahasiswa selesai diisi dan user men-klik tombol **Update** (submit form), maka data tersebut akan dikirim ke route tersendiri, yakni **put** `/mahasiswa/{mahasiswa}`.

Jika dilihat dari tabel REST, sebenarnya kita bebas memilih apakah ingin menggunakan **put**

/mahasiswa/{mahasiswa} atau **patch** /mahasiswa/{mahasiswa}. Laravel tidak membedakan kedua method ini. Akan tetapi dalam teori REST ada sedikit perbedaan antara keduanya.

Ketika mengupdate data, method **put** butuh semua data baru, meskipun data itu tidak berubah. Sedangkan **patch** hanya butuh sebagian data saja, yakni data yang berubah.

Sebagai contoh, misalkan kita ingin mengubah pilihan jurusan dari mahasiswa dengan nim 19005011. Data lain seperti nama, jenis kelamin dan alamat masih sama seperti sebelumnya.

Jika menggunakan method put, semua data harus dikirim ke server, meliputi id, nim, nama, jenis kelamin, pilihan jurusan dan alamat. Sedangkan jika menggunakan method patch, yang perlu dikirim hanya id dan pilihan jurusan saja. Id diperlukan sebagai identifikasi dari mahasiswa tersebut.

Karena menggunakan eloquent, setiap kali terjadi update, semua data tetap dikirim ke server, sehingga yang dipakai adalah konsep put. Akan tetapi dari salah satu artikel di forum laracasts.com terdapat penjelasan bahwa itu pun sebenarnya bukan put "murni", sebab di tabel masih ada kolom `created_at` dan `updated_at` yang diisi eloquent secara otomatis. Jadi kita tidak benar-benar mengirim seluruh data baru.

Terlepas dari teorinya, laravel membebaskan kita menggunakan put maupun patch. Karena lebih umum dipakai, dalam praktek ini saya akan pakai **put** saja. Anda bebas jika ingin menggunakan patch.

Silahkan tambah route berikut untuk proses update tabel:

routes/web.php

```
Route::put('/mahasiswas/{mahasiswa}', [MahasiswaController::class, 'update'])
->name('mahasiswas.update');
```

Di sini kita berkenalan dengan method baru milik **Route** class, yakni `Route::put()`. Route **put** nantinya butuh sedikit pengaturan, karena HTTP hanya mendukung method get dan post saja.

Route ini juga akan mengirim route parameter `{mahasiswa}` ke method `update()` di `MahasiswaController`. Sama seperti sebelumnya, `{mahasiswa}` tidak lain berisi nilai **id** dari mahasiswa yang akan di update.

Berikut kode program untuk method `update()` ini:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 //...
3     public function update(Request $request, Mahasiswa $mahasiswa)
4     {
5         dump($request->all());
6         dump($mahasiswa);
7     }
}
```

Betul, ini hanya *dummy code* sekedar uji coba apakah data `$request` (berasal dari form) dan `$mahasiswa` (berasal dari route) sudah sampai ke dalam controller atau tidak.

Sebelum percobaan, silahkan buka kembali file view untuk form update, yakni `edit.blade.php`. Kita harus modifikasi nilai atribut `action` dari tag `<form>`.

Dari sebelumnya:

```

1 ...
2 <form action="" method="POST">
3   @csrf
4   <div class="mb-3">
5     <label class="form-label" for="nim">NIM</label>
6     <input type="text" id="nim" name="nim"
7   ...

```

Menjadi:

```

1 ...
2 <form action="{{ route('mahasiswas.update', ['mahasiswa' => $mahasiswa->id]) }}" method="POST">
3   @method('PUT')
4   @csrf
5   <div class="mb-3">
6     <label class="form-label" for="nim">NIM</label>
7     <input type="text" id="nim" name="nim"
8   ...

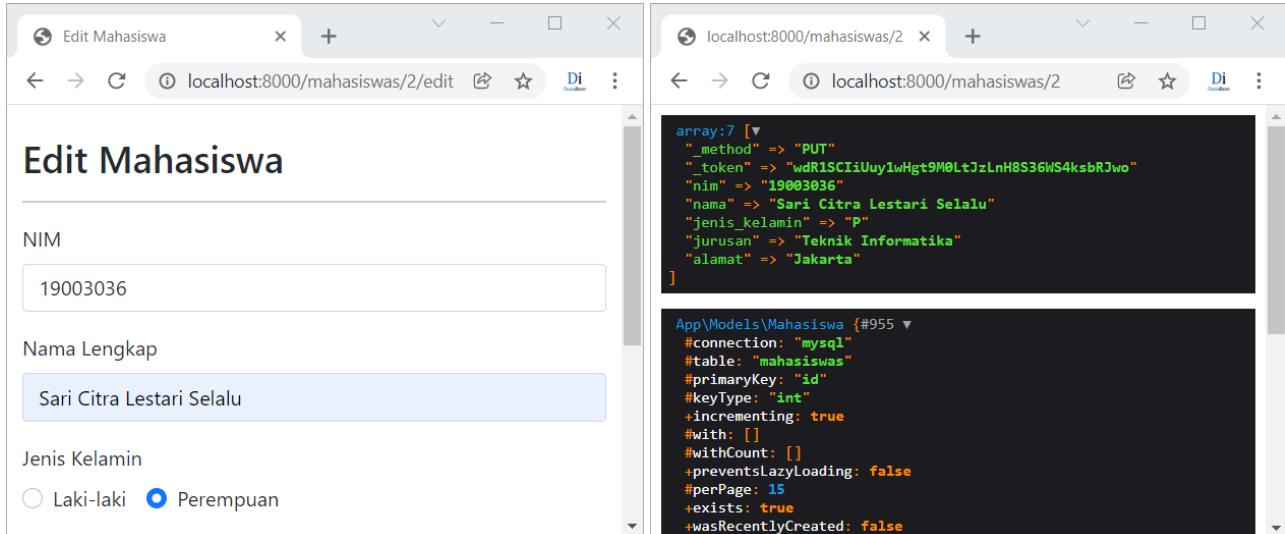
```

Dalam materi pembuatan form update, saya sengaja tidak mengisi atribut `action` karena kita harus membuat dahulu *named route* yang akan dipakai, yakni `{{ route('mahasiswas.update', ['mahasiswa' => $mahasiswa->id]) }}` seperti di baris 2.

Artinya pada saat form update di submit, data form akan dikirim ke *named route* `'mahasiswas.update'`, ikut juga dikirim nilai route parameter `$mahasiswa->id`.

Nilai form ini sebenarnya harus dikirim dengan `method="PUT"`, namun spesifikasi HTTP tidak mendukung method ini. Laravel menggunakan cara khusus untuk mengatasinya, yakni tetap menggunakan `method="POST"` dengan tambahan perintah blade `@method('PUT')` seperti di baris 4.

Simpan perubahan di atas, lalu buka Form Edit, misalnya dari halaman localhost:8000/mahasiswas/2/edit, ubah beberapa data lalu klik tombol Update. Jika semuanya sudah benar, alamat URL di web browser akan pindah ke `localhost:8000/mahasiswas/2` dan tampil hasil `dump()` `$request` dan juga `$mahasiswa`:



Gambar: Hasil proses update data mahasiswa, di form update (kiri), dan setelah di submit (kanan)

Baik, semuanya sesuai dengan rancangan kita. Lanjut dengan membuat kode untuk proses update ke tabel `mahasiswas`. Berikut kode program yang diperlukan:

app/Http/Controllers/MahasiswaController.php

```

1  <?php
2  //...
3  public function update(Request $request, Mahasiswa $mahasiswa)
4  {
5      $validateData = $request->validate([
6          'nim'           => 'required|size:8|unique:mahasiswas',
7          'nama'          => 'required|min:3|max:50',
8          'jenis_kelamin' => 'required|in:P,L',
9          'jurusan'        => 'required',
10         'alamat'        => '',
11     ]);
12
13     Mahasiswa::where('id',$mahasiswa->id)->update($validateData);
14
15     return redirect()->route('mahasiswas.show',[ 'mahasiswa'=>$mahasiswa->id])
16     ->with('pesan',"Update data {$validateData['nama']} berhasil");
17
18 }

```

Seperti biasa, karena data yang di update berasal dari form, kita perlu buat sebuah proses validasi (baris 5 - 11). Validasi ini sama persis seperti yang dipakai pada proses input data (Form Pendaftaran).

Di baris 13 terdapat perintah Eloquent untuk proses update. Caranya, cari terlebih dahulu data yang ingin di update menggunakan method `where('id',$mahasiswa->id)`, lalu setelah data didapat, jalankan proses `update()` dengan nilai yang berasal dari `$validateData`.

Terakhir di baris 15-16 adalah proses `redirect` ke named route '`'mahasiswas.show'`', yakni halaman yang akan menampilkan 1 data mahasiswa (halaman yang kita buat pada pembahasan **Read**).

Karena *named route* 'mahasiswa.show' butuh sebuah parameter id mahasiswa, maka juga harus ditulis array ['mahasiswa' =>\$mahasiswa->id] sebagai argument kedua dari method *route()*.

Saya juga mengirim flash data berupa teks 'Update data {nama mahasiswa} berhasil'. Nantinya untuk menampilkan pesan flash ini kita perlu modifikasi file show.blade.php, tapi untuk sementara ini bisa diabaikan saja.

Simpan method di atas, dan mari uji coba proses update. Buka halaman <localhost:8000/mahasiswa/2/edit>, ubah nama mahasiswa, lalu klik tombol Update:

Gambar: Error karena nim terdeteksi sudah ada di tabel mahasiswa

Ternyata tampil pesan error yang menyatakan bahwa nim sudah ada di dalam tabel mahasiswa. Tentu saja nim tersebut sudah ada karena yang kita lakukan saat ini adalah proses update mahasiswa, bukan menginput data baru. Masalah ini berasal dari syarat validasi untuk inputan nim berikut:

```
'nim' => 'required|size:8|unique:mahasiswa'
```

Error terjadi karena nim yang ada di dalam form tidak lolos syarat validasi `unique:mahasiswa`. Kita bisa saja menghapus syarat ini, tapi itu akan mendatangkan masalah lain karena tidak ada pembatasan nomor nim. Untungnya (dan tentu saja), Laravel sudah menyediakan solusi untuk masalah ini.

Syarat validasi `unique:mahasiswa` bisa menerima pengaturan tambahan agar mengabaikan nim untuk id tertentu. Caranya, tulis nama kolom yang ingin diabaikan beserta **id**-nya, seperti contoh berikut:

```
'nim' => 'required|size:8|unique:mahasiswa,nim,2',
```

Dengan tambahan syarat '`unique:mahasiswa,nim,2`', Laravel akan mengabaikan validasi `unique` untuk kolom `nim` bagi mahasiswa yang ber-id = 2.

Tapi tentu saja tidak bisa ditulis seperti ini karena id mahasiswa yang di update akan terus berganti. Namun kita juga sudah memiliki akses terhadap data **id** ini, yaitu dari `$mahasiswa->id`. Variabel `$mahasiswa` sendiri sudah berisi object **Mahasiswa** karena berasal dari proses *route model binding*.

Dengan demikian, berikut modifikasi dari method `update()`:

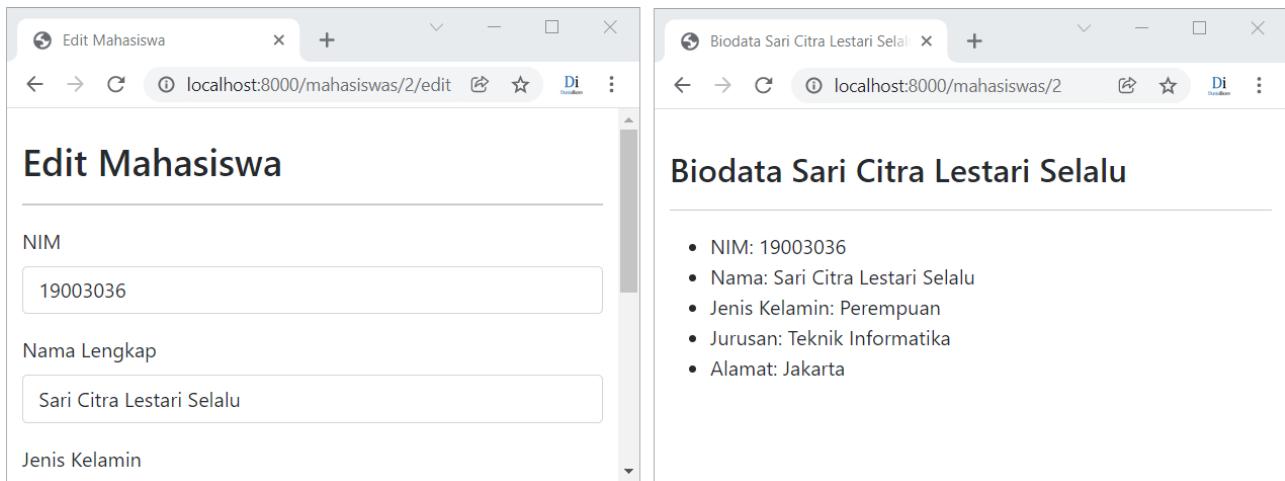
`app/Http/Controllers/MahasiswaController.php`

```

1  <?php
2  //...
3  public function update(Request $request, Mahasiswa $mahasiswa)
4  {
5      $validateData = $request->validate([
6          'nim'          => 'required|size:8|unique:mahasiswas,nim,'.$mahasiswa->id,
7          'nama'         => 'required|min:3|max:50',
8          'jenis_kelamin' => 'required|in:P,L',
9          'jurusan'       => 'required',
10         'alamat'        => '',
11     ]);
12
13     Mahasiswa::where('id',$mahasiswa->id)->update($validateData);
14
15     return redirect()->route('mahasiswas.show',[ 'mahasiswa'=>$mahasiswa->id])
16     ->with('pesan',"Update data {$validateData['nama']} berhasil");
17
18 }

```

Perubahannya ada di baris 6, dimana syarat validasi unique sekarang ditulis sebagai `unique:mahasiswas,nim,'.$mahasiswa->id`. Save file controller dan mari kita coba edit kembali data mahasiswa dengan id = 2 dari halaman localhost:8000/mahasiswas/2/edit:



Gambar: Proses update berhasil, pada saat pengisian form (kiri) dan setelah di update (kanan)

Sip, proses update sudah berhasil.

Syarat validasi unique tetap mengizinkan perubahan kolom `nim`. Misalnya jika nomor `nim` untuk mahasiswa 'Sari Citra Lestari' ingin saya tukar menjadi 19003037, itu bisa dilakukan.

Namun jika nim yang ditukar sudah dimiliki oleh mahasiswa lain, tetap akan keluar pesan error '*The nim has already been taken*'.

Modifikasi berikutnya (atau lebih ke alternatif penulisan), adalah method untuk proses update. Sebelumnya kita menggunakan kode berikut:

```
Mahasiswa::where('id', $mahasiswa->id)->update($validateData);
```

Alternatif cara lain, bisa dengan:

```
$mahasiswa->update($validateData);
```

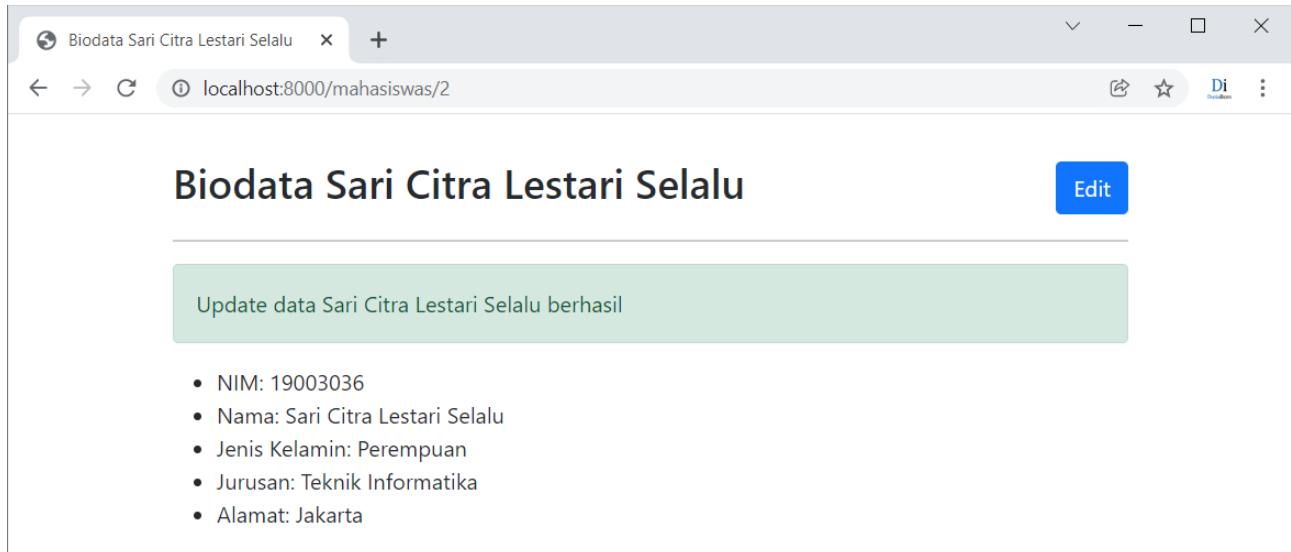
Di sini kita langsung menjalankan proses update dari variabel `$mahasiswa`, tidak perlu lewat class `Mahasiswa`.

Modifikasi File View

Sebelum masuk ke materi tentang **Delete**, saya ingin modifikasi sedikit file `show.index.php`, yakni view yang dipakai untuk menampilkan data untuk 1 mahasiswa. Saya ingin menambah 2 fitur:

1. Menampilkan pesan flash yang berasal dari hasil proses update.
2. Menambah tombol **Edit** yang ketika di klik akan membuka form update.

Hasil akhir yang di inginkan adalah sebagai berikut:



Tampilan pesan flash ada sebelum list dan tombol Edit berada di sisi kanan judul.

Silahkan anda coba edit sebentar file `show.index.php`. Untuk menampilkan pesan flash, cukup copy dari halaman `index.blade.php` karena kodennya mirip. Untuk Tombol **Edit**, juga bisa disamakan dengan kode di `index.blade.php`, namun kali ini isi atribut `href` (tujuan link) harus

menuju route 'mahasiswa.edit'.

Berikut perubahan dari show.blade.php:

resources/views/mahasiswa/show.blade.php

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link href="/css/bootstrap.min.css" rel="stylesheet">
8      <title>Biodata {{$mahasiswa->nama}}</title>
9  </head>
10 <body>
11
12 <div class="container mt-3">
13     <div class="row">
14         <div class="col-12">
15
16             <div class="pt-3 d-flex justify-content-between align-items-center">
17                 <h2>Biodata {{$mahasiswa->nama}}</h2>
18                 <a href="{{ route('mahasiswa.edit',['mahasiswa' => $mahasiswa->id]) }}}"
19                     class="btn btn-primary">Edit</a>
20             </div>
21
22             <hr>
23
24             @if(session()->has('pesan'))
25                 <div class="alert alert-success" role="alert">
26                     {{ session()->get('pesan') }}
27                 </div>
28             @endif
29
30             <ul>
31                 <li>NIM: {{$mahasiswa->nim}} </li>
32                 <li>Nama: {{$mahasiswa->nama}} </li>
33                 <li>Jenis Kelamin:
34                     {{$mahasiswa->jenis_kelamin == 'P' ? 'Perempuan' : 'Laki-laki'}}</li>
35                 <li>Jurusan: {{$mahasiswa->jurusan}} </li>
36                 <li>Alamat:
37                     {{$mahasiswa->alamat == '' ? 'N/A' : $mahasiswa->alamat}}</li>
38             </ul>
39
40         </div>
41     </div>
42 </div>
43 </div>
44 </div>
45
46 </body>
47 </html>
```

Baris 16 – 20 adalah kode untuk menampilkan judul serta tombol ke Form Edit. Kemudian diikuti baris 24 – 28 untuk menampilkan pesan flash.

19.8. Delete

Proses CRUD terakhir yang akan kita bahas adalah **Delete**, yakni penghapusan data dari database. Berdasarkan tabel REST, ini dilakukan oleh route berikut:

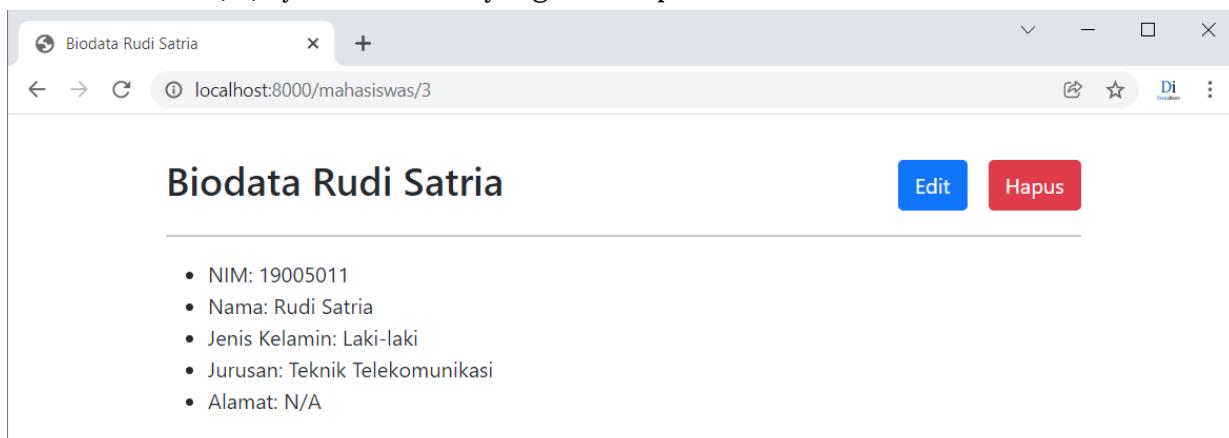
routes/web.php

```
Route::delete('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'destroy'])
->name('mahasiswas.destroy');
```

Silahkan tambah route ini ke dalam file routes\web.php. Alamat URL route delete terdiri dari 2 segmen: /mahasiswas/, dan {mahasiswa} yang dipakai untuk menampung **id** mahasiswa. Terlihat method yang digunakan adalah **delete**.

Karena tidak menggunakan method **get**, maka route ini tidak bisa kita akses dengan sekedar menulis alamat URL, tapi harus dari sebuah form. Di dalam form inilah nantinya terdapat tombol "delete" untuk proses penghapusan. Ketika tombol di klik, data mahasiswa akan dikirim ke method **destroy()** di **MahasiswaController**.

Di manakah tombol **Delete** ini harus ditempatkan? Tidak ada aturan baku karena lebih ke desain tampilan. Bisa saja tombol delete berada di halaman **index.blade.php**, pada di sisi kanan setiap baris data mahasiswa. Atau untuk praktik kali ini saya akan menempatkannya ke view **show.blade.php**, yakni halaman yang menampilkan data dari 1 mahasiswa:



Gambar: Tombol Hapus di view show.blade.php

Terdapat tambahan tombol **Hapus** di sisi kanan atas setelah tombol Edit. Berikut kode untuk membuat tombol ini:

resources/views/mahasiswa/show.blade.php

```
1  <!DOCTYPE html>
2  ...
```

```

3 <div class="pt-3 d-flex justify-content-between align-items-center">
4   <h2>Biodata {$mahasiswa->nama}</h2>
5
6   <div class="d-flex">
7     <a href="{{ route('mahasiswas.edit',['mahasiswa' => $mahasiswa->id]) }}">
8       class="btn btn-primary">Edit</a>
9     <form method="POST" action="{{ route('mahasiswas.destroy',
10        ['mahasiswa' => $mahasiswa->id]) }}">
11       @method('DELETE')
12       @csrf
13       <button type="submit" class="btn btn-danger ms-3">Hapus</button>
14     </form>
15   </div>
16
17 </div>
18 ...

```

Seperti yang telah di singgung sebelumnya, proses penghapusan data dilakukan dengan method **delete** yang dikirim dari sebuah form, tidak bisa langsung dari tombol biasa.

Namun sama seperti **patch**, HTTP juga tidak mendukung method **delete**. Sehingga caranya adalah, tetap kirim form menggunakan `method="POST"`, lalu tambah perintah blade `@method('DELETE')` ke dalam form seperti di baris 11. Tidak lupa perintah `@csrf` untuk pembuatan CSRF token. Form ini akan dikirim ke *named route* `'mahasiswas.destroy'` dengan menyertakan id mahasiswa yang ingin dihapus.

Isi form sendiri hanya 1 element, yakni tombol submit dari tag `<button type="submit">`. Pada saat tombol ini di klik, data mahasiswa akan dikirim ke method `destroy()` di `Mahasiswa Controller` untuk proses penghapusan. Berikut kode program dari method ini:

app/Http/Controllers/MahasiswaController.php

```

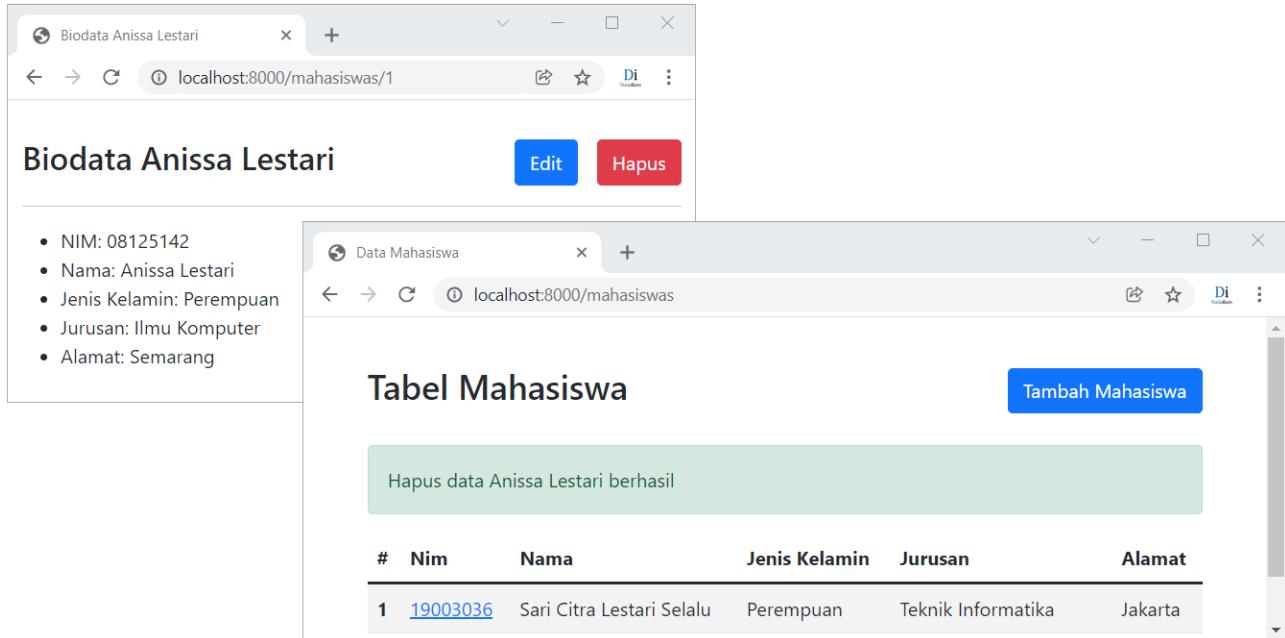
1 <?php
2 /**
3  public function destroy(Mahasiswa $mahasiswa)
4  {
5    $mahasiswa->delete();
6    return redirect()->route('mahasiswas.index')
7      ->with('pesan','Hapus data $mahasiswa->nama berhasil');
8  }

```

Method `destroy()` menerima sebuah argument `$mahasiswa`, yang dengan teknik *route model binding*, akan berisi object dari 1 mahasiswa.

Proses penghapusan di lakukan di baris 5, yakni dengan perintah `$mahasiswa->delete()`. Setelah itu di baris 6 terdapat perintah redirect ke route `mahasiswas.index`, beserta pesan "Hapus data `$mahasiswa->nama berhasil`".

Save file `MahasiswaController`, dan coba buka halaman localhost:8000/mahasiswas/1. Ini akan menampilkan data mahasiswa yang memiliki id = 1. Kemudian klik tombol **Hapus**:



Gambar: Proses penghapusan data mahasiswa

Jika tidak ada masalah, halaman akan redirect ke `localhost:8000/mahasiswa` dan tampil pesan flash: "*Hapus data {nama mahasiswa} berhasil*". Untuk memastikan, kita bisa cek langsung ke database, seharusnya data mahasiswa dengan id = 1 sudah tidak ada.

Untuk proses penghapusan data seperti ini, sebenarnya akan lebih user friendly jika ketika tombol Hapus di klik, tampilkan terlebih dahulu jendela popup untuk konfirmasi. Karena tentu saja user akan sangat jengkel jika tidak sengaja ter-klik tombol Hapus. Namun untuk membatasi agar kode program kita tidak terlalu kompleks, tampilan CRUD saya batasi sampai di sini.

19.9. Resource Controllers

Apa yang sudah di buat sepanjang bab ini telah menerapkan **Resource Controllers** dari Laravel, dimana kita merancang route sesuai prinsip **RESTfull**. Berikut ke-7 route tersebut:

routes/web.php

```

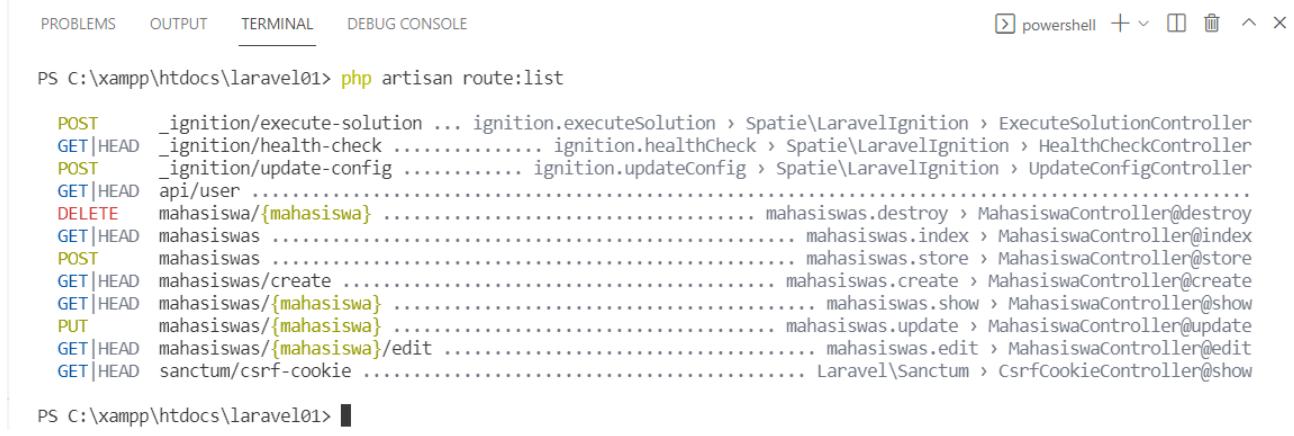
1 <?php
2 ...
3
4 Route::get('/mahasiswa', [MahasiswaController::class, 'index'])
5 ->name('mahasiswa.index');
6
7 Route::get('/mahasiswa/create', [MahasiswaController::class, 'create'])
8 ->name('mahasiswa.create');
9
10 Route::post('/mahasiswa', [MahasiswaController::class, 'store'])
11 ->name('mahasiswa.store');
12
13 Route::get('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'show'])

```

CRUD

```
14 ->name('mahasiswa.show');
15
16 Route::get('/mahasiswa/{mahasiswa}/edit', [MahasiswaController::class, 'edit'])
17 ->name('mahasiswa.edit');
18
19 Route::put('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'update'])
20 ->name('mahasiswa.update');
21
22 Route::delete('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'destroy'])
23 ->name('mahasiswa.destroy');
```

Daftar route ini juga bisa di lihat dari perintah `php artisan route:list`:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
powershell + × ^ ×

PS C:\xampp\htdocs\laravel01> php artisan route:list

POST _ignition/execute-solution ... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user ..... mahasiswa.destroy > MahasiswaController@destroy
DELETE mahasiswa/{mahasiswa} ..... mahasiswa.index > MahasiswaController@index
GET|HEAD mahasiswa ..... mahasiswa.store > MahasiswaController@store
POST mahasiswa ..... mahasiswa.create > MahasiswaController@create
GET|HEAD mahasiswa/{mahasiswa} ..... mahasiswa.show > MahasiswaController@show
PUT mahasiswa/{mahasiswa} ..... mahasiswa.update > MahasiswaController@update
GET|HEAD mahasiswa/{mahasiswa}/edit ..... mahasiswa.edit > MahasiswaController@edit
GET|HEAD sanctum/csrf-cookie ..... Laravel\Sanctum > CsrfCookieController@show

PS C:\xampp\htdocs\laravel01>
```

Gambar: Daftar route Laravel dilihat dari terminal VS Code

Jika kita membuat CRUD mengikuti tabel REST, maka penulisan routenya akan selalu sama, yang berbeda hanya di nama model saja. Sebagai contoh, jika saya ingin membuat aplikasi CRUD untuk tabel barang, maka penulisan routenya adalah:

```
1 Route::get('/barang', [BarangController::class, 'index'])
2 ->name('barang.index');
3
4 Route::get('/barang/create', [BarangController::class, 'create'])
5 ->name('barang.create');
6
7 ...
```

Saya yakin anda sudah bisa melanjutkan route untuk proses `store`, `show`, `edit`, `update` dan `destroy`. Namun Laravel ingin 'memanjakan' penggunanya dengan menyediakan 1 perintah untuk semua route, yakni dengan menulis:

```
Route::resource('barang', BarangController::class);
```

Dengan perintah `Route::resource`, maka ke-7 route untuk proses CRUD langsung tersedia.

Sebagai contoh praktik, silahkan hapus semua route yang sudah kita tulis di file `route.php`, lalu ganti dengan kode berikut:

CRUD

routes/web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::resource('mahasiswas', MahasiswaController::class);
```

Kemudian buka halaman <localhost:8000/mahasiswas/>, maka halaman index dari CRUD mahasiswa tetap tampil normal, termasuk semua tombol untuk proses create, store, show, edit, update dan destroy.

Untuk memastikan routenya tetap sama, jalankan kembali perintah `php artisan route:list`:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
powershell + ×

PS C:\xampp\htdocs\laravel01> php artisan route:list
POST   _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController...
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST   _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user .....
GET|HEAD mahasiswas ..... mahasiswas.index > MahasiswaController@index
POST   mahasiswas ..... mahasiswas.store > MahasiswaController@store
GET|HEAD mahasiswas/create ..... mahasiswas.create > MahasiswaController@create
GET|HEAD mahasiswas/{mahasiswa} ..... mahasiswas.show > MahasiswaController@show
PUT|PATCH mahasiswas/{mahasiswa} ..... mahasiswas.update > MahasiswaController@update
DELETE  mahasiswas/{mahasiswa} ..... mahasiswas.destroy > MahasiswaController@destroy
GET|HEAD mahasiswas/{mahasiswa}/edit ..... mahasiswas.edit > MahasiswaController@edit
GET|HEAD sanctum/csrf-cookie .....
Laravel\Sanctum > CsrfCookieController@show
```

Gambar: Daftar route Laravel hasil Route::resource

Nyaris tidak ada perbedaan dengan route yang ditulis manual. Yang berubah hanya kolom method untuk proses update dimana sekarang mendukung 2 method yakni PUT dan PATCH.

Tidak hanya itu, Laravel juga bisa meng-generate struktur Controller untuk semua method yang diperlukan dalam membuat CRUD. Caranya, tambah option `-r` atau `--resource` ketika membuat model. Sebagai contoh, saya ingin membuat CRUD untuk menyimpan data **barang**. Maka perintah yang bisa dipakai adalah:

```
php artisan make:model Barang -mcr
```

```
Laravel01 Project
C:\xampp\htdocs\laravel01>php artisan make:model Barang -mcr
Model created successfully.
Created Migration: 2022_01_25_143654_create_barangs_table
Controller created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Membuat file resources untuk barang

Perintah ini akan membuat file migration, model, dan controller sekaligus. Selain itu, ketika file `BarangController.php` di buka, sudah berisi struktur method CRUD:

CRUD

app/Http/Controllers/BarangController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\Barang;
6 use Illuminate\Http\Request;
7
8 class BarangController extends Controller
9 {
10     public function index()
11     {
12         //
13     }
14
15     public function create()
16     {
17         //
18     }
19
20     public function store(Request $request)
21     {
22         //
23     }
24
25     public function show(Barang $barang)
26     {
27         //
28     }
29
30     public function edit(Barang $barang)
31     {
32         //
33     }
34
35     public function update(Request $request, Barang $barang)
36     {
37         //
38     }
39
40     public function destroy(Barang $barang)
41     {
42         //
43     }
44 }
```

Semua method sudah tersedia dan tinggal diisi. Ini sangat memudahkan kita dalam pembuatan aplikasi web.

Dalam bab ini kita telah membahas cukup detail salah satu materi terpenting di aplikasi web, yakni proses CRUD. Selain itu kita juga menerapkan konsep **RESTfull Controller** yang disarankan Laravel dan menjadi standar di dunia web development.

Jika anda bisa memahami semua materi dalam bab ini, proses pembuatan CRUD tidak akan butuh waktu lama, malah mayoritas waktu kita akan lebih banyak terpakai untuk merancang tampilan / design form, tidak lagi memikirkan kode PHP untuk memproses CRUD tersebut.

Isi method di controller juga hanya beberapa baris karena sebagian besar sudah di kelola oleh kode bawaan Laravel melalui **Request object**, **Eloquent**, **Form Validation**, dan tidak lupa **route model binding**. Sangat praktis!

Lanjut, materi berikutnya akan membahas tentang pembuatan form upload file di Laravel.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

20. File Upload

Upload file merupakan salah fitur umum yang sering kita jumpai. Misalnya untuk form registrasi, tidak jarang juga perlu mengupload foto profil, upload berkas pdf, dsb. Laravel sudah menyediakan fitur untuk memproses file upload, termasuk bagaimana mengelola lokasi penempatan file tersebut.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css**, karena kita akan memakai sedikit kode Bootstrap.

20.1. Persiapan Awal

Persiapan awal ini mencakup pembuatan beberapa file yang diperlukan sebagai bahan praktek. Pertama, kita butuh sebuah controller:

```
php artisan make:controller FileUploadController
```

```
C:\xampp\htdocs\laravel01>php artisan make:controller FileUploadController
Controller created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan perintah php artisan

FileUploadController nantinya berisi beberapa method. Untuk saat ini saya akan buat 2 buah method: **fileUpload()** dan **prosesFileUpload()**. Berikut kode program yang diperlukan:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
```

File Upload

```
4 use Illuminate\Http\Request;
5
6 class FileUploadController extends Controller
7 {
8     public function fileUpload(){
9         return view('file-upload');
10    }
11
12
13    public function prosesFileUpload(Request $request){
14        return "Pemrosesan file upload di sini";
15    }
16 }
```

Method `fileUpload()` berisi kode program yang akan memanggil file view `file-upload.blade.php`. File blade inilah yang berisi kode HTML dan CSS untuk membuat form (akan kita buat sesaat lagi).

Sedangkan method `prosesFileUpload()` saya siapkan untuk memproses hasil submit form. Di sinilah kita akan banyak membuat kode program, diantaranya validasi form dan serta proses pemindahan file yang sudah di upload. Di bagian argument terdapat variabel `$request` yang akan berisi **Request** object.

Lanjut, kita perlu 2 buah route:

`routes/web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\FileUploadController;
5
6 Route::get('/file-upload', [FileUploadController::class, 'fileUpload']);
7 Route::post('/file-upload', [FileUploadController::class, 'prosesfileUpload']);
```

Route pertama dipakai untuk menampilkan form, yakni dengan cara mengakses method `fileUpload()` yang ada di `FileUploadController`. Alamat URLnya adalah `'/file-upload'`.

Route kedua berfungsi untuk pemrosesan form, yang akan mengakses method `prosesfileUpload()` di `FileUploadController`. Route ini menggunakan method **post** yang menjadi tujuan proses submit form.

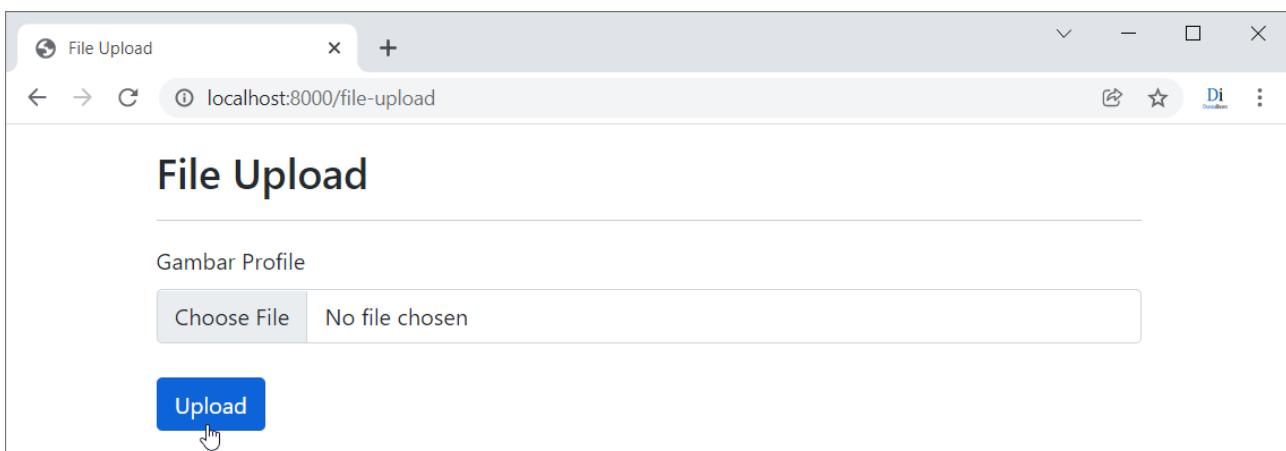
File terakhir yang kita perlukan adalah file view `file-upload.blade.php` dengan kode sebagai berikut:

`resources/views/file-upload.blade.php`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

File Upload

```
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <link href="/css/bootstrap.min.css" rel="stylesheet">
8   <title>File Upload</title>
9   </head>
10  <body>
11
12  <div class="container mt-3">
13    <h2>File Upload</h2>
14    <hr>
15
16    <form action="{{url('/file-upload')}}" method="POST"
17      enctype="multipart/form-data">
18      @csrf
19
20      <div class="mb-3">
21        <label for="berkas" class="form-label">Gambar Profile</label>
22        <input type="file" class="form-control" id="berkas" name="berkas">
23        @error('berkas')
24          <div class="text-danger">{{ $message }}</div>
25        @enderror
26      </div>
27
28      <button type="submit" class="btn btn-primary my-2">Upload</button>
29    </form>
30  </div>
31
32  </body>
33 </html>
```



Gambar: Tampilan form file upload

Untuk membuat tampilan form, saya kembali menggunakan sedikit class CSS bawaan Bootstrap. Karena kita akan mengupload file, maka di dalam tag `<form>` harus ditambah atribut `enctype="multipart/form-data"` seperti di baris 17. Form ini dikirim menggunakan method POST ke `url('/file-upload')`.

Seperti biasa, form yang dikirim menggunakan method post juga harus menyertakan perintah `@csrf`. Kode ini ada di baris 18.

File Upload

Form ini hanya terdiri dari 1 inputan, yakni <input type="file" name="berkas">. Kemudian terdapat blok @error('berkas') untuk menampilkan pesan error validasi di baris 23-25. Terakhir, tombol submit saya isi dengan nama "**Upload**".

Langsung saja kita coba test. Silahkan upload sembarang file, lalu klik tombol "Upload":



Gambar: Hasil Upload

Jika tampil teks "*Pemrosesan file upload di sini*", maka artinya semua file persiapan kita sudah selesai.

20.2. Informasi File Upload

Informasi seputar file yang sudah di upload bisa kita akses melalui **Request** object, yakni variabel \$request. Silahkan isi kode berikut ke dalam method prosesfileUpload() di **FileUploadController**:

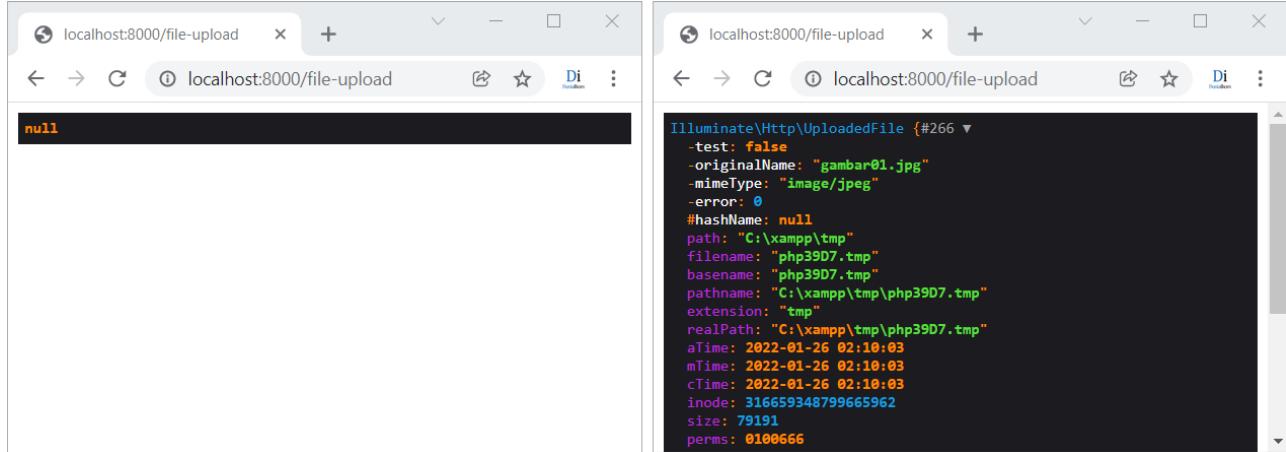
app/Http/Controllers/FileUploadController.php

```
1  <?php
2  ...
3  public function prosesFileUpload(Request $request){
4      dump($request->berkas);
5  }
```

Isi dari method ini hanya 1 baris, dimana saya langsung men-dump \$request->berkas. Nama "berkas" ini berasal dari nilai atribut name pada tag <input type="file" name="berkas">.

Mari lakukan 2 percobaan, pertama langsung submit form tanpa mengisi file apapun. Dan kedua, upload file sembarang dan submit. Berikut hasilnya:

File Upload



Gambar: Hasil submit form upload, tanpa file (kiri) dan dengan file (kanan)

Jika tidak ada file yang di upload, perintah `$request->berkas` akan mengembalikan nilai `NULL`. Namun jika ada file yang di upload, akan menampilkan beragam informasi mengenai file tersebut. Diantaranya nama file, `mimetype`, tanggal upload serta ukuran file yang di upload.

Untuk memeriksa apakah terdapat sebuah file yang di upload, bisa menggunakan perintah `$request->hasFile('berkas')`. Hasilnya `true` jika ada file yang di upload dan `false` jika tidak ada file yang di upload.

Berikut cara mengambil beberapa info dari file yang di upload:

app/Http/Controllers/FileUploadController.php

```
1  <?php
2  ...
3  public function prosesFileUpload(Request $request){
4
5      if($request->hasFile('berkas'))
6      {
7          echo "path(): ".$request->berkas->path();
8          echo "<br>";
9          echo "extension(): ".$request->berkas->extension();
10         echo "<br>";
11         echo "getClientOriginalExtension(): ".
12             $request->berkas->getClientOriginalExtension();
13         echo "<br>";
14         echo "getMimeType(): ".$request->berkas->getMimeType();
15         echo "<br>";
16         echo "getClientOriginalName(): ".
17             $request->berkas->getClientOriginalName();
18         echo "<br>";
19         echo "getSize(): ".$request->berkas->getSize();
20     }
21     else
22     {
23         echo "Tidak ada berkas yang diupload";
24     }
25 }
```



Gambar: Informasi dari file yang di upload

Di baris 5 terdapat sebuah kondisi if yang akan dijalankan jika `$request->hasFile('berkas')` bernilai **true**. Di dalam blok ini, saya mengakses beberapa method. File sample yang saya upload adalah sebuah file gambar bernama `gambar01.jpg`, berukuran sekitar 78kb. Berikut penjelasan dari method yang ada di baris 7 – 19:

- `$request->berkas->path()`, menampilkan alamat path dari file yang sudah di upload. Perhatikan bahwa isinya adalah alamat *temporary* dari pengaturan PHP. Dalam contoh ini, alamat file yang saya upload akan disimpan di `C:\xampp\tmp\phpD41.tmp`.
- `$request->berkas->extension()`, menampilkan extension file. Dalam contoh ini file yang saya upload bernama `gambar01.jpg`, sehingga extensionnya adalah `jpg`.
- `$request->berkas->getClientOriginalExtension()`, menampilkan extension yang diambil dari nama file. Karena file yang di upload adalah `gambar01.jpg`, maka hasil method ini adalah `jpg`.
- `$request->berkas->getMimeType()`, menampilkan mimetype dari file yang di upload. Dalam contoh ini hasilnya `image/jpeg`. Mimetype sendiri adalah sejenis standar daftar jenis file yang ditulis dalam format "type/subtype". Dalam hal ini, file `gambar01.jpg` bertipe "image", dan subtype "jpeg". Lebih lanjut tentang mimetype bisa dibaca-baca ke: [MIME types \(IANA media types\)](#).
- `$request->berkas->getClientOriginalName()`, menampilkan nama asli dari file yang di upload, yang dalam contoh ini berupa "`gambar01.jpg`".
- `$request->berkas->getSize()`, menampilkan ukuran file yang di upload (dalam satuan byte). Dalam contoh ini, file `gambar01.jpg` yang saya upload berukuran 79191 byte, atau 77,33 kilobyte (1 kilobyte = 1024 byte).

Inilah beberapa informasi yang bisa kita ambil dari file yang telah di upload.

20.3. Validasi File Upload

File yang di upload tentunya butuh proses validasi. Malah ini menjadi lebih penting karena

File Upload

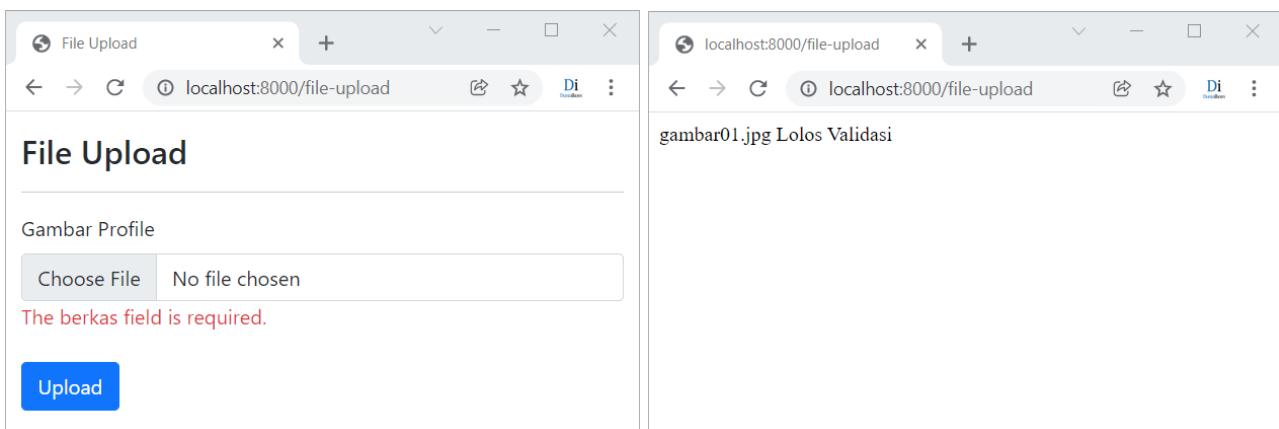
tentu kita tidak ingin ada yang mengupload file berbahaya, atau malah mengupload file .php yang berisi kode/script tertentu.

Membuat validasi file upload mirip seperti cara membuat validasi inputan form biasa, yakni menggunakan method `$request->validate()`. Berikut contoh penggunaannya:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas' => 'required',
6         ]);
7
8         echo $request->berkas->getClientOriginalName()." Lelos Validasi";
9     }
```

Di sini saya membuat syarat `required` untuk file berkas, sehingga akan tampil pesan error jika form di submit tanpa meng-upload sebuah file:



Gambar: Validasi form, jika tidak menyertakan file (kiri) dan lolos validasi (kanan)

Jika syarat validasi tidak terpenuhi, akan langsung di redirect untuk menampilkan pesan error. Di halaman view `file-upload.blade.php` kita sudah menyiapkan perintah `@error('berkas')` untuk menampilkan pesan error ini.

Syarat validasi selanjutnya adalah membatasi jenis file dan maksimal ukuran file:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas' => 'required|file|image|max:5000',
6         ]);
7
8         echo $request->berkas->getClientOriginalName()." Lelos Validasi";
9     }
```

Berikut penjelasan dari syarat validasi ini:

- **required**: inputan form tidak boleh kosong.
- **file**: memastikan bahwa file sudah berhasil di upload.
- **image**: file yang di upload harus file image (gambar), yakni salah satu dari jpeg, png, bmp, gif, svg, atau webp.
- **max:5000**, artinya file yang di upload berukuran maksimal 5000 byte atau sekitar 5 MB.

Yang perlu sedikit penjelasan adalah tentang syarat **max:5000**. Meskipun ini artinya file dengan ukuran 5MB bisa di upload, tapi ini masih dibatasi oleh pengaturan **upload_max_filesize** di file **php.ini**.

Di versi XAMPP 8.1.1 yang saya pakai, batasan **upload_max_filesize** ada di 40MB. Nilai ini bisa saja lebih kecil terutama di XAMPP versi lama.

```

841 ; Temporary directory for HTTP uploaded files (will use system default if not
842 ; specified).
843 ; https://php.net/upload-tmp-dir
844 upload_tmp_dir="C:\xampp\tmp"
845
846 ; Maximum allowed size for uploaded files.
847 ; https://php.net/upload-max-filesize
848 upload_max_filesize=40M
849
850 ; Maximum number of files that can be uploaded via a single request
851 max_file_uploads=20
852

```

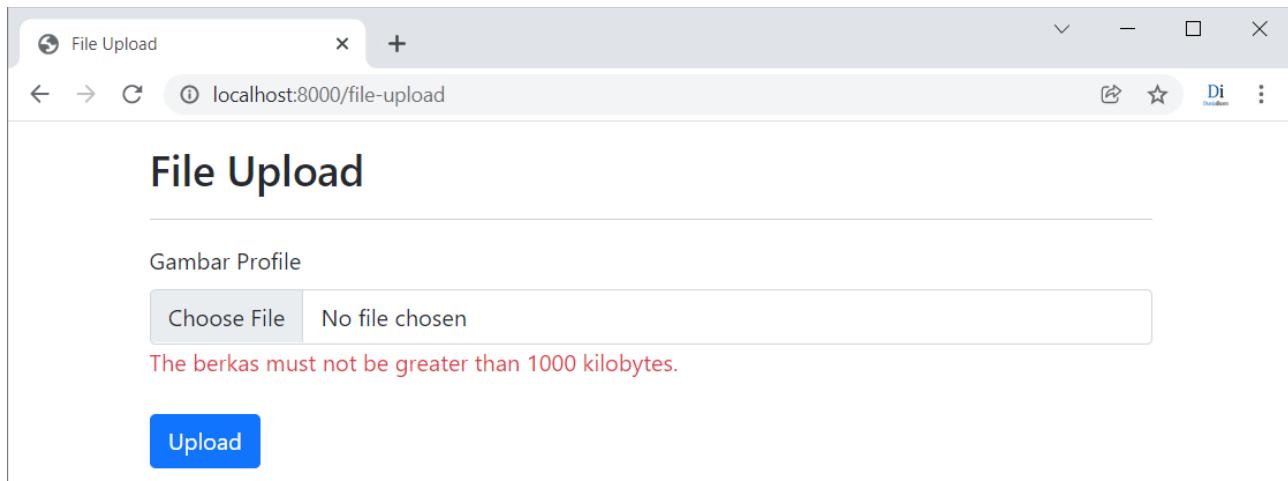
Gambar: Batasan upload_max_filesize di php.ini

Jika anda ingin mengupload file yang lebih besar dari 40MB, pengaturan ini juga harus diubah.

Sebenarnya masih ada pengaturan PHP lain terkait proses upload file, seperti **post_max_size**. Mengenai hal ini sudah pernah kita bahas di buku **PHP Uncover**.

Untuk contoh kali ini, saya akan ubah syarat validasi menjadi 1MB, dan test meng-upload file berukuran 1,3MB:

```
'berkas' => 'required|file|image|max:1000',
```



Gambar: Gagal upload karena batasan ukuran file dari validasi

Hasilnya, tampil pesan error: "The berkas may not be greater than 1000 kilobytes"

Laravel memang menyediakan validasi khusus untuk file gambar (image), namun bagaimana untuk file lain seperti pdf? Caranya, terdapat syarat validasi mimes, yang bisa diisi dengan jenis mimetipe yang kita inginkan. Sebagai contoh, untuk membatasi agar file yang diupload hanya boleh bertipe pdf saja, bisa gunakan syarat berikut:

```
'berkas' => 'required|file|mimes:pdf|max:1000',
```

Atau jika ingin untuk file pdf dan juga png, bisa ditulis sebagai berikut:

```
'berkas' => 'required|file|mimes:pdf,png|max:1000',
```

Berdasarkan dokumentasi Laravel, proses pemeriksaan mimetipe ini tidak hanya membaca extension file saja, tapi juga melihat isi file tersebut. Ini bisa memproteksi perubahan file misalnya file doc.exe yang diubah menjadi doc.png.

Daftar mimetipe bisa dilihat ke: [Media Types](#), atau versi yang lebih mudah dibaca ke: [Common MIME types](#).

20.4. Memindahkan File Upload

Semua file yang diupload akan tersimpan sementara ke folder *temporary* yang dalam PHP bawaan XAMPP berada di C:\xampp\tmp\. Agar bisa diakses, file ini harus dipindah ke folder aplikasi kita. Laravel menyediakan beragam cara untuk memindahkan file ini, diantaranya method `store()`, `storeAs()`, dan `move()`. Kita akan bahas secara bertahap.

Cara pertama adalah menggunakan method `store()` yang bisa diakses dari **Request** object. Berikut contoh penggunaannya:

File Upload

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas' => 'required|file|image|max:1000',
6         ]);
7
8         $path = $request->berkas->store('uploads');
9         echo "Proses upload berhasil, file berada di: $path";
10    }
```

Di baris 4-6 terdapat kode untuk proses validasi yang sudah di bahas sebelumnya.

Proses pemindahan file dilakukan di baris 8, yakni dengan perintah `$request->berkas->store('uploads')`. Method `store()` akan mengambil file upload dari folder temporary dan memindahkannya ke folder `storage\app\uploads` di aplikasi Laravel.

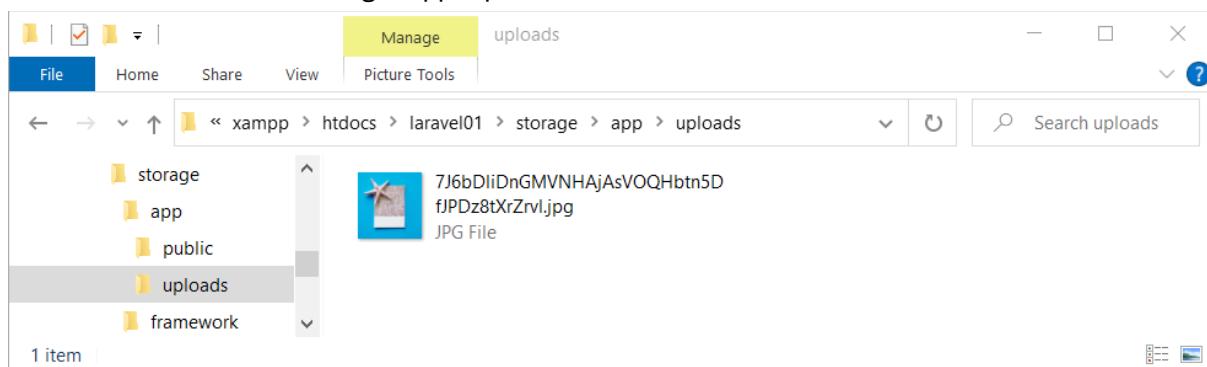
Method `store()` ini butuh sebuah argument berupa nama folder, sehingga perintah `$request->berkas->store('uploads')` akan memindahkan file upload ke folder `storage\app\uploads`. Nilai kembalian dari method ini berupa alamat path dari file akhir, yang dalam contoh ini saya simpan ke dalam variabel `$path`.

Mari kita coba, silahkan upload sebuah file gambar dengan ukuran kurang dari 1MB:



Gambar: Proses upload berhasil

Setelah itu buka folder `storage\app\uploads\` untuk melihat file ini:



Gambar: Lokasi penyimpanan file upload

Sip, file yang di upload sudah bisa diakses. Namun kenapa nama file acak seperti itu?

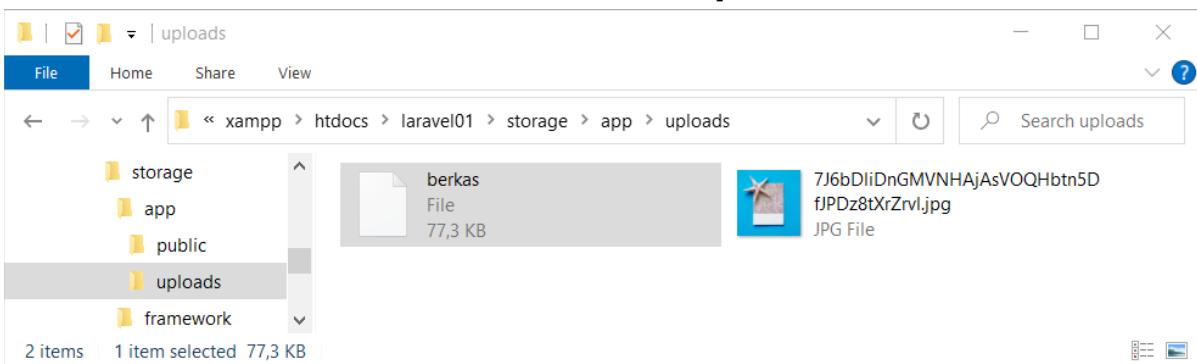
Method `store()` memang akan men-generate nama acak untuk setiap file yang di upload. Kesannya memang agak aneh, tapi sebenarnya sangat bermanfaat:

- Menghindari kemungkinan file ditimpas. Jika nama file yang sudah di upload sama dengan file asal, bisa saja di kemudian hari ada yang mengupload file dengan nama yang sama. Jika ini terjadi, maka file yang baru akan menimpa file lama.
- Menghindari error karena nama file mengandung spasi. Di dalam penulisan alamat *path*, karakter spasi ini sering menjadi masalah.

Bagaimana jika kita tetap ingin membuat nama file sendiri? Tidak masalah, Laravel menyediakan method `storeAs()` untuk keperluan ini. Berikut contoh penggunaannya:

```
$path = $request->berkas->storeAs('uploads', 'berkas');
```

Method `storeAs()` butuh 2 buah argument. Argument pertama berupa nama folder, dan argument kedua berupa nama file yang ingin di buat. Jika kita meng-upload file dengan perintah di atas, nama file akhir adalah "berkas", dan tanpa extension!



Gambar: File bernama "berkas"

Sehingga kita harus menambah sendiri extension file ini. Selain itu, nama file tidak bisa di tulis langsung seperti ini karena akan terus tertimpa oleh file lain karena sama-sama bernama "berkas".

Salah satu solusi adalah dengan mengambil nama file dari fungsi `getClientOriginalName()` seperti contoh berikut:

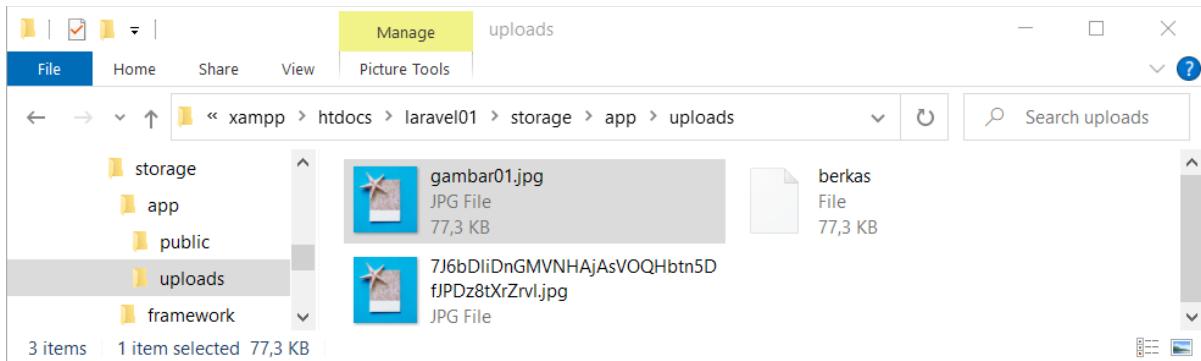
app/Http/Controllers/FileUploadController.php

```
1  <?php
2  ...
3      public function prosesFileUpload(Request $request){
4          $request->validate([
5              'berkas'  => 'required|file|image|max:1000',
6          ]);
7
8          $namaFile = $request->berkas->getClientOriginalName();
9          $path = $request->berkas->storeAs('uploads', $namaFile);
10
11         echo "Proses upload berhasil, file berada di: $path";
12     }
```

File Upload

Di baris 8 saya mengambil nama file asal dengan perintah `$request->berkas->getClientOriginalName()`, yang hasilnya ditampung oleh variabel `$namaFile`. Variabel `$namaFile` ini kemudian diinput sebagai argument kedua dari method `storeAs()`.

Dengan cara ini maka file yang di upload akan memiliki nama yang sama dengan file asal:



Gambar: Nama file upload sama dengan nama file asal

Namun terdapat kelemahan dengan teknik seperti ini. Jika ada yang meng-upload file dengan nama yang sama, maka file pertama akan tertimpa. Selain itu bisa timbul masalah jika nama file yang di upload mengandung karakter spasi.

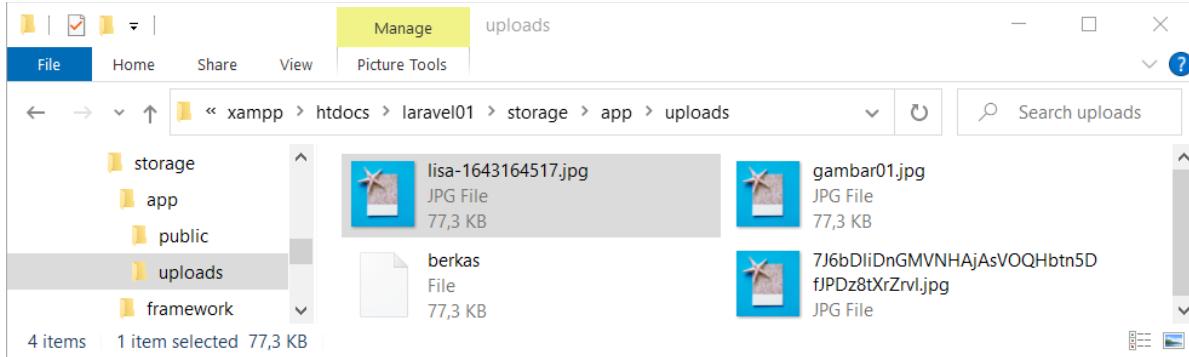
Cara lain adalah dengan meng-generate nama acak sendiri. Sebagai contoh, saya akan membuat nama file upload berdasarkan nama user, ditambah dengan digit acak dari fungsi `time()`. Berikut kode programnya:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas' => 'required|file|image|max:1000',
6         ]);
7
8         $extFile = $request->berkas->getClientOriginalExtension();
9         $namaFile = 'lisa-' . time() . '.' . $extFile;
10        $path = $request->berkas->storeAs('uploads', $namaFile);
11
12        echo "Proses upload berhasil, file berada di: $path";
13    }
```

Di baris 8, method `$request->berkas->getClientOriginalExtension()` di pakai untuk mengambil extension file asal. Kemudian di baris 9 saya menyambung 3 buah string, yakni 'lisa-' yang diibaratkan sebagai nama user, hasil timestamp dari fungsi `time()` bawaan PHP, serta extension file yang berasal dari variabel `$extFile`.

File Upload



Gambar: Nama file upload di-generate manual

Hasilnya, file yang di upload bernama `lisa-1574953613.jpg`. Nama user "lisa" ini nantinya bisa berasal dari database, yang akan berbeda-beda sesuai dengan id login.

Nama ini relatif tidak bermasalah, kecuali user tersebut meng-upload beberapa file dalam detik yang sama. Agar lebih aman lagi (supaya tidak ada nama file yang bentrok), bisa ditambah dengan karakter acak lain seperti hasil fungsi `hash md5()`.

Atau daripada repot, bisa pakai method `store()` saja supaya Laravel yang langsung meng-generate nama file secara otomatis.

20.5. Membuat Symlink

File yang kita upload sudah bisa diakses, tapi baru secara manual menggunakan Windows Explorer. File tersebut belum bisa dibuka dari halaman web karena secara default (dan memang sudah seharusnya), Laravel hanya mengizinkan web browser mengakses file yang ada di folder `public` saja.

Sebenarnya bisa saja file upload langsung disimpan ke folder `public`, tapi Laravel memilih solusi yang lebih elegan, yakni menggunakan metode yang disebut sebagai *symbolic link*, atau sering disingkat sebagai *symlink*.

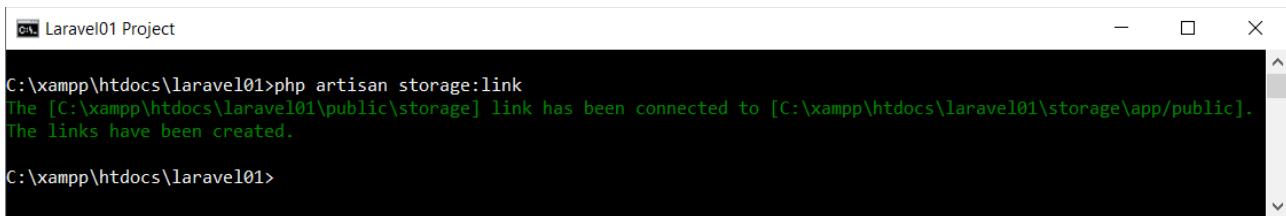
Symlink mirip seperti shortcut tapi seolah-olah mengcopy isi satu folder ke folder lain (*mirroring*). Pada saat ini, semua file yang sudah di upload berada di folder `storage\app\`. Kita bisa membuat symlink agar file ini juga bisa diakses dari folder `public\`.

Istilah *symlink* lebih populer di Linux, karena ada beberapa pengaturan Linux yang butuh *symlink*. Di OS Windows, fitur ini relatif jarang dipakai meskipun juga tersedia.

Untuk membuat *symlink* yang menghubungkan folder `storage\app\` dengan folder `public\`, bisa dengan menjalankan perintah berikut:

```
php artisan storage:link
```

File Upload

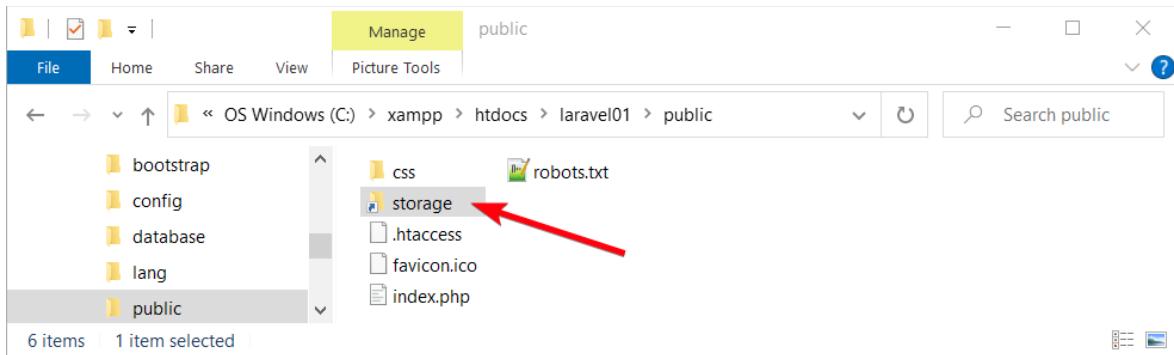


```
C:\xampp\htdocs\laravel01>php artisan storage:link
The [C:\xampp\htdocs\laravel01\public\storage] link has been connected to [C:\xampp\htdocs\laravel01\storage\app\public].
The links have been created.

C:\xampp\htdocs\laravel01>
```

Gambar: Proses pembuatan symlink

Dengan perintah ini, akan tampil sebuah symlink bernama **storage** di folder **public**, silahkan buka folder ini:

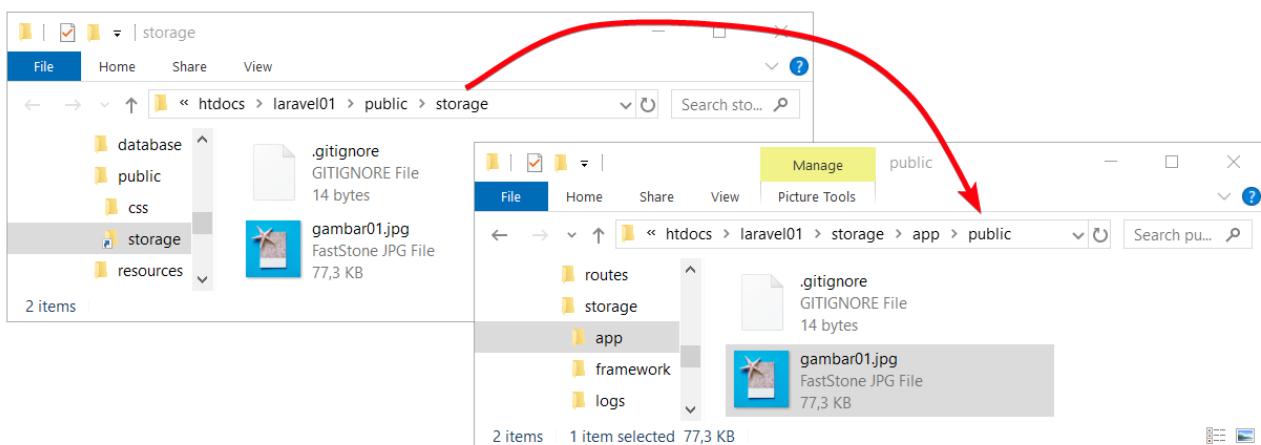


Gambar: Symlink 'storage'

Ketika di klik, terdapat file **.gitignore** di dalam symlink **storage**. Ini hanya file bantu yang berfungsi sebagai tanda agar isi folder **storage** tidak perlu di backup oleh aplikasi **Git**.

Symlink **storage** pada dasarnya merupakan shortcut atau mirror dari folder **storage\app\public**. Dengan kata lain, semua file yang akan kita simpan di **storage\app\public**, juga akan ada di folder **public\storage**.

Untuk uji coba, silahkan copy sebuah file ke folder **storage\app\public**, maka 'secara ajaib' file tersebut juga ada di folder **public\storage**. Dan ketika file itu dihapus, maka di folder lain akan ikut terhapus.



Gambar: Proses percobaan symlink

Saya sangat sarankan untuk mencobanya sendiri agar lebih paham cara kerja symlink. Dan ini

File Upload

berlaku dua arah, jika file di folder `public\storage` di tambah atau di hapus, file yang ada di `storage\app\public` jika akan mengikuti.

Balik kode kode program, tugas kita sekarang adalah meletakkan file yang di upload ke folder `storage\app\public\`, bukan lagi ke folder `storage\app\uploads`. Berikut modifikasi dari method `prosesFileUpload()`:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas' => 'required|file|image|max:1000',
6         ]);
7
8         $extFile = $request->berkas->getClientOriginalExtension();
9         $namaFile = 'lisa-' . time() . '.' . $extFile;
10        $path = $request->berkas->storeAs('public', $namaFile);
11
12        echo "Proses upload berhasil, file berada di: $path";
13    }
```

Perhatikan bahwa argumen pertama dari method `storeAs()` adalah 'public'. Inilah folder tempat file upload akan disimpan. Silahkan upload sebuah file, maka akan tampil hasil berikut:

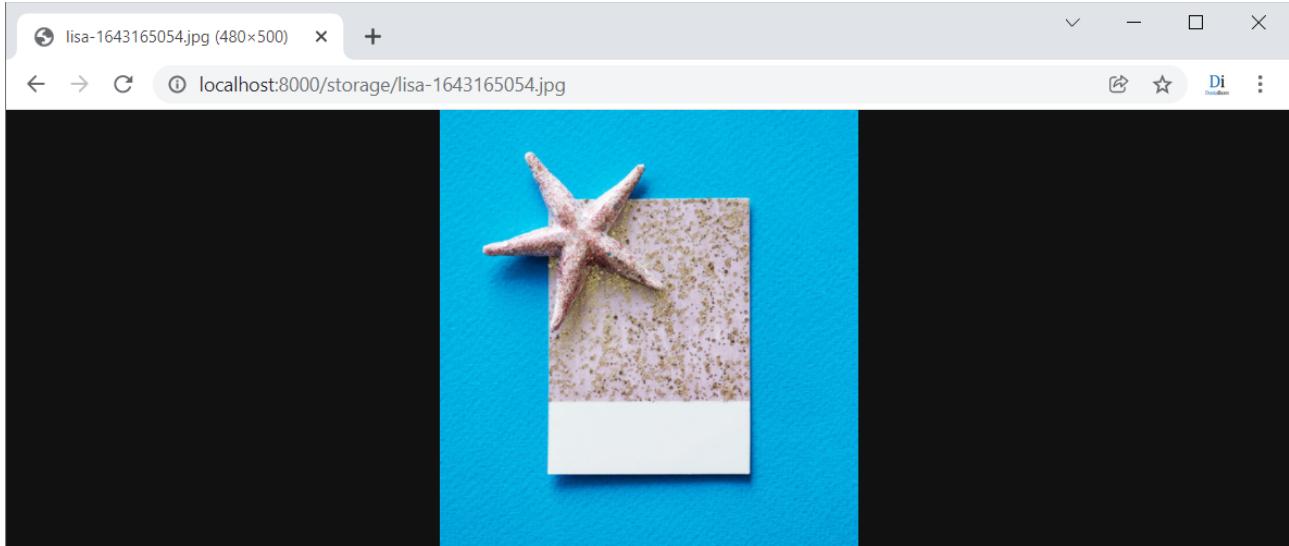


Gambar: Proses upload ke file storage/app/public/

Proses upload berhasil, dan file tersebut ada di `storage\app\public\lisa-1643165054.jpg`. Nama file yang akan anda dapati pastinya berbeda karena di sini saya meng-generate nama file berdasarkan fungsi `time()`.

Karena file upload sudah berada di dalam folder `symlink`, maka bisa diakses dari web browser. Silahkan buka alamat berikut (sesuaikan nama filenya): <http://localhost:8000/storage/lisa-1643165054.jpg>

File Upload



Gambar: File upload sudah bisa diakses dari web browser

Sip, file upload sudah bisa diakses.

Sekarang saya ingin ketika proses upload berhasil, tampilkan teks yang langsung menuju ke alamat yang bisa diakses, yakni folder `storage/<nama_file>`. Saat ini, teks yang tampil adalah sebagai berikut:



Gambar: Proses upload ke file storage/app/public/

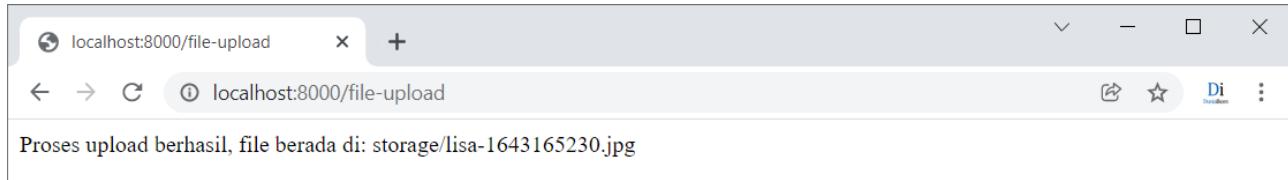
File yang di upload memang akan tersimpan di `public/lisa-1575034421.jpg`, tapi alamat yang bisa diakses adalah `storage/lisa-1575034421.jpg`. Dengan demikian, kita harus modifikasi sedikit isi method `prosesFileUpload()`:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas'  => 'required|file|image|max:1000',
6         ]);
7
8         $extFile = $request->berkas->getClientOriginalExtension();
9         $namaFile = 'lisa-' . time() . '.' . $extFile;
10        $path = $request->berkas->storeAs('public', $namaFile);
11
12        echo "Proses upload berhasil, file berada di: storage/$namaFile";
13    }
```

File Upload

Perubahannya ada di baris 12, dimana nama lokasi file ditulis dari `storage/$namaFile`, tidak lagi dari `$path`.



Gambar: Proses upload, dan menampilkan alamat file dari folder storage

Agar lebih pas lagi, bagaimana jika teks alamat file kita buat menjadi sebuah link? Tidak masalah, tinggal menulisnya ke dalam tag `<a>`:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function prosesFileUpload(Request $request){
4         $request->validate([
5             'berkas'  => 'required|file|image|max:1000',
6         ]);
7
8         $extFile = $request->berkas->getClientOriginalExtension();
9         $namaFile = 'lisa-' . time() . '.' . $extFile;
10        $path = $request->berkas->storeAs('public', $namaFile);
11
12        $pathBaru = asset('storage/' . $namaFile);
13        echo "Proses upload berhasil, file berada di: <a href='\$pathBaru'>
14            \$pathBaru</a>";
15    }
```



Gambar: Proses upload, dan menampilkan link ke folder storage

Di baris 12 saya membuat variabel `$pathBaru` yang berasal dari hasil fungsi `asset('storage/' . $namaFile)`. Alamat inilah yang menjadi nilai untuk atribut `href` di dalam tag `<a>`.

Salah satu fungsi dari penggunaan **symlink** adalah memudahkan pengelolaan file untuk project besar. Bisa saja file upload ini ditempatkan ke server terpisah, misalnya menggunakan layanan cloud storage seperti AWS (Amazon Web Services). Atau jika ingin men-sharing file-file yang sudah di upload ke banyak aplikasi.

20.6. Method move()

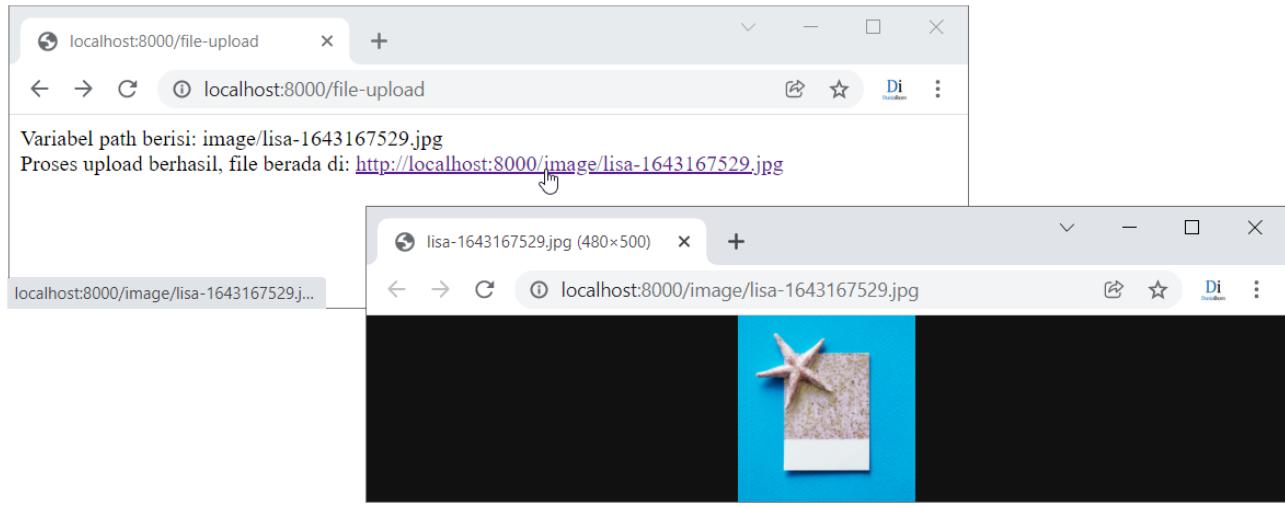
Jika karena sesuatu hal kita tidak bisa (atau tidak ingin) menggunakan `symlink`, Laravel juga menyediakan cara agar file yang di upload langsung tersimpan ke folder public, tidak harus lewat folder storage. Caranya, gunakan method `move()`:

app/Http/Controllers/FileUploadController.php

```

1  <?php
2  ...
3      public function prosesFileUpload(Request $request){
4          $request->validate([
5              'berkas' => 'required|file|image|max:1000',
6          ]);
7
8          $extFile = $request->berkas->getClientOriginalExtension();
9          $namaFile = 'lisa-' . time() . '.' . $extFile;
10
11         $path = $request->berkas->move('image', $namaFile);
12         $path = str_replace('\\', '/', $path);
13         echo "Variabel path berisi: $path <br>";
14
15         $pathBaru = asset('image/' . $namaFile);
16         echo "Proses upload berhasil, file berada di:
17             <a href='$pathBaru'>$pathBaru</a>";
18     }

```



Gambar: Hasil proses upload menggunakan `move()`

Di baris 11 saya menggunakan perintah `$request->berkas->move('image', $namaFile)` untuk memindahkan file upload. Sama seperti method `store()` dan `storeAs()`, method `move()` butuh 2 buah argument. Argument pertama diisi dengan nama folder penyimpanan, dan argument kedua berupa nama file.

Hasilnya, di folder `public` akan muncul folder `image` yang berisi file `lisa-1575099669.jpg`. Karena sudah berada di dalam folder `public`, maka file ini bisa langsung di akses dari web

File Upload

browser.

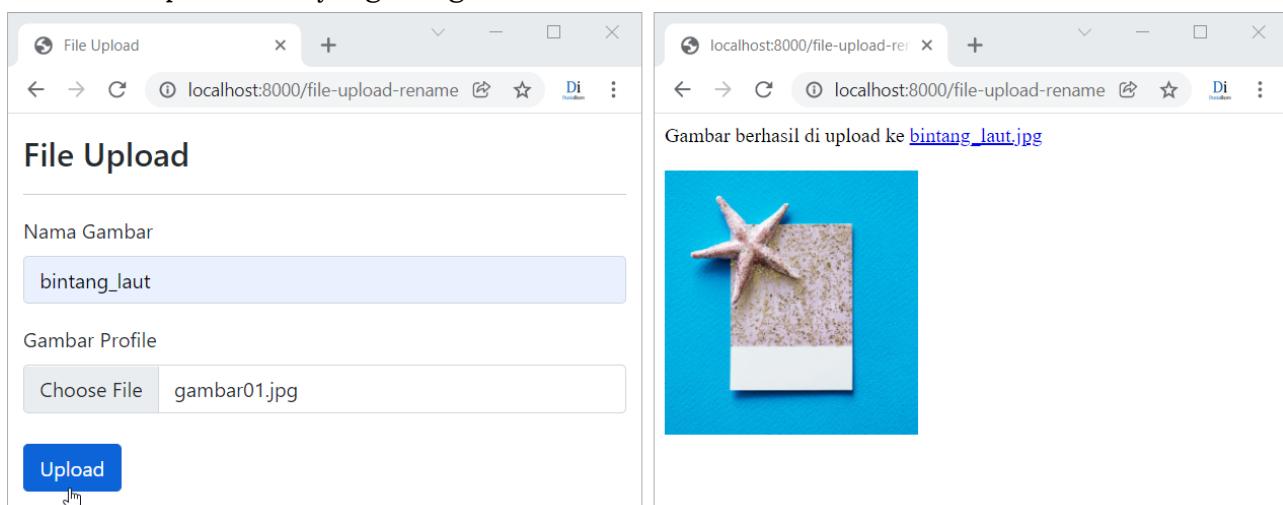
Tambahan fungsi `str_replace()` di baris 12 bertujuan untuk mengganti tanda pemisah folder dari forward slash "\" menjadi back slash "/" untuk variabel `$path`. Misalnya dari `image\lisa-1643167529.jpg` menjadi `image/lisa-1643167529.jpg`.

Tanda pemisah folder ini sebenarnya tergantung jenis OS yang dipakai. Di OS Windows, alamat `http://localhost:8000/image\lisa-1643167529.jpg` tetap bisa diakses. Namun akan lebih rapi jika semua pemisah folder menggunakan karakter back slash "/".

20.7. Mini Case Study: File Upload Rename

Menutup bab tentang file upload ini, saya ingin membuat sebuah studi kasus sederhana atau juga bisa disebut sebagai *exercise*. Idenya adalah, membuat form file upload dimana kita bisa menentukan sendiri nama file akhir. Nama file ini diambil dari inputan text box yang juga ada di dalam form tersebut. Setelah di submit, langsung tampilkan file tersebut di web browser menggunakan tag ``.

Berikut tampilan akhir yang di inginkan:



Gambar: Upload gambar dengan nama yang diisi sendiri

Jika berminat, anda bisa coba buat project ini. Kode yang dibutuhkan sudah kita pelajari semua, yakni pemrosesan form dan juga file upload.

Baik, untuk membuatnya saya akan tetap memakai `FileUploadController`. Tapi kita butuh route dan file view baru. Berikut route yang dibutuhkan:

routes/web.php

```
1 Route::get('/file-upload-rename', [FileUploadController::class,
2   'fileUploadRename']);
3
4 Route::post('/file-upload-rename', [FileUploadController::class,
5   'prosesFileUploadRename']);
```

File Upload

Route pertama akan dipakai untuk menampilkan form, dan route kedua untuk tujuan submit form (pemrosesan file upload).

Untuk menampilkan form, akan diproses oleh method berikut:

app/Http/Controllers/FileUploadController.php

```
1 <?php
2 ...
3     public function fileUploadRename() {
4         return view('file-upload-rename');
5     }
```

Isi dari method `fileUploadRename()` hanya memanggil view `file-upload-rename.blade.php`, dan berikut kode program untuk view ini:

resources/views/file-upload-rename.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>File Upload</title>
9 </head>
10 <body>
11
12 <div class="container mt-3">
13     <h2>File Upload</h2>
14     <hr>
15
16     <form action="{{url('/file-upload-rename')}}" method="POST"
17         enctype="multipart/form-data">
18         @csrf
19
20         <div class="mb-3">
21             <label for="nama_gambar" class="form-label">Nama Gambar</label>
22             <input type="text" class="form-control" id="nama_gambar"
23             name="nama_gambar" value="{{ old('nama_gambar') }}">
24             @error('nama_gambar')
25                 <div class="text-danger">{{ $message }}</div>
26             @enderror
27         </div>
28
29         <div class="mb-3">
30             <label for="gambar_profile" class="form-label">Gambar Profile</label>
31             <input type="file" class="form-control" id="gambar_profile"
32             name="gambar_profile">
33             @error('gambar_profile')
34                 <div class="text-danger">{{ $message }}</div>
35             @enderror
36         </div>
```

File Upload

```
37      <button type="submit" class="btn btn-primary my-2">Upload</button>
38  </form>
39 </div>
40
41
42 </body>
43 </html>
```

Di dalam form saya menambah sebuah inputan text dengan atribut `name="nama_gambar"`. Selain itu inputan file juga memakai `name="gambar_profile"`. Kedua atribut inilah yang akan kita gunakan untuk proses validasi dan proses file upload.

Ketika di submit, data form dikirim ke `url('/file-upload-rename')`, yang akan diterima oleh method `prosesFileUploadRename()` berikut:

app/Http/Controllers/FileUploadController.php

```
1  <?php
2  ...
3  public function prosesFileUploadRename(Request $request){
4      $request->validate([
5          'nama_gambar' => 'required|min:5|alpha_dash',
6          'gambar_profile' => 'required|file|image|max:1000',
7      ]);
8
9      // ambil nama extension file asal
10     $extFile = $request->gambar_profile->getClientOriginalExtension();
11     // generate nama file akhir, diambil dari inputan nama_gambar + extension
12     $namaFile = $request->nama_gambar.".".$extFile;
13     // pindahkan file upload ke folder storage/app/public/gambar/
14     $request->gambar_profile->storeAs('public/gambar', $namaFile);
15
16     // generate path gambar yang bisa diakses (path di folder public)
17     $pathPublic = asset('storage/gambar/'.$namaFile);
18
19     echo "Gambar berhasil di upload ke <a href=". $pathPublic. ">$namaFile</a>";
20     echo "<br><br>";
21     echo "";
22 }
```

Untuk proses validasi form, terdapat tambahan syarat `alpha_dash` di baris 5. Ini berfungsi untuk membatasi inputan `nama_gambar` agar hanya boleh diisi karakter huruf, angka, serta karakter dash " - " dan underscore " _ ".

Berikutnya di baris 10 adalah kode untuk mengambil extension file menggunakan method `getClientOriginalExtension()`, ini sudah sering kita pakai sebelumnya.

Proses pembuatan nama file dilakukan di baris 12, di sini saya menggabung hasil `$request->nama_gambar` yang berasal dari inputan form dengan variabel `$extFile` yang berisi string extension file asal.

Perintah untuk pemindahan file upload ada di baris 14. Saya memindahkan file upload ke

File Upload

folder 'public/gambar'. Artinya file upload akan berada di dalam sub folder gambar, tidak langsung di dalam folder public. Ini bukan sebuah keharusan, hanya agar struktur folder kita menjadi lebih rapi saja.

Alamat path di rangkai pada baris 17 dengan fungsi `asset('storage/gambar/' . $namaFile)`. Ini akan dipakai sebagai nilai atribut `href` di baris 19, dan atribut `src` di baris 21.

Idealnya, proses tampilan seperti tag `<a>` dan `` dibuat di dalam view, tapi saya yakin anda sudah bisa memahami konsep dasar dari proses file upload. Malah biasanya, untuk sebuah aplikasi nama gambar perlu di simpan ke database untuk diakses di lain waktu. Ini akan menjadi salah satu fitur untuk studi kasus di bab terakhir nanti.

Dalam bab ini kita telah membahas cara melakukan file upload menggunakan Laravel. Yang perlu sedikit pemahaman ada di struktur folder tempat menyimpan file, serta penggunaan metode **symlink**.

Lanjut, kita akan bahas tentang **Middleware**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

21. Middleware

Materi kali ini berhubungan dengan proses *request* dan *respond* dari aplikasi Laravel. Menggunakan **middleware**, kita bisa menginterupsi request dari web browser, misalnya melakukan pemeriksaan apakah user sudah login atau belum.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

21.1. Pengertian Middleware

Middleware adalah sebuah class khusus yang berfungsi untuk men-filter HTTP request dari web browser ke aplikasi kita. Atau dalam pengertian yang lebih sederhana, *middleware* adalah kode program yang bisa meng-interupsi setiap permintaan halaman web.

Implementasi paling cocok dari middleware adalah untuk *autentikasi*, yakni proses login dan logout user. Umum kita jumpai sebuah halaman hanya bisa diakses jika user sudah login terlebih dahulu. Jika belum, maka user akan di *redirect* ke form login. Setelah login, barulah halaman tersebut bisa diakses kembali.

21.2. Persiapan Awal

Seperti biasa, sebelum masuk ke praktek pembuatan *middleware*, kita perlu persiapan berupa route dan controller. Untuk controller, saya kembali menggunakan **MahasiswaController** yang bisa dibuat dengan perintah berikut:

```
php artisan make:controller MahasiswaController
```

Silahkan tambah beberapa method ke controller ini:

```
app/Http/Controllers/MahasiswaController.php
```

```
1 <?php  
2
```

```

3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class MahasiswaController extends Controller
8  {
9      public function daftarMahasiswa()
10     {
11         return 'Form pendaftaran mahasiswa';
12     }
13
14     public function tabelMahasiswa()
15     {
16         return 'Tabel data mahasiswa';
17     }
18
19     public function blogMahasiswa()
20     {
21         return 'Halaman blog mahasiswa';
22     }
23 }

```

Terdapat 3 method, yakni `daftarMahasiswa()`, `tabelMahasiswa()` dan `blogMahasiswa()`. Semua method ini lebih ke method *dummy* yang ketika diakses mengembalikan string sederhana.

Tidak masalah, karena fokus utama kita sekarang adalah memahami prinsip kerja dari middleware saja. Dalam prakteknya nanti, setiap method bisa berisi kode program yang lebih kompleks seperti mengakses database atau menampilkan view. Dan berikut route untuk mengakses ketiga method di atas:

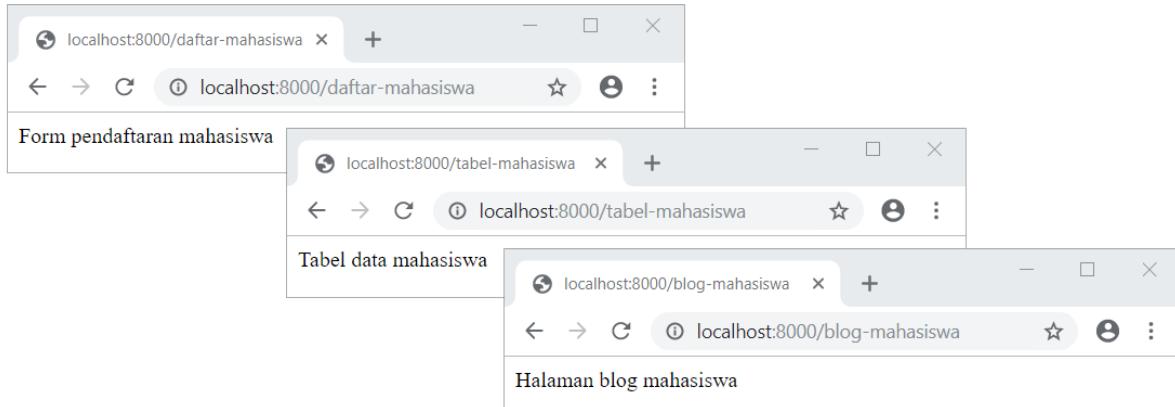
`routes/web.php`

```

1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\MahasiswaController;
5
6  Route::get('/daftar-mahasiswa', [MahasiswaController::class, 'daftarMahasiswa']);
7  Route::get('/tabel-mahasiswa', [MahasiswaController::class, 'tabelMahasiswa']);
8  Route::get('/blog-mahasiswa', [MahasiswaController::class, 'blogMahasiswa']);

```

Juga tidak ada sesuatu yang baru, hanya route biasa yang akan mengakses method milik `MahasiswaController`. Untuk memastikan, silahkan buka halaman localhost:8000/daftar-mahasiswa, localhost:8000/tabel-mahasiswa dan localhost:8000/blog-mahasiswa.



Gambar: Ketiga halaman sudah berhasil diakses

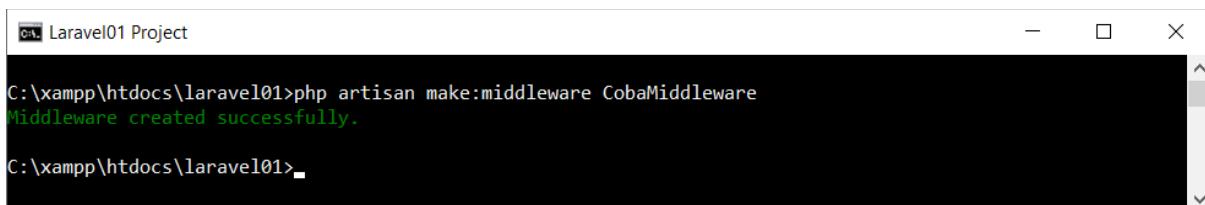
21.3. Membuat Middleware

Middleware pada dasarnya berbentuk sebuah file PHP. Untuk membuat middleware, bisa menggunakan perintah php artisan dengan format berikut:

```
php artisan make:middleware <NamaMiddleware>
```

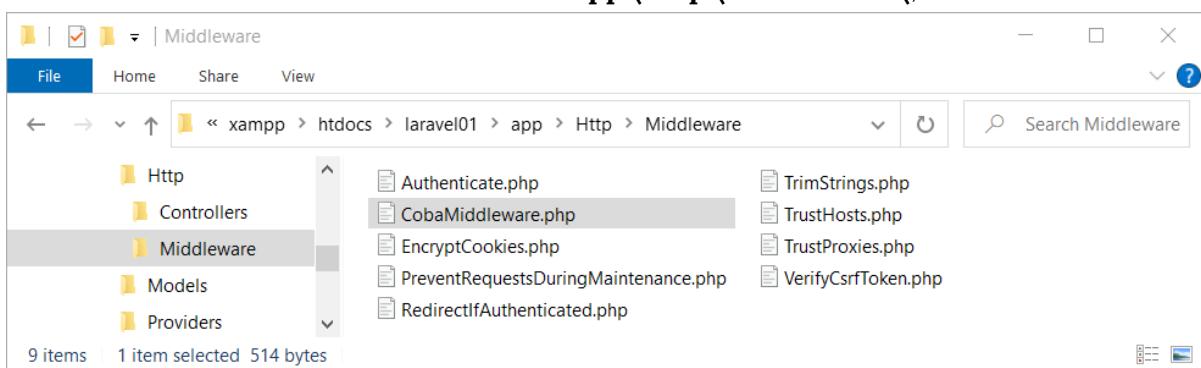
Middleware nantinya berbentuk sebuah class PHP, sehingga saran penamaan memakai **PascalCase** (huruf besar di setiap awal kata). Nama middleware sebaiknya mencerminkan fungsi dari middleware tersebut, tapi kali ini saya akan membuat middleware bernama **CobaMiddleware**:

```
php artisan make:middleware CobaMiddleware
```



Gambar: Pembuatan middleware CobaMiddleware

File Middleware akan berada di dalam folder **app\Http\Middleware**, silahkan buka folder ini:



Gambar: Isi folder app\Http\Middleware\

Selain `CobaMiddleware.php` yang baru saja kita buat, di dalam folder ini juga terdapat beberapa file middleware lain bawaan Laravel. Diantaranya `VerifyCsrfToken.php` yang akan meng-interupsi (menampilkan halaman error) jika kita lupa menulis perintah `@crfs` saat submit form.

Mari lihat isi file `CobaMiddleware.php`:

`app/Http/Middleware/CobaMiddleware.php`

```

1 <?php
2
3 namespace App\Http\Middleware;
4
5 use Closure;
6 use Illuminate\Http\Request;
7
8 class CobaMiddleware
9 {
10     /**
11      * Handle an incoming request.
12      *
13      * @param \Illuminate\Http\Request $request
14      * @param \Closure $next
15      * @return mixed
16      */
17     public function handle(Request $request, Closure $next)
18     {
19         return $next($request);
20     }
21 }
```

Ini merupakan struktur dasar dari middleware Laravel. Di baris 3 terdapat penulisan `namespace` yang sama dengan lokasi file ini, yakni di dalam folder `app\Http\Middleware`. Kemudian di baris 5 dan 6 terdapat proses import class `Closure` menggunakan perintah `use Closure` serta import class `Request` dengan perintah `use Illuminate\Http\Request`.

Class `CobaMiddleware` di buat mulai dari baris 8, yang diikuti beberapa komentar di baris 10–16.

Isi dari class `CobaMiddleware` hanya terdiri dari 1 method bernama `handle()`. Method ini memiliki 2 buah parameter bernama `$request` dan `$next`. Khusus untuk `$next`, di-type hint dengan class `Closure`.

Perintah `return $next($request)` artinya middleware ini akan lanjut memproses request berikutnya. Sehingga jika kita ingin meng-interupsi alur request, tulis sebelum baris ini. Sebagai contoh sederhana, saya akan tambah perintah `dd('CobaMiddleware aktif')`:

`app/Http/Middleware/CobaMiddleware.php`

```

1 <?php
2 ...
3 class CobaMiddleware
4 {
5     public function handle($request, Closure $next)
```

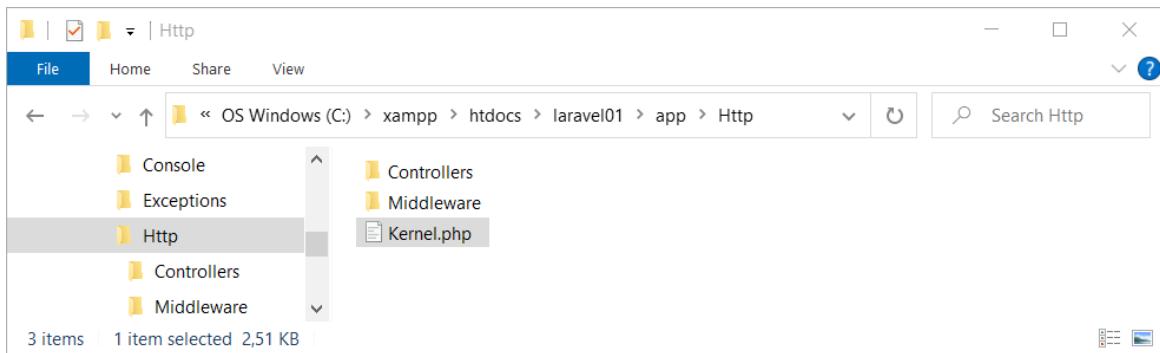
Middleware

```
6      {
7          dd('CobaMiddleware aktif');
8          return $next($request);
9      }
10 }
```

Dengan tambahan ini, maka semua halaman yang menerapkan CobaMiddleware akan menampilkan teks "CobaMiddleware aktif" dan karena dd() berisi perintah die(), maka request berikutnya tidak akan dijalankan lagi.

21.4. Mendaftarkan Middleware

Middleware yang baru saja kita buat tidak langsung aktif, tapi harus di daftarkan terlebih dahulu ke file app\Http\Kernel.php:



Gambar: File Kernel.php di folder app\Http\Kernel.php

Kode program dalam file ini cukup panjang (sekitar 66 baris) karena juga berisi semua daftar middleware bawaan Laravel:

app/Http/Kernel.php

```
1  <?php
2  namespace App\Http;
3  use Illuminate\Foundation\Http\Kernel as HttpKernel;
4
5  class Kernel extends HttpKernel
6  {
7      protected $middleware = [
8          \App\Http\Middleware\TrustProxies::class,
9          ...
10 ];
11     protected $middlewareGroups = [
12         'web' => [
13             \App\Http\Middleware\EncryptCookies::class,
14             ...
15         ],
16         'api' => [
17             'throttle:api',
18             \Illuminate\Routing\Middleware\SubstituteBindings::class,
19         ],
20     ];
21 }
```

```

20     ];
21     protected $routeMiddleware = [
22         'auth' => \App\Http\Middleware\Authenticate::class,
23         ...
24     ];
25 ];
26 }

```

Untuk menghemat tempat, saya memangkas sedikit kode ini, diantaranya menghapus semua baris komentar.

Di baris 5 terdapat perintah pembuatan class `Kernel` yang di extends dari class `HttpKernel`. Fokus utama kita ada di 4 property class ini, yaitu: `$middleware`, `$middlewareGroups`, `$routeMiddleware` dan `$middlewarePriority`. Semua property ini di set dengan hak akses `protected`.

Setiap property berisi array dari middleware class, dengan penjelasan sebagai berikut:

- **`$middleware`**: Berisi daftar middleware global yang selalu aktif untuk setiap request.
- **`$middlewareGroups`**: Berisi daftar middleware yang aktif per-group. Secara bawaan sudah ada 2 buah group, yakni 'web' dan 'api'. Semua middleware yang ada di dalam group 'web' akan berjalan ketika terdapat request yang berasal dari web browser.
- **`$routeMiddleware`**: Berisi daftar middleware opsional yang aktif untuk route tertentu saja. Umumnya di sinilah kita akan mendaftarkan middleware yang dibuat.

Sebagai praktek pertama, silahkan tambah class `CobaMiddleware` ke property `$middleware`:

```

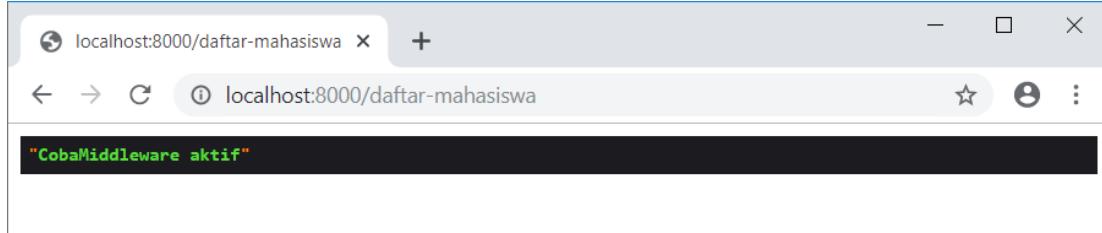
1  class Kernel extends HttpKernel
2  {
3     protected $middleware = [
4         // \App\Http\Middleware\TrustHosts::class,
5         \App\Http\Middleware\TrustProxies::class,
6         \Fruitcake\Cors\HandleCors::class,
7         \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
8         \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
9         \App\Http\Middleware\TrimStrings::class,
10        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
11        \App\Http\Middleware\CobaMiddleware::class
12    ];
13 }

```

Untuk mendaftarkan middleware, kita harus menulis lengkap namespace beserta nama class dan diakhiri dengan tambahan `::class`. Di baris 11 saya menambah class `\App\Http\Middleware\CobaMiddleware::class` ke dalam array `$middleware`.

Property `$middleware` sendiri adalah tempat untuk mendaftarkan **global middleware**, yakni middleware yang selalu aktif untuk setiap request. Silahkan akses salah satu route yang sudah kita siapkan, misalnya <localhost:8000/daftar-mahasiswa>.

Hasilnya akan tampil teks "CobaMiddleware aktif" yang berasal dari fungsi dd() di dalam CobaMiddleware class. Ini berlaku untuk semua route yang ada di Laravel.



Gambar: CobaMiddleware sudah aktif

Percobaan berikutnya, pindahkan baris \App\Http\Middleware\CobaMiddleware::class ke dalam property \$middlewareGroups:

```

1  class Kernel extends HttpKernel
2  {
3      ...
4      protected $middlewareGroups = [
5          'web' => [
6              ...
7              \App\Http\Middleware\VerifyCsrfToken::class,
8              \Illuminate\Routing\Middleware\SubstituteBindings::class,
9              \App\Http\Middleware\CobaMiddleware::class,
10         ],
11
12         'api' => [
13             ...
14         ],
15     ];
16 }
```

Property \$middlewareGroups terdiri dari 2 nested array, yakni 'web' dan 'api'. Jika sebuah middleware di daftarkan ke dalam group 'web', maka itu akan aktif untuk setiap request yang berasal dari web browser. Sedangkan jika didaftarkan ke group 'api', maka hanya akan berjalan jika ada request yang meminta **API**.

Di baris 9 saya menambah class CobaMiddleware ke dalam group 'web'. Hasilnya sama seperti sebelumnya, untuk ketiga route akan menghasilkan teks "CobaMiddleware aktif". Ini karena kita masih menggunakan route web, bukan route untuk API.

Jika kita ingin middleware aktif untuk route tertentu saja, maka tempatkan class middleware ke dalam property \$routeMiddleware. Silahkan hapus class CobaMiddleware dari \$middlewareGroups dan pindahkan ke \$routeMiddleware:

```

1  class Kernel extends HttpKernel
2  {
3      ...
4      protected $routeMiddleware = [
5          'auth' => \App\Http\Middleware\Authenticate::class,
6          ...
```

```

7      'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
8      'coba' => \App\Http\Middleware\CobaMiddleware::class,
9  ];
10 }

```

Pengisian CobaMiddleware ke dalam array \$routeMiddleware perlu di tulis dalam bentuk associative array, yakni dengan pasangan 'key' dan nama class.

Key ini akan dipakai pada saat mengaktifkan middleware ke dalam route atau controller (dibahas sesaat lagi). Dalam contoh ini saya menggunakan key 'coba' seperti di baris 8.

Middleware yang ada di \$routeMiddleware tidak langsung berjalan, tapi harus di aktifkan terlebih dahulu. Ketika halaman <localhost:8000/daftar-mahasiswa> diakses, akan tetap tampil seperti biasa.

21.5. Mengaktifkan Middleware

Dalam praktek sebelumnya, middleware CobaMiddleware sudah berjalan namun baru sebagai *global middleware*. Kali ini kita akan bahas cara mengaktifkan middleware untuk route tertentu saja. Syaratnya, middleware tersebut harus didaftarkan terlebih dahulu ke dalam property \$routeMiddleware seperti yang sudah kita lakukan.

Terdapat 2 cara mengaktifkan middleware, yakni dari route dan dari controller:

Mengaktifkan Middleware dari Route

Untuk mengaktifkan middleware dari route, bisa dengan men-*chaining* method `middleware('<nama_key>')` ke dalam route, seperti contoh berikut:

routes/web.php

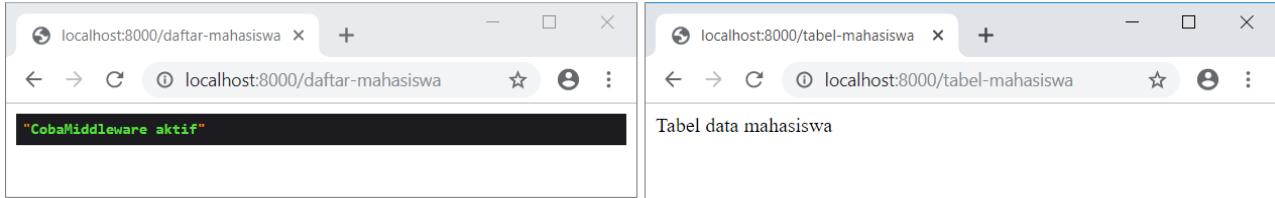
```

1 Route::get('/daftar-mahasiswa', [MahasiswaController::class, 'daftarMahasiswa'])
2 ->middleware('coba');
3
4 Route::get('/tabel-mahasiswa', [MahasiswaController::class, 'tabelMahasiswa']);
5
6 Route::get('/blog-mahasiswa', [MahasiswaController::class, 'blogMahasiswa'])
7 ->middleware('coba');

```

Di sini saya mengaktifkan CobaMiddleware ke dalam route '/daftar-mahasiswa' dan '/blog-mahasiswa'. Key 'coba' adalah key yang kita tulis pada saat mendaftarkan middleware ke property \$routeMiddleware.

Hasilnya, ketika halaman <localhost:8000/daftar-mahasiswa> dan <localhost:8000/blog-mahasiswa> diakses, akan tampil teks "CobaMiddleware aktif". Namun halaman <localhost:8000/tabel-mahasiswa> akan tampil normal karena tidak di set dengan middleware coba.



Gambar: Middleware aktif untuk halaman tertentu saja

Jika terdapat beberapa middleware yang ingin diaktifkan untuk satu route, penulisannya bisa disambung sebagai berikut:

```
Route::get('/nama-route', [ContohController::class, 'foo'])->middleware('pertama', 'kedua');
```

Dalam prakteknya nanti, kita bisa menentukan halaman apa saja yang hanya bisa diakses oleh user dengan hak akses tertentu. Caranya, tambah middleware pemeriksaan login ke dalam route yang ingin di proteksi.

Mengaktifkan Middleware dari Controller

Alternatif cara kedua untuk mengaktifkan middleware adalah dari constructor di dalam Controller.

Sebelum menjalankan praktek ini, silahkan hapus middleware coba yang sebelumnya kita aktifkan dari route.

Sebagai contoh, agar semua method yang terdapat di `MahasiswaController` menjalankan middleware `coba`, bisa tambah kode berikut:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2 ...
3 class MahasiswaController extends Controller
4 {
5     public function __construct()
6     {
7         $this->middleware('coba');
8     }
9     ...
10    ...
11 }
```

Kita memang belum pernah menulis constructor di dalam Controller. Namun karena Controller pada dasarnya merupakan class PHP, maka cara penulisan constructor sama seperti class PHP biasa, yakni menggunakan method `__construct()`.

Perintah untuk mengaktifkan middleware ditulis dengan format: `$this->middleware`

('<nama_key>'). Sehingga untuk menjalankan middleware coba, perintahnya adalah:

```
$this->middleware('coba');
```

Jika ditulis seperti ini, maka semua method yang ada di dalam `MahasiswaController` akan menjalankan middleware 'coba'.

Bagaimana jika kita ingin mengaktifkan middleware untuk method tertentu saja? Caranya, chaining dengan method `only('<nama_method>')`, seperti contoh berikut:

```
$this->middleware('coba')->only('daftarMahasiswa');
```

Perintah ini akan mengaktifkan middleware coba hanya untuk method `daftarMahasiswa()` saja. Jika kita ingin menambah method lain, bisa ditulis sebagai berikut:

```
$this->middleware('coba')->only('daftarMahasiswa', 'tabelMahasiswa');
```

Sekarang method `daftarMahasiswa()` dan `tabelMahasiswa()` akan menampilkan teks "CobaMiddleware aktif" ketika diakses.

Penulisan lain, kita bisa mengaktifkan middleware untuk semua method, kecuali method tertentu:

```
$this->middleware('coba')->except('tabelMahasiswa');
```

Perintah ini akan mengaktifkan middleware coba untuk semua method, kecuali method `tabelMahasiswa()`.

Berikut kode program lengkap dari `MahasiswaController` dengan tambahan constructor:

app/Http/Controllers/MahasiswaController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9
10    public function __construct()
11    {
12        $this->middleware('coba')->except('tabelMahasiswa');
13    }
14
15    public function daftarMahasiswa()
16    {
17        return 'Form pendaftaran mahasiswa';
18    }
19
20    public function tabelMahasiswa()
```

```

21     {
22         return 'Tabel data mahasiswa';
23     }
24
25     public function blogMahasiswa()
26     {
27         return 'Halaman blog mahasiswa';
28     }
29
30 }
```

Perintah di baris 12 akan mengaktifkan middleware coba untuk semua method, kecuali method tabelMahasiswa().

Jika kita mengaktifkan middleware dari controller seperti ini, maka itu akan berlaku untuk semua route yang mengakses method tersebut. Karena bisa jadi ada beberapa route yang di proses oleh 1 method di controller.

21.6. Redirect Middleware

Dalam praktek sebelumnya kita sudah berhasil menjalankan CobaMiddleware. Namun hasilnya memang baru sekedar teks saja. Biasanya middleware ini akan me-redirect user ke halaman lain jika suatu kondisi tidak terpenuhi.

Kondisi tersebut bisa di rancang sesuai kebutuhan, misalnya jika user belum login atau user tidak memiliki hak akses tertentu. Namun proses pemeriksaan login dan hak akses ini butuh materi tambahan lain yang belum kita pelajari.

Sebagai bahan praktek untuk membuat middleware yang akan me-redirect user, saya akan memodifikasi CobaMiddleware menjadi sebagai berikut:

app/Http/Middleware/CobaMiddleware.php

```

1 <?php
2
3 namespace App\Http\Middleware;
4 use Closure;
5
6 class CobaMiddleware
7 {
8     public function handle($request, Closure $next)
9     {
10         if (time() % 2 == 0) {
11             return redirect('/tabel-mahasiswa');
12         }
13         return $next($request);
14     }
15 }
```

Fokus utama kita ada di baris 10 – 12. Di sini saya membuat kondisi if (time() % 2 == 0).

Kondisi ini akan bernilai **true** jika detik saat ini habis dibagi 2, hasilnya web browser akan di-redirect ke halaman `/tabel-mahasiswa`.

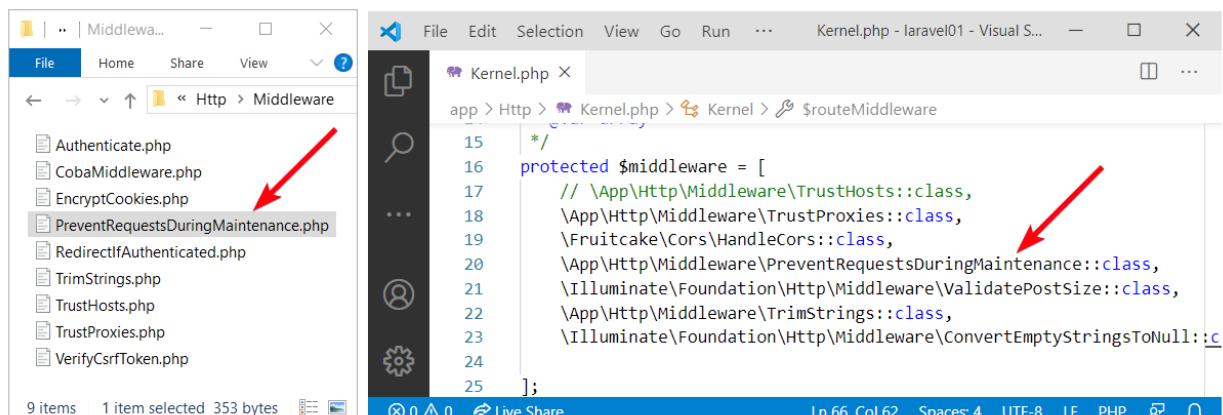
Percobaan ini memang cukup aneh, tapi mudah-mudahan anda bisa menangkap idenya.

Pastikan middleware `coba` aktif untuk halaman `/daftar-mahasiswa`, lalu test akses dari web browser. Jika beruntung, pada saat halaman `/daftar-mahasiswa` tampil, akan langsung berganti ke `/tabel-mahasiswa`.

Kondisi ini nanti bisa kita tukar dengan perintah yang lebih rumit, misalnya memeriksa apakah di dalam session terdapat variabel tertentu atau tidak (sebagai tanda seseorang sudah login). Lebih jauh tentang session dan juga proses autentikasi akan kita bahas di bab terpisah.

21.7. Laravel Maintenance Mode

Menutup bab ini, saya ingin mempraktekkan salah satu middleware bawaan Laravel, yakni `PreventRequestsDuringMaintenance.php`. File ini sudah ada di dalam folder `app\Http\Middleware\` dan juga sudah terdaftar sebagai global middleware.



Gambar: File PreventRequestsDuringMaintenance.php

Middleware ini dipakai untuk membuat Laravel masuk ke **maintenance mode**, dimana halaman web tidak bisa diakses dari luar. Ini cocok dipakai ketika kita ingin melakukan maintenance atau pengelolaan aplikasi yang sudah berjalan. Akan jadi masalah jika ada user yang melihat web kita "berantakan" karena memang sedang ada perubahan kode program.

Untuk mengaktifkan maintenance mode, bisa dilakukan dari cmd dengan perintah berikut:

```
php artisan down
```

```

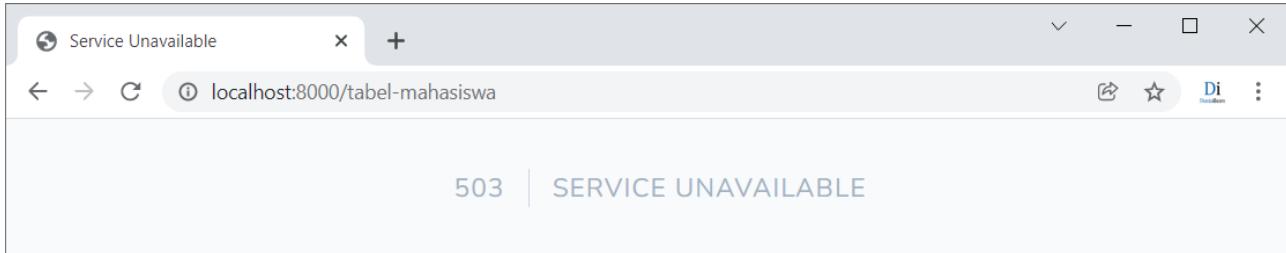
C:\xampp\htdocs\laravel01>php artisan down
Application is now in maintenance mode.

C:\xampp\htdocs\laravel01>

```

Gambar: Masuk ke maintenance mode

Setelah perintah dijalankan, setiap request ke web browser akan menampilkan hasil berikut:



Gambar: Halaman error 503

Setelah proses maintenance selesai, tutup middleware maintenance ini dengan perintah:

```
php artisan up
```

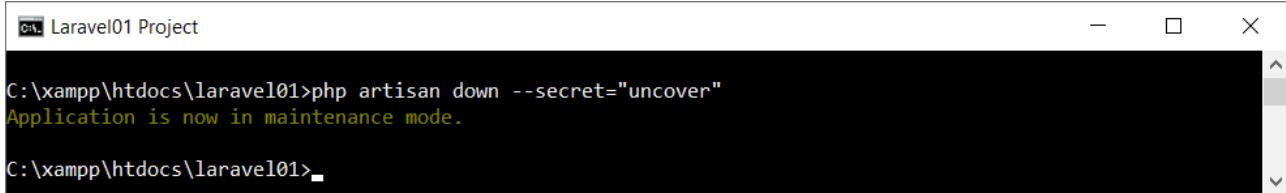


Gambar: Maintenance mode selesai

Sekarang, aplikasi Laravel bisa diakses kembali secara normal.

Penggunaan php artisan down memang praktis, tapi itu menutup tampilan untuk semua user. Agar kita sebagai admin tetap bisa mengaksesnya, tambah kode rahasia dengan flag --secret. Sebagai contoh saya ingin membuat kode rahasia "uncover", maka jalankan perintah:

```
php artisan down --secret="uncover"
```



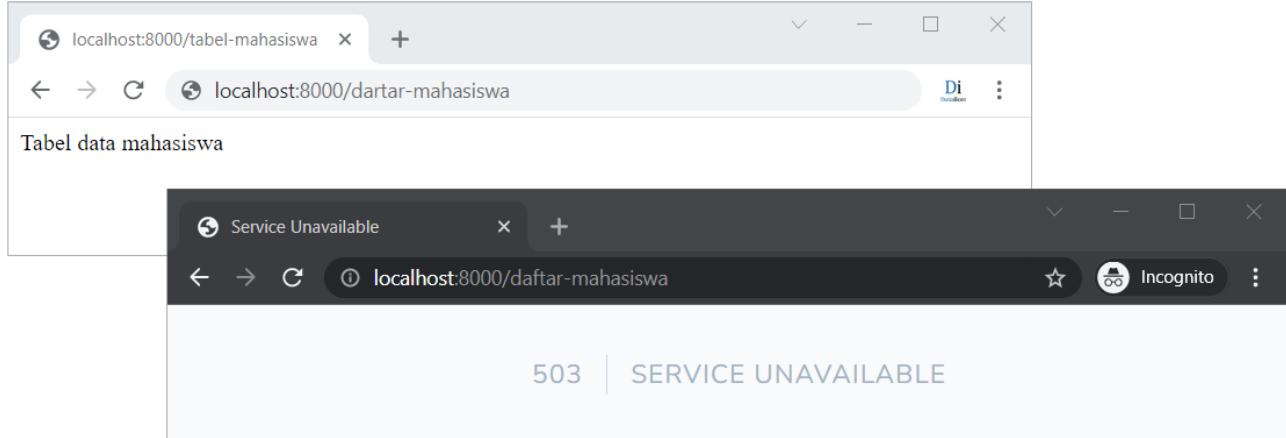
Gambar: Maintenance mode dengan tambahan --secret="uncover"

Selanjutnya, buka alamat URL homepage dengan tambahan "/uncover", yakni:

<http://localhost:8000/uncover>. Dalam praktek ini bisa saja tampil halaman 404 karena saya sudah menghapus route "/" bawaan laravel. Tapi itu tidak masalah, begitu halaman <http://localhost:8000/uncover> diakses sekali, Laravel akan membuat session di web browser.

Sekarang kita tetap bisa mengakses URL lain seperti <http://localhost:8000/tabel-mahasiswa>, atau <http://localhost:8000/dartar-mahasiswa>. Sedangkan user lain yang mengakses web akan menemukan halaman 503 | SERVICE UNAVAILABLE.

Untuk testing, bisa tes dengan web browser berbeda atau pakai incognito mode:



Gambar: Halaman bisa diakses dengan secret key (atas), dan tidak bisa diakses (bawah)

Dalam bab ini kita telah mempelajari tentang middleware. Untuk praktik lengkapnya akan kembali kita bahas dalam studi kasus pembuatan autentikasi. Namun sebelum itu kita akan masuk ke materi **Session** terlebih dahulu.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

22. Session

Secara sederhana, **session** adalah data yang bisa kita "titip" ke dalam web browser untuk diakses oleh halaman yang berbeda. Jika anda pernah membuat kode program dengan proses login dan logout, besar kemungkinan itu menggunakan session (atau bisa juga lewat cookie).

Session dan cookie merupakan materi dasar PHP, sehingga saya berasumsi keduanya sudah dipahami dengan baik. Kali ini kita akan bahas bagaimana cara membuat, mengakses dan menghapus cookie di Laravel.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

22.1. Persiapan Awal

Materi tentang session ini tidak terlalu panjang, tapi tetap perlu beberapa persiapan, yakni controller dan route. Untuk controller, saya akan membuat **SessionController** dengan perintah di bawah ini:

```
php artisan make:controller SessionController
```

Dan berikut kode program sementara dari **SessionController.php**:

```
app/Http/Controllers/SessionController.php
```

```

1 <?php
2
3 namespace App\Http\Controllers;
4 use Illuminate\Http\Request;
5
6 class SessionController extends Controller
7 {
8     public function index()
9     {
10         echo '<ul>';
11         echo '<li><a href=/buat-session>Buat Session</a></li>';
12         echo '<li><a href=/akses-session>Akses Session</a></li>';

```

Session

```
13     echo '<li><a href=/hapus-session>Hapus Session</a></li>';
14     echo '<li><a href=/flash-session>Flash Session</a></li>';
15     echo '</ul>';
16 }
17
18 public function buatSession()
19 {
20     //
21 }
22
23 public function aksesSession()
24 {
25     //
26 }
27
28 public function hapusSession()
29 {
30     //
31 }
32
33 public function flashSession()
34 {
35     //
36 }
37 }
```

SessionController memiliki 5 buah method. Method `index()` berisi 6 baris perintah `echo` untuk membuat list yang berisi link ke 4 halaman lain. Tampilan ini sebenarnya lebih cocok jika ditulis ke dalam view, namun untuk menyederhanakan pembahasan, saya tulis saja di controller.

Empat method lain akan kita isi secara bertahap, yakni `buatSession()`, `aksesSession()`, `hapusSession()`, dan `flashSession()`. Sesuai dengan namanya, method-method ini di pakai untuk membuat, mengakses, menghapus, serta mengakses flash session.

Berikut route yang diperlukan:

routes/web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\SessionController;
5
6 Route::get('/', [SessionController::class, 'index']);
7 Route::get('/buat-session', [SessionController::class, 'buatSession']);
8 Route::get('/akses-session', [SessionController::class, 'aksesSession']);
9 Route::get('/hapus-session', [SessionController::class, 'hapusSession']);
10 Route::get('/flash-session', [SessionController::class, 'flashSession']);
```

Route ini mengakses kelima method milik SessionController sebelumnya. Untuk memastikan persiapan kita sudah berjalan lancar, silahkan akses halaman localhost:8000:



Gambar: Tampilan halaman http://localhost:8000

Sip, saatnya masuk ke cara pembuatan session.

22.2. Membuat Session

Terdapat beberapa cara untuk membuat session di Laravel, antara lain:

- ◆ Menggunakan function `helper session(<session_name>, <session_value>)`
- ◆ Melalui method `$request->session()->put(<session_name>, <session_value>)` dari **Request** object
- ◆ Melalui method `Session::put(<session_name>, <session_value>)` dari **Session** facade

Sebagai contoh, untuk membuat session bernama `hakAkses` yang berisi string '`admin`', bisa menggunakan salah satu kode berikut:

- ◆ `session(['hakAkses' => 'admin']);`
- ◆ `$request->session()->put('hakAkses', 'admin');`
- ◆ `Session::put('hakAkses', 'admin');`

Untuk cara pertama, yakni dari function `session()`, bisa langsung kita tulis ke dalam controller:

```
1 public function buatSession()
2 {
3     session(['hakAkses' => 'admin']);
4 }
```

Cara kedua adalah melalui **Request** object, sehingga kita harus menulis type hint `Request $request` sebagai argument method:

```
1 public function buatSession(Request $request)
2 {
3     $request->session()->put('hakAkses', 'admin');
4 }
```

Cara ketiga menggunakan facade `Session::put()`, sehingga kita juga harus menambah perintah import `use Illuminate\Support\Facades\Session` di bagian atas controller:

Session

```
1 ...  
2 use Illuminate\Support\Facades\Session;  
3  
4     public function buatSession()  
5     {  
6         Session::put('hakAkses', 'admin');  
7     }
```

Ketiga cara ini bisa dipakai, namun dengan function `session()` penulisannya jadi lebih singkat.

Jika kita ingin membuat beberapa session sekaligus, bisa ditulis dalam bentuk array:

- ◆ `session(['hakAkses' => 'admin', 'nama' => 'Anto']);`
- ◆ `$request->session()->put(['hakAkses' => 'admin', 'nama' => 'Anto']);`
- ◆ `Session::put(['hakAkses' => 'admin', 'nama' => 'Anto']);`

Dengan salah satu perintah ini, akan tersedia 2 buah session: `hakAkses` dan `nama`. Untuk keperluan yang lebih kompleks, session juga bisa dibuat dalam bentuk nested array:

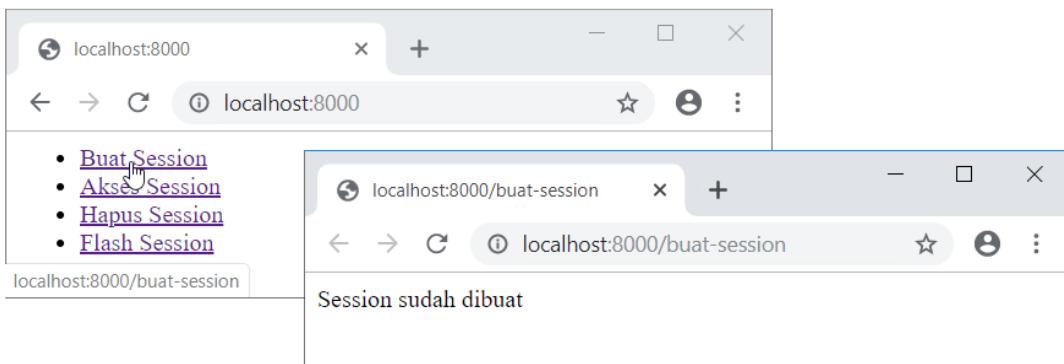
```
session(['siswa'=>['Anto', 'Rudi', 'Elsa', 'Friska']]);
```

Mengenai cara mengakses session akan kita bahas sesaat lagi. Sebagai bahan praktek, silahkan tambah kode berikut ke dalam method `buatSession()`:

app/Http/Controllers/SessionController.php

```
1 ...  
2 public function buatSession()  
3 {  
4     session(['hakAkses' => 'admin', 'nama' => 'Anto']);  
5     return "Session sudah dibuat";  
6 }
```

Kemudian akses alamat localhost:8000/buat-session di web browser atau buka dari link di halaman root sebelumnya:



Gambar: Membuat session

Dengan membuka alamat tersebut, maka session `hakAkses` dan `nama` sudah tersedia.

22.3. Membaca Session

Sama seperti pembuatan, proses pembacaan session juga bisa dilakukan dengan beberapa cara:

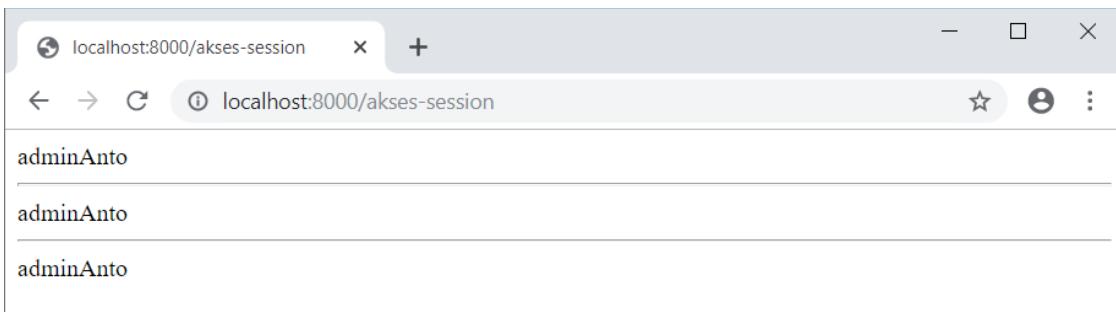
- ◆ Menggunakan function `helper session(<session_name>)`
- ◆ Melalui method `$request->session()->get(<session_name>)` dari **Request** object
- ◆ Melalui method `Session::get(<session_name>)` dari **Session** facade

Sebagai contoh, untuk mengakses kedua session yang sudah kita buat sebelumnya, bisa menggunakan kode berikut:

app/Http/Controllers/SessionController.php

```

1  ...
2  use Illuminate\Support\Facades\Session;
3
4  public function aksesSession(Request $request)
5  {
6      // Menggunakan helper function
7      echo session('hakAkses');
8      echo session('nama');
9
10     echo '<hr>';
11
12     // Dari Request object
13     echo $request->session()->get('hakAkses');
14     echo $request->session()->get('nama');
15
16     echo '<hr>';
17
18     // Menggunakan facade Session
19     echo Session::get('hakAkses');
20     echo Session::get('nama');
21 }
```



Gambar: Hasil pembacaan session

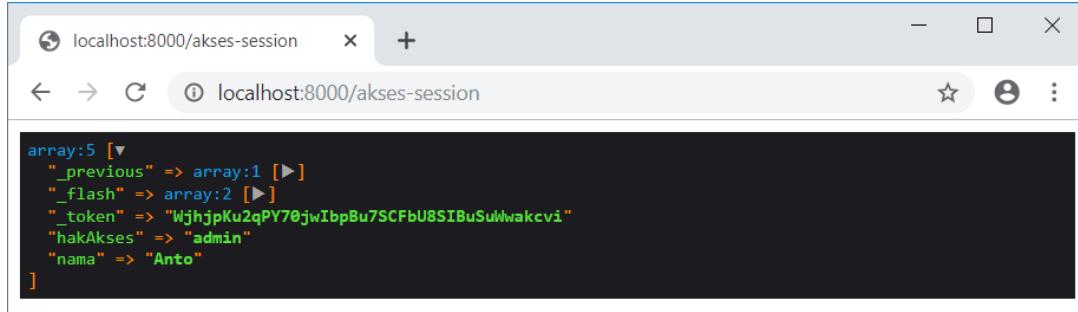
Hasilnya, tampil teks "admin" dan "Anto" yang berasal dari session `hakAkses` dan `nama`.

Untuk mengakses semua isi session, tersedia method `session()->all()` seperti contoh berikut:

Session

app/Http/Controllers/SessionController.php

```
1 public function aksesSession(Request $request)
2 {
3     dump( session()->all() );
4 }
```



Gambar: Hasil dump dari session()->all()

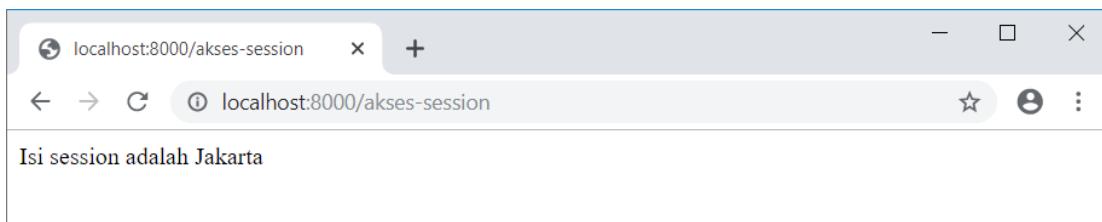
Dalam kode program ini saya langsung men-dump `session()->all()`. Hasilnya semua session ditampilkan dalam bentuk array. Selain session `hakAkses` dan `nama`, ternyata ada 3 session lain, yakni `_previous`, `_flash`, dan `_token`. Ketiga session ini di-generate otomatis oleh Laravel.

Beberapa fitur bawaan Laravel juga memanfaatkan session untuk menyimpan data, diantaranya proses menampilkan pesan kesalahan validasi dan menampilkan repopulate inputan form. Tambahan session ini juga bisa di lihat dari `session()->all()`.

Fitur lanjutan method `get()` adalah, bisa memberikan nilai default. Jika sebuah session tidak ditemukan, nilai inilah yang akan dipakai. Nilai default ditulis sebagai argument kedua dari method `get()`:

app/Http/Controllers/SessionController.php

```
1 public function aksesSession(Request $request)
2 {
3     $isiSession = $request->session()->get('kota', 'Jakarta');
4     echo "Isi session adalah $isiSession";
5 }
```



Gambar: Nilai default untuk session 'kota'

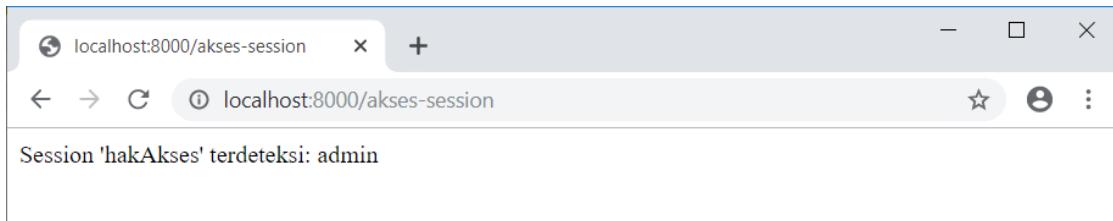
Di baris 3, variabel `$isiSession` akan berisi string "`Jakarta`" karena session `kota` memang tidak kita buat sebelumnya.

Dalam beberapa situasi, kita ingin melakukan pemeriksaan session terlebih dahulu, yakni jika session ada, jalankan kode tertentu. Ada atau tidaknya sebuah session bisa diperiksa

menggunakan method `session()->has()` seperti contoh berikut:

app/Http/Controllers/SessionController.php

```
1 public function aksesSession(Request $request)
2 {
3     if (session()->has('hakAkses'))
4     {
5         echo "Session 'hakAkses' terdeteksi: ". session('hakAkses');
6     }
7 }
```



Gambar: Hasil pemeriksaan session

Method `session()->has('hakAkses')` akan mengembalikan nilai `true` jika session dengan nama 'hakAkses' tersedia dan bisa diakses.

22.4. Menghapus Session

Untuk menghapus session juga bisa dilakukan dengan 3 cara:

- ◆ Menggunakan `function helper session->forget(<session_name>)`
- ◆ Melalui method `$request->session()->forget(<session_name>)` dari **Request** object
- ◆ Melalui method `Session::forget(<session_name>)` dari **Session** facade

Berikut contoh penggunaannya:

app/Http/Controllers/SessionController.php

```
1 ...
2 use Illuminate\Support\Facades\Session;
3
4 public function hapusSession(Request $request)
5 {
6     // Menghapus 1 session menggunakan helper function
7     session()->forget('hakAkses');
8
9     // Menghapus 1 session menggunakan Request object
10    $request->session()->forget('hakAkses');
11
12    // Menghapus 1 session menggunakan facade Session
13    Session::forget('hakAkses');
14
15    echo "Session hakAkses sudah dihapus";
16 }
```

Tentunya kita tidak harus menggunakan ketiga cara ini secara bersamaan, tapi cukup pilih salah satu saja.

Laravel juga menyediakan method `flush()` untuk menghapus semua session:

app/Http/Controllers/SessionController.php

```

1  ...
2  use Illuminate\Support\Facades\Session;
3
4  public function hapusSession(Request $request)
5  {
6      // Menghapus semua session menggunakan helper function
7      session()->flush();
8
9      // Menghapus semua session menggunakan Request object
10     $request->session()->flush();
11
12     // Menghapus semua session menggunakan facade Session
13     Session::flush();
14
15     echo "Semua session sudah dihapus";
16 }
```

Meskipun method `flush()` akan menghapus semua session, tapi jika kita cek kembali dengan method `session()->all()`, Laravel otomatis akan menggenerate kembali session `_previous`, `_flash`, dan `_token`.

22.5. Flash Session

Flash session adalah sebutan untuk session yang hanya bisa diakses 1 kali saja, setelah itu isinya langsung terhapus. Flash session ini sebenarnya sudah pernah kita pakai saat menampilkan flash data di bab CRUD.

Untuk membuat flash session, juga bisa dengan 3 cara berikut:

- ◆ Menggunakan function helper `session()->flash(<session_name>, <session_value>)`
- ◆ Melalui method `$request->session()->flash(<session_name>, <session_value>)` dari **Request** object
- ◆ Melalui method `Session::flash(<session_name>, <session_value>)` dari **Session** facade

Berikut contoh penggunaannya:

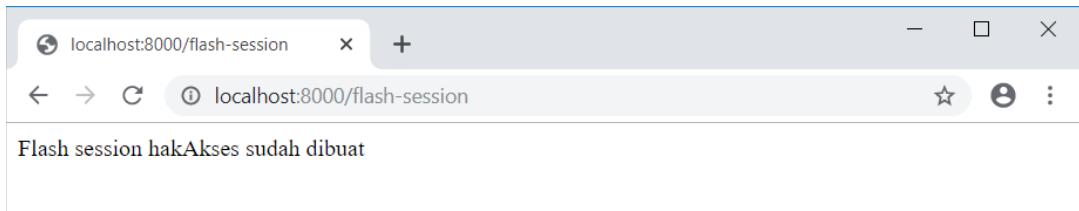
app/Http/Controllers/SessionController.php

```

1  ...
2  use Illuminate\Support\Facades\Session;
3
```

Session

```
4  public function flashSession(Request $request)
5  {
6      // Membuat 1 flash session menggunakan helper function
7      session()->flash('hakAkses', 'admin');
8
9      // Membuat 1 flash session menggunakan Request object
10     $request->session()->flash('hakAkses', 'admin');
11
12     // Membuat 1 flash session menggunakan facade Session
13     Session::flash('hakAkses', 'admin');
14
15     echo "Flash session hakAkses sudah dibuat";
16 }
```



Gambar: Pembuatan flash session

Setelah sebuah flash session dibuat, maka hanya bisa diakses 1 kali saja, misalnya menggunakan perintah `echo session('hakAkses')`. Jika perintah ini dijalankan untuk kali kedua, session `hakAkses` sudah tidak tersedia lagi.

Cara lain untuk membuat flash session juga bisa dari method `with()` yang di chaining dengan function `redirect()`, seperti contoh yang pernah kita pakai pada bab CRUD:

```
redirect()->route('mahasiswa.index')->with('pesan', "Penambahan data berhasil");
```

Banyaknya cara yang disediakan Laravel memang kadang membuat bingung, namun ini juga salah satu sisi fleksibilitas dari Laravel. Anda bisa luangkan waktu sejenak untuk mencoba berbagai method session yang sudah kita pelajari.

Dalam bab ini kita telah membahas tentang pemrosesan session di dalam Laravel. Session sendiri memegang peranan penting di banyak fitur web programming, terutama proses login dan logout. Dan berikutnya kita akan buat studi kasus untuk hal ini.

23. Case Study: Login Middleware

Bab kali ini lebih ke latihan dari konsep **middleware** dan **session** yang baru saja kita pelajari. Ideanya adalah, saya ingin membatasi akses ke beberapa halaman menggunakan middleware. Middleware tersebut akan memeriksa apakah user sudah login atau belum. Jika belum, blok hak akses dan redirect ke form login.

Proses pembatasan login ini dilakukan dengan cara memeriksa apakah sebuah session tersedia atau tidak. Jika session ditemukan, artinya user sudah login dan halaman bisa diakses. Namun jika session tidak ditemukan, tolak request dan redirect user ke form login.

Fitur login/logout seperti ini sebenarnya sudah ada di Laravel, yakni dari fitur **Authentication** yang akan di bahas pada bab berikutnya, namun tidak ada salahnya kita buat versi manual agar bisa lebih memahami dasar middleware dan session Laravel.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Setelah itu input file **bootstrap.min.css** ke dalam folder **public/css**, serta file **jquery-3.3.1.slim.min.js**, **popper.min.js** dan **bootstrap.min.js** ke dalam folder **public/js**, karena kita akan memakai sedikit kode Bootstrap.

23.1. Pembuatan Route

Hampir semua materi yang akan kita pakai sudah di bahas sebelumnya, oleh karena itu saya akan langsung menampilkan file akhir dari project "**login middleware**".

Kita akan mulai dari route terlebih dahulu:

```
routes/web.php
```

```
1 <?php  
2  
3 use Illuminate\Support\Facades\Route;
```

```
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/login', [MahasiswaController::class, 'login']);
7 Route::post('/login', [MahasiswaController::class, 'prosesLogin']);
8
9 Route::get('/logout', [MahasiswaController::class, 'logout']);
10
11 Route::redirect('/', '/login');
12
13 Route::get('/daftar-mahasiswa', [MahasiswaController::class, 'daftarMahasiswa'])
14 ->middleware('login');
15
16 Route::get('/tabel-mahasiswa', [MahasiswaController::class, 'tabelMahasiswa'])
17 ->middleware('login');
18
19 Route::get('/blog-mahasiswa', [MahasiswaController::class, 'blogMahasiswa'])
20 ->middleware('login');
```

Semua route ini akan mengakses method milik `MahasiswaController`, yang akan di buat sesaat lagi.

Route di baris 6 dipakai untuk menampilkan form login, yang ketika di submit akan di proses oleh route di baris 7. Keduanya memakai alamat '/login' dengan perbedaan di method, yakni `get` untuk menampilkan form, dan `post` untuk pemrosesan form.

Route di baris 9 ditujukan untuk proses logout. Sedangkan route di baris 11 hanya sekedar `redirect`, yakni jika halaman root di akses, langsung di redirect ke halaman '/login'.

Tiga route terakhir di baris 13 – 20 merupakan halaman yang akan kita proteksi (hanya bisa diakses jika user sudah login). Perhatikan tambahan method `->middleware('login')` di akhir setiap route, inilah cara yang saya pakai untuk membatasi hak akses.

23.2. Pembuatan Controller

Seperi biasa, proses pembuatan `MahasiswaController` bisa dilakukan dengan kode berikut:

```
php artisan make:controller MahasiswaController
```

Dan berikut kode program untuk `MahasiswaController`:

```
app/Http/Controllers/MahasiswaController.php
```

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     public function daftarMahasiswa()
```

Case Study: Login Middleware

```
10     {
11         return view('halaman',[ 'judul' => 'Daftar Mahasiswa']);
12     }
13
14     public function tabelMahasiswa()
15     {
16         return view('halaman',[ 'judul' => 'Tabel Mahasiswa']);
17     }
18
19     public function blogMahasiswa()
20     {
21         return view('halaman',[ 'judul' => 'Blog Mahasiswa']);
22     }
23
24     public function login()
25     {
26         return view('form-login');
27     }
28
29     public function prosesLogin(Request $request)
30     {
31         $request->validate([
32             'username' => 'required',
33         ]);
34
35         $validUsername = [ 'andi','rani','lisa','joko'];
36         // Jika inputan username ada di array, buat session 'username'
37         if (in_array($request->username, $validUsername))
38         {
39             session(['username' => $request->username]);
40             return redirect('/daftar-mahasiswa');
41         }
42         else {
43             // Username tidak ada di daftar
44             return back()->withInput()->with( 'pesan',"Username tidak valid");
45         }
46     }
47
48     public function logout()
49     {
50         // Hapus session 'username'
51         session()->forget('username');
52         return redirect('login')->with( 'pesan', 'Logout berhasil');
53     }
54 }
```

Saya juga menampilkan kode lengkap dari `MahasiswaController` yang akan kita bahas secara bertahap.

Tiga method pertama, yakni `daftarMahasiswa()`, `tabelMahasiswa()` dan `blogMahasiswa()` dipakai untuk mengakses view yang sama, yakni `halaman.blade.php`. Selain itu, ketiganya juga mengirim variabel `judul` dengan yang nilai string berbeda. Variabel judul ini bisa kita akses dari dalam view `halaman.blade.php`.

Method `login()` di baris 24 dipakai untuk menampilkan view `form-login.blade.php`. Inilah view yang akan menampilkan form login.

Di baris 29 – 46 terdapat method `prosesLogin()` yang cukup panjang. Method ini dipakai untuk memproses hasil submit form login. Di bagian awal, terdapat perintah validasi `'username' => 'required'` yang akan memastikan inputan `username` tidak boleh kosong. Artinya, di dalam form login nanti harus ada inputan form dengan kode `<input type="text" name="username">`.

Jika validasi terpenuhi, maka kode di baris 35 – 45 akan di proses. Yang dimulai dengan membuat array `$validUsername = ['andi', 'rani', 'lisa', 'joko']`. Apa fungsi dari array ini? Untuk menyederhanakan contoh kita, hanya `username` yang ada di array `$validUsername` saja yang saya izinkan untuk login. Pemeriksaan ini di lakukan dengan perintah `if (in_array($request->username, $validUsername))` di baris 37.

Function `in_array()` adalah fungsi bawaan PHP yang akan mengembalikan `true` jika nilai `$request->username` ada di dalam array `$validUsername`. Nilai `$request->username` sendiri nantinya berasal dari inputan user pada saat mengisi form login.

Jika user yang login adalah salah satu dari `'andi', 'rani', 'lisa'` atau `'joko'`, maka lanjut dengan membuat session bernama `username` yang diisi dengan nilai `$request->username`. Setelah itu user akan di redirect ke halaman `'/daftar-mahasiswa'`.

Namun jika nama user tidak ditemukan dalam array `$validUsername`, kode program akan masuk ke blok `else` di baris 44. Di sini user akan di redirect ke halaman login dengan pesan `'Username tidak valid'`.

Perintah `return back()` merupakan cara lain untuk proses redirect. Dimana user akan di redirect ke halaman sebelumnya, atau mirip seperti kita menekan tombol **back** di web browser. Method `with()` sendiri dipakai untuk membuat *flash session*.

Terakhir, method `logout()` di baris 48 – 53 akan menghapus session `'username'` dengan perintah `session()->forget('username')`, lalu user di redirect ke halaman login dengan pesan flash `'Logout berhasil'`.

Dari penjelasan ini, kurang lebih kita sudah bisa membaca alur program yang akan dijalankan. Ini memang bukan sebuah proses login dan logout "yang sebenarnya", karena validasi user hanya saya lakukan dengan mencocokkan nama user dengan isi array `$validUsername`. Proses ini bisa saja diganti dengan logika lain yang lebih kompleks, seperti mencocokkan nama user yang login dengan daftar user di tabel `admin`.

23.3. Pembuatan Middleware

Untuk membatasi hak akses ke halaman tertentu, akan ditangani oleh middleware yang saya

beri nama CekLogin. Berikut perintah php artisan untuk membuatnya:

```
php artisan make:middleware CekLogin
```

Dan berikut kode program ini middleware ini:

app/Http/Middleware/CekLogin.php

```
1 <?php
2
3 namespace App\Http\Middleware;
4
5 use Closure;
6 use Illuminate\Http\Request;
7
8 class CekLogin
9 {
10     /**
11      * Handle an incoming request.
12      *
13      * @param \Illuminate\Http\Request $request
14      * @param \Closure $next
15      * @return mixed
16      */
17     public function handle(Request $request, Closure $next)
18     {
19         if (session()->has('username'))
20         {
21             return $next($request);
22         }
23         else {
24             return redirect('/login')->with('pesan', "Maaf,
25             silahkan login terlebih dahulu");;
26         }
27     }
28 }
```

Tambahan kode program ada di baris 19 – 26 yang diawali dengan pemeriksaan kondisi apakah session 'username' di temukan atau tidak. Jika ditemukan (berarti user sudah login), maka middleware CekLogin akan meneruskannya ke proses lain, yakni jalankan perintah `return $next($request)`.

Namun jika session 'username' tidak ditemukan, blok akses dengan cara me-redirect user ke halaman '/login' beserta pesan "Maaf, silahkan login terlebih dahulu".

Urusan kita belum selesai, karena sebuah middleware harus didaftarkan terlebih dahulu ke file Kernel.php. Saya ingin menjadikan CekLogin ini sebagai optional middleware, yakni middleware yang baru akan aktif jika ditambahkan secara manual ke route atau controller. Caranya, tulis CekLogin ke dalam property `$routeMiddleware`:

Case Study: Login Middleware

app/Http/Kernel.php

```
1 <?php
2
3 namespace App\Http;
4
5 use Illuminate\Foundation\Http\Kernel as HttpKernel;
6
7 class Kernel extends HttpKernel
8 {
9     ...
10    protected $routeMiddleware = [
11        'auth' => \App\Http\Middleware\Authenticate::class,
12        ...
13        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
14        'login' => \App\Http\Middleware\CekLogin::class,
15    ];
16 }
```

Di baris 14 saya membuat key 'login' untuk middleware CekLogin. Key ini sudah kita pasang ke dalam route untuk membatasi akses ke halaman '/daftar-mahasiswa', '/tabel-mahasiswa' dan '/blog-mahasiswa'.

23.4. Pembuatan View

Untuk mini project ini saya butuh 2 buah view: `form-login.blade.php` untuk membuat form login, serta `halaman.blade.php` untuk membuat view halaman.

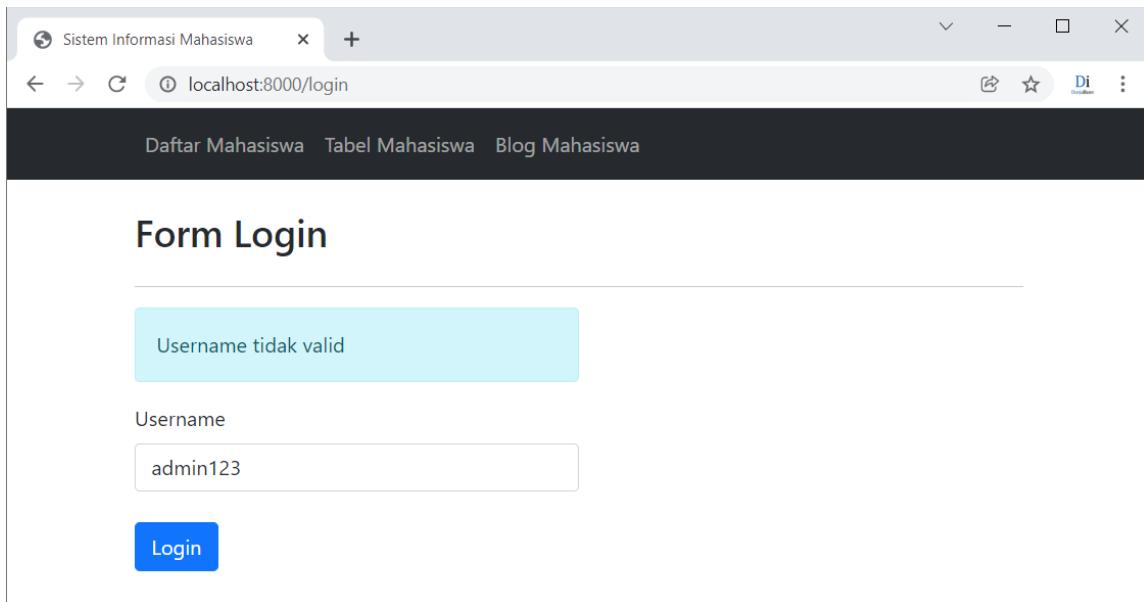
Berikut kode untuk view `form-login.blade.php`:

resources/views/form-login.blade.php

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link href="/css/bootstrap.min.css" rel="stylesheet">
8     <title>Sistem Informasi Mahasiswa</title>
9 </head>
10 <body>
11
12 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13     <div class="container">
14         <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
15             data-bs-target="#navbar">
16             <span class="navbar-toggler-icon"></span>
17         </button>
18         <div class="collapse navbar-collapse" id="navbar">
19             <ul class="navbar-nav">
20                 <li class="nav-item">
```

Case Study: Login Middleware

```
21      <a class="nav-link" href="{{url('/daftar-mahasiswa')}}">
22          Daftar Mahasiswa
23      </a>
24  </li>
25  <li class="nav-item">
26      <a class="nav-link" href="{{url('/tabel-mahasiswa')}}">
27          Tabel Mahasiswa
28      </a>
29  </li>
30  <li class="nav-item">
31      <a class="nav-link" href="{{url('/blog-mahasiswa')}}">
32          Blog Mahasiswa
33      </a>
34  </li>
35  </ul>
36  </div>
37 </div>
38 </nav>
39
40 <div class="container">
41     <h2 class="my-4">Form Login</h2>
42     <hr>
43
44     @if(session()->has('pesan'))
45     <div class="alert alert-info w-50">
46         {{ session()->get('pesan') }}
47     </div>
48     @endif
49
50     <form action="{{url('/login')}}" method="POST">
51         @csrf
52         <div class="mb-3">
53             <label for="username" class="form-label">Username</label>
54             <input type="text" class="form-control w-50" id="username"
55                 name="username" value="{{ old('username') }}">
56             @error('username')
57                 <div class="text-danger">{{ $message }}</div>
58             @enderror
59         </div>
60         <button type="submit" class="btn btn-primary my-2">Login</button>
61     </form>
62 </div>
63
64 <script src="/js/bootstrap.bundle.min.js"></script>
65 </body>
66 </html>
```



Gambar: Form Login

Sama seperti view pada umumnya, di sini lebih banyak kode HTML dan CSS. Agar tampilan lebih menarik, saya juga menggunakan beberapa komponen Bootstrap.

Di baris 1 sampai 9, berisi struktur awal file HTML, tidak ada perubahan mendasar di bagian ini. Kemudian antara baris 12 – 38 merupakan kode untuk membuat menu navigasi di bagian atas halaman. Di sini saya menggunakan komponen **navbar** bawaan Bootstrap.

Navbar ini terdiri dari 3 menu yang menuju ke halaman `/daftar-mahasiswa`, `/tabel-mahasiswa`, dan `/blog-mahasiswa`.

Lanjut, di baris 44 terdapat pemeriksaan kondisi `@if(session()->has('pesan'))`. Artinya saya ingin memeriksa apakah ada session 'pesan' yang berasal dari halaman lain. Jika anda lihat kembali kode program di controller dan middleware, session pesan bisa berasal dari beberapa proses, misalnya jika user tidak terdaftar (tidak ada di dalam array), atau sekedar menampilkan pesan kalau user sudah logout.

Form login sendiri di tulis antara baris 50 – 61. Di sini terdapat sebuah inputan `<input type="text" name="username">` yang ketika di submit akan mengirim data ke route `'/login'`. Form ini juga dilengkapi kode untuk menampilkan pesan error di baris 56 – 58.

Di baris paling bawah terdapat tag `<script>` untuk mengaktifkan kode JavaScript Bootstrap. Tambahan script ini diperlukan agar komponen navbar Bootstrap bisa berjalan sempurna (membuat responsive navbar).

Dengan kode ini, kita sudah bisa melakukan beberapa hal, misalnya silahkan test submit form tanpa mengisi inputan username sama sekali, maka akan tampil pesan error validasi form. Kemudian test isi username sembarang (yang tidak ada di dalam array), maka akan tampil pesan 'Username tidak valid'.

Test juga klik salah satu menu halaman di bagian atas, akan tampil pesan 'Maaf, silahkan login terlebih dahulu'. Pesan error yang sama juga akan tampil jika kita mengetik manual alamat `localhost:8000/daftar-mahasiswa`, karena halaman ini sudah di proteksi dengan middleware `CekLogin`.

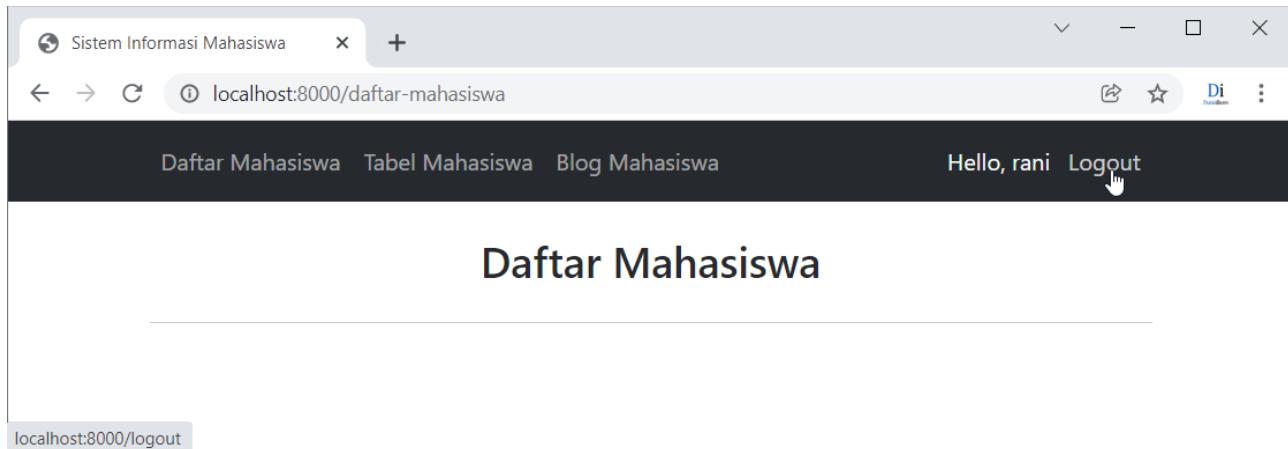
Baik, saatnya kode program untuk view `halaman.blade.php`:

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link href="/css/bootstrap.min.css" rel="stylesheet">
8      <title>Sistem Informasi Mahasiswa</title>
9  </head>
10 <body>
11
12 <nav class="navbar navbar-expand-sm navbar-dark bg-dark">
13     <div class="container">
14         <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
15             data-bs-target="#navbar">
16             <span class="navbar-toggler-icon"></span>
17         </button>
18         <div class="collapse navbar-collapse d-md-flex justify-content-between"
19             align-items-center" id="navbar">
20             <ul class="navbar-nav me-auto">
21                 <li class="nav-item">
22                     <a class="nav-link" href="{{url('/daftar-mahasiswa')}}">
23                         Daftar Mahasiswa
24                     </a>
25                 </li>
26                 <li class="nav-item">
27                     <a class="nav-link" href="{{url('/tabel-mahasiswa')}}">
28                         Tabel Mahasiswa
29                     </a>
30                 </li>
31                 <li class="nav-item">
32                     <a class="nav-link" href="{{url('/blog-mahasiswa')}}">
33                         Blog Mahasiswa
34                     </a>
35                 </li>
36             </ul>
37             <ul class="navbar-nav mt-2 mt-md-0">
38                 <li class="nav-item">
39                     <span class="text-light">Hello, {{ session('username') }} </span>
40                     <a class="nav-link d-inline" href="{{url('/logout')}}">Logout</a>
41                 </li>
42             </ul>
43         </div>
44     </div>
45 </nav>
46

```

```
47 <div class="container">
48   <h2 class="text-center my-4" >{{$judul}}</h2>
49   <hr>
50 </div>
51
52 <script src="/js/bootstrap.bundle.min.js"></script>
53 </body>
54 </html>
```



Gambar: Tampilan halaman Daftar Mahasiswa

Mayoritas kode HTML dan CSS halaman ini sama seperti halaman `form-login`. Khusus untuk menu navbar, terdapat tambahan kode di baris 37 – 42, dimana terdapat menu baru untuk "menyapa" user yang sedang login. Nama user bisa di ambil dari session '`username`', ditambah link ke halaman `/logout` untuk proses logout.

View halaman `blade.php` ini sebenarnya akan diakses oleh 3 method di `MahasiswaController`, yakni `daftarMahasiswa()`, `tabelMahasiswa()` dan `blogMahasiswa()`. Yang berubah hanya data variabel `judul` yang dikirim dari method tersebut. Variabel ini akan ditampilkan di baris 47 dengan perintah `{{$judul}}`, sehingga akan berganti-ganti sesuai halaman yang dibuka.

Ini memang lebih sebagai halaman *sample / dummy* saja. Dalam prakteknya nanti, bisa saja ketika halaman `daftar-mahasiswa` di buka, tampilkan semua data mahasiswa dari database.

Dalam bab ini kita membuat sebuah praktek sederhana proses login dan logout dengan memanfaatkan session dan middleware Laravel. Meskipun sederhana, tapi memakai cukup banyak fitur Laravel, diantaranya: *route*, *controller*, *view*, *validasi form*, *session*, dan juga *middleware*.

Berikutnya, kita akan lihat fitur login dan logout bawaan Laravel, yang tentu saja lebih canggih dari contoh praktek ini.

24. Authentication

Salah satu alasan **Taylor Otwell** mengembangkan Laravel karena pada saat itu framework Code Igniter tidak memiliki fitur *authentication* bawaan. **Authentication** adalah sebutan untuk proses registrasi user, termasuk mekanisme login dan logout. Hampir semua project skala menengah ke atas butuh sistem seperti ini.

Oleh sebab itulah Laravel menyediakan fitur *authentication* bawaan yang sangat powerfull namun juga sedikit rumit karena banyaknya hal 'magic' di Laravel. Maksudnya, untuk bisa memodifikasi proses authentication, kita harus pelajari semua mekanisme yang dipakai Laravel.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Dalam bab ini kita juga akan menjalankan ulang proses migration, maka jika ada tabel di database **laravel**, silahkan hapus terlebih dahulu.

24.1. Instalasi Laravel Authentication

Di Laravel 9, terdapat berbagai modul untuk membuat fitur authentication. Diantaranya [Laravel UI](#), [Laravel Breeze](#), dan [Laravel Jetstream](#). Perbedaan dari ketiganya ada di library front-end yang dipakai, serta kelengkapan fitur.

Laravel UI menggunakan framework CSS **Bootstrap** sebagai basis desain tampilan, sedangkan **Laravel Breeze** menggunakan framework **Tailwind CSS**. Keduanya menawarkan fitur dasar authentication seperti registrasi user, login/logout, password reset, email verification, dan password confirmation.

Di sisi lain, **Laravel Jetstream** menyediakan fitur yang paling lengkap. Selain apa yang terdapat di UI dan Breeze, Jetstream menawarkan tambahan two factor authentication, session management, serta API support via Laravel Sanctum. Namun ini juga sebanding dengan kompleksitas library front-end yang dipakai, dimana selain menggunakan TailwindCSS,

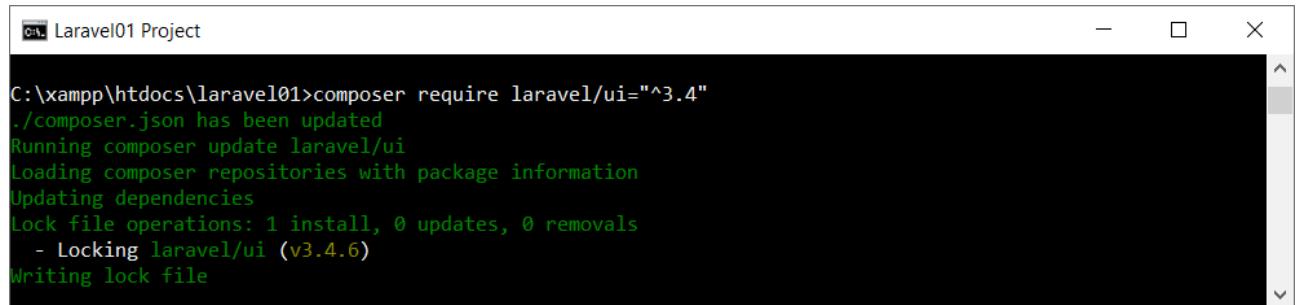
Jetstream juga butuh [Laravel Livewire](#) dan [Inertia](#).

Jadi apa yang mesti dipakai?

Ini tergantung kebutuhan dan skill dasar yang sudah dimiliki. Karena lebih familiar dengan Bootstrap, maka saya akan memakai **Laravel UI** dalam bab ini. Jika anda sudah pernah belajar Tailwind CSS, tidak ada salahnya nanti mencoba Laravel Breeze. Khusus untuk Jetstream, lebih saya sarankan bagi yang sudah paham dengan Livewire atau Intertia.

Untuk instalasi Laravel UI, masuk ke folder laravel01 dari cmd, lalu jalankan perintah berikut:

```
composer require laravel/ui="^3.4"
```



```
C:\xampp\htdocs\laravel01>composer require laravel/ui="^3.4"
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking laravel/ui (v3.4.6)
Writing lock file
```

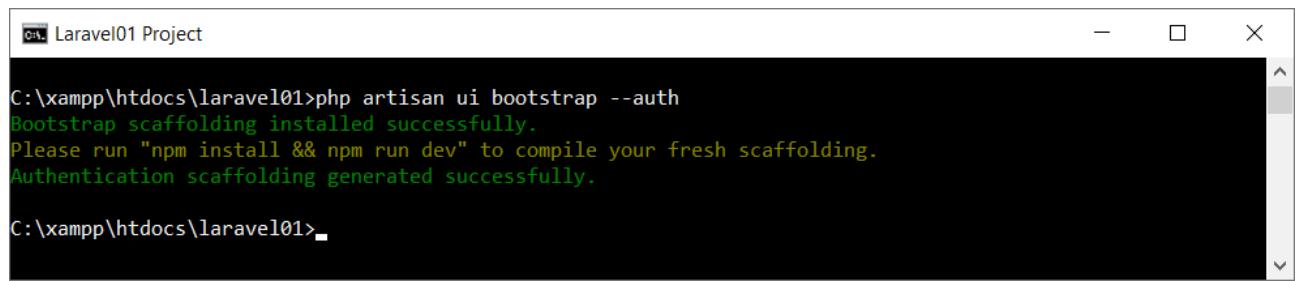
Gambar: Instalasi Laravel UI

Setelah itu kita harus memilih salah satu *preset* yang tersedia, yakni **vue**, **react** atau **bootstrap**. Karena tidak menggunakan vue dan react, saya akan memilih bootstrap saja.

Jika anda memilih *preset* vue atau react, sebenarnya juga tetap menggunakan bootstrap untuk kode tampilan, plus tambahan file kode khusus untuk vue dan react. Sehingga meskipun memilih *preset* vue atau react untuk Laravel UI, materi yang akan kita bahas masih tetap sama.

Untuk instalasi Laravel Authentication, tambah option `--auth` di bagian akhir, seperti perintah berikut:

```
php artisan ui bootstrap --auth
```



```
C:\xampp\htdocs\laravel01>php artisan ui bootstrap --auth
Bootstrap scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Instalasi preset bootstrap

Setelah selesai, jalankan 3 perintah berikut secara berurutan untuk instalasi module front-end Laravel, install Laravel Mix, dan compile file asset:

Authentication

```
npm install  
npm install laravel-mix --save-dev  
npx mix
```

Seperti biasa, perintah `npx mix` mungkin perlu dijalankan 2 kali. Berikut tampilan saat proses compile file asset dengan Laravel Mix sudah selesai:



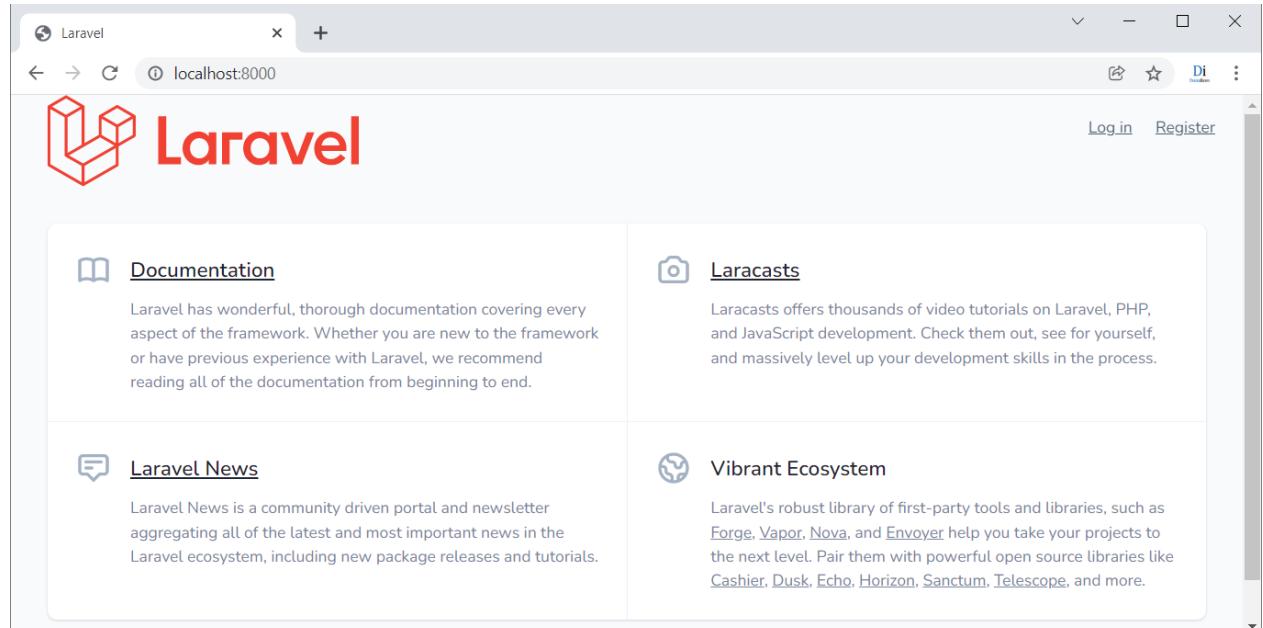
```
Laravel01 Project  
Laravel Mix v6.0.49  
  
Compiled Successfully in 6856ms  


|  | File        | Size     |
|--|-------------|----------|
|  | /js/app.js  | 2.23 MiB |
|  | css/app.css | 200 KiB  |

  
1 WARNING in child compilations (Use 'stats.children: true' resp. '--stats-children' for more details)  
webpack compiled with 1 warning  
C:\xampp\htdocs\laravel01>
```

Gambar: Proses compile selesai

Sip, proses instalasi Laravel Authentication sudah selesai. Untuk memastikan, silahkan buka halaman home di localhost:8000:



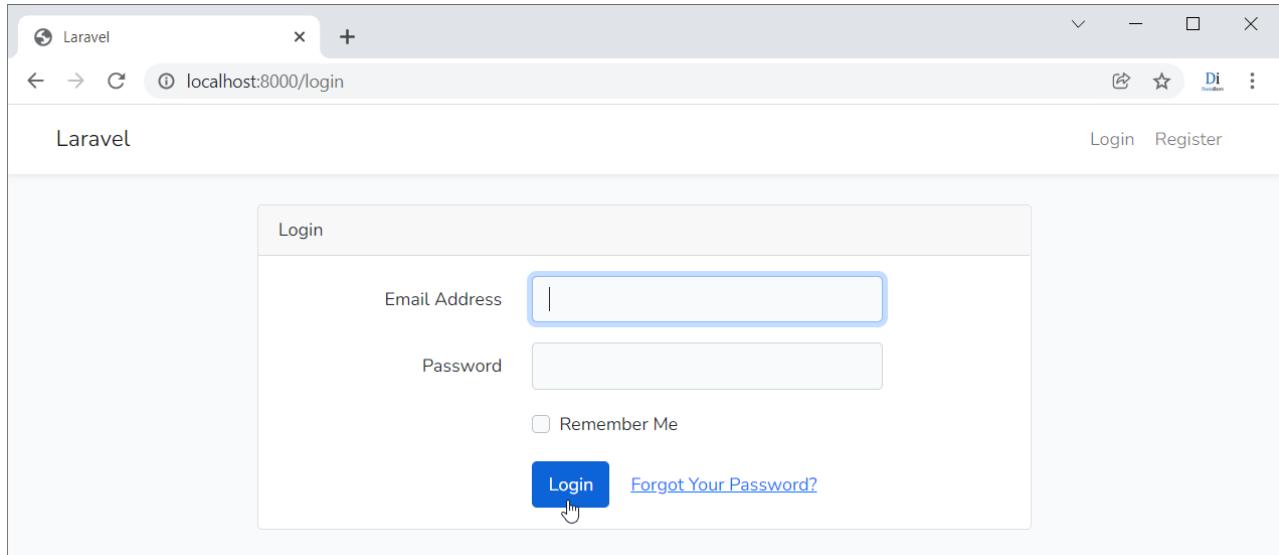
Gambar: Tampilan halaman home Laravel

Seklas tidak ada perbedaan yang berarti, halaman home tetap tampil seperti yang kita lihat selama ini. Namun perhatikan di sudut kanan atas, terdapat 2 link tambahan untuk **login** dan **register**. Inilah sistem authentication yang baru saja di install ke dalam Laravel.

Authentication

Proses instalasi Laravel Authentication akan merombak ulang seluruh kode halaman home (file `welcome.blade.php`), termasuk jika halaman tersebut sudah di modifikasi sebelumnya. Oleh karena itu sangat disarankan untuk menginstall authentication di awal project agar tidak menimpa view yang sudah ada.

Silahkan klik menu Login:

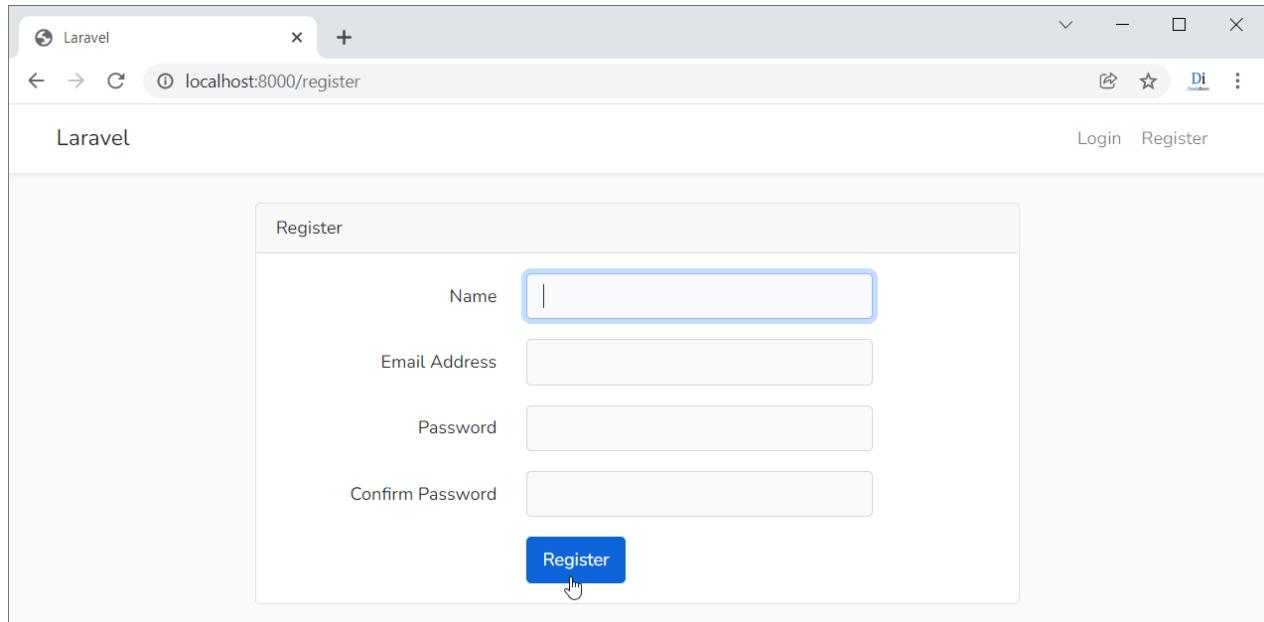


The screenshot shows a browser window titled "Laravel". The address bar displays "`localhost:8000/login`". The main content area shows a "Login" form. The form includes fields for "Email Address" and "Password", a "Remember Me" checkbox, and two buttons: "Login" and "Forgot Your Password?". The "Login" button is highlighted with a blue background and white text, indicating it is the active button.

Gambar: Halaman login

Secara bawaan, form login akan meminta alamat email dan password, lengkap dengan checkbox *remember me* yang umum kita jumpai di berbagai website. Selain itu juga ada link untuk reset password.

Dan berikut tampilan halaman Register:



The screenshot shows a browser window titled "Laravel". The address bar displays "`localhost:8000/register`". The main content area shows a "Register" form. The form includes fields for "Name", "Email Address", "Password", and "Confirm Password", and a single "Register" button. The "Register" button is highlighted with a blue background and white text, indicating it is the active button.

Gambar: Halaman register

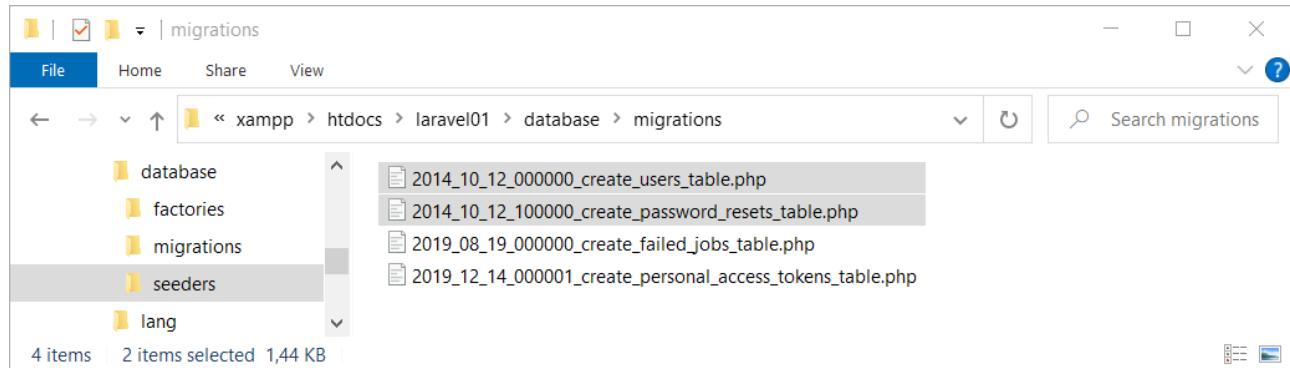
Bawaan Laravel, inputan untuk form register terdiri dari `name`, `email`, `password` serta `confirm password`. Nantinya kita bisa tambah dengan inputan lain sesuai dengan data yang diperlukan.

Jika kita coba login atau register saat ini, akan tampil pesan error karena proses authentication butuh koneksi database. Selain itu kita juga perlu membuat tabel untuk menampung data user.

24.2. Users Table

Tabel untuk proses *authentication* ini sebenarnya sudah ada di Laravel sejak awal. Apakah anda masih ingat tabel `users` yang di generate Laravel pada saat pertama kali kita mempelajari migration? Itulah tabel yang dimaksud.

Silahkan buka folder `database\migrations\`, akan terdapat 4 file migration:



Gambar: File migration bawaan Laravel

Dua file pertama ditujukan untuk proses *authentication*, yakni tabel `users` dan tabel `password_resets`.

Berikut isi dari method `up()` yang ada di file `2014_10_12_000000_create_users_table.php`:

```

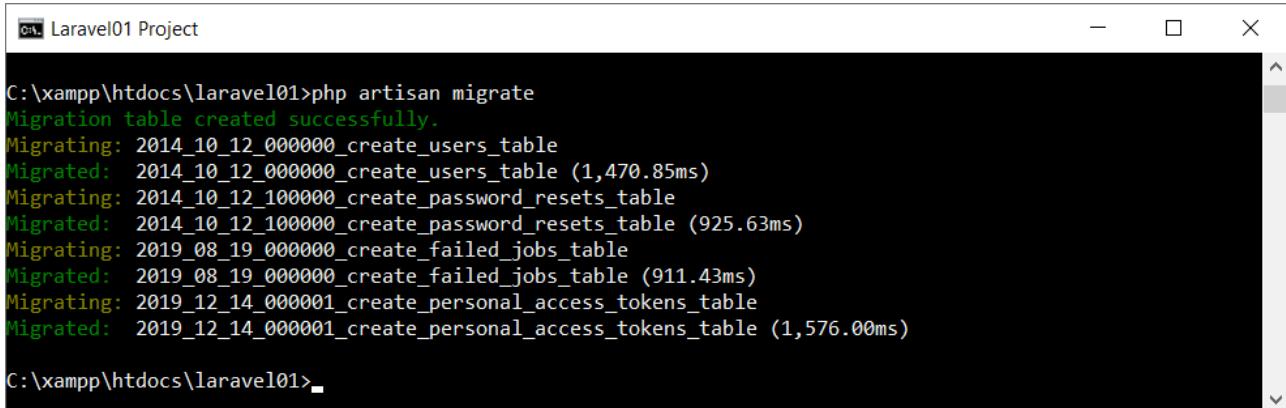
1  public function up()
2  {
3      Schema::create('users', function (Blueprint $table) {
4          $table->id();
5          $table->string('name');
6          $table->string('email')->unique();
7          $table->timestamp('email_verified_at')->nullable();
8          $table->string('password');
9          $table->rememberToken();
10         $table->timestamps();
11     });
12 }
```

Inilah daftar kolom yang ada di tabel `users`. Jika dibutuhkan, kita bisa menambah sendiri daftar kolom lain, misalnya jika user ini dianggap sebagai mahasiswa, perlu menambah kolom `nim`, `jurusan`, dst.

Untuk tabel `password_resets` saat ini belum akan saya bahas karena ingin fokus ke proses register, login dan logout terlebih dahulu.

Agar kita bisa melakukan proses registrasi, jalankan migration dengan perintah berikut (pastikan MySQL Server sudah aktif dan database `laravel` dalam keadaan kosong):

```
php artisan migrate
```

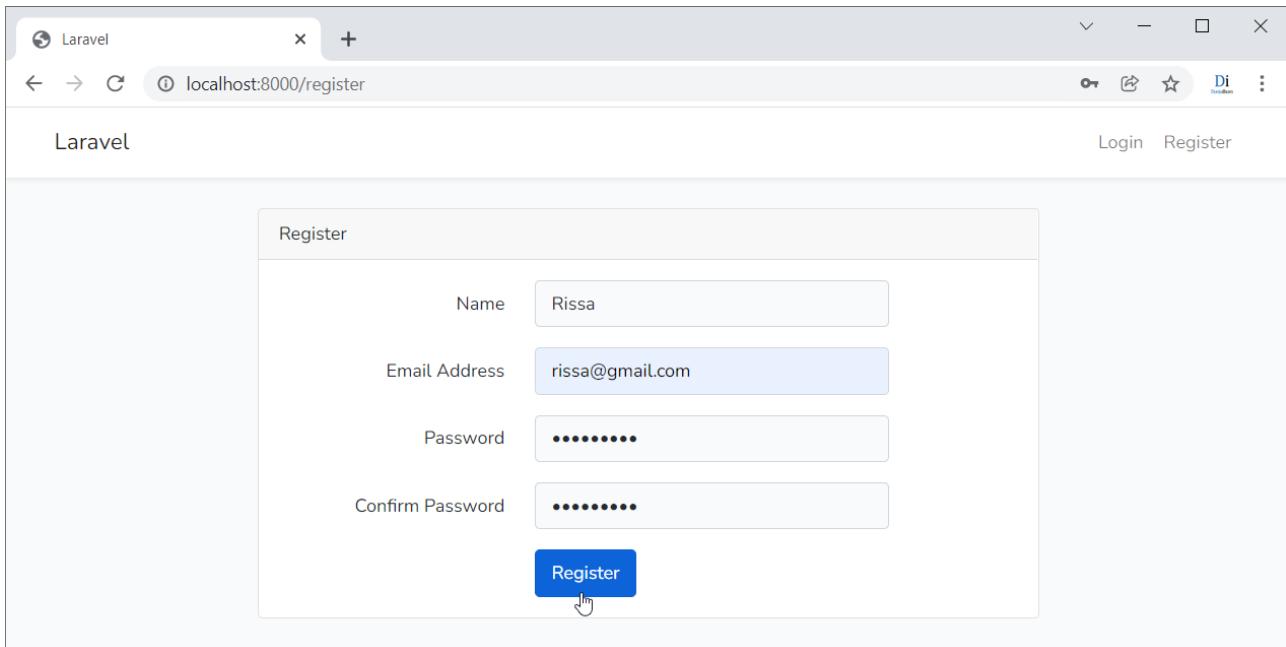


```
C:\xampp\htdocs\laravel01>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (1,470.85ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (925.63ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (911.43ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (1,576.00ms)

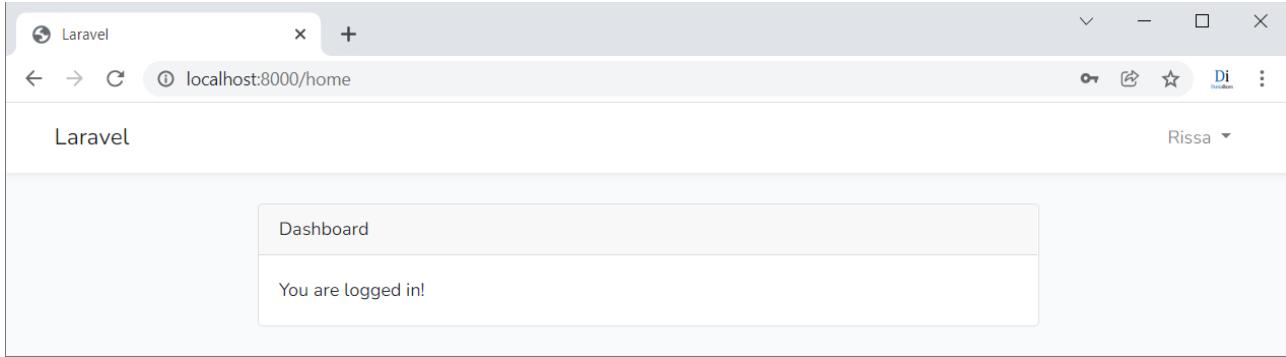
C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan migration

Sekarang kita sudah bisa melakukan register dan login. Silahkan buat satu akun melalui link Register:



Gambar: Register user baru



Gambar: Halaman home ketika sudah login

Setelah register, secara otomatis user akan di redirect ke halaman home (langsung login). Ini ditandai dengan tulisan "Dashboard" di tengah halaman, serta nama user di sisi kanan atas beserta tombol **logout**.

Anda bisa test logout, lalu login kembali dan kita akan sampai ke halaman yang sama.

24.3. File Laravel Authentication

Secara internal, Laravel authentication tetap menggunakan fitur dasar yang sudah kita pelajari sepanjang buku ini, mulai dari route, view, controller, model, migration, session, dan juga middleware. Mari lihat apa saja kode bawaan dari authentication.

File Route

Berikut isi route setelah proses install authentication:

```
routes/web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Auth::routes();
10
11 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
12 ->name('home');
```

Selain route untuk halaman welcome, terdapat tambahan `Auth::routes()` dan juga route untuk halaman '/home' yang akan mengakses method `index()` di `HomeController`.

Perintah `Auth::routes()` ini sebenarnya mewakili 11 route yang tidak tertulis. Untuk melihat daftar lengkapnya, jalankan perintah `php artisan route:list` di terminal VS Code:

Authentication

```
PS C:\xampp\htdocs\laravel01> php artisan route:list

GET|HEAD / ..... 
POST _ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user ..... 
GET|HEAD home ..... home > HomeController@index
GET|HEAD login ..... 1 login > Auth\LoginController@showLoginForm
POST login ..... 2 Auth\LoginController@login
POST logout ..... 3 logout > Auth\LoginController@logout
GET|HEAD password/confirm ..... password.confirm > Auth\ConfirmPasswordController@showConfirmForm
POST password/confirm ..... 4 Auth\ConfirmPasswordController@confirm
POST password/email ..... 5 password.email > Auth\ForgotPasswordController@sendResetLinkEmail
GET|HEAD password/reset ..... password.request > Auth\ForgotPasswordController@showLinkRequestForm
POST password/reset ..... 6 password.update > Auth\ResetPasswordController@reset
GET|HEAD password/reset/{token} ..... password.reset > Auth\ResetPasswordController@showResetForm
GET|HEAD register ..... register > Auth\RegisterController@showRegistrationForm
POST register ..... 7 register > Auth\RegisterController@register
GET|HEAD sanctum/csrf-cookie ..... Laravel\Sanctum > CsrfCookieController@show

PS C:\xampp\htdocs\laravel01>
```

Gambar: Daftar route Laravel

Tabel route ini lumayan lebar sehingga terpaksa saya kecilkan agar muat dalam 1 baris. Terlihat cukup banyak route yang dibawa oleh authentication dengan rincian sebagai berikut:

- ◆ 2 route untuk proses login (1).
- ◆ 1 route untuk proses logout (2).
- ◆ 2 route untuk konfirmasi password (3).
- ◆ 1 route untuk memproses link reset password yang akan dikirim ke email (4).
- ◆ 3 route untuk proses reset password (5).
- ◆ 2 route untuk proses register (6).

Dalam bab ini kita tidak akan membahas semua route, fokus utama hanya untuk proses login dan register saja. Untuk proses reset password dan konfirmasi ke email, rencana akan menjadi jatah seri **Laravel in Depth**.

Jika anda menggunakan VS Code, baris `Auth::routes()` mungkin ditampilkan error seperti gambar berikut:



```
5 use App\Http\Controllers\MahasiswaController;
6
7 Route::get('/', function () {
8     return view('welcome');
9 });
10
11 Auth::routes();
12
13 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home')
```

Gambar: Error VS Code di baris `Auth::routes()`

Sebenarnya baris tersebut bukan error, hanya saja extension PHP intelephense bawaan VS Code mendeteksi tidak ada pendefinisian class **Auth** dalam halaman ini, dan juga tidak ada perintah import namespace untuk class tersebut.

Ini tidak masalah karena Laravel punya mekanisme sendiri untuk mengakses class khusus tanpa perlu di import. Atau agar membuat VS Code "senang", silahkan tambah baris berikut ke bagian atas file route:

```
use Illuminate\Support\Facades\Auth;
```

Sekarang tidak tampil lagi garis merah error dari VS Code. Namun jika tidak ditambah juga tidak masalah.

File Controller

File berikutnya yang akan kita lihat adalah controller. Dari daftar route sebelumnya, terlihat `HomeController`. File ini berada di folder `app\Http\Controllers\`:

`app\Http\Controllers\Controller.php`

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use Illuminate\Http\Request;
5
6 class HomeController extends Controller
7 {
8     public function __construct()
9     {
10         $this->middleware('auth');
11     }
12
13     public function index()
14     {
15         return view('home');
16     }
17 }
```

Saya menghapus beberapa baris komentar bawaan Laravel, namun perhatikan kode yang ada, saya yakin anda sudah bisa membaca maksud perintah ini.

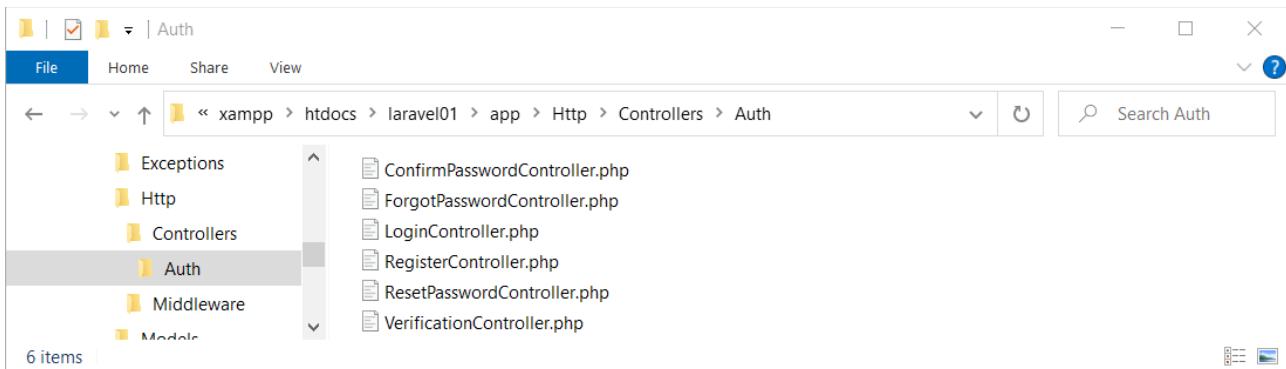
Isi method constructor berupa perintah `$this->middleware('auth')`. Artinya, semua method yang ada di dalam controller di proteksi oleh middleware bernama '**auth**'.

Method yang akan di proteksi cuma ada satu, yakni `index()`. Isi method `index` sendiri hanya 1 baris: `return view('home')`. View `home` ini merupakan tampilan dari halaman `dashboard` yang sudah pernah kita lihat setelah proses login.

Dengan mempelajari kode ini, maka jika kita ingin memproteksi route atau halaman baru, tinggal tambahkan middleware `auth`. Hasilnya, route tersebut hanya bisa diakses bagi yang sudah login.

Sisa controller lain ada di dalam folder **Auth**, yang muncul ketika instalasi Laravel UI dengan tambahan `--auth`:

Authentication



Gambar: Isi folder app\Http\Controllers\Auth\

Dalam folder ini terdapat 6 file controller yang semuanya dipakai untuk proses *authentication*. Anda bisa samakan daftar file ini dengan hasil perintah `php artisan route:list` untuk melihat route mana yang akan mengakses controller apa.

Kita akan bahas 2 diantaranya, yakni `RegisterController.php` dan `LoginController.php`.

app/Http/Controllers/Auth/RegisterController.php

```
1  <?php
2
3  namespace App\Http\Controllers\Auth;
4
5  use App\Http\Controllers\Controller;
6  use App\Providers\RouteServiceProvider;
7  use App\Models\User;
8  use Illuminate\Foundation\Auth\RegistersUsers;
9  use Illuminate\Support\Facades\Hash;
10 use Illuminate\Support\Facades\Validator;
11
12 class RegisterController extends Controller
13 {
14
15     use RegistersUsers;
16     protected $redirectTo = RouteServiceProvider::HOME;
17
18     public function __construct()
19     {
20         $this->middleware('guest');
21     }
22
23     protected function validator(array $data)
24     {
25         return Validator::make($data, [
26             'name' => ['required', 'string', 'max:255'],
27             'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
28             'password' => ['required', 'string', 'min:8', 'confirmed'],
29         ]);
30     }
31
32     protected function create(array $data)
```

```

33     {
34         return User::create([
35             'name' => $data['name'],
36             'email' => $data['email'],
37             'password' => Hash::make($data['password']),
38         ]);
39     }
40 }
```

File RegisterController ini butuh 6 buah file import, yakni:

- `App\Http\Controllers\Controller`: merupakan class dasar dari semua controller.
- `App\Providers\RouteServiceProvider`: berisi class untuk proses *redirect* setelah register berhasil.
- `App\Models\User`: sebuah model dari tabel `users` (tempat menyimpan data user).
- `Illuminate\Foundation\Auth\RegistersUsers`: berisi class dan method untuk proses registrasi user.
- `Illuminate\Support\Facades\Hash`: class facade untuk fungsi *hashing* (dipakai untuk men-hash inputan password).
- `Illuminate\Support\Facades\Validator`: class facade untuk proses validasi (sudah pernah kita bahas dalam bab form validation).

Di baris 16 terdapat property `protected $redirectTo`. Ini dipakai untuk menentukan alamat route yang dituju setelah user menyelesaikan proses register. Secara default, user akan di redirect ke route `'/home'`, yang disimpan ke dalam konstanta `RouteServiceProvider::HOME`.

Nilai ini bisa diubah dengan alamat route lain, misalnya saya ingin agar setelah register, user di redirect ke halaman `'/tabel-mahasiswa'`, maka bisa ditulis di sini.

Method `validator()` di baris 23 – 30 berisi syarat validasi untuk proses register user. Hampir semua syarat validasi ini sudah kita pelajari sebelumnya, termasuk validasi `unique:users` untuk inputan email. Yang berarti nilai inputan email harus unik dengan memeriksa kolom `email` yang ada di tabel `users`.

Salah satu syarat validasi baru adalah `'confirmed'` untuk password (baris 28). Syarat ini akan mencari inputan bernama `password_confirmation` di dalam form, lalu menyamakannya dengan nilai inputan `password`. Jika nilai inputan `password` dan `password_confirmation` tidak sama, tampilkan pesan error.

Method `create()` pada baris 32 – 39 dipakai untuk proses input ke dalam tabel `users` menggunakan **Eloquent**. Ada 3 kolom yang akan diisi, yakni `name`, `email` dan `password`. Khusus untuk password, nilai inputan form akan dilewatkan terlebih dahulu ke static method `Hash::make()`. Method ini berguna untuk mengacak isi password menggunakan teknik *hashing*. Hasil hashing inilah yang akan diinput ke kolom `password` di tabel `users`.

Jika nantinya kita ingin menambah inputan baru pada saat proses registrasi, isi method `validator()` dan `create()` juga harus disesuaikan.

Selanjutnya, berikut isi dari `LoginController.php`:

`app/Http/Controllers/Auth/LoginController.php`

```

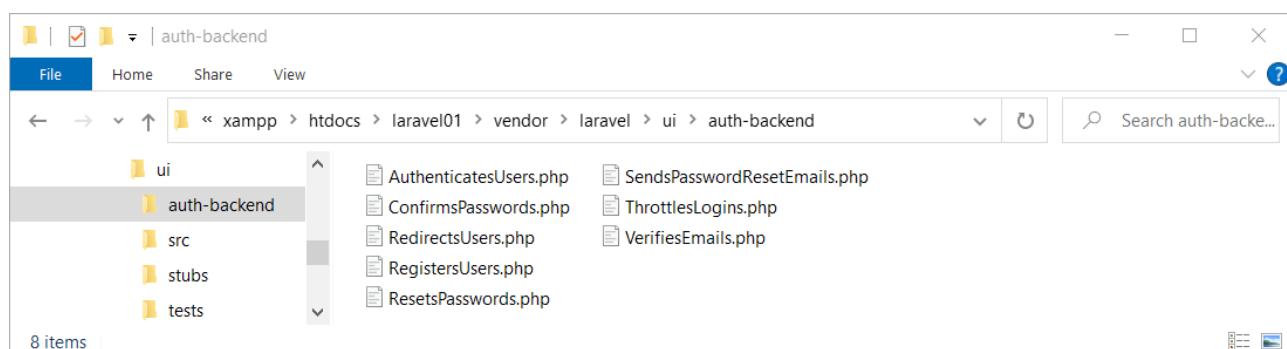
1 <?php
2
3 namespace App\Http\Controllers\Auth;
4
5 use App\Http\Controllers\Controller;
6 use App\Providers\RouteServiceProvider;
7 use Illuminate\Foundation\Auth\AuthenticatesUsers;
8
9 class LoginController extends Controller
10 {
11     use AuthenticatesUsers;
12     protected $redirectTo = RouteServiceProvider::HOME;
13
14     public function __construct()
15     {
16         $this->middleware('guest')->except('logout');
17     }
18 }
```

Isi controller ini lebih singkat karena "kerja keras" di lakukan oleh class `Illuminate\Foundation\Auth\AuthenticatesUsers`, yang di import di baris 7.

Di baris 12 terdapat property `$redirectTo` yang sama seperti di `RegisterController`. Fungsinya, sebagai alamat tujuan *redirect* setelah proses login berhasil.

Kemudian terdapat constructor yang akan mendaftarkan middleware 'guest' untuk semua method kecuali method `logout()`. Pertanyaannya, di manakah letak method `logout()` ini? Lalu untuk apa ada perintah middleware padahal tidak ada method lain di dalam `LoginController`?

Method `logout()` yang dimaksud serta method-method lain untuk proses login ada di class `AuthenticatesUsers` yang di import di baris 7. Untuk memahaminya kita harus buka lagi file ini yang berada di `vendor\laravel\ui\auth-backend\AuthenticatesUsers.php`.



Gambar: Folder `vendor\laravel\framework\src\Illuminate\Foundation\Auth\`

Terus terang, inilah salah salah satu tantangan dari penggunaan framework besar seperti Laravel (dibandingkan PHP native). Yakni jika kita ingin mendalami apa yang "sebenarnya terjadi", harus masuk lebih dalam ke file-file lain.

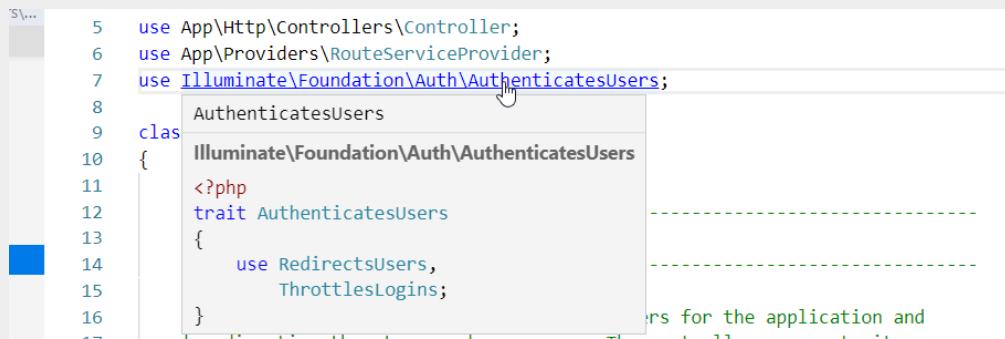
Berbagai proses authentication tidak hanya ada pada file controller yang berada di `app\Http\Controllers\Auth`, tapi juga di folder `vendor\laravel\ui\auth-backend\`.

Folder ini sebenarnya bagian dari Laravel UI dan tidak disarankan untuk di utak-atik. Cara yang sering dilakukan adalah dengan menimpa method yang sama dari dalam controller.

Misalnya jika kita ingin memodifikasi method `logout()` milik class `AuthenticatesUsers`, maka bisa menulis ulang method ini di dalam `LoginController`.

Teknik ini memang butuh skill lanjutan yang dalam buku ini tidak sempat saya bahas. Tapi setidaknya kita sudah paham bahwa kalau ingin memahami proses authentication lebih lanjut, bisa digali dari folder `vendor\laravel\ui\auth-backend\`.

Jika anda sudah menginstall **Laravel Extension Pack** di VS Code (pernah kita bahas di bab tentang blade), bisa tahan tombol CRTL + klik untuk "lompat" ke file lain di dalam aplikasi Laravel:



Gambar: Tahan tombol CRTL + klik untuk membuka file lain

Ini akan memudahkan proses pencarian file, terutama file yang di import Laravel seperti Facade class.

File Model

Terdapat 1 file model untuk proses authentication, yang berada di folder `app\Models`:

`app/Models/User.php`

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Contracts\Auth\MustVerifyEmail;
6 use Illuminate\Database\Eloquent\Factories\HasFactory;
7 use Illuminate\Foundation\Auth\User as Authenticatable;
8 use Illuminate\Notifications\Notifiable;
9 use Laravel\Sanctum\HasApiTokens;

```

```

10 class User extends Authenticatable
11 {
12     use HasApiTokens, HasFactory, Notifiable;
13
14     protected $fillable = [
15         'name', 'email', 'password',
16     ];
17
18     protected $hidden = [
19         'password', 'remember_token',
20     ];
21
22     protected $casts = [
23         'email_verified_at' => 'datetime',
24     ];
25 }

```

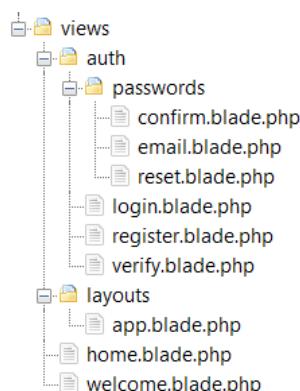
Class User memiliki 3 property:

- **\$fillable**: Membatasi kolom yang bisa diisi dengan teknik mass assignment. Dalam kode ini, kolom yang boleh diisi adalah name, email dan password.
- **\$hidden**: Menyembunyikan kolom ketika diakses menggunakan method yang akan menampilkan semua nilai tabel seperti method User::all(). Di sini, kolom password dan remember_token akan disembunyikan.
- **\$casts**: Menkonversi tipe data kolom. Dalam hal ini, kolom email_verified_at akan dikonversi menjadi tipe data datetime pada saat diakses (terutama saat dikonversi menjadi JSON).

Dari ketiga property ini, yang baru kita pelajari dalam buku ini adalah property **\$fillable**. Property **\$hidden** dan **\$cast** akan kita bahas di buku **Laravel In Depth #1** (termasuk materi Model lain seperti accessor dan mutator).

File View

Laravel authentication membawa cukup banyak file view, total terdapat tambahan 8 file view yang bisa dilihat dari struktur folder berikut:



Gambar: Struktur view Laravel authentication

Di dalam folder **auth** terdapat file `login.blade.php` untuk tampilan halaman login, file `register.blade.php` untuk tampilan halaman register dan file `verify.blade.php` untuk tampilan halaman verifikasi pengiriman token ke email.

Folder auth ini juga memiliki folder lain, yakni **passwords** yang berisi 3 buah file view untuk proses reset password.

Selain itu juga ada folder **layouts** yang berisi `app.blade.php`. Ini adalah semacam view "master" yang berisi bagian header HTML serta menu navigasi. File view lain akan meng-extends file `app.blade.php` ini.

Terakhir, terdapat file `home.blade.php` untuk tampilan halaman dashboard, yakni halaman yang terlihat ketika user berhasil login ke dalam aplikasi.

Karena keterbatasan tempat, saya tidak bisa menampilkan isi kode program dari semua view. Anda bisa lihat-lihat sekilas file ini, terutama file `app.blade.php`, `register.blade.php`, `login.blade.php` serta `home.blade.php`.

Kode yang ada memang tampak rumit, tapi mayoritas perintah blade yang digunakan sudah kita pelajari sebelumnya. File view ini juga cukup banyak menggunakan komponen Bootstrap, seperti navbar, form dan card.

Untuk tampilan teks, file view ini banyak menggunakan fungsi `__()` yang pernah kita bahas di bab localization, sebagai contoh di file `register.blade.php`, judul form ditulis sebagai berikut:

```
<div class="card-header">{{ __('Register') }}</div>
```

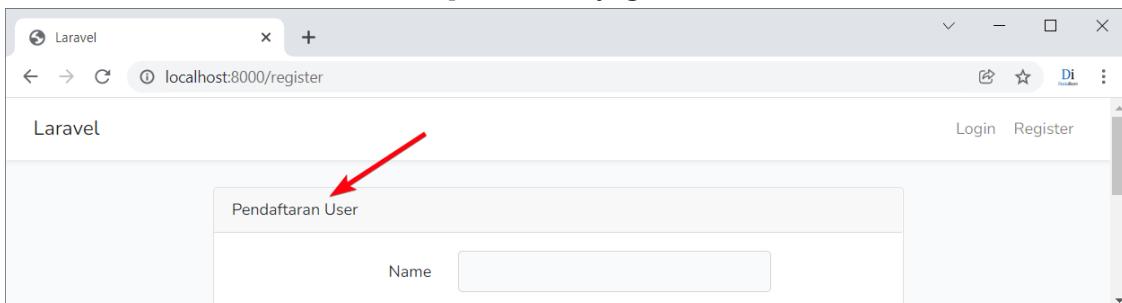
Tujuannya agar judul form bisa diubah dari file localization. Namun kita bisa saja mengedit langsung di file view:

```
<div class="card-header">{{ __('Pendaftaran User') }}</div>
```

Atau mengganti fungsi `__()` dengan teks biasa:

```
<div class="card-header">Pendaftaran User</div>
```

Save file `register.blade.php`, dan tampilan akhir juga akan berubah:



Gambar: Judul form register sudah berubah jadi "Pendaftaran User"

Silahkan anda coba utak-atik sekilas file view yang ada dan lihat perubahan yang terjadi.

24.4. Case Study: Menambah Halaman Baru

Kali ini saya ingin praktik menambah beberapa halaman baru yang hanya bisa diakses jika user sudah login. Praktek kali ini sangat mirip seperti studi kasus login middleware yang sudah kita buat, tapi kali ini memanfaatkan fitur authentication Laravel.

Pertama, kita butuh sebuah controller:

```
php artisan make:controller MahasiswaController
```

Lalu tambah route berikut:

```
routes/web.php
```

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/', function () {
7     return view('welcome');
8 });
9
10 Auth::routes();
11
12 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
13 ->name('home');
14
15 Route::get('/daftar-mahasiswa', [MahasiswaController::class, 'daftarMahasiswa'])
16 ->middleware('auth');
17
18 Route::get('/tabel-mahasiswa', [MahasiswaController::class, 'tabelMahasiswa'])
19 ->middleware('auth');
20
21 Route::get('/blog-mahasiswa', [MahasiswaController::class, 'blogMahasiswa'])
22 ->middleware('auth');
```

Ketiga route di baris 15 - 22 di proteksi dengan middleware 'auth' sehingga hanya bisa diakses jika user sudah login saja.

Lanjut ke file `MahasiswaController.php`:

```
app/Http/Controllers/MahasiswaController.php
```

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MahasiswaController extends Controller
8 {
9     public function daftarMahasiswa()
```

Authentication

```
10     {
11         return view('halaman',[ 'judul' => 'Daftar Mahasiswa']);
12     }
13
14     public function tabelMahasiswa()
15     {
16         return view('halaman',[ 'judul' => 'Tabel Mahasiswa']);
17     }
18
19     public function blogMahasiswa()
20     {
21         return view('halaman',[ 'judul' => 'Blog Mahasiswa']);
22     }
23 }
```

Isi controller ini sama persis seperti yang kita pakai untuk studi kasus "Login Middleware", dimana setiap method akan menampilkan view `halaman.blade.php` dan mengirim variabel `$judul`.

Sekarang kita akan masuk ke view. Saya ingin agar ketiga halaman ini tampil di menu navbar yang tampil pada saat user sudah login, yakni di posisi berikut:



Gambar: Posisi menu yang diinginkan

Kode untuk bagian navbar ini ada di file `layouts\app.blade.php`, yakni file "master" layout yang akan dipakai untuk semua file view authentication.

Sebelum itu, kita akan bahas 2 perintah blade yang berhubungan dengan authentication, yakni pasangan `@guest` dan `@endguest` serta `@auth` dan `@endauth`.

Kedua perintah ini sebenarnya kondisi if khusus, dimana blok `@guest` hanya akan tampil untuk user yang belum login, dan blok `@auth` akan ditampilkan untuk user yang sudah login.

Perhatikan potongan kode berikut:

```
1 <nav>
2   <ul>
3     @guest
4       <li><a href="#">Menu 1</a></li>
5       <li><a href="#">Menu 2</a></li>
6     @endguest
7     @auth
```

```

8      <li><a href="#">Menu 3</a></li>
9      <li><a href="#">Menu 4</a></li>
10     @endauth
11   </ul>
12 </nav>
```

Dalam contoh ini, Menu 1 dan Menu 2 hanya bisa dilihat oleh user yang belum login, sedangkan Menu 3 dan Menu 4 hanya bisa diakses oleh user yang sudah login. Teknik inilah yang bisa kita pakai dalam pembuatan menu navigasi atau bagian web lain yang butuh pembatasan.

Untuk menambahkan menu navigasi, silahkan buka file `layouts\app.blade.php`, lalu cari baris komentar `<!-- Left Side Of Navbar -->`:



```

33   <div class="collapse navbar-collapse" id="navbarSupportedContent">
34     <!-- Left Side Of Navbar -->
35     <ul class="navbar-nav me-auto">
36       </ul>
37
38
39     <!-- Right Side Of Navbar -->
40     <ul class="navbar-nav ms-auto">
41       <!-- Authentication Links -->
```

Gambar: Posisi untuk menambah menu di bagian kiri navbar

Kode dari file `layouts\app.blade.php` cukup panjang, sehingga untuk menghemat tempat tidak saya tampilkan di sini. Anda bisa buka langsung file tersebut di teks editor.

Ke dalam tag ``, tambah kode berikut:

```

<!-- Left Side Of Navbar -->
<ul class="navbar-nav me-auto">
  @auth
    <li class="nav-item">
      <a class="nav-link" href="{{url('daftar-mahasiswa')}}">Daftar Mahasiswa</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{{url('tabel-mahasiswa')}}">Tabel Mahasiswa</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{{url('blog-mahasiswa')}}">Blog Mahasiswa</a>
    </li>
  @endauth
</ul>
```

Kode ini akan menambah 3 buah menu baru di bagian kiri atas navbar. Selain itu kita juga bisa lihat beberapa perintah blade lain dalam file `layouts\app.blade.php`, misalnya kondisi `@guest` untuk menampilkan link **Login** dan **Register**. Kedua link ini seharusnya memang hanya ditujukan untuk user yang belum login.

Kemudian terdapat juga kondisi `@if (Route::has('register'))` yang dipakai untuk

memeriksa apakah route untuk registrasi user tersedia atau tidak. Pemeriksaan ini dilakukan karena Laravel mengizinkan kita me-nonaktifkan fitur registrasi user, sehingga authentication hanya untuk sekedar login dan logout saja.

Lalu juga ada perintah {{ Auth::user()->name }} untuk menampilkan nama user yang sedang login saat ini. Dan seperti yang bisa di tebak, kita juga bisa menampilkan alamat email user dengan perintah {{ Auth::user()->email }}.

Di bagian akhir file layouts\app.blade.php terdapat perintah berikut:

```
<main class="py-4">
    @yield('content')
</main>
```

Perintah @yield('content') adalah "tanda" untuk tempat mengisi konten bagi child view, yakni view yang akan meng-extends file layouts\app.blade.php. Hampir semua view bawaan auth akan meng-extends file ini.

Baik, proses penambahan menu sudah selesai. Kita akan buat file view halaman.blade.php yang akan diakses oleh menu tersebut:

resources/views/halaman.blade.php

```
1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container">
5      <div class="row justify-content-center">
6          <div class="col-md-8">
7              <div class="card">
8                  <div class="card-header">{{$judul}}</div>
9                  <div class="card-body">
10                     <p>Bla... bla..</p>
11                 </div>
12             </div>
13         </div>
14     </div>
15 </div>
16 @endsection
```

Di baris 1 terdapat perintah untuk meng-extends file layouts\app.blade.php. Dengan ini, file kita sudah otomatis sudah memiliki header HTML beserta menu navigasi (navbar) yang kita edit sebelumnya.

Isi dari section content terdiri dari struktur dasar grid Bootstrap beserta komponen **card**. Sebagai judul card di baris 8 saya mengisinya dengan hasil variabel \$judul dari controller.

Mari kita test. Silahkan buka halaman login atau register. Menu navbar tetap tampil normal tanpa ada tambahan apapun. Namun begitu login, di sisi kiri akan terdapat 3 buah menu baru:



Gambar: Tiga menu tambahan

Ketika menu ini di klik, judul card di tengah halaman juga akan berubah-ubah, sesuai dengan isi variabel `$judul` yang dikirim dari controller.

24.5. Menampilkan Data User

Di dalam view `layouts/app.blade.php` kita sudah melihat perintah untuk menampilkan nama user yang sedang login, yakni dengan kode `{{ Auth::user()->name }}`. Di sini sebenarnya yang diakses adalah method `user()` milik facade class **Auth**. Di dalam view, class `Auth` ini sudah bisa langsung diakses.

Cara yang sama juga bisa dilakukan dari controller, hanya saja kita harus meng-import class `Auth` ini terlebih dahulu. Sebagai contoh praktek, saya akan memodifikasi isi method `daftarMahasiswa()` sebagai berikut:

`app/Http/Controllers/MahasiswaController.php`

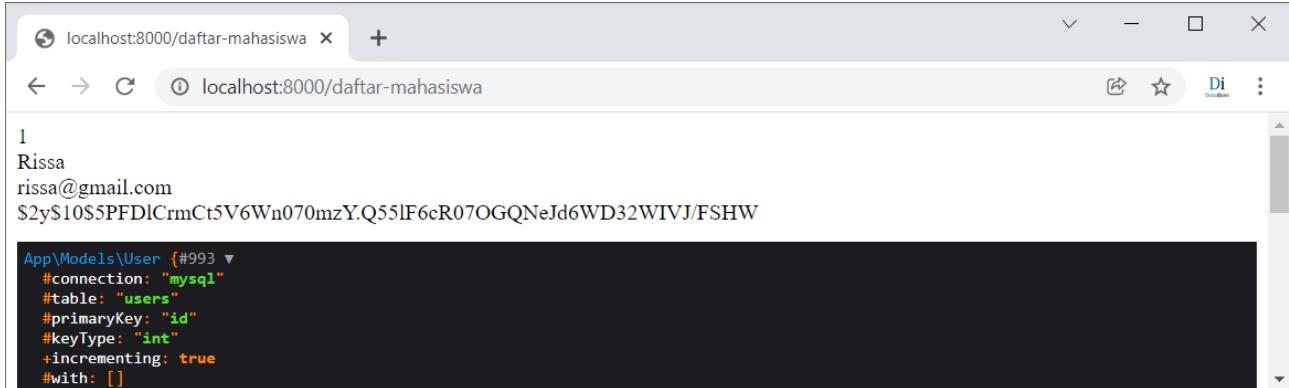
```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Auth;
7
8 class MahasiswaController extends Controller
9 {
10     public function daftarMahasiswa()
11     {
12         echo Auth::user()->id."<br>";
13         echo Auth::user()->name."<br>";
14         echo Auth::user()->email."<br>";
15         echo Auth::user()->password."<br>";
16
17         dump(Auth::user());
18     }
19 // ...
20 }
```

Di baris 6 terdapat tambahan perintah import Illuminate\Support\Facades\Auth. Kemudian di dalam method daftarMahasiswa() saya mengakses 4 data user menggunakan static method Auth::user() yang di chaining dengan nama kolom tabel users.

Perintah di baris 12 – 15 ini akan menampilkan nilai id, name, email dan password user yang sedang login. Kemudian di baris 17 saya men-dump Auth::user() sekedar melihat apa hasil yang di dapat.

Silahkan buka web browser, login dan langsung klik menu daftar-mahasiswa:



Gambar: Menampilkan data user

Hasilnya akan terlihat semua data user yang saat ini sedang login. Khusus untuk nilai password, itu merupakan hasil dari proses *hashing*. Perintah dump(Auth::user()) sendiri akan mengembalikan sebuah **User** object.

Alternatif cara lain untuk mengakses data user ini adalah dari **Request** object, seperti berikut:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2 // ...
3
4 class MahasiswaController extends Controller
5 {
6     public function daftarMahasiswa(Request $request)
7     {
8         echo $request->user()->id."<br>";
9         echo $request->user()->name."<br>";
10        echo $request->user()->email."<br>";
11        echo $request->user()->password."<br>";
12    }
13 }

```

Seperti biasa, jika kita akan mengakses **Request** object dari dalam method, maka perlu menulis type hint Request \$request sebagai argument (untuk menjalankan *dependency injection*).

Kemudian untuk mengakses data **user**, bisa dilakukan dengan perintah \$request->user()-><nama_kolom>. Misalnya untuk menampilkan alamat email dari user, perintahnya adalah \$request->user()->name.

Class Auth ini juga menyediakan beberapa method lain yang berhubungan dengan User object, misalnya untuk mengetahui apakah user sudah login atau tidak bisa menggunakan perintah `Auth::check()`, seperti contoh berikut:

app/Http/Controllers/MahasiswaController.php

```

1 <?php
2 // ...
3
4 class MahasiswaController extends Controller
5 {
6     public function daftarMahasiswa(Request $request)
7     {
8         if(Auth::check())
9         {
10             echo "Selamat datang, ".$request->user()->name;
11         }
12         else
13         {
14             echo "Silahkan login terlebih dahulu";
15         }
16     }
17 }
```

Method `Auth::check()` mengembalikan nilai **true** jika user sudah login. Jika ini terjadi, maka perintah di baris 10 akan dijalankan. Jika hasilnya **false** (user belum login), blok else yang akan dijalankan.



Gambar: Hasil pemeriksaan kondisi `Auth::check()`

Materi tentang **Laravel Authentication** memang sangat luas dan masih banyak yang belum kita bahas, misalnya cara reset password, konfirmasi ke email serta mengenal **Auth** class dengan lebih dalam.

Saya rencanakan materi ini akan dibahas di buku Laravel In Depth #2. Namun setidaknya apa yang kita bahas di sini sudah bisa sebagai gambaran kasar cara penggunaan fitur authentication di Laravel.

Dalam bab selanjutnya, kita akan masuk ke studi kasus menggabungkan CRUD dengan authentication.

25. Case Study: CRUD dan Authentication

Kali ini saya akan mengajak anda untuk membuat studi kasus sederhana dengan menggabungkan konsep CRUD dan authentication. Kita akan membuat sistem CRUD yang hanya bisa diakses oleh user yang sudah login. Materi ini juga akan menjadi dasar dari pembahasan bab berikutnya.

Untuk menghemat quota karena ukuran **npm** yang lumayan besar, saya akan melanjutkan aplikasi Laravel dari bab sebelumnya, yakni dari bab authentication. Dengan demikian di dalam folder **laravel01** sudah terinstall npm beserta Laravel Authentication.

25.1. Tampilan Akhir "CRUD Jurusan"

Aplikasi yang akan kita rancang berupa CRUD untuk tabel **jurusans**. Setelah user login, dia bisa melihat, menambah, mengubah dan menghapus daftar jurusan yang tersedia. Berikut tampilan akhir dari studi kasus ini:

#	Nama Jurusan	Nama Dekan	Jumlah Mahasiswa
1	Ilmu Komputer	Dr. Syahrial, M.Kom	80
2	Teknik Informatika	Prof. Silvia Nst	100
3	Sistem Informasi	Dr. Umar Agustinus, M.Sc	120

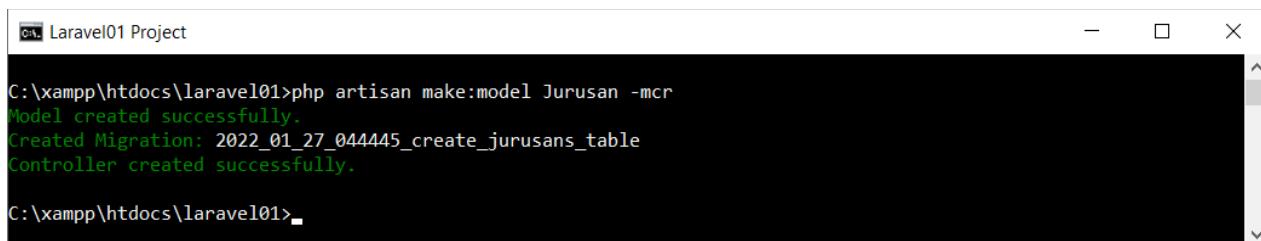
Gambar: Tampilan akhir studi kasus "CRUD Jurusan"

Kita akan bahas komponen apa saja yang perlu disiapkan.

25.2. Membuat Migration

Hal pertama yang perlu dilakukan adalah membuat tabel dari file migration. Namun nantinya kita juga butuh file model dan juga controller. Ketiganya bisa dibuat dengan perintah berikut:

```
php artisan make:model Jurusan -mcr
```



```
C:\xampp\htdocs\laravel01>php artisan make:model Jurusan -mcr
Model created successfully.
Created Migration: 2022_01_27_044445_create_jurusans_table
Controller created successfully.

C:\xampp\htdocs\laravel01>
```

Gambar: Membuat model, migration dan controller Jurusan

Setelah itu, buka file migration jurusan dan ubah method up() sebagai berikut:

```
1 <?php
2 /**
3  class CreateJurusansTable extends Migration
4 {
5     public function up()
6     {
7         Schema::create('jurusans', function (Blueprint $table) {
8             $table->id();
9             $table->string('nama_jurusan');
10            $table->string('nama_dekan');
11            $table->integer('jumlah_mahasiswa');
12            $table->timestamps();
13        });
14    }
15 }
```

Selain kolom bawaan id, created_at dan updated_at, saya menambah kolom nama_jurusan, nama_dekan dan jumlah_mahasiswa ke dalam tabel **jurusans**.

Karena dalam bab sebelumnya kita sudah menjalankan migration, maka ketika ada perubahan atau tambahan file migration, bisa dijalankan dengan perintah berikut:

```
php artisan migrate:fresh
```

Perintah ini akan menghapus semua tabel dan membuatnya kembali. Silahkan cek di cmd MySQL Client atau phpMyAdmin untuk memastikan tabel jurusans sudah tersedia.

25.3. Menyiapkan Route

Dalam bab sebelumnya kita sudah memiliki beberapa route yang diantaranya dipakai untuk mengakses halaman daftar-mahasiswa, tabel-mahasiswa dan blog-mahasiswa. Selain itu juga terdapat route untuk proses authentication dengan perintah Auth::routes().

Saya akan biarkan semua route ini, namun kita butuh tambahan route baru untuk pembuatan CRUD:

routes/web.php

```
1 <?php
2 /**
3 use App\Http\Controllers\JurusanController;
4
5 Route::get('/', [JurusanController::class, 'index'])->middleware('auth');
6 Route::resource('jurusans', JurusanController::class)->middleware('auth');
```

Route di baris 5 berfungsi untuk mengubah halaman root atau home aplikasi kita. Dengan tambahan ini, maka ketika alamat localhost:8000 dibuka, itu akan mengakses method `index()` yang ada di `JurusanController`.

Route selanjutnya sudah pernah kita bahas di akhir bab CRUD, dimana `Route::resource` akan mewakili 7 route lain, yakni untuk proses: `index`, `create`, `store`, `show`, `edit`, `update`, dan `destroy`. Daftar lengkapnya bisa dilihat dengan menjalankan perintah `php artisan route:list`.

Selain itu terdapat tambahan `->middleware('auth')` yang berarti semua route ini hanya bisa diakses bagi user yang sudah login terlebih dahulu.

25.4. Menyiapkan Model

Dalam controller nanti kita akan menggunakan teknik *mass assignment*, sehingga perlu menambah property `$guarded` ke dalam model **Jurusan**:

app/Http/Jurusan.php

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Jurusan extends Model
9 {
10     use HasFactory;
11     protected $guarded = [];
12 }
```

25.5. Menyiapkan JurusanController

File `JurusanController.php` saat ini sudah berisi struktur dasar method RESTfull, karena pada saat proses pembuatan kita menambah option `-r` ke perintah `php artisan make:model Jurusan -mcr`. Method yang dimaksud adalah `index()`, `create()`, `store()`, `show()`, dll.

Berikut kode program lengkap dari JurusanController.php:

app/Http/Controllers/JurusanController.php

```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Jurusan;
6  use Illuminate\Http\Request;
7
8  class JurusanController extends Controller
9  {
10     public function index()
11     {
12         return view('jurusan.index',[ 'jurusans' => Jurusan::all()]);
13     }
14
15     public function create()
16     {
17         return view('jurusan.create');
18     }
19
20     public function store(Request $request)
21     {
22         $validateData = $request->validate([
23             'nama_jurusan'          => 'required',
24             'nama_dekan'           => 'required',
25             'jumlah_mahasiswa'   => 'required|min:10|integer',
26         ]);
27
28         Jurusan::create($validateData);
29         return redirect('/')->with('pesan',"Jurusan $request->nama_jurusan
30                             berhasil ditambahkan");
31     }
32
33     public function show(Jurusan $jurusan)
34     {
35         return view('jurusan.show',compact('jurusan'));
36     }
37
38     public function edit(Jurusan $jurusan)
39     {
40         return view('jurusan.edit',compact('jurusan'));
41     }
42
43     public function update(Request $request, Jurusan $jurusan)
44     {
45         $validateData = $request->validate([
46             'nama_jurusan'          => 'required',
47             'nama_dekan'           => 'required',
48             'jumlah_mahasiswa'   => 'required|min:10|integer',
49         ]);
50
51         $jurusan->update($validateData);

```

```

52         return redirect('/jurusans/'.$jurusan->id)
53         ->with('pesan',"Jurusan $jurusan->nama_jurusan berhasil diupdate");
54     }
55
56
57     public function destroy(Jurusan $jurusan)
58     {
59         $jurusan->delete();
60         return redirect('/')
61         ->with('pesan',"Jurusan $jurusan->nama_jurusan berhasil dihapus");
62     }
63 }
```

Kodenya memang lumayan panjang karena terdapat 7 method RESTfull.

Method `index()` berfungsi untuk menampilkan semua isi tabel jurusan. Di sini saya mengirim hasil `Jurusan::all()` ke dalam view `jurusan.index`. Data ini nantinya bisa diakses dari array `$jurusan` di dalam view.

Method `create()` dipakai untuk menampilkan form penambahan jurusan, yang akan diproses oleh view `jurusany.create`.

Method `store()` dipakai untuk memproses inputan form jurusan. Terdapat 3 syarat validasi form, yang jika lolos akan disimpan ke dalam tabel menggunakan perintah `Jurusany::create($validateData)`. Setelah itu halaman di `redirect` ke root ('/') beserta pesan flash.

Method `show()` dipakai untuk menampilkan 1 data jurusan. Menggunakan teknik *route model binding*, variabel `$jurusan` akan langsung berisi 1 data object. Untuk mengirim data ini ke view `jurusany.show`, bisa dengan cara yang sama seperti di bab CRUD, yakni:

```
return view('jurusany.show',[ 'jurusan' => $jurusan]);
```

Tapi kali ini saya menggunakan function `compact()`:

```
return view('jurusany.show',compact('jurusan'));
```

Ini hanya sekedar alternatif saja. Anda bebas ingin menggunakan cara pertama (dalam bentuk array), atau menggunakan function `compact()`.

Selanjutnya method `edit()` dipakai untuk menampilkan form update jurusan, yang akan diproses oleh view `jurusany.edit`.

Method `update()` dipakai untuk memproses hasil update. Syarat validasi sama seperti di method `store()`. Jika lolos, data di update dengan perintah `$jurusan->update($validateData)`. Setelah itu halaman di `redirect` ke `/jurusans/` beserta pesan flash.

Perhatikan untuk proses `redirect` saya langsung menulis alamat URL, tidak menggunakan named route seperti di bab tentang CRUD. Maksudnya, jika perintah `redirect` menggunakan named route, maka penulisannya adalah sebagai berikut:

```
return redirect()->route('jurusans.show',[ 'jurusan'=>$jurusan->id])
->with('pesan','Jurusan $jurusan->nama_jurusan berhasil diupdate');
```

Ini lebih ke variasi saja, karena di Laravel memang terdapat banyak cara untuk menyelesaikan sebuah masalah.

Terakhir, method `destroy()` dipakai untuk menghapus data jurusan dengan perintah `$jurusan->delete()`, kemudian halaman di `redirect` ke root ('/') beserta pesan flash.

25.6. Modifikasi Auth Controller

Di dalam file route, sebelumnya kita sudah mengubah pemrosesan untuk halaman root. Ketika dibuka alamat `localhost:8000`, itu akan mengakses method `index()` dari `JurusansController`.

Saya juga ingin agar ketika user selesai melakukan proses registrasi atau sesudah login, akan langsung di redirect ke halaman '/', tidak lagi ke halaman `localhost:8000/home`.

Untuk mengubah ini cukup *simple*, silahkan buka file `app\Http\Controllers\Auth\RegisterController.php` dan `app\Http\Controllers\Auth\LoginController.php`, lalu cari baris berikut:

```
protected $redirectTo = RouteServiceProvider::HOME;
```

Ubah menjadi:

```
protected $redirectTo = '/';
```

Sekarang setelah login dan register, user akan di `redirect` ke halaman root.

25.7. Membuat View

Sama seperti di bab CRUD, kita perlu membuat 4 file view yang akan ditempatkan ke dalam folder **jurusans** di `resources\views\`. Berikut nama file yang akan kita buat:

- `index.blade.php`
- `show.blade.php`
- `create.blade.php`
- `edit.blade.php`

Namun sebelum masuk ke view ini, saya akan modifikasi sedikit view `layouts\app.blade.php` untuk menambah 2 buah menu baru di bagian navbar. Silahkan buka file ini dan cari blok komentar `<!-- Left Side Of Navbar -->`, lalu tambah kode berikut:

```
resources/views/layouts/app.blade.php
```

```
<!-- Left Side Of Navbar -->
<ul class="navbar-nav me-auto">
    @auth
```

```

...
<li class="nav-item">
    <a class="nav-link" href="{{url('/')}}">Tabel Jurusan</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{{url('/jurusans/create')}}">Tambah Jurusan</a>
</li>
@endauth
</ul>

```

Menu **Tabel Jurusan** dipakai untuk menampilkan semua isi tabel, sedangkan menu **Tambah Jurusan** akan menampilkan form untuk menambah jurusan baru.

Lanjut, berikut kode program untuk view jurusan\index.blade.php:

resources/views/jurusan/index.blade.php

```

1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container mt-3">
5      <div class="row">
6          <div class="col-12">
7
8              <div class="py-4 d-flex justify-content-between align-items-center">
9                  <h1 class="h2">Tabel Jurusan</h1>
10                 <a href="{{url('/jurusans/create')}}" class="btn btn-primary">
11                     Tambah Jurusan
12                 </a>
13             </div>
14
15             @if(session()->has('pesan'))
16                 <div class="alert alert-success" role="alert">
17                     {{ session()->get('pesan') }}
18                 </div>
19             @endif
20
21             <table class="table table-striped">
22                 <thead>
23                     <tr>
24                         <th>#</th>
25                         <th>Nama Jurusan</th>
26                         <th>Nama Dekan</th>
27                         <th>Jumlah Mahasiswa</th>
28                     </tr>
29                 </thead>
30                 <tbody>
31                     @forelse ($jurusans as $jurusan)
32                         <tr>
33                             <th>{{$loop->iteration}}</th>
34                             <td>
35                                 <a href="{{ url('/jurusans/'.$jurusan->id) }">
36                                     {{$jurusan->nama_jurusan}}
37                                 </a>

```

```

38      </td>
39      <td>{{$jurusan->nama_dekan}}</td>
40      <td>{{$jurusan->jumlah_mahasiswa}}</td>
41  </tr>
42  @empty
43      <td colspan="6" class="text-center">Tidak ada data...</td>
44  @endforelse
45  </tbody>
46  </table>
47  </div>
48 </div>
49 </div>
50 @endsection

```

Untuk semua view CRUD jurusan, saya meng-extends view `layouts.app`. Dengan demikian halaman kita sudah memiliki header HTML lengkap dengan menu navigasi.

Struktur `index.blade.php` ini sangat mirip seperti praktek di bab CRUD, hanya saja sekarang yang ditampilkan adalah tabel jurusan.

Pada baris 10 – 12 terdapat kode untuk membuat tombol "Tambah Jurusan" yang ketika di klik akan membuka halaman `url('/jurusans/create')`. Di sini saya menggunakan route "tradisional", tidak lagi *named route* seperti praktek di bab CRUD. Kembali, ini hanya sebuah alternatif saja, anda tetap bisa menggunakan method `route('jurusans.create')`.

Kemudian di baris 15 – 19 merupakan kode program untuk menampilkan flash session pesan, misalnya untuk memberitahu proses penambahan atau penghapusan berhasil.

Proses menampilkan data tabel ada di baris 31 – 44 dengan perulangan `@forelse`. Proses looping ini akan mengakses array `$jurusans` yang dikirim dari controller. Khusus untuk kolom Nama Jurusan, terdapat tambahan tag `<a>` agar menjadi link. Atribut `href` dari link ini berisi `url('/jurusans/' . $jurusan->id)`, yang nantinya akan menjadi `localhost:8000/jurusans/1`, `localhost:8000/jurusans/2`, dst.

Save file blade ini, lalu kita coba akses halaman root localhost:8000. Akan tampil form login karena halaman root ini menggunakan route berikut:

```
Route::get('/', [JurusanController::class, 'index'])->middleware('auth');
```

Artinya ketika diakses, akan di proses oleh method `index()` milik `JurusanController`. Namun route ini juga di proteksi oleh middleware auth. Halaman login di atas tampil karena middleware ini.

Silahkan buat user baru karena user dari bab sebelumnya sudah terhapus ketika kita refresh migration. Tepat setelah pendaftaran, halaman akan di redirect ke method `index()` di `JurusanController` yang pada gilirannya akan menampilkan view `index.blade.php` yang baru saja kita buat:



Gambar: Tampilan halaman index

Saat ini tabel jurusan belum berisi data apapun, sehingga tidak ada yang bisa ditampilkan.

Lanjut, kita akan buat form tambah data jurusan:

`resources/views/jurusan/create.blade.php`

```

1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container mt-3">
5      <div class="row">
6          <div class="col-sm-8 col-md-6">
7              <h1 class="h2 pt-4">Pendaftaran Jurusan</h1>
8              <hr>
9
10         <form action="{{url('/jurusans')}}" method="POST">
11             @csrf
12             <div class="mb-3">
13                 <label class="form-label" for="nama_jurusan">Nama Jurusan</label>
14                 <input type="text"
15                     class="form-control" @error('nama_jurusan') is-invalid @enderror"
16                     id="nama_jurusan" name="nama_jurusan"
17                     value="{{ old('nama_jurusan') }}">
18                 @error('nama_jurusan')
19                     <div class="text-danger">{{ $message }}</div>
20                 @enderror
21             </div>
22
23             <div class="mb-3">
24                 <label class="form-label" for="nama_dekan">Nama Dekan</label>
25                 <input type="text"
26                     class="form-control" @error('nama_dekan') is-invalid @enderror"
27                     id="nama_dekan" name="nama_dekan" value="{{ old('nama_dekan') }}">
28                 @error('nama_dekan')
29                     <div class="text-danger">{{ $message }}</div>
30                 @enderror
31             </div>
32

```

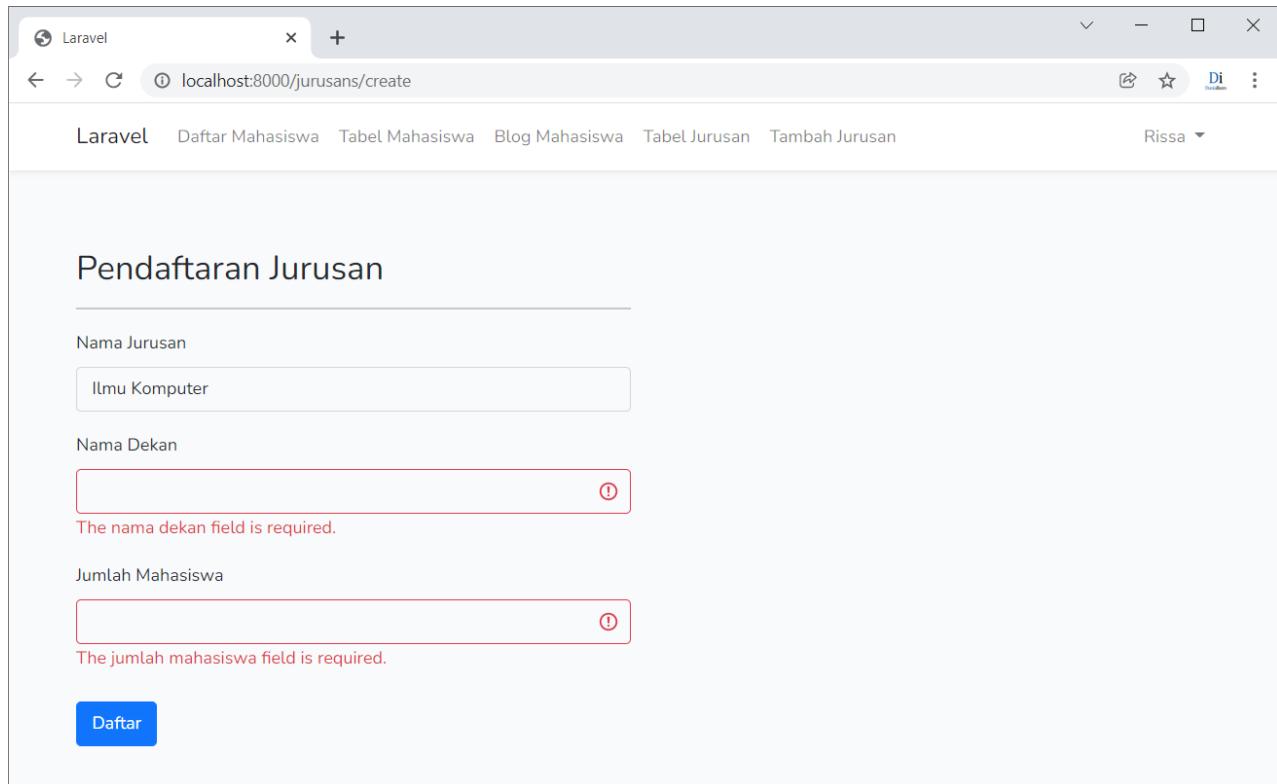
```

33     <div class="mb-3">
34         <label class="form-label" for="jumlah_mahasiswa">
35             Jumlah Mahasiswa</label>
36         <input type="text"
37             class="form-control @error('jumlah_mahasiswa') is-invalid @enderror"
38             id="jumlah_mahasiswa" name="jumlah_mahasiswa"
39             value="{{ old('jumlah_mahasiswa') }}"/>
40             @error('jumlah_mahasiswa')
41                 <div class="text-danger">{{ $message }}</div>
42             @enderror
43     </div>
44
45     <button type="submit" class="btn btn-primary my-2">Daftar</button>
46 </form>
47
48 </div>
49 </div>
50 </div>
51
52 @endsection

```

View `create.blade.php` butuh 3 buah inputan form: `nama_jurusan`, `nama_dekan` dan `jumlah_mahasiswa`. Struktur form yang digunakan juga sama seperti di bab CRUD, termasuk menampilkan pesan kesalahan dengan perintah `@error`, serta proses re-populate inputan dengan method `old()`. Ketika di submit, form akan mengirim data ke route `url('/jurusans')`.

Halaman ini bisa diakses dari menu Tambah Jurusan di navbar, men-klik tombol Tambah Jurusan di halaman index, atau mengetik alamat localhost:8000/jurusans/create:



Gambar: Tampilan halaman create dengan error validasi

Silahkan test untuk melihat pesan error validasi. Setelah itu input data jurusan, dan halaman akan di redirect ke home yang menampilkan seluruh isi tabel jurusan:

#	Nama Jurusan	Nama Dekan	Jumlah Mahasiswa
1	Ilmu Komputer	Dr. Syahrial, M.Kom	80
2	Teknik Informatika	Prof. Silvia Nst	100
3	Sistem Informasi	Dr. Umar Agustinus, M.Sc	120

Gambar: Tampilan halaman index dengan data tabel jurusan

Silahkan tambah beberapa data ke tabel jurusan ini.

Lanjut, kita akan rancang view untuk menampilkan data 1 jurusan:

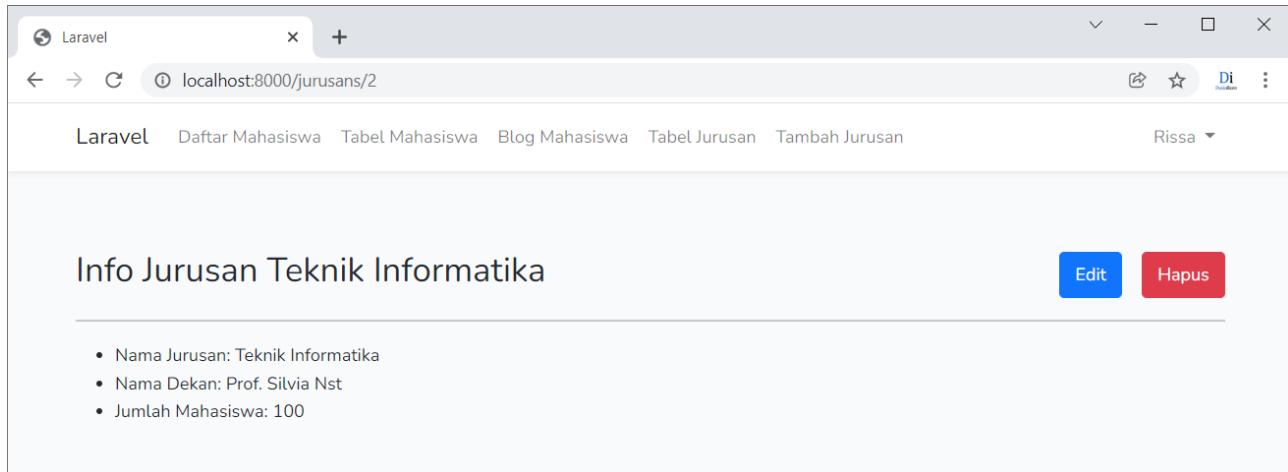
resources/views/jurusan/show.blade.php

```
1 @extends('layouts.app')
2
3 @section('content')
4
5 <div class="container mt-3">
6   <div class="row">
7     <div class="col-12">
8
9       <div class="pt-4 d-flex justify-content-between align-items-center">
10         <h2>Info Jurusan {{$jurusan->nama_jurusan}}</h2>
11         <div class="d-flex">
12           <a href="{{url('/jurusans/'.$jurusan->id.'/edit')}}" class="btn btn-primary">Edit</a>
13           <form action="{{url('/jurusans/'.$jurusan->id)}}" method="POST">
14             @method('DELETE')
15             <button type="submit" class="btn btn-danger ms-3">Hapus</button>
16             @csrf
17           </form>
18         </div>
19       </div>
20     </div>
21     <hr>
22
23     @if(session()->has('pesan'))
```

```
24     <div class="alert alert-success" role="alert">
25         {{ session()->get('pesan')}}
26     </div>
27     @endif
28
29     <ul>
30         <li>Nama Jurusan: {{$jurusan->nama_jurusan}} </li>
31         <li>Nama Dekan: {{$jurusan->nama_dekan}} </li>
32         <li>Jumlah Mahasiswa: {{$jurusan->jumlah_mahasiswa}} </li>
33     </ul>
34     </div>
35 </div>
36 </div>
37
38 @endsection
```

Halaman ini bisa diakses dengan men-klik daftar jurusan di halaman index, atau dengan mengunjungi halaman dengan format: `localhost:8000/jurusans/<id_jurusan>`.

Berikut tampilannya:



Gambar: Halaman untuk menampilkan 1 data jurusan

Tampilan dari halaman ini juga sangat mirip dengan praktik CRUD, dimana terdapat tombol **Edit** dan **Hapus** di sisi kanan atas. Ini dibuat dengan kode di baris 11 – 17. Tombol **Edit** akan mengakses halaman `url('/jurusan/'.$jurusan->id.'/edit')`, dan tombol **Hapus** akan mengirim hasil submit form ke `url('/jurusan/'.$jurusan->id)`.

Di baris 21 – 25 terdapat kode untuk menampilkan flash session pesan, dan diikuti proses menampilkan data jurusan di baris 28 – 30.

View terakhir kita adalah untuk menampilkan form edit:

`resources/views/jurusan/edit.blade.php`

```
1 @extends('layouts.app')
2
3 @section('content')
4 <div class="container mt-3">
```

```

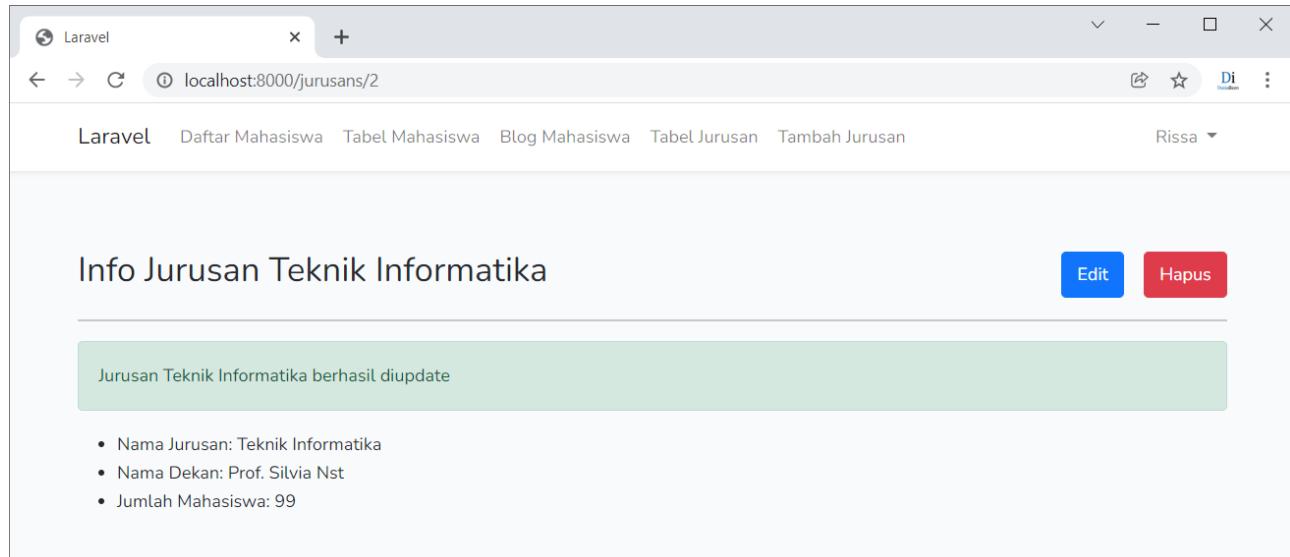
5   <div class="row">
6     <div class="col-sm-8 col-md-6">
7       <h1 class="h2 pt-4">Edit Jurusan</h1>
8       <hr>
9
10      <form action="{{url('/jurusans/'.$jurusan->id)}}" method="POST">
11        @method('PUT')
12        @csrf
13        <div class="mb-3">
14          <label class="form-label" for="nama_jurusan">Nama Jurusan</label>
15          <input type="text"
16            class="form-control @error('nama_jurusan') is-invalid @enderror"
17            id="nama_jurusan" name="nama_jurusan"
18            value="{{ old('nim') ?? $jurusan->nama_jurusan }}">
19            @error('nama_jurusan')
20              <div class="text-danger">{{ $message }}</div>
21            @enderror
22        </div>
23
24        <div class="mb-3">
25          <label class="form-label" for="nama_dekan">Nama Dekan</label>
26          <input type="text"
27            class="form-control @error('nama_dekan') is-invalid @enderror"
28            id="nama_dekan" name="nama_dekan"
29            value="{{ old('nim') ?? $jurusan->nama_dekan }}">
30            @error('nama_dekan')
31              <div class="text-danger">{{ $message }}</div>
32            @enderror
33        </div>
34
35        <div class="mb-3">
36          <label class="form-label" for="jumlah_mahasiswa">
37            Jumlah Mahasiswa</label>
38          <input type="text"
39            class="form-control @error('jumlah_mahasiswa') is-invalid @enderror"
40            id="jumlah_mahasiswa" name="jumlah_mahasiswa"
41            value="{{ old('nim') ?? $jurusan->jumlah_mahasiswa }}">
42            @error('jumlah_mahasiswa')
43              <div class="text-danger">{{ $message }}</div>
44            @enderror
45        </div>
46
47          <button type="submit" class="btn btn-primary my-2">Update</button>
48        </form>
49
50      </div>
51    </div>
52  </div>
53 @endsection

```

Form untuk edit.blade.php hampir sama dengan create.blade.php. Yang berbeda hanya di alamat pengiriman form yang sekarang ke url('/jurusan/'.\$jurusan->id) serta menggunakan @method('PUT'). Selain itu juga perlu 'trik' untuk proses re-populate inputan

form dengan operator ?? atau *null coalescing operator*.

Dengan penambahan view ini, silahkan coba edit dan hapus data jurusan.



Gambar: Pesan proses edit berhasil

Aplikasi kita sudah selesai! Sekarang proses CRUD hanya bisa dilakukan oleh user yang sudah login terlebih dahulu.

Studi kasus dalam bab ini sebenarnya lebih ke latihan CRUD dan juga authentication. Kedua fitur ini sangat memudahkan kita membuat aplikasi web. Materi ini juga menjadi bahan praktik dari bab berikutnya, yakni tentang **Laravel Policy**.

26. Policy

Dalam bahasa Inggris, "policy" berarti "kebijakan". Istilah ini umum dijumpai dalam dunia ekonomi atau pemerintah. Misalnya sebuah perusahaan membuat kebijakan agar karyawan harus datang jam 8:00 pagi dan baru boleh pulang jam 4 sore. Atau pemerintah mengeluarkan kebijakan tentang batas atas dan batas bawah harga tiket pesawat.

Dalam dua contoh ini, kebijakan adalah sebuah *pembatasan*. Di Laravel, policy ini juga dipakai untuk membatasi "sesuatu".

Sebagai bahan praktik, saya akan melanjutkan kode program hasil dari bab sebelumnya, yakni **Case Study: CRUD dan Authentication**. Dengan demikian kita sudah memiliki proses authentication serta tabel jurusan.

26.1. Pengertian Laravel Policy

Di dalam Laravel, **Policy** dipakai untuk membatasi hak akses **user** terhadap sebuah **model**. Lebih spesifik lagi, model yang dimaksud adalah salah satu dari **Restfull** method yang tersedia dari sebuah aplikasi CRUD.

Dengan policy, kita bisa membatasi hanya user dengan nama "admin" saja yang boleh menginput data ke tabel jurusan. Atau user "rissa" tidak bisa menghapus, tetapi dia diberikan untuk melihat dan mengedit data jurusan. Inilah *kebijakan* bisa diatur dengan **Laravel Policy**.

Sekilas policy mirip seperti middleware. Policy bisa disebut sebagai bentuk yang lebih khusus dari middleware.

26.2. Membuat Policy

Policy pada dasarnya berbentuk sebuah class yang disimpan dalam file khusus (mirip seperti file controller, file model, file middleware, dll). Untuk membuat policy, tersedia perintah php artisan dengan format berikut:

```
php artisan make:policy <NamaPolicy>
```

Karena policy akan dipakai untuk membatasi akses ke suatu model, bisa ditambah option -m dengan format sebagai berikut:

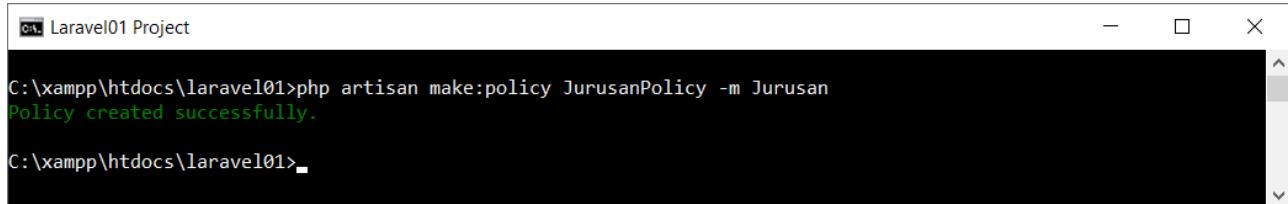
Policy

```
php artisan make:policy <NamaPolicy> -m NamaModel
```

Dengan tambahan option `-m`, class Policy otomatis berisi struktur dasar untuk membatasi model (sudah berisi berbagai method yang siap pakai).

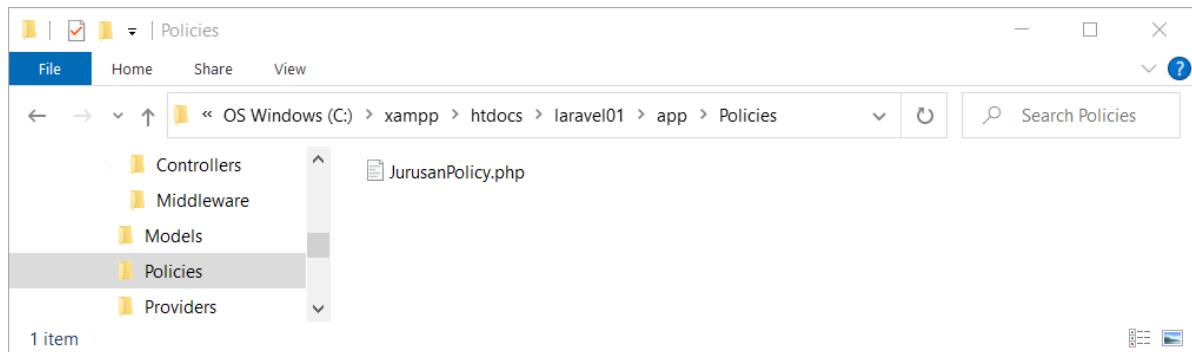
Sebagai contoh, saya ingin membuat **JurusanPolicy** yang akan kita pakai untuk membatasi akses ke **Jurusan** model. Perintah yang diperlukan adalah:

```
php artisan make:policy JurusanPolicy -m Jurusan
```



Gambar: Pembuatan JurusanPolicy

Setelah menjalankan perintah ini, **JurusanPolicy.php** bisa diakses dari folder `app\Policies`:



Gambar: File JurusanPolicy.php

Berikut isi file ini:

app/Policies/JurusanPolicy.php

```
1 <?php
2
3 namespace App\Policies;
4
5 use App\Models\Jurusan;
6 use App\Models\User;
7 use Illuminate\Auth\Access\HandlesAuthorization;
8
9 class JurusanPolicy
10 {
11     use HandlesAuthorization;
12
13     public function viewAny(User $user)
14     {
15         //
16     }
17 }
```

Policy

```
18     public function view(User $user, Jurusan $jurusan)
19     {
20         //
21     }
22
23     public function create(User $user)
24     {
25         //
26     }
27
28     public function update(User $user, Jurusan $jurusan)
29     {
30         //
31     }
32
33     public function delete(User $user, Jurusan $jurusan)
34     {
35         //
36     }
37
38     public function restore(User $user, Jurusan $jurusan)
39     {
40         //
41     }
42
43     public function forceDelete(User $user, Jurusan $jurusan)
44     {
45         //
46     }
47 }
```

Class `JurusanPolicy` butuh mengimport 2 buah class model, yakni `User` model (berisi data user yang sedang login), serta `Jurusan` model (tabel yang aksesnya ingin kita batasi). Selain itu juga perlu mengimport class `Illuminate\Auth\Access\HandlesAuthorization` yang berfungsi sebagai class dasar untuk operasional policy.

Terdapat 7 method di dalam `JurusanPolicy` yang nantinya mengembalikan nilai `true` atau `false` tergantung batasan yang ingin kita buat. Semua method memiliki pasangan dengan method RESTfull di Controller:

Policy Method	Controller Method
<code>viewAny()</code>	<code>index()</code>
<code>view()</code>	<code>show()</code>
<code>create()</code>	<code>store()</code>
<code>update()</code>	<code>update()</code>
<code>delete()</code>	<code>destroy()</code>
<code>restore()</code>	<code>restore()</code>
<code>forceDelete()</code>	<code>forceDelete()</code>

Tabel policy method ini saya ambil dari dokumentasi laravel tentang [Authorization](#).

Cara bacanya adalah, jika kita ingin membatasi user agar tidak bisa melihat satu data tabel yang di proses oleh method `show()` di controller, maka kode programnya di tulis dalam method `view()` di Policy.

Atau jika kita ingin membatasi agar user tidak bisa menghapus data tabel (yang di proses oleh method `destroy()` di controller), maka kode programnya harus ditulis dalam method `delete()` di Policy.

Penjelasan di atas memang agak membingungkan, ini akan lebih jelas saat kita masuk ke praktek kode program.

26.3. Membatasi Proses Create

Sebagai contoh pertama, saya ingin membatasi proses input data jurusan. Di dalam `JurusanController`, proses input ini ditangani oleh method `store()`. Dengan melihat daftar tabel sebelumnya, method `store()` berpasangan dengan method `create()` di `JurusanPolicy`.

Oleh karena itu silahkan tambah kode berikut ke dalam method `create()`:

app/Policies/JurusanPolicy.php

```

1 <?php
2 /**
3 class JurusanPolicy
4 {
5     public function create(User $user)
6     {
7         return $user->email === 'admin@gmail.com';
8     }
9 }
```

Kita sudah singgung sedikit bahwa setiap method di dalam Policy harus mengembalikan nilai **true** atau **false**. Jika hasilnya true, maka proses bisa dijalankan. Jika hasilnya false, proses tersebut akan ditolak.

Dalam contoh ini saya ingin memeriksa apakah email dari user yang sedang login sama dengan '`admin@gmail.com`' atau tidak. Jika sama (true), maka proses input data jurusan bisa dilakukan.

Informasi mengenai alamat email dari user yang sedang login bisa diakses dari `$user->email`, dimana variabel `$user` berasal dari penulisan argument method `create()` di baris 5.

Syarat berikutnya, kita harus tambahkan perintah khusus ke dalam method `store()` di `JurusanController`:

app/Http/Controllers/JurusanController.php

```

1 <?php
2 /**
```

Policy

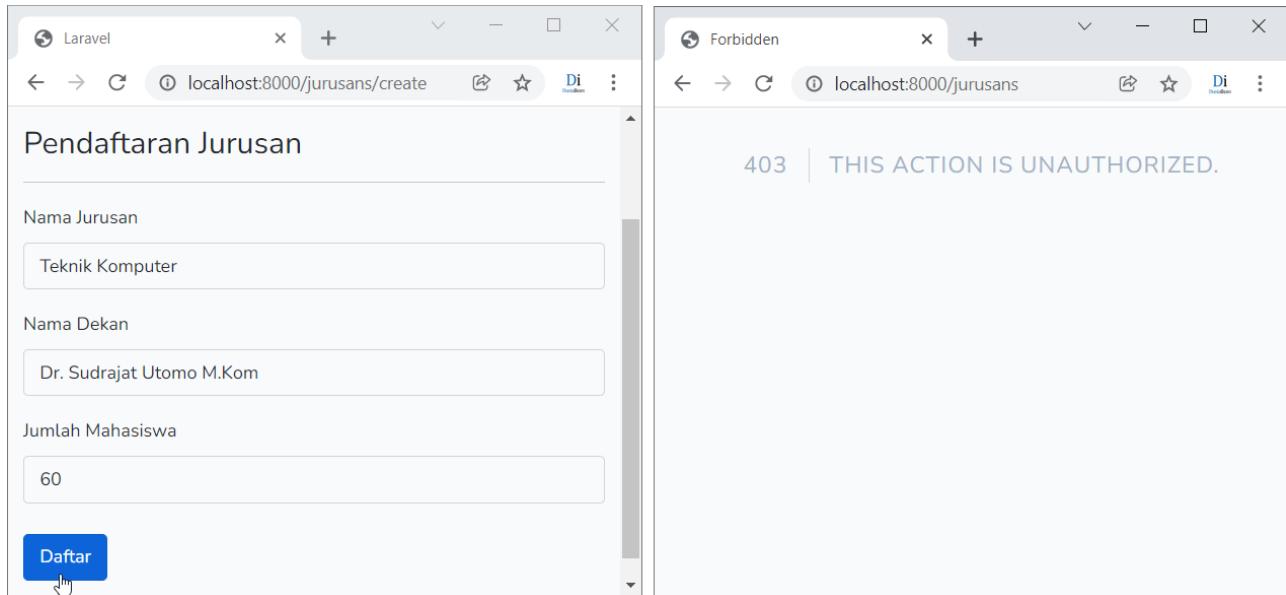
```
3 class JurusanController extends Controller
4 {
5     public function store(Request $request)
6     {
7         // batasi hak akses untuk proses store
8         $this->authorize('create', Jurusan::class);
9
10        $validateData = $request->validate([
11            'nama_jurusan'          => 'required',
12            ...
13        });
14    }
}
```

Tambahan perintah yang dimaksud ada di baris 8, yakni method `$this->authorize()`.

Method `$this->authorize()` butuh 2 buah argument. Argument pertama berupa nama method yang dipakai dalam Policy class, serta argument kedua diisi dengan class Model yang akan dibatasi. Dalam contoh ini, nama method yang kita pakai adalah 'create' serta class yang akan dibatasi adalah `Jurusan::class`.

Save file policy dan controller ini, dan mari kita coba.

Saya sudah memiliki 1 user bernama "Rissa" hasil dari praktek bab sebelumnya. User ini memiliki alamat email "rissa@gmail.com". Berikut percobaan menambah 1 jurusan baru:

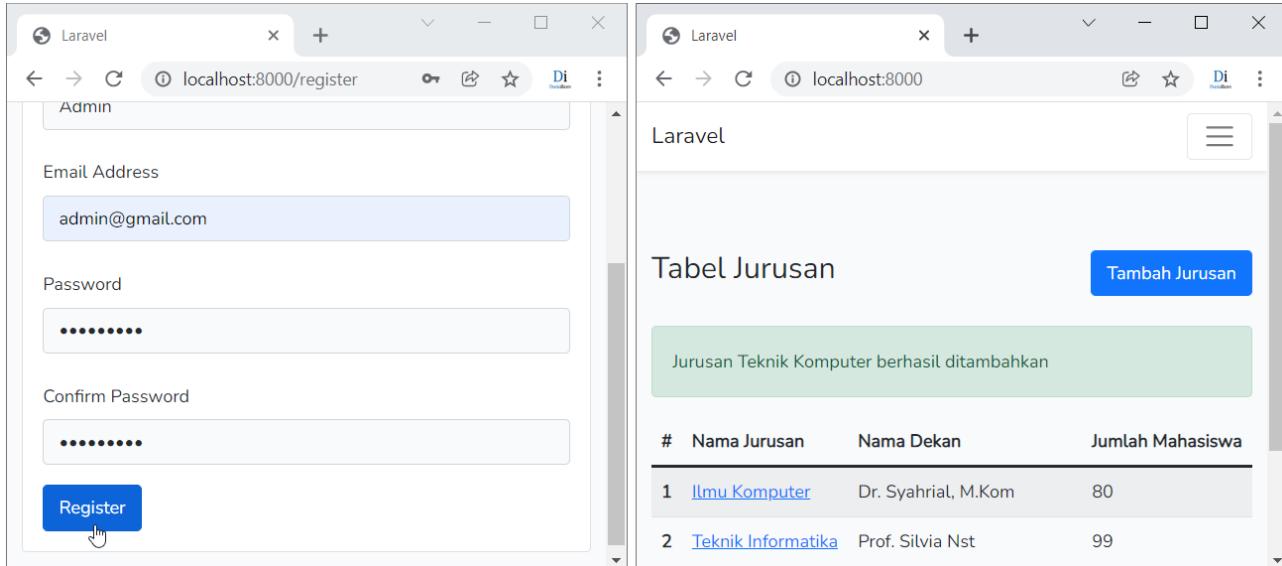


Gambar: Proses input data jurusan

Tepat pada saat tombol "**Daftar**" di tekan, tampil error "403 | This action is unauthorized". Yang berarti user Rissa tidak diizinkan menginput data baru.

Percobaan selanjutnya, silahkan logout dari user Rissa, lalu buat user baru bernama "Admin" dengan email "admin@gmail.com". Kemudian lakukan proses yang sama:

Policy



Gambar: Proses input berhasil

Proses input jurusan berhasil! Karena alamat email 'admin@gmail.com' sesuai dengan syarat di method `create()`. Inilah cara kerja dari **Laravel Policy**.

Syarat policy ini juga sangat fleksibel, tidak harus email tapi juga bisa menggunakan data lain untuk proses identifikasi user. Misalnya bisa saya tulis syarat seperti ini:

```
public function create(User $user)
{
    return $user->name === 'Admin';
}
```

Kode ini akan mengizinkan user dengan name 'Admin' untuk melakukan input data. Tapi hati-hati karena secara bawaan kolom `name` di tabel `users` bawaan Laravel tidak di set dengan atribut `unique`. Sehingga bisa saja terdapat lebih dari 1 user bernama 'Admin' dan siapa saja bisa mendaftarkan diri dengan nama 'Admin'.

Bagaimana jika kita ingin ada 2 user yang diberi hak akses? Tidak masalah, yang perlu diubah hanya di logika pemeriksaan user.

Perhatikan contoh berikut:

```
public function create(User $user)
{
    return ($user->email === 'admin@gmail.com')
        OR ($user->email === 'support@gmail.com');
}
```

Di sini saya menggabung 2 buah operasi perbandingan dengan operator `OR`, sehingga user dengan email `admin@gmail.com` dan `support@gmail.com` akan bisa melakukan proses input data.

Agar lebih rapi, logika di atas juga bisa ditulis menggunakan fungsi `in_array()`:

```
public function create(User $user)
{
    return in_array($user->email,[  

        'admin@gmail.com',  

        'support@gmail.com',
    ]);
}
```

Fungsi `in_array()` merupakan function bawaan PHP yang dipakai untuk memeriksa apakah alamat email di argument pertama ada di dalam array yang diinput sebagai argument kedua.

Untuk aplikasi yang lebih kompleks, proses pemeriksaan hak akses ini bisa dimodifikasi lebih lanjut, misalnya dengan menambah kolom 'role' ke tabel `users`. Jika nilai dari kolom role adalah 'admin', beri hak akses. Sehingga kita bisa membuat semacam pembagian tugas untuk user yang bertindak sebagai admin, moderator, editor, dsb.

26.4. Membatasi Tampilan di View

Saat ini proses input data hanya bisa dilakukan oleh user `admin`. Namun di dalam view, menu dan tombol "Tambah Jurusan" tetap tampil untuk semua user. Idealnya, tombol tersebut hanya bisa dilihat untuk user `admin` saja, atau lebih baik lagi hanya bisa dilihat oleh user yang memiliki hak akses untuk menambah data.

Blade memiliki fitur khusus untuk keperluan tersebut, yakni perintah `@can` dan `@cannot` serta pasangan penutup `@endcan` dan `@endcannot`. Perintah `@can` dan `@cannot` butuh 2 argument yang sama seperti method `$this->authorize()` di controller. Berikut contoh penggunaannya:

```
1 @can('create', App\Models\Jurusan::class)  

2     <p>Hanya bisa dilihat oleh user yang bisa men-create jurusan</p>  

3 @endcan
```

Dengan penulisan ini, kode HTML di baris 2 hanya bisa dilihat oleh user yang memiliki hak akses untuk melakukan proses `create` ke dalam tabel `jurusan`. Perhatikan penulisan argument '`create`' dan `App\Models\Jurusan::class` untuk perintah `@can`. Ini sama seperti yang kita pakai di dalam method `$this->authorize()`.

Sebaliknya, perintah `@cannot` hanya bisa dilihat oleh user yang tidak memiliki hak akses:

```
1 @cannot('create', App\Models\Jurusan::class)  

2     <p>Hanya bisa dilihat oleh user yang tidak bisa men-create jurusan</p>  

3 @endcan
```

Sekarang kode HTML di baris 2 hanya boleh dilihat oleh user yang **tidak bisa** melakukan proses `create` ke dalam tabel `jurusan`.

Dalam praktek kita, terdapat 2 komponen yang memiliki link ke form tambah jurusan, yakni menu navbar dan tombol di sisi kanan atas. Untuk menu navbar, kode programnya ada di view

app\layouts.blade.php. Penulisan menu navbar bisa dimodifikasi sebagai berikut:

resources/views/layouts/app.blade.php

```
1 <ul class="navbar-nav me-auto">
2     // ...
3     <li class="nav-item">
4         <a class="nav-link" href="{{url('/')}}">Tabel Jurusan</a>
5     </li>
6     @can('create', App\Models\Jurusan::class)
7         <li class="nav-item">
8             <a class="nav-link" href="{{url('/jurusans/create')}}">Tambah Jurusan</a>
9         </li>
10    @endcan
11 </ul>
```

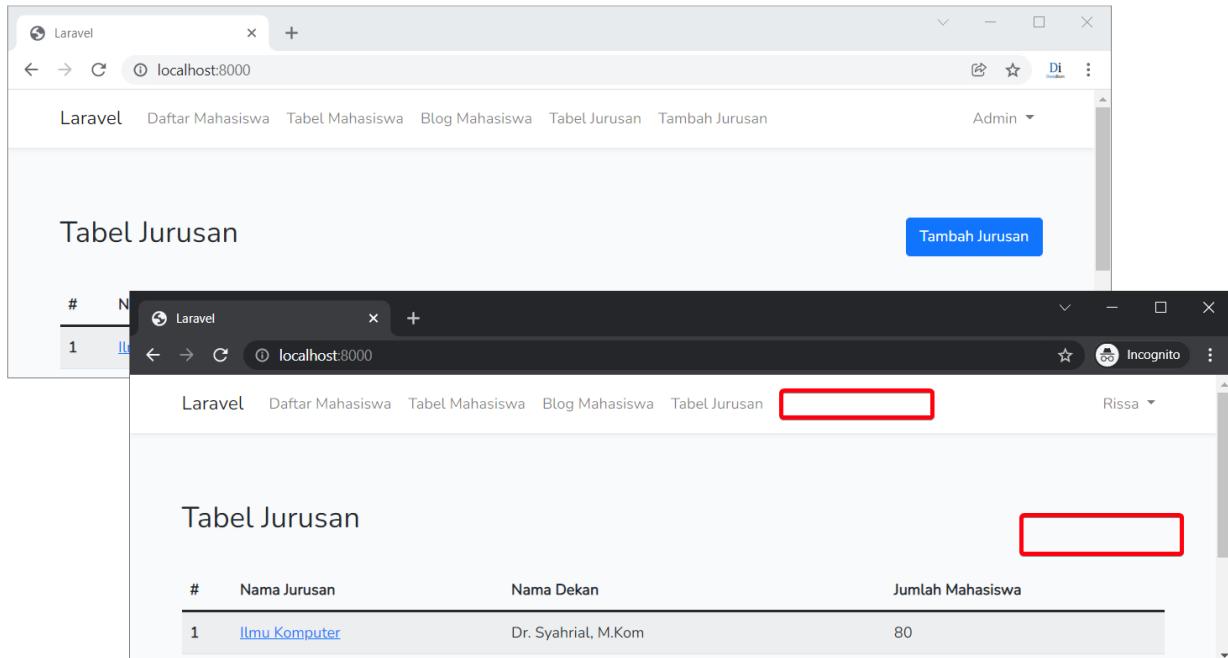
Dengan tambahan ini, menu **Tambah Jurusan** hanya bisa dilihat oleh user yang memiliki hak akses saja.

Komponen kedua adalah tombol "**Tambah Jurusan**" di view jurusan\index.blade.php:

resources/views/jurusan/index.blade.php

```
1 <div class="py-4 d-flex justify-content-end align-items-center">
2     <h1>Tabel Jurusan</h1>
3     @can('create', App\Models\Jurusan::class)
4         <a href="{{url('/jurusans/create')}}" class="btn btn-primary">
5             Tambah Jurusan
6         </a>
7     @endcan
8 </div>
```

Hasilnya, terdapat perbedaan tampilan dari admin yang memiliki hak akses, dengan user biasa:



Gambar: Tampilan user admin (atas) dan user rissa (bawah)

Dengan perintah @can dan @cannot, kita bisa membedakan tampilan untuk masing-masing user (sesuai hak aksesnya).

26.5. Membatasi Proses Delete

Percobaan selanjutnya adalah membatasi proses delete. Di dalam JurusanController, method yang bertindak untuk proses delete adalah `destroy()`. Berdasarkan tabel pasangan method sebelumnya, method `destroy()` di Controller berpasangan dengan method `delete()` di Policy.

Berikut isi dari method `delete()` di `JurusanPolicy.php`:

app/Policies/JurusanPolicy.php

```
1 public function delete(User $user, Jurusan $jurusan)
2 {
3     return in_array($user->email,[
4         'admin@gmail.com'
5     ]);
6 }
```

Dengan kode ini, maka hanya user dengan email `admin@gmail.com` saja yang bisa melakukan proses delete. Berikutnya, kita harus daftarkan batasan hak akses ke method `destroy()` di `JurusanController.php`:

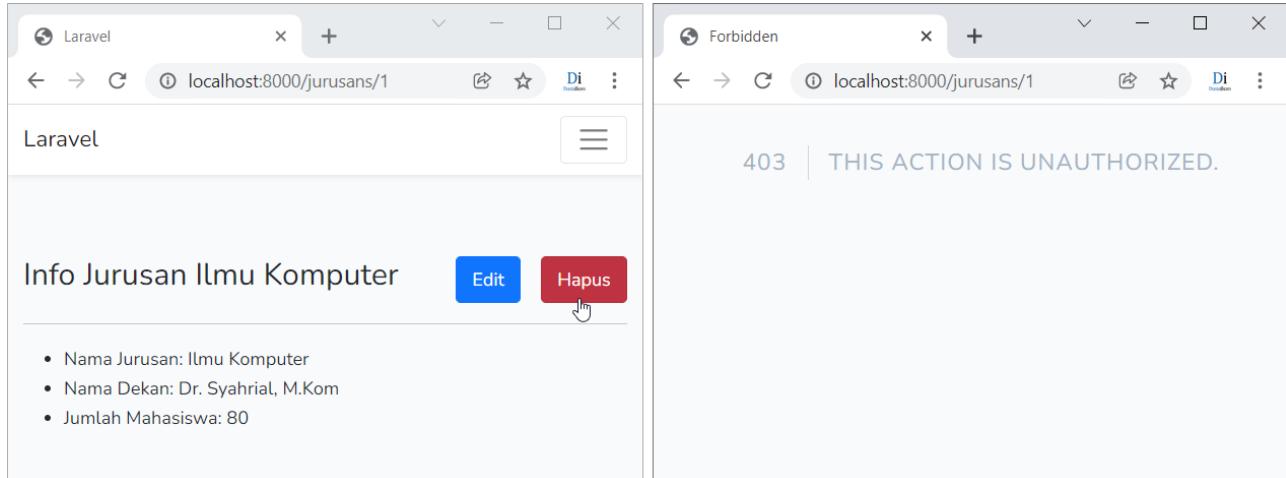
app/Http/Controllers/JurusanController.php

```
1 public function destroy(Jurusan $jurusan)
2 {
3     $this->authorize('delete',$jurusan);
4
5     $jurusan->delete();
6     return redirect('/')
7     ->with('pesan',"Jurusan $jurusan->nama_jurusan berhasil dihapus");
8 }
```

Ada sedikit perubahan di perintah `$this->authorize()`, sekarang penulisan argument kedua tidak lagi `Jurusan::class`, tapi cukup variabel `$jurusan`. Ini karena di dalam method `destroy()` kita sudah memiliki akses ke `Jurusan` model yang berasal dari penulisan argument di baris 3, sedangkan di method `create()` sebelumnya, variabel ini masih belum tersedia.

Mari coba pembatasan ini. Dengan user `rissa`, proses penghapusan akan ditolak:

Policy



Gambar: User rissa tidak bisa menghapus jurusan

Tapi jika kita login dengan user **admin**, proses penghapusan ini bisa dilakukan:

The image contains two side-by-side browser windows. The left window, titled 'Laravel', shows the same 'Info Jurusan Ilmu Komputer' page as before, with the 'Hapus' button being clicked. The right window, also titled 'Laravel', shows a 'Tabel Jurusan' page with a green success message: 'Jurusan Ilmu Komputer berhasil dihapus'. Below the message is a table with columns: '#', 'Nama Jurusan', 'Nama Dekan', and 'Jumlah Mahasiswa'.

Gambar: User admin berhasil menghapus jurusan

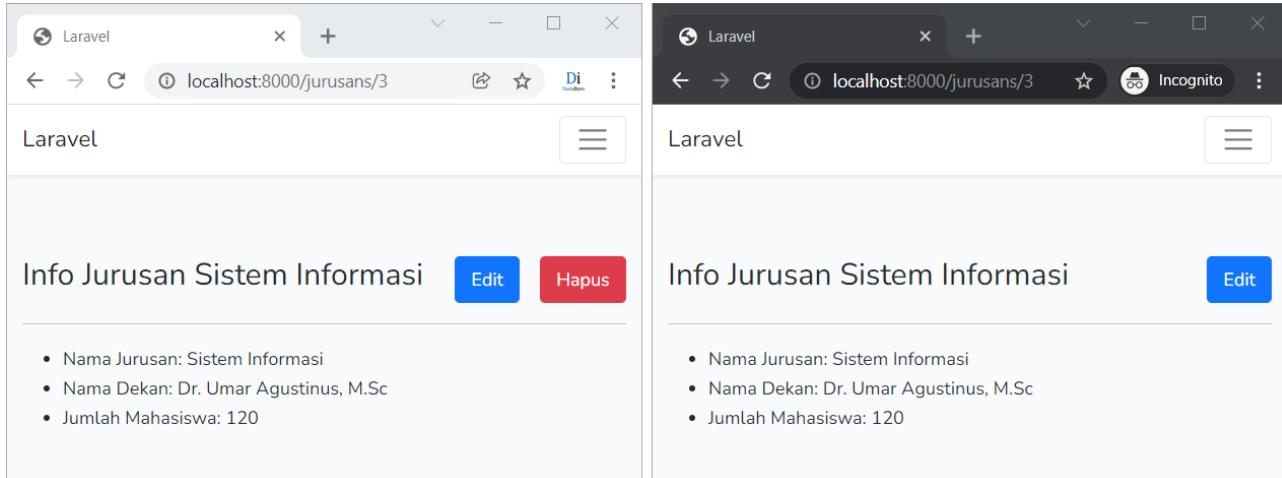
Sip, pembatasan hak akses delete sudah berhasil. Sekarang, bagaimana cara menyembunyikan tombol **Hapus** ini? Sama seperti sebelumnya, kita harus cari view tempat tombol tersebut dibuat, lalu pindahkan ke dalam blok @can:

resources/views/jurusan/show.blade.php

```
1 //...
2 @can('delete',$jurusan)
3 <form action="{{url('/jurusans/'.$jurusan->id)}}" method="POST">
4     @method('DELETE')
5     <button type="submit" class="btn btn-danger ms-3">Hapus</button>
6     @csrf
7 </form>
8 @endcan
9 //...
```

Sedikit perubahan, argument kedua dari perintah @can diisi \$jurusan, bukan lagi App\Models\

Jurusan::class. Ini karena di dalam view show.blade.php, kita juga sudah bisa mengakses variabel \$jurusan yang berisi **Jurusan**. Variabel \$jurusan ini yang berasal dari method show() di JurusanController, yakni method yang dipakai untuk membuka view show.blade.php.



Gambar: Tampilan user admin (kiri) dan user rissa (kanan)

26.6. Pembatasan di Route

Sampai di sini kita sudah membatasi 2 buah RESTfull method, yakni proses **create** dan **delete**. Caranya adalah dengan menulis syarat pembatasan di policy, kemudian menerapkannya ke dalam method controller.

Selain ditulis ke dalam method controller, penerapan batasan policy ini juga bisa dilakukan dari route menggunakan "bantuan" middleware.

Sebagai bahan praktik, saya ingin membatasi akses ke tampilan 1 data tabel yang ditangani oleh method **view()**:

app/Policies/JurusanPolicy.php

```
1 public function view(User $user, Jurusan $jurusan)
2 {
3     return in_array($user->email, [
4         'admin@gmail.com'
5     ]);
6 }
```

Method **view()** ini berpasangan dengan method **show()** di JurusanController, yakni method yang dipakai untuk melihat 1 data detail seperti `localhost:8000/jurusans/1`, `localhost:8000/jurusans/2`, dst. Alamat inilah yang ingin kita batasi.

Jika memakai cara sebelumnya, tinggal tambah perintah `$this->authorize('view', $jurusan)` ke dalam method **show()** di JurusanController. Namun kali ini saya ingin pakai cara alternatif dengan menulisnya di route.

Policy

Caranya adalah, men-chaning pemanggilan method berikut ke dalam route:

```
->middleware('can:<nama_method_policy>,<class_model_yang_dibatasi>')
```

Method ini harus ditulis ke dalam route yang sesuai dengan method policy. Dalam contoh kita, method `view()` di `JurusanPolicy` dipakai untuk membatasi hak akses method `show()` di `JurusanController`, maka penulisan route-nya menjadi:

```
Route::get('jurusans/{jurusan}',[JurusanController::class,'show'])  
->name('jurusans.show')->middleware('can:view,jurusan');
```

Perhatikan argument dari `middleware()`, yakni 'can:view,jurusan'. Inilah cara membatasi route dengan policy.

Jika kita membuka file `routes\web.php` saat ini, route di atas tidak akan ditemukan karena sudah diwakili oleh `Route::resource`:

```
Route::resource('jurusans',JurusanController::class)->middleware('auth');
```

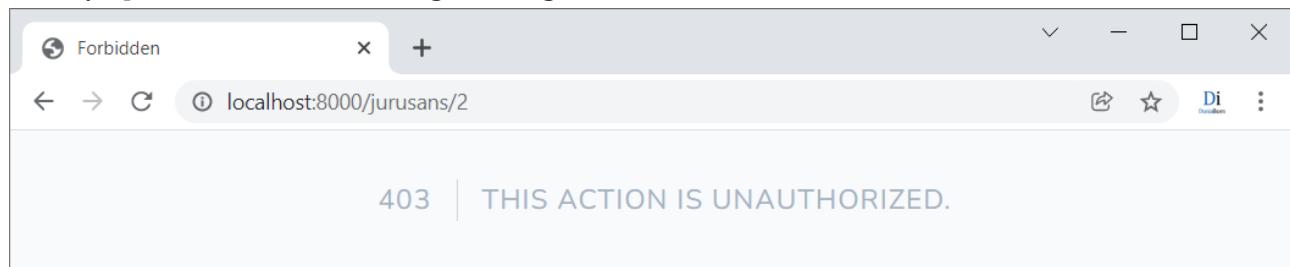
Route ini secara tidak langsung mewakili 7 route untuk proses CRUD, termasuk route `show()` yang ingin kita batasi. Agar policy bisa diterapkan, timpa dengan membuat route baru:

`routes\web.php`

```
1 ...  
2 Route::resource('jurusans',JurusanController::class)->middleware('auth');  
3  
4 Route::get('jurusans/{jurusan}',[JurusanController::class,'show'])  
5 ->name('jurusans.show')->middleware('auth')->middleware('can:view,jurusan');
```

Meskipun di dalam `Route::resource` sudah terdapat route yang sama, tapi route di baris 4-5 akan menimpa route tersebut. Di sini saya menulis 2 buah middleware, yakni `middleware('auth')` untuk batasan authentication, serta `middleware('can:view,jurusan')` untuk batasan policy.

Saatnya percobaan. Silahkan login sebagai user biasa, lalu akses `localhost:8000/jurusans/2`:



Gambar: Hak akses ditolak

Hasilnya, tampil error "403 | This action is unauthorized" karena user biasa tidak diizinkan mengakses halaman `show()`.

Tes lagi dengan user admin, dan hasilnya halaman bisa di akses seperti biasa.

Policy

The screenshot shows a browser window with the URL `localhost:8000/jurusans/2`. The page title is "Info Jurusan Teknik Informatika". Below the title, there is a list of items:

- Nama Jurusan: Teknik Informatika
- Nama Dekan: Prof. Silvia Nst
- Jumlah Mahasiswa: 99

At the top right of the page, there are two buttons: "Edit" (blue) and "Hapus" (red).

Gambar: Halaman show() bisa ditampilkan

Pembatasan policy di route merupakan cara alternatif selain menulis perintah `$this->authorize()` ke dalam controller.

Sedikit catatan, penulisan method chaining `middleware('can:view,jurusan')` tidak boleh berisi spasi, misalnya jika ditulis sebagai `middleware('can:view, jurusan')` maka policy tidak akan berjalan.

Exercise

Dengan tambahan batasan untuk melihat 1 data jurusan, maka link tabel jurusan di halaman `index.blade.php` sudah tidak relevan lagi. Sebagai latihan, bisakah anda mengubah kode view `index.blade.php` agar link jurusan hanya tampil untuk user yang memiliki hak akses saja?

Berikut tampilan yang dimaksud:

The screenshot shows two browser windows side-by-side. Both windows have the URL `localhost:8000` and the title "Laravel".

Left Window (Administrator View):

#	Nama Jurusan	Nama Dekan	Jumlah Mahasiswa
1	Teknik Informatika	Prof. Silvia Nst	99
2	Sistem Informasi	Dr. Umar Agustinus, M.Sc	120
3	Teknik Komputer	Dr. Sudrajat Utomo M.Kom	60

Right Window (User View):

#	Nama Jurusan	Nama Dekan	Jumlah Mahasiswa
1	Teknik Informatika	Prof. Silvia Nst	99
2	Sistem Informasi	Dr. Umar Agustinus, M.Sc	120
3	Teknik Komputer	Dr. Sudrajat Utomo M.Kom	60

Gambar: Tampilan admin (kiri) dan user biasa (kanan)

Ketika halaman dibuka oleh user `admin` yang memiliki hak akses, tampilkan link seperti biasa (gambar kiri). Namun jika halaman di buka oleh user biasa yang tidak memiliki hak

akses, cukup tampilkan nama jurusan tanpa link (gambar kanan).

Answer

Untuk keperluan ini, perintah yang kita butuhkan adalah `@can` dan `@cannot`. Berikut kode programnya:

`resources/views/jurusan/index.blade.php`

```

1  ...
2  <tbody>
3  @forelse ($jurusans as $jurusan)
4  <tr>
5    <th>{{$loop->iteration}}</th>
6    <td>
7      @can('view', $jurusan)
8        <a href="{{ url('/jurusans/'.$jurusan->id) }}">
9          {{$jurusan->nama_jurusan}}
10         </a>
11       @endcan
12     @cannot('view', $jurusan)
13       {{$jurusan->nama_jurusan}}
14     @endcannot
15   </td>
16   <td>{{$jurusan->nama_dekan}}</td>
17   <td>{{$jurusan->jumlah_mahasiswa}}</td>
18 </tr>
19 ...

```

Blok kode program `@can('view', $jurusan)` hanya tampil untuk user yang memiliki hak akses `view()`. Jika ini dipenuhi, buat link ke `url('/jurusans/'.$jurusan->id)`.

Sedangkan blok perintah `@cannot('view', $jurusan)` hanya dipenuhi jika user tidak memiliki hak akses untuk `view()`. Untuk hal ini, cukup tampilkan teks nama jurusan tanpa tag `<a>`.

Dalam bab ini kita telah membahas cara pembatasan hak akses menggunakan Laravel Policy. Materi ini sangat bermanfaat untuk pembuatan aplikasi CRUD karena seharusnya tidak semua user bisa melakukan hal yang sangat krusial seperti penghapusan data. Dengan policy, hanya user yang berhak saja bisa melakukan hal tersebut.

Berikutnya, kita akan masuk ke bab terakhir dalam buku ini, yakni **Case Study: ILKOON Profile Manager**.

27. Case Study: ILKOOM Profile Manager

Dalam bab ini saya ingin membuat case study yang sedikit kompleks. Kita akan merancang aplikasi *profile manager*, dimana setiap user bisa mendaftarkan diri dengan mengisi beberapa inputan form serta mengupload gambar profil. Semua user bisa melihat profil user lain, tapi hanya bisa mengedit data user itu sendiri (pembatasan hak akses).

Studi kasus ini merangkum hampir semua materi yang kita bahas sepanjang buku, mulai dari konsep MVC, blade, Laravel mix, migration, eloquent, form processing, form validation, localization, file upload, middleware, serta policy.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, saya akan mulai dari installer baru Laravel 9. Anda bisa hapus isi folder **laravel01**, atau menginstall Laravel ke folder lain, misalnya **laravel02**.

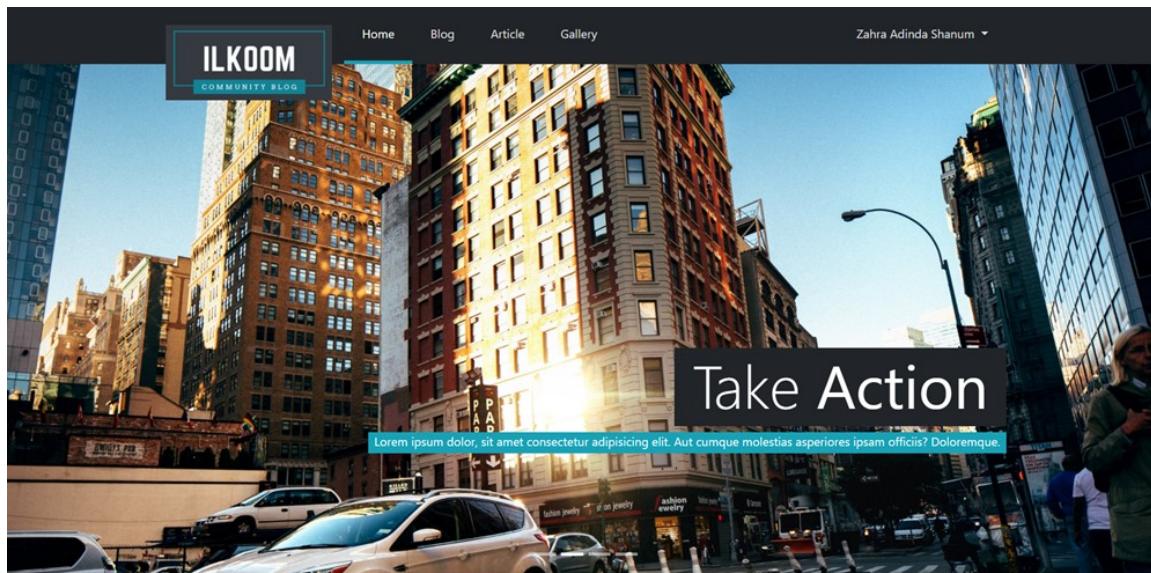
Berikut perintah untuk menginstall Laravel ke folder **laravel01**:

```
composer create-project laravel/laravel="^9.0" laravel01
```

Dalam bab ini kita juga akan menjalankan ulang proses migration, maka jika ada tabel di database **laravel**, silahkan hapus terlebih dahulu.

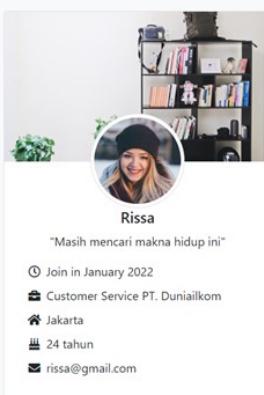
Berikut tampilan akhir dari studi kasus kita:

Case Study: ILKOOM Profile Manager



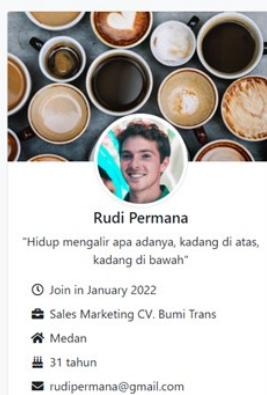
Member List

Lore ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, vitae!



Rissa
"Masih mencari makna hidup ini"

⌚ Join in January 2022
📍 Customer Service PT. DuniaIlkom
🏡 Jakarta
📅 24 tahun
✉ rissa@gmail.com



Rudi Permana
"Hidup mengalir apa adanya, kadang di atas, kadang di bawah"

⌚ Join in January 2022
📍 Sales Marketing CV. Bumi Trans
🏡 Medan
📅 31 tahun
✉ rudipermana@gmail.com



Zahra Adinda Shanum
"Mencari berkah untuk dunia dan akhirat"

⌚ Join in January 2022
📍 CEO Zahra Cosmetics
🏡 Yogyakarta
📅 25 tahun
✉ zahraadinda@icloud.com



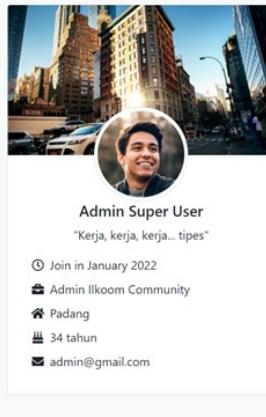
Anissa Lestari
"Hidup hanya sekali, harus bermakna dan bermanfaat untuk orang lain"

⌚ Join in January 2022
📍 Assistant Manager PT. Jaya Selalu
🏡 Surabaya
📅 21 tahun
✉ anissa@gmail.com



Budi Hardika Oktavianus Saputra
"Berkarya tiada hentil"

⌚ Join in January 2022
📍 Wiraswasta
🏡 Makassar
📅 34 tahun
✉ budihardika@outlook.com



Admin Super User
"Kerja, kerja, kerja... tips"

⌚ Join in January 2022
📍 Admin Ilkoom Community
🏡 Padang
📅 34 tahun
✉ admin@gmail.com



ILKOOM
COMMUNITY BLOG

Lore ipsum dolor sit amet consectetur adipisicing elit. Aperiam cumque, esse modi maxime veniam nulla delectus dolorem

© ILKOOM 2022

Community

- Activity
- Members
- Groups
- Forums

Our Services

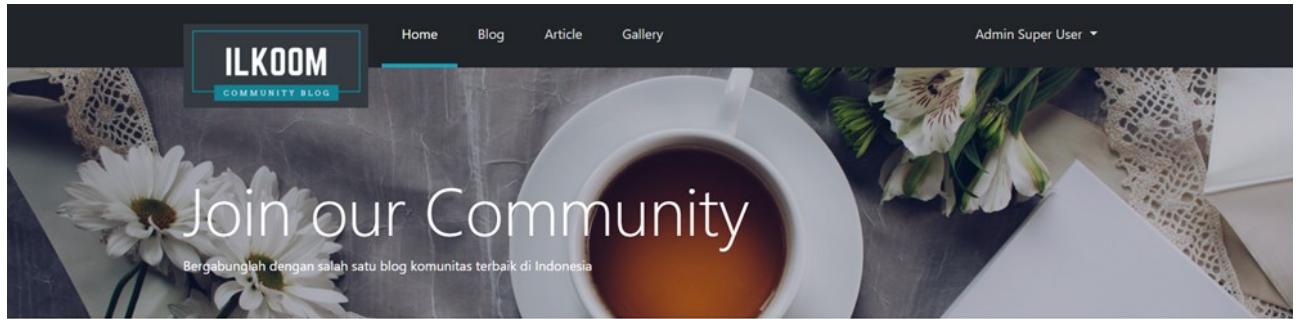
- Our mission
- Help/Contact Us
- Privacy Policy
- Cookie Policy
- Terms & Conditions

Hubungi Kami

- ✉ andre@ilkoom.com
- 📞 (021) 123456
- 🌐 www.ilkoom.com



Gambar: Tampilan halaman utama ILKOOM Profile Manager



Edit Data

Email *

Nama *

Tanggal Lahir * April

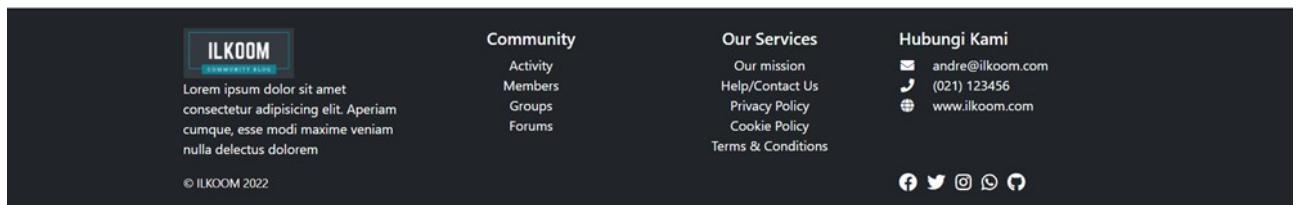
Pekerjaan

Kota

Bio Profil

Gambar Profil 
 No file chosen

Background Profil 



Gambar: Tampilan halaman form register ILKOOM Profile Manager

Jika anda pernah membaca buku **Bootstrap Uncover** duniaIlkom, sudah bisa langsung menebak kalau design tampilan yang saya pakai adalah studi kasus bab terakhir dari buku tersebut, yakni **ILKOOM Community Blog Template**.

Untuk menghemat tempat, saya tidak akan membahas kode-kode CSS dan Bootstrap yang dipakai, kita fokus ke kode program Laravel saja.

27.1. Instalasi Laravel Authentication

Setelah menginstall Laravel, proses pertama yang kita butuhkan adalah instalasi Laravel Authentication yang tidak lain juga menginstall **Laravel UI** dan **npm** (node module).

Pertama, install **Laravel UI** dengan perintah:

```
composer require laravel/ui="^3.4"
```

Setelah itu install **Laravel Authentication**:

```
php artisan ui bootstrap --auth
```

Lanjutkan dengan proses install **npm**, **laravel mix** dan **compile assets**:

```
npm install
```

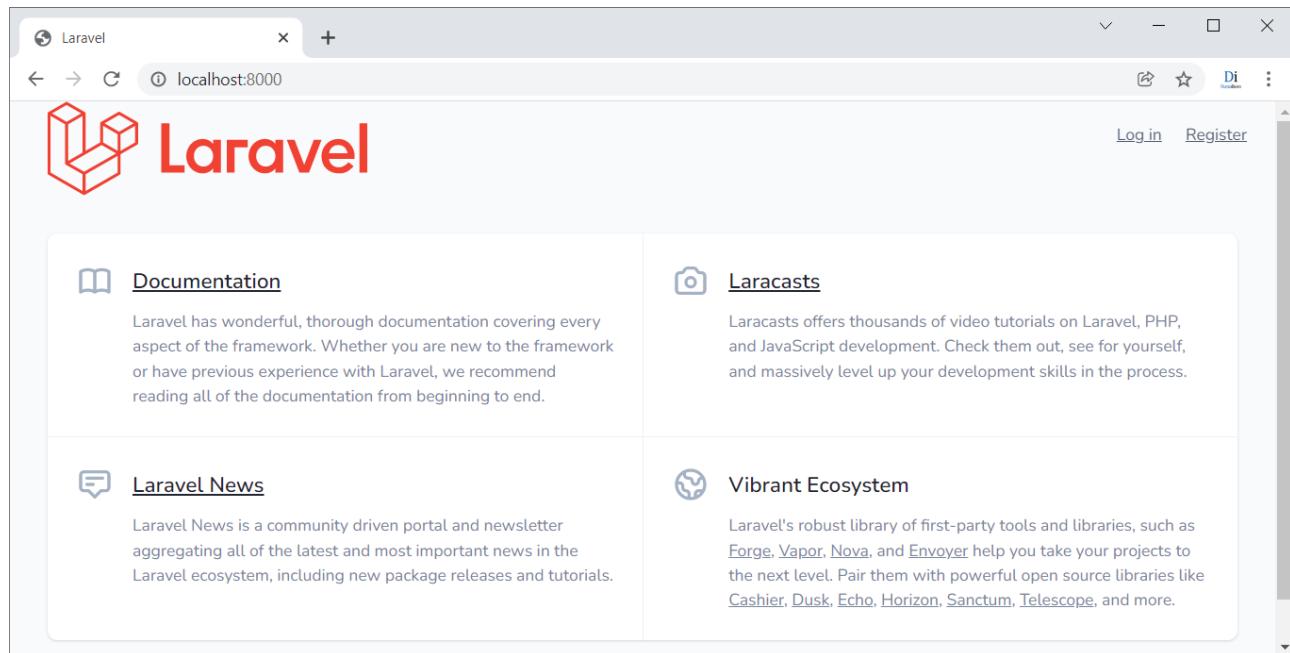
```
npm install laravel-mix --save-dev
```

```
npx mix
```

Setelah instalasi selesai, jalankan langsung **migration**:

```
php artisan migrate:fresh
```

Dan tes buka halaman <http://localhost:8000> di web browser:



Gambar: Halaman default Laravel + Authentication

Sip, fitur authentication sudah terinstall dan siap digunakan.

27.2. Modifikasi Tabel Users

Dalam studi kasus ini saya ingin membuat data profil user yang mencakup `username`, `password`, `email`, `tanggal_lahir`, `gambar_profil`, dan `pesan_profil`.

Sebenarnya terdapat 2 pilihan, apakah membuat tabel baru atau memodifikasi tabel `users` bawaan authentication. Jika membuat tabel baru, maka kita harus membuat sendiri fitur authentication. Sedangkan jika memodifikasi tabel `users`, fitur authentication sudah tersedia namun perlu mengubah beberapa pengaturan.

Kali ini saya akan memilih untuk modifikasi tabel `users`. Meskipun sudah terdapat fitur login dan logout bawaan, akan tetapi nantinya tetap perlu memodifikasi berbagai file agar sesuai dengan keinginan.

Pertama, saya akan modifikasi file migration untuk tabel `users`. Silahkan buka file `2014_10_12_000000_create_users_table.php`. Berikut isi dari method `up()` yang sudah ada:

`database/migrations/2014_10_12_000000_create_users_table.php`

```

1 public function up()
2 {
3     Schema::create('users', function (Blueprint $table) {
4         $table->id();
5         $table->string('name');
6         $table->string('email')->unique();
7         $table->timestamp('email_verified_at')->nullable();
8         $table->string('password');
9         $table->rememberToken();
10        $table->timestamps();
11    });
12 }
```

Saya akan modifikasi menjadi:

```

1 public function up()
2 {
3     Schema::create('users', function (Blueprint $table) {
4         $table->id();
5         $table->string('email')->unique();
6         $table->string('password');
7         $table->string('nama');
8         $table->date('tanggal_lahir');
9         $table->string('pekerjaan')->nullable();
10        $table->string('kota')->nullable();
11        $table->text('bio_profil')->nullable();
12        $table->string('gambar_profil')->default('default_profile.jpg');
13        $table->tinyInteger('background_profil')->default(1);
14        $table->timestamp('email_verified_at')->nullable();
15        $table->rememberToken();
16        $table->timestamps();
17    });
18 }
```

Perubahan pertama adalah kolom 'name' yang saya tukar menjadi 'nama' dan posisinya di pindah ke baris 7. Posisi kolom ini tidak berpengaruh apa-apa, hanya sekedar lebih rapi saja. Untuk proses login user, tetap menggunakan kolom 'email' dan 'password' bawaan Laravel.

Kemudian terdapat tambahan 6 kolom baru: 'tanggal_lahir', 'pekerjaan', 'kota', 'bio_profil', 'gambar_profil' dan 'background_profil'.

Kolom 'tanggal_lahir' dipakai untuk menampung tanggal lahir dari user. Kolom ini dibuat dengan tipe data date MySQL, yang artinya akan menyimpan data dengan format yyyy-mm-dd. Nantinya kita butuh proses konversi dari tanggal biasa menjadi format ini dan sebaliknya.

Kolom 'pekerjaan' dan 'kota' dipakai untuk menyimpan keterangan tentang pekerjaan dan asal kota user. Keduanya di set sebagai string yang akan dikonversi menjadi VARCHAR(255) di MySQL.

Kolom 'bio_profil' dipakai untuk menampung semacam biografi singkat user, atau bisa juga disebut dengan status yang biasa kita input di WA, Facebook atau Instagram. Agar bisa menampung teks yang cukup panjang, kolom ini saya set dengan tipe text.

Kolom 'pekerjaan', 'kota' dan 'bio_profil' ini mendapat tambahan atribut nullable(). Artinya ketiga kolom ini bersifat opsional dan boleh tidak diisi.

Kolom 'gambar_profil' dipakai untuk menampung nama dari gambar profil yang akan di upload oleh user. Meskipun user akan mengupload sebuah file gambar, yang kita simpan ke database cukup nama filenya saja. File gambar itu sendiri akan disimpan ke folder khusus di storage\app\public.

Kolom gambar ini juga memiliki nilai default 'default_profile.jpg'. Jika user memutuskan untuk tidak mengupload gambar profil, gambar default inilah yang akan dipakai.

Kolom 'background_profil' berfungsi untuk menampung pilihan gambar background dari tampilan profil. Berbeda dengan gambar profil yang bisa di upload oleh user, untuk gambar background ini nantinya user hanya bisa memilih 1 dari 9 pilihan gambar yang sudah tersedia.

Saya memilih tipe kolom tinyInteger karena inputan untuk background_profil hanya angka 1 sampai 9, yang mewakili nama gambar 1.jpg, 2.jpg, sampai dengan 9.jpg. Gambar ini nantinya akan siapkan di folder public. Jika gambar background tidak dipilih, secara default akan menggunakan 1.jpg.

Terakhir terdapat kolom 'email_verified_at', rememberToken() dan timestamps() bawaan tabel user.

Save file migration, lalu buat ulang tabel users dengan cara me-refresh migration:

```
php artisan migrate:fresh
```

Jika diperlukan, cek dari MySQL client apakah struktur tabel users sudah berubah atau belum:

```
MySQL Folder cmd - mysql -u root
MariaDB [(none)]> DESC laravel.users;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id   | bigint(20) unsigned | NO  | PRI | NULL    | auto_increment |
| email | varchar(255)        | NO  | UNI | NULL    |                |
| password | varchar(255)        | NO  |      | NULL    |                |
| nama | varchar(255)        | NO  |      | NULL    |                |
| tanggal_lahir | date        | NO  |      | NULL    |                |
| pekerjaan | varchar(255)        | YES |      | NULL    |                |
| kota | varchar(255)        | YES |      | NULL    |                |
| bio_profil | text         | YES |      | NULL    |                |
| gambar_profil | varchar(255)        | NO  |      | default_profile.jpg |
| background_profil | tinyint(4) | NO  |      | 1       |                |
| email_verified_at | timestamp | YES |      | NULL    |                |
| remember_token | varchar(100)        | YES |      | NULL    |                |
| created_at | timestamp | YES |      | NULL    |                |
| updated_at | timestamp | YES |      | NULL    |                |
+-----+-----+-----+-----+-----+-----+
14 rows in set (0.005 sec)
```

Gambar: Struktur dari tabel users

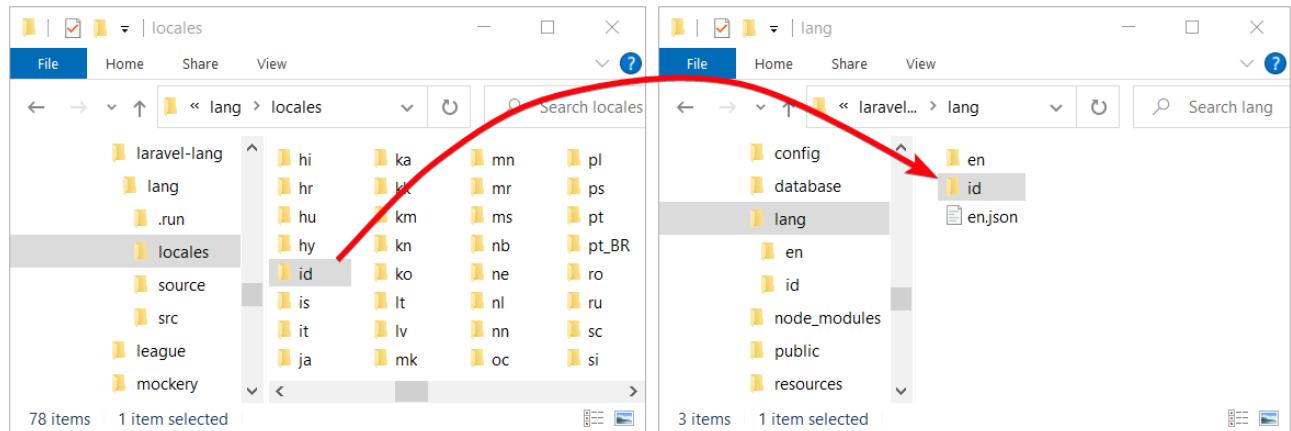
27.3. Instalasi Localization

Agar lebih user friendly, saya ingin pesan error form tampil dalam Bahasa Indonesia. Caranya juga simple, cukup tempatkan file localization ke dalam folder **lang**. Mengenai hal ini sudah pernah kita bahas dalam bab Localization.

Buka cmd, masuk ke folder `laravel01` lalu jalankan perintah berikut:

```
composer require laravel-lang/lang="^10.0"
```

Setelah selesai, copy isi folder `vendor\laravel-lang\lang\src\` ke dalam folder `lang\`.



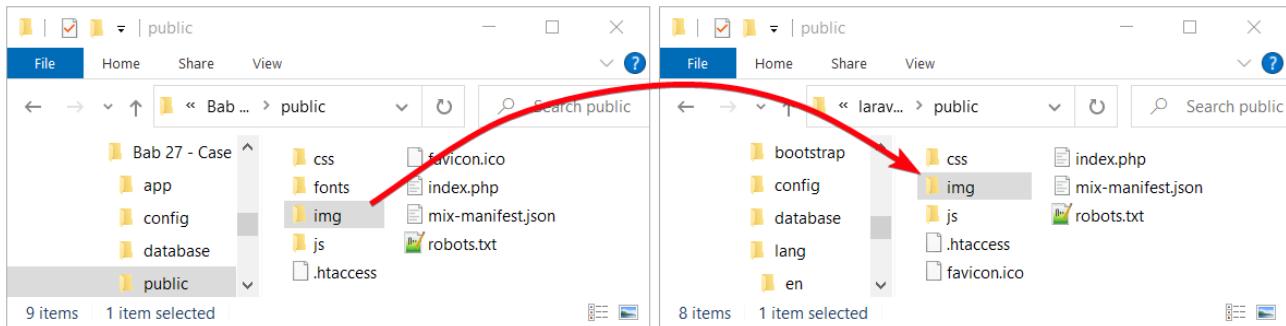
Gambar: Copy folder lang id ke lang

Setelah itu ubah pengaturan `locale` di file `config\app.php` dari '`locale' => 'en'`, menjadi '`locale' => 'id'` agar secara default aplikasi Laravel menggunakan bahasa indonesia.

27.4. Persiapan File Gambar

Untuk design tampilan, saya butuh beberapa file gambar yang akan dipakai pada background profil, background slider, dan logo. File ini sudah saya siapkan di file `belajar_laravel.zip`, tepatnya pada folder Bab 27 - Case Study ILKOOM Profile Manager\public\img.

Silahkan copy folder `img` ini ke dalam folder `public`\



Gambar: Copy folder img

27.5. Instalasi Font Awesome

Tamplate atau view dari ILKOOM Profile Manager butuh library CSS **Font Awesome**. Font awesome merupakan sebuah font berbentuk icon yang populer dipakai untuk membuat icon-icon kecil pada design web. Font awesome tersedia dalam versi gratis dan juga versi berbayar di fontawesome.com.

Tujuan penggunaan font awesome untuk project kita hanya sekedar mempercantik tampilan, tidak berhubungan langsung dengan kode Laravel.

File untuk font awesome ini bisa di akses dari **CDN**, yakni link ke sebuah file server di internet (seperti praktik penggunaan Bootstrap di bab View). Namun kali ini saya ingin mendownloadnya menggunakan perintah **npm**.

Pada saat kita menginstall **Laravel UI**, itu juga ikut menginstall npm. **Npm** sendiri adalah aplikasi pengelolaan library yang mirip seperti **composer**. Bedanya, npm lebih fokus ke library CSS dan JavaScript, sedangkan composer fokus ke library PHP.

Jika project kita butuh library JavaScript atau CSS tertentu, tinggal ketik perintah `install` menggunakan npm, dan file tersebut akan di download ke dalam folder `node_modules`. Setelah sampai di `node_modules`, tinggal "ditarik" menggunakan file `Sass` dan di compile ulang dari **Laravel Mix**.

File Bootstrap bawaan Laravel UI sebenarnya juga diakses dengan cara ini, hanya saja file tersebut sudah langsung terinstall sebagai bawaan Laravel UI.

Baik, untuk mulai proses instalasi font awesome silahkan buka cmd, masuk ke folder `laravel01` dan jalankan perintah berikut:

```
npm install @fortawesome/fontawesome-free
```

```
C:\xampp\htdocs\laravel01>npm install @fortawesome/fontawesome-free
> @fortawesome/fontawesome-free@5.15.4 postinstall C:\xampp\htdocs\laravel01\node_modules\@fortawesome\fontawesome-free
> node attribution.js

Font Awesome Free 5.15.4 by @fortawesome - https://fontawesome.com
License - https://fontawesome.com/license/free (Icons: CC BY 4.0, Fonts: SIL OFL 1.1, Code: MIT License)

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
```

Gambar: Proses instalasi font awesome menggunakan npm

Proses instalasi akan berjalan beberapa saat. Setelah selesai, di dalam folder `node_modules` akan tampil folder baru bernama `@fortawesome`. Dalam folder inilah file `font-awesome` tersimpan. File ini tidak perlu kita utak-atik, karena tinggal di akses dari file `Sass`.

Silahkan buka file `resources\sass\app.scss`, lalu tambah 4 baris kode berikut di bagian bawah file:

```
@import '~@fortawesome/fontawesome-free/scss/fontawesome';
@import '~@fortawesome/fontawesome-free/scss/regular';
@import '~@fortawesome/fontawesome-free/scss/solid';
@import '~@fortawesome/fontawesome-free/scss/brands';
```

Empat perintah inilah yang akan "mengambil" file font awesome dari `node_modules` ke dalam file `Sass`.

27.6. Membuat File Sass

Selain file untuk font awesome, saya juga perlu membuat 1 file `Sass` tambahan yang berisi kode CSS untuk design tampilan. Kembali, ini juga tidak berhubungan langsung dengan kode Laravel, tapi hanya untuk mempercantik tampilan akhir.

Masih di folder `resources\sass\`, silahkan buat file baru bernama `my-style.scss`, lalu ketik kode berikut (atau boleh di copas saja):

```
resources\sass\my-style.scss
```

```
1 /* NAVBAR */
2
3 #main-navbar .nav-link{
4   border-bottom: 5px solid #212529;
5   color: #e2e1e1;
6 }
7 #main-navbar .nav-link:hover, #main-navbar a.active {
8   border-bottom: 5px solid #17a2b8;
9   color: white;
```

```
10 }
11
12 .main-logo {
13   height: 100px;
14   margin-bottom: -70px; /* Supaya logo offset ke bawah */
15 }
16 .small-logo {
17   height: 60px; /* Untuk logo kecil di footer */
18 }
19
20 /* Untuk style dropdown */
21
22 .dropdown-menu{
23   padding:0;
24   border:0;
25   width: 100%;
26 }
27
28 .dropdown-item {
29   background-color: #212529;
30   color: #e2e1e1;
31   border-bottom: 5px solid #212529;
32 }
33 .dropdown-item:hover {
34   background-color: #212529;
35   color: white;
36   border-bottom: 5px solid #17a2b8;
37 }
38
39 /* FORM REGISTER - Halaman Register */
40
41 .pilihan-background-profil{
42   display: inline-block;
43   position: relative;
44   cursor: pointer;
45 }
46
47 .overlay { /* Untuk membuat nomor urut gambar background */
48   background-color: #ffffff;
49   position: absolute;
50   z-index: 100;
51   padding: 0 4px;
52   opacity: 0.8;
53   font-size: 1.2em;
54 }
55
56 .invalid-feedback { /* Agar semua pesan error form bisa terlihat */
57   display: block;
58 }
59
60 #display_gambar_profil {
61   cursor: pointer;
62 }
63
64 /* MAIN SLIDER - Halaman Home */
```

```
65
66 #main-navbar{
67   z-index: 100; /* Agar logo tidak tertutup slider */
68 }
69
70 #main-slide .carousel-item{
71   background-size: cover;
72   background-repeat: no-repeat;
73   width: 100%;
74   height: 90vh;
75   background-position: center;
76   background-attachment: fixed;
77 }
78
79 .bg-info{ /* Agar warna background bg-info sedikit lebih gelap */
80   background-color: #17a2b8 !important;
81 }
82
83 #main-slide h1 {
84   margin-top: 50vh; /* Posisikan teks di tengah slider */
85 }
86
87 #slide1{
88   background-image: url('/img/full-image13.jpg');
89 }
90
91 #slide2{
92   background-image: url('/img/full-image5.jpg');
93 }
94
95 #slide3{
96   background-image: url('/img/full-image14.jpg');
97 }
98
99 #slide4{
100   background-image: url('/img/full-image12.jpg');
101 }
102
103 /* Agar tombol btn-info sedikit gelap */
104
105 .btn-info {
106   color: white;
107   background-color: #17a2b8;
108   border-color: #17a2b8;
109 }
110
111 .btn-info:hover {
112   color: white;
113   background-color: #138496;
114   border-color: #117a8b;
115 }
116
117 /* MEMBER LIST - Halaman Home */
118
119 #member-list .card-body img {
```

Case Study: ILKOOM Profile Manager

```
120   width: 125px;
121   height: 125px;
122   margin-top: -80px;
123 }
124
125 .btn-action {           /* Untuk mengatur tombol edit dan delete */
126   position: absolute;
127   top: 10px;
128   left: 10px;
129 }
130
131 .btn-action button, .btn-action a {
132   opacity: 0.7;
133 }
134 .btn-action button:hover {
135   opacity: 1;
136 }
137
138 /* HEADER IMAGE - Untuk selain halaman Home */
139
140 .header-image {
141   background-repeat: no-repeat;
142   background-size: cover;
143   background-attachment: fixed;
144   height: 300px;
145 }
146
147 .header-overlay {
148   background-color: rgba(0,0,0,0.2);
149   width: 100%;
150   height: 300px;
151 }
152
153 .header-image h1{
154   margin-top:120px;
155 }
156
157 /* HEADER IMAGE - Gambar untuk setiap halaman */
158
159 #register-header{
160   background-image: url('/img/full-image8.jpg');
161   background-position-y: -200px;
162 }
163
164 #login-header{
165   background-image: url('/img/full-image10.jpg');
166   background-position-y: -100px;
167 }
168
169 #update-header{
170   background-image: url('/img/full-image7.jpg');
171   background-position-y: -200px;
172 }
173
174 /* FOOTER - */
```

```
175  
176 #main-footer a {  
177   text-decoration: none;  
178 }  
179  
180 #main-footer a:hover {  
181   text-decoration: underline;  
182 }  
183  
184 /*-- Nonaktifkan XXL Breakpoint (opsional) --*/  
185  
186 @media (min-width: 1200px) {  
187   .container {  
188     max-width: 1140px;  
189   }  
190 }
```

Kode-kode ini tidak akan saya bahas karena lebih ke design tampilan. Isinya juga kode CSS biasa, tidak ada kode Sass meskipun file ini berekstensi `.scss`. Beberapa property CSS saya pakai untuk membuat gambar background bagian header dan juga slider.

Penjelasan dari kode-kode di atas ada di buku **Bootstrap Uncover**. Design tampilan yang saya pakai merupakan studi kasus bab terakhir dari buku tersebut (plus sedikit modifikasi tambahan).

Setelah itu, import file `my-style.scss` dari dalam daftar file `app.scss`:

```
@import 'my-style';
```

Dengan demikian, berikut kode lengkap dari `resources\sass\app.scss`:

```
resources/sass/app.scss
```

```
1 // Fonts  
2 // @import url('https://fonts.googleapis.com/css?family=Nunito');  
3  
4 // Variables  
5 // @import 'variables';  
6  
7 // Bootstrap  
8 @import '~bootstrap/scss/bootstrap';  
9  
10 // Font Awesome  
11 @import '~@fortawesome/fontawesome-free/scss/fontawesome';  
12 @import '~@fortawesome/fontawesome-free/scss/regular';  
13 @import '~@fortawesome/fontawesome-free/scss/solid';  
14 @import '~@fortawesome/fontawesome-free/scss/brands';  
15  
16 // Custom Style  
17 @import 'my-style';
```

Perintah di baris 2 dan 5 merupakan kode Sass bawaan Laravel yang tidak ingin saya pakai

(sehingga dijadikan sebagai komentar). Kode di baris 1 – 5 ini bisa saja dihapus karena memang tidak diperlukan, tapi sengaja saya biarkan agar tidak bingung.

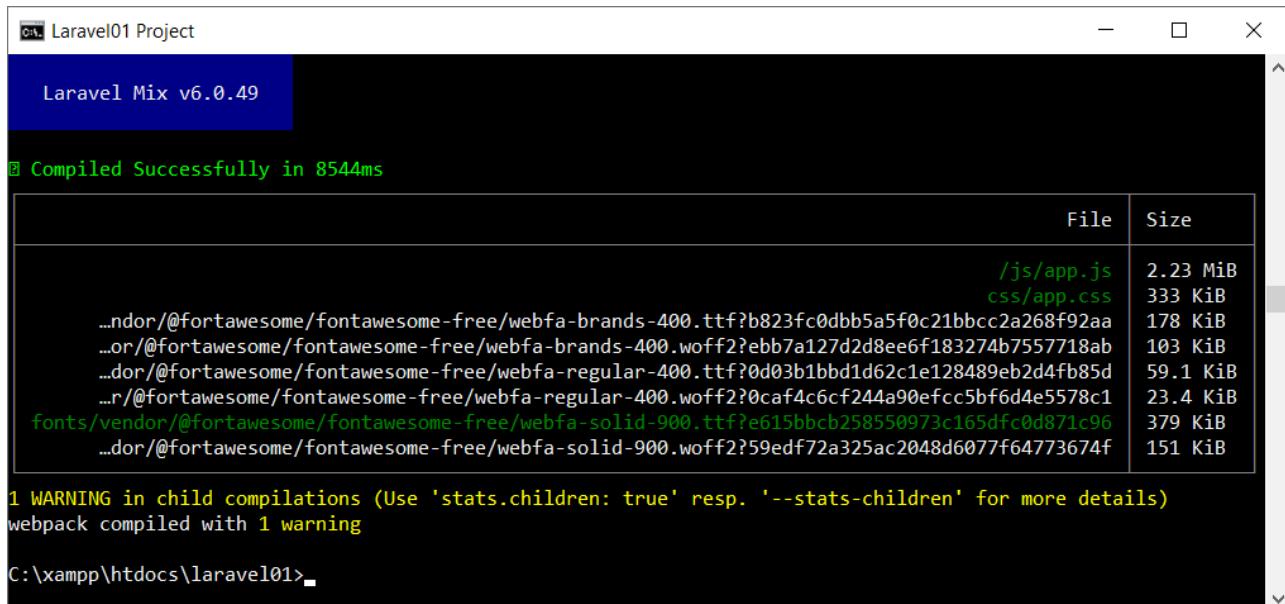
Perintah di baris 2 akan mendownload font **Nunito** dari Google Font, sedangkan perintah di baris 4 akan meng-import file `_variables.scss` yang berisi beberapa style bawaan Laravel. Style ini tidak saya perlukan karena bisa mempengaruhi design kita.

Di baris 8 merupakan kode untuk import file bootstrap, lalu di baris 11 – 14 untuk import file font awesome, serta di baris 17 untuk import file `my-style.scss` yang baru saja kita buat.

Jika anda masih ingat, file `app.scss` ini adalah file "master" untuk proses compile assets CSS. Setelah di compile dengan Laravel Mix, nantinya akan disatukan menjadi file `app.css` di folder `public\css\`.

Maka langkah terakhir adalah compile file `app.scss` dengan perintah Laravel Mix berikut:

```
npx mix
```



File	Size
/js/app.js	2.23 MiB
css/app.css	333 KiB
...ndor/@fontawesome/fontawesome-free/webfa-brands-400.ttf?b823fc0dbb5a5f0c21bcc2a268f92aa	178 KiB
...or/@fontawesome/fontawesome-free/webfa-brands-400.woff2?ebb7a127d2d8ee6f183274b7557718ab	103 KiB
...dor/@fontawesome/fontawesome-free/webfa-regular-400.ttf?0d03b1bbd1d62c1e128489eb2d4fb85d	59.1 KiB
...r/@fontawesome/fontawesome-free/webfa-regular-400.woff2?0caf4c6cf244a90efcc5bf6d4e5578c1	23.4 KiB
fonts/vendor/@fontawesome/fontawesome-free/webfa-solid-900.ttf?e615bbc258550973c165dfc0d871c96	379 KiB
...dor/@fontawesome/fontawesome-free/webfa-solid-900.woff2?59edf72a325ac2048d6077f64773674f	151 KiB

```
1 WARNING in child compilations (Use 'stats.children: true' resp. '--stats-children' for more details)
webpack compiled with 1 warning

C:\xampp\htdocs\laravel01>
```

Gambar: Proses compile file assets

Setelah proses compile, silahkan cek file `public\css\app.css`, seharusnya file ini berisi kode CSS Bootstrap, Font Awesome dan juga kode CSS yang kita tulis ke dalam file `my-style.scss`.

27.7. Modifikasi View layout\app.blade.php

Kita akan memodifikasi banyak file blade bawaan authentication, yang dimulai dari file `layout\app.blade.php`. Ini merupakan *parent file* dari semua file blade lain. Dalam file tersebut sudah ada kode untuk bagian header dan menu navigasi. Selain itu saya juga akan tambah kode untuk bagian footer.

Bagian header, navbar dan footer akan sama dari satu halaman dengan halaman lain, sehingga

pas dijadikan parent file atau file induk. Nantinya file blade lain tinggal meng-extends file layout\app.blade.php untuk mengisi bagian konten.

Berikut kode program lengkap dari file app.blade.php:

resources/views/layouts/app.blade.php

```

1  <!doctype html>
2  <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6
7      <!-- CSRF Token -->
8      <meta name="csrf-token" content="{{ csrf_token() }}>
9
10     <title>ILKOOM - Community Blog</title>
11     <link rel="icon" href="{{ asset('img/favicon.png') }}" type="image/png">
12
13     <!-- Scripts -->
14     <script src="{{ asset('js/app.js') }}" defer></script>
15
16     <!-- Styles -->
17     <link href="{{ asset('css/app.css') }}" rel="stylesheet">
18 </head>
19 <body>
20     <!-- NAVBAR -->
21     <nav id="main-navbar" class="navbar navbar-expand-md navbar-dark bg-dark py-0">
22         <div class="container">
23             <a class="navbar-brand" href="{{ route('home') }}>
24                 <span class="d-none" >ILKOOM</span>
25                 
27                 
29             </a>
30             <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
31                 data-bs-target="#navbarNav">
32                 <span class="navbar-toggler-icon"></span>
33             </button>
34
35             <div class="collapse navbar-collapse" id="navbarNav">
36                 <!-- Left Side Of Navbar -->
37                 <ul class="navbar-nav">
38                     <li class="nav-item">
39                         <a class="nav-link p-4 active" href="{{ route('home') }}>Home</a>
40                     </li>
41                     <li class="nav-item">
42                         <a class="nav-link p-4" href="#">Blog</a>
43                     </li>
44                     <li class="nav-item">
45                         <a class="nav-link p-4" href="#">Article</a>
46                     </li>
47             </div>

```

```

48         <li class="nav-item">
49             <a class="nav-link p-4" href="#">Gallery</a>
50         </li>
51     </ul>
52
53     <!-- Right Side Of Navbar -->
54     <ul class="navbar-nav ms-auto">
55         <!-- Authentication Links -->
56         @guest
57             @if (Route::has('login'))
58                 <li class="nav-item">
59                     <a class="nav-link p-4" href="{{ route('login') }}>Login</a>
60                 </li>
61             @endif
62
63             @if (Route::has('register'))
64                 <li class="nav-item">
65                     <a class="nav-link p-4" href="{{ route('register') }}>
66                         Register</a>
67                 </li>
68             @endif
69         @else
70             <li class="nav-item dropdown">
71                 <a id="navbarDropdown" class="nav-link dropdown-toggle p-4"
72                   href="#" data-bs-toggle="dropdown">
73                     {{ Auth::user()->nama }}
74                 </a>
75
76                 <div class="dropdown-menu dropdown-menu-end">
77                     <a class="dropdown-item p-4 py-md-2"
78                       href="{{ route('logout') }}"
79                         onclick="event.preventDefault();
80                               document.getElementById('logout-form').submit();">
81                         {{ __('Logout') }}
82                     </a>
83
84                     <form id="logout-form" action="{{ route('logout') }}"
85                       method="POST" class="d-none">
86                         @csrf
87                     </form>
88                 </div>
89             </li>
90         @endguest
91     </ul>
92     </div>
93 </div>
94 </nav>
95
96     @yield('content')
97
98     <!-- FOOTER -->
99     <footer id="main-footer" class="text-white bg-dark py-4">
100        <div class="container">
101            <div class="row">
102                <div class="col-md-3 text-center text-md-start">
```

```

103     <a href="index.html">
104         
105     </a>
106     <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
107         Aperiam cumque, esse modi maxime veniam nulla delectus dolorem
108         </p>
109     </div>
110
111     <div class="col-md-3 text-center">
112         <h5>Community</h5>
113         <ul class="list-unstyled">
114             <li><a href="#" class="text-white">Activity</a></li>
115             <li><a href="#" class="text-white">Members</a></li>
116             <li><a href="#" class="text-white">Groups</a></li>
117             <li><a href="#" class="text-white">Forums</a></li>
118         </ul>
119     </div>
120
121     <div class="col-md-3 text-center">
122         <h5>Our Services</h5>
123         <ul class="list-unstyled">
124             <li><a href="#" class="text-white">Our mission</a></li>
125             <li><a href="#" class="text-white">Help/Contact Us</a></li>
126             <li><a href="#" class="text-white">Privacy Policy</a></li>
127             <li><a href="#" class="text-white">Cookie Policy</a></li>
128             <li><a href="#" class="text-white">Terms & Conditions</a></li>
129         </ul>
130     </div>
131
132     <div class="col-md-3 text-center text-md-start">
133         <h5>Hubungi Kami</h5>
134         <div class="text-nowrap"><i class="fas fa-envelope fa-fw me-3"></i>
135             andre@ilkoom.com</div>
136         <div class="text-nowrap"><i class="fas fa-phone fa-fw me-3"></i>
137             (021) 123456</div>
138         <div class="text-nowrap"><i class="fas fa-globe fa-fw me-3"></i>
139             www.ilkoom.com</div>
140     </div>
141 </div>
142
143     <div class="row mt-3 mt-md-0">
144         <div class="col-md-3 me-md-auto text-center text-md-start">
145             <small>&copy; ILKOOM {{ date("Y") }}</small>
146         </div>
147         <div class="col-md-3 text-center text-md-start">
148             <div>
149                 <a href="#" class="text-white text-decoration-none me-2">
150                     <i class="fab fa-facebook fa-lg"></i>
151                 </a>
152                 <a href="#" class="text-white text-decoration-none me-2">
153                     <i class="fab fa-twitter fa-lg"></i>
154                 </a>
155                 <a href="#" class="text-white text-decoration-none me-2">
156                     <i class="fab fa-instagram fa-lg"></i>
157                 </a>

```

```
158      <a href="#" class="text-white text-decoration-none me-2">
159        <i class="fab fa-whatsapp fa-lg"></i>
160      </a>
161      <a href="#" class="text-white text-decoration-none me-2">
162        <i class="fab fa-github fa-lg"></i>
163      </a>
164    </div>
165  </div>
166</div>
167
168</div>
169</footer>
170</body>
171</html>
```

Kode programnya memang cukup panjang, tapi untuk aplikasi "real world", seperti inilah kode yang dibutuhkan. Saya tetap mempertahankan beberapa kode bawaan authentication Laravel, tetapi juga menambah sebagian besar kode baru.

Baris 1 – 18 merupakan bagian `<head>` HTML. Sedikit perubahan ada di tag `<title>` yang saya ganti menjadi **ILKOOM - Community Blog**, serta tambahan untuk gambar favicon di baris 11.

Baris 21 – 94 berisi kode untuk navbar. Di sini saya merombak cukup banyak kode program, termasuk menampilkan logo ILKOOM di baris 24 – 30, yakni file '`img/ilkoom_logo.png`'.

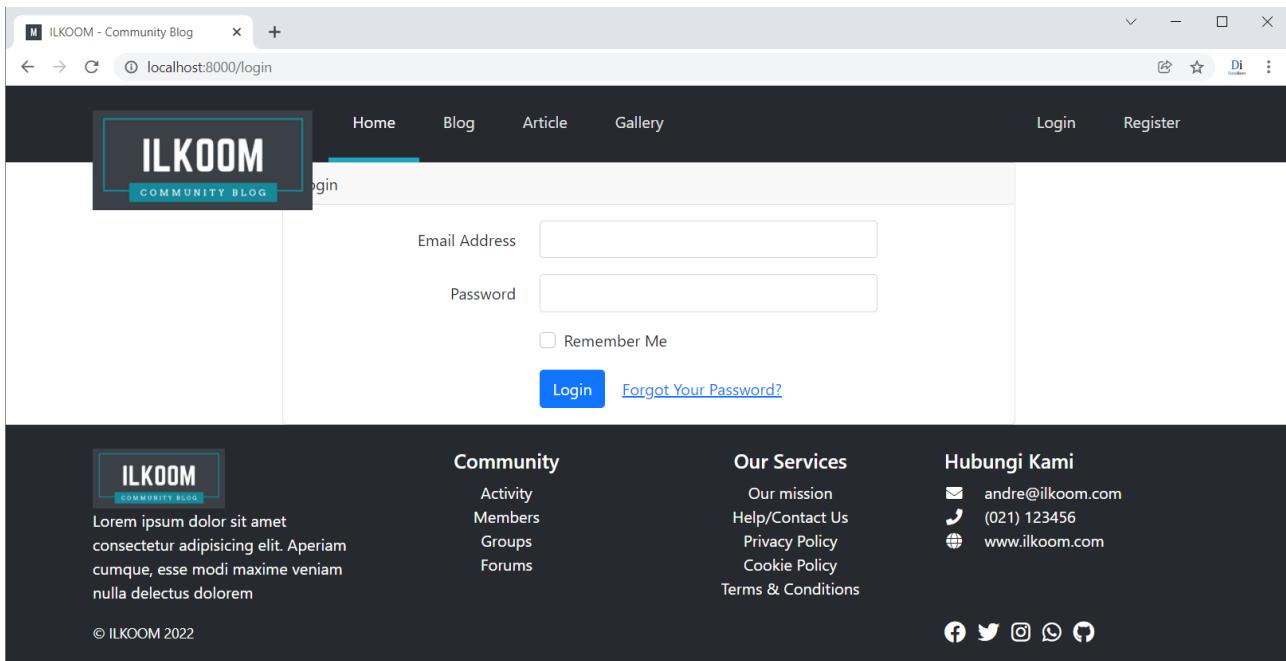
Kemudian diikuti dengan 4 menu link di baris 38 – 51. Selain menu Home, ketiga menu lain hanya *dummy* link menuju `href="#"` karena fokus utama kita kali ini hanya di bagian profil user saja.

Semua menu ILKOOM ini berada di sisi kiri halaman. Di bagian kanan, terdapat menu untuk proses Login dan Register. Untuk tampilan ini saya mengikuti kode bawaan authentication Laravel. Terlihat perintah kondisi `@guest` yang dipakai untuk membatasi tampilan bagi user yang belum login.

Di baris 96 terdapat perintah `@yield('content')`. Inilah tempat konten utama halaman yang nantinya diisi oleh file blade yang meng-extends file layout ini.

Terakhir di baris 99 – 169 adalah kode untuk membuat tampilan footer. Struktur yang dipakai terdiri dari 4 buah grid class `.col-md-3` bawaan Bootstrap. Terdapat kode untuk menampilkan gambar logo di baris 104, serta beberapa icon font awesome di baris 149 – 163. Penjelasan lebih jauh tentang design footer ini ada di buku **Bootstrap Uncover**.

Save file `layout/app.blade.css` ini lalu akses halaman localhost:8000/login atau halaman localhost:8000/register, akan terlihat struktur menu navigasi serta footer dari project kita:



Gambar: Tampilan sementara halaman login

Sip, tampilannya sudah cukup cantik.

Jika anda iseng membuka halaman root di localhost:8000, tidak akan tampak perubahan apapun karena view `welcome.blade.php` tidak men-extends file layout ini. Nantinya kita juga akan ganti halaman tersebut dengan view baru.

27.8. Modifikasi View auth\register.blade.php

Selanjutnya view yang akan di modifikasi adalah file `auth\register.blade.php`. View ini berisi tampilan halaman form register yang bawaan Laravel terdiri dari inputan `name`, `email`, `password` dan `confirm password`.

Ketika mengubah struktur tabel `users`, saya menambah cukup banyak kolom baru. Semua kolom ini harus diinput dari form register sehingga kita perlu menambah beberapa tag `<input>`. Atau lebih tepatnya, apa yang akan saya lakukan adalah merombak ulang semua kode form register agar sesuai dengan rancangan design ILKOOM Profile Manager.

Berikut kode untuk view `auth\register.blade.php`:

`resources/views/auth/register.blade.php`

```
1 @extends('layouts.app')
2
3 @section('content')
4
5 <!-- HEADER IMAGE -->
6 <header id="register-header" class="header-image text-white d-none d-md-block">
7   <div class="header-overlay">
```

```
8   <div class="container">
9     <div class="row">
10    <div class="col">
11      <h1 class="display-1">Join our Community</h1>
12      <p>Bergabunglah dengan salah satu blog komunitas terbaik
13        di Indonesia</p>
14    </div>
15  </div>
16 </div>
17 </div>
18 </header>
19
20 <div class="container my-5">
21   <div class="row ">
22     <div class="col-lg-8">
23       <h1>Form Pendaftaran</h1>
24       <hr>
25       <form method="POST" action="{{ route('register') }}"
26         enctype="multipart/form-data">
27         @include('layouts.form',['tombol' => 'Daftar'])
28       </form>
29     </div>
30   </div>
31 </div>
32
33 @endsection
```

View `register.blade.php` ini meng-extends file `layouts\app.blade.php`, sehingga tampilan halaman sudah memiliki bagian head, menu navbar serta tampilan footer. Proses extends ini dilakukan di baris 1.

Selanjutnya kode program di baris 3 – 33 adalah tempat untuk konten halaman register. Diawali dengan gambar header di baris 6 – 18 sekedar mempercantik tampilan, lalu pembuatan form di baris 20 – 31.

Tapi kenapa kode programnya sangat singkat? selain itu dimana tag `<input>` berada?

Saya memutuskan untuk memindahkan isi form menjadi file terpisah, yakni ke dalam file `layouts\form.blade.php` (yang akan kita bahas sesaat lagi). Tujuannya agar isi form ini bisa digunakan ulang oleh form update user.

Setelah beberapa kali membuat kode program untuk CRUD, kita bisa ambil kesimpulan bahwa struktur form untuk proses *create* dan *update* sebenarnya sangat mirip. Jadi daripada membuat 2 buah form terpisah, lebih baik kode yang sama digunakan ulang.

Alur pembuatan dengan teknik ini memang jadi lebih rumit, tapi dalam jangka panjang akan memudahkan dalam pengelolaan kode program.

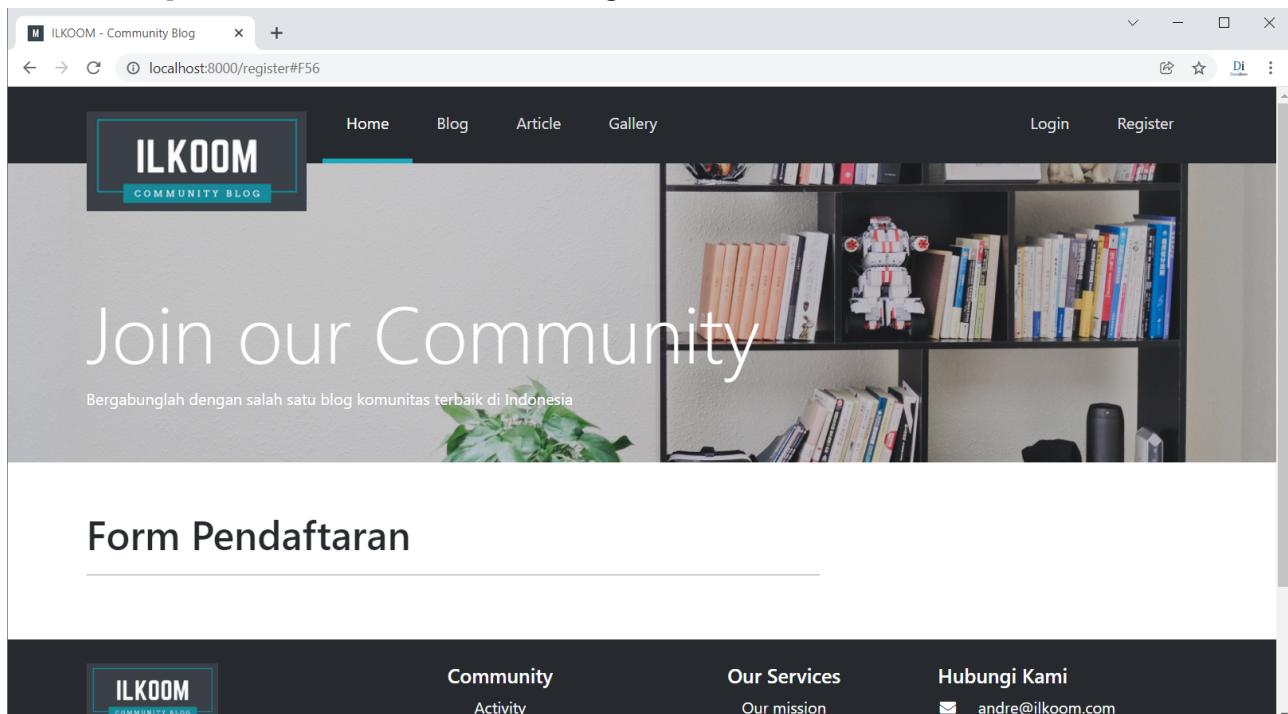
Dalam view `register.blade.php`, struktur form yang diperlukan hanya tag `<form>` beserta atribut `method`, `action` dan `enctype`.

Isi atribut `method` berupa `route('register')`, yang merupakan *named route* bawaan

authentication Laravel. Dengan melihat hasil dari `php artisan route:list`, route ini menuju ke method `showRegistrationForm()` di `RegisterController`. Di dalam controller inilah nantinya kita akan menulis syarat validasi dan melakukan proses input ke database.

Struktur form sendiri di panggil dengan perintah `@include('', ['tombol' => 'Daftar'])`. Artinya, selain proses incude file `layouts.form`, saya juga mengirim variabel 'tombol' yang berisi string 'Daftar'. Nantinya variabel `tombol` ini akan dipakai sebagai nilai untuk tombol submit form.

Dengan menghapus sementara baris `@include('', ['tombol' => 'Daftar'])` agar tidak error, berikut tampilan sementara dari halaman register:



Kode untuk inputan form nantinya akan tampil di bagian tengah halaman, yakni tepat di bawah judul "**Form Pendaftaran**".

Sebelum lanjut, silahkan tambahkan kembali kode `@include('', ['tombol' => 'Daftar'])` jika anda juga ikut menghapusnya untuk melihat tampilan ini.

27.9. Membuat View layout\form.blade.php

Struktur utama form register (dan juga nantinya form update), saya tempatkan ke dalam file `layout\form.blade.css`. File ini sebelumnya memang tidak ada, maka silahkan dibuat terlebih dahulu. Berikut kode program yang diperlukan:

resources/views/layouts/form.blade.php

```

1  @csrf
2  <div class="mb-3 row">
3      <label for="email" class="col-md-3 col-form-label text-md-end">
4          Email * </label>
5      <div class="col-md-6">
6          <input id="email" type="email"
7              class="form-control @error('email') is-invalid @enderror"
8              name="email" value="{{ old('email') ?? $user->email ?? '' }}"
9              autocomplete="email" autofocus>
10         @error('email')
11             <span class="invalid-feedback" role="alert">
12                 <strong>{{ $message }}</strong>
13             </span>
14         @enderror
15     </div>
16 </div>
17
18 {{-- Jika form ini untuk halaman register, maka tampilkan inputan password --}}
19@if ($tombol == 'Daftar')
20
21 <div class="mb-3 row">
22     <label for="password" class="col-md-3 col-form-label text-md-end">
23         Password * </label>
24     <div class="col-md-6">
25         <input id="password" type="password"
26             class="form-control @error('password') is-invalid @enderror"
27             name="password" autocomplete="new-password">
28         @error('password')
29             <span class="invalid-feedback" role="alert">
30                 <strong>{{ $message }}</strong>
31             </span>
32         @enderror
33     </div>
34 </div>
35
36 <div class="mb-3 row">
37     <label for="password-confirm" class="col-md-3 col-form-label text-md-end">
38         Ulangi Password * </label>
39     <div class="col-md-6">
40         <input id="password-confirm" type="password" class="form-control"
41             name="password_confirmation" autocomplete="new-password">
42     </div>
43 </div>
44
45 @endif
46
47 <div class="mb-3 row">
48     <label for="nama" class="col-md-3 col-form-label text-md-end">
49         Nama * </label>
50     <div class="col-md-6">
51         <input id="nama" type="text" autocomplete="nama"
52             class="form-control @error('nama') is-invalid @enderror"
53             name="nama" value="{{ old('nama') ?? $user->nama ?? '' }}>
```

```

54     @error('nama')
55         <span class="invalid-feedback" role="alert">
56             <strong>{{ $message }}</strong>
57         </span>
58     @enderror
59     </div>
60 </div>
61
62 <div class="mb-3 row">
63     <label for="tanggal_lahir" class="col-md-3 col-form-label text-md-end">
64         Tanggal Lahir *</label>
65     <div class="col-md-6">
66         <div class="row g-2">
67             <div class="col-3">
68                 <input type="number" name="tgl" id="tgl"
69                     class="form-control col-md-3 d-inline"
70                     @error('tanggal_lahir') is-invalid @enderror
71                     placeholder="dd" value="{{ old('tgl') ?? $user->tgl ?? '' }}">
72             </div>
73             <div class="col-5">
74                 @php
75                     // Siapkan nama-nama bulan dalam bahasa Indonesia
76                     $namaBulan = ["Januari", "Februari", "Maret", "April",
77                         "Mei", "Juni", "Juli", "Agustus", "September",
78                         "Oktober", "November", "Desember"];
79                 @endphp
80                 <select class="form-select col-md-4 d-inline"
81                     @error('tanggal_lahir') is-invalid @enderror
82                     style="vertical-align: baseline;" name="bln" id="bln">
83                     @for ($i = 0; $i < 12; $i++)
84                         @if ($i+1 == (old('bln') ?? $user->bln ?? ''))
85                             <option value="{{ $i+1 }}" selected>{{ $namaBulan[$i] }}</option>
86                         @else
87                             <option value="{{ $i+1 }}">{{ $namaBulan[$i] }}</option>
88                         @endif
89                     @endfor
90                 </select>
91             </div>
92             <div class="col-4">
93                 <input type="number" id="thn" class="form-control col-md-3 d-inline"
94                     @error('tanggal_lahir') is-invalid @enderror name="thn"
95                     placeholder="yyyy" value="{{ old('thn') ?? $user->thn ?? '' }}>
96             </div>
97             @error('tanggal_lahir')
98                 <span class="invalid-feedback" role="alert">
99                     <strong>{{ $message }}</strong>
100                 </span>
101             @enderror
102             </div>
103         </div>
104     </div>
105
106 <div class="mb-3 row">
107     <label for="pekerjaan" class="col-md-3 col-form-label text-md-end">
108         Pekerjaan </label>

```

```

109 <div class="col-md-6">
110   <input id="pekerjaan" type="text"
111   class="form-control @error('pekerjaan') is-invalid @enderror"
112   name="pekerjaan" value="{{ old('pekerjaan') ?? $user->pekerjaan ?? '' }}">
113   @error('pekerjaan')
114     <span class="invalid-feedback" role="alert">
115       <strong>{{ $message }}</strong>
116     </span>
117   @enderror
118 </div>
119 </div>
120
121 <div class="mb-3 row">
122   <label for="kota" class="col-md-3 col-form-label text-md-end">Kota </label>
123   <div class="col-md-6">
124     <input id="kota" type="text"
125     class="form-control @error('kota') is-invalid @enderror"
126     name="kota" value="{{ old('kota') ?? $user->kota ?? '' }}">
127     @error('kota')
128       <span class="invalid-feedback" role="alert">
129         <strong>{{ $message }}</strong>
130       </span>
131     @enderror
132   </div>
133 </div>
134
135 <div class="mb-3 row">
136   <label for="bio_profil" class="col-md-3 col-form-label text-md-end">
137   Bio Profil</label>
138   <div class="col-md-6">
139     <textarea class="form-control" id="bio_profil" name="bio_profil"
140     placeholder = "Bio singkat anda...">&lt;
141     old('bio_profil') ?? $user->bio_profil ?? ''
142   &lt;/textarea>
143   @error('bio_profil')
144     <span class="invalid-feedback" role="alert">
145       <strong>{{ $message }}</strong>
146     </span>
147   @enderror
148 </div>
149 </div>
150
151 <div class="mb-3 row">
152   <label for="gambar_profil" class="col-md-3 col-form-label text-md-end">
153   Gambar Profil</label>
154   <div class="col-md-6">
155     gambar_profil) }}"
160     @endif
161     >
162     <input type="file" id="gambar_profil" name="gambar_profil" accept="image/*"
163     class="form-control @error('gambar_profil') is-invalid @enderror">
```

```

164     @error('gambar_profil')
165         <span class="invalid-feedback" role="alert">
166             <strong>{{ $message }}</strong>
167         </span>
168     @enderror
169     </div>
170 </div>
171
172 <div class="mb-3 row">
173     <label for="background_profil" class="col-md-3 col-form-label text-md-end">
174         Background Profil</label>
175     <div class="col-md-6">
176         <select name="background_profil" class="form-select col-md-5"
177             @error('background_profil') is-invalid @enderror" id="background_profil" >
178             @for ($i = 1; $i <= 12; $i++)
179                 @if($i == (old('background_profil') ?? $user->background_profil ?? ''))
180                     <option value="{{ $i }}" selected >{{ 'Gambar '.$i }}</option>
181                 @else
182                     <option value="{{ $i }}>{{ 'Gambar '.$i }}</option>
183                 @endif
184             @endfor
185         </select>
186     @error('background_profil')
187         <span class="invalid-feedback" role="alert">
188             <strong>{{ $message }}</strong>
189         </span>
190     @enderror
191     <div class="my-2 row row-cols-3 g-1">
192         @for ($i = 1; $i <= 12; $i++)
193             <div class="pilihan-background-profil">
194                 <div class='overlay'>{{ $i }}</div>
195                 
196             </div>
197         @endfor
198     </div>
199 </div>
200 </div>
201 </div>
202
203 <div class="mb-3 row mb-0">
204     <div class="col-md-6 offset-md-3">
205         <button type="submit" class="btn btn-primary px-4">{{$tombol}}</button>
206     </div>
207 </div>
```

Kodenya lumayan panjang karena terdapat banyak inputan form. Ditambah lagi setiap inputan perlu menampilkan pesan error dan proses re-populate nilai. Form ini juga akan dipakai untuk proses update, sehingga butuh kode untuk menampilkan nilai form dari database.

Karena keterbatasan lebar halaman, beberapa kode program yang seharusnya bisa ditulis dalam 1 baris panjang terpaksa saya pecah menjadi beberapa baris, terutama untuk tag `<input>` yang butuh banyak atribut.

Sebelum mulai membahasnya, berikut tampilan dari semua kode di atas:

Form Pendaftaran

Email *

Password *

Ulangi Password *

Nama *

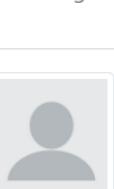
Tanggal Lahir *

 dd Januari yyyy

Pekerjaan

Kota

Bio Profil



Gambar Profil

Choose File

No file chosen

Background Profil

Gambar 1

▼



Daftar

Gambar: Tampilan form daftar

Di baris 1 terdapat perintah @csrf untuk pembuatan token form. Tag <form> sendiri tidak ada dalam file ini karena sudah ada di parent view.

Inputan form pertama dipakai untuk alamat **email** di baris 2 – 16. Tidak ada kode yang baru di sini, hanya berisi perintah yang sudah kita bahas di bab tentang form processing, termasuk menampilkan pesan error, re-populate nilai, pemberian nilai awal (untuk proses update), serta penggunaan class `.is-invalid` bawaan Bootstrap ketika terjadi error.

Untuk proses update, nilai awal didapat dari variabel `$user->email` di baris 8. Nantinya, variabel `$user` akan dikirim dari controller.

Di baris 19 terdapat blok kondisi `@if($tombol == 'Daftar')`. Kondisi ini akan terpenuhi jika pada saat proses include di parent view, variabel `$tombol` dikirim dengan nilai 'Daftar'. Ini di perlukan karena untuk form update, nilai variabel `$tombol` akan berisi string 'Update'.

Block `@if` dimulai dari baris 21 sampai 45 yang berisi inputan password dan ulangi password. Logikanya adalah, jika view ini dipakai untuk menampilkan form daftar, maka kita butuh inputan password. Namun jika view di pakai untuk menampilkan form update, maka inputan password ini tidak diperlukan (karena toh user sudah login).

Kode untuk inputan password dan ulangi password juga sama seperti inputan form biasa dengan atribut `name="password"` dan `name="password_confirmation"`. Atribut `name` inilah yang akan kita pakai pada saat proses validasi.

Lanjut, di baris 47-60 terdapat inputan `name="nama"` yang kodennya mirip seperti inputan biasa.

Yang sedikit kompleks ada di inputan tanggal lahir pada baris 62 – 104. Di sini saya memutuskan untuk memakai 2 buah text box senahai inputan tanggal dan tahun, serta 1 tag `<select>` untuk inputan nama bulan:

The form consists of three main input fields. The first field is labeled 'Nama *' with a text input box. The second field is labeled 'Tanggal Lahir *' and includes three components: a dropdown menu for 'dd' (day), a dropdown menu for 'Januari' (month), and a text input box for 'yyyy' (year). The third field is labeled 'Pekerjaan' with a text input box.

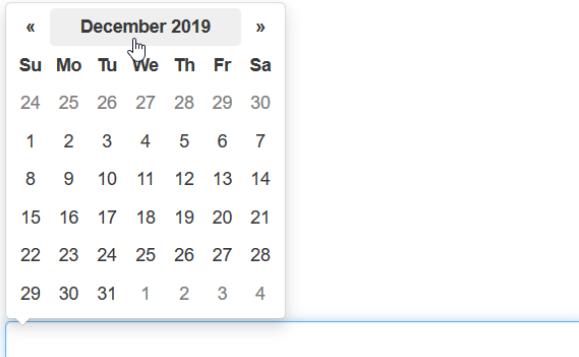
Gambar: Tampilan form tanggal lahir

Inputan seperti ini terinspirasi dari form yang dipakai oleh gmail (pada saat pendaftaran email):

The screenshot shows the Gmail sign-up form's birth date input section. It includes a field for 'Alamat email pemulihan (opsional)' (optional recovery email address) with a note below stating 'Kami akan menggunakan ini untuk menjaga keamanan akun Anda' (We will use this to protect your account). Below this is a 'Tanggal lahir Anda' (Your birthday) section with 'Tanggal' (Day), 'Bulan' (Month) with a dropdown menu showing 'Januari' and 'Februari' (February is highlighted), and 'Tahun' (Year). To the right, there is a decorative graphic of two envelopes and a note: 'Info pribadi Anda bersifat rahasia & aman' (Your personal information is confidential and safe).

Gambar: Inputan tanggal lahir pada saat pendaftaran gmail

Inputan tanggal memang berbeda-beda tergantung prinsip user experience dari web designer yang merancangnya. Ada yang menggunakan 3 buah tag `<select>`, dan ada pula yang menggunakan *date picker* dengan bantuan JavaScript.



Gambar: Contoh inputan date picker

Meskipun kesannya lebih modern, sebaiknya *date picker* tidak dipakai untuk inputan tanggal lahir. Saya pribadi sering merasa repot jika harus meng-klik tombol mundur berkali-kali untuk mencari tahun lahir yang dibuat dengan *date picker*. Sering kali date picker ini posisinya ada di tanggal hari ini, dan user harus mencari tanggal yang sangat jauh, misalnya di 8 Maret 1980.

Proses merancang form (dan juga tampilan web) agar lebih *user friendly* sudah menjadi cabang ilmu sendiri yang dikenal sebagai **UX** (User eXperience) dan sering digabung dengan **UI** (User Interface).

Profesi sebagai **UI/UX Designer** juga banyak dibutuhkan, terutama untuk perusahaan software besar yang peduli dengan tampilan aplikasi.

Kembali ke kode program, inputan tanggal dan tahun menjadi lebih sederhana karena dibuat dari tag `<input type="number">` yang tidak berbeda dengan `<input type="text">`. Atribut `type="number"` bawaan HTML5 membatasi inputan form hanya bisa diisi dengan angka saja.

Khusus untuk inputan bulan butuh kode tersendiri karena saya ingin pilihan bulan ini tampil dalam bahasa Indonesia. Tekniknya adalah, buat array `$namaBulan` di baris 76, kemudian pakai perulangan `@for` di baris 83 – 89 untuk meng-generate tag `<option>` berdasarkan nilai array `$namaBulan` ini.

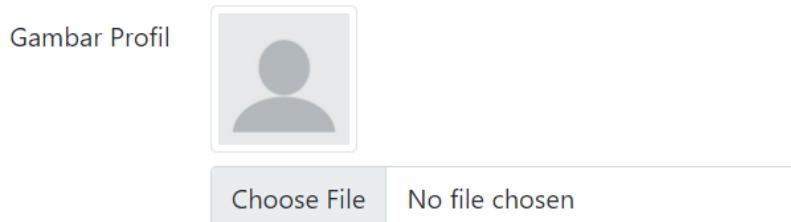
Terdapat juga kondisi `@if` untuk proses re-populate inputan form. Di dalam kondisi perbandingan, saya memakai `$i+1 == (old('b1n'))` karena terdapat perbedaan index array yang dimulai dari 0 dengan urutan nomor bulan yang dimulai dari angka 1 untuk bulan Januari.

Dua inputan selanjutnya juga tidak ada yang baru, yakni untuk pekerjaan di baris 106 – 119 dan nama kota di baris 121 – 133. Keduanya sama-sama memakai tag `<input type="text">`. Inputan `<textarea>` bio profil di baris 135 – 149 juga sudah pernah kita bahas di bab Form Processing.

Inputan selanjutnya dipakai untuk upload gambar profil. Di sini saya menggunakan tag `<input`

yang sudah kita bahas di bab file upload. Selain itu terdapat sedikit tambahan pemeriksaan kondisi @if antara baris 156 – 160. Kode ini dipakai untuk menampilkan gambar awal dari preview profil.

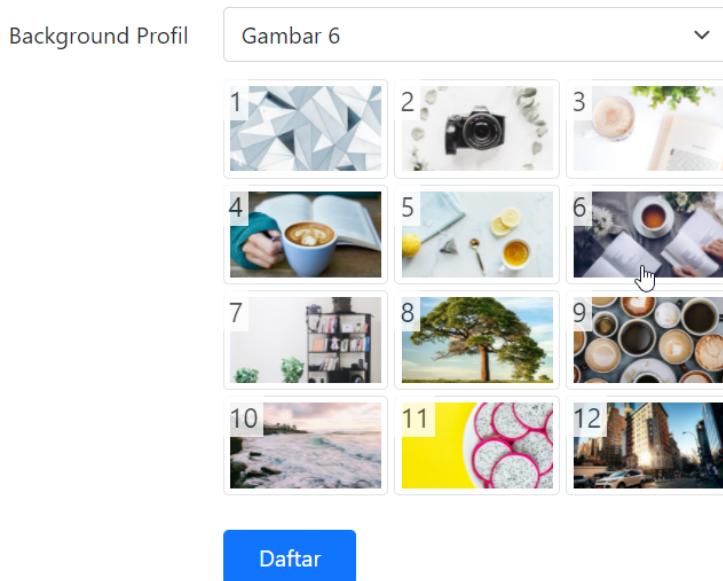
Jika form tampil saat register, yakni kondisi @if(\$tombol=='Daftar') terpenuhi, maka nilai atribut src dari tag akan berisi asset('img/default_profile.jpg'). Gambar default_profile.jpg ini sudah kita copy sebelumnya dan ada di dalam folder public\img\.



Gambar: File img/default_profile.jpg berbentuk gambar avatar abu-abu

Namun jika form tampil saat proses edit, giliran kondisi @elseif(\$tombol=='Update') yang terpenuhi, sehingga nilai atribut src akan berisi asset('storage/uploads/'.\$user->gambar_profil). Ini nantinya akan menampilkan gambar yang sudah di upload oleh user.

Inputan terakhir dipakai untuk memilih gambar background:



Gambar: Pilihan gambar background

Untuk gambar background, user hanya bisa memilih satu di antara 12 gambar yang tersedia. Proses input dilakukan dengan tag <select> yang berisi nilai teks 'Gambar 1' sampai dengan 'Gambar 12'.

Saya menggunakan perulangan @for untuk meng-generate tag <option> pilihan gambar. Termasuk di dalamnya kondisi @if untuk repopulate dan form update, kurang lebih sama seperti nama bulan sebelumnya.

Tampilan gambar dari tag juga di proses dengan perulangan @for di baris 171 – 177. Semua gambar ini sudah tersedia di folder public\img yang kita copy sebelumnya. Nama gambar saya rancang dengan format 'gambar1.jpg', 'gambar2.jpg', dst sampai 'gambar12.jpg', sehingga bisa di proses dengan perulangan for.

Kode untuk tampilan form pendaftaran user sebenarnya sudah selesai. Namun saya ingin menambah sedikit kode JavaScript untuk mempermudah pilihan gambar background.

27.10. Membuat File JavaScript

Sebenarnya file JavaScript ini bisa saja langsung kita tulis ke dalam view form.blade.php (internal JavaScript), tapi agar lebih rapi kode ini akan saya tempatkan ke dalam file terpisah (external JavaScript). Selain itu kita juga sudah mempelajari Laravel Mix yang sayang untuk tidak di praktikkan.

Untuk membuat file JavaScript external, silahkan buat file baru dengan nama my-script.js di dalam folder resources\js. Lalu isi dengan kode berikut:

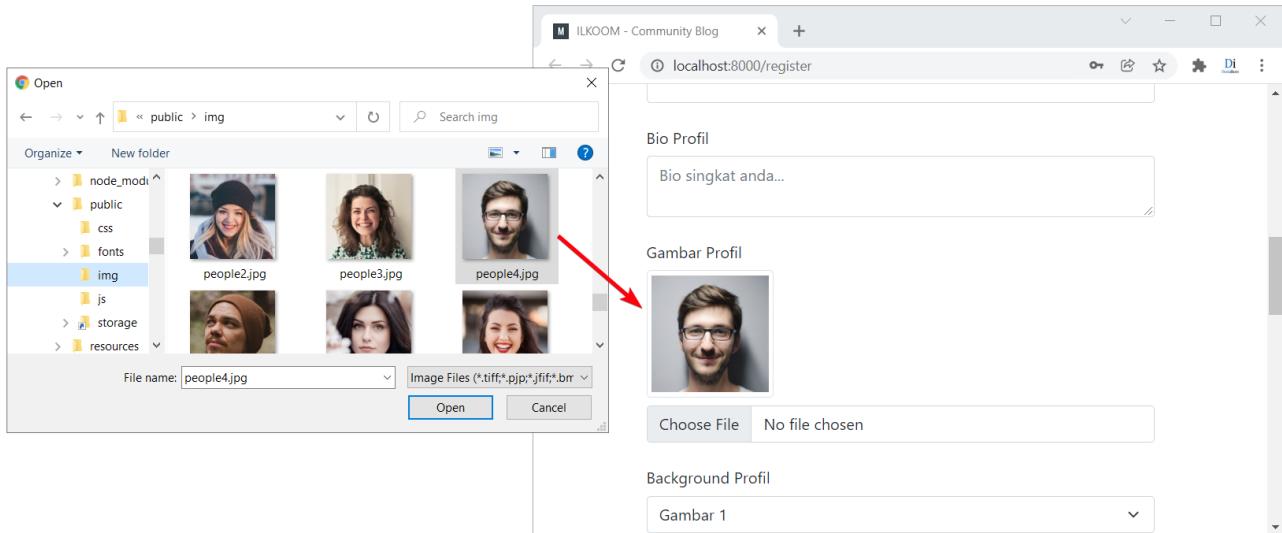
```
resources/js/my-script.js

1  /* =====
2  // Kode untuk membuat fitur preview gambar profil
3  // Dipakai pada form register dan form update
4  =====*/
5
6  let inputGambarProfile = document.getElementById('gambar_profil');
7  let displayGambarProfile = document.getElementById('display_gambar_profil');
8
9  if (inputGambarProfile) {
10    inputGambarProfile.addEventListener("change", previewGambar);
11
12    function previewGambar() {
13      const [file] = inputGambarProfile.files;
14      displayGambarProfile.src = URL.createObjectURL(file);
15    }
16
17    // agar ketika gambar profil di klik, file upload juga langsung terbuka
18    displayGambarProfile.addEventListener("click", () =>
19      inputGambarProfile.click()
20    );
21  }
22
23 /* =====
24 // Kode untuk membuat gambar pilihan background profil bisa di klik
25 // Dipakai pada form register dan form update
26 =====*/
27
28 let pilihanBg = document.getElementsByClassName('pilihan-background-profil');
29 let inputanBg = document.getElementById('background_profil');
```

```

31 if (pilihanBg) {
32 [...pilihanBg].forEach(element => element.addEventListener("click",updateBg))
33
34 function updateBg() {
35     inputanBg.value = this.children[0].innerHTML;
36 }
37 }
```

Blok pertama di baris 6 – 21 berguna untuk membuat preview dari gambar profil yang akan diupload oleh user. Berikut tampilan yang dimaksud:



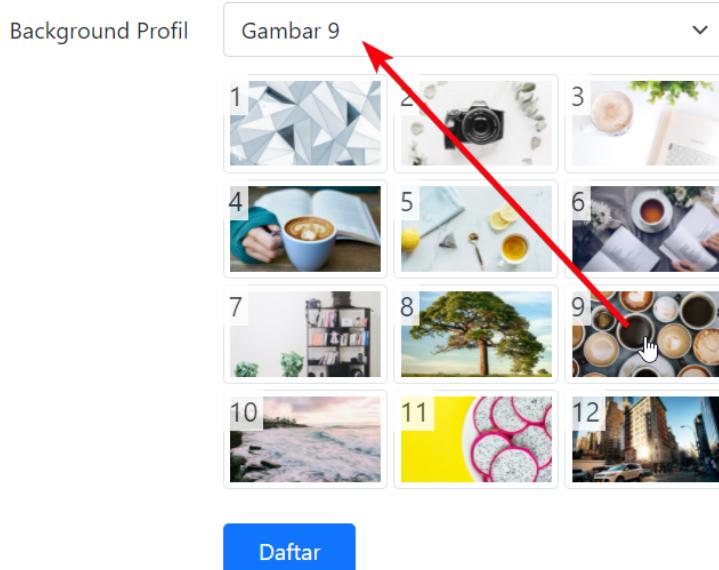
Gambar: Preview gambar profil tampil ke dalam tag

Begini user memilih gambar yang akan diupload, previewnya langsung tampil di dalam tag . Ini didapat dengan cara mengupdate property `src` milik node element tag dengan `URL.createObjectURL(file)`. Variabel `file` sendiri merujuk ke file gambar yang diupload user melalui tag <input type="file">. Proses ini berlangsung di dalam fungsi `previewGambar()` pada baris 12 – 15.

Kondisi `if(inputGambarProfile)` di baris 9 berguna untuk menghindari error saat file JS ini diakses oleh halaman lain (bukan dari halaman form). Ini dimungkinkan karena kita memakai Laravel Mix secara global. Dengan pemeriksaan kondisi if, proses penambahan event hanya bisa berlangsung jika halaman saat ini terdeteksi ada inputan form gambar profil.

Tambahan kode di baris 18–20 berguna untuk menambah event click ke tag . Ketika gambar di klik, otomatis akan membuka jendela upload file.

Blok kedua di baris 28 – 37 dipakai untuk membuat event click pilihan gambar background. Idenya adalah, jika user men-klik sample gambar background, itu akan mengubah pilihan dari tag <select>.



Gambar: Hasil kode JS untuk memudahkan memilih gambar background

Di baris 28, variabel `pilihanBg` akan berisi collection dari semua node element yang memiliki class `pilihan-background-profil`. Ini merujuk ke seluruh gambar background, sehingga total terdapat 12 element di dalam variabel `pilihanBg`.

Di baris 29, variabel `inputanBg` akan menampung node element dari tag `<select id="background_profil">`.

Selanjutnya di baris 32 saya "menempelkan" event `click` ke semua element yang tersimpan di dalam variabel `pilihanBg`. Operator `spread ...` dipakai untuk mengkonversi HTML collection hasil dari `getElementsByClassName()` menjadi array biasa. Ini diperlukan agar kita bisa melakukan perulangan dengan method `forEach()`.

Event `click` tersebut akan menjalankan fungsi `updateBg()` yang didefinisikan pada baris 29 - 31. Karena fungsi ini diakses dari dalam `addEventListener()`, maka `this` merujuk ke setiap element gambar yang sedang diklik. Perintah di baris 30 akan mengupdate nilai atribut `value` tag `<select id="background_profil">` sesuai dengan urutan gambar.

Sama seperti blok pertama, proses "penempelan" event `click` berada di dalam kondisi `if (pilihanBg)` agar tidak terjadi error jika kode JS ini diakses bukan dari halaman form

Save kode di atas, lalu modifikasi file `app.js` agar bisa di compile oleh Laravel mix:

`resources/js/app.js`

```
1 import './bootstrap';
2 import './my-script';
```

Kemudian buka cmd dan jalankan perintah `npx mix`.

Setelah proses compile selesai, silahkan buka kembali form pendaftaran, lalu klik salah satu gambar background. Harusnya itu akan mengupdate nilai tag `<select>` dari background profil.

Tambahan kode JavaScript ini sebenarnya lebih ke 'aksesoris tambahan', tidak berhubungan langsung dengan aplikasi Laravel untuk project kita.

27.11. Validasi Form Pendaftaran

Form pendaftaran sudah selesai, saatnya tulis kode untuk proses validasi dan proses input ke database. Kita bisa saja membuat controller baru, atau bisa juga dengan memodifikasi controller bawaan authentication.

Secara default, hasil form register user akan di proses oleh file RegisterController.php di folder app\Http\Controllers\Auth. Jika kita membuka file ini, terdapat method `validator()` untuk proses validasi inputan form serta method `create()` untuk proses input ke database.

Berikut kode bawaan yang tersedia:

app/Http/Controllers/Auth/RegisterController.php

```
1 <?php
2 /**
3  * Class RegisterController extends Controller
4 {
5     /**
6     * protected function validator(array $data)
7     {
8         return Validator::make($data, [
9             'name' => ['required', 'string', 'max:255'],
10            'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
11            'password' => ['required', 'string', 'min:8', 'confirmed'],
12        ]);
13    }
14
15   protected function create(array $data)
16   {
17       return User::create([
18           'name' => $data['name'],
19           'email' => $data['email'],
20           'password' => Hash::make($data['password']),
21       ]);
22   }
23 }
```

Bawaan dari authentication, sudah terdapat 3 buah inputan form yang di validasi, yakni `name`, `email` dan `password`. Kita akan modifikasi agar sesuai dengan inputan form pendaftaran user saat ini.

Pertama, perubahan untuk method `validator()`:

app/Http/Controllers/Auth/RegisterController.php

```
1 protected function validator(array $data)
2 {
```

```

3 // Satukan ketiga komponen tanggal
4 $tanggal_lahir = $data["thn"].str_pad($data["bln"],2,0,STR_PAD_LEFT).
5           str_pad($data["tgl"],2,0,STR_PAD_LEFT);
6 // Input kedalam array $data agar $tanggal_lahir bisa ikut di validasi
7 $data['tanggal_lahir'] = $tanggal_lahir;
8
9 return Validator::make($data, [
10   'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
11   'password' => ['required', 'string', 'min:8', 'confirmed'],
12   'nama' => ['required', 'string', 'max:255'],
13   'tanggal_lahir' => ['required', 'date', 'before:-10 years',
14                         'after:-100 years'],
15   'pekerjaan' => ['sometimes', 'nullable', 'string', 'max:255'],
16   'kota' => ['sometimes', 'nullable', 'string', 'max:255'],
17   'bio_profil' => ['sometimes', 'nullable', 'string'],
18   'gambar_profil' => ['sometimes','file','image','max:2000'],
19   'background_profil' => ['required', 'integer', 'min:1', 'max:12' ],
20 ]);
21 }

```

Method `validator()` ini memiliki argument `$data` yang berisi nilai semua inputan form, mirip seperti hasil `$request->all()`.

Di baris 4 saya membuat variabel `$tanggal_lahir` yang dipakai untuk menggabung 3 inputan tanggal lahir. Jika anda masih ingat, di dalam form sebelumnya tanggal lahir ini dipecah ke dalam 3 buah inputan: `tgl`, `bln` dan `thn`. Di sini saya menggabungkan ketiganya dengan urutan `thn+bln+tgl` sesuai dengan format tipe data date di MySQL.

Fungsi `str_pad()` dipakai untuk mengatasi angka bulan dan tanggal yang hanya terdiri dari 1 digit, misalnya tanggal '8' akan di konversi menjadi string '08'. Sehingga jika yang diinput dari form adalah `tgl=8, bln=7, thn=1980`, maka variabel `$tanggal_lahir` akan berisi `19800708`.

Di baris 7, hasil variabel `$tanggal_lahir` saya input ke dalam array `$data` agar bisa di validasi.

Syarat validasi ada di baris 10 – 19. Mayoritas syarat ini sudah sering kita pakai sebelumnya, termasuk syarat '`unique:users`' untuk inputan email, dimana Laravel akan memeriksa apakah sudah ada alamat email yang sama di tabel `users` atau belum.

Syarat baru yakni '`before:-10 years`' dan '`after:-100 years`' untuk inputan `tanggal_lahir`. Secara bawaan, syarat '`before`' dipakai untuk memastikan inputan harus berisi nilai yang kurang dari nominal tertentu, sedangkan syarat `after` untuk memastikan inputan harus lebih dari syarat tertentu.

Di sini saya memberikan nilai negatif sebagai trik untuk membuat pembatasan agar tanggal lahir hanya bisa diisi dengan tanggal yang kurang dari 10 tahun namun tidak lebih dari 100 tahun dari sekarang. Hasilnya, user dengan umur kurang dari 10 tahun dan lebih dari 100 tidak diizinkan untuk mendaftar.

Untuk inputan yang bersifat opsional, menggunakan syarat '`sometimes`', yang mencakup `pekerjaan`, `kota`, `bio_profil` dan `gambar_profil`.

Jika ada salah satu syarat ini yang tidak terpenuhi, halaman akan di redirect kembali ke dalam form beserta pesan error. Namun jika semuanya sudah sesuai, Laravel akan lanjut mengeksekusi method `create()`, yang akan saya modifikasi sebagai berikut:

app/Http/Controllers/Auth/RegisterController.php

```

1 //...
2 use Illuminate\Support\Str;
3
4 protected function create(array $data)
5 {
6     // Satukan ketiga komponen tanggal
7     $tanggal_lahir = $data["thn"].str_pad($data["bln"],2,0,STR_PAD_LEFT).
8         str_pad($data["tgl"],2,0,STR_PAD_LEFT);
9
10    // Ambil request object untuk proses upload file
11    $request = request();
12
13    // Proses upload file gambar profil
14    if ($request->hasFile('gambar_profil')) {
15        // gunakan slug helper agar "nama" bisa dipakai sebagai bagian
16        // dari nama gambar_profil
17        $slug = Str::slug($data['nama']);
18
19        // Ambil extensi file asli
20        $extFile = $request->gambar_profil->getClientOriginalExtension();
21
22        // Generate nama gambar, gabungan dari slug "nama"+time()+extensi file
23        $namaFile = $slug.'-'.$time().'.'.$extFile;
24
25        // Proses upload, simpan ke dalam folder "uploads"
26        $request->gambar_profil->storeAs('public/uploads',$namaFile);
27    }
28    else {
29        // jika user tidak mengupload gambar, isi dengan gambar default
30        $namaFile = 'default_profile.jpg';
31    }
32
33    // input data ke database
34    return User::create([
35        'email' => $data['email'],
36        'password' => Hash::make($data['password']),
37        'nama' => $data['nama'],
38        'tanggal_lahir' => $tanggal_lahir,
39        'pekerjaan' => $data['pekerjaan'],
40        'kota' => $data['kota'],
41        'bio_profil' => $data['bio_profil'],
42        'gambar_profil' => $namaFile,
43        'background_profil' => $data['background_profil'],
44    ]);
45 }
```

Di baris 2 saya menambah 1 buah perintah import untuk **Str** facade. Ini diperlukan untuk

mengubah nama file upload. Method `create()` sendiri dimulai dari baris 4 dengan tambahan argument `$data` yang berisi semua data inputan form.

Di baris 4 saya kembali meng-generate variabel `$tanggal_lahir`, sama seperti di method `validator()`.

Kode program di baris 11 – 31 dipakai untuk memproses upload file. Caranya sama seperti bahasan kita di bab file upload. Pertama, saya membuat variabel `$request` yang berisi `Request` object di baris 11.

Kondisi if di baris 14 diperlukan untuk memeriksa apakah ada gambar profile yang di upload pada saat proses registrasi. Jika ada, maka kode di baris 17 – 26 akan di proses. Setiap baris ini memiliki baris komentar mengenai fungsinya masing-masing.

Yang baru berupa penggunaan method facade `Str::slug($data['nama'])` di baris 17. Method ini berfungsi untuk menghasilkan string `slug` dari nama user, dimana nama user di dapat dari `$data['nama']`.

Slug sendiri adalah istilah untuk menyebut 'penamaan yang aman untuk URL', dimana spasi akan diganti dengan tanda hubung / strip " - ", serta menghapus tanda aneh seperti koma, dollar, atau karakter khusus lain.

Jadi misalkan nama user adalah "Rissa Lestari Putri", maka variabel `$slug` akan berisi string "`rissa-lestari-putri`".

Hasil dari `slug` ini saya gabung dengan `timestamp` dari fungsi `time()`, serta extension file asal di baris 23. Nama gambar inilah yang nantinya juga akan di simpan ke dalam tabel, seperti: `rissa-caem-1577104473.jpg`. Nama file ini akan ditampung oleh variabel `$namaFile`.

Penamaan ini relatif aman karena akan berbeda-beda setiap detik, kecuali ada user dengan nama yang sama, mengupload gambar profil dalam detik yang sama pula. Tentu saja jika anda ingin memastikan tidak ada bentrok nama file, maka bisa menggunakan nama yang di-generate laravel saja (sudah kita bahas di bab form upload).

Di baris 26, file gambar dipindah dari folder `temporary` ke folder `public\uploads` di `storage\app` menggunakan method `$request->gambar_profil->storeAs()`.

Jika ternyata tidak ada file yang di upload, maka variabel `$namaFile` akan diisi dengan string '`default_profile.jpg`', yakni gambar profile default yang nantinya akan kita siapkan.

Lanjut di baris 34, terdapat perintah `return User::create()`, dimana kita mengembalikan hasil dari method `create()` dari `User` object. Ini adalah teknik *mass assignment* dari Eloquent.

Sebagai argument untuk method `User::create()`, diisi dengan associative array berupa pasangan `nama_kolom => value`. Hampir semua value ini di dapat dari variabel `$data`, kecuali variabel `$tanggal_lahir` yang sudah kita rangkai manual, serta variabel `$namaFile` yang berisi nama file upload. Untuk inputan password, di proses lagi dengan method `Hash::make()` agar

yang disimpan ke dalam database berupa password hasil hashing.

Dengan tambahan ini, maka kita sudah bisa test proses validasi. Silahkan isi inputan yang akan menghasilkan error dan periksa apakah pesan validasi sudah bisa tampil serta proses repopulate sudah berjalan.

Form Pendaftaran

The screenshot shows a registration form with several fields and their validation messages:

- Email *: A red-bordered input field with an exclamation mark icon. Below it is the message "Email wajib diisi."
- Password *: A red-bordered input field with an exclamation mark icon. Below it is the message "Password wajib diisi."
- Ulangi Password *: An empty input field.
- Nama *: A red-bordered input field with an exclamation mark icon. Below it is the message "Nama wajib diisi."
- Tanggal Lahir *: A date input field showing "22 April 2020". It has three red-bordered dropdown menus for day, month, and year, each with an exclamation mark icon. Below it is the message "Tanggal lahir harus berisi tanggal sebelum -10 years."

Gambar: Pesan error validasi

Namun jika anda mencoba input data yang benar, akan tampil pesan error karena kita harus melakukan 1 langkah lagi, yakni mengatur property `$fillable` di `User` model.

27.12. Modifikasi User Model

Dalam bahasan sebelumnya, terlihat bahwa proses input ke tabel `users` menggunakan teknik `mass assignment`, sehingga kita perlu modifikasi file `app\Models\User.php` untuk menambah property `$fillable` atau `$guarded`.

Secara bawaan, sudah ada property `$fillable` dengan kode berikut:

app/Models/User.php

```
1 protected $fillable = [
2     'name', 'email', 'password',
3 ];
```

Ini merupakan kolom default dari sistem authentication Laravel. Silahkan modifikasi menjadi:

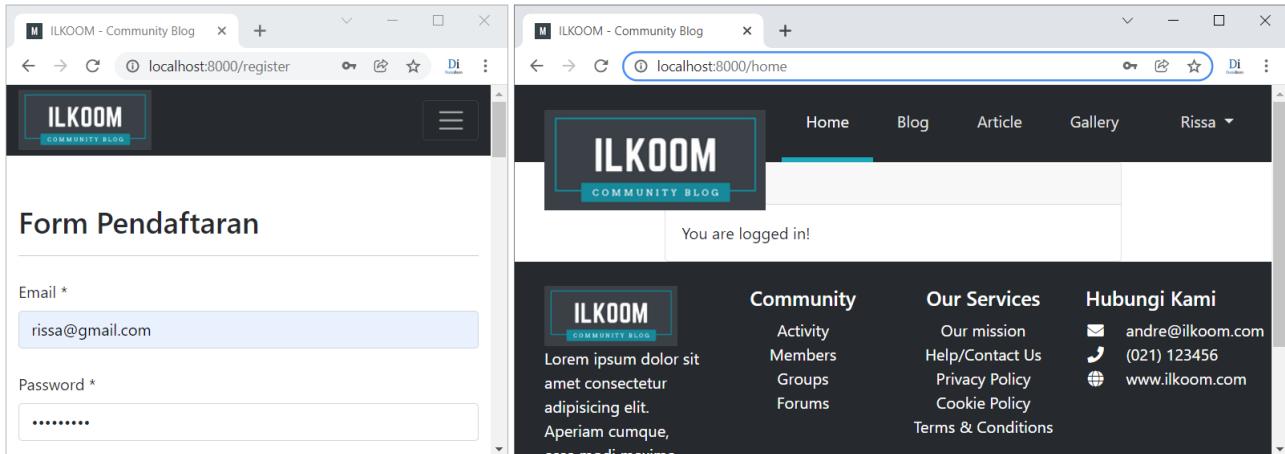
app/Models/User.php

```
1 protected $fillable = [
2     'email', 'password', 'nama', 'tanggal_lahir', 'pekerjaan',
3     'kota', 'bio_profil', 'gambar_profil', 'background_profil',
4 ];
```

Case Study: ILKOOM Profile Manager

Artinya, saya mengizinkan proses *mass assignment* untuk 9 kolom ke dalam tabel users.

Dengan tambahan ini, kita sudah bisa uji coba input data dari form pendaftaran:



Gambar: Hasil percobaan proses register

Jika lolos validasi, akan langsung login dan di redirect ke halaman localhost:8000/home. Tampilan halaman ini memang masih sedikit berantakan karena belum kita modifikasi. Tapi setidaknya sudah terlihat pesan 'You are logged in!'.

Untuk memastikan data sudah sampai ke tabel, bisa cek dari MySQL Client atau phpMyAdmin:

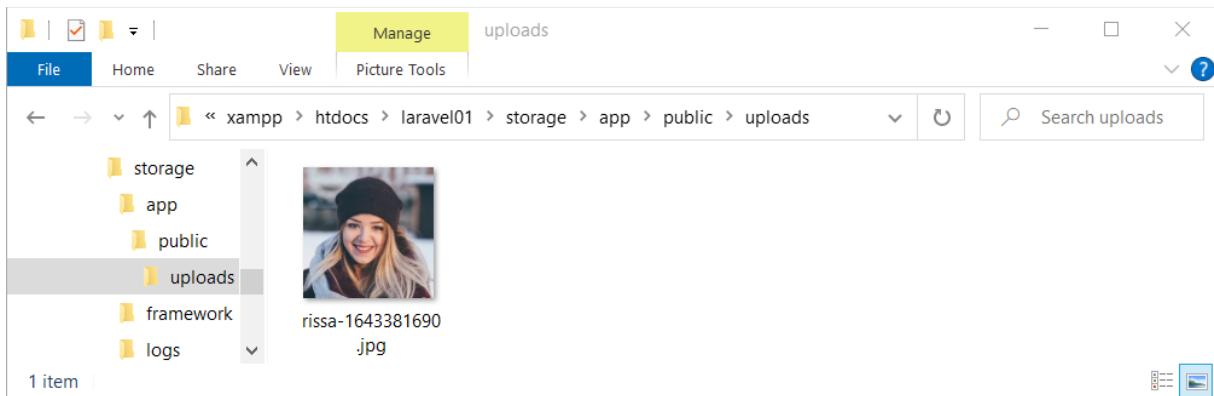
```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM users \G
***** 1. row *****
    id: 1
    email: rissa@gmail.com
    password: $2y$10$ls5/g1P2439rSLq9YqRIn.erZbzvzGBhqVMrKa1RgdYMsEddhQzM5m
        nama: Rissa
    tanggal_lahir: 1997-08-19
        pekerjaan: Customer Service PT. DuniaIlkom
        kota: Jakarta
    bio_profil: Masih mencari makna hidup ini
    gambar_profil: rissa-1643381690.jpg
background_profil: 7
email_verified_at: NULL
remember_token: NULL
    created_at: 2022-01-28 14:54:51
    updated_at: 2022-01-28 14:54:51
1 row in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Proses registrasi user Rissa sudah berhasil

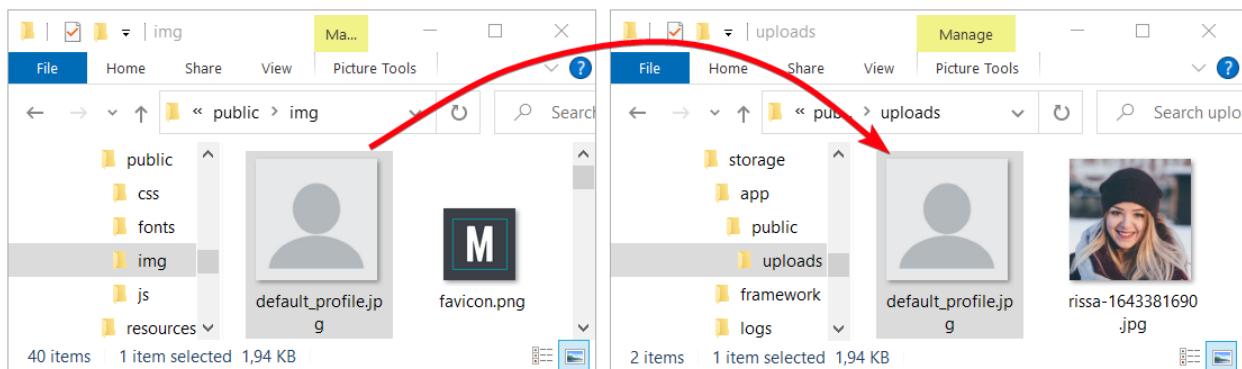
Terlihat isi password sudah di hashing serta kolom `gambar_profile` berisi string `rissa-1643381690.jpg` yang di generate dari nama user + timestamp.

Untuk file gambar profil, cek juga ke folder `storage\app\public\uploads`:



Gambar: File gambar profil yang sudah di upload

Selagi kita di folder ini, silahkan copy gambar `default_profile.jpg` dari folder `public\img` ke `storage\app\public\uploads`.



Gambar: Memindahkan file `default_profile.jpg`

Gambar `default_profile.jpg` ini saya siapkan sebagai gambar bagi user yang memutuskan untuk tidak mengupload gambar profilnya. Folder `public\img` sendiri berasal dari file `belajar_laravel.zip` di Google Drive.

27.13. Membuat symlink

Kita sudah berhasil meng-upload gambar profil. Tapi gambar tersebut belum bisa diakses dari view karena tidak berada di folder `public`, melainkan di `storage\app\public\uploads`.

Solusinya, tinggal membuat sebuah `symlink` sebagai mana yang kita bahas di bab file upload.

Silahkan buka cmd, masuk ke folder `laravel01`, lalu jalankan perintah berikut:

```
php artisan storage:link
```

```
Laravel01 Project
C:\xampp\htdocs\laravel01>php artisan storage:link
The [C:\xampp\htdocs\laravel01\public\storage] link has been connected to [C:\xampp\htdocs\laravel01\storage\app\public].
The links have been created.

C:\xampp\htdocs\laravel01>
```

Gambar: Membuat symlink

Sip, symlink berhasil di buat sehingga gambar profil yang di upload user sudah bisa diakses dari view.

27.14. Modifikasi HomeController

Ketika kita selesai melakukan proses pendaftaran (atau login), halaman akan langsung di redirect ke `localhost:8000/home`. Tampilan untuk halaman home ini di proses oleh view `home.blade.php` yang dipanggil dari method `index()` di `HomeController.php`. Ini sebenarnya berasal dari route bawaan berikut:

```
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
->name('home');
```

Saya ingin mengubah tampilan halaman home ini, yang dimulai dari proses modifikasi file `HomeController`.

Secara bawaan, file `HomeController.php` berisi kode program yang cukup singkat:

app/Http/Controllers/HomeController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class HomeController extends Controller
8 {
9     public function __construct()
10    {
11        $this->middleware('auth');
12    }
13
14    public function index()
15    {
16        return view('home');
17    }
18 }
```

Hanya ada 2 buah method, yakni sebuah constructor untuk menjalankan middleware, dan method `index()` yang memanggil view `home`.

Saya akan modifikasi isi dari kedua method ini. Untuk constructor, akan dihapus karena saya ingin halaman home bisa diakses oleh siapa saja, tidak harus login terlebih dahulu. Atau jika dihapus terasa cukup ekstrim, bisa dijadikan baris komentar saja.

Sedangkan untuk method `index()`, saya akan tambah sedikit kode program agar kita bisa mengirim semua isi tabel `users` ke dalam view `home`.

Berikut kode akhir dari `HomeController` setelah modifikasi:

app/Http/Controllers/HomeController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\User;
7
8 class HomeController extends Controller
9 {
10     # Di nonaktifkan agar halaman home bisa diakses oleh siapa saja
11     # (tidak harus login dulu)
12     // public function __construct()
13     // {
14     //     $this->middleware('auth');
15     // }
16
17     public function index()
18     {
19         $users = User::all();
20         return view('home',compact('users','users'));
21     }
22 }
```

Tambahan pertama ada di baris 6, dimana saya mengimport User model yang akan dipakai untuk mengambil isi tabel users. Proses pengambilan ini dilakukan pada baris 19 dengan method User::all(). Variabel \$users selanjutnya dikirim ke view home.blade.php.

Method __construct() saya non aktifkan agar halaman home ini bisa diakses oleh siapa saja (tidak perlu login terlebih dahulu).

27.15. Modifikasi View home.blade.php

Berikutnya, saya akan tulis ulang kode program untuk view home.blade.php. Ini merupakan halaman utama dari aplikasi **ILKOOM Profile Manager**. Di sini akan tampil profil dari setiap user lengkap dengan foto serta semua data yang diinput pada saat proses registrasi.

Sebelum itu, silahkan buat 5 user baru agar tampilan ini menjadi lebih pas (total menjadi 6 user). Berikut kode program yang diperlukan:

resources/views/home.blade.php

```
1 @extends('layouts.app')
2
3 @section('content')
4
5 <!-- MEMBER LIST -->
6 <section id="member-list" class="py-5 bg-light text-center">
7     <div class="container">
8         <div class="row">
```

```

9      <div class="col text-center" >
10     <h1>Member List</h1>
11     <hr class="w-25 mx-auto">
12     <p class="lead">Lorem ipsum dolor sit amet consectetur adipisicing elit. Dignissimos, vitae!</p>
13
14
15     {{-- Untuk flash message --}}
16     @if(session()->has('pesan'))
17       @if(session()->get('pesan') == 'update')
18         <div class="alert alert-success alert-dismissible w-50 mx-auto fade show">
19           Data <b>{{session()->get('nama')}}</b> berhasil di update
20           <button type="button" class="btn-close" data-bs-dismiss="alert">
21             </button>
22         </div>
23     @endif
24     @if(session()->get('pesan') == 'delete')
25       <div class="alert alert-danger alert-dismissible w-50 mx-auto fade show">
26         Data <b>{{session()->get('nama')}}</b> sudah dihapus
27         <button type="button" class="btn-close" data-bs-dismiss="alert">
28           </button>
29       </div>
30     @endif
31   @endif
32
33
34
35   </div>
36 </div>
37
38 {{-- Proses looping untuk menampilkan semua user --}}
39 <div class="row row-cols-1 row-cols-md-2 row-cols-lg-3">
40   @forelse ($users as $user)
41     <div class="col mt-3">
42       <div class="card">
43         
45         <div class="card-body">
46           
48           <h5 class="card-title">{{$user->nama}}</h5>
49           <p class="card-text">"{{$user->bio_profil ?? '...'}}"</p>
50           <ul class="fa-ul text-start">
51             <li class="mb-2">
52               <span class="fa-li"><i class="far fa-clock"></i></span>
53               Join in {{ date('F Y', strtotime($user->created_at)) }}
54             </li>
55             <li class="mb-2">
56               <span class="fa-li"><i class="fas fa-briefcase"></i></span>
57               {{$user->pekerjaan ?? '...'}}

```

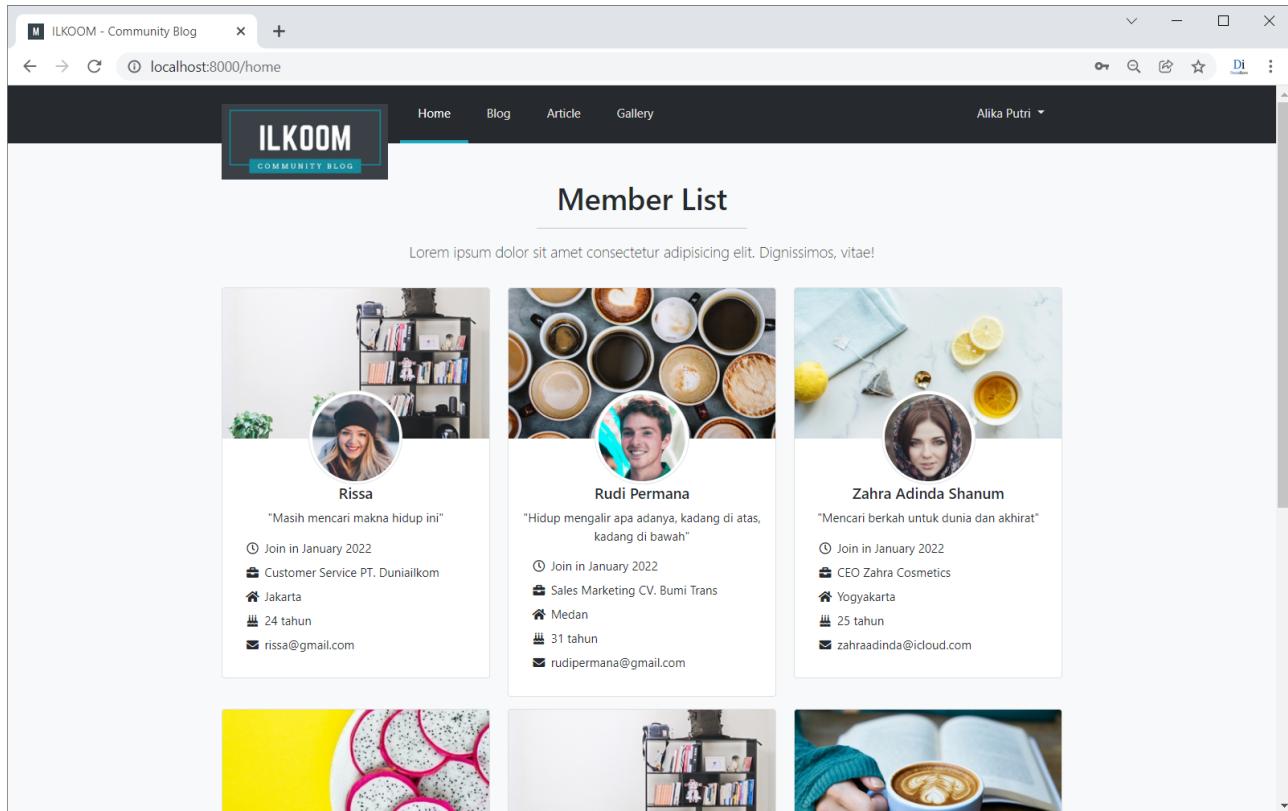
```

64          <span class="fa-li"><i class="fas fa-birthday-cake"></i></span>
65          {{ date_diff(date_create($user->tanggal_lahir ),
66                         date_create('now'))->y }} tahun
67      </li>
68      <li class="mb-2">
69          <span class="fa-li"><i class="fas fa-envelope"></i></span>
70          {{$user->email}}
71      </li>
72  </ul>
73  {{{-- Tombol edit & hapus hanya untuk user sendiri atau admin --}}}
74  {{{-- Ini menggunakan laravel policy --}}}
75  @can('update', $user)
76      <div class="btn-action">
77          <a href="{{ url('/users/'.$user->id.'/edit')}}" 
78              class="btn btn-primary d-inline-block">Edit</a>
79          <button class="btn btn-danger btn-hapus"
80              data-id="{{{$user->id}}}" data-bs-toggle="modal"
81              data-bs-target="#DeleteModal">Hapus</button>
82      </div>
83      @endcan
84  </div>
85  </div>
86  </div>
87  @empty
88      <p>Tidak ada data...</p>
89  @endforelse
90  </div>
91 </div>
92 </section>
93
94 {{{-- Modal untuk konfirmasi proses delete --}}}
95
96 <div id="DeleteModal" class="modal fade" role="dialog">
97     <div class="modal-dialog ">
98         <!-- Modal content-->
99         <form action="" id="deleteForm" method="post">
100             @csrf
101             @method('DELETE')
102             <div class="modal-content">
103                 <div class="modal-header">
104                     <h4 class="modal-title text-center">Konfirmasi</h4>
105                     <button type="button" class="btn-close" data-bs-dismiss="modal">
106                         </button>
107                     </div>
108                     <div class="modal-body">
109                         <p class="text-center mb-0">Anda yakin akan menghapus User ini?</p>
110                     </div>
111                     <div class="modal-footer">
112                         <button type="button" class="btn btn-success" data-bs-dismiss="modal">
113                             Cancel</button>
114                         <button type="submit" class="btn btn-danger" data-bs-dismiss="modal">
115                             Ya, Hapus</button>
116                     </div>
117                 </div>
118             </form>

```

```
119  </div>
120 </div>
121
122 @endsection
```

Berikut tampilan view `home.index.php` dengan 6 user yang sudah terdaftar:



Gambar: Tampilan halaman home

Untuk membuat design tampilan seperti ini saya menggunakan komponen **Card** Bootstrap. Mengenai design tidak akan di bahas terlalu dalam karena fokus utama kita lebih ke kode Laravel saja. Jika tertarik, bahasan detail tentang design ini ada di studi kasus bab terakhir buku **Bootstrap Uncover**.

Seperti view lainnya, di baris pertama terdapat kode untuk meng-extends `layout\app.blade.php`. Sehingga view `home` sudah berisi kode `<head>` HTML, menu header dan footer. Kemudian di baris 6 – 13 terdapat kode pembuka struktur grid Bootstrap yang berisi judul "Member List" serta teks dummy `lorem ipsum`. Anda bisa ganti dengan teks lain jika diinginkan. Kode di baris 16 – 33 dipakai untuk menampilkan flash session pesan. Di sini saya menggunakan komponen **alert** Bootstrap untuk menampilkan isi pesan. Terdapat penggunaan nested if, pertama perintah `@if(session()->has('pesan'))` di sisi luar untuk memeriksa apakah ada flash session pesan atau tidak. Jika ada, maka diperiksa lagi apakah pesan ini dari proses `update`, atau dari proses `delete`. Pemeriksaan dilakukan dengan perintah `@if` di baris 17 dan 25. Untuk setiap kondisi, tampilkan pesan yang sesuai.

Proses menampilkan data profil untuk semua user dilakukan di baris 39 – 91. Di sini saya menggunakan fitur **row columns** dari Bootstrap dengan menulis class `row row-cols-1 row-cols-md-2 row-cols-lg-3` di baris 39. Ini akan membuat efek responsive untuk setiap card.

Kemudian terdapat perulangan `@forelse ($users as $user)` dari baris 40 – 89. Variabel `$users` sendiri berasal dari method `index()` di `HomeController`. Isinya berupa array dari semua user yang ada di tabel `users`. Dalam perulangan ini saya mengakses semua data user, mulai dari `$user->background_profil` yang berisi gambar background, `$user->gambar_profil` yang berisi gambar profil, hingga `$user->email`.

Setiap data disusun dengan rapi ke dalam komponen **Card** Bootstrap, termasuk tambahan icon font awesome. Untuk menampilkan gambar, saya memakai function helper `asset()`, dengan gambar background tersimpan di folder `img` (baris 44), dan gambar profil ada di folder `storage/uploads` (baris 46).

Saya juga menampilkan informasi kapan user mendaftar dengan teks 'Join in' di baris 53. Informasi ini berasal dari kolom `created_at`. Function `date()` dan `strtotime()` dipakai untuk menformat tampilan tanggal.

Untuk menampilkan data pekerjaan dan kota, saya tulis dengan operator `??`, seperti `>{{$user->pekerjaan ?? '...'}}`. Artinya jika user tersebut tidak mengisi kolom pekerjaan, maka tampilkan teks '...'. Ini saya buat agar design kita tidak berantakan jika user tidak mengisi data pekerjaan dan kota.

Meskipun di tabel `users` terdapat kolom `tanggal_lahir`, saya tidak menampilkan langsung informasi tanggal lahir, tapi umur dari user tersebut. Caranya adalah dengan fungsi `date_diff()` bawaan PHP seperti di baris 65.

Di baris 75 – 83 terdapat blok kode `@can('update', $user)`. Ini saya siapkan untuk fitur policy yang akan kita buat sesaat lagi. Artinya, kode ini hanya tampil bagi user yang memiliki hak akses untuk proses update. Jika dipenuhi, tampilkan tombol **Edit** dan tombol **Delete**.

Kode di baris 96 – 120 merupakan kode tambahan untuk jendela popup yang saya buat dari komponen **Modal** Bootstrap. Jendela ini dipakai untuk konfirmasi proses delete, dimana jika user men-klik tombol Delete, akan tampil jendela konfirmasi dengan pertanyaan *Anda yakin akan menghapus User ini?*

Jendela modal ini butuh sedikit kode JavaScript yang akan saya bahas kembali ketika kita sampai ke perancangan fitur delete user.

27.16. Membuat UserController

Sekarang kita akan masuk ke perancangan proses edit/update serta proses delete user. Kode ini saya tempatkan ke sebuah controller baru bernama `UserController`.

Silahkan buka cmd dan jalankan perintah berikut:

```
php artisan make:controller UserController
```

Di dalam file UserController.php inilah kita akan tulis beberapa method untuk proses edit dan delete user.

27.17. Modifikasi Route

Sampai sekarang kita masih menggunakan route bawaan authentication Laravel, berikut isi route saat ini:

routes/web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Auth::routes();
10
11 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
12 ->name('home');
```

Saya akan modifikasi menjadi:

routes/web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use Illuminate\Support\Facades\Auth;
5 use App\Http\Controllers\UserController;
6
7 Auth::routes();
8
9 Route::get('/', [App\Http\Controllers\HomeController::class, 'index'])
10 ->name('home');
11
12 Route::get('/users/{user}/edit', [UserController::class, 'edit']);
13 Route::put('/users/{user}', [UserController::class, 'update']);
14 Route::delete('/users/{user}', [UserController::class, 'destroy']);
```

Route di baris 9 berfungsi agar ketika halaman root diakses, yakni alamat localhost:8000, langsung tampilkan view home, tidak lagi view welcome bawaan Laravel.

Saya juga menghapus route Route::get('/home'), sehingga kita perlu modifikasi beberapa controller bawaan authentication. Ini karena secara default ketika user selesai melakukan

proses registrasi atau melakukan proses login, halaman akan langsung *redirect* ke route '/home'. Kita perlu ubah ini menjadi ke route '/' sesaat lagi.

Kemudian di baris 12 - 14 terdapat tambahan 3 route baru. Route ini merupakan bagian dari **RESTfull** route yang sudah beberapa kali kita gunakan dalam perancangan CRUD. Ketiganya dipakai untuk menampilkan form edit, proses update, serta proses delete user. Route ini akan mengakses method `edit()`, `update()` dan `destroy()` milik `UserController`.

27.18. Modifikasi Property \$redirectTo

Ketika memodifikasi route, saya menghapus route ke halaman '/home', sehingga halaman 'localhost:8000/home' sudah tidak bisa diakses lagi. Namun '/home' menjadi halaman default ketika user selesai register dan login.

Untungnya, cara untuk mengubah proses ini sangat mudah, cukup dengan mengganti nilai dari property `$redirectTo` di file `Auth\RegisterController.php` dan `Auth\LoginController.php`. Silahkan anda buka kedua file ini, lalu cari baris berikut:

```
protected $redirectTo = RouteServiceProvider::HOME;
```

dan ubah menjadi:

```
protected $redirectTo = '/';
```

Dengan perubahan ini, maka ketika user selesai register atau login, akan langsung di redirect ke halaman root, yakni 'localhost:8000'.

27.19. Menyiapkan Data Update

Untuk membuat fitur update user, kita perlu 2 buah route yang sudah ditambahkan sebelumnya, yakni:

```
Route::get('/users/{user}/edit', [UserController::class, 'edit']);  
Route::put('/users/{user}', [UserController::class, 'update']);
```

Route pertama dipakai untuk menampilkan form update, sedangkan route kedua dipakai untuk proses update ke database. Keduanya akan di proses oleh method `edit()` dan `update()` di `UserController`.

Berikut kode program method `edit()` untuk menampilkan form update:

app/Http/Controllers/UserController.php

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4 use App\Models\User;
```

```

5  use Illuminate\Http\Request;
6  use Illuminate\Support\Str;
7
8  class UserController extends Controller
9  {
10     public function edit(User $user)
11     {
12         // Extract komponen tanggal_lahir dari MySQL, masukkan ke array $user
13         $time = strtotime($user->tanggal_lahir);
14
15         $user['tgl'] = date('d', $time);
16         $user['bln'] = date('m', $time);
17         $user['thn'] = date('Y', $time);
18
19         return view('user.edit', compact('user'));
20     }
21 }

```

Tambahan pertama berupa perintah import `use App\Models\User` di baris 4, ini diperlukan karena kita akan memakai teknik *route model binding* serta beberapa Eloquent method. Kemudian juga perlu facade class `Illuminate\Support\Str` untuk memproses file upload.

Sebagai argument dari method `edit()` adalah variabel `$user` yang otomatis berisi semua data user dari database (menggunakan *route model binding*). Namun sebelum di teruskan ke form update, saya perlu mengonversi isi kolom `tanggal_lahir` terlebih dahulu.

Di dalam database, tanggal lahir ini tersimpan dalam 1 kolom: `tanggal_lahir`. Namun di dalam form nanti, tanggal lahir di pecah ke dalam 3 buah inputan, yakni `$tgl`, `$bln` dan `$thn`.

Kode program di baris 13 – 17 dipakai untuk memecah tanggal lahir ini. Pertama saya mengonversi data `$user->tanggal_lahir` menjadi tipe data date PHP dengan fungsi `strtotime`. Kemudian memakai function `date()` untuk mengambil komponen tanggal, bulan dan tahun. Ketiga data ini diinput kembali ke dalam array `$user` agar bisa diakses dari view.

Terakhir, perintah `return` di baris 19 akan menampilkan view `user\edit.blade.php` dengan mengirim variabel `user`. Di dalam form update, data `$user` ini akan menjadi nilai awal inputan form.

27.20. Membuat Form Update

Tampilan form update akan di proses oleh view `user\edit.blade.php`. Silahkan buat folder `user` di `resources\views\`, lalu buat file `edit.blade.php` dengan kode berikut:

`resources/views/user/edit.blade.php`

```

1  @extends('layouts.app')
2
3  @section('content')
4
5  <!-- HEADER IMAGE -->

```

```

6  <header id="update-header" class="header-image text-white d-none d-md-block">
7      <div class="header-overlay">
8          <div class="container">
9              <div class="row">
10                 <div class="col">
11                     <h1 class="display-1">Join our Community</h1>
12                     <p>Bergabunglah dengan salah satu blog komunitas terbaik
13                         di Indonesia</p>
14                 </div>
15             </div>
16         </div>
17     </div>
18 </header>
19
20 <div class="container my-5">
21     <div class="row ">
22         <div class="col-lg-8">
23             <h1>Edit Data</h1>
24             <hr>
25             <form method="POST" action="{{ url('/users/'.$user->id) }}"
26               enctype="multipart/form-data">
27                 @method('PUT')
28                 @include('layouts.form',['tombol' => 'Update'])
29             </form>
30         </div>
31     </div>
32 </div>
33
34 @endsection

```

Kode program untuk view `edit.blade.php` ini sangat mirip seperti `register.blade.php`, yakni view yang kita pakai untuk menampilkan form pendaftaran.

Perbedaan utama terdapat pada tag `<form>` di baris 25 – 29. Sekarang alamat pengiriman data form adalah ke `url('/users/'.$user->id)`. Selain itu karena ini adalah proses update, saya menggunakan perintah `method('PUT')` agar sesuai dengan struktur **RESTfull**.

Kode untuk menampilkan berbagai tag `<input>` ditangani oleh view `layouts\form.blade.php`. View ini sudah kita bahas karena memang saya rancang untuk form pendaftaran dan juga form update. Sebagai pembeda, saya mengirim variabel `tombol` dengan string 'Update'.

Save view `edit.blade.php`, dan untuk uji coba silahkan akses localhost:8000/users/1/edit atau localhost:8000/users/2/edit. Maka seharusnya akan tampil form edit lengkap dengan data dari user dengan id 1 dan 2:

The screenshot shows a web browser window titled 'ILKOOM - Community Blog' with the URL 'localhost:8000/users/4/edit'. The page is titled 'Edit Data' and contains a form for updating user profile information. The fields include:

- Email *: anissa@gmail.com
- Nama *: Anissa Lestari
- Tanggal Lahir *: 23 April 2000
- Pekerjaan: Assistant Manager PT. Jaya Selalu
- Kota: Surabaya
- Bio Profil: Hidup hanya sekali, harus bermakna dan bermanfaat untuk orang lain
- Gambar Profil: A thumbnail of a woman smiling.
- Background Profil: A dropdown menu showing 'Gambar 11' and a grid of 12 numbered thumbnail images (1-12) representing different backgrounds.

At the bottom of the form is a blue 'Update' button.

Gambar: Tampilan form edit

Ketika tombol **Update** di klik, masih tampil pesan error karena kita belum membuat method untuk memprosesnya.

27.21. Memproses Update User

Proses dari form update akan ditangani oleh method `update()` di `UserController`. Berikut kode program yang diperlukan:

app/Http/Controllers/UserController.php

```
1 public function update(Request $request, User $user)
2 {
3     // Satukan ketiga komponen tanggal
4     $tanggal_lahir = $request["thn"].str_pad($request["bln"],2,0,STR_PAD_LEFT).
5                     str_pad($request["tgl"],2,0,STR_PAD_LEFT);
6
```

```

7 // Input kedalam array $request agar $tanggal_lahir bisa ikut di validasi
8 $request['tanggal_lahir'] = $tanggal_lahir;
9
10 $validateData = request()->validate([
11     'email' => ['required', 'string', 'email', 'max:255',
12                 'unique:users,email,'.$user->id],
13     'nama' => ['required', 'string', 'max:255'],
14     'tanggal_lahir' => ['required','date', 'before:-10 years',
15                           'after:-100 years'],
16     'pekerjaan' => ['sometimes', 'nullable', 'string', 'max:255'],
17     'kota' => ['sometimes', 'nullable', 'string', 'max:255'],
18     'bio_profil' => ['sometimes', 'nullable', 'string'],
19     'gambar_profil' => ['sometimes','file','image','max:2000'],
20     'background_profil' => ['required', 'integer', 'min:1', 'max:12' ],
21 ]);
22
23 // Proses upload file gambar profil
24 if ($request->hasFile('gambar_profil')) {
25     // gunakan slug helper agar "nama" bisa dipakai sebagai bagian
26     // dari nama gambar_profil
27     $slug = Str::slug($request['nama']);
28
29     // Ambil extensi file asli
30     $extFile = $request->gambar_profil->getClientOriginalExtension();
31
32     // Generate nama gambar, gabungan dari slug "nama"+time()+extensi file
33     $namaFile = $slug.'-'.$time().'.'.$extFile;
34
35     // Proses upload, simpan ke dalam folder "uploads"
36     $request->gambar_profil->storeAs('public/uploads',$namaFile);
37 }
38 else {
39     // jika user tidak mengupload gambar, isi variabel $path dengan null
40     $namaFile = $user->gambar_profil;
41 }
42
43 $validateData['gambar_profil'] = $namaFile;
44
45 $user->update($validateData);
46
47 return redirect('/#member-list')->with(['pesan' => 'update',
48 'nama' => $user->nama]);
49 }

```

Kode program untuk proses update ini sangat mirip seperti proses **create** (pendaftaran user). Namun kali ini kita mengakses data inputan form dari **Request** object melalui variabel **\$request**, tidak lagi dari variabel **\$data** bawaan authentication.

Secara berurutan, proses yang terjadi adalah: penggabungan data tanggal lahir di baris 4 - 5, input tanggal lahir ke array **\$request** di baris 8, validasi inputan form di baris 10 – 21, proses upload file gambar profil di baris 24 – 41, update ke tabel user di baris 45, dan *redirect* ke halaman root di baris 47.

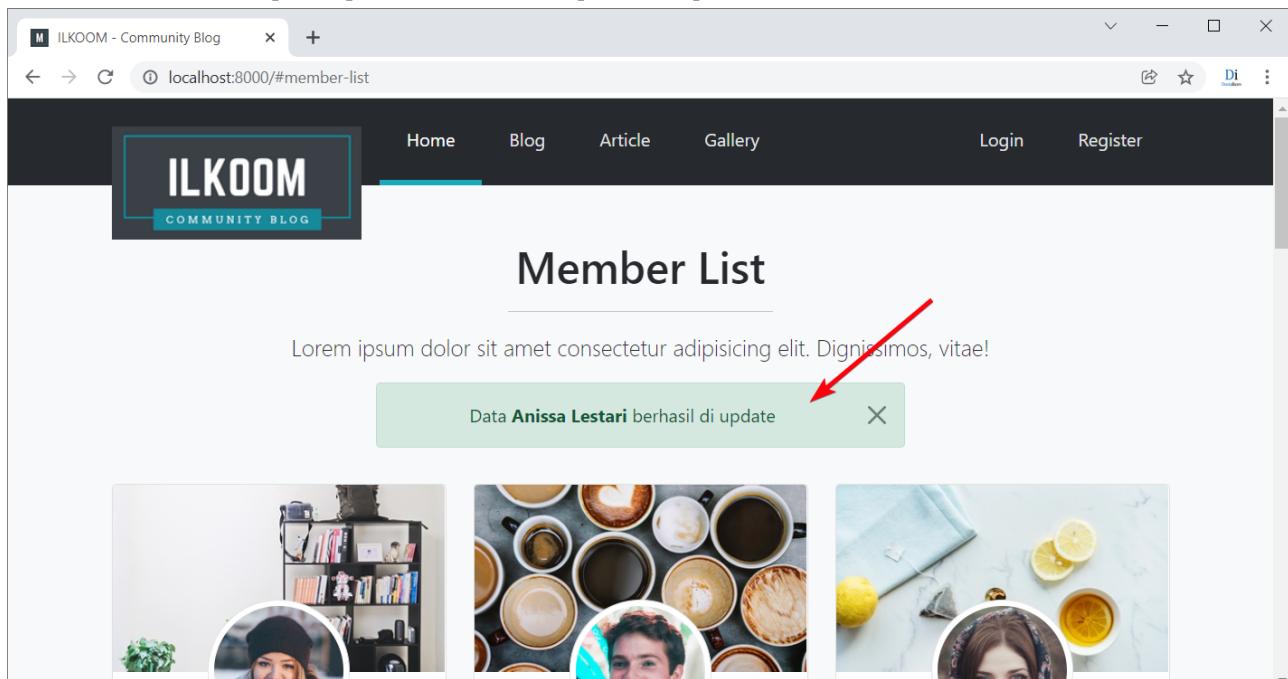
Syarat validasi yang saya pakai nyaris sama seperti di pendaftaran user, perbedaan ada di tambahan syarat email 'unique:users,email,'.\$user->id yang dipakai untuk menghindari masalah dengan alamat email yang sama.

Begitu juga dengan proses file upload yang tidak ada perbedaan dengan kode di pendaftaran user, dimana nama file nanti adalah gabungan dari nama user + time().

Proses update ke tabel user dilakukan hanya dengan 1 perintah Eloquent, yakni \$user->update(\$validateData).

Setelah proses update berhasil, user akan di redirect ke halaman root dengan flash session \$pesan yang berisi string 'update' dan \$nama yang berisi hasil dari \$user->nama. Kedua flash session ini akan diperiksa di bagian atas view home.blade.php.

Berikut contoh tampilan pesan flash hasil proses update:



Gambar: Tampilan flash message proses update berhasil

Saat ini untuk masuk ke form update masih dilakukan manual, yakni dengan mengetik langsung alamat URL di web browser, seperti <localhost:8000/users/2/edit>. Sebenarnya, tombol edit dan juga tombol delete sudah ada di view home.blade.php, tapi masih "tersembunyi" karena hak akses (policy) yang perlu kita rancang sesaat lagi.

27.22. Membuat Proses Delete User

Route untuk proses delete sudah ada sebelumnya, yakni:

```
Route::delete('/users/{user}', [UserController::class, 'destroy']);
```

Sehingga kita tinggal membuat isi dari method `destroy()` di `UserController`.

Form untuk menjalankan route ini ada di view `home.blade.php`, yakni jendela popup (komponen Modal Bootstrap). Tampilannya memang belum bisa dilihat karena masih dibatasi dengan hak akses policy. Berikut kode yang dimaksud:

`resources/views/home.blade.php`

```
1 ...
2 {{-- Modal untuk konfirmasi proses delete --}}
3
4 <div id="DeleteModal" class="modal fade" role="dialog">
5   <div class="modal-dialog ">
6     <!-- Modal content-->
7       <form action="" id="deleteForm" method="post">
8         @csrf
9         @method('DELETE')
10        <div class="modal-content">
11          <div class="modal-header">
12            <h4 class="modal-title text-center">Konfirmasi</h4>
13            <button type="button" class="btn-close" data-bs-dismiss="modal">
14            </button>
15          ...
16        ...
17      ...
18    ...
19  ...
20</div>
21</div>
```

Ringkasnya, jika user men-klik tombol delete, itu akan mengakses method `destroy()` berikut:

`app/Http/Controllers/UserController.php`

```
1 public function destroy(User $user)
2 {
3   $user->delete();
4   return redirect('/#member-list')->with(['pesan' => 'delete',
5   'nama' => $user->nama]);
6 }
```

Tambahkan kode di atas ke dalam `UserController.php`. Dengan *route model binding* dari Eloquent, proses penghapusan data cukup dengan 1 perintah sederhana, yakni `$user->delete()`. Setelah itu user akan di redirect ke halaman root dengan flash session pesan dan nama, mirip seperti proses update sebelumnya.

Praktek penghapusan user belum bisa kita lakukan karena tombol Delete masih "terkurung" oleh hak akses. Inilah yang akan kita rancang berikutnya.

27.23. Membuat Policy

Untuk aplikasi ILKOOM Profile Manager ini saya ingin membuat pembatasan bahwa proses edit dan delete hanya bisa dilakukan oleh user itu sendiri. Ini tentu sangat logis bahwa yang bisa mengedit data Rissa hanya user Rissa saja. User Alex tidak bisa mengedit data Rissa.

Selain itu saya juga ingin sebuah "super user", yakni 1 user admin yang bisa mengedit dan

menghapus semua data user. Dengan menggunakan fitur Laravel Policy, batasan seperti ini cukup mudah di implementasikan.

Pertama, kita perlu membuat sebuah file policy terlebih dahulu, yang saya beri nama **UserPolicy**:

```
php artisan make:policy UserPolicy -m User
```

Setelah itu edit file **UserPolicy.php** dengan kode berikut:

```
app/Policies/UserPolicy.php
```

```
1 <?php
2
3 namespace App\Policies;
4
5 use App\Models\User;
6 use Illuminate\Auth\Access\HandlesAuthorization;
7
8 class UserPolicy
9 {
10     use HandlesAuthorization;
11
12     //...
13
14     public function update(User $user, User $model)
15     {
16         return in_array($user->email, [
17             'admin@gmail.com',
18             $model->email,
19         ]);
20     }
21
22     public function delete(User $user, User $model)
23     {
24         return in_array($user->email, [
25             'admin@gmail.com',
26             $model->email,
27         ]);
28     }
29     //...
30 }
```

Method yang kita perlukan hanya **update()** dan **delete()**. Isinya berupa function **in_array()** untuk memeriksa apakah email user saat ini '**admin@gmail.com**' atau **\$model->email**.

Perhatikan isi argument dari kedua method tersebut, terdapat variabel **\$user** dan **\$model** yang sama-sama **User** object. Variabel **\$user** berisi data user yang sedang login saat ini, sedangkan variabel **\$model** berisi data model yang ingin di edit (diambil dari database).

Batasan di baris 16-19 dan 24-27 hanya mengizinkan proses edit untuk user yang memiliki email '**admin@gmail.com**' atau user yang alamat emailnya sama dengan data yang akan di edit.

Langkah berikutnya adalah menerapkan batasan ini ke dalam route atau controller. Kali ini saya memilih mendaftarkan lewat route saja. Maka berikut modifikasi file routes\web.php:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/users/{user}/edit', [UserController::class, 'edit'])
4 ->middleware('can:update,user');
5
6 Route::patch('/users/{user}', [UserController::class, 'update'])
7 ->middleware('can:update,user');
8
9 Route::delete('/users/{user}', [UserController::class, 'destroy'])
10->middleware('can:delete,user');
```

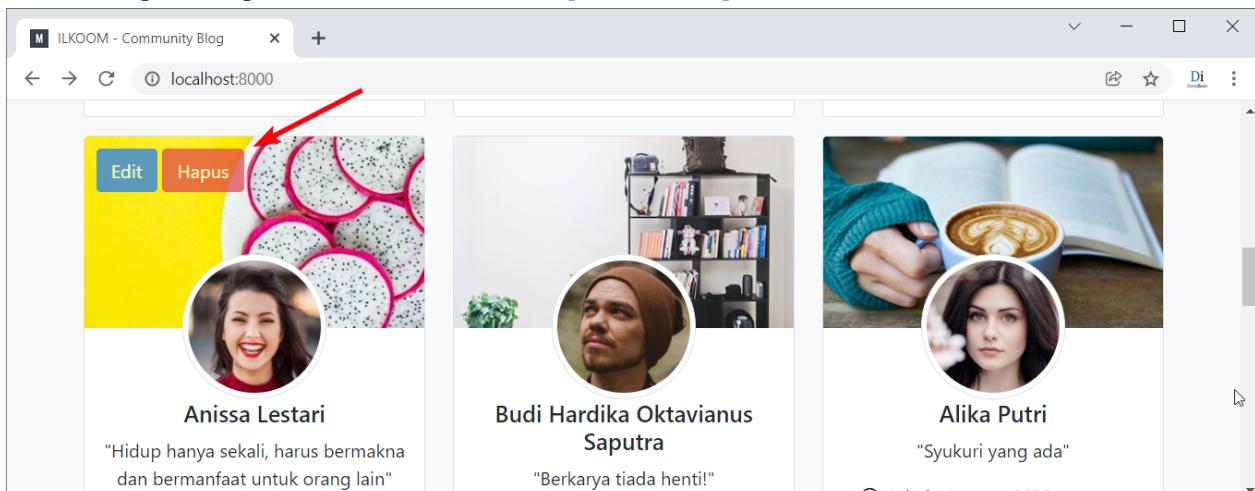
Dengan tambahan ini, maka route untuk proses edit dan delete sudah tidak bisa diakses dengan bebas.

Jika anda masih ingat, kode untuk policy ini juga sudah ada di view home.blade.php, yakni bagian yang menampilkan tombol **Edit** dan **Hapus**. Berikut potongan kode program yang dimaksud:

resources\views\home.blade.php

```
1 @can('update', $user)
2 <div class="btn-action">
3   <a href="{{ url('/users/'.$user->id.'/edit')}}" 
4     class="btn btn-primary d-inline-block">Edit</a>
5   <button class="btn btn-danger btn-hapus"
6     data-id="{{ $user->id }}" data-bs-toggle="modal"
7     data-bs-target="#DeleteModal">Hapus</button>
8 </div>
9 @endcan
```

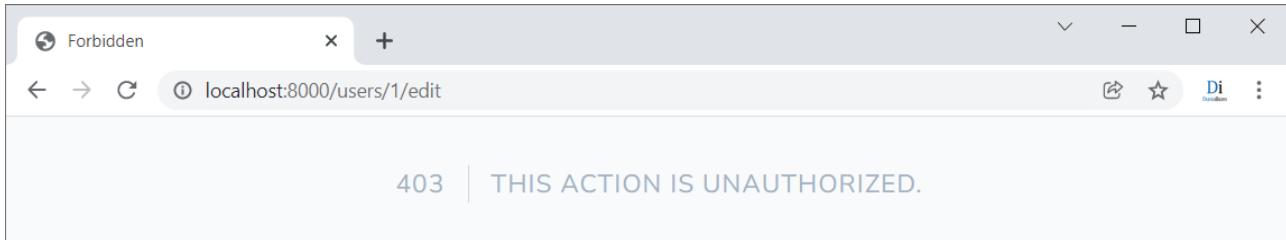
Silahkan login dengan salah satu user, lalu perhatikan profil user tersebut di halaman home:



Gambar: Tampilan tombol Edit dan Hapus

Di sisi kiri atas profil tampil 2 tombol: **Edit** dan **Hapus**. Tombol ini hanya bisa dilihat oleh profil yang sedang login saja. Misalnya jika saya login sebagai user Anissa, maka kedua tombol ini hanya tampil di kiri atas profil Anissa saja. Ketika tombol **Edit** di klik, akan tampil form edit sesuai dengan alamat URL dari user tersebut.

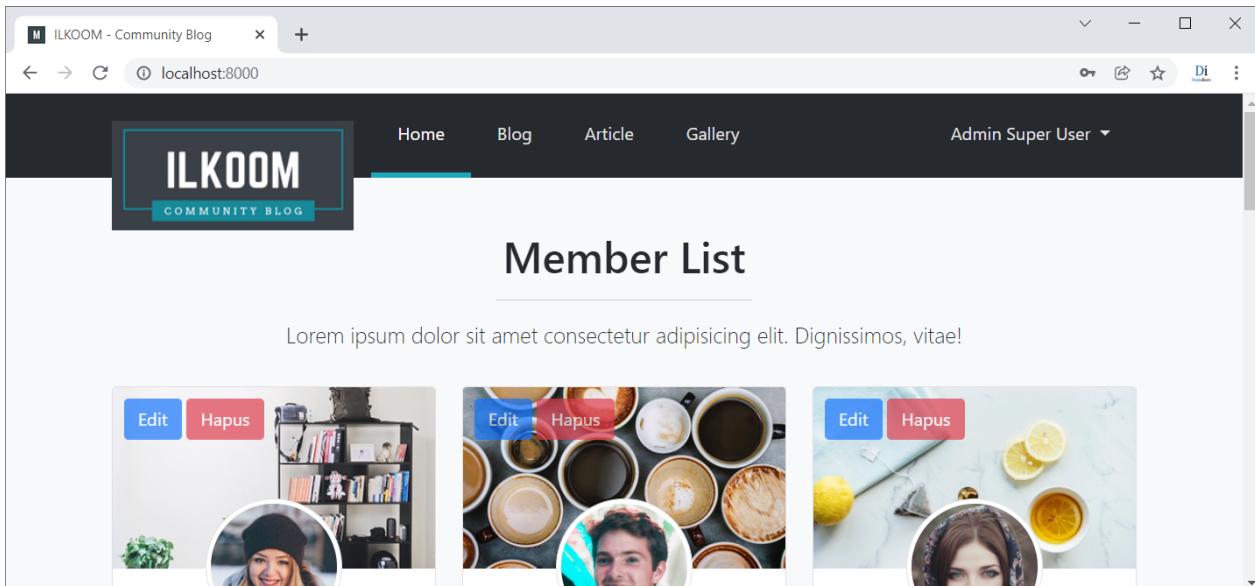
Untuk memastikan batasan hak akses sudah berjalan, mari test dengan menulis manual alamat edit untuk user lain:



Gambar: Hak akses untuk edit user lain ditolak

User Anissa yang saat ini saya pakai memiliki `id = 4`, maka ketika uji coba buka halaman <localhost:8000/users/1/edit>, hasilnya adalah '403 | This action is unauthorized'. Ini karena `id = 1` dimiliki oleh user lain.

Sekarang kita akan coba login sebagai super user. Silahkan buat user baru dengan alamat email 'admin@gmail.com'. Lalu cek kembali tampilan halaman home:

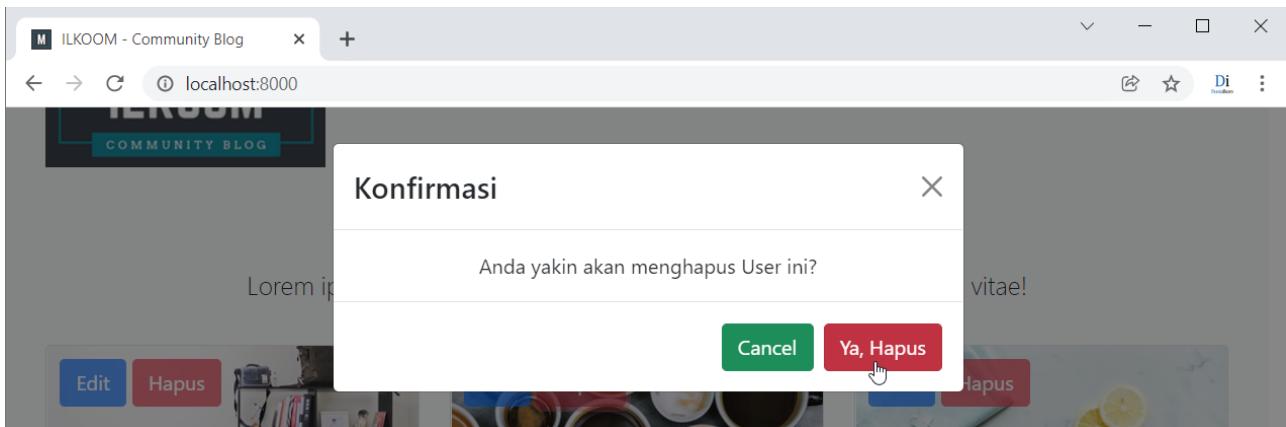


Gambar: Tampilan untuk user admin

Hasilnya, user admin bisa men-klik tombol **Edit** dan **Hapus** untuk semua user.

27.24. Tambahan JavaScript Untuk Delete

Jendela popup untuk konfirmasi delete juga sudah berjalan. Silahkan anda klik tombol **Hapus** dari salah satu user, dan akan tampil jendela popup berikut:



Gambar: Jendela Konfirmasi Hapus

Jendela ini saya buat dari komponen **Modal** Bootstrap. Secara internal, efek seperti ini butuh kode JavaScript yang sudah disediakan Bootstrap.

Ketika tombol **Cancel** di klik (atau bisa juga dengan men-klik bagian sembarang di luar kotak), jendela popup akan langsung tertutup. Ini artinya user batal melakukan proses penghapusan. Namun jika tombol **Ya, Hapus** di klik, tampil pesan error. Kenapa? Karena kita perlu sedikit bantuan kode JavaScript.

Untuk bisa memahami kenapa seperti itu, berikut saya sajikan kembali kode HTML yang dipakai untuk membuat jendela popup:

resources/views/home.blade.php

```
1  {{-- Modal untuk konfirmasi proses delete --}}
2
3  <div id="DeleteModal" class="modal fade" role="dialog">
4      <div class="modal-dialog ">
5          <!-- Modal content-->
6              <form action="" id="deleteForm" method="post">
7                  @csrf
8                  @method('DELETE')
9                  <div class="modal-content">
10                      <div class="modal-header">
11                          <h4 class="modal-title text-center">Konfirmasi</h4>
12                          <button type="button" class="btn-close" data-bs-dismiss="modal">
13                              </button>
14                      </div>
15                      <div class="modal-body">
16                          <p class="text-center mb-0">Anda yakin akan menghapus User ini?</p>
17                      </div>
18                      <div class="modal-footer">
19                          <button type="button" class="btn btn-success" data-bs-dismiss="modal">
20                              Cancel</button>
21                          <button type="submit" class="btn btn-danger" data-bs-dismiss="modal">
22                              Ya, Hapus</button>
23                      </div>
24                  </div>
25              </form>
```

```
26    </div>
27 </div>
```

Kode di atas sudah kita tulis ke view `home.blade.php` di bagian paling bawah.

Semua isi jendela popup sebenarnya berada di dalam tag `<form>` yang dimulai dari baris 6 sampai 25. Form ini saya perlukan untuk mengakses method `destroy()` di `UserController`. Menggunakan konsep **RESTfull**, kita perlu menambah perintah `@method('DELETE')` yang ada di baris 8.

Namun perhatikan isi atribut `action` dari tag `<form>`, tidak berisi nilai apapun. Inilah yang menyebabkan ketika tombol **Ya, Hapus** di klik, terjadi error. Seharusnya, di sinilah kita menulis alamat URL seperti `action="{{url('/users/'.$user->id)}}`, namun tidak bisa saya lakukan karena jendela popup ini tampil secara dinamis.

Maksudnya, ketika tombol **Hapus** di klik, akan muncul jendela konfirmasi beserta tag `<form>` untuk proses delete (seperti kode di atas). Masalahnya, **id** apa yang akan dihapus? Id user ini baru bisa diketahui pada saat user men-klik tombol **Hapus**. Kode PHP tidak bisa dipakai untuk mengambil informasi ini karena terjadi setelah halaman di load.

Atau jika kita tetap ingin menulis atribut `action="{{url('/users/'.$user->id)}}` langsung ke dalam tag `<form>`, maka saya harus membuat 1 jendela popup untuk setiap user. Jika di halaman terdapat 10 user, maka harus disiapkan 10 buah jendela popup.

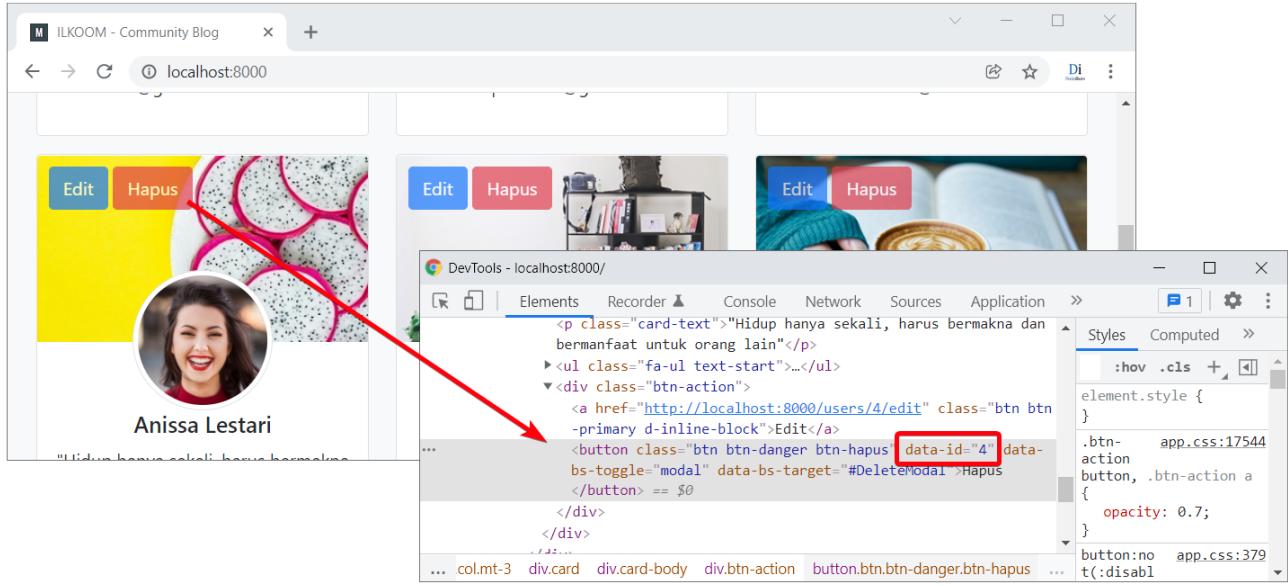
Solusi yang lebih elegan adalah dengan memakai JavaScript. Idenya, gunakan event `click` dari tombol **Hapus** untuk meng-generate nilai atribut `action` tag `<form>`. Namun kembali, dari mana informasi id user yang akan dihapus bisa diketahui?

Trik yang akan saya pakai adalah menyisipkan sebuah atribut `data-id="{{$user->id}}` ke setiap tombol **Hapus**.

Berikut saya tampilkan kembali kode program untuk meng-generate tombol **Edit** dan **Hapus**:

```
1 //...
2 @can('update', $user)
3 <div class="btn-action">
4   <a href="{{ url('/users/'.$user->id.'/edit')}}" 
5     class="btn btn-primary d-inline-block">Edit</a>
6   <button class="btn btn-danger btn-hapus"
7     data-id="{{$user->id}}" data-bs-toggle="modal"
8     data-bs-target="#DeleteModal">Hapus</button>
9 </div>
10 @endcan
```

Perhatikan di awal baris 7, terdapat tambahan `data-id="{{$user->id}}`. Inilah informasi yang akan di ambil menggunakan JavaScript untuk mengetahui id dari user yang sedang di hapus.



Gambar: Atribut data-id yang berisi id user

Dalam gambar ini, di bagian kanan saya menampilkan kode HTML yang sampai ke web browser (hasil pemrosesan PHP). Untuk user Anissa Lestari, di dalam tombol hapus tertulis atribut `data-id="4"`, yang di dapat dari kode `data-id="{{ $user->id }}`. Untuk user lain, nilainya juga akan berbeda-beda sesuai **id** setiap user.

Nilai dari atribut `data-id` inilah yang akan saya ambil menggunakan JavaScript, yang kemudian dirangkai menjadi nilai untuk atribut `action` dari tag `<form>`. Berikut kode JavaScript yang diperlukan:

```

1  /* =====
2 // Kode untuk mengisi atribut action dari tag form untuk delete user
3 // Dipakai dalam halaman home
4 =====*/
5
6 let btnHapus = document.getElementsByClassName('btn-hapus');
7 let formDelete = document.getElementById('deleteForm');
8
9 if (btnHapus) {
10   [...btnHapus].forEach(element => element.addEventListener("click", inputId))
11
12   function inputId() {
13     let idHapus = this.getAttribute('data-id');
14     formDelete.setAttribute('action', '/users/' + idHapus);
15   }
16 }
```

Bagian awal kode ini mirip seperti yang kita pakai saat membuat pilihan gambar background di halaman form. Untuk setiap element HTML yang memiliki class `.btn-hapus`, tambah event click yang akan menjalankan fungsi `inputId()`, inilah maksud dari kode di baris 10.

Pendefinisian fungsi `inputId()` ada di baris 12 – 15. Variabel `idHapus` akan berisi nilai atribut

data-id dari tombol hapus yang saat ini sedang di klik. Nilai inilah yang kemudian diisi ke dalam atribut action dari formDelete di baris 14.

Tambahan kode di atas menjadi contoh kasus perlunya memahami JavaScript meskipun kita sedang mendalami back-end. Sebuah web modern biasanya mengombinasikan 5 materi dasar web programming, yakni **HTML, CSS, PHP, MySQL** dan **JavaScript**. Jika kita tidak paham salah satunya, akan kesulitan untuk membuat fitur-fitur tertentu.

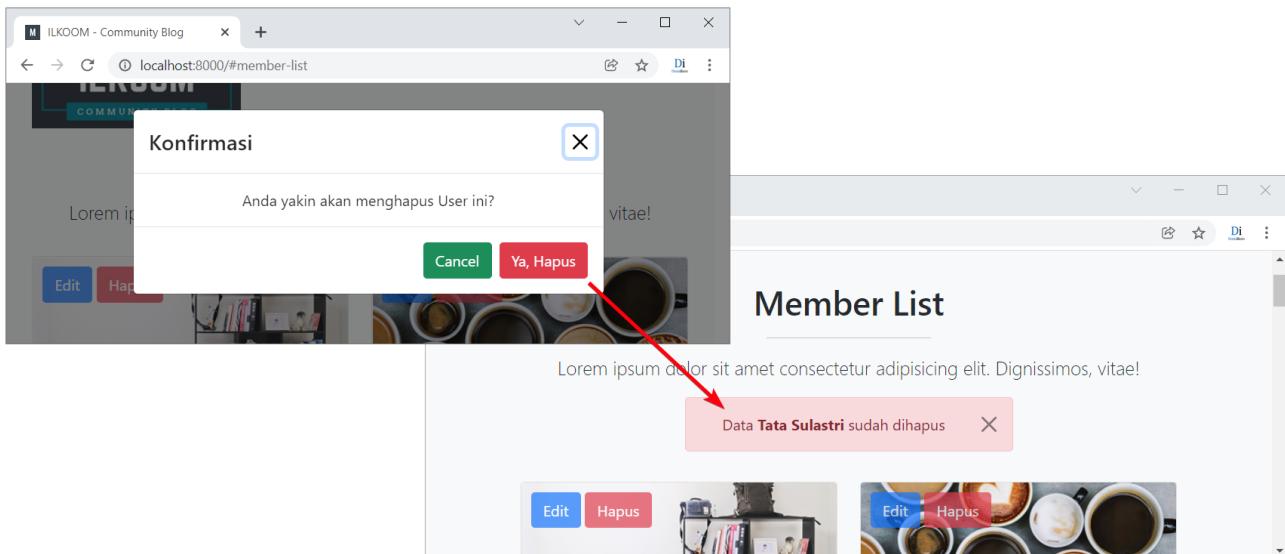
Lalu, dimana kode JS ini ditempatkan?

Bisa saja ditulis menggunakan tag <script> ke dalam view home.blade.php (sebagai internal JavaScript). Namun saya memilih untuk menempatkannya ke dalam file resources/js/my-script.js, yakni file JavaScript yang akan di compile menggunakan Laravel mix.

Silahkan tambah ke baris paling bawah file my-script.js, lalu compile ulang dengan perintah:

```
npx mix
```

Setelah itu, saatnya test hapus salah satu user. Jika anda login sebagai admin@gmail.com, maka itu bisa dipakai untuk menghapus user apa saja. Tapi jika login sebagai user biasa, hanya bisa menghapus user sendiri dan otomatis logout setelah proses delete selesai.



Gambar: Proses penghapusan user

Sesaat setelah menekan tombol **Ya, Hapus**, Laravel akan menghapus user tersebut dan kita di redirect kembali ke localhost:8000. Sesampainya di halaman home, akan tampil pesan flash 'Data <nama_user> sudah dihapus'. Ini semua di proses oleh method destroy() di UserController.

Dengan selesainya materi delete, maka penerapan CRUD di aplikasi **ILKOOM Profile Manager** sudah selesai. Silahkan anda tes ulang semua fitur yang ada, mulai dari pendaftaran user, edit user, hingga hapus user.

27.25. Modifikasi View auth\login.blade.php

Hingga saat ini halaman login masih memakai tampilan bawaan authentication. Saya ingin modifikasi sedikit dengan menambah gambar header serta perbaikan tampilan.

Silahkan buka view auth\login.blade.php, lalu modifikasi dengan kode berikut:

resources\views\auth\login.blade.php

```

1  @extends('layouts.app')
2
3  @section('content')
4
5  <!-- HEADER IMAGE -->
6  <header id="login-header" class="header-image text-white d-none d-md-block">
7      <div class="header-overlay">
8          <div class="container">
9              <div class="row">
10                 <div class="col">
11                     <h1 class="display-1">Login</h1>
12                     <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
13                         Soluta harum aperiam in facilis dicta.</p>
14                 </div>
15             </div>
16         </div>
17     </div>
18 </header>
19
20 <div class="container my-5">
21     <div class="row justify-content-center">
22         <div class="col-md-7">
23             <h1 class="text-center">Login</h1>
24             <hr>
25
26             <form method="POST" action="{{ route('login') }}">
27                 @csrf
28
29                 <div class="mb-3 row">
30                     <label for="email" class="col-md-4 col-form-label text-md-end">
31                         Email</label>
32                     <div class="col-md-6">
33                         <input id="email" type="email" class="form-control @error('email')
34                             is-invalid @enderror" name="email" value="{{ old('email') }}"
35                             required autocomplete="email" autofocus>
36                         @error('email')
37                             <span class="invalid-feedback" role="alert">
38                                 <strong>{{ $message }}</strong>
39                             </span>
40                         @enderror
41                     </div>
42                 </div>
43
44                 <div class="mb-3 row">
45                     <label for="password" class="col-md-4 col-form-label text-md-end">
```

```

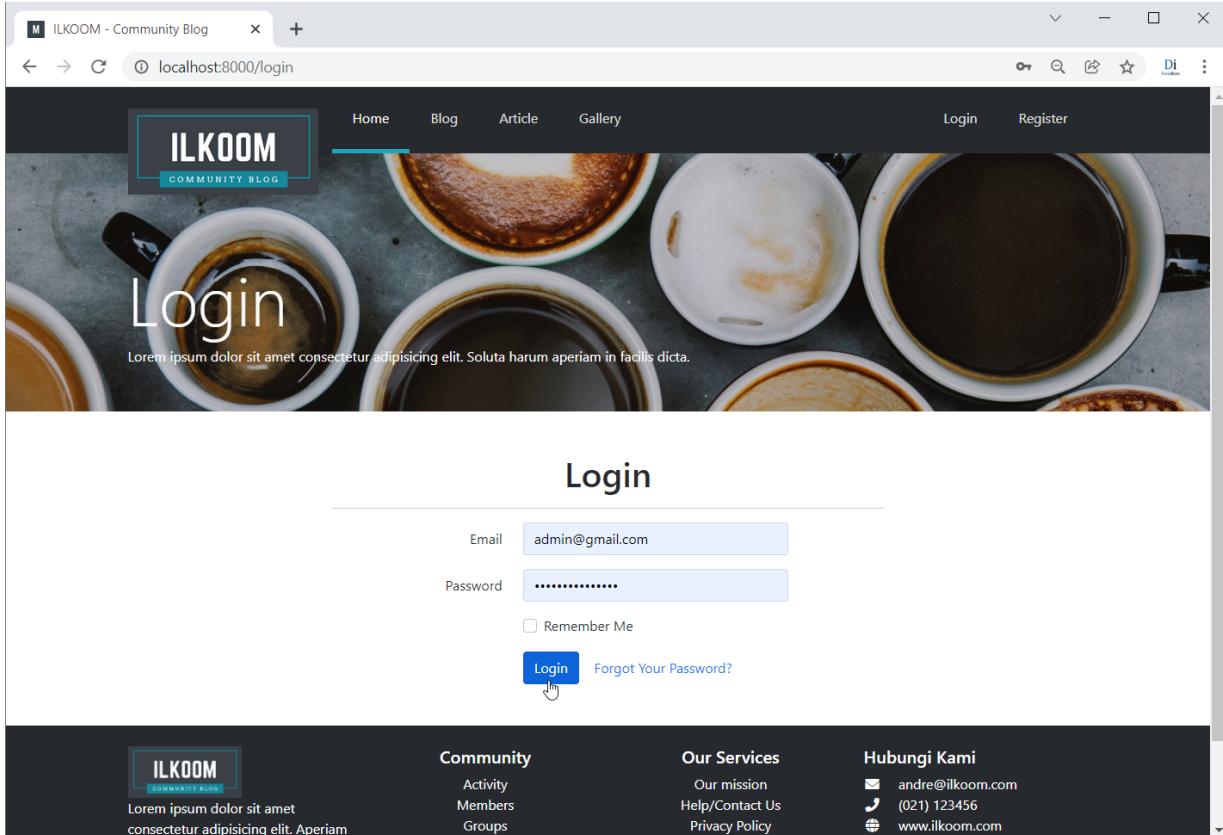
46     Password</label>
47     <div class="col-md-6">
48         <input id="password" type="password" class="form-control"
49             @error('password') is-invalid @enderror name="password"
50             required autocomplete="current-password">
51             @error('password')
52                 <span class="invalid-feedback" role="alert">
53                     <strong>{{ $message }}</strong>
54                 </span>
55             @enderror
56         </div>
57     </div>
58
59     <div class="mb-3 row">
60         <div class="col-md-6 offset-md-4">
61             <div class="form-check">
62                 <input class="form-check-input" type="checkbox" name="remember"
63                     id="remember" {{ old('remember') ? 'checked' : '' }}>
64                 <label class="form-check-label" for="remember">
65                     {{ __('Remember Me') }}
66                 </label>
67             </div>
68         </div>
69     </div>
70
71     <div class="form-group row mb-0">
72         <div class="col-md-8 offset-md-4">
73             <button type="submit" class="btn btn-primary">
74                 {{ __('Login') }}
75             </button>
76             @if (Route::has('password.request'))
77                 <a class="btn btn-link text-decoration-none"
78                     href="{{ route('password.request') }}"
79                     {{ __('Forgot Your Password?') }}>
80                 </a>
81             @endif
82         </div>
83     </div>
84     </form>
85
86     </div>
87 </div>
88 </div>
89 @endsection

```

Setelah meng-extends view `layouts.app` di baris 1, kode program di baris 6 – 18 dipakai untuk membuat gambar header serta teks 'Login'.

Kemudian di baris 20 – 88 adalah kode untuk menampilkan form login yang terdiri dari inputan `email` dan `password`. Struktur dasar form ini tetap bagian bawaan Laravel dengan sedikit tambahan class CSS Bootstrap agar tampil lebih menarik.

Berikut tampilan form login dengan perubahan ini:



Gambar: Tampilan form login

Tampilannya menjadi lebih menarik dan satu tema dengan halaman-halaman lain.

27.26. Modifikasi View home.blade.php

Modifikasi terakhir yang ingin saya buat adalah menambah gambar slider ke halaman utama, yakni view home.blade.php. Tambahan ini sebenarnya tidak wajib, sekedar mempercantik halaman saja. Dan itu pun hanya berisi kode HTML dan beberapa class Bootstrap.

Silahkan buka file home.blade.php lalu tambah kode berikut di bagian atas:

resources\views\home.blade.php

```
1 @extends('layouts.app')
2
3 @section('content')
4
5 <!-- SLIDER -->
6 <header id="main-slide">
7 <div id="mySlide" class="carousel slide carousel-fade" data-bs-ride="carousel">
8   <ol class="carousel-indicators">
9     <li data-bs-target="#mySlide" data-bs-slide-to="0" class="active"></li>
10    <li data-bs-target="#mySlide" data-bs-slide-to="1"></li>
11    <li data-bs-target="#mySlide" data-bs-slide-to="2"></li>
12    <li data-bs-target="#mySlide" data-bs-slide-to="3"></li>
13  </ol>
```

```
14 <div class="carousel-inner text-white">
15   <div class="carousel-item active" id="slide1">
16     <div class="container">
17       <div class="d-none d-md-block">
18         <h1 class="display-1 bg-info px-4 pb-2 d-inline-block">
19           Get<strong> Inspired</strong>
20         </h1>
21         <br>
22         <p class="bg-dark px-2 pb-1 d-inline-block">Lorem ipsum dolor, sit
23           amet consectetur adipisicing elit. Aut cumque molestias asperiores
24           ipsam officiis? Doloremque.</p>
25       </div>
26     </div>
27   </div>
28   <div class="carousel-item" id="slide2">
29     <div class="container">
30       <div class="d-none d-md-block text-end">
31         <h1 class="display-1 bg-dark px-4 pb-2 d-inline-block">
32           Take<strong> Action</strong>
33         </h1>
34         <br>
35         <p class="bg-info px-2 pb-1 d-inline-block">Lorem ipsum dolor, sit
36           amet consectetur adipisicing elit. Aut cumque molestias asperiores
37           ipsam officiis? Doloremque.</p>
38       </div>
39     </div>
40   </div>
41   <div class="carousel-item" id="slide3">
42     <div class="container">
43       <div class="d-none d-md-block">
44         <h1 class="display-1 bg-info px-4 pb-2 d-inline-block">
45           Be<strong> Social</strong>
46         </h1>
47         <br>
48         <p class="bg-dark px-2 pb-1 d-inline-block">Lorem ipsum dolor, sit
49           amet consectetur adipisicing elit. Aut cumque molestias asperiores
50           ipsam officiis? Doloremque.</p>
51       </div>
52     </div>
53   </div>
54   <div class="carousel-item" id="slide4">
55     <div class="container">
56       <div class="d-none d-md-block text-end">
57         <h1 class="display-1 bg-dark px-4 pb-2 d-inline-block">
58           Find<strong> Stories</strong>
59         </h1>
60         <br>
61         <p class="bg-info px-2 pb-1 d-inline-block">Lorem ipsum dolor, sit
62           amet consectetur adipisicing elit. Aut cumque molestias asperiores
63           ipsam officiis? Doloremque.</p>
64       </div>
65     </div>
66   </div>
67 </div>
68 <a class="carousel-control-prev" href="#mySlide" data-bs-slide="prev">
```

Case Study: ILKOOM Profile Manager

```
69      <span class="carousel-control-prev-icon"></span>
70      <span class="sr-only">Previous</span>
71  </a>
72  <a class="carousel-control-next" href="#mySlide" data-bs-slide="next">
73      <span class="carousel-control-next-icon"></span>
74      <span class="sr-only">Next</span>
75  </a>
76 </div>
77 </header>
78
79 //.... sisa kode view home.blade.php di sini ...
```

The screenshot shows the ILKOOM Profile Manager homepage. The top features a large banner image of a city street with buildings and cars, overlaid with a dark box containing the text "Take Action" and a smaller box with placeholder text "Lorem ipsum dolor, sit amet consectetur adipisicing elit. Aut cumque molestias asperiores ipsam officiis? Doloremque.". Below the banner is a navigation bar with links for Home, Blog, Article, and Gallery. A dropdown menu for Zahra Adinda Shanum is visible. The main content area is titled "Member List" and displays a grid of six member profiles. Each profile card includes a circular profile picture, the member's name, a quote, their joining date, and a list of their roles and contact information. At the bottom, there is a footer section with links for ILKOOM, Community, Our Services, and Contact Information, along with social media icons.

Gambar: Tampilan halaman utama ILKOOM Profile Manager

Dan... inilah tampilan akhir dari studi kasus **ILKOOM Profile Manager**. Design tampilan memang bukan bagian dari materi buku **Laravel Uncover** ini, jadi jika anda masih asing dengan kode-kode Bootstrap yang saya pakai, itu bisa dimaklumi.

Namun jika terdapat kode Laravel yang kurang paham sepanjang pembuatan project ini, bisa luangkan waktu sejenak untuk membaca kembali penjelasan yang ada, atau jika perlu pelajari ulang bab yang membahas materi tersebut.

Dalam bab terakhir buku Laravel Uncover ini kita telah membuat studi kasus yang cukup menantang. Ini merupakan sebuah aplikasi **CRUD + File Upload + Authentication**. Hampir semua materi yang dipelajari sejak awal buku juga terpakai dalam project ini.

Anda bisa explore dan modifikasi project ini lebih lanjut, misalnya menambah menu gallery untuk menampilkan semua gambar yang di upload user, mengizinkan user membuat artikel-artikel sendiri (ini akan butuh tabel baru), atau tentu saja juga bisa membuat project lain dari awal.

Practice make perfect, sehingga saya sangat sarankan anda untuk coba membuat project sendiri dari awal, tidak perlu yang rumit dulu, cukup yang sederhana saja. Jadikan materi di buku ini sebagai panduan. Jika ada kendala, bisa coba googling untuk mencari tau apa penyebabnya.

Ketika membuat project ini, saya juga butuh waktu yang tidak sebentar (sekitar 1 minggu). Itu pun tidak berurutan seperti ini, berulang kali saya bolak-balik dari view ke controller untuk memastikan semuanya sudah sesuai. Jika ada kode yang tidak jalan, fungsi `dump()` dan `dd()` sangat membantu.

Nyaris tidak ada programmer yang bisa menulis langsung dari baris 1 sampai selesai tanpa error dan tidak butuh modifikasi. Tipsnya, selalu uji coba setiap menambah fitur baru, jangan tunggu sampai kodenya menjadi panjang.

Misal, ketika membuat kode untuk **create** user, pastikan validasi sudah jalan semua. Test semua syarat yang sudah ditulis. Jika ada error, dump variabel `$request` atau `$data`, cek satu persatu apakah data dari tabel sudah sampai atau belum.

Begini juga untuk di view. Jika fitur **re-populate** tidak jalan, cek apakah fungsi `old()` berisi nilai atau tidak. Setelah itu baru lanjut ke memproses file upload, lalu test lagi apakah file memang sudah berhasil di upload atau tidak. Jika yakin, baru lanjut ke eloquent untuk input ke tabel `users`, dst.

Penutup Laravel Uncover

Akhirnya.., setelah nyaris 700 halaman (mencakup 27 bab), kita sampai di penghujung buku **Laravel Uncover**. Semoga semua materi ini bisa menjadi pondasi dasar yang kuat bagi anda untuk memahami apa itu framework Laravel dan bagaimana cara penggunaannya.

Pertanyaan utama, *apakah ini sudah selesai?*

Well... seperti yang bisa di tebak, jawaban ideal adalah: **Belum**.

Meskipun kita sudah mempelajari cukup banyak materi dasar Laravel, tapi ini baru sebagian kecil dari "dunia Laravel" yang sesungguhnya. Oleh karena itu saya berencana membuat buku lanjutan dari Laravel Uncover ini. Beberapa materi yang belum kita pelajari adalah:

- **Eloquent Relationship:** Fitur eloquent untuk mengakses 2 tabel database yang saling terhubung (ber-relasi). Nantinya akan membahas tentang *one-to-one*, *one-to-many* serta *many-to-many* relationship.
- **Accessor dan Mutator:** Membuat property dan method ke dalam Model, bisa dipakai untuk memproses data dari tabel sebelum sampai ke controller.
- **Seeding & Model Factory:** Teknik meng-generate data tabel. Ini mirip seperti migration, tapi yang akan kita generate adalah isi dari tabel tersebut, bukan lagi struktur tabelnya.
- **Email Verification:** Mengirim pesan email untuk proses verifikasi, misalnya ketika user lupa password.
- **Events & Listeners:** Menjalankan kode program tertentu jika suatu event terjadi, misalnya mengirim email ke admin jika ada user yang baru mendaftar.
- **Queues:** Mengatur urutan prioritas pemrosesan.

Karena keterbatasan tempat dan agar versi cetak dari buku ini tidak lebih tebal lagi, materi-materi di atas akan saya buat menjadi buku terpisah, yakni seri **Laravel In Depth**. Saat ini sudah tersedia [eBook Laravel In Depth #1](#) yang berfokus ke Eloquent Relationship.

Pelajari Materi Web Programming Lain

Alternatif lanjutan adalah pelajari materi web programming yang masih berhubungan dengan Laravel. Nama besar Laravel memang mendatangkan peminat tersendiri. Tidak jarang ada yang baru sadar bahwa Laravel ini adalah materi lanjutan yang perlu pemahaman web programming secara utuh.

Ibaratnya, ada yang dari HTML langsung lompat ke Laravel. Jika anda termasuk salah satunya, bisa pertimbangkan untuk mundur sejenak dan pelajari materi lain dari web programming (yang masih dirasa kurang).

Pada studi kasus **ILKOOOM Profil Manager** saya menggunakan kode HTML, CSS, PHP, JavaScript, MySQL, Bootstrap dan jQuery sekaligus. Jika ingin memahami semua kode yang ada, mau tidak mau materi-materi tersebut juga harus dipelajari.

Lowongan kerja web programmer yang mensyaratkan "menguasai Laravel" biasanya memang tidak mencantumkan skill lain seperti HTML, CSS atau PHP. Tapi bagi pemberi kerja, materi dasar seperti ini seharusnya sudah dipahami bagi yang mengaku "bisa Laravel".

Jangan Berhenti Belajar!

Programming adalah bidang yang berkembang sangat cepat. Malah Laravel sendiri memiliki jadwal update setiap 1 tahun sekali. Mau tidak mau kita juga harus *stay update* jika ingin bertahan sebagai programmer profesional. Caranya? Jangan berhenti belajar.

Jika anda punya waktu lebih (terutama bagi mahasiswa), bisa juga pertimbangkan teknologi web lain, seperti framework **Code Igniter** atau masuk ke dunia library / framework JavaScript seperti **NodeJS**, **Vue**, **React**, dan **Angular**.

Atau jika anda memutuskan untuk fokus di Laravel saja (yang juga tidak salah), silahkan lanjut ke materi yang lebih advanced dan mulai membuat berbagai studi kasus seperti Sistem Informasi, Aplikasi Inventory atau CMS sederhana.

Pelajari juga cara mengonlinekan web tersebut dan pasang sebagai portofolio (daftar hasil karya). Ini akan menjadi nilai tambah yang sangat bagus ketika melamar kerja. Hingga saat ini, lowongan kerja bagi programmer Laravel masih sangat terbuka. Besaran gaji bervariasi tergantung skill. Semakin tinggi skill anda, semakin besar peluang untuk di bayar lebih.

Akhir kata, semoga materi yang ada di buku **Laravel Uncover** ini bisa bermanfaat. Mohon maaf jika ada kata-kata atau penjelasan yang salah.

Terima kasih juga atas dukungannya dengan membeli versi asli buku Laravel Uncover (bukan dari sumber bajakan / copy-an). Donasi pembelian buku ini menjadi penyemangat saya untuk bisa terus berkarya.

Sampai jumpa di buku DuniaIlkom selanjutnya, semoga ilmu yang di dapat bisa berkah dan bermanfaat :)

Daftar Pustaka

Sepanjang penulisan buku Laravel Uncover, saya mengumpulkan bahan dari berbagai sumber. Anda bisa mengunjungi daftar pustaka ini untuk menambah pengetahuan seputar Laravel:

- Laravel Manual: <https://laravel.com/docs/9.x>
- Laravel-best-practices: <https://github.com/alexeymezenin/laravel-best-practices>
- Traversy Media: <https://www.youtube.com/user/TechGuyWeb>
- Bitfumes: <https://www.youtube.com/bitfumes>
- Coder's Tape: <https://coderstape.com>
- Laracast Forum: <https://laracasts.com/discuss>
- Stackoverflow Laravel: <https://stackoverflow.com/questions/tagged/laravel>