

Andre Pratama



Laravel In Depth #1

Panduan Belajar Laravel 8 Lanjutan

DuniaIlkom

Laravel in Depth #1

Panduan Belajar Laravel 8 Lanjutan

Andre Pratama

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

13 Oktober 2020

~ Update Log ~

- ✓ Full Release – 24 September 2020
- ✓ Update ke Laravel 8 – 13 Oktober 2020

Cover Photo by [Karly Gomez](#) on [Unsplash](#)

© 2020 DuniaIlkom

Daftar Isi

Ucapan Terima kasih.....	12
Tentang Penulis.....	13
Lisensi.....	14
Kata Pengantar.....	16
Asumsi / Pengetahuan Dasar.....	17
Contoh Kode Program.....	18
1. Pembuka Laravel In Depth.....	19
1.1. Apa Saja Yang Akan di Bahas?.....	19
1.2. Persiapan Awal.....	23
2. Tinker.....	25
2.1. Pengertian Laravel Tinker.....	25
2.2. Cara Akses Database dengan Tinker.....	30
3. Accessor dan Mutator.....	39
3.1. Persiapan Awal.....	39
3.2. Pengertian Laravel Accessor.....	42
3.3. Cara Penulisan Accessor.....	45
Custom Accessor.....	49
Mengakses Data Kolom Asal di Accessor.....	50
3.4. Pengertian Laravel Mutator.....	51
3.5. Cara Penulisan Mutator.....	52
3.6. Attribute Casting.....	55
4. Carbon.....	58
4.1. Pengertian Carbon Library.....	58
4.2. Carbon Instantiation.....	59
Instansiasi Untuk Tanggal dan Waktu Tertentu.....	62
Instansiasi Relatif ke Hari Ini.....	63
4.3. Getter.....	64

4.4. Setter.....	68
4.5. Addition dan Subtraction.....	70
4.6. Comparison.....	71
4.7. Difference.....	74
4.8. Modifier.....	75
4.9. Copying Object.....	76
4.10. Timezone.....	77
4.11. Localization.....	79
4.12. Mengatur Timezone dan Localization dari Laravel.....	81
4.13. Carbon dan Eloquent.....	82
5. Scope.....	87
5.1. Pengertian Laravel Scope.....	87
5.2. Cara Penulisan Scope.....	90
5.3. Dynamic Scope.....	91
6. Faker.....	94
6.1. Pengertian Faker Library.....	94
6.2. Faker Instantiation.....	94
6.3. Faker Formatters.....	98
Base Format.....	98
Number Format.....	99
Alphanumeric Format.....	99
Word dan Sentence Format.....	101
Name Format.....	102
Address Format.....	103
Phone dan Company Format.....	104
Internet Format.....	104
Date Format.....	105
NIK Format.....	108
6.4. Faker Seed.....	109
6.5. Faker Modifiers.....	110
Unique Modifiers.....	110
Optional Modifiers.....	112
6.6. Faker dan Eloquent.....	114
7. Seeder.....	120

7.1. Pengertian Seeder.....	120
7.2. Menjalankan Seeder.....	120
7.3. Seeder dan Faker.....	125
7.4. Membuat File Seeder.....	128
7.5. Menjalankan Banyak File Seeder.....	132
7.6. Seeder dari File CSV.....	133
Menyiapkan File CSV.....	134
Download Library CSV Seeder.....	136
8. Factory.....	139
8.1. Pengertian Factory.....	139
8.2. Membuat File Factory.....	140
8.3. Menjalankan Factory.....	142
8.4. Factory (Faker) Localization.....	146
8.5. Input Factory ke Database.....	147
8.6. File UserFactory.php.....	149
8.7. Mengakses Factory dari Seeder.....	151
8.8. Mengakses Factory dari Tinker.....	152
8.9. Shortcut untuk Generate File Factory.....	153
9. Case Study: Generating Data.....	155
9.1. Data Mata Kuliah.....	155
9.2. Generating File.....	156
9.3. Isi File Migration.....	156
9.4. Isi File Factory.....	157
9.5. Isi File Seeder.....	160
9.6. Isi File Controller.....	162
9.7. Membuat View.....	163
9.8. Menambah Accessor.....	165
10. Pagination.....	168
10.1. Pengertian Pagination.....	168
10.2. Membuat Pagination.....	169
10.3. Pengaturan Pagination di Controller.....	173
10.4. Pengaturan Pagination di View.....	174
Menambah Query String (appends).....	175
Menambah Hash Fragment (fragment).....	176

Mengatur Pagination Link Window (onEachSide).....	178
Generate Template Link Pagination.....	179
Membuat Template Link Pagination.....	181
10.5. Paginator Instance Methods.....	183
11. Eloquent Lanjutan.....	187
11.1. Mengubah Nama Tabel.....	187
11.2. Menonaktifkan Timestamp.....	189
11.3. Membuat Nilai Default.....	191
11.4. Menampilkan Data Tabel.....	192
Menampilkan Eloquent Collection.....	195
Menampilkan 1 Data Eloquent.....	199
Method all() dan get().....	202
Method select().....	203
Method orderBy(), orderByDesc(), dan inRandomOrder().....	204
Method where().....	205
Method orWhere().....	209
Method whereBetween() dan whereNotBetween().....	211
Method whereIn() dan whereNotIn().....	212
Method whereNull() dan whereNotNull().....	214
Method whereDate(), whereMonth(), whereDay(), whereYear() dan whereTime().....	215
Method limit().....	216
Method skip() dan take().....	217
Method first() dan firstWhere().....	217
Method latest().....	219
Method find() dan findOrFail().....	220
Method value().....	221
Method pluck().....	222
Method exists() dan doesntExist().....	223
Method count(), max(), min() dan avg().....	224
Method firstOrCreate().....	224
Method firstOrNew().....	227
Method firstOr().....	229
Method updateOrCreate().....	230
12. Eloquent Relationship: One to One.....	231
12.1. Pengertian One to One Relationship.....	231

12.2. Persiapan Awal.....	232
Pengertian Primary Key dan Foreign Key.....	233
Generate Data Sample.....	234
12.3. Membuat Join dengan DB Facade (Raw SQL Queries).....	239
12.4. Menginstall Laravel Debugbar.....	243
12.5. Eloquent Relationship hasOne().....	245
Menampilkan Satu Gabungan Data.....	247
Menampilkan Banyak Gabungan Data.....	252
Lazy Loading vs Eager Loading.....	254
Menampilkan Batasan has() dan whereHas().....	256
Menampilkan Batasan doesntHave() dan whereDoesntHave().....	259
Menginput Data Relationship.....	261
Mengupdate Data Relationship.....	264
Menghapus Data Relationship.....	266
12.6. Eloquent Relationship belongsTo().....	272
Menampilkan Satu Gabungan Data.....	273
Menampilkan Batasan has().....	275
Menginput Data Relationship.....	276
Menghapus Data Relationship.....	280
13. Eloquent Relationship: One to Many.....	282
13.1. Pengertian One to Many Relationship.....	282
13.2. Persiapan Awal.....	283
Generate Data Sample.....	284
13.3. Membuat Join dengan DB Facade (Raw SQL Queries).....	287
13.4. Eloquent Relationship hasMany().....	290
Menampilkan Satu Gabungan Data.....	292
Menampilkan Banyak Gabungan Data.....	295
Menampilkan Batasan has() dan whereHas().....	297
Menampilkan Batasan doesntHave().....	299
Menghitung Data dengan withCount() dan loadCount().....	299
Menginput Data Relationship.....	301
Mengupdate Data Relationship.....	303
Menghapus Data Relationship.....	305
13.5. Eloquent Relationship belongsTo().....	306
Menampilkan Satu Gabungan Data.....	307
Menampilkan Batasan has() dan whereHas().....	308

Menampilkan Batasan doesntHave().....	309
Menginput Data Relationship.....	310
Mengupdate Data Relationship.....	311
Menghapus Data Relationship.....	311
Memutus Hubungan Relationship.....	312
14. Eloquent Relationship: Many to Many	316
14.1. Pengertian Many to Many Relationship.....	316
14.2. Persiapan Awal.....	317
Generate Data Sample.....	318
14.3. Eloquent Relationship belongsToMany().....	322
Menampilkan Semua Data Mahasiswa.....	323
Menginput Data Relationship.....	324
Menambah Method withTimestamps().....	325
Menginput Banyak Data Relationship.....	326
Menampilkan Data Relationship.....	327
Menghitung Data dengan withCount().....	329
Menghapus Data Relationship.....	330
Menginput Data Relationship dengan sync().....	331
Menginput Data Relationship dengan syncWithoutDetaching().....	334
Menginput dan Menghapus Data Relationship dengan toggle().....	335
Menghapus Data Relationship.....	335
14.4. Eloquent Relationship belongsToMany() (lagi).....	336
Menampilkan Semua Data Matakuliah.....	338
Menginput Data Relationship.....	338
Menampilkan Data Relationship.....	340
Menghapus Data Relationship.....	340
Menginput Data Relationship dengan sync().....	341
14.5. Mengakses Tabel Pivot.....	342
14.6. Membuat Composite Unique Key.....	343
15. Eloquent Relationship: Has One Through	345
15.1. Pengertian Has One Through Relationship.....	345
15.2. Persiapan Awal.....	346
Generate Data Sample.....	346
15.3. Pendefinisian Relationship One to One	350
15.4. Eloquent Relationship hasOneThrough().....	355

16. Eloquent Relationship: Has Many Through.....	360
16.1. Pengertian Has Many Through Relationship.....	360
16.2. Persiapan Awal.....	361
Generate Data Sample.....	361
16.3. Pendefinisian Relationship One to Many.....	365
16.4. Eloquent Relationship hasManyThrough().....	371
16.5. Haruskah Menggunakan Relationship Has Through?.....	373
17. Eloquent Relationship: One to One Polymorphic.....	375
17.1. Pengertian Polymorphic Relationship.....	375
17.2. Pengertian One to One Polymorphic Relationship.....	377
17.3. Persiapan Awal.....	378
Generate Data Sample.....	378
17.4. Eloquent Relationship morphOne() dan morphTo().....	381
Menginput Data Relationship.....	384
Menampilkan Data Relationship.....	386
Menampilkan Batasan whereHasMorph().....	388
Mengupdate Data Relationship.....	390
Menghapus Data Relationship.....	391
18. Eloquent Relationship: One to Many Polymorphic.....	393
18.1. Pengertian One to Many Polymorphic Relationship.....	393
18.2. Generate Data Sample.....	394
18.3. Eloquent Relationship morphMany() dan morphTo().....	396
Menginput Data Relationship.....	398
Menampilkan Data Relationship.....	400
Menampilkan Batasan whereHasMorph().....	403
Mengupdate Data Relationship.....	403
Menghapus Data Relationship.....	405
19. Eloquent Relationship: Many to Many Polymorphic.....	407
19.1. Pengertian Many to Many Polymorphic Relationship.....	407
19.2. Generate Data Sample.....	408
19.3. Eloquent Relationship morphToMany() & morphedByMany().....	411
Menginput Data Relationship.....	414
Menampilkan Data Relationship.....	416
Mengupdate Data Relationship.....	420

Menghapus Data Relationship.....	421
20. Sistem Informasi Universitas ILKOOM: Seed.....	425
20.1. Proses Analisa Awal.....	425
20.2. Perancangan Design Database.....	426
20.3. Membuat File Migration.....	430
20.4. Generate Tabel (migration).....	431
20.5. Generate Isi Tabel (factory dan seeder).....	433
20.6. Pendefinisian Relationship.....	443
21. Sistem Informasi Universitas ILKOOM: Read.....	446
21.1. Instalasi Laravel UI dan Laravel Mix.....	446
21.2. Persiapan File Asset.....	448
21.3. Membuat View Template (app.blade.php).....	453
21.4. Menampilkan Semua Data Jurusan.....	461
21.5. Menampilkan Semua Data Dosen.....	466
Optimisasi dengan Eager Loading.....	470
21.6. Menampilkan Semua Data Mahasiswa.....	472
21.7. Menampilkan Semua Data Mata Kuliah.....	474
21.8. Menampilkan Daftar Dosen dan Mahasiswa Jurusan.....	476
Memproses Tombol Dosen.....	477
Memproses Tombol Mahasiswa.....	479
21.9. Menampilkan Data Satu Dosen.....	480
21.10. Menampilkan Data Satu Mahasiswa.....	483
21.11. Menampilkan Data Satu Matakuliah.....	486
22. SweetAlert.....	489
22.1. Pengertian SweetAlert.....	489
22.2. Download File SweetAlert2.....	489
22.3. Cara Penggunaan SweetAlert2.....	490
22.4. Pilihan Icon SweetAlert2.....	492
22.5. Fitur SweetAlert2.....	494
22.6. Instalasi realrashid/sweetalert.....	503
22.7. Realrashid/sweetalert Facade.....	504
22.8. Realrashid/sweetalert Helper Function.....	507
22.9. Realrashid/sweetalert Helper Method.....	508
22.10. Realrashid/sweetalert Flash Session.....	511

22.11. Realrashid/sweet-alert Validation Alert.....	514
22.12. Realrashid/sweet-alert Delete Confirmation.....	520
23. Sistem Informasi Universitas ILKOOOM: Create.....	528
23.1. Menginstall Realrashid/sweet-alert.....	528
23.2. Menambah Data Jurusan.....	529
Instalasi Localization.....	538
23.3. Menambah Data Dosen.....	539
23.4. Menambah Data Mahasiswa.....	547
23.5. Menambah Data Matakuliah.....	555
23.6. Menambah Matakuliah untuk Dosen.....	562
23.7. Mengambil Matakuliah untuk Mahasiswa.....	567
23.8. Mendaftarkan Mahasiswa untuk Matakuliah.....	578
24. Sistem Informasi Universitas ILKOOOM: Update.....	585
24.1. Mengedit Data Jurusan.....	585
24.2. Mengedit Data Dosen.....	589
24.3. Mengedit Data Mahasiswa.....	595
24.4. Mengedit Data Matakuliah.....	602
25. Sistem Informasi Universitas ILKOOOM: Delete.....	609
25.1. Menghapus Data Jurusan.....	609
25.2. Menghapus Data Dosen.....	615
25.3. Menghapus Data Mahasiswa.....	618
25.4. Menghapus Data Matakuliah.....	619
25.5. Menambah Tombol Edit dan Hapus ke Halaman Show.....	621
26. Sistem Informasi Universitas ILKOOOM: Final.....	626
26.1. Membuat Halaman Search.....	626
26.2. Pembatasan Route dengan Middleware.....	634
26.3. Redirect Setelah Login.....	637
Penutup.....	640
Daftar Pustaka.....	641

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 ke sepuluh DuniaIlkom: **Laravel In Depth #1**.

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, OOP PHP dan Laravel Uncover. Karena dari *feedback* dan dukungan rekan-rekan lah saya bisa lanjut menulis buku ini. Terima kasih :)

Padang Panjang, 2020

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 menafkahsi 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.

Kata Pengantar

Ini adalah buku pertama dari seri **Laravel In Depth** DuniaIlkom. Kata *in depth* bermakna "secara mendalam", yang berarti buku ini akan mengupas secara mendalam materi lanjutan Framework Laravel.

Luasnya materi Laravel nyaris mustahil bisa dibahas dalam 1 buku saja. Buku **Laravel Uncover** yang sudah ada di DuniaIlkom saat ini saya tujuhan untuk perkenalan dasar framework Laravel. Sedangkan materi lanjutan yang lebih advanced akan menjadi jatah seri *Laravel In Depth*. Di sebut sebagai "seri" karena rencananya akan ada beberapa buku Laravel In Depth dengan penekanan materi berbeda.

Buku **Laravel In Depth #1** kali ini lebih banyak membahas materi yang berhubungan dengan database. Fokus utama ada pada *eloquent relationship*, yakni materi Laravel yang berhubungan dengan pengaksesan banyak tabel (lebih dari satu).

Eloquent relationship akan kita bahas dengan cukup detail dan mencakup 8 bab, yakni *one to one*, *one to many*, *many to many*, *has one trough*, *has many trough*, *one to one polymorphic*, *one to many polymorphic*, serta *many to many polymorphic*.

Selain itu juga ada materi Laravel lain yang masih berhubungan dengan pengolahan data. Diantaranya: *tinker*, *accessor*, *mutator*, *carbon*, *scope*, *faker*, *seeder*, *factory*, *pagination*, hingga library tambahan *sweetalert*.

Di akhir buku terdapat studi kasus yang cukup besar, yakni sebuah aplikasi CRUD yang melibatkan gabungan 4 buah tabel.

Akhir kata, semoga buku **Laravel In Depth #1** ini bisa memandu rekan-rekan untuk membuat project yang butuh banyak tabel di Laravel. Sampai jumpa di bab terakhir :)

Asumsi / Pengetahuan Dasar

Buku ini berisi materi lanjutan Laravel, sehingga saya berasumsi rekan-rekan sudah paham konsep dasar Laravel seperti cara instalasi, *route*, *view*, *blade*, *controller*, *eloquent*, hingga RESTfull CRUD. Atau lebih tepatnya ini adalah buku lanjutan dari [Laravel Uncover](#).

Namun tidak menutup kemungkinan buku ini juga bisa dipelajari tanpa harus dari Laravel Uncover, selama sudah cukup familiar dengan penggunaan dasar Laravel.

Materi utama buku ini sangat berkaitan dengan database, sehingga pemahaman query MySQL sangat dibutuhkan, terutama konsep dasar seperti *primary key*, *foreign key*, *referential integrity* hingga efek perintah `ON DELETE CASCADE`.

Jika diperlukan, di DuniaIlkom juga tersedia buku [MySQL Uncover](#) yang membahas mendalam tentang penggunaan aplikasi MySQL.

Contoh Kode Program

Seluruh kode program dalam buku **Laravel In Depth #1** bisa di download dari folder sharing Google Drive yang di kirim pada saat pembelian: **belajar_laravel_in_depth_1.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\`, dst. 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_in_depth_1.zip**.

1. Pembuka Laravel In Depth

Dalam bab pertama buku **Laravel In Depth #1** ini saya ingin membahas ringkasan singkat dari keseluruhan isi buku.

Laravel in depth adalah seri buku materi advanced Laravel, atau istilah keren dari seri Laravel Lanjutan. Di sebut "seri" karena rencananya akan ada beberapa buku *Laravel in depth* dengan berbagai fokus bahasan. Pembagian ini diperlukan karena materi Laravel sendiri sangat luas.

1.1. Apa Saja Yang Akan di Bahas?

Tema besar dari buku adalah membahas fitur-fitur Laravel terkait pengolahan data dari dan ke database. Di luar itu, cukup banyak materi lain yang akan kita pelajari.

Pembahasan dimulai dari **Tinker**, sebuah aplikasi berbentuk perintah cmd untuk mengakses fungsi Laravel dengan cepat. Menggunakan tinker, kita bisa terhubung ke database serta melakukan manipulasi data tanpa perlu membuat kode program di Controller.

Setelah itu akan masuk ke materi tentang **Accessor** dan **Mutator**, yakni method untuk modifikasi data yang dibaca dan disimpan ke database.

Ketika mengolah data, kita akan sering bertemu tipe data tanggal (*date* dan *time*). Manipulasi data tanggal memang cukup kompleks, apalagi jika sudah menyangkut perbedaan jumlah hari dalam satu bulan. Untungnya, Laravel menyertakan library **Carbon** untuk masalah ini. Materi tentang carbon akan dibahas dalam 1 bab tersendiri.

Bahasan berikutnya adalah tentang **Model Scope**, yakni sebuah cara untuk menyimpan query atau perintah eloquent yang sering dipakai ke dalam Model.

Di buku Laravel Uncover, kita sudah mempelajari tentang migration. Migration dipakai untuk menggenerate struktur tabel di dalam Laravel. Tapi bagaimana dengan mengisi data awal atau data sample untuk tabel tersebut? Inilah yang akan dibahas pada bab 3 bab terpisah tentang **Faker**, **Seeder** serta **Factory**.

Data yang tersimpan di tabel bisa saja berisi ratusan hingga ribuan baris (bahkan lebih).

Menampilkan 1000 data dalam 1 halaman sekaligus tidaklah efisien. Cara yang umum dipakai adalah dengan memecah data tersebut ke dalam beberapa halaman. Teknik ini disebut sebagai **Pagination**, yang fiturnya juga sudah tersedia di Laravel.

Setelah beberapa materi awal tersebut, kita akan masuk ke bahasan utama dari buku ini, yaitu

Eloquent Relationship. Eloquent relationship adalah sebutan dari penerapan konsep eloquent untuk mengakses tabel yang saling terhubung (saling ber-relasi). *Relationship* di sini merujuk ke hubungan antar tabel di database.

Jika anda sudah pernah belajar teori database atau materi tentang MySQL, tentu tidak asing dengan query `JOIN`. Query `JOIN` di pakai untuk menampilkan hasil gabungan dari dua atau banyak tabel di database. Eloquent relationship berupaya menyederhanakan penulisan query `JOIN` ini.

Dalam konsep teori database, hubungan antar tabel bisa dibagi menjadi 3 kategori besar:

- ◆ *One to one relationship*
- ◆ *One to many relationship*
- ◆ *Many to many relationship*

Sebagai contoh, hubungan dari tabel `mahasiswa` dengan tabel `nilai` adalah **one to one relationship**. Maksudnya, satu mahasiswa hanya bisa memiliki satu nilai, begitu juga satu nilai adalah milik dari satu mahasiswa. Dengan eloquent relationship, kita bisa membuat logika hubungan seperti ini dan menampilkan data tanpa perlu menulis query `JOIN` secara manual.

Contoh untuk hubungan **one to many relationship** adalah antara tabel `jurusan` dan `mahasiswa`. Sebuah jurusan bisa memiliki banyak mahasiswa, tapi satu id mahasiswa hanya bisa terdaftar di satu jurusan saja.

Sedangkan contoh hubungan **many to many relationship** adalah tabel `mahasiswa` dan `matakuliah`. Satu mahasiswa bisa mengambil banyak mata kuliah, dan setiap mata kuliah bisa diambil oleh banyak mahasiswa. Hubungan inilah yang kita bahas di eloquent relationship.

Tidak hanya itu, Laravel juga mendukung 5 variasi *relationship* lain:

- ◆ *Has one trough relationship*
- ◆ *Has many trough relationship*
- ◆ *One to one polymorphic relationship*
- ◆ *One to many polymorphic relationship*
- ◆ *Many to many polymorphic relationship*

Total, kita akan mempelajari 8 jenis relationship yang disediakan Eloquent!

Sekilas materi ini tampak rumit dan menakutkan, akan tetapi inilah yang akan kita hadapi pada saat membuat aplikasi web sebenarnya. Cukup jarang aplikasi "real world" yang hanya butuh satu tabel saja. Biasanya perlu dua, tiga bahkan puluhan tabel yang saling terhubung satu sama lain.

Agar bisa memahami konsep relationship, sebenarnya perlu pemahaman terkait teori database. Paling ideal, anda sudah pernah menggunakan query `JOIN` di MySQL serta paham istilah-istilah dasar database seperti *primary key*, *foreign key*, *referential integrity* dan *cascade*.

Namun nantinya saya juga akan bahas secara singkat tentang konsep ini.

Sebagai penerapan dari eloquent relationship, di akhir buku kita akan membuat sebuah studi kasus sederhana (well... sebenarnya tidak terlalu sederhana). Berikut tampilan final project tersebut:

The screenshot shows the homepage of the ILKOOOM University Information System. At the top, there is a navigation bar with the university logo, "ILKOOOM UNIVERSITY", and links for JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and LOGIN. Below the navigation bar, the main title "Sistem Informasi Universitas ILKOOOM" is displayed. The page is divided into six boxes, each representing an academic program:

- Ekonomi Manajemen**: Kepala Jurusan: **Maya Susanti M.M**, Total Mahasiswa: 0 (daya tampung 300). Buttons: Dosen (blue), Mahasiswa (gold).
- Sistem Informasi**: Kepala Jurusan: **Dr. Marsito Purnawati**, Total Mahasiswa: 16 (daya tampung 80). Buttons: Dosen (blue), Mahasiswa (gold).
- Teknik Industri**: Kepala Jurusan: **Alex Situmorang M.T**, Total Mahasiswa: 0 (daya tampung 120). Buttons: Dosen (blue), Mahasiswa (gold).
- Ilmu Komputer**: Kepala Jurusan: **Dr. Mustofa Simanjuntak**, Total Mahasiswa: 19 (daya tampung 60). Buttons: Dosen (blue), Mahasiswa (gold).
- Teknik Elektro**: Kepala Jurusan: **Rudi Perdana M.T**, Total Mahasiswa: 0 (daya tampung 70). Buttons: Dosen (blue), Mahasiswa (gold).
- Teknik Informatika**: Kepala Jurusan: **Dr. Ika Puspasari**, Total Mahasiswa: 15 (daya tampung 50). Buttons: Dosen (blue), Mahasiswa (gold).

The footer section of the ILKOOOM University Information System. It includes the university logo, a copyright notice ("© ILKOOOM 2020"), and a placeholder text ("Lorem ipsum dolor sit amet veniam consecetur adipisicing elit. Aperiam cumque, esse modi maxime."). On the right, there are links for "Information" (Jurusan, Dosen, Mahasiswa, Mata Kuliah) and "Our Services" (Register, Help/Contact Us, Privacy Policy, Terms & Conditions). There is also a "Hubungi Kami" section with contact information: email (info@ilkoom.ac.id), phone ((021) 123456), and website (www.ilkoom.ac.id). Social media icons for Facebook, Twitter, Instagram, Google+, and YouTube are also present.

Gambar: Halaman home dari Sistem Informasi Universitas ILKOOOM

The screenshot shows a table listing 10 faculty members (Dosen) from Universitas ILKOOM. The columns are: #, NID, Name, Jurusan Dosen, and Action (Edit/Hapus). The data is as follows:

#	NID	Nama Dosen	Jurusan Dosen	Action
1	99754972	Anissa Lestari	Teknik Industri	Edit Hapus
2	99957158	Cici Farida M.Sc	Sistem Informasi	Edit Hapus
3	99198789	Farah Purwanti M.Si	Ilmu Komputer	Edit Hapus
4	99488567	Galiono Safitri M.T	Sistem Informasi	Edit Hapus
5	99521834	Jais Uwais M.Kom	Sistem Informasi	Edit Hapus
6	99222608	Karja Putra M.Sc	Sistem Informasi	Edit Hapus
7	99840302	Karja Salahudin M.Si	Teknik Informatika	Edit Hapus
8	99546403	Purwanto Nainggolan M.T	Teknik Informatika	Edit Hapus
9	99066212	Rini Usamah M.Kom	Teknik Informatika	Edit Hapus
10	99318639	Shakila Padmasari M.Sc	Ilmu Komputer	Edit Hapus

Tambah Dosen

1 2 >

Gambar: Daftar Dosen Universitas ILKOOOM

The screenshot shows a modal dialog box confirming student enrollment. The message says: "Terdapat 5 mahasiswa yang mengambil Dasar Rekayasa Perangkat Lunak". There is an "OK" button at the bottom.

Informasi Mata Kuliah

Daftar Mahasiswa:

- 1. Ciaobella Widastuti (10023242)
- 2. Endra Suryatni (10025005)
- 3. Kasiyah Hastuti (10918942)
- 4. Tira Kurniawan (10789452)
- 5. Zamira Dongoran (10054918)

Daftarkan Mahasiswa

Gambar: Konfirmasi Pendaftaran Mahasiswa ke Mata Kuliah

Dalam aplikasi **Sistem Informasi Universitas ILKOOM**, terdapat 4 tabel yang saling ber-relasi satu sama lain, yakni tabel jurusan, dosen, mahasiswa dan matakuliah. Beberapa batasan logika dari tabel-tabel ini adalah:

1. Satu jurusan memiliki banyak dosen, mahasiswa dan mata kuliah. Namun ada batasan jumlah mahasiswa di satu jurusan.
2. Satu dosen terdaftar di satu jurusan, tapi bisa mengajar mata kuliah untuk jurusan lain. Dosen bisa mengajar lebih dari satu mata kuliah.
3. Satu mahasiswa terdaftar di satu jurusan (selama masih ada kuota / jurusan belum penuh). Satu mahasiswa bisa mengambil banyak mata kuliah sepanjang dalam jurusan yang sama.
4. Satu mata kuliah terdaftar di satu jurusan. Mata kuliah yang sama bisa diajar oleh dosen yang berbeda.

Syarat batasan inilah yang akan kita pelajari. Selain menerapkan konsep eloquent relationship, studi kasus ini juga memperkuat konsep dasar Laravel terutama pembuatan sistem CRUD (Create - Read - Update - Delete).

Untuk membuat CRUD pada satu tabel saja perlu 4 file view, sehingga dalam project ini akan butuh 16 view! belum termasuk view untuk halaman lain seperti pencarian.

Sangat menantang? Betul. Tapi sekaligus sangat menarik karena kita bisa memahami Laravel dengan lebih baik lagi. Latihan ini juga sebagai persiapan untuk membuat aplikasi web yang sebenarnya.

1.2. Persiapan Awal

Seri **Laravel In Depth** adalah buku lanjutan dari **Laravel Uncover**. Oleh karena itu saya berasumsi rekan-rekan sudah memahami semua materi yang ada di dalam buku tersebut, termasuk cara menginstall dan menjalankan aplikasi Laravel.

Di hampir setiap awal bab, kita akan mulai dari installer baru **Laravel 8** (kecuali dinyatakan lain). Berikut perintah instalasi yang diperlukan:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Nama folder tidak harus "laravel01", bisa juga memakai nama lain sesuai keinginan. Agar lebih rapi, folder ini bisa disimpan ke c:\xampp\htdocs, namun juga bukan hal wajib.

Untuk menjalankan Laravel, saya memakai web server bawaan PHP dengan perintah `php artisan serve` sehingga aplikasi Laravel bisa diakses dari alamat `localhost:8000`.

Untuk database, saya menggunakan MySQL Server bawaan XAMPP. Kemudian membuat satu database bernama `laravel` agar sesuai dengan pengaturan default dari Laravel. Nama

database ini juga bisa diganti, selama pengaturan DB_DATABASE di file .env juga disesuaikan. Untuk teks editor saya menggunakan VS Code serta menampilkan hasilnya di web browser Google Chrome.

Semua cara ini tidak mengikat. Jika anda sudah cukup familiar dengan Laravel, tidak masalah ingin memakai sistem yang berbeda seperti menggunakan Laragon untuk menjalankan Laravel.

Baik, persiapan awal sudah selesai. Kita akan masuk ke materi pembahasan dengan mempelajari cara kerja **Tinker** pada bab berikutnya.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

2. Tinker

Dalam bab ini kita akan berkenalan dengan **Tinker**, sebuah aplikasi bawaan Laravel yang sering dipakai untuk mengakses database dari cmd.

Kita akan mulai dari installer baru Laravel 8. Berikut perintah yang bisa digunakan:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Selain itu pastikan di dalam MySQL sudah terdapat database bernama 'laravel' untuk proses migration nantinya.

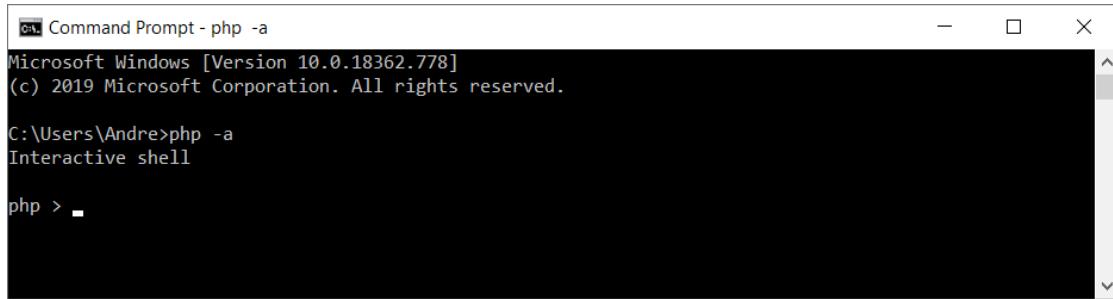
2.1. Pengertian Laravel Tinker

Tinker adalah fitur Laravel yang berbentuk aplikasi cmd, atau istilah kerennya: "*interactive shell*".

Dengan tinker, kita bisa menjalankan kode Laravel secara langsung tanpa perlu menulisnya di route maupun controller terlebih dahulu. Namun penggunaan paling umum dari tinker adalah untuk mengakses data dari database.

Secara teknis, tinker termasuk kategori aplikasi **REPL** (singkatan dari Read–Eval–Print Loop), yakni aplikasi cmd interaktif yang akan mengeksekusi perintah secara baris per baris. Hasil dari perintah ini bisa langsung terlihat pada saat itu juga. Laravel tinker sendiri dikembangkan dari library [PsySH](#).

Sebenarnya PHP juga sudah memiliki *interactive shell* bawaan. Sebelum mencoba Tinker, mari lihat sekilas bagaimana cara penggunaan *interactive shell* bawaan PHP. Silahkan buka aplikasi cmd, lalu langsung ketik perintah `php -a`:

A screenshot of a Windows Command Prompt window titled "Command Prompt - php -a". The window shows the following text:

```
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Andre>php -a
Interactive shell

php >
```

The window has standard Windows-style window controls (minimize, maximize, close) at the top right.

Gambar: Tampilan PHP interactive shell

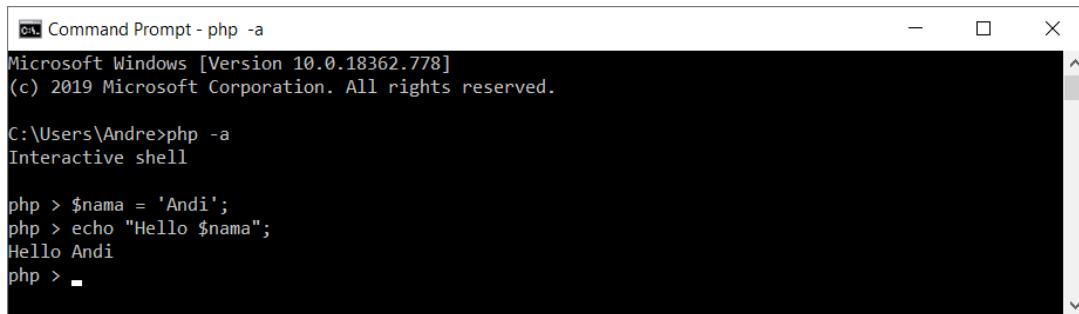
Sesaat setelah menekan tombol Enter, akan terlihat judul "Interactive shell" dan cursor cmd berubah menjadi `php >`. Ini artinya kita sudah masuk ke PHP interactive shell.

Jika pada saat mengetik perintah `php -a` keluar pesan error '`php` is not recognized as an internal or external command, operable program or batch file, maka artinya file `php.exe` belum bisa diakses secara global.

Solusinya, bisa pakai perintah `c:\xampp\php\php.exe -a` atau tambah file `c:\xampp\php\php.exe` ke PATH system variable.

Di dalam PHP interactive shell kita bisa menjalankan berbagai perintah PHP secara langsung. Sebagai contoh, silahkan ketik 2 perintah berikut dan akhiri setiap baris dengan tombol Enter:

```
$nama = 'Andi';
echo "Hello $nama";
```

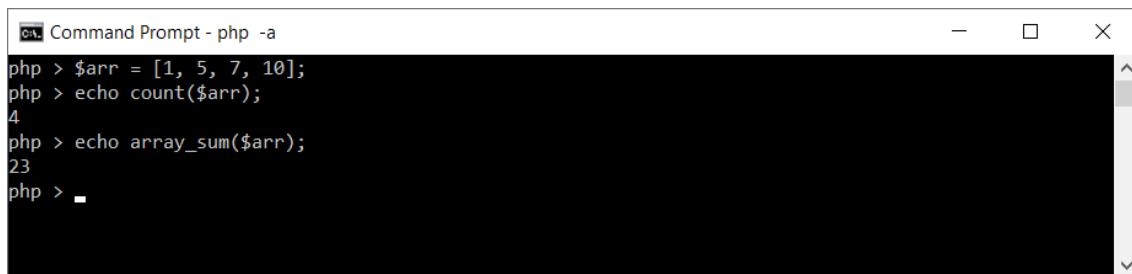


Gambar: Mencoba PHP interactive shell

Ini merupakan perintah sederhana PHP, yakni mengisi string 'Andi' ke dalam variabel `$nama` di baris pertama, lalu di `echo` di baris kedua. Hasil dari perintah `echo` bisa langsung terlihat di dalam cmd.

Kita juga bisa menjalankan berbagai function bawaan PHP seperti contoh berikut:

```
$arr = [1, 5, 7, 10];
echo count($arr);
echo array_sum($arr);
```



Gambar: Menjalankan function di PHP interactive shell

Tinker

Kali ini saya membuat array `$arr` dan mengisinya dengan beberapa element. Kemudian menjalankan perintah `echo` untuk mencari jumlah element dengan function `count()`, serta mencari nilai rata-rata dengan function `array_sum()`. Sama seperti sebelumnya, begitu tombol Enter di tekan, hasil perintah `echo` langsung terlihat.

Untuk keluar dari PHP *interactive shell*, ketik perintah `exit` atau langsung tutup aplikasi cmd.



```
Command Prompt
php > echo count($arr);
4
php > echo array_sum($arr);
23
php > exit

C:\Users\Andree>
```

Gambar: Cara keluar dari PHP interactive shell

Aplikasi **Tinker** dari Laravel juga mirip seperti ini, plus berbagai fitur tambahan. Untuk menjalankan Tinker, silahkan masuk ke folder tempat Laravel terinstall, lalu ketik perintah `php artisan tinker`.

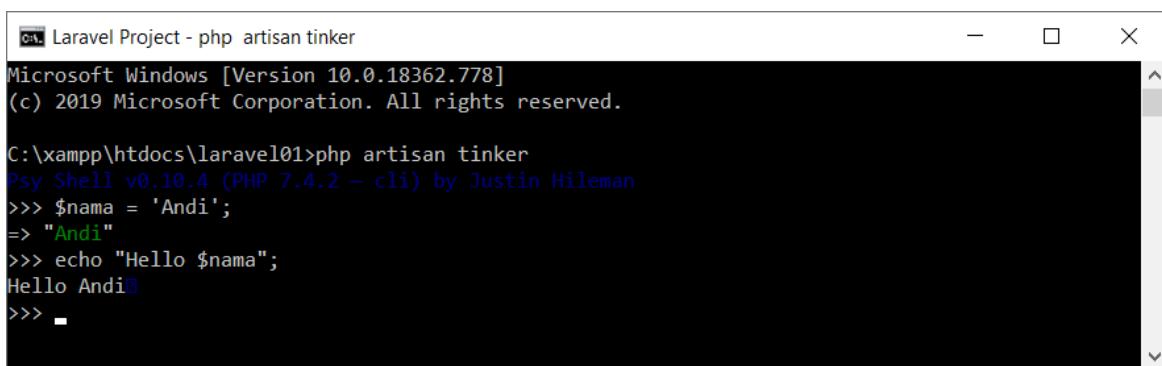


```
Laravel Project - php artisan tinker
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\laravel01>php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.2 - cli) by Justin Hileman
>>> .
```

Gambar: Tampilan Tinker

Jika tidak ada masalah, cursor cmd akan berubah menjadi tanda `>>>`. Ini artinya kita sudah masuk ke dalam Tinker. Mari coba perintah yang sama seperti sebelumnya:



```
Laravel Project - php artisan tinker
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\laravel01>php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.2 - cli) by Justin Hileman
>>> $nama = 'Andi';
=> "Andi"
>>> echo "Hello $nama";
Hello Andi
>>> .
```

Gambar: Mencoba Tinker

Hasilnya kurang lebih sama seperti PHP *Interactive Shell*. Namun dalam Tinker tanda titik koma di akhir baris tidak perlu ditulis. Selain itu untuk menampilkan teks atau isi variabel juga

Tinker

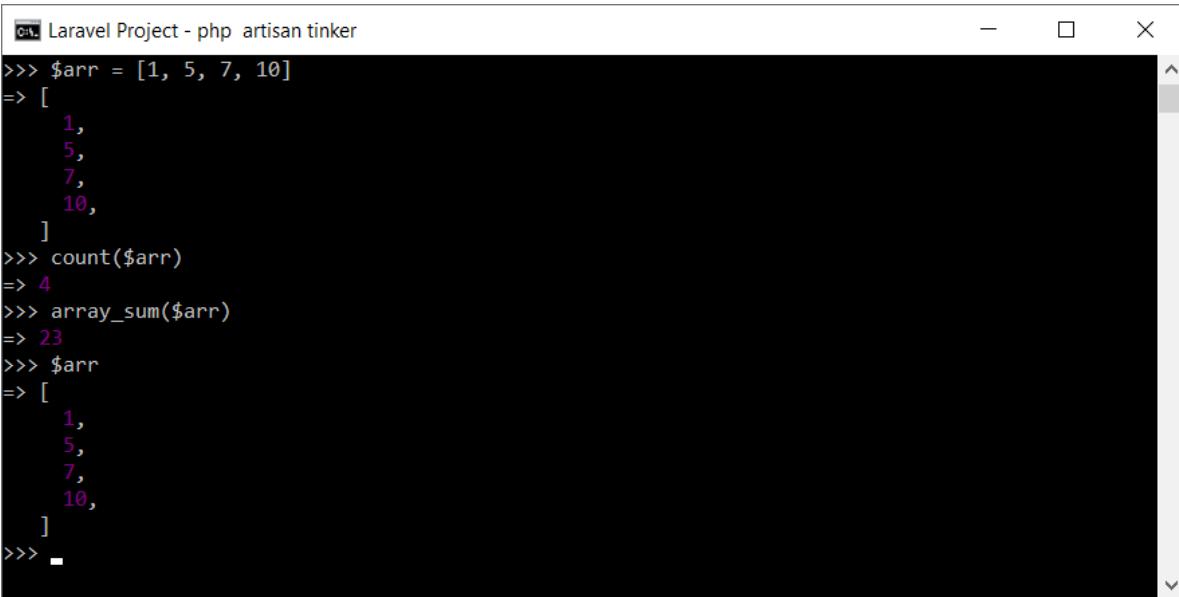
bisa dilakukan secara langsung tanpa perintah echo:



```
Laravel Project - php artisan tinker
>>> $nama = 'Lisa'
=> "Lisa"
>>> "Hello $nama"
=> "Hello Lisa"
>>> $nama
=> "Lisa"
>>> -
```

Gambar: Menampilkan variabel di Tinker

Variabel dengan data kompleks seperti array juga ditampilkan dengan lebih rapi:



```
Laravel Project - php artisan tinker
>>> $arr = [1, 5, 7, 10]
=> [
    1,
    5,
    7,
    10,
]
>>> count($arr)
=> 4
>>> array_sum($arr)
=> 23
>>> $arr
=> [
    1,
    5,
    7,
    10,
]
>>> -
```

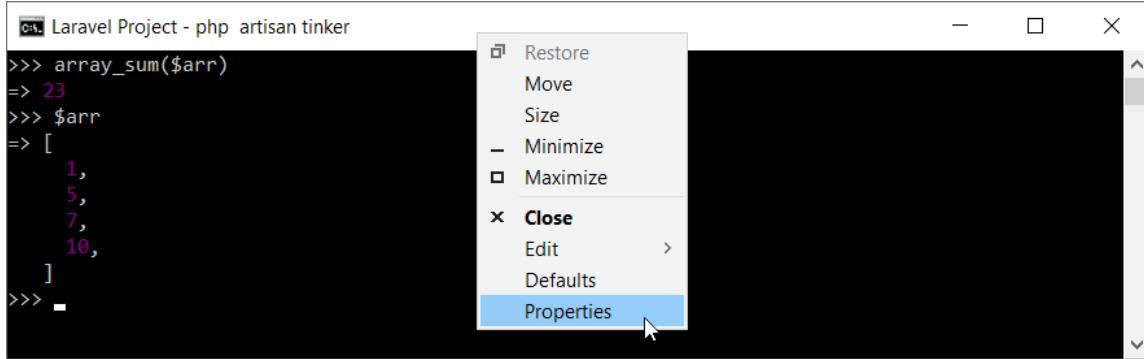
Gambar: Menampilkan array di Tinker

Kurang lebih seperti inilah fungsi dasar dari Tinker, yaitu mengeksekusi perintah PHP dan langsung menampilkan hasilnya. Untuk keluar dari Tinker, bisa menggunakan perintah `exit`.

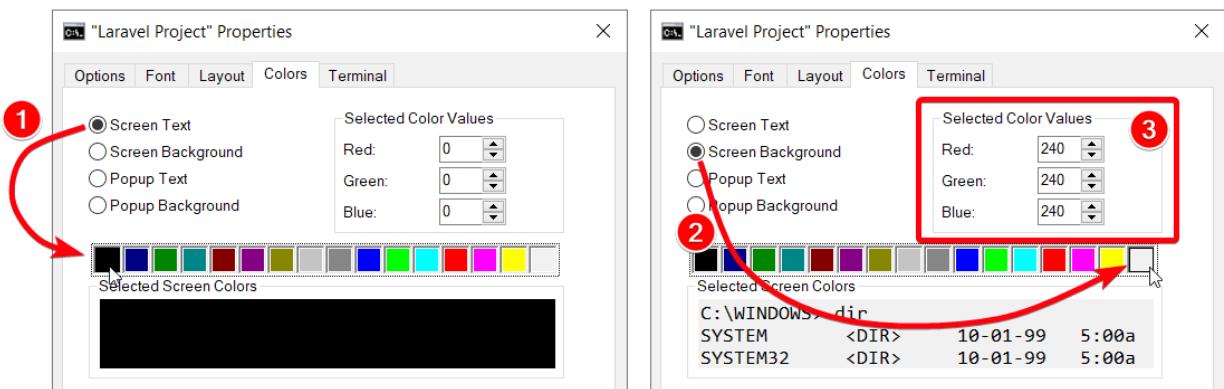
Di dalam Laravel, Tinker lebih sering dipakai untuk mengakses tabel di database. Namun sebelum itu, saya ingin mengubah tampilan warna layar cmd terlebih dahulu. Alasannya karena teks hasil Tinker tampil dalam berbagai warna (biru, hijau dan ungu). Warna-warna ini sedikit susah dilihat dengan background hitam.

Untuk mengubah warna teks dan warna background cmd, klik kanan bingkai atas aplikasi cmd lalu pilih menu **Properties** dan buka tab **Colors**.

Tinker



Gambar: Akses menu Properties di cmd

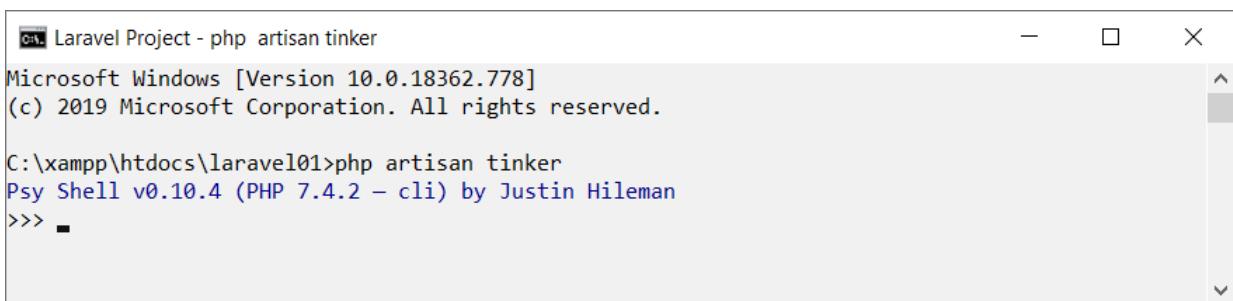


Gambar: Tukar warna teks dan background cmd

Di dalam tab **Colors**, klik pilihan **Screen Text** lalu tukar ke warna hitam (1). Setelah itu pindah ke **Screen Background** dan tukar ke warna putih (2). Agar warna putih ini tidak terlalu terang, saya akan pakai warna abu-abu dengan menginput angka 240 di ketiga kotak RGB (3). Setelah itu akhiri dengan klik tombol **OK**.

Kita kembali ke tampilan cmd dan terlihat tidak ada perubahan apa-apa. Ini karena aplikasi cmd perlu di restart terlebih dahulu.

Silahkan tutup cmd, lalu buka kembali. Sekarang tampilannya sudah berubah sesuai dengan warna yang dipilih. Anda bebas ingin memakai variasi warna lain sesuai keinginan.



Gambar: Tampilan warna cmd sudah berubah

2.2. Cara Akses Database dengan Tinker

Penggunaan paling umum dari Tinker adalah untuk mengakses database, baik itu menampilkan data atau menginput data baru ke dalam database. Ini sangat praktis karena kita tidak perlu menulis query di Route atau Controller.

Sebagai bahan percobaan, saya akan buat sebuah model Mahasiswa. Silahkan ketik perintah berikut ke dalam cmd:

```
php artisan make:model Mahasiswa -m
```

Perintah ini akan membuat model Mahasiswa beserta migration untuk tabel mahasiswas. Buka file migration mahasiswas ini lalu tambah pendefinisian 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->date('tanggal_lahir');
8         $table->decimal('ipk',3,2)->default(1.00);
9         $table->timestamps();
10    });
11 }
```

Dengan kode di atas, tabel mahasiswas akan memiliki kolom id, nim, nama, tanggal_lahir, ipk, date_created dan date_modified. Lanjut, jalankan migration dengan perintah:

```
php artisan migrate
```

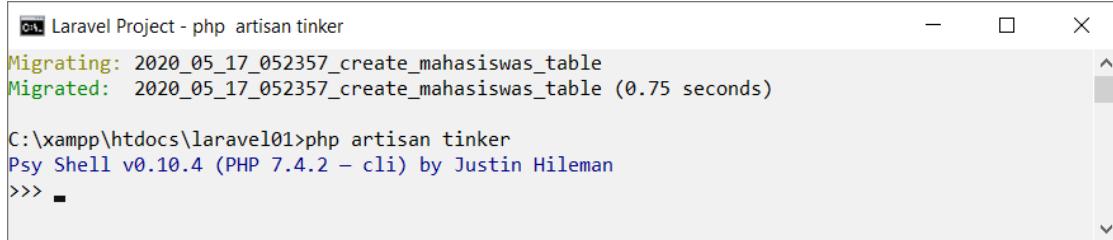
```
C:\xampp\htdocs\laravel01>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (555.83ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (511.84ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (722.11ms)
Migrating: 2020_10_07_022445_create_mahasiswas_table
Migrated: 2020_10_07_022445_create_mahasiswas_table (244.04ms)

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

Gambar: Jalankan proses migration

Sip, tabel mahasiswas sudah siap di akses. Sekarang buka kembali Tinker dengan perintah php artisan tinker:

Tinker



```
Laravel Project - php artisan tinker
Migrating: 2020_05_17_052357_create_mahasiswa_table
Migrated: 2020_05_17_052357_create_mahasiswa_table (0.75 seconds)

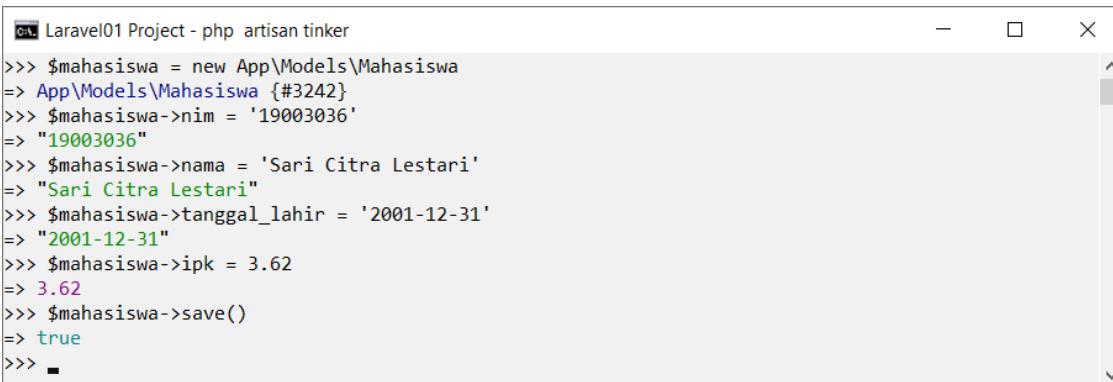
C:\xampp\htdocs\laravel01>php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.2 - cli) by Justin Hileman
>>> -
```

Gambar: Jalankan Tinker

Praktek pertama kita adalah menambah 1 data baru ke tabel `mahasiswa` menggunakan Tinker. Sama seperti di Laravel, terdapat banyak cara yang bisa dipakai. Kita akan coba dengan Eloquent terlebih dahulu.

Silahkan ketik perintah berikut satu per satu ke dalam Tinker, akhiri setiap baris dengan menekan tombol Enter:

```
1 $mahasiswa = new App\Models\Mahasiswa
2 $mahasiswa->nim = '19003036'
3 $mahasiswa->nama = 'Sari Citra Lestari'
4 $mahasiswa->tanggal_lahir = '2001-12-31'
5 $mahasiswa->ipk = 3.62
6 $mahasiswa->save()
```



```
Laravel01 Project - php artisan tinker
>>> $mahasiswa = new App\Models\Mahasiswa
=> App\Models\Mahasiswa {#3242}
>>> $mahasiswa->nim = '19003036'
=> "19003036"
>>> $mahasiswa->nama = 'Sari Citra Lestari'
=> "Sari Citra Lestari"
>>> $mahasiswa->tanggal_lahir = '2001-12-31'
=> "2001-12-31"
>>> $mahasiswa->ipk = 3.62
=> 3.62
>>> $mahasiswa->save()
=> true
>>> -
```

Gambar: Insert data menggunakan Tinker

Setiap kali sebuah perintah selesai di input, Tinker akan menampilkan sesuatu, entah itu nilai awal yang diketik, atau nilai True seperti hasil perintah `$mahasiswa->save()`. True di sini artinya proses input berjalan dengan sukses.

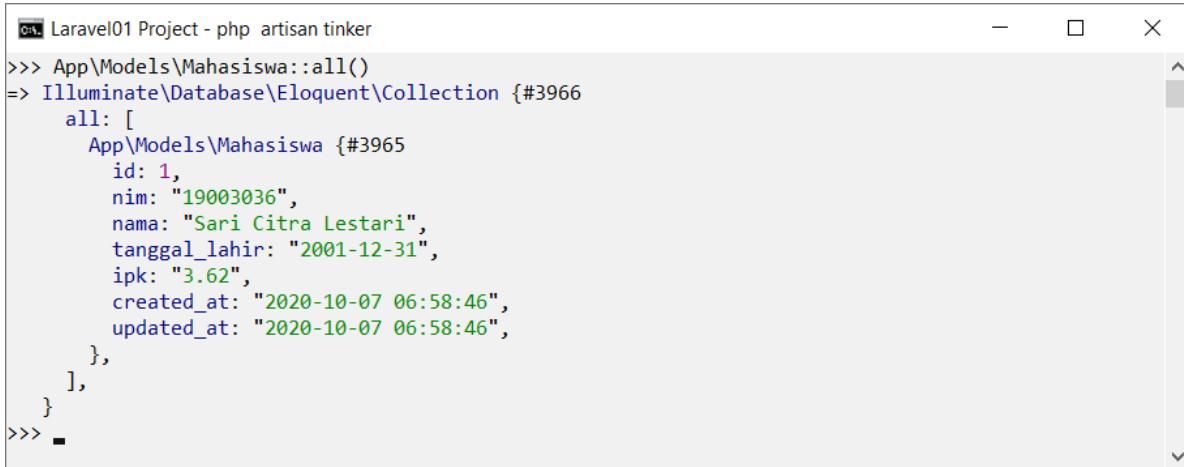
Perintah di atas merupakan proses input menggunakan Eloquent. Untuk penjelasan lengkapnya, sudah pernah kita bahas di buku Laravel Uncover.

Di baris pertama saya mengisi variabel `$mahasiswa` dengan Eloquent object `App\Models\Mahasiswa`. Penulisan object `Mahasiswa` harus disertai namespace `App` agar bisa diakses oleh Tinker. Kemudian satu per satu data diinput untuk kolom `nim`, `nama`, `tanggal_lahir` dan `ipk` ke variabel `$mahasiswa`.

Untuk melihat hasilnya, juga bisa memakai perintah Eloquent biasa, yakni:

Tinker

```
1 App\Models\Mahasiswa::all()
```



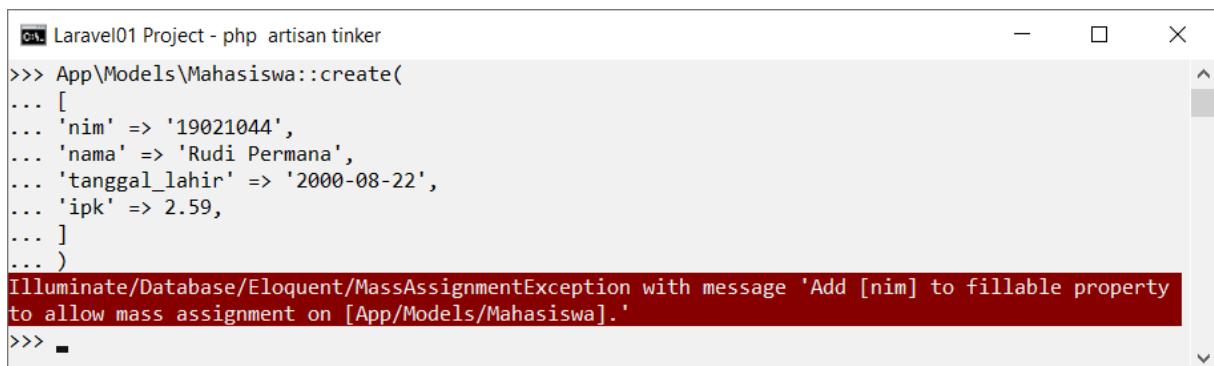
```
Laravel01 Project - php artisan tinker
>>> App\Models\Mahasiswa::all()
=> Illuminate\Database\Eloquent\Collection {#3966
    all: [
        App\Models\Mahasiswa {#3965
            id: 1,
            nim: "19003036",
            nama: "Sari Citra Lestari",
            tanggal_lahir: "2001-12-31",
            ipk: "3.62",
            created_at: "2020-10-07 06:58:46",
            updated_at: "2020-10-07 06:58:46",
        },
    ],
}
>>> -
```

Gambar: Menampilkan data menggunakan Tinker

Perintah `App\Models\Mahasiswa::all()` akan mengembalikan Collection object yang berisi semua data dari tabel `mahasiswas`. Karena hanya 1 data mahasiswa, maka hasilnya juga hanya 1 object.

Selanjutnya saya akan coba input data kedua tapi kali ini menggunakan teknik mass assignment Eloquent:

```
1 App\Models\Mahasiswa::create(
2 [
3     'nim' => '19021044',
4     'nama' => 'Rudi Permana',
5     'tanggal_lahir' => '2000-08-22',
6     'ipk' => 2.59,
7 ]
8 )
```



```
Laravel01 Project - php artisan tinker
>>> App\Models\Mahasiswa::create(
... [
...     'nim' => '19021044',
...     'nama' => 'Rudi Permana',
...     'tanggal_lahir' => '2000-08-22',
...     'ipk' => 2.59,
... ]
...
)
Illuminate\Database\Eloquent/MassAssignmentException with message 'Add [nim] to fillable property to allow mass assignment on [App/Models/Mahasiswa].'
>>> -
```

Gambar: Input data mass assignment menggunakan Tinker error

Ups, tampil pesan error. Error ini bukalah karena salah ketik perintah, tapi karena hal lain. Bisakah anda menebak apa yang kurang?

Betul, untuk bisa menginput data menggunakan *mass assignment*, kita harus tambah property `$fillable` atau `$guarded` ke dalam model Mahasiswa. Maka silahkan tambah kode `protected`

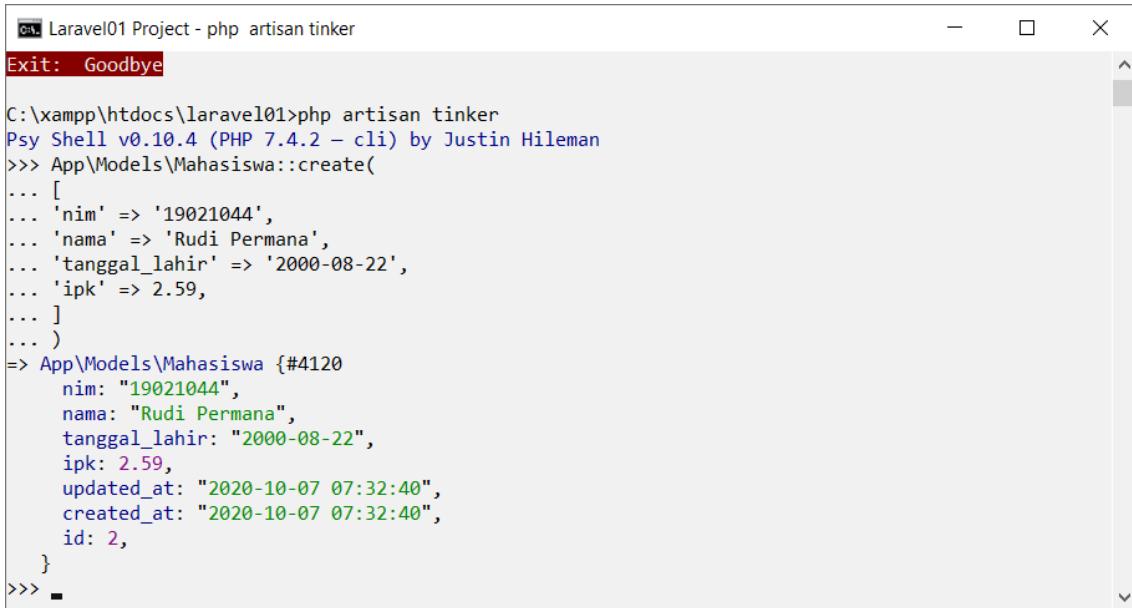
\$guarded = [] ke dalam file app\Models\Mahasiswa.php.

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 $guarded = [];
12 }
```

Tambahan property \$guarded = [] artinya kita mengizinkan semua kolom dari model Mahasiswa untuk diinput menggunakan mass assignment.

Agar pengaturan ini efektif, Tinker harus di restart terlebih dahulu. Silahkan ketik perintah exit, lalu masuk lagi ke Tinker dan jalankan ulang perintah mass assignment sebelumnya:



```
Laravel01 Project - php artisan tinker
Exit: Goodbye

C:\xampp\htdocs\laravel01>php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.2 - cli) by Justin Hileman
>>> App\Models\Mahasiswa::create(
... [
... 'nim' => '19021044',
... 'nama' => 'Rudi Permana',
... 'tanggal_lahir' => '2000-08-22',
... 'ipk' => 2.59,
... ]
...
=> App\Models\Mahasiswa {#4120
    nim: "19021044",
    nama: "Rudi Permana",
    tanggal_lahir: "2000-08-22",
    ipk: 2.59,
    updated_at: "2020-10-07 07:32:40",
    created_at: "2020-10-07 07:32:40",
    id: 2,
}
>>> -
```

Gambar: Input data mass assignment menggunakan Tinker berhasil

Jika terlihat tampilan di atas maka proses input dengan mass assignment sudah berhasil. Untuk memastikannya, silahkan ketik kembali perintah App\Models\Mahasiswa::all(). Seharusnya sekarang tampil 2 buah eloquent object.

Sebagai latihan, bisakah anda menambah data mahasiswa berikut menggunakan Tinker?

- Nim: 19002032
- Nama: Rina Kumala Sari,
- Tanggal Lahir: 28 Juni 2000

- IPK: 3.4

Tidak masalah ingin menggunakan cara biasa atau mass assignment.

Baik, berikut perintah jika menggunakan mass assignment yang sama seperti cara kita sebelumnya:

```
1 App\Models\Mahasiswa::create(
2 [
3 'nim' => '19002032',
4 'nama' => 'Rina Kumala Sari',
5 'tanggal_lahir' => '2000-06-28',
6 'ipk' => 3.4,
7 ]
8 )
```

```
Laravel01 Project - php artisan tinker
>>> App\Models\Mahasiswa::create(
... [
... 'nim' => '19002032',
... 'nama' => 'Rina Kumala Sari',
... 'tanggal_lahir' => '2000-06-28',
... 'ipk' => 3.4,
... ]
...
=> App\Models\Mahasiswa {#3254
    nim: "19002032",
    nama: "Rina Kumala Sari",
    tanggal_lahir: "2000-06-28",
    ipk: 3.4,
    updated_at: "2020-10-07 07:36:27",
    created_at: "2020-10-07 07:36:27",
    id: 3,
}
>>> -
```

Gambar: Input data mass assignment menggunakan Tinker

Dengan tambahan ini, di dalam tabel `mahasiswas` sudah terdapat 3 data.

Berbagai perintah Eloquent lain juga bisa di jalankan, misalnya untuk proses pencarian seperti contoh berikut:

```
1 App\Models\Mahasiswa::find(3)
2 App\Models\Mahasiswa::where('ipk','>',3)->count()
3 App\Models\Mahasiswa::pluck('nama')->toArray()
```

Tinker

```
Laravel01 Project - php artisan tinker
>>> App\Models\Mahasiswa::find(3)
=> App\Models\Mahasiswa {#4176
    id: 3,
    nim: "19002032",
    nama: "Rina Kumala Sari",
    tanggal_lahir: "2000-06-28",
    ipk: "3.40",
    created_at: "2020-10-07 07:36:27",
    updated_at: "2020-10-07 07:36:27",
}
>>> App\Models\Mahasiswa::where('ipk','>',3)->count()
=> 2
>>> App\Models\Mahasiswa::pluck('nama')->toArray()
=> [
    "Sari Citra Lestari",
    "Rudi Permana",
    "Rina Kumala Sari",
]
>>> -
```

Gambar: Menjalankan berbagai perintah Eloquent di Tinker

Perintah pertama, `App\Models\Mahasiswa::find(3)` dipakai untuk menampilkan data mahasiswa dengan `id = 3`. Hasilnya berbentuk eloquent object atau `null` jika `id` tidak ditemukan.

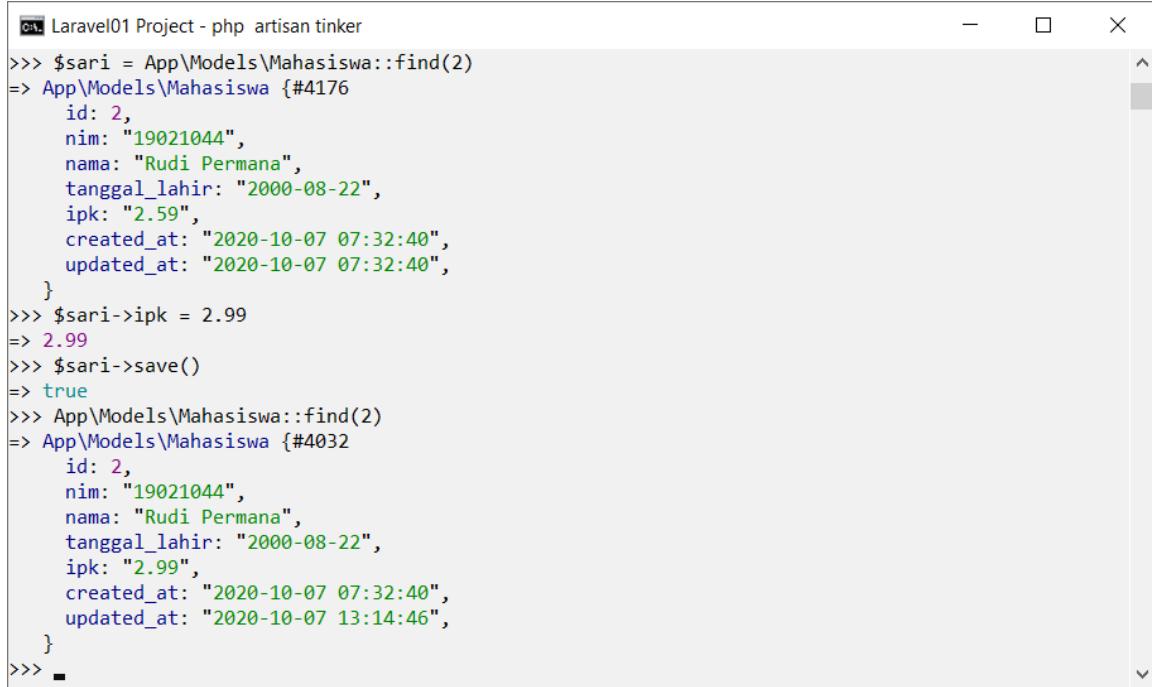
Perintah kedua, `App\Models\Mahasiswa::where('ipk','>',3)->count()` akan mencari jumlah mahasiswa yang memiliki IPK di atas 3. Hasilnya berupa angka 2 yang berarti terdapat 2 mahasiswa dengan IPK lebih dari 3.

Perintah ketiga, `App\Models\Mahasiswa::pluck('nama')->toArray()` berfungsi untuk menampilkan semua nama mahasiswa yang ditampilkan dalam bentuk array.

Jika anda butuh penjelasan lebih lanjut tentang perintah-perintah ini, bisa buka kembali bab Collection dan Eloquent di buku **Laravel Uncover**.

Lanjut, bagaimana dengan proses update dan delete? Tidak masalah, perintah yang diperlukan juga sama seperti Eloquent biasa:

```
1 $sari = App\Models\Mahasiswa::find(2)
2 $sari->ipk = 2.99
3 $sari->save()
4
5 App\Models\Mahasiswa::find(2)
```



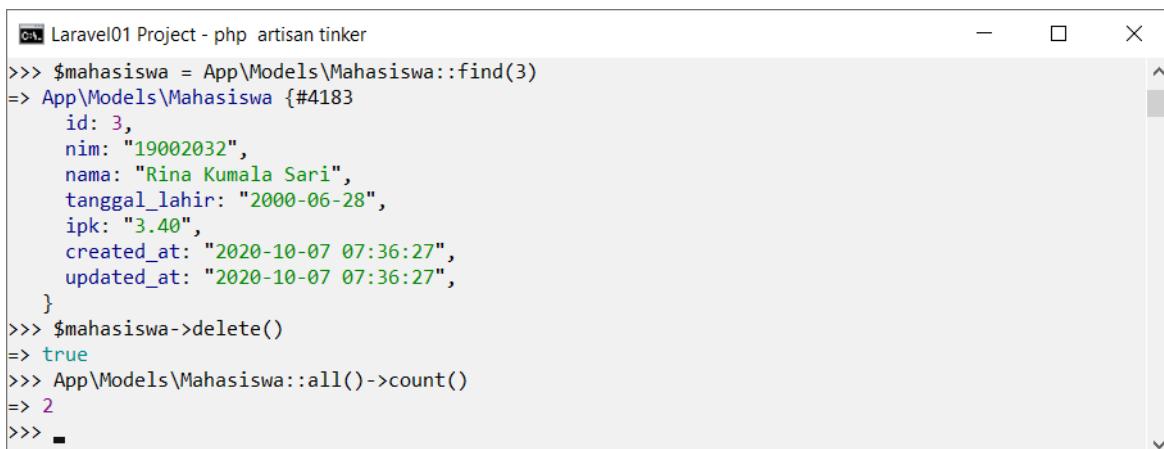
```
Laravel01 Project - php artisan tinker
>>> $sari = App\Models\Mahasiswa::find(2)
=> App\Models\Mahasiswa {#4176
    id: 2,
    nim: "19021044",
    nama: "Rudi Permana",
    tanggal_lahir: "2000-08-22",
    ipk: "2.59",
    created_at: "2020-10-07 07:32:40",
    updated_at: "2020-10-07 07:32:40",
}
>>> $sari->ipk = 2.99
=> 2.99
>>> $sari->save()
=> true
>>> App\Models\Mahasiswa::find(2)
=> App\Models\Mahasiswa {#4032
    id: 2,
    nim: "19021044",
    nama: "Rudi Permana",
    tanggal_lahir: "2000-08-22",
    ipk: "2.99",
    created_at: "2020-10-07 07:32:40",
    updated_at: "2020-10-07 13:14:46",
}
>>> -
```

Gambar: Proses Update di Tinker

Di sini saya mencoba untuk mengubah nilai IPK dari mahasiswa dengan id = 2 menjadi 2.99.

Untuk proses delete, bisa dilakukan dengan kode berikut:

```
1 $mahasiswa = App\Models\Mahasiswa::find(3)
2 $mahasiswa->delete()
3
4 App\Models\Mahasiswa::all()->count()
```



```
Laravel01 Project - php artisan tinker
>>> $mahasiswa = App\Models\Mahasiswa::find(3)
=> App\Models\Mahasiswa {#4183
    id: 3,
    nim: "19002032",
    nama: "Rina Kumala Sari",
    tanggal_lahir: "2000-06-28",
    ipk: "3.40",
    created_at: "2020-10-07 07:36:27",
    updated_at: "2020-10-07 07:36:27",
}
>>> $mahasiswa->delete()
=> true
>>> App\Models\Mahasiswa::all()->count()
=> 2
>>> -
```

Gambar: Proses Delete di Tinker

Hasilnya, data di dalam tabel mahasiswa hanya tinggal 2, sesuai dengan hasil dari App\Models\Mahasiswa::all()->count().

Dari beberapa percobaan ini, kita sudah melakukan praktik operasi CRUD menggunakan Tinker. Selain lewat eloquent, proses yang sama juga bisa dibuat dengan cara lain, misalnya

menggunakan query builder:

```

1 DB::table('mahasiswa')->insert(
2 [
3   'nim' => '18012012',
4   'nama' => 'James Situmorang',
5   'tanggal_lahir' => '1999-04-02',
6   'ipk' => 2.74,
7   'created_at' => now(),
8   'updated_at' => now(),
9 ]
10 )

```

```

Laravel Project - php artisan tinker
>>> DB::table('mahasiswa')->insert(
... [
...   'nim' => '18012012',
...   'nama' => 'James Situmorang',
...   'tanggal_lahir' => '1999-04-02',
...   'ipk' => 2.74,
...   'created_at' => now(),
...   'updated_at' => now(),
... ]
... )
=> true
>>> -

```

Gambar: Input data menggunakan query builder di Tinker

Kali ini saya menggunakan query builder untuk menginput data 'James Situmorang' ke dalam tabel `mahasiswa`. Apabila diperlukan, perintah raw query-pun juga bisa di jalankan dari Tinker.

Exercise

Sebagai latihan tambahan terkait Tinker dan Eloquent, bisakah anda menambah 1 data baru ke tabel `users` bawaan Laravel?

Tabel `users` terdiri dari 3 kolom utama: `name`, `email` dan `password`. Ketika menjalankan proses migration, tabel ini otomatis dibuat dan juga sudah memiliki model `User` sehingga sudah siap pakai.

Catatan: untuk kolom `password` sebenarnya bisa diisi dengan string biasa, atau bisa juga menggunakan perintah `Hash::make('<string_password>')` untuk menggenerate nilai hash password.

Answer

Berikut perintah yang bisa dipakai:

```

1 $user = new App\Models\User
2 $user->name = 'Admin'

```

Tinker

```
3 $user->email = 'admin@gmail.com'  
4 $user->password = Hash::make('qwerty')  
5 $user->save()
```



```
Laravel01 Project - php artisan tinker  
>>> $user = new App\Models\User  
=> App\Models\User {#4120}  
>>> $user->name = 'Admin'  
=> "Admin"  
>>> $user->email = 'admin@gmail.com'  
=> "admin@gmail.com"  
>>> $user->password = Hash::make('qwerty')  
=> "$2y$10$zKZTyV3zKn/.OsT9r7Gm7e3PV.XbBFqvPIrJEbHF69r7dtZsR3A8C"  
>>> $user->save()  
=> true  
>>> -
```

Gambar: Proses input ke tabel users menggunakan Tinker

Untuk memastikan data memang sudah ada di database, bisa di cek menggunakan phpMyAdmin atau cmd MySQL client.

Dalam bab ini kita telah membahas sekilas tentang cara penggunaan Tinker. Pada dasarnya, perintah yang dipakai sama seperti yang kita tulis di Controller. Hanya saja jika perlu cepat, tinker lebih efisien. Selain itu Tinker juga bisa dipakai untuk menjalankan berbagai perintah Laravel lain, termasuk perintah dasar PHP.

Berikutnya kita akan bahas fitur **Accessor** dan **Mutator** dari Laravel.

3. Accessor dan Mutator

Kedua istilah ini terdengar rumit, tapi **Accessor** dan **Mutator** sebenarnya cukup sederhana. Accessor dipakai untuk memproses data sebelum ditampilkan dari database, sedangkan Mutator dipakai untuk memproses data sebelum diinput ke dalam database. Dalam bab ini kita akan bahas cara penggunaan keduanya.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

3.1. Persiapan Awal

Sebelum masuk ke materi tentang **Accessor** dan **Mutator**, kita perlu siapkan bahan praktek. Sama seperti bab sebelumnya, saya kembali menggunakan model Mahasiswa. Berikut perintah yang diperlukan untuk membuat model Mahasiswa beserta file migration:

```
php artisan make:model Mahasiswa -m
```

Kemudian buka file migration mahasiswas lalu tambah pendefinisian 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->date('tanggal_lahir');
8          $table->decimal('ipk',3,2)->default(1.00);
9          $table->timestamps();
10     });
11 }
```

Jalankan migration dengan perintah:

```
php artisan migrate
```

Agar tidak bermasalah dengan proses *mass assignment*, tambah property `protected $guarded = []` ke dalam model Mahasiswa:

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 }
```

Terakhir, saya ingin mengisi beberapa data awal ke tabel `mahasiswas` dan juga tabel `users`. Ini bisa saja dilakukan dari Tinker, namun agar lebih mudah di copy paste dan dijalankan ulang, saya akan tulis di route saja. Silahkan buka file `route\web.php`, lalu modifikasi sebagai berikut:

routes\web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Models\Mahasiswa;
5 use App\Models\User;
6 use Illuminate\Support\Facades\Hash;
7
8 Route::get('/', function () {
9     return view('welcome');
10 });
11
12 Route::get('/generate', function () {
13     Mahasiswa::create(
14         [
15             'nim' => '19003036',
16             'nama' => 'Sari Citra Lestari',
17             'tanggal_lahir' => '2001-12-31',
18             'ipk' => 3.62,
19         ]
20     );
21     Mahasiswa::create(
22         [
23             'nim' => '19021044',
24             'nama' => 'Rudi Permana',
25             'tanggal_lahir' => '2000-08-22',
26             'ipk' => 2.99,
27         ],
28     );
29     Mahasiswa::create(
```

```

30      [
31          'nim' => '19002032',
32          'nama' => 'Rina Kumala Sari',
33          'tanggal_lahir' => '2000-06-28',
34          'ipk' => 3.82,
35      ],
36  );
37  Mahasiswa::create(
38      [
39          'nim' => '18012012',
40          'nama' => 'James Situmorang',
41          'tanggal_lahir' => '1999-04-02',
42          'ipk' => 2.74,
43      ]
44  );
45
46 User::create(
47     [
48         'name' => 'Admin',
49         'email' => 'admin@gmail.com',
50         'password' => Hash::make('qwerty'),
51     ]
52 );
53
54 return "Penambahan data tabel berhasil";
55 });

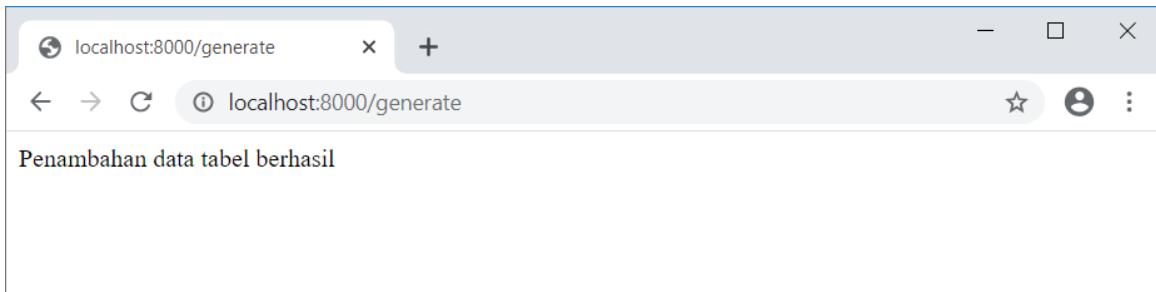
```

Di baris 3-6 terdapat 4 perintah import namespace. Perintah pertama untuk import **Route** facade yang merupakan kode bawaan route Laravel. Kemudian perintah kedua dan ketiga dipakai untuk import model **Mahasiswa** dan **User**. Terakhir terdapat import **Hash** facade yang diperlukan untuk proses generate nilai hashing kolom password di tabel users.

Baris 8-10 merupakan route welcome bawaan Laravel. Route ini tidak akan kita pakai dan boleh saja jika ingin dihapus.

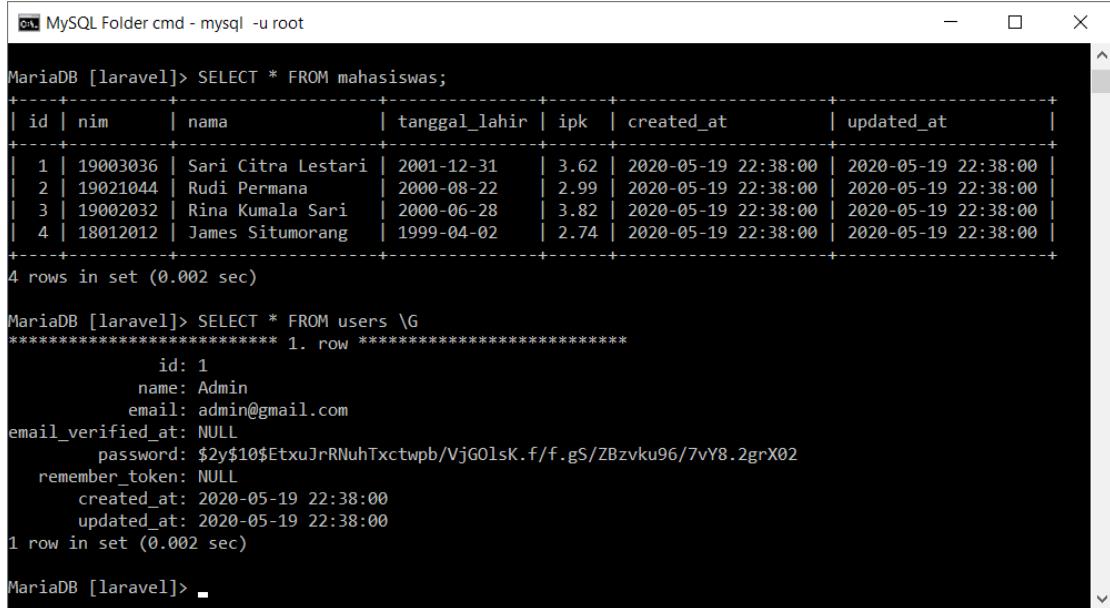
Route untuk proses pengisian tabel ada di baris 12-55. Isinya berupa perintah *mass assignment* dari Eloquent untuk meng-input 4 data ke dalam tabel **mahasiswas** dan 1 data ke dalam tabel **users**. Jika semua proses sukses, route akan menampilkan "Penambahan data tabel berhasil".

Save file route di atas, lalu akses dari alamat <http://localhost:8000/generate>.



Gambar: Proses generate data berhasil

Untuk membuktikan, silahkan buka cmd MySQL Client atau phpMyAdmin dan periksa isi tabel mahasiswa dan juga tabel users.



```

MySQL [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.62 | 2020-05-19 22:38:00 | 2020-05-19 22:38:00 |
| 2 | 19021044 | Rudi Permana | 2000-08-22 | 2.99 | 2020-05-19 22:38:00 | 2020-05-19 22:38:00 |
| 3 | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.82 | 2020-05-19 22:38:00 | 2020-05-19 22:38:00 |
| 4 | 18012012 | James Situmorang | 1999-04-02 | 2.74 | 2020-05-19 22:38:00 | 2020-05-19 22:38:00 |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM users \G
***** 1. row *****
    id: 1
    name: Admin
    email: admin@gmail.com
email_verified_at: NULL
password: $2y$10$EttxuJrRNuhTxctwpb/VjG0lsK.f/f.gS/ZBzvku96/7vY8.2grX02
remember_token: NULL
created_at: 2020-05-19 22:38:00
updated_at: 2020-05-19 22:38:00
1 row in set (0.002 sec)

MariaDB [laravel]>

```

Gambar: Isi tabel mahasiswa dan tabel users

Sip, bahan praktik kita sudah siap dan saatnya masuk ke materi tentang Accessor.

3.2. Pengertian Laravel Accessor

Accessor adalah sebuah method di dalam Model yang dipakai untuk mengolah data sebelum ditampilkan dari Eloquent. Pengertian ini akan lebih mudah dipahami dengan contoh praktik.

Silahkan tambah kode berikut ke dalam route, tempatkan setelah kode untuk proses generate sebelumnya:

routes\web.php

```

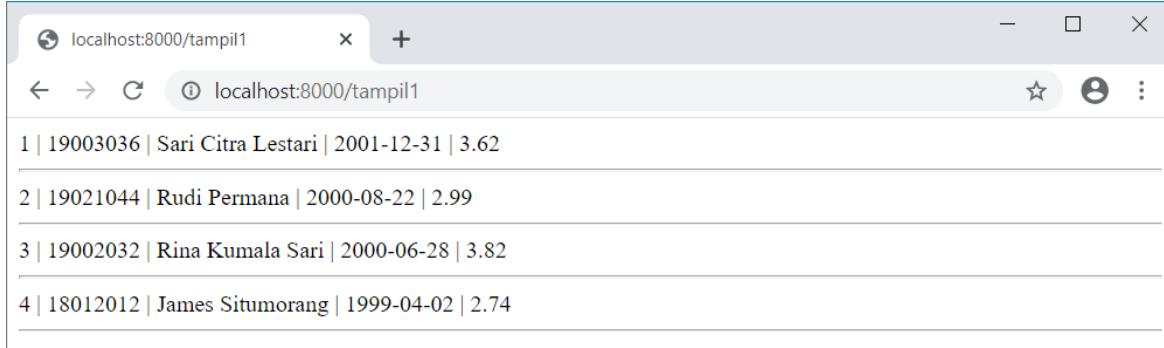
1 ...
2 ...
3 Route::get('/tampil1', function () {
4     $mahasiswa = Mahasiswa::all();
5     foreach ($mahasiswa as $mahasiswa) {
6         echo "$mahasiswa->id | ";
7         echo "$mahasiswa->nim | ";
8         echo "$mahasiswa->nama | ";
9         echo "$mahasiswa->tanggal_lahir | ";
10        echo "$mahasiswa->ipk <hr>";
11    }
12 });

```

Di sini saya membuat route '/tampil1' yang ketika diakses akan menampilkan semua isi tabel mahasiswa. Proses menampilkan data dilakukan dengan men-foreach isi variabel

\$mahasiswa. Variabel \$mahasiswa sendiri berisi Collection hasil dari perintah Mahasiswa::all() di baris 4. Kembali, kode yang ada merupakan perintah Eloquent biasa yang sudah kita pelajari di Laravel Uncover.

Berikut hasil tampilan ketika alamat localhost:8000/tampil1 diakses:



Gambar: Menampilkan semua data tabel mahasiswa

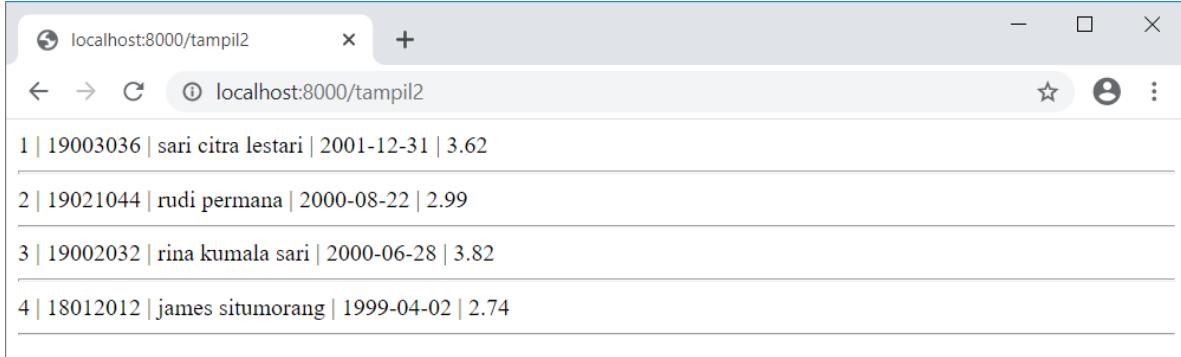
Dalam praktik di atas saya menulis kode Eloquent langsung di dalam route menggunakan *anonymous function / closure*. Ini semata-mata untuk menyederhanakan pembahasan. Idealnya, kode ini di tulis dari Controller dan hasilnya ditampilkan di View.

Dalam bab-bab berikutnya cara seperti ini juga banyak saya pakai agar kita tidak repot membuat file Controller dan View untuk setiap materi.

Sekarang, bagaimana supaya nilai kolom nama tampil dalam huruf kecil semua? Tidak masalah, tinggal lewatkannya nilai \$mahasiswa->nama ke dalam function strtolower() bawaan PHP:

routes\web.php

```
1 ...  
2 ...  
3 Route::get('/tampil2', function () {  
4     $mahasiswa = Mahasiswa::all();  
5     foreach ($mahasiswa as $mahasiswa) {  
6         echo "$mahasiswa->id | ";  
7         echo "$mahasiswa->nim | ";  
8         echo strtolower($mahasiswa->nama)." | ";  
9         echo "$mahasiswa->tanggal_lahir | ";  
10        echo "$mahasiswa->ipk <br>";  
11    }  
12});
```



Gambar: Data nama tampil dalam huruf kecil

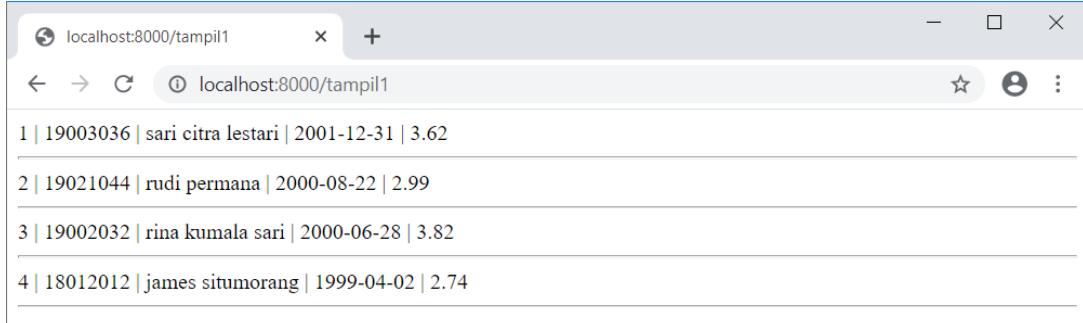
Sekarang bayangkan jika proses mengakses kolom `nama` ini ada di berbagai tempat (misal di 20 controller terpisah). Kemudian tiba-tiba terdapat aturan baru agar `nama` harus tampil dalam huruf besar. Akan cukup repot untuk mencari satu per satu function `strtolower($mahasiswa->nama)` dan menggantinya ke `strtoupper($mahasiswa->nama)` di setiap file.

Accessor bisa jadi solusi dari masalah kita. Silahkan edit model Mahasiswa dan tambah satu method baru bernama `getNamaAttribute()`. Berikut kode lengkap dari model Mahasiswa:

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 $guarded = [];
12
13     public function getNamaAttribute($value)
14     {
15         return strtoupper($value);
16     }
17 }
```

Kode tambahan ada di baris 13 – 16. Setelah itu buka kembali alamat `localhost:8000/tampil1`, yakni route yang menampilkan data mahasiswa tanpa tambahan function `strtolower()`:



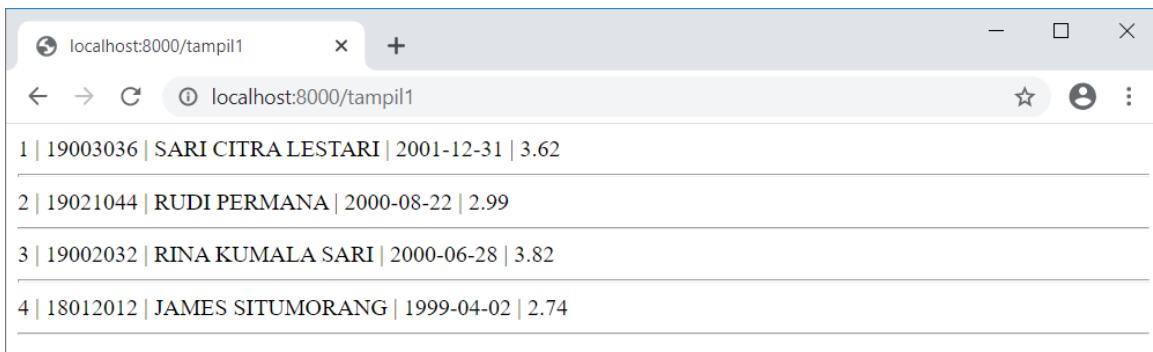
Gambar: Hasil accessor getNamaAttribute()

Hasilnya, kolom `nama` langsung tampil dengan huruf kecil.

Sebagai percobaan, silahkan ganti function `strtolower()` di method `getNamaAttribute()` menjadi `strtoupper()`, maka kolom `nama` juga otomatis berubah jadi huruf besar:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function getNamaAttribute($value)  
4     {  
5         return strtoupper($value);  
6     }
```



Gambar: Hasil accessor getNamaAttribute()

Inilah prinsip kerja dari accessor, yakni memproses data di dalam model sebelum ditampilkan oleh Eloquent.

3.3. Cara Penulisan Accessor

Mari kita bahas lebih jauh tentang cara penulisan Accessor. Sebuah method accessor harus ditulis dengan format berikut:

```
public function get<NamaKolomTabel>Attribute($value)  
{  
    return ....($value);  
}
```

Penulisan nama method harus diawali dengan kata "**get**", kemudian diikuti dengan nama kolom tabel, lalu disambung dengan kata "**Attribute**".

Nama kolom disarankan ditulis dalam **CamelCase**, dimana setiap karakter pertama kata ditulis dalam huruf besar. Jika nama kolom terdapat spasi, spasi tersebut dibuang dan huruf pertama berikutnya ditulis dalam huruf besar.

Sebagai contoh, jika ingin membuat accessor untuk kolom `nim` maka nama methodnya adalah `getNimAttribute()`, atau untuk kolom `tanggal_lahir` maka nama methodnya adalah `getTanggalLahirAttribute()`.

Sebuah method accessor bisa menerima 1 argument, yang akan diisi Laravel dengan nilai asal kolom tersebut. Dalam contoh di atas, argument ini saya tampung ke dalam parameter `$value`. Nama `$value` ini hanya sebagai *placeholder* saja dan bisa diganti menjadi nama variabel lain seperti `$nilai` atau `$a`.

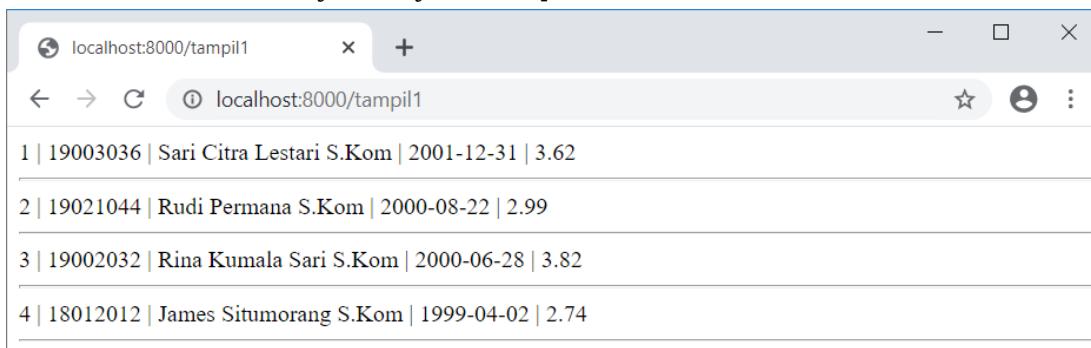
Di dalam accessor, nilai `$value` bisa diolah lebih lanjut, misalnya diinput ke dalam function `strtolower()` atau proses lain. Hasil dari pengolahan data ini di kembalikan dengan perintah `return`.

Sebagai contoh, bisakah anda menebak hasil akhir dari kolom `nama` dengan penulisan accessor berikut?

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function getNamaAttribute($value)  
4     {  
5         return $value." S.Kom";  
6     }
```

Yup, semua mahasiswa sudah jadi sarjana komputer!



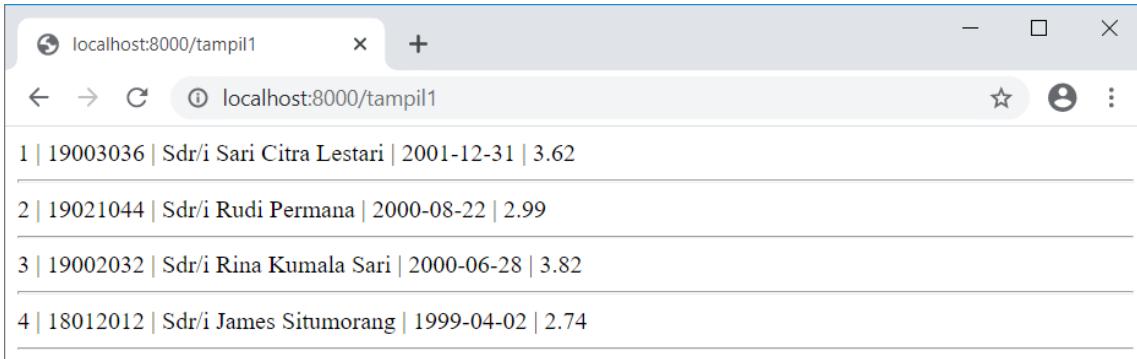
Gambar: Hasil accessor `getNamaAttribute()`

Cara lain untuk mengakses nilai awal suatu kolom adalah dari property `$this->attributes` `['nama_kolom']`, seperti contoh berikut:

Accessor dan Mutator

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function getNamaAttribute()  
4     {  
5         return "Sdr/i ".$this->attributes['nama'];  
6     }
```



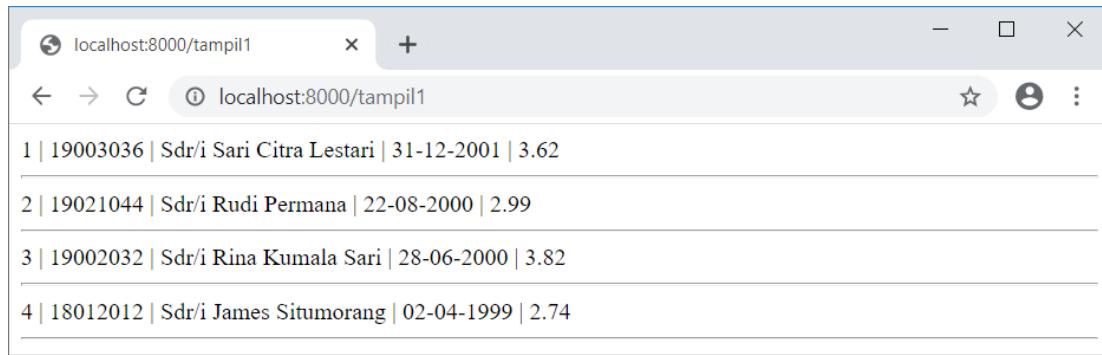
Gambar: Hasil accessor getNamaAttribute()

Kali ini accessor `getNamaAttribute()` tidak lagi menerima argument `$value`. Namun nilai awal kolom nama tetap bisa diakses dari property `$this->attributes['nama']`.

Contoh lain, saya ingin membuat accessor untuk kolom `tanggal_lahir` supaya tampil sesuai format yang biasa kita lihat, yakni DD-MM-YYYY, bukan dalam format YYYY-MM-DD bawaan tipe data DATE MySQL. Berikut penulisan methodnya:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function getTanggalLahirAttribute($value)  
4     {  
5         return date("d-m-Y", strtotime($value));  
6     }
```



Gambar: Hasil accessor getTanggalLahirAttribute()

Di akhir baris 5, saya menginput parameter `$value` ke dalam function `strtotime()` bawaan PHP. Hasilnya adalah tanggal dalam nilai timestamp yang menjadi argument kedua dari

function date(). Dengan menulis kode format "d-m-Y", maka tanggal akan tampil dalam bentuk DD-MM-YYYY.

Agar lebih menantang, bagaimana jika digit bulan ini ditampilkan dalam kata bahasa Indonesia? Misalnya 31 Desember 2001 atau 02 April 1999?

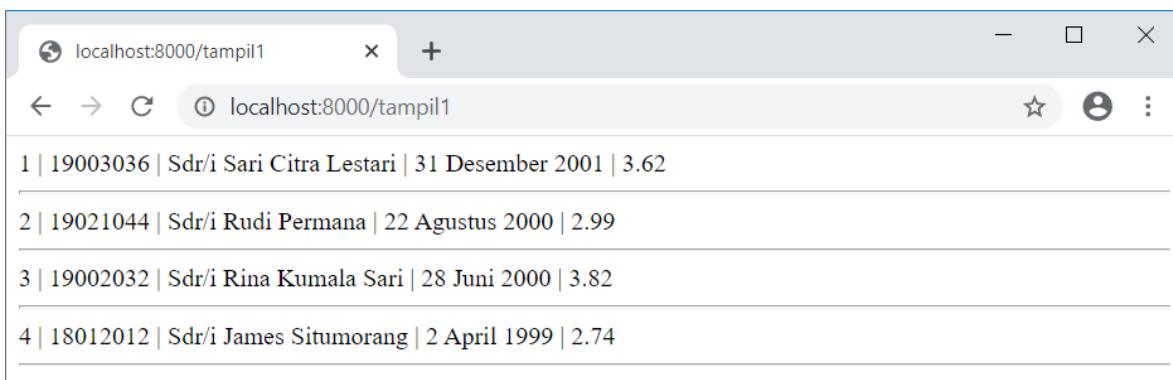
Yang perlu kita pikirkan adalah bagaimana mengkonversi digit bulan seperti 01 menjadi string 'Januari'. Salah satu solusi bisa dengan menyiapkan array berisi nama-nama bulan, lalu akses dengan nomor index:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function getTanggalLahirAttribute($value)  
4     {  
5         $nama_bulan = ["Januari", "Februari", "Maret", "April", "Mei",  
6             "Juni", "Juli", "Agustus", "September", "Oktober",  
7             "November", "Desember"];  
8  
9         $tanggal = date("j", strtotime($value));  
10        $bulan = date("n", strtotime($value)) - 1;  
11        $tahun = date("Y", strtotime($value));  
12  
13        return "$tanggal $nama_bulan[$bulan] $tahun";  
14    }
```

Di baris 5 terdapat pendefinisian array \$nama_bulan yang berisi 12 string nama bulan. Nama bulan ini sudah terurut agar mudah diakses dengan nomor index. Namun karena index array mulai dari 0, maka di baris 10 saya harus mengurangi nilai digit bulan dari database sebanyak 1 angka.

Dengan demikian jika digit bulan dari database adalah 5, perintah di baris 13 akan diakses menjadi \$nama_bulan[4], sehingga menghasilkan 'April'.



Gambar: Hasil accessor getTanggalLahirAttribute() dengan nama bulan

Untuk memudahkan pengolahan data tanggal (*date and time*), Laravel sebenarnya sudah

menyertakan library **Carbon**. Library carbon ini akan kita bahas dalam bab selanjutnya.

Custom Accessor

Kita juga bisa membuat accessor untuk kolom yang tidak ada di tabel, atau bisa disebut sebagai *custom accessor*. Sebagai contoh, perhatikan kode berikut:

app\Models\Mahasiswa.php

```

1 ...
2 ...
3     public function getNamaBesarAttribute()
4     {
5         return strtoupper($this->attributes['nama']);
6     }

```

Kali ini saya membuat accessor `getNamaBesarAttribute()`. Di dalam tabel mahasiswas tidak ada kolom `nama_besar`, namun tidak masalah. Accessor ini saya buat untuk mengkonversi nilai `$this->attributes['nama']` menjadi huruf besar.

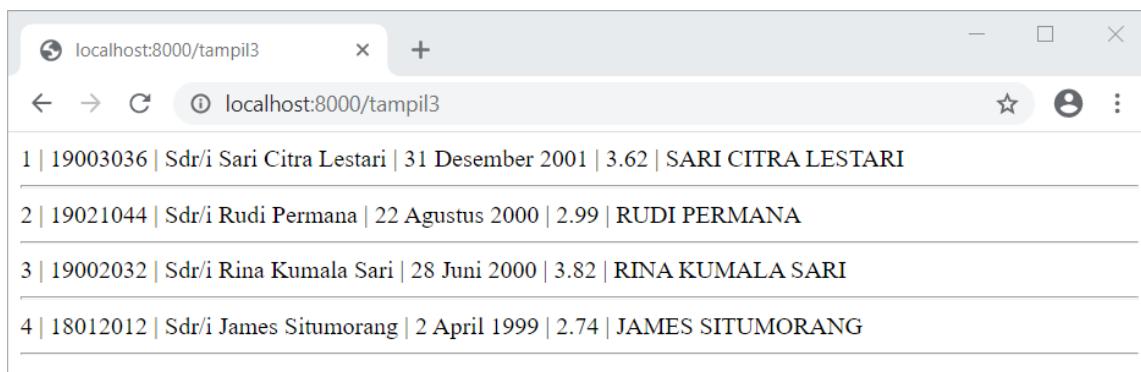
Cara mengaksesnya juga sama seperti mengakses kolom yang ada di dalam tabel Mahasiswa:

routes\web.php

```

1 ...
2 ...
3 Route::get('/tampil3', function () {
4     $mahasiswas = Mahasiswa::all();
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->id | ";
7         echo "$mahasiswa->nim | ";
8         echo "$mahasiswa->nama | ";
9         echo "$mahasiswa->tanggal_lahir | ";
10        echo "$mahasiswa->ipk | ";
11        echo "$mahasiswa->nama_besar <br>";
12    }
13 });

```



Gambar: Hasil custom accessor

Custom accessor ini cocok dipakai jika kita tetap ingin mengakses kolom asal namun jika ingin menampilkan data yang sudah diolah.

Mengakses Data Kolom Asal di Accessor

Sama seperti mayoritas materi lain di Laravel, terdapat banyak cara untuk mengakses data kolom asal dari dalam Accessor. Ini memang membuatnya fleksibel tapi kadang bikin bingung karena bisa jadi ada perbedaan antara satu cara dengan cara lain.

Tergantung situasi dan kebutuhan, nilai asal kolom tabel bisa diakses dari 3 perintah berikut:

- ◆ `$this->attributes['nama_kolom']`
- ◆ parameter `$value`
- ◆ `$this->nama_kolom`

Untuk cara pertama sudah kita praktekkan pada accessor `getNamaBesarAttribute()` sebelumnya. Di situ saya mengakses nilai kolom `nama` menggunakan property `$this->attributes['nama']`. Ini bisa dipakai untuk kolom-kolom lain.

Cara kedua adalah dengan mengakses parameter `$value`. Ini hanya bisa dilakukan untuk accessor yang namanya sesuai dengan kolom yang ada di tabel. Accessor `getNamaBesarAttribute()` tidak bisa menggunakan cara ini karena di dalam tabel mahasiswa tidak ada kolom `nama_besar`. Tapi tidak masalah untuk accessor `getNimAttribute($values)` dimana Laravel akan mengisi parameter `$values` dengan nilai kolom `nim` dari tabel mahasiswa.

Cara ketiga adalah dengan mengakses property `$this->nama_kolom`. Mari kita coba dengan memodifikasi accessor `getNamaBesarAttribute()`:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function getNamaBesarAttribute()  
4     {  
5         return strtoupper($this->nama);  
6     }
```

Sekarang yang diinput ke function `strtoupper()` adalah hasil dari `$this->nama`. Dan berikut hasilnya ketika route `localhost:8000/tampil3` di akses:



Gambar: Mengakses nilai \$this->nama di Accessor

Hasilnya, \$this->nama tetap mengembalikan nilai nama, tapi diambil dari hasil accessor `getNamaAttribute()`, bukan nilai nama asal tabel `mahasiswa`.

Namun jika di dalam Model kita tidak mengubah nilai kolom dengan membuat accessor untuk kolom tersebut, maka hasil dari `$this->nama_kolom` akan sama dengan `$this->attributes['nama_kolom']`.

3.4. Pengertian Laravel Mutator

Mutator adalah sebuah method di dalam Model yang dipakai untuk mengolah data sebelum diinput ke database. Sama seperti accessor, pengertian ini akan lebih mudah dipahami dengan praktek.

Misalnya saya ingin data untuk kolom `nama` di tabel `mahasiswa` harus tersimpan dalam huruf kecil. Data ini biasanya berasal dari form yang bisa saja diisi dengan huruf besar atau huruf kecil (suka-suka si user yang menginput). Agar nilai nama selalu menjadi huruf kecil, maka hasil inputan form harus dilewatkan terlebih dahulu ke dalam function `strtolower()`.

Sama seperti kasus di accessor, bayangkan jika terdapat banyak form yang dipakai untuk menginput data mahasiswa, maka kita harus selalu ingat untuk mengkonversi kolom nama menggunakan `strtolower()` di setiap pemrosesan form. **Mutator** bisa jadi solusi untuk masalah ini.

Sebagai contoh praktek, silahkan tambah method berikut ke dalam model Mahasiswa:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function setNamaAttribute($value)  
4     {  
5         $this->attributes['nama'] = strtolower($value);  
6     }
```

Kemudian tambah route berikut:

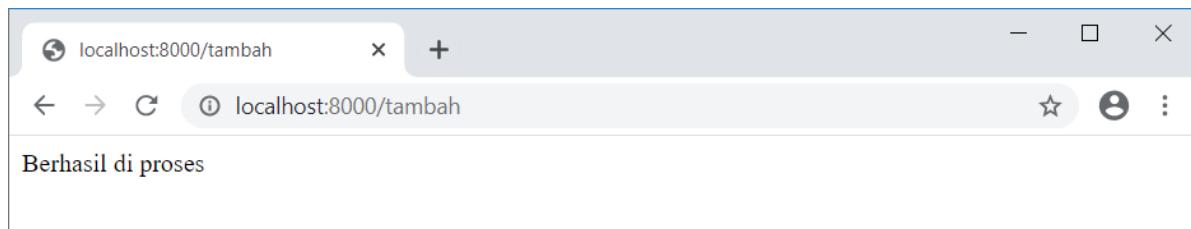
routes\web.php

```
1 ...  
2 ...  
3 Route::get('/tambah', function () {  
4     Mahasiswa::create(  
5         [  
6             'nim' => '19005011',  
7             'nama' => 'Riana Putria',  
8             'tanggal_lahir' => '2000-11-23',  
9             'ipk' => 2.9,  
10        ]  
11    );  
12  
13    return "Berhasil di proses";  
14});
```

Di sini saya menginput data menggunakan *mass assignment* ke model Mahasiswa. Perhatikan

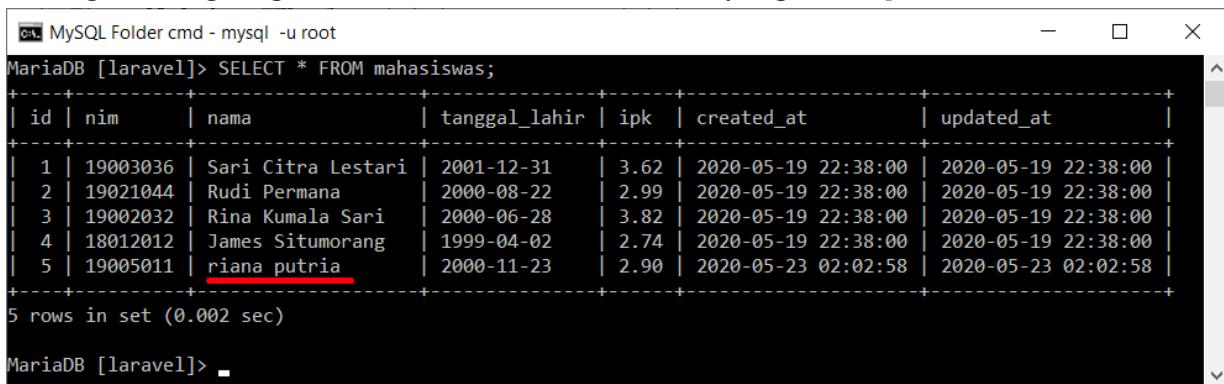
nilai untuk kolom nama berisi 'Riana Putria', dimana awal setiap kata dalam huruf besar.

Jalankan route ini dengan mengakses alamat localhost:8000/tambah.



Gambar: Proses penambahan berhasil

Sekarang cek langsung ke database untuk melihat data yang tersimpan:



id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.62	2020-05-19 22:38:00	2020-05-19 22:38:00
2	19021044	Rudi Permana	2000-08-22	2.99	2020-05-19 22:38:00	2020-05-19 22:38:00
3	19002032	Rina Kumala Sari	2000-06-28	3.82	2020-05-19 22:38:00	2020-05-19 22:38:00
4	18012012	James Situmorang	1999-04-02	2.74	2020-05-19 22:38:00	2020-05-19 22:38:00
5	19005011	riana putria	2000-11-23	2.90	2020-05-23 02:02:58	2020-05-23 02:02:58

Gambar: Isi data tabel mahasiswa

Hasilnya, nilai nama yang masuk adalah 'riana putria', yakni dalam bentuk huruf kecil. Inilah cara kerja dari mutator.

3.5. Cara Penulisan Mutator

Sebuah method mutator harus ditulis dengan format berikut:

```
public function set<NamaKolomTabel>Attribute($value)
{
    $this->attributes['nama_kolom'] = ....($value);
}
```

Penulisan nama method harus diawali dengan kata "**set**", kemudian diikuti dengan nama kolom tabel, lalu disambung dengan kata "**Attribute**".

Sama seperti accessor, penulisan kolom untuk nama method mutator disarankan ditulis dalam CamelCase, dimana karakter pertama setiap kata ditulis dalam huruf besar. Jika nama kolom terdapat spasi, spasi tersebut dibuang dan huruf pertama berikutnya ditulis dalam huruf besar.

Sebagai contoh jika ingin membuat mutator untuk kolom `nim` maka nama methodnya adalah `setNimAttribute()`, atau untuk kolom `tanggal_lahir` maka nama methodnya adalah `setTanggalLahirAttribute()`.

Sebuah method mutator bisa menerima 1 argument yang akan diisi Laravel dengan nilai asal. Dalam contoh di atas, argument ini saya tampung ke dalam parameter `$value`. Nilai asal inilah yang bisa kita proses terlebih dahulu sebelum diinput ke property `$this->attributes['nama_kolom']`.

Untuk contoh yang lebih real, saya akan pakai tabel `users` bawaan Laravel. Seperti yang pernah kita coba di bab sebelumnya, kolom yang harus diisi untuk tabel `users` ini adalah `name`, `email` dan `password`.

Khusus untuk `password`, sangat disarankan (dan memang sudah seharusnya) disimpan dalam bentuk hashing. Proses hashing dilakukan dengan menjalankan perintah `Hash::make()` milik facade class **Hash**. Berikut praktek input 1 data ke dalam tabel `users` secara langsung (tanpa menggunakan mutator):

```
routes\web.php

1 ...
2 ...
3 Route::get('/tambah-user1', function () {
4     User::create(
5         [
6             'name' => 'Rudi',
7             'email' => 'rudi@gmail.com',
8             'password' => Hash::make('rahasia'),
9         ]
10    );
11
12    return "Berhasil di proses";
13});
```

Dengan mengakses alamat `localhost:8000/tambah-user1`, user **Rudi** sudah masuk ke dalam database.

Yang akan kita lakukan adalah, memindahkan pemanggilan method `Hash::make()` ke dalam mutator agar tidak perlu ditulis setiap kali ingin melakukan input data.

Sebagai bahan percobaan, berikut route yang nantinya akan di jalankan:

routes\web.php

```
1 ...  
2 ...  
3 Route::get('/tambah-user2', function () {  
4     User::create(  
5         [  
6             'name' => 'Alex',  
7             'email' => 'alex@gmail.com',  
8             'password' => 'qwert',  
9         ]  
10    );  
11  
12    return "Berhasil di proses";  
13});
```

Perhatikan nilai untuk kolom password, tertulis 'qwert' saja. Tapi saya ingin yang tersimpan ke database sudah langsung dalam bentuk hashing. Bisakah anda coba buat mutator untuk keperluan ini? Silahkan rancang sebentar.

Baik, berikut mutator yang bisa digunakan:

app\Models\User.php

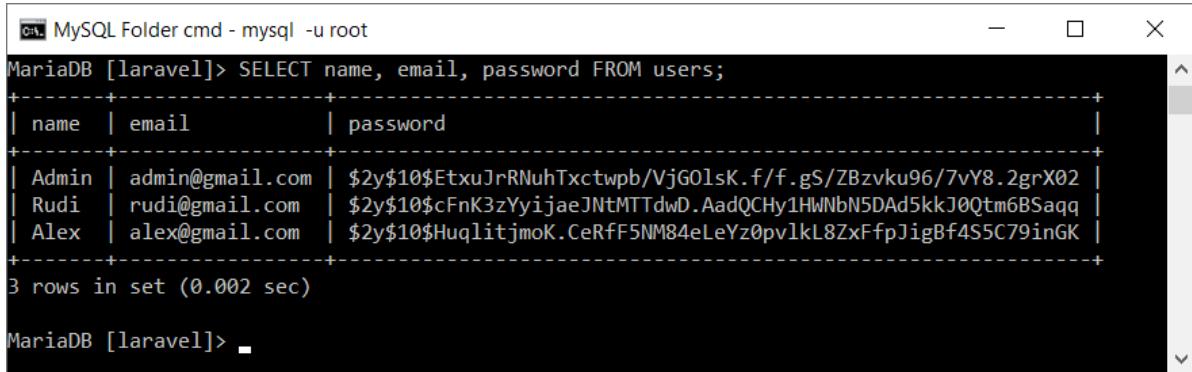
```
1 ...  
2 use Illuminate\Support\Facades\Hash;  
3 ...  
4 ...  
5     public function setPasswordAttribute($value)  
6     {  
7         $this->attributes['password'] = Hash::make($value);  
8     }
```

Karena kita sekarang membuat mutator untuk tabel `users`, maka model yang digunakan adalah `User`, bukan lagi `Mahasiswa`. Mutator diatas harus ditulis ke dalam file `app\Models\User.php`.

Secara konsep, isi mutator ini mirip seperti `setNamaAttribute()` yang sudah kita coba di model `Mahasiswa`. Dimana tinggal menginput `Hash::make($value)` ke dalam `$this->attributes['password']`.

Karena di sini terdapat perintah yang mengakses facade `Hash`, maka di bagian atas file model `User` harus ditambah proses impor `Illuminate\Support\Facades\Hash` seperti di baris 2.

Setelah selesai, silahkan akses route `localhost:8000/tambah-user2`, dan periksa hasil akhir tabel `users`:



```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT name, email, password FROM users;
+-----+-----+
| name | email      | password          |
+-----+-----+
| Admin | admin@gmail.com | $2y$10$EtxuJrRNuhTxctwpb/VjG0lsK.f/f.gS/ZBzvku96/7vY8.2grX02 |
| Rudi  | rudi@gmail.com   | $2y$10$cFnk3zYijaeJNtMTTdwD.AadQCHy1HWNbN5DAAd5kkJ0Qtm6BSaqq |
| Alex   | alex@gmail.com   | $2y$10$HuqlitjmoK.CeRff5NM84eLeYz0pvIkL8ZxFfpJigBF4S5C79inGK |
+-----+-----+
3 rows in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi data tabel users

Sip, data Alex sukses diinput dan password sudah langsung ter-hash. Inilah konsep dari mutator dalam Laravel.

3.6. Attribute Casting

Attribute casting sebenarnya bukan bagian dari accessor maupun mutator, tapi karena konsepnya mirip seperti accessor, maka akan saya bahas di sini.

Attribute casting adalah fitur untuk mengubah (*casting*) tipe data dari suatu kolom sebelum ditampilkan oleh Eloquent. Attribute casting ini bisa juga disebut sebagai versi sederhana dari accessor.

Cara pembuatan attribute casting adalah dengan menambah sebuah property `$casts` ke dalam Model. Property `$casts` ini diisi pasangan nama kolom dan tipe data dalam bentuk array.

Berikut format penulisannya:

```
protected $casts = [
    'nama_kolom' => 'tipe_data',
];
```

Sebagai contoh praktek, saya ingin mengubah (*casting*) tipe data kolom `ipk` di dalam tabel mahasiswa menjadi integer pada saat ditampilkan. Saat ini, kolom `ipk` berisi tipe data float, yakni dalam bentuk pecahan di database. Maka ketika di konversi menjadi tipe data integer, nilai pecahan di belakang koma akan dibuang.

Silahkan tambah kode berikut ke dalam model Mahasiswa:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     protected $casts = [  
4         'ipk' => 'integer'  
5     ];
```

Kemudian, akses kembali url <http://localhost:8000/tampil1>:



Gambar: Hasil casting kolom ipk ke integer

Perhatikan angka paling kanan dari setiap baris, itulah nilai ipk yang diakses dari tabel mahasiswa. Setelah proses casting ke integer, nilai ipk tampil dalam angka bulat, bukan lagi dalam bentuk pecahan.

Attribute casting mendukung berbagai casting tipe data bawaan PHP, termasuk jika kita ingin konversi ke boolean, string, float, array, object dan termasuk 'tipe data' bawaan Laravel seperti casting ke collection.

Untuk tipe data angka sendiri, terdapat juga tambahan untuk casting ke `decimal:<digits>`, dimana kita bisa konversi tipe data float menjadi angka dengan jumlah angka pecahan tertentu. Sebagai contoh, silahkan ubah nilai property `$casts` untuk kolom ipk menjadi sebagai berikut:

app\Models\Mahasiswa.php

```
1 ...  
2 ...  
3 protected $casts = [  
4     'ipk' => 'decimal:1'  
5 ];
```



Gambar: Hasil casting kolom ipk ke decimal:1

Kali ini kolom ipk akan ditampilkan dengan 1 digit pecahan. Digit pecahan ini juga dibulatkan, misalnya nilai ipk Rudi Permana di database sebenarnya 2.99, namun ketika di konversi ke

decimal:1 akan tampil sebagai 3.00.

Teknik attribute casting ini juga sudah dipakai oleh model User bawaan Laravel. Silahkan anda buka file app\Models\User.php, dan akan terlihat baris berikut:

app\Models\User.php

```
1 ...  
2 ...  
3     protected $casts = [  
4         'email_verified_at' => 'datetime',  
5     ];
```

Kode di atas dipakai untuk men-casting kolom email_verified_at di tabel users menjadi tipe data datetime PHP.

Dalam bab ini kita telah membahas konsep **Accessor**, **Mutator** dan juga **Attribute Casting** dalam Laravel. Ketiganya bisa menjadi salah satu solusi dari masalah yang akan kita temui sepanjang pembuatan kode program nantinya.

Yang perlu menjadi catatan adalah, semua teknik ini hanya berjalan jika proses input dan output dilakukan menggunakan Eloquent. Jika tabel database di akses menggunakan raw query maupun query builder, maka efek dari accessor, mutator dan attribute casting akan diabaikan.

Lanjut, kita akan bahas tentang library **Carbon** yang disediakan Laravel untuk mengolah tipe data tanggal.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

4. Carbon

Mengolah tipe data tanggal (*date and time*) termasuk materi yang gampang-gampang susah.

Gampang karena di PHP sudah ada **date** function serta **DateTime** object. Namun juga susah karena masih banyak fitur yang belum tersedia seperti menampilkan tanggal dalam bahasa Indonesia, atau menampilkan info waktu relatif seperti "1 hari yang lalu", "5 hari berikutnya", dst.

Semua ini bisa kita lakukan dengan library **Carbon** yang sudah disertakan Laravel.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

4.1. Pengertian Carbon Library

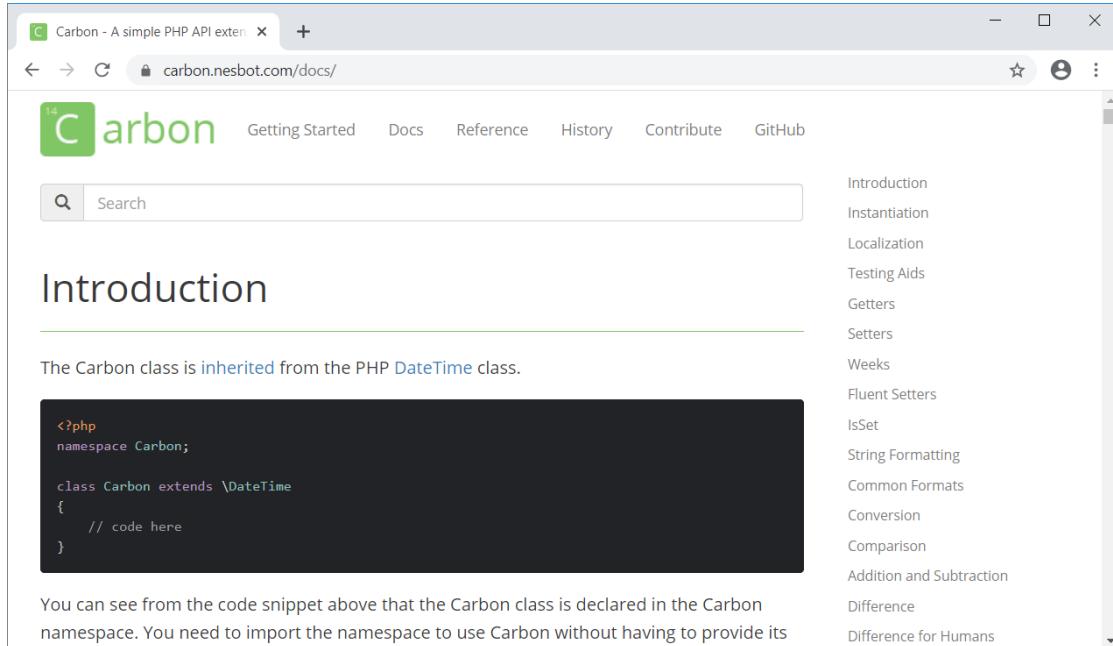
Carbon adalah nama sebuah library untuk mengolah *date and time* (tanggal dan waktu). Meskipun PHP sudah menyediakan **DateTime** object (materinya ada di buku OOP PHP Uncover), `carbon` menyediakan fitur yang lebih lengkap dan lebih mudah dipakai.

Bisa juga disebut bahwa Carbon adalah versi *upgrade* dari `DateTime` object bawaan PHP. Dan sebenarnya secara internal Carbon juga men-*extends* `DateTime` object.

Carbon sendiri merupakan library pihak ketiga yang bukan ditulis oleh tim pengembang Laravel. Seperti yang kita ketahui bersama, framework Laravel banyak mengambil *best practice* yang sudah ada di industri. Jadi daripada membuat komponen pengolahan tanggal sendiri, tim Laravel memutuskan untuk menggunakan library yang sudah ada saja.

Karena merupakan library terpisah, kita pun bisa memakai Carbon secara independen (tidak bersama Laravel). File library Carbon bisa diakses dari github.com/briannesbitt/Carbon.

Dokumentasi Carbon (yang sangat panjang) bisa diakses di carbon.nesbot.com/docs. Apa yang akan kita bahas di sini merupakan potongan kecil dari semua fitur yang disediakan carbon. Jika tertarik, anda bisa lanjut mempelajari dari dokumentasi ini.



Gambar: Tampilan dokumentasi Carbon library

Pemilihan nama **Carbon** terinspirasi dari [Radiocarbon dating](#), yakni teknik yang biasa dipakai ilmuwan untuk memperkirakan umur fosil kuno. Caranya adalah dengan menghitung jejak isotop carbon-14.

4.2. Carbon Instantiation

Sama seperti kebanyakan library PHP, **Carbon** hadir dalam bentuk sebuah class. Untuk bisa menggunakan, class tersebut harus diinstansiasi menjadi object terlebih dahulu dan disimpan dalam sebuah variabel. Dari variabel inilah nantinya kita bisa jalankan berbagai method bawaan carbon.

Di dalam Laravel, **Carbon** class berada dalam namespace `Carbon`. Sehingga jika tanpa proses import, instansiasi carbon bisa dilakukan dengan perintah berikut:

```
$carbon1 = new \Carbon\Carbon();
```

Dengan perintah di atas, variabel `$carbon1` sudah berisi object dari class **Carbon**. Jika class Carbon diinstansiasi tanpa menyertakan argument seperti ini, maka akan berisi tanggal sekarang yang ada di sistem.

Mari masuk ke contoh praktik. Silahkan ketik route berikut:

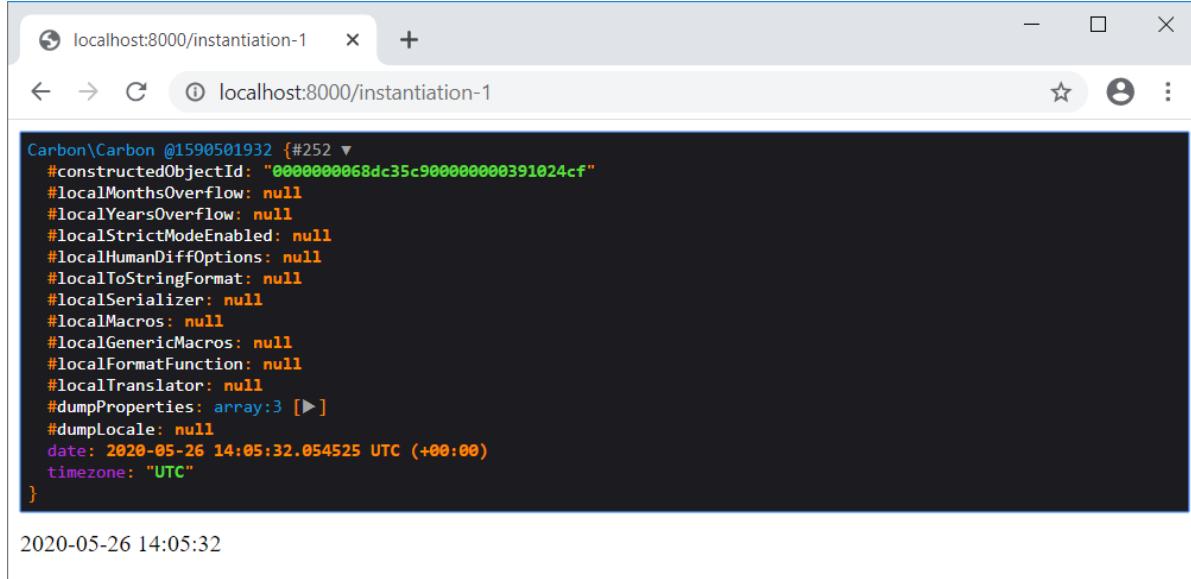
```
routes\web.php
```

```
1 Route::get('/instantiation-1', function () {
2     $carbon1 = new \Carbon\Carbon();
```

Carbon

```
3     dump($carbon1);
4     echo $carbon1;
5 );
```

Kemudian akses alamat `localhost:8000/instantiation-1` untuk melihat hasilnya:



```
Carbon\Carbon @1590501932 {#252 ▾
  #constructedObjectId: "0000000068dc35c900000000391024cf"
  #localMonthsOverflow: null
  #localYearsOverflow: null
  #localStrictModeEnabled: null
  #localHumanDiffOptions: null
  #localToStringFormat: null
  #localSerializer: null
  #localMacros: null
  #localGenericMacros: null
  #localFormatFunction: null
  #localTranslator: null
  #dumpProperties: array:3 [▶]
  #dumpLocale: null
  date: 2020-05-26 14:05:32.054525 UTC (+00:00)
  timezone: "UTC"
}

2020-05-26 14:05:32
```

Gambar: Proses instansiasi Carbon

Untuk menyederhanakan pembahasan, semua materi dalam bab ini juga saya tulis langsung di route (sama seperti bab - bab sebelumnya). Pada prakteknya nanti, proses pembuatan object Carbon biasanya di buat dari dalam Controller dan ditampilkan melalui View.

Pada saat URL '`/instantiation-1`' di akses, proses instansiasi Carbon akan berjalan dan hasilnya disimpan ke variabel `$carbon1` (baris 2).

Isi variabel `$carbon1` selanjutnya saya `dump()` di baris 3. Hasilnya terlihat bahwa variabel `$carbon1` ini berisi object **Carbon** dengan berbagai data.

Perhatikan data `date: 2020-05-26 14:05:32.834080 UTC (+00:00)`, dan `timezone: "UTC"` di bagian paling bawah. Ini adalah tanggal yang tersimpan dalam **Carbon** object yang diambil dari komputer server. Karena kita menjalankan Laravel di komputer `localhost` (server lokal), maka komputer server yang dimaksud adalah komputer kita sendiri.

Secara default, tanggal ini menggunakan waktu UTC atau GMT (Greenwich Mean Time). Jika komputer anda di set dengan waktu Indonesia, maka akan ada selisih sekitar 7 jam. Artinya waktu di komputer saya saat kode di atas dijalankan adalah pukul 21:05:32, bukan 14:05:32. Nantinya perbedaan timezone ini bisa kita atur agar sesuai dengan zona waktu Indonesia.

Lanjut, di baris 4 saya men-`echo` langsung variabel `$carbon1`. Biasanya perintah seperti ini tidak bisa dilakukan karena PHP harus mengkonversi sebuah object menjadi string. Namun

karena di dalam **Carbon** sudah terdapat magic function `__toString()`, maka tidak masalah. Hasilnya, tampil tanggal dalam format `YYYY-MM-DD HH:MM:SS`.

Kembali ke kode program, jika kita banyak melakukan proses instansiasi, lebih baik namespace Carbon di import ke dalam file saat ini. Caranya, tambah perintah `use Carbon\Carbon` di bagian paling atas file. Setelah itu, instansiasi cukup dipanggil dengan perintah `new Carbon()`:

```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use Carbon\Carbon;
5
6 ...
7 ...
8
9 Route::get('/instantiation-2', function () {
10     $carbon1 = new Carbon();
11     dump($carbon1);
12     echo $carbon1;
13 });


```

Perintah import lain yang bisa dipakai adalah `use Illuminate\Support\Carbon`. Ini karena Carbon juga disertakan sebagai kode inti dari Laravel.

Proses pembuatan **Carbon** object bisa dilakukan dengan berbagai perintah, tidak hanya `new Carbon()` saja. Setidaknya terdapat 5 cara untuk mengambil tanggal dan waktu sistem saat ini:

```
routes\web.php

1 ...
2 ...
3 Route::get('/instantiation-3', function () {
4     $carbon1 = new Carbon;
5     $carbon2 = new Carbon();
6     $carbon3 = Carbon::now();
7     $carbon4 = Carbon::parse();
8     $carbon5 = Carbon::create('now');
9
10    echo $carbon1;    echo "<hr>";    // 2020-05-26 22:57:09
11    echo $carbon2;    echo "<hr>";    // 2020-05-26 22:57:09
12    echo $carbon3;    echo "<hr>";    // 2020-05-26 22:57:09
13    echo $carbon4;    echo "<hr>";    // 2020-05-26 22:57:09
14    echo $carbon5;    echo "<hr>";    // 2020-05-26 22:57:09
15 });


```

Di sini saya membuat 5 buah variabel mulai dari `$carbon1` hingga `$carbon5`. Setiap variabel akan berisi Carbon object yang diambil dari tanggal dan waktu sistem saat ini.

Dua perintah pertama menggunakan keyword `new` untuk membuat Carbon object. Sedangkan

3 perintah berikutnya mengakses static method Carbon::now(), Carbon::parse() dan Carbon::create('now'). Anda bebas ingin memakai versi yang disukai.

Instansiasi Untuk Tanggal dan Waktu Tertentu

Pada prakteknya, kita akan sering membuat Carbon object untuk tanggal dan waktu tertentu. Tidak masalah, tinggal input nilai tersebut sebagai argument pada saat proses instansiasi.

Dan seperti yang bisa di tebak, terdapat banyak cara untuk keperluan ini:

routes\web.php

```

1 ...
2 ...
3 Route::get('/instantiation-4', function () {
4     $carbon1 = new Carbon('1 January 2021');
5     $carbon2 = new Carbon('12:30:15');
6     $carbon3 = new Carbon('01-01-2021 12:30:15');
7
8     $carbon4 = Carbon::parse('1 January 2021 12:30:15');
9     $carbon5 = Carbon::create('1 January 2021');
10    $carbon6 = Carbon::create('1 January 2021 12:30:15');
11
12    $carbon7 = Carbon::createFromDate(2021, 1, 1);
13    $carbon8 = Carbon::createFromTime(12, 30, 15);
14    $carbon9 = Carbon::createFromTimestamp(1609504215);
15
16    $carbon10 = Carbon::create(2021, 1, 1, 12, 30, 15);
17    $carbon11 = Carbon::create(2021, 1, 1);
18
19    echo $carbon1;    echo "<hr>";    // 2021-01-01 00:00:00
20    echo $carbon2;    echo "<hr>";    // 2020-05-26 12:30:15
21    echo $carbon3;    echo "<hr>";    // 2021-01-01 12:30:15
22    echo $carbon4;    echo "<hr>";    // 2021-01-01 12:30:15
23    echo $carbon5;    echo "<hr>";    // 2021-01-01 00:00:00
24    echo $carbon6;    echo "<hr>";    // 2021-01-01 12:30:15
25    echo $carbon7;    echo "<hr>";    // 2021-01-01 23:21:26
26    echo $carbon8;    echo "<hr>";    // 2020-05-26 12:30:15
27    echo $carbon9;    echo "<hr>";    // 2021-01-01 12:30:15
28    echo $carbon10;   echo "<hr>";    // 2021-01-01 12:30:15
29    echo $carbon11;   echo "<hr>";    // 2021-01-01 00:00:00
30 });

```

Beragamnya cara pembuatan tanggal dan waktu memang membuat bingung, tapi sekaligus menawarkan fleksibilitas tergantung masalah yang dihadapi nantinya.

Kode program di baris 4 - 6 merupakan cara paling dasar, yakni dengan menginput string tanggal, waktu, serta tanggal dan waktu sekaligus. Karena **Carbon** men-extends DateTime object, maka format string yang didukung juga cukup beragam, sama seperti argument yang bisa diterima oleh **DateTime** object PHP.

Dalam contoh ini saya menginput 3 string yang disimpan ke dalam 3 variabel berbeda, yakni

tanggal '1 January 2021' ke dalam variabel \$carbon1, pukul '12:30:15' ke dalam variabel \$carbon2 dan '01-01-2021 12:30:15' ke dalam variabel \$carbon3.

Jika argument yang diinput tidak disertai waktu, maka akan dianggap 00:00:00 seperti hasil \$carbon1. Namun jika argument tidak disertai tanggal (hanya waktu saja), maka akan diambil dari tanggal di sistem saat ini seperti hasil \$carbon2.

Format argument yang sama juga bisa diinput ke dalam static method Carbon::parse() dan Carbon::create() seperti di baris 8 – 10.

Di baris 12 – 14 saya menggunakan 3 method berbeda. Kali ini argument yang diinput bukan lagi berbentuk string, tapi langsung berupa angka sesuai dengan nama method.

Method Carbon::createFromDate() bisa menerima 3 argument tanggal dalam format YYYY, MM, DD. Sedangkan untuk method Carbon::createFromTime() bisa diinput dengan 3 argument waktu dalam format HH, MM, SS.

Perhatikan isi variabel \$carbon7, meskipun didapat dari Carbon::createFromDate(2021, 1, 1) tanpa menyertakan waktu, yang tersimpan bukan 00:00:00, tapi waktu sistem saat ini.

Kemudian untuk method Carbon::createFromTimestamp() bisa menerima argument berupa nilai *timestamp*.

Terakhir, method Carbon::create() bisa menerima argument dalam bentuk (YYYY, MM, DD, HH, MM, SS). Namun tidak semua argument harus ditulis, bisa saja kita mengisi bagian tanggal tanpa menginput waktu seperti \$carbon11.

Instansiasi Relatif ke Hari Ini

Proses instansiasi juga bisa dilakukan dengan perhitungan relatif ke hari ini. Berikut contoh kode programnya:

```
routes\web.php

1 ...
2 ...
3 Route::get('/instantiation-5', function () {
4     $carbon1 = Carbon::now();
5     $carbon2 = Carbon::today();
6     $carbon3 = Carbon::yesterday();
7     $carbon4 = Carbon::tomorrow();
8
9     $carbon5 = Carbon::create('now');
10    $carbon6 = Carbon::create('today');
11    $carbon7 = Carbon::create('yesterday');
12    $carbon8 = Carbon::create('tomorrow');
13
14    echo $carbon1; echo "<br>"; // 2020-05-27 01:28:03
15    echo $carbon2; echo "<br>"; // 2020-05-27 00:00:00
16    echo $carbon3; echo "<br>"; // 2020-05-26 00:00:00
```

Carbon

```
17 echo $carbon4; echo "<br>"; // 2020-05-28 00:00:00
18 echo $carbon5; echo "<br>"; // 2020-05-27 01:28:03
19 echo $carbon6; echo "<br>"; // 2020-05-27 00:00:00
20 echo $carbon7; echo "<br>"; // 2020-05-26 00:00:00
21 echo $carbon8; echo "<br>"; // 2020-05-28 00:00:00
22});
```

Pada saat kode di atas dijalankan, tanggal dan waktu di komputer saya adalah 2020-05-27 08:28:03. Jadi waktu *yesterday* dan *tomorrow* akan relatif ke nilai tersebut. Khusus untuk waktu, terdapat perbedaan zona waktu / timezone sehingga yang tercatat adalah 01:28:03 (dikurangi 7 jam).

Cara lain untuk membuat tanggal dan waktu relatif adalah menginput manual selisih yang ada dalam bentuk hari, bulan, tahun, jam, menit dan detik:

routes\web.php

```
1 ...
2 ...
3 Route::get('/instantiation-6', function () {
4     $carbon1 = Carbon::now();
5     $carbon2 = Carbon::create('+1 hour');
6     $carbon3 = Carbon::create('+4 day');
7     $carbon4 = Carbon::create('-20 day');
8     $carbon5 = Carbon::create('+4 month -1 hour');
9     $carbon6 = Carbon::create('+5 day 6 month 1 year');
10
11    echo $carbon1; echo "<br>"; // 2020-05-27 01:28:03
12    echo $carbon2; echo "<br>"; // 2020-05-27 02:28:03
13    echo $carbon3; echo "<br>"; // 2020-05-31 01:28:03
14    echo $carbon4; echo "<br>"; // 2020-05-07 01:28:03
15    echo $carbon5; echo "<br>"; // 2020-09-27 00:28:03
16    echo $carbon6; echo "<br>"; // 2021-12-02 01:28:03
17});
```

Untuk membuat waktu relatif yang diinput manual, bisa menggunakan perintah `Carbon::create()`. Tanda positif menunjukkan tanggal dan waktu ke depan, dan tanda negatif untuk waktu sebelumnya.

Kata yang dipakai adalah kata dalam bahasa inggris seperti *day*, *month*, *year*, *hour*, *minute* dan *second*. Kata ini juga bisa ditulis secara singular maupun plural seperti *days*, *months* atau *years*. Dengan teknik ini, kita bisa dengan mudah menampilkan tanggal 1 tahun ke depan dan akan selalu update sesuai tanggal yang ada di sistem.

4.3. Getter

Getter adalah sebutan untuk kumpulan method yang dipakai untuk menampilkan tanggal dan waktu. Carbon menyediakan berbagai format output tergantung kebutuhan.

Pertama, kita akan lihat bentuk yang paling sederhana:

routes\web.php

```

1 ...
2 ...
3 Route::get('/getter-1', function () {
4     $carbon1 = Carbon::create(2021, 1, 1, 12, 30, 15);
5
6     echo $carbon1;                      echo "<br>";    // 2021-01-01 12:30:15
7     echo $carbon1->toTimeString();      echo "<br>";    // 12:30:15
8     echo $carbon1->toDateString();     echo "<br>";    // 2021-01-01
9     echo $carbon1->format('d M Y');   echo "<br>";    // 01 Jan 2021
10 });

```

Di baris 4 saya menyiapkan variabel \$carbon1 yang berisi tanggal 1 Januari 2021, pukul 12:30:15. Ini menjadi contoh tanggal yang akan kita coba tampilkan.

Cara output pertama adalah langsung men-echo \$carbon1 seperti di baris 6. Hasilnya, tampil tanggal dalam format YYYY-MM-DD HH:MM:SS. Format seperti ini biasanya hanya kita pakai untuk proses debugging atau pencarian kesalahan. Jika ingin menampilkan tanggal saja atau waktu saja, bisa menggunakan method `toTimeString()` dan `toDateString()` seperti di baris 7-8.

Di baris 9 terdapat pemanggilan method `$carbon1->format('d M Y')`. Method ini menerima argument yang sama dengan pola format function `date()` di PHP. Format 'd M Y' akan menampilkan tanggal seperti 01 Jan 2021. Lebih lanjut tentang format ini bisa diakses dari www.php.net/manual/en/function.date.php.

Penulisan pola format seperti di method `format()` memang sudah umum, tapi cukup susah menghafal format tersebut. Carbon memberikan cara alternatif menggunakan method `isoFormat()`:

routes\web.php

```

1 ...
2 ...
3 Route::get('/getter-2', function () {
4     $carbon1 = Carbon::create(2021, 1, 1, 12, 30, 15);
5
6     echo $carbon1;                      echo "<br>";
7     echo $carbon1->isoFormat('D/M/YY HH:mm'); echo "<br>";
8     echo $carbon1->isoFormat('ddd');       echo "<br>";
9     echo $carbon1->isoFormat('ddd, D MMMM YYYY HH:mm'); echo "<br>";
10    echo $carbon1->isoFormat('LLL');       echo "<br>";
11 });

```

Hasil kode program:

2021-01-01 12:30:15
1/21 12:30

Friday

Friday, 1 January 2021 12:30

Friday, January 1, 2021 12:30 PM

Pola untuk method isoFormat() ini lebih mudah di tebak karena menggunakan penulisan umum seperti D, MM atau YYYY. Berikut daftar beberapa pola yang sering dipakai:

- ◆ **D**: Urutan hari dalam 1 bulan (dari 1 sampai 31)
- ◆ **DD**: Urutan hari dalam 1 bulan dengan awalan 0 (dari 01 sampai 31)
- ◆ **dddd**: Nama hari (dari Sunday to Saturday), nantinya bisa di translate ke bahasa lain
- ◆ **M**: Urutan bulan dalam 1 tahun (dari 1 sampai 12)
- ◆ **MM**: Urutan bulan dalam 1 tahun dengan awalan 0 (dari 01 sampai 12)
- ◆ **MMM**: Singkatan nama bulan (dari Jan sampai Dec)
- ◆ **MMMM**: Nama bulan lengkap (dari January sampai December)
- ◆ **YY**: Dua digit tahun (dari 00 sampai 99)
- ◆ **YYYY**: Empat digit tahun (dari 0000 sampai 9999)
- ◆ **H**: Jam (dari 0 sampai 23)
- ◆ **HH**: Jam dengan awalan 0 (dari 00 sampai 23)
- ◆ **h**: Jam dalam format 12 jam (dari 0 sampai 12)
- ◆ **hh**: Jam dalam format 12 jam dengan awalan 0 (dari 0 sampai 12)
- ◆ **A**: AM atau PM (dikombinasikan dengan format h atau hh)
- ◆ **a**: am atau pm (dikombinasikan dengan format h atau hh)
- ◆ **m**: Menit (dari 0 sampai 59)
- ◆ **mm**: Menit dengan awalan 0 (dari 00 sampai 59)
- ◆ **s**: Detik (dari 0 sampai 59)
- ◆ **ss**: Detik dengan awalan 0 (dari 00 sampai 59)

Selain kode di atas, masih banyak format lain yang bisa dipakai. Lengkapnya bisa dilihat ke carbon.nesbot.com/docs/#iso-format-available-replacements.

Jika kita ingin mengakses komponen tanggal dan waktu secara individu, tersedia juga method terpisah seperti contoh berikut:

routes\web.php

```

1 ...  

2 ...  

3 Route::get('/getter-3', function () {  

4     $carbon1 = Carbon::create(2021, 4, 25, 12, 30, 15);  


```

Carbon

```
5
6 echo $carbon1->second;      echo "<hr>";    // 15
7 echo $carbon1->minute;      echo "<hr>";    // 30
8 echo $carbon1->hour;        echo "<hr>";    // 12
9 echo $carbon1->day;         echo "<hr>";    // 25
10 echo $carbon1->month;       echo "<hr>";    // 4
11 echo $carbon1->year;        echo "<hr>";    // 2021
12 echo $carbon1->quarter;     echo "<hr>";    // 2
13 echo $carbon1->dayName;     echo "<hr>";    // Sunday
14 echo $carbon1->monthName;   echo "<hr>";    // April
15
16 $carbon2 = Carbon::create(1993, 4, 25, 12, 30, 15);
17 echo $carbon2->age;          echo "<hr>";    // 27
18 echo $carbon2->timespan();   echo "<hr>";    // 27 years, 1 month, 1 day, 16 hours, 22 minutes, 38 seconds
19
20});
```

Yang cukup menarik adalah method `age()` dan `timespan()`, dimana kita bisa menghitung umur atau jangka waktu sampai tingkat ketelitian detik.

Namun method getter yang paling menarik adalah `diffForHumans()`. Berikut contoh penggunaannya:

routes\web.php

```
1 ...
2 ...
3 Route::get('/getter-4', function () {
4     $carbon1 = Carbon::now();
5     $carbon2 = Carbon::create('-1 hour');
6     $carbon3 = Carbon::create('+4 hour');
7     $carbon4 = Carbon::create('-2 week');
8     $carbon5 = Carbon::create(2020, 6, 1);
9     $carbon6 = Carbon::create(2021, 1, 1, 12, 30, 15);
10    $carbon7 = Carbon::create(1945, 8, 17);
11
12    echo $carbon1;                  echo "<hr>";
13    echo $carbon2->diffForHumans(); echo "<hr>";
14    echo $carbon3->diffForHumans(); echo "<hr>";
15    echo $carbon4->diffForHumans(); echo "<hr>";
16    echo $carbon5->diffForHumans(); echo "<hr>";
17    echo $carbon6->diffForHumans(); echo "<hr>";
18    echo $carbon7->diffForHumans(); echo "<hr>";
19});
```

Hasil kode program:

```
2020-05-27 03:29:25
1 hour ago
3 hours from now
2 weeks ago
4 days from now
7 months from now
74 years ago
```

Antara baris 5 sampai 10 saya membuat berbagai tanggal dan waktu yang masing-masingnya disimpan ke dalam variabel berbeda. Dari setiap variabel kemudian dijalankan method `diffForHumans()`.

Method `diffForHumans()` menampilkan perbedaan waktu dengan kata-kata yang sering dipakai sehari-hari, seperti *1 hour ago*, *2 weeks ago* atau *4 days from now*. Ini sangat menarik dan sering kita jumpai pada website social media, terutama untuk menandakan kapan seseorang online atau kapan sebuah status di update. Nantinya kata-kata ini juga bisa di translate ke bahasa Indonesia.

4.4. Setter

Kebalikan dari getter, **Setter** adalah kumpulan method yang dipakai untuk men-set atau mengubah tanggal dan waktu dari sebuah Carbon object. Ini juga bisa dilakukan dengan banyak cara.

Cara pertama adalah dengan menginput nilai tanggal dan waktu sebagai property:

```
routes\web.php

1 ...  
2 ...  
3 Route::get('/setter-1', function () {  
4     $carbon1 = Carbon::now();  
5     echo $carbon1; echo "<br>"; // 2020-05-27 05:19:36  
6  
7     $carbon1->second = 15;  
8     $carbon1->minute = 30;  
9     $carbon1->hour = 12;  
10    $carbon1->day = 25;  
11    $carbon1->month = 4;  
12    $carbon1->year = 2021;  
13  
14    echo $carbon1; // 2021-04-25 12:30:15  
15});
```

Di baris 4 variabel `$carbon1` saya instansiasi dengan tanggal saat ini. Tanggal tersebut kemudian ditimpa dengan mengisi property `second`, `minute`, `hour`, `day`, `month` dan `year` antara baris 7 – 12. Hasilnya, tanggal dan waktu `$carbon1` sudah berubah seperti tampilan baris 14.

Cara selanjutnya adalah dengan beberapa method berikut:

```
routes\web.php

1 ...  
2 ...  
3 Route::get('/setter-2', function () {  
4     $carbon1 = Carbon::now();  
5     echo $carbon1; echo "<br>"; // 2020-05-27 09:41:14
```

Carbon

```
6
7     $carbon1->setDay(25);
8     $carbon1->setMonth(4);
9     $carbon1->setYear(2021);
10    echo $carbon1; echo "<br>";      // 2021-04-25 09:41:14
11
12    $carbon1->set('day',17);
13    $carbon1->set('month',8);
14    $carbon1->set('year',1945);
15    echo $carbon1; echo "<br>";      // 1945-08-17 09:41:14
16
17    $carbon1->setTime(10,15,50);
18    $carbon1->setDate(2000,10,10);
19    echo $carbon1; echo "<br>";      // 2000-10-10 10:15:50
20
21    $carbon1->setDateTime(2021, 4, 25, 12, 30, 15);
22    echo $carbon1; echo "<br>";      // 2021-04-25 12:30:15
23});
```

Dalam kode ini terdapat 4 cara lanjutan untuk mengubah nilai tanggal yang tersimpan di Carbon object.

Di baris 7 – 9 saya menggunakan method `setDay()`, `setMonth()` dan `setYear()` untuk mengubah tanggal. Nilai tanggal diinput sebagai argument dari method-method ini. Dilihat dari namanya, bisa di tebak terdapat method lanjutan seperti `setHour()`, `setMinute()` dan `setSecond()`.

Variasi penulisan lain ada di baris 12 – 14, yakni method `set()` dengan 2 argument. Argument pertama diisi dengan string komponen tanggal seperti 'day', 'month', 'year', 'hour', 'minute' atau 'second' serta argument kedua berupa nilai dari komponen tersebut.

Proses setter juga bisa dilakukan dengan method `setTime()` dan `setDate()` seperti di baris 17 – 18. Sesuai namanya, masing-masing method ini menerima argument berupa 3 komponen waktu dan 3 komponen tanggal. Jika ingin langsung men-set 6 komponen sekaligus, bisa memakai method `setDateTime()` seperti baris 21.

Method setter terakhir yang akan kita bahas disebut dalam dokumentasi sebagai **fluent setter**, yakni cara mengubah nilai Carbon object dengan men-chaining pemanggilan method:

routes\web.php

```
1 ...
2 ...
3 Route::get('/setter-3', function () {
4     $carbon1 = Carbon::now();
5     echo $carbon1; echo "<br>";      // 2020-05-28 00:10:58
6
7     $carbon1->year(2021)->month(8)->day(17)->hour(10)->minute(50)->second(10);
8     echo $carbon1; echo "<br>";      // 2021-08-17 10:50:10
9});
```

Perintah *fluent* yang dimaksud ada di baris 7, dimana variabel \$carbon1 di-chaining dengan pemanggilan method `year()`, `month()`, `day()`, `hour()`, `minute()` dan `second()`.

Beragamnya pilihan method setter ini memberikan banyak alternatif yang bisa kita pakai.

Selain itu nama method juga bisa ditulis dalam bentuk jamak dengan akhiran 's' seperti `days()` atau `setDays()`.

4.5. Addition dan Subtraction

Addition dan **Subtraction** adalah kumpulan method untuk menambah serta mengurangi nilai tanggal dan waktu yang sudah tersimpan dalam Carbon object. Berikut contoh penggunaannya:

routes\web.php

```

1  ...
2  ...
3 Route::get('/add-sub', function () {
4     $carbon1 = Carbon::create(2021, 1, 1, 12, 30, 15);
5     echo $carbon1; echo "<br>";      // 2021-01-01 12:30:15
6
7     $carbon1->addDay();
8     echo $carbon1; echo "<br>";      // 2021-01-02 12:30:15
9
10    $carbon1->addDay(40);
11    echo $carbon1; echo "<br>";      // 2021-02-11 12:30:15
12
13    $carbon1->addMonth(6);
14    echo $carbon1; echo "<br>";      // 2021-08-11 12:30:15
15
16    $carbon1->subMonth(4);
17    echo $carbon1; echo "<br>";      // 2021-04-11 12:30:15
18
19    $carbon1->subYear(2);
20    echo $carbon1; echo "<br>";      // 2019-04-11 12:30:15
21
22    $carbon2 = Carbon::create(2021, 1, 1, 12, 30, 15);
23    echo $carbon2; echo "<br>";      // 2021-01-01 12:30:15
24
25    $carbon2->addWeekday(3);
26    echo $carbon2; echo "<br>";      // 2021-01-06 12:30:15
27
28    $carbon2->addHour(3)->addMinute(15)->addSecond(13);
29    echo $carbon2; echo "<br>";      // 2021-01-06 15:45:28
30 });

```

Di baris 4 saya membuat Carbon object untuk tanggal 2021-01-01 12:30:15 dan menyimpannya ke dalam variabel \$carbon1.

Ketika dijalankan method `addDay()` seperti di baris 7, itu akan menambah 1 hari ke dalam `$carbon1`. Method `addDay()` juga bisa menerima sebuah argument berupa jumlah hari seperti di baris 10. Jika penambahan ini melebihi jumlah hari dalam 1 bulan, Carbon akan mengkonversinya ke bulan berikutnya.

Selain `addDay()`, juga tersedia method penambahan lain, diantaranya `AddMonth()` seperti di baris 6, `addYear()`, `addHour()`, `addMinute()` dan juga `addSecond()`.

Proses pengurangan atau *subtraction* dilakukan dengan method yang memiliki awalan 'sub', seperti `subDay()`, `subMonth()`, `subYear()`, `subHour()`, `subMinute()` dan `subSecond()`. Dua diantaranya di contohkan pada baris 16 dan 19.

Selain per komponen, juga terdapat method khusus seperti `addWeekday()` seperti di baris 25. Ini akan menambah tanggal sebanyak jumlah hari kerja. Hari kerja di sini adalah hari senin – jumat. Method ini sangat cocok untuk sistem penanggalan kerja dimana biasanya kantor hanya beroperasi 5 hari selama 1 minggu.

Terakhir di baris 28 kita bisa men-*chaining* berbagai method *addition* dan *subtraction* dalam 1 perintah.

4.6. Comparison

Operasi lain yang sering dilakukan untuk data tanggal adalah membandingkan apakah tanggal yang satu didahului oleh tanggal yang lain. Carbon menyediakan beberapa operasi **comparison** untuk keperluan ini.

Metode pertama yang bisa dipakai adalah menggunakan operator perbandingan yang sudah ada di PHP. Berikut contoh penggunaannya:

`routes\web.php`

```

1 ...
2 ...
3 Route::get('/comparison-1', function () {
4     $carbon1 = Carbon::create(2021, 1, 1);
5     $carbon2 = Carbon::create(2020, 1, 1);
6
7     dump($carbon1 == $carbon2);      // false
8     dump($carbon1 != $carbon2);      // true
9     dump($carbon1 > $carbon2);      // true
10    dump($carbon1 < $carbon2);      // false
11    dump($carbon1 >= $carbon2);     // true
12    dump($carbon1 <= $carbon2);     // false
13
14    dump( $carbon1->greaterThan($carbon2) ); // true
15    dump( $carbon1->gt($carbon2) );           // true
16
17    dump( $carbon1->lessThan($carbon2) );      // false
18    dump( $carbon1->lt($carbon2) );            // false

```

Carbon

```
19});
```

Di baris 4 dan 5 saya membuat 2 buah Carbon object yang masing-masingnya diisi dengan 2 tanggal yang berbeda. Variabel \$carbon1 berisi tanggal 1-1-2021 dan variabel \$carbon2 berisi tanggal 1-1-2020.

Saya yakin anda sudah bisa mengambil kesimpulan dari proses perbandingan antara baris 7 – 12. Hasilnya akhir dari operasi ini berupa boolean **true** atau **false**.

Sebagai alternatif penulisan, tersedia juga method `greaterThan()` dan `gt()` untuk memeriksa apakah tanggal yang tersimpan di variabel saat ini lebih besar daripada tanggal yang diinput sebagai argument. Misalnya jika `$carbon1->greaterThan($carbon2)` menghasilkan **true**, maka tanggal yang tersimpan di `$carbon1` lebih besar daripada tanggal yang ada di `$carbon2`.

Begitu juga dengan method `lessThan()` dan `lt()` yang bisa dipakai untuk memeriksa apakah tanggal yang tersimpan di variabel saat ini lebih kecil daripada tanggal yang diinput sebagai argument.

Proses perbandingan lain yang bisa kita lakukan adalah memeriksa apakah sebuah tanggal berada di antara 2 tanggal lain. Carbon menyediakan method `between()` untuk keperluan ini:

routes\web.php

```
1 ...
2 ...
3 Route::get('/comparison-2', function () {
4     $carbon1 = Carbon::create(2021, 1, 1);
5     $carbon2 = Carbon::create(2020, 1, 1);
6     $carbon3 = Carbon::create(2019, 1, 1);
7
8     dump( $carbon1->between($carbon2, $carbon3) );    // false
9     dump( $carbon2->between($carbon1, $carbon3) );    // true
10
11    dump( Carbon::create(2020, 1, 1)->between($carbon1, $carbon2) );    // true
12    dump( Carbon::create(2020, 1, 1)->between($carbon1, $carbon2, false) );
13    // false
14});
```

Di baris 4 – 6 saya membuat 3 buah object Carbon untuk tanggal yang berbeda.

Method `between()` butuh minimal 2 argument, yakni tanggal awal dan tanggal akhir. Perintah di baris 8 artinya saya ingin memeriksa apakah tanggal yang tersimpan di variabel `$carbon1` berada di antara `$carbon2` dan `$carbon3`.

Sedangkan perintah di baris 9 artinya saya ingin memeriksa apakah tanggal yang tersimpan di variabel `$carbon2` berada di antara `$carbon1` dan `$carbon3`. Hasil akhirnya berupa boolean **true** atau **false**.

Merthod `between()` juga bisa dipanggil langsung dari Carbon object seperti di baris 11, dimana tidak harus disimpan ke dalam variabel terlebih dahulu.

Secara default, method `between()` akan menghasilkan nilai `true` meskipun tanggal yang dibandingkan sama. Ini dibuktikan dari perintah di baris 11, yang meskipun `Carbon::create(2020, 1, 1)` sama dengan isi tanggal di `$carbon1`, hasilnya adalah `true`.

Jika kita ingin membatasi agar tanggal yang diperiksa harus lebih besar, maka bisa tambah nilai `false` sebagai argument ketiga pada saat menulis method `between()`. Dengan tambahan ini, maka jika tanggal yang dibandingkan sama, hasilnya akan `false` dan baru `true` jika tanggal tersebut lebih besar.

Proses perbandingan lain adalah untuk memeriksa nama hari, seperti contoh berikut:

`routes\web.php`

```

1 ...
2 ...
3 Route::get('/comparison-3', function () {
4     $carbon1 = Carbon::create(2021, 1, 1);
5
6     dump($carbon1->isWeekday());           // true
7     dump($carbon1->isWeekend());          // false
8     dump($carbon1->isMonday());           // false
9     dump($carbon1->isTuesday());          // false
10    dump($carbon1->isWednesday());        // false
11    dump($carbon1->isThursday());         // false
12    dump($carbon1->isFriday());           // true
13    dump($carbon1->isSaturday());         // false
14    dump($carbon1->isSunday());           // false
15 });

```

Method ini memang cukup unik dan sangat membantu ketika dibutuhkan, misalnya method `isWeekday()` dan `isWeekend()` bisa dipakai untuk mencari tau apakah tanggal saat ini masuk ke dalam hari kerja atau bukan hari kerja.

Proses perbandingan ini juga bisa dilakukan untuk tanggal relative seperti contoh berikut:

`routes\web.php`

```

1 ...
2 ...
3 Route::get('/comparison-4', function () {
4     $carbon1 = Carbon::create('+ 1 day');
5
6     echo $carbon1;                         // 2020-06-01 02:23:36
7     dump($carbon1->isTomorrow());         // true
8     dump($carbon1->isNextDay());          // true
9     dump($carbon1->isFuture());           // true
10    dump($carbon1->isPast());             // false
11    dump($carbon1->isLeapYear());         // true
12    dump($carbon1->isNextYear());          // false
13 });

```

Di baris 4 saya membuat carbon Object untuk tanggal esok hari ('+ 1 day'). Kemudian

memeriksa apakah tanggal tersebut ada di hari berikutnya, hari sebelumnya, atau apakah itu berada di tahun kabisat (leap year) atau berada di tahun depan.

Selain apa yang kita bahas di sini, masih ada beberapa method perbandingan lain yang sangat spesifik, seperti `isSameQuarter()` untuk memeriksa apakah tanggal yang diperiksa dalam quartal yang sama, atau `isLastOfMonth()` untuk memeriksa apakah tanggal tersebut merupakan tanggal terakhir di sebuah bulan. Daftar lengkapnya bisa anda pelajari dari dokumentasi Carbon: carbon.nesbot.com/docs/#api-comparison.

4.7. Difference

Jika di method comparison hanya menghasilkan true atau false, maka di method **difference** kita bisa mencari berapa banyak perbedaan antara tanggal yang satu dengan tanggal yang lain. Berikut contoh penggunaannya:

routes\web.php

```

1 ...  

2 ...  

3 Route::get('/diff', function () {  

4     $carbon1 = Carbon::create(2021, 1, 1);  

5     $carbon2 = Carbon::create(1945, 8, 17);  

6  

7     echo $carbon1->diffInYears($carbon2);    echo "<br>"; // 75  

8     echo $carbon1->diffInMonths($carbon2);   echo "<br>"; // 904  

9     echo $carbon1->diffInDays($carbon2);      echo "<br>"; // 27531  

10 });

```

Hasil akhir dari perbedaan antara kedua tanggal tergantung kepada method yang dipakai. Jika menggunakan `diffInYears()`, maka hasilnya berupa angka dalam satuan tahun. Jika yang dijalankan adalah `diffInMonths()` maka hasilnya berupa angka dalam satuan bulan, dan begitu juga dengan `diffInDays()` yang mengasilkan perbedaan dalam satuan hari.

Karena di sini yang dicari adalah perbedaan (difference), maka tidak masalah mau dipanggil sebagai `$carbon1->diffInYears($carbon2)`, maupun `$carbon2->diffInYears($carbon1)`, hasilnya akan tetap 75.

Selain 3 method di atas, tersedia juga method `diffInHours()`, `diffInMinutes()` dan `diffInSeconds()` jika kita ingin mencari perbedaan waktu. Atau jika ingin mencari perbedaan yang lebih spesifik, tersebut method `diffInWeekdays()`, `diffInWeekendDays()`, `diffInWeeks()` dan `diffInQuarters()`.

Atau tersedia juga method `diffForHumans()`:

routes\web.php

```

1 ...  

2 ...

```

Carbon

```
3 // Difference for Humans
4 Route::get('/diff-for-human', function () {
5     $carbon1 = Carbon::create(2021, 1, 1, 21, 30, 0);
6
7     echo $carbon1->diffForHumans($carbon1);           echo "<hr>";
8     echo $carbon1->diffForHumans(Carbon::create(2021, 1, 10));    echo "<hr>";
9     echo $carbon1->diffForHumans(Carbon::create(2021, 2, 10));    echo "<hr>";
10    echo $carbon1->diffForHumans(Carbon::today());        echo "<hr>";
11});
```

Hasil kode program:

```
1 second before
1 week before
1 month before
7 months after
```

Method `diffForHumans()` otomatis mencari tampilan paling pas dari 2 tanggal. Misalnya jika tidak terlalu jauh, akan ditampilkan seperti `1 second before` atau `1 week before`, namun jika perbedaan tanggal cukup jauh, akan ditampilkan dalam satuan bulan atau tahun.

4.8. Modifier

Modifier adalah method yang dipakai untuk memodifikasi tanggal yang sudah tersimpan di Carbon object. Sekilas ini mirip seperti Setter, tapi modifier memodifikasi tanggal sekaligus, bukan per komponen seperti setter.

Berikut contoh penggunaan dari method modifier:

routes\web.php

```
1 ...
2 ...
3 Route::get('/modifier', function () {
4     $carbon1 = Carbon::create(2021, 1, 1, 10, 10, 10);
5     echo $carbon1->endOfDay();      echo "<hr>";    // 2021-01-01 23:59:59
6     echo $carbon1;                 echo "<hr>";    // 2021-01-01 23:59:59
7
8     echo Carbon::create(2021, 1, 1)->startOfDay();      echo "<hr>";
9     // 2021-01-01 00:00:00
10
11    echo Carbon::create(2021, 1, 1)->startOfWeek();      echo "<hr>";
12    // 2020-12-28 00:00:00
13
14    echo Carbon::create(2021, 1, 1)->endOfWeek();       echo "<hr>";
15    // 2021-01-03 23:59:59
16
17    echo Carbon::create(2021, 1, 1)->nextWeekday();      echo "<hr>";
18    // 2021-01-04 00:00:00
19
20    echo Carbon::create(2021, 1, 1)->endOfMonth();       echo "<hr>";
```

Carbon

```
21 // 2021-01-31 23:59:59
22
23 echo Carbon::create(2021, 1, 1)->endOfDay();      echo "<hr>";
24 // 2021-12-31 23:59:59
25
26 echo Carbon::create(2021, 1, 1)->endOfDecade();    echo "<hr>";
27 // 2029-12-31 23:59:59
28});
```

Di baris 4 saya membuat variabel \$carbon1 dan mengisinya dengan tanggal 1-1-2021 pukul 10:10:10. Ketika di jalankan method modifier endOfDay() di baris 5, maka waktu di tanggal \$carbon1 akan berubah menjadi 23:59:59, yakni tepat di detik akhir hari atau *end of day*.

Prinsip seperti inilah yang juga ada di method lainnya seperti endOfDay(), startOfWeek(), dst hingga endOfDecade(). Pemanggilan method *modifier* akan mengubah tanggal yang tersimpan di variabel \$carbon1, sehingga dalam contoh di atas saya langsung menjalankannya dari Carbon object baru.

4.9. Copying Object

Dari beberapa percobaan yang kita lakukan, mayoritas pemanggilan method akan mengubah nilai yang tersimpan dalam Carbon object tersebut. Berikut salah satu contohnya:

routes\web.php

```
1 ...
2 ...
3 Route::get('/copy-1', function () {
4     $carbon1 = Carbon::create(2021, 1, 1, 10, 10, 10);
5     $carbon2 = $carbon1->endOfDay();
6     echo $carbon1."<hr>"; // 2021-12-31 23:59:59
7     echo $carbon2."<hr>"; // 2021-12-31 23:59:59
8});
```

Di baris 4 saya membuat sebuah Carbon object dan menyimpannya ke variabel \$carbon1. Kemudian di baris 5 menjalankan method \$carbon1->endOfDay() yang hasilnya disimpan ke dalam \$carbon2. Terakhir kedua variabel ini di echo pada baris 6 dan 7.

Hasilnya, baik variabel \$carbon1 maupun \$carbon2 berisi tanggal yang sama, yakni tanggal yang sudah di modifikasi oleh \$carbon1->endOfDay(). Di sini tanggal awal yang sebelumnya ada di \$carbon1 sudah tertimpas akibat pemanggilan method endOfDay().

Bagaimana cara agar tanggal yang tersimpan di \$carbon1 tidak ikut berubah?

Solusi pertama adalah dengan menambah pemanggilan method copy() seperti contoh berikut:

routes\web.php

```
1 ...
2 ...
```

```

3 Route::get('/copy-2', function () {
4     $carbon1 = Carbon::create(2021, 1, 1, 10, 10, 10);
5     $carbon2 = $carbon1->copy()->endOfYear();
6     echo $carbon1."<br>"; // 2021-01-01 10:10:10
7     echo $carbon2."<br>"; // 2021-12-31 23:59:59
8 });

```

Tambahan method `copy()` di baris 5 sebelum perintah method `endOfYear()`, akan mencegah proses penimpaan nilai yang tersimpan di `$carbon1`. Trik ini berhasil karena object yang diubah oleh method `endOfYear()` adalah hasil copyan dari `$carbon1`.

Solusi ke dua adalah dengan melakukan proses instansiasi menggunakan perintah `CarbonImmutable::create()`:

`routes\web.php`

```

1 ...
2 ...
3 Route::get('/copy-3', function () {
4     $carbon1 = \Carbon\CarbonImmutable::create(2021, 1, 1, 10, 10, 10);
5     $carbon2 = $carbon1->endOfYear();
6     echo $carbon1."<br>"; // 2021-01-01 10:10:10
7     echo $carbon2."<br>"; // 2021-12-31 23:59:59
8 });

```

Di baris 4 saya membuat object Carbon dengan static method `\Carbon\CarbonImmutable::create()`. Ini adalah method khusus untuk proses instansiasi *immutable* Carbon object, yakni Carbon object yang nilainya tidak bisa di ubah. Sehingga meskipun langsung di jalankan dengan `endOfYear()`, isi dari `$carbon1` tidak ikut berubah.

Secara umum tidak ada perbedaan antara Carbon object 'biasa' dengan *immutable* Carbon object, sehingga semua method yang kita pelajari sebelumnya tetap bisa dijalankan untuk yang versi immutable ini.

4.10. Timezone

Sejak awal pembahasan tentang Carbon, waktu yang tampil adalah zona GMT atau UTC. Sedangkan kita di Indonesia menggunakan waktu GMT+7 atau UTC+7. Carbon juga bisa di set agar sesuai dengan zona waktu (*timezone*) Indonesia.

Waktu **GMT** (Greenwich Mean Time) sebenarnya sama dengan **UTC** (Coordinated Universal Time).

Bedanya, GMT merujuk ke zona waktu tertentu, yakni ke kota Greenwich, Inggris. Sedangkan UTC adalah standar waktu global yang menjadi penentu awal timezone. Istilah UTC inilah yang sering di pakai untuk merujuk ke standar waktu internasional.

Carbon

Untuk menentukan sebuah timezone dari Carbon object, tambahkan string `timezone` sebagai argument kedua pada saat proses instansiasi. Berikut contoh penggunaannya:

routes\web.php

```
1 ...  
2 ...  
3 Route::get('/timezone-1', function () {  
4     $carbon1 = Carbon::create('now');  
5     echo $carbon1; echo "<br>"; // 2020-05-31 08:20:29  
6  
7     $carbon2 = Carbon::create('now', 'UTC');  
8     echo $carbon2; echo "<br>"; // 2020-05-31 08:20:29  
9  
10    $carbon3 = Carbon::create('now', 'Europe/London');  
11    echo $carbon3; echo "<br>"; // 2020-05-31 09:20:29  
12  
13    $carbon4 = Carbon::create('now', 'Asia/Jakarta');  
14    echo $carbon4; echo "<br>"; // 2020-05-31 15:20:29  
15  
16    $carbon5 = Carbon::create('now', 'Asia/Makassar');  
17    echo $carbon5; echo "<br>"; // 2020-05-31 16:20:29  
18  
19    $carbon6 = Carbon::create('now', 'Asia/Jayapura');  
20    echo $carbon6; echo "<br>"; // 2020-05-31 17:20:29  
21});
```

Jika Carbon object di instansiasi tanpa menulis timezone, maka yang dipakai adalah waktu UTC. Ini diperlihatkan dari hasil `$carbon1` dan `$carbon2` yang memiliki waktu sama.

Untuk variabel `$carbon3` saya isi dengan timezone 'Europe/London' yang ternyata berbeda 1 jam dengan UTC,

Penggunaan zona waktu Indonesia terdapat pada variabel `$carbon4`, `$carbon5` dan `$carbon6`, dimana masing-masingnya saya isi dengan string 'Asia/Jakarta', 'Asia/Makassar' dan 'Asia/Jayapura'. Ketiga string ini mewakili zona waktu WIB, WITA dan WIT.

Daftar lengkap dari string timezone untuk kota dan negara lain bisa dilihat ke en.wikipedia.org/wiki/List_of_tz_database_time_zones.

Penambahan timezone ini juga di dukung oleh proses instansiasi lain carbon Object, misalnya dari `Carbon::now()`, `Carbon::yesterday()`, `Carbon::tomorrow()`, maupun `Carbon::createFromDate()`.

routes\web.php

```
1 ...  
2 ...  
3 Route::get('/timezone-2', function () {  
4     $carbon1 = Carbon::now('Asia/Jakarta');
```

Carbon

```
5 echo $carbon1;    echo "<hr>";      // 2020-06-01 07:33:51
6
7 $carbon2 = Carbon::yesterday('Asia/Jakarta');
8 echo $carbon2;    echo "<hr>";      // 2020-05-31 00:00:00
9
10 $carbon3 = Carbon::tomorrow('Asia/Jakarta');
11 echo $carbon3;    echo "<hr>";      // 2020-06-02 00:00:00
12
13 $carbon4 = Carbon::createFromDate(2021,1,1,'Asia/Jakarta');
14 echo $carbon4;    echo "<hr>";      // 2021-01-01 07:33:51
15});
```

Dengan men-set timezone sebagai 'Asia/Jakarta', maka waktu yang akan di generate oleh Carbon akan sesuai dengan waktu di komputer kita saat ini (standar waktu WIB). Jika anda berada di wilayah WITA atau WIT, bisa disesuaikan dengan timezone tersebut.

4.11. Localization

Localization adalah istilah yang merujuk ke proses translate kata atau bahasa yang digunakan untuk menampilkan tanggal. Biasanya ini dipakai untuk menerjemahkan nama hari dan nama bulan, namun Carbon localization juga akan mengubah hasil dari method 'for human' yang akan kita praktekkan sesaat lagi.

Terdapat beberapa cara mengatur localization di Carbon. Cara pertama adalah dengan menjalankan method Carbon::setLocale() seperti contoh berikut:

routes\web.php

```
1 ...
2 ...
3 Route::get('/locale-1', function () {
4     Carbon::setLocale('id');
5     $carbon1 = Carbon::create(2021, 1, 1, 12, 30, 15);
6
7     echo $carbon1;
8     echo $carbon1->isoFormat('M/D/YY HH:mm');
9     echo $carbon1->isoFormat('dddd');
10    echo $carbon1->isoFormat('dddd, D MMMM YYYY HH:mm');
11    echo $carbon1->isoFormat('LLLL');
12});
```

Hasil kode program:

```
2021-01-01 12:30:15
1/1/21 12:30
Jumat
Jumat, 1 Januari 2021 12:30
Jumat, 1 Januari 2021 pukul 12.30
```

Method Carbon::setLocale() butuh argument berupa 2 digit kode bahasa atau 5 digit (jika

menyertakan dialek). String 'id' di baris 4 mewakili bahasa Indonesia. Untuk kode bahasa lain bisa dilihat ke en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

Setelah menulis kode tersebut, maka semua pemanggilan method `isoFormat()` akan tampil dalam bahasa Indonesia.

Jika anda coba praktekkan localization untuk tanggal hari ini dengan perintah `Carbon::now()`, itu tidak otomatis mengubah timezone. Agar waktu yang tampil sesuai dengan zona waktu Indonesia, jangan lupa menulis timezone juga:

`routes\web.php`

```

1  ...
2  ...
3  Route::get('/locale-2', function () {
4      Carbon::setLocale('id');
5
6      $carbon1 = Carbon::now();
7      echo $carbon1->isoFormat('ddd, D MMMM YYYY HH:mm');
8
9      echo "<br>";
10
11     $carbon2 = Carbon::now('Asia/Jakarta');
12     echo $carbon2->isoFormat('ddd, D MMMM YYYY HH:mm');
13 });

```

Hasil kode program:

```

Senin, 1 Juni 2020 01:06
Senin, 1 Juni 2020 08:06

```

Istilah **timezone** dan **localization** kadang saling di pertukarkan atau dianggap sama. Padahal timezone berkaitan dengan zona waktu, sedangkan localization berhubungan dengan proses translate bahasa. Seperti yang kita lihat, pengaturan timezone tidak otomatis mengubah localization, dan begitu pula sebaliknya.

Cara lain untuk mengatur localization adalah dengan memanggil method `locale()` pada saat pembuatan Carbon object:

`routes\web.php`

```

1  ...
2  ...
3  Route::get('/locale-3', function () {
4
5      $carbon1 = Carbon::create(2021, 1, 1, 12, 30, 15)->locale('id');
6
7      echo $carbon1;                      echo "<br>";      // 2021-01-01 12:30:15
8      echo $carbon1->diffForHumans();    echo "<br>";      // 7 bulan dari sekarang
9      echo $carbon1->dayName;           echo "<br>";      // Jumat

```

Carbon

```
10     echo $carbon1->monthName;           echo "<hr>";      // Januari
11
12     $carbon2 = Carbon::tomorrow('Asia/Jakarta')->locale('id');
13
14     echo $carbon2;                      echo "<hr>";      // 2020-06-02 00:00:00
15     echo $carbon2->diffForHumans();    echo "<hr>";      // 13 jam dari sekarang
16     echo $carbon2->format('l, d F Y'); echo "<hr>";      // Tuesday, 02 June 2020
17 );;
```

Di baris 5 dan 12 saya memanggil method `->locale('id')` pada saat instansiasi Carbon object. Hasilnya, hampir semua method getter tampil dalam bahasa Indonesia, termasuk method `diffForHumans()` serta property `dayName` dan `monthName`.

Pengecualian adalah untuk method `format()` di baris 16. Dalam Carbon, hasil dari method `format()` langsung diambil dari DateTime object bawaan PHP, sehingga tidak otomatis di translate. Karena keterbatasan ini, sebaiknya selalu pakai method `isoFormat()` untuk mengatur tampilan tanggal di Carbon.

4.12. Mengatur Timezone dan Localization dari Laravel

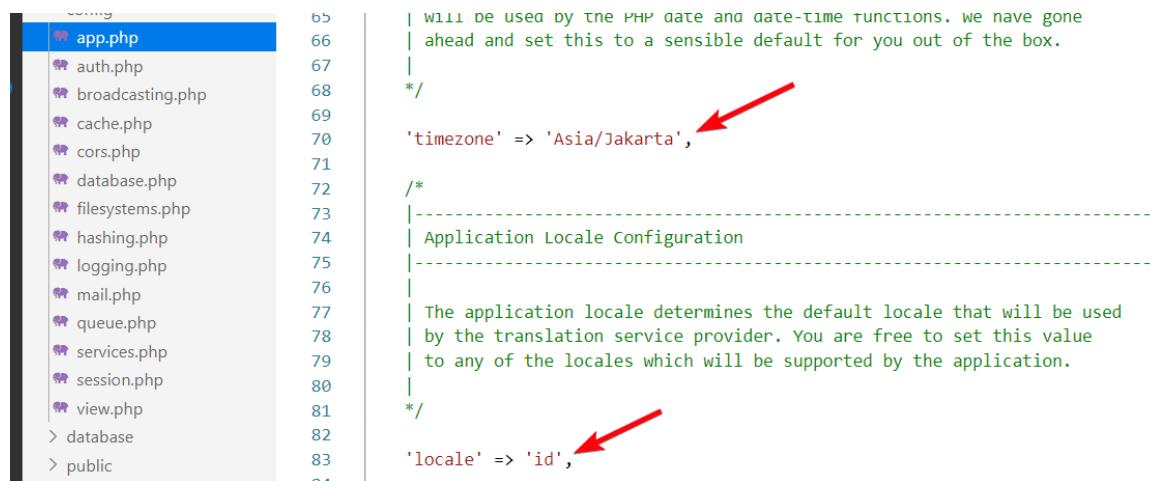
Timezone dan localization biasanya berlaku secara global. Sangat jarang kita membuat aplikasi web yang butuh banyak timezone berbeda. Oleh karena itu Laravel mendukung mengaturan timezone dan localization yang berlaku secara global.

Pengaturan ini bisa di set dari file `config/app.php`, lalu cari kedua baris berikut:

```
'timezone' => 'UTC',
'locale' => 'en',
```

Ini adalah pengaturan awal Laravel, yakni menggunakan timezone UTC dan localization dalam bahasa inggris. Agar sesuai dengan waktu dan bahasa Indonesia, silahkan tukar menjadi:

```
'timezone' => 'Asia/Jakarta',
'locale' => 'id',
```



Gambar: Mengubah pengaturan timezone dan locale Laravel

Setelah itu save file config\app.php. Sebagai pembuktian, jalankan route berikut:

routes\web.php

```

1 ...  

2 ...  

3 Route::get('/locale-4', function () {  

4  

5     $carbon1 = Carbon::now();  

6     echo $carbon1->isoFormat('dddd, D MMMM YYYY HH:mm');  

7  

8     echo "<hr>";  

9  

10    $carbon2 = Carbon::create(2022, 1, 1, 12, 30, 15);  

11    echo $carbon2->diffForHumans(); echo "<hr>";  

12 });

```

Hasil kode program:

```

Senin, 1 Juni 2020 11:30  

1 tahun dari sekarang

```

Meskipun pada saat pembuatan Carbon object saya tidak menyertakan perintah timezone dan localization, hasilnya sudah tampil dalam zona waktu WIB dan dalam bahasa Indonesia.

Dan apabila di butuhkan, kita tetap bisa membuat manual timezone dan localization untuk setiap Carbon object pada saat proses instantiasi. Pengaturan lokal ini akan menimpa pengaturan global yang ada di file config\app.php.

4.13. Carbon dan Eloquent

Setelah memahami cara penggunaan Carbon, saatnya kita terapkan materi tersebut ke Eloquent, terutama ketika menampilkan kolom bertipe date.

Sebagai bahan percobaan, silahkan buat model Mahasiswa sekaligus migration untuk tabel mahasiswas:

```
php artisan make:model Mahasiswa -m
```

Isi file migration mahasiswas sebagai 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->date('tanggal_lahir');  

8         $table->decimal('ipk',3,2)->default(1.00);

```

Carbon

```
9         $table->timestamps();
10    });
11 }
```

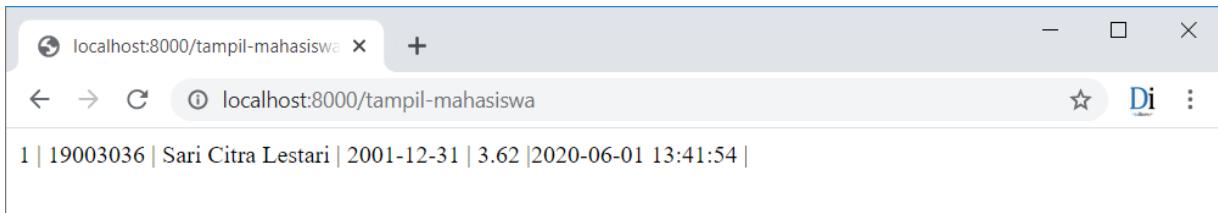
Jalankan migration dengan perintah: `php artisan migrate`. Setelah itu input 1 data menggunakan tinker:

```
$mahasiswa = new App\Models\Mahasiswa
$mahasiswa->nim = '19003036'
$mahasiswa->nama = 'Sari Citra Lestari'
$mahasiswa->tanggal_lahir = '2001-12-31'
$mahasiswa->ipk = 3.62
$mahasiswa->save()
```

Test apakah data `mahasiswas` sudah bisa diakses dengan menjalankan route berikut:

`routes\web.php`

```
1 ...
2 ...
3 Route::get('/tampil-mahasiswa', function() {
4     $mahasiswa = App\Models\Mahasiswa::find(1);
5     echo "$mahasiswa->id | $mahasiswa->nim | $mahasiswa->nama | ".
6         "$mahasiswa->tanggal_lahir | $mahasiswa->ipk | ".
7         "$mahasiswa->created_at | $mahasiswa->update_at";
8});
```



Gambar: Data mahasiswa sudah bisa diakses

Sip, data `mahasiswas` sudah tersedia.

Dalam kode program ini saya menampilkan semua kolom yang ada di tabel `mahasiswa`, termasuk kolom `id`, `created_at` dan `update_at`. Khusus untuk kolom `update_at` belum berisi nilai apapun karena baris ini belum pernah kita update.

Baik, di tabel `mahasiswas` terdapat 3 kolom yang berisi tanggal, yakni kolom `tanggal_lahir`, `created_at` dan `update_at`.

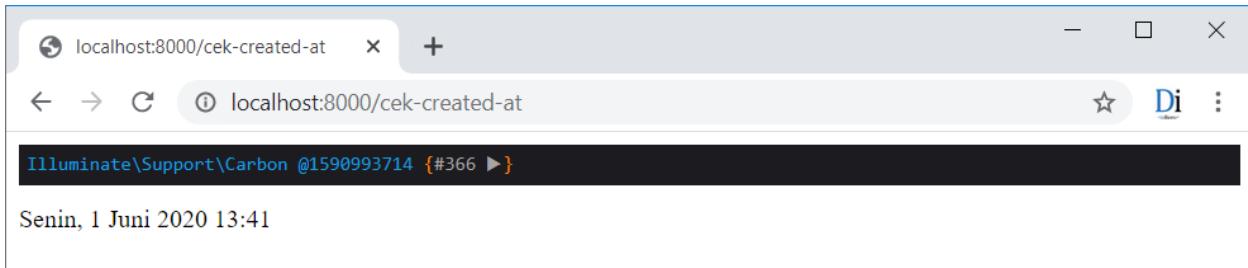
Di dalam Laravel, kolom `created_at` dan `update_at` sudah otomatis langsung berbentuk `carbon object` pada saat diakses, berikut percobaannya:

`routes\web.php`

```
1 ...
2 ...
3 Route::get('/cek-created-at', function () {
```

Carbon

```
4 $mahasiswa = App\Models\Mahasiswa::find(1);
5 dump($mahasiswa->created_at);
6 echo $mahasiswa->created_at->isoFormat('ddd, D MMMM YYYY HH:mm');
7});
```

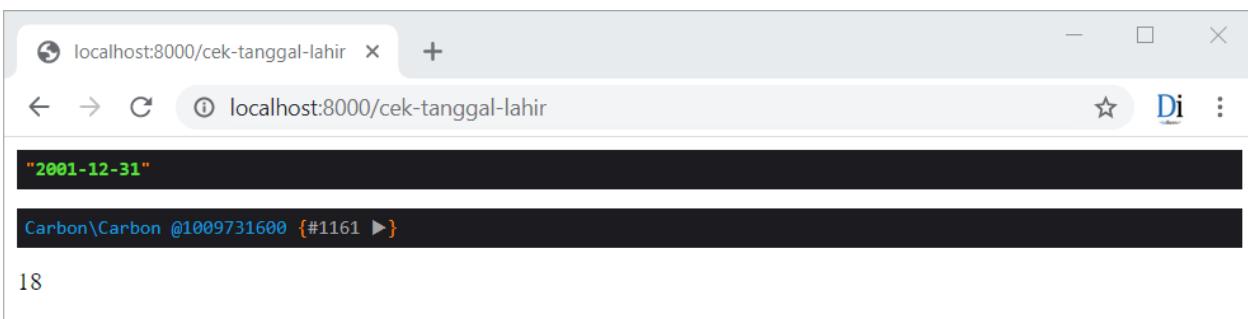


Karena kolom update_at belum berisi nilai apa-apa, dalam contoh ini saya hanya mengakses kolom created_at saja.

Namun untuk kolom bertipe date yang kita definisikan sendiri, itu belum berbentuk Carbon object sehingga harus di instansiasi terlebih dahulu:

routes\web.php

```
1 ...
2 ...
3 Route::get('/cek-tanggal-lahir', function () {
4     $mahasiswa = App\Models\Mahasiswa::find(1);
5     dump($mahasiswa->tanggal_lahir);
6     dump(Carbon::create($mahasiswa->tanggal_lahir));
7     echo Carbon::create($mahasiswa->tanggal_lahir)->age;
8 });
```



Hasil `dump()` di baris 5 memperlihatkan kolom tanggal_lahir hanya berisi tipe data string. Agar bisa mengakses method-method Carbon, string tanggal_lahir saya input sebagai nilai awal untuk `Carbon::create()`. Setelah itu barulah kita bisa mengakses property `age()` seperti di baris 7.

Dalam beberapa situasi, penulisan seperti ini tidak menjadi masalah. Namun jika kita sering mengakses method Carbon untuk kolom tanggal_lahir, akan lebih praktis apabila kolom tersebut langsung terkonversi menjadi Carbon object, kurang lebih seperti kolom created_at.

Cara yang langsung terpikir adalah memakai accessor, yakni dengan membuat sebuah method `getTanggalLahirAttribute()` berisi perintah `return Carbon::create($value)`.

Akan tetapi Laravel juga sudah menyediakan cara yang lebih singkat. Caranya, input nama kolom ke property khusus bernama `$date`.

Silahkan buka model Mahasiswa, lalu modifikasi 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
12     protected $dates = [
13         'tanggal_lahir'
14     ];
15 }
```

Dengan tambahan property ini, maka kolom `tanggal_lahir` akan otomatis di konversi menjadi Carbon object. Sebagai pembuktian, jalankan route berikut:

routes\web.php

```

1 ...
2 ...
3 Route::get('/cek-tanggal-lahir-carbon', function () {
4     $mahasiswa = App\Models\Mahasiswa::find(1);
5     dump($mahasiswa->tanggal_lahir);
6     echo "$mahasiswa->nama berusia {$mahasiswa->tanggal_lahir->age} tahun";
7 });
```



Gambar: Kolom tanggal_lahir sudah langsung menjadi Carbon object

Hasil dari perintah `dump()` memperlihatkan kolom `tanggal_lahir` sudah langsung berbentuk Carbon object, sehingga bisa kita pakai untuk mengakses property `age` di baris 6.

Jika nantinya ada kolom lain yang ingin langsung di konversi ke dalam Carbon, cukup tambah

nama kolom tersebut ke dalam property `$date`.

Dalam bab ini kita telah membahas cukup panjang tentang library Carbon. Dengan Carbon, pengolahan data tanggal menjadi lebih praktis dan lebih mudah. Juga karena Carbon merupakan library terpisah, maka bisa dipakai di luar Laravel.

Berikutnya kita akan masuk ke materi tentang **Scope**.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

5. Scope

Bab kali ini masih membahas tentang method yang bisa kita tambah ke dalam Model. Materi tentang **scope** sendiri tidak terlalu panjang namun sangat bermanfaat untuk beberapa situasi.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

5.1. Pengertian Laravel Scope

Scope adalah sebuah method di dalam Model yang bisa dipakai menyimpan query (terutama query WHERE). Pada dasarnya, scope berfungsi untuk mempersingkat penulisan query yang sering digunakan. Mari kita lihat contohnya.

Sebagai data sample, saya kembali menggunakan tabel `mahasiswa` serta model **Mahasiswa**:

```
php artisan make:model Mahasiswa -m
```

Kemudian isi file migration `mahasiswa` sebagai berikut:

```
database\migrations\<timestamp>_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    });
11 }
```

Jalankan migration dengan perintah: `php artisan migrate`. Setelah itu input 4 data berikut menggunakan Tinker (silahkan di copy paste ke cmd):

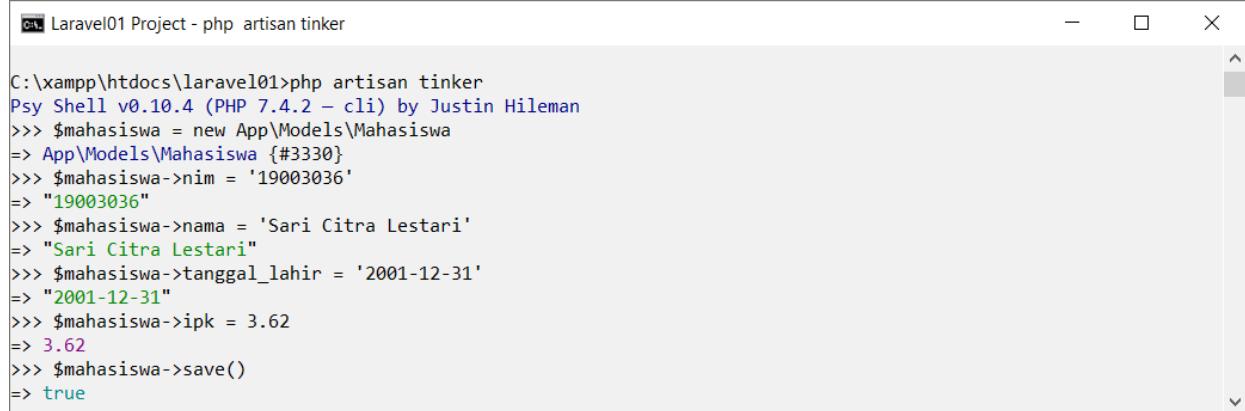
Scope

```
$mahasiswa = new App\Models\Mahasiswa
$mahasiswa->nim = '19003036'
$mahasiswa->nama = 'Sari Citra Lestari'
$mahasiswa->tanggal_lahir = '2001-12-31'
$mahasiswa->ipk = 3.62
$mahasiswa->save()

$mahasiswa = new App\Models\Mahasiswa
$mahasiswa->nim = '19021044'
$mahasiswa->nama = 'Rudi Permana'
$mahasiswa->tanggal_lahir = '2000-08-22'
$mahasiswa->ipk = 2.99
$mahasiswa->save()

$mahasiswa = new App\Models\Mahasiswa
$mahasiswa->nim = '19002032'
$mahasiswa->nama = 'Rina Kumala Sari'
$mahasiswa->tanggal_lahir = '2000-06-28'
$mahasiswa->ipk = 3.82
$mahasiswa->save()

$mahasiswa = new App\Models\Mahasiswa
$mahasiswa->nim = '18012012'
$mahasiswa->nama = 'James Situmorang'
$mahasiswa->tanggal_lahir = '1999-04-02'
$mahasiswa->ipk = 2.74
$mahasiswa->save()
```



```
C:\xampp\htdocs\laravel01>php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.2 - cli) by Justin Hileman
>>> $mahasiswa = new App\Models\Mahasiswa
=> App\Models\Mahasiswa {#3330}
>>> $mahasiswa->nim = '19003036'
=> "19003036"
>>> $mahasiswa->nama = 'Sari Citra Lestari'
=> "Sari Citra Lestari"
>>> $mahasiswa->tanggal_lahir = '2001-12-31'
=> "2001-12-31"
>>> $mahasiswa->ipk = 3.62
=> 3.62
>>> $mahasiswa->save()
=> true
```

Gambar: Proses input data ke tabel mahasiswas menggunakan Tinker

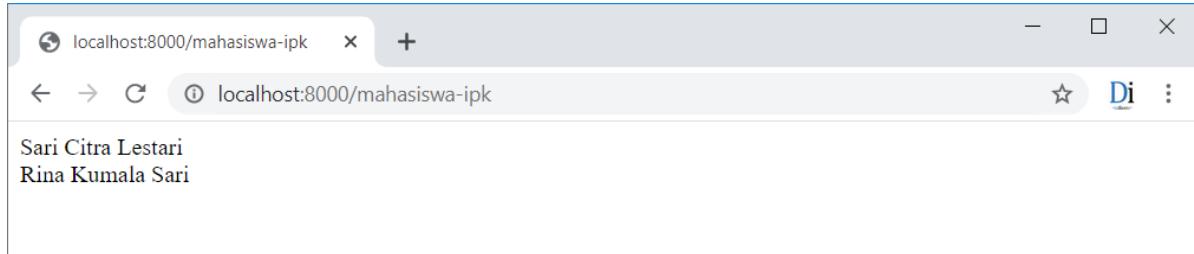
Sip, data *sample* sudah siap. Percobaan pertama, silahkan ketik dan akses route berikut:

```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Models\Mahasiswa;
5
6 Route::get('/mahasiswa-ipk', function () {
7     $mahasiswas = Mahasiswa::where('ipk', '>', '3.5')->get();
8     foreach ($mahasiswas as $mahasiswa) {
```

Scope

```
9         echo($mahasiswa->nama). '<br>';
10    }
11});
```



Gambar: Tampilkan mahasiswa dengan IPK > 3.5

Dalam kode program ini saya menampilkan semua data mahasiswa dengan IPK di atas 3.5. Query berjalan normal dan hasilnya sesuai dengan keinginan. Namun jika query tersebut cukup sering dipakai (tidak hanya sekali saja), bisa kita simpan ke dalam **Scope**. Caranya, tambah method berikut di model Mahasiswa:

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
12     public function scopeCumlaude($query)
13     {
14         return $query->where('ipk','>','3.5');
15     }
16 }
```

Kembali ke route, sekarang jalankan kode berikut:

routes\web.php

```
1 ...
2 ...
3 Route::get('/mahasiswa-ipk-scope', function () {
4     $mahasiswas = Mahasiswa::cumlaude()->get();
5     foreach ($mahasiswas as $mahasiswa) {
6         echo($mahasiswa->nama). '<br>';
7     }
8});
```

Hasil kode program:

Scope

Sari Citra Lestari
Rina Kumala Sari

Perhatikan pemanggilan method `::cumlaude()` di baris 4. Itulah perintah yang dipakai untuk mengakses method `scopeCumlaude()` dari model Mahasiswa.

Dalam percobaan ini saya memindahkan penulisan query `where('ipk', '>', '3.5')` ke dalam method `scopeCumlaude()`. Untuk query singkat seperti ini manfaatnya memang tidak terlihat. Namun bayangkan jika query tersebut cukup panjang dan kompleks, maka dengan memindahkannya ke dalam scope, proses menampilkan data menjadi lebih efisien.

5.2. Cara Penulisan Scope

Berikut format penulisan method Scope ke dalam model :

```
public function scope<NamaScope>($query)
{
    return ....($query);
}
```

Penulisan nama method harus diawali dengan kata "**scope**", kemudian diikuti dengan nama scope sesuai keinginan. Nama scope ini sebaiknya mengikuti penulisan CamelCase.

Scope memiliki 1 parameter `$query` yang akan diisi Laravel dengan **Builder object** pada saat pemanggilan scope. Builder object sendiri merupakan object khusus Laravel yang berisi perintah query (bisa berasal dari query builder maupun eloquent). Method scope nantinya juga harus mengembalikan builder object ini dengan perintah `return`.

Sebagai contoh selanjutnya, saya memiliki route berikut:

```
routes\web.php

1 ...
2 ...
3 Route::get('/mahasiswa-lahir', function () {
4     $mahasiswas = Mahasiswa::whereYear('tanggal_lahir', '=', '2000')->get();
5     foreach ($mahasiswas as $mahasiswa) {
6         echo($mahasiswa->nama). '<br>';
7     }
8});
```

Hasil kode program:

Rudi Permana
Rina Kumala Sari

Query pada baris 4 dipakai untuk mencari mahasiswa yang lahir di tahun 2000. Method `whereYear()` merupakan salah satu method milik query builder bawaan Laravel*.

*Secara internal, eloquent menggunakan query builder. Sehingga kita juga bisa memakai method-method milik query builder di eloquent.

Merasa penulisan query ini cukup panjang, saya ingin memindahkannya ke sebuah scope bernama `lahir2000()` agar nantinya bisa diakses sebagai berikut:

`routes\web.php`

```

1 ...
2 ...
3 Route::get('/mahasiswa-lahir-scope', function () {
4     $mahasiswas = Mahasiswa::lahir2000()->get();
5     foreach ($mahasiswas as $mahasiswa) {
6         echo($mahasiswa->nama). '<br>';
7     }
8 });

```

Bisakah anda rancang method scopenya? Silahkan coba sebentar...

Baik, berikut kode yang bisa digunakan:

`App\Models\Mahasiswa.php`

```

1 ...
2 ...
3     public function scopeLahir2000($query)
4     {
5         return $query->whereYear('tanggal_lahir', '=', '2000');
6     }

```

Dengan scope di atas, kita cukup akses `Mahasiswa::lahir2000()->get()` ketika ingin menampilkan mahasiswa yang lahir di tahun 2000, tidak perlu lagi menulis lengkap query `whereYear()`.

Bisa juga diperhatikan bahwa nilai yang dikembalikan oleh sebuah scope, berangkat dari variabel `$query` yang berisi Builder object. Dari `$query` ini kita bisa sambung berbagai method query builder untuk membuatnya lebih spesifik, seperti `where()`, `whereYear()`, `skip()`, `take()`, `limit()` hingga `orderBy()`.

5.3. Dynamic Scope

Dynamic scope adalah sebutan untuk scope yang bisa menerima satu / beberapa argument.

Melanjutkan contoh scope `lahir2000()`, saya ingin membuat scope yang mirip tapi kali ini bisa menerima angka tahun lahir sebagai argument. Dengan demikian, scope menjadi lebih fleksibel karena bisa menampilkan tahun-tahun lain. Scope ini saya beri nama `scopeLahir()`, berikut kode programnya:

Scope

App\Models\Mahasiswa.php

```
1 ...  
2 ...  
3     public function scopeLahir($query,$tahun)  
4     {  
5         return $query->whereYear('tanggal_lahir','=',$tahun);  
6     }
```

Setelah parameter \$query, terdapat tambahan parameter \$tahun di baris 3. Parameter \$tahun ini kemudian diinput sebagai argument ketiga dari method whereYear() pada baris 4.

Pada saat mengakses scope, input angka tahun sebagai argument kedua:

routes\web.php

```
1 ...  
2 ...  
3 Route::get('/tampil-lahir-scope-dynamic', function () {  
4     $mahasiswas = Mahasiswa::lahir(2001)->get();  
5     foreach ($mahasiswas as $mahasiswa) {  
6         echo($mahasiswa->nama). '<br>';  
7     }  
8});
```

Hasil kode program:

Sari Citra Lestari

Di baris 4 saya menjalankan method Mahasiswa::lahir(2001)->get(). Angka 2001 ini akan sampai ke parameter \$tahun di scopeLahir(). Tergantung kebutuhan, sebuah scope bisa memiliki dua, tiga atau banyak argument tambahan lain.

Sama seperti method query biasa, scope juga bisa di-chaining seperti contoh berikut:

routes\web.php

```
1 ...  
2 ...  
3 Route::get('/scope-chain', function () {  
4     $mahasiswas = Mahasiswa::lahir(2000)->cumlaude()->get();  
5     foreach ($mahasiswas as $mahasiswa) {  
6         echo($mahasiswa->nama). '<br>';  
7     }  
8});  
9
```

Hasil kode program:

Rina Kumala Sari

Perintah di baris 4 artinya saya ingin mencari mahasiswa yang lahir di tahun 2000 dan memiliki IPK di atas 3.5.

Dalam bab ini kita telah mempelajari tentang pengertian dan cara penggunaan scope. Di dokumentasi Laravel, apa yang kita bahas di sini dikenal sebagai **Local Scope**, yakni scope yang baru aktif ketika dipanggil.

Selain itu ada juga istilah **Global Scope**, yakni scope yang selalu aktif untuk setiap query yang dijalankan. Karena menurut saya penggunaan global scope relatif jarang, saya tidak akan bahas lebih lanjut. Jika tertarik anda bisa baca ke [Eloquent – Global Scopes](#).

Berikutnya kita bahas library **Faker** yang sangat menarik.

6. Faker

Selain Carbon, **Faker** juga merupakan library pihak ketiga yang disertakan dalam paket Laravel framework. Dalam bab ini kita akan bahas apa itu Faker serta cara penggunaannya.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel101
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

6.1. Pengertian Faker Library

Faker adalah sebuah library PHP untuk men-generate data *dummy* atau *fake data*. Faker biasa dipakai jika ingin membuat data awal atau data random sebagai bahan percobaan. Misalnya kita sedang membuat sistem informasi mahasiswa, maka akan terasa lebih "real" apabila saat uji coba sudah terdapat ratusan data mahasiswa di dalam database.

Data dummy yang bisa digenerate oleh Faker tidak hanya nama saja, tapi sangat beragam mulai dari nama perusahaan, alamat, username, email, hingga kata dan paragraf.

Dalam Laravel, faker sering dipakai bersama **Seeder** dan **Factory**. Materi tentang seeder dan factory akan di bahas pada bab selanjutnya, sekarang kita akan fokus ke cara penggunaan faker terlebih dahulu.

Sama seperti Carbon, fitur yang tersedia di faker juga sangat banyak. Panduan lengkap dari faker bisa diakses pada alamat github.com/fzaninotto/Faker. Dan karena merupakan library terpisah, faker bisa di pakai secara independen, tidak harus bersama framework Laravel.

6.2. Faker Instantiation

Faker hadir dalam bentuk class yang harus di instansiasi menjadi object. Karena Laravel sudah menyertakan library faker secara bawaan, kita sudah bisa langsung mengaksesnya.

Dalam Laravel, **Faker** class berada dalam namespace `Faker\Factory`. Sehingga jika tanpa proses import, instansiasi faker bisa dilakukan dengan kode berikut:

```
$faker = \Faker\Factory::create();
```

Dengan perintah di atas, variabel `$faker` sudah berisi object dari class **Faker**. Di sini kita menggunakan static method `create()` untuk membuat Faker object, bukan menggunakan perintah `new` seperti object biasa.

Perintah `Factory` yang ada pada proses instansiasi faker bukan merujuk ke fitur **factory** dari Laravel (yang nantinya juga akan kita bahas), tetapi ke *factory design pattern*.

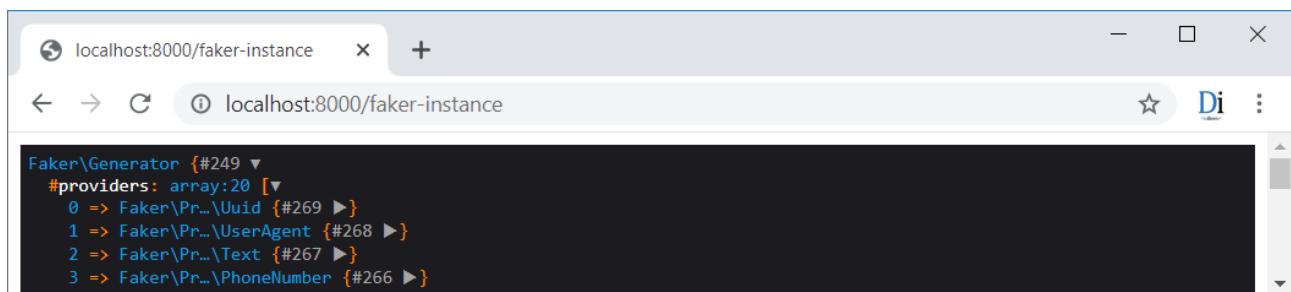
Design pattern adalah teknik khusus dalam ilmu programming untuk memecahkan masalah, yang salah satunya bernama *factory*. Ciri khas dari *factory design pattern* adalah proses instansiasi object dilakukan dari static method bernama `::create()`.

Salah satu teknik lain dari design pattern adalah *singleton*, yakni membuat class yang hanya bisa di-instansiasi 1 kali saja seperti yang pernah kita terapkan di buku **OOP PHP Uncover**.

Berikut praktek proses instansiasi Faker object yang bisa dijalankan di route:

```
routes\web.php

1 Route::get('/faker-instance', function () {
2     $faker = \Faker\Factory::create();
3     dump($faker);
4 });
```



Gambar: Hasil `dump()` instansiasi Faker class

Faker object yang tersimpan di variabel `$faker` tidak bisa langsung kita echo, tapi harus dipanggil dengan berbagai method yang akan di bahas sepanjang bab ini.

Agar tidak perlu menulis `\Faker\Factory` setiap kali melakukan proses instansiasi, biasanya namespace tersebut di import ke dalam file saat ini. Caranya, tambah perintah `use Faker\Factory as Faker` di bagian paling atas file. Dengan demikian proses instansiasi cukup dipanggil dengan perintah `Faker::create()`:

Faker

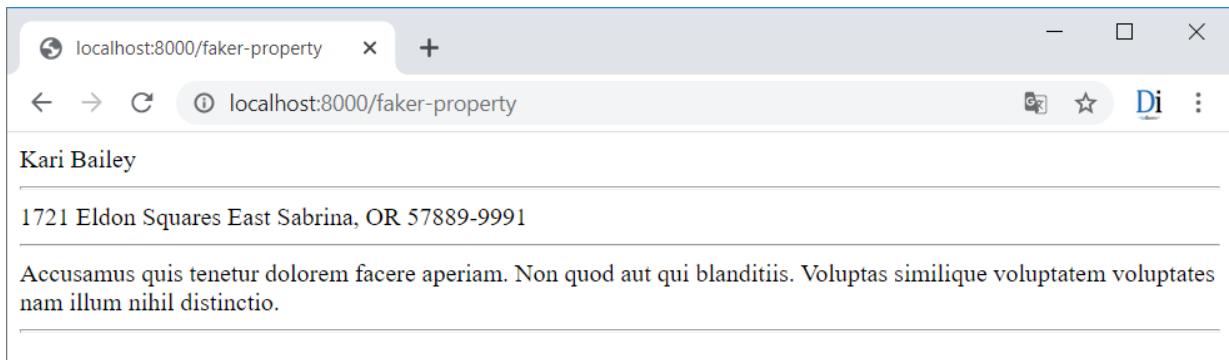
```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use Faker\Factory as Faker;
5
6 ...
7 ...
8
9 Route::get('/faker-instance-import', function () {
10     $faker = Faker::create();
11     dump($faker);
12});
```

Untuk bisa men-generate data dummy, cukup akses beberapa property dari variabel \$faker seperti contoh berikut:

```
routes\web.php

1 <?php
2 ...
3 Route::get('/faker-property', function () {
4     $faker = Faker::create();
5     echo $faker->name;    echo "<hr>";
6     echo $faker->address; echo "<hr>";
7     echo $faker->text;    echo "<hr>";
8});
```



Gambar: Contoh hasil pengaksesan property name, address dan text dari \$faker

Dalam kode di atas saya mengakses 3 property Faker, yakni `name`, `address` dan `text`. Ketiga property ini akan menghasilkan data dummy untuk nama, alamat dan text. Setiap kali halaman tersebut di refresh, data yang tampil juga akan terus berganti (random).

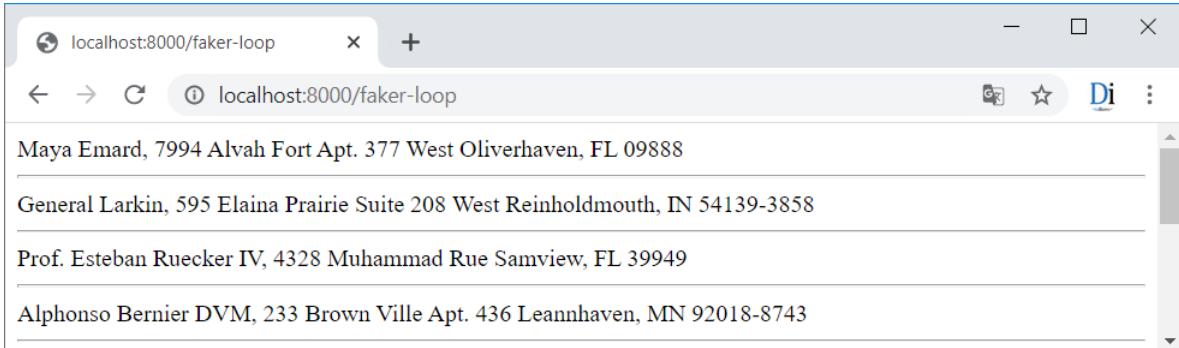
Jika kita ingin membuat beberapa data dummy, tinggal akses property yang sama beberapa kali atau bisa juga ditulis ke dalam perulangan:

```
routes\web.php

1 <?php
2 ...
```

Faker

```
3 Route::get('/faker-loop', function () {
4     $faker = Faker::create();
5     for ($i=0; $i<10; $i++) {
6         echo "$faker->name, $faker->address <hr>";
7     }
8});
```



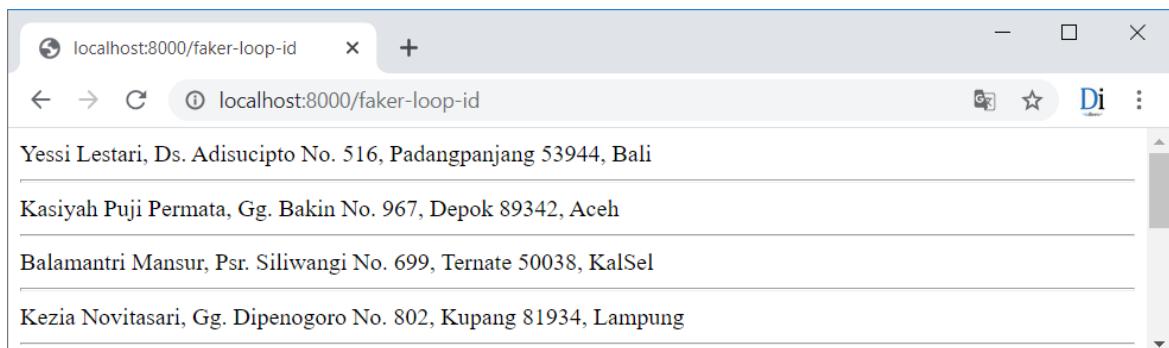
Gambar: Hasil looping \$faker

Hasilnya, akan tampil 10 buah nama serta alamat *dummy* yang di-generate Faker.

Yang sangat menarik, Faker mendukung localization Indonesia. Dengan demikian nama dan alamat yang di generate sesuai dengan nama lokal di Indonesia:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-loop-id', function () {
4     $faker = Faker::create('id_ID');
5     for ($i=0; $i<10; $i++) {
6         echo "$faker->name, $faker->address <hr>";
7     }
8});
```



Gambar: Faker dengan localization id_ID

Untuk mengatur localization di Faker, input 5 digit kode bahasa sebagai argument pada saat proses instansiasi. Di baris 4 kode yang saya pakai adalah 'id_ID' untuk bahasa Indonesia. Dengan demikian semua property yang diakses dari Faker akan menggunakan versi lokal bahasa Indonesia.

Namun jika di perhatikan, data alamat terlihat agak aneh dimana kota dan provinsi tidak sesuai. Misalnya kota Padangpanjang berada di Bali, atau kota Depok berada di Aceh. Tapi karena tujuannya hanya untuk data *dummy*, ini tidak terlalu masalah.

6.3. Faker Formatters

Faker menyediakan ratusan property untuk men-generate berbagai data *dummy*. Kita akan bahas beberapa yang cukup sering dipakai. Untuk versi lebih lengkap, bisa langsung ke dokumentasi Faker di github.com/fzaninotto/Faker.

Base Format

Disebut sebagai **base** karena terdiri dari format yang umum dipakai. Berikut contoh prakteknya:

routes\web.php

```

1 <?php
2 ...
3 Route::get('/faker-base', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->name;      echo "<hr>";
6     echo $faker->address;   echo "<hr>";
7     echo $faker->email;     echo "<hr>";
8     echo $faker->company;   echo "<hr>";
9     echo $faker->date;      echo "<hr>";
10    echo $faker->userName; echo "<hr>";
11    echo $faker->password; echo "<hr>";
12    echo $faker->url;      echo "<hr>";
13 });

```

Hasil kode program:

```

Vicky Puspasari M.M.
Psr. Basoka Raya No. 227, Jambi 57265, DKI
rahmawati.uchita@budiman.tv
PT Hidayat Suartini
1974-05-08
salwa.yulianti
ExoG#:4lGwm}
https://www.yolanda.sch.id/sequi-ducimus-fugiat-omnis-eius-itaque-sunt

```

Pada baris 4 saya menggunakan localization **id_ID** sehingga seluruh data Faker akan tampil dalam versi bahasa Indonesia. Fungsi dari semua property sudah bisa di tebak dari namanya, sehingga tidak akan saya bahas lagi.

Number Format

Number format dipakai untuk men-generate angka acak. Selain property, juga terdapat method agar bisa menampung argument untuk pengaturan lebih lanjut:

routes\web.php

```

1 <?php
2 ...
3 Route::get('/faker-number', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->randomDigit;           echo "<hr>";
6     echo $faker->randomDigitNot(5);    echo "<hr>";
7     echo $faker->randomDigitNotNull;   echo "<hr>";
8     echo $faker->randomNumber(7);      echo "<hr>";
9     echo $faker->randomFloat(5, 1, 4); echo "<hr>";
10    echo $faker->numberBetween(100,199); echo "<hr>";
11 });

```

Hasil kode program:

```

8
3
5
7163067
1.33884
104

```

Berikut penjelasan singkat dari setiap property/method:

- randomDigit mengembalikan 1 digit angka acak antara 0 – 9.
- randomDigitNot(5) mengembalikan 1 digit angka acak antara 0 – 9, kecuali angka 5.
- randomDigitNotNull mengembalikan 1 digit angka acak antara 1 – 9.
- randomNumber(7) mengembalikan 7 digit angka acak.
- randomFloat(5, 1, 4) mengembalikan angka pecahan dengan maksimal 5 digit desimal (5 angka di belakang koma), serta 1 digit angka bulat dengan nilai antara 1 – 4.
- numberBetween(100,199) mengembalikan angka acak antara 100 – 199.

Alphanumeric Format

Format **alphanumeric** dipakai untuk men-generate kode acak perpaduan huruf dan angka:

routes\web.php

```

1 <?php
2 ...
3 Route::get('/faker-alphanumeric', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->randomLetter;           echo "<hr>";

```

```

6 echo $faker->numerify('ID##AB##');
7 echo $faker->lexify('ID??AB??');
8 echo $faker->bothify('ID##AB??');
9 echo $faker->shuffle("DuniaIlkom");
10 print_r ($faker->shuffle([1, 2, 3, 4, 5]));
11 echo $faker->randomElement(['a', 'b', 'c']);
12 print_r ($faker->randomElements(['a', 'b', 'c']));
13 print_r ($faker->randomElements(['a', 'b', 'c'], 2));
14 });

```

Hasil kode program:

```

1
ID43AB29
IDtwAByt
ID25ABko
iluoinakDm
Array ( [0] => 5 [1] => 1 [2] => 4 [3] => 2 [4] => 3 )
c
Array ( [0] => a )
Array ( [0] => b [1] => c )

```

Berikut penjelasan singkat dari setiap property/method:

- `randomLetter` mengembalikan 1 digit huruf acak.
- `numerify('ID##AB##')` mengembalikan pola `ID##AB##` dimana karakter # akan diganti dengan angka acak.
- `lexify('ID??AB??')` mengembalikan pola `ID??AB??` dimana karakter ? akan diganti dengan huruf acak.
- `bothify('ID##AB??')` mengembalikan pola `ID##AB??` dimana karakter # akan diganti dengan angka acak dan karakter ? akan diganti dengan huruf acak.
- `shuffle("DuniaIlkom")` mengembalikan string acak dari susunan karakter "DuniaIlkom".
- `shuffle([1, 2, 3, 4, 5])` mengembalikan array baru dengan mengacak urutan element dari array `[1, 2, 3, 4, 5]`.
- `randomElement(['a', 'b', 'c'])` mengembalikan 1 nilai acak yang diambil dari array `['a', 'b', 'c']`.
- `randomElements(['a', 'b', 'c'])` mengembalikan 1 element acak yang diambil dari array `['a', 'b', 'c']`.
- `randomElements(['a', 'b', 'c'], 2)` mengembalikan 2 element acak yang diambil dari array `['a', 'b', 'c']`. Jumlah element diinput sebagai argument kedua dari method `randomElements()`.

Word dan Sentence Format

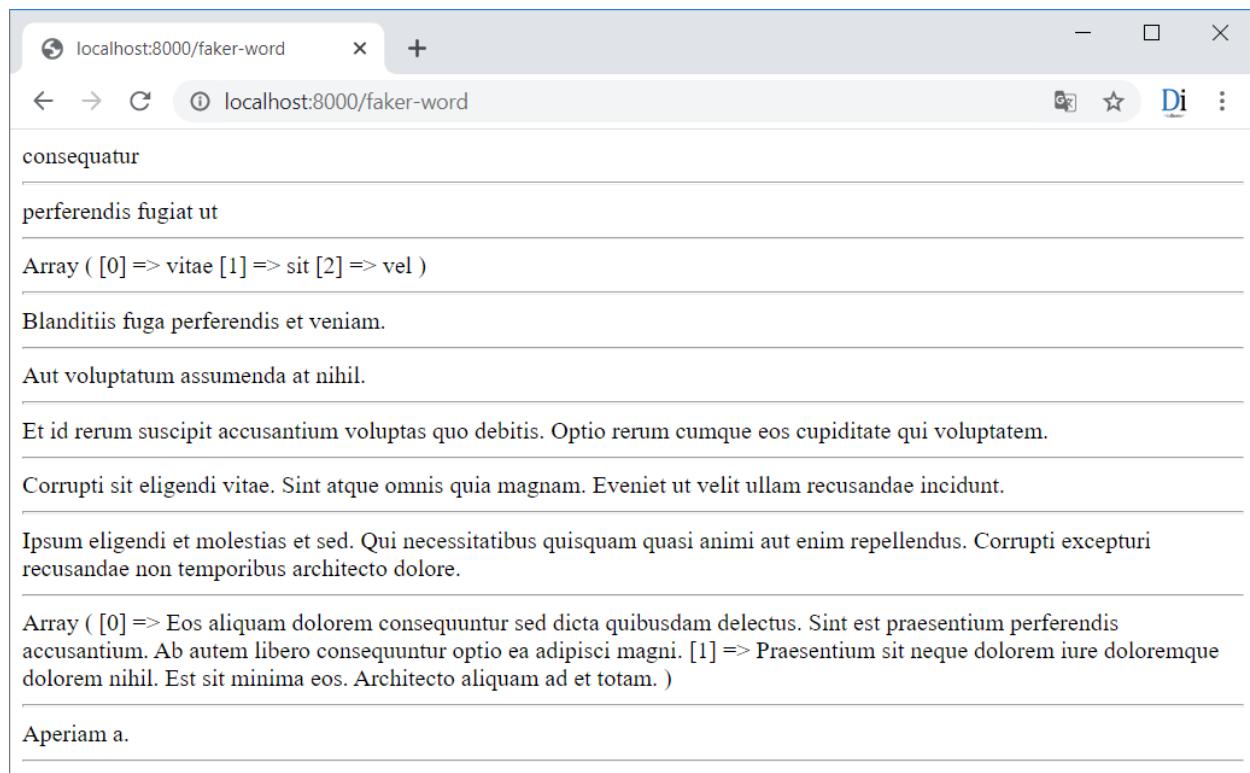
Word dan **Sentence** format dipakai untuk men-generate data dummy yang terdiri dari susunan kata dan kalimat:

routes\web.php

```

1 <?php
2 ...
3 Route::get('/faker-word', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->word;
6     echo $faker->words(3, true);
7     print_r ($faker->words(3, false));
8     echo $faker->sentence(5);
9     echo $faker->sentence(5, false);
10    echo $faker->sentences(2, true);
11    echo $faker->paragraph(3);
12    echo $faker->paragraph(3, false);
13    print_r ($faker->paragraphs(2, false));
14    echo $faker->text(10);
15 });

```



Gambar: Hasil dari word dan sentence format

Berikut penjelasan singkat dari setiap property/method:

- `word` mengembalikan 1 kata acak.
- `words(3, true)` mengembalikan 3 kata acak. Argument `true` dipakai agar hasil akhir

berbentuk string. Nilai `true` merupakan nilai default jika tidak ditulis. Beberapa method Faker lain juga memiliki pengaturan seperti ini.

- `words(3, false)` mengembalikan 3 kata acak. Argument `false` dipakai agar hasil akhir berbentuk array.
- `sentence(5)` mengembalikan 1 kalimat yang terdiri dari 5 kata acak. Karena tidak ada tambahan argument kedua maka akan ada variasi kata dalam 1 kalimat (plus-minus 1 kata).
- `sentence(5, false)` mengembalikan 1 kalimat yang terdiri dari 5 kata acak. Karena argument kedua bernilai `false`, maka jumlah kata akan selalu 5 (tidak ada variasi).
- `sentences(2, true)` mengembalikan 2 kalimat acak. Karena argument kedua di isi `true`, maka hasilnya berbentuk string. Jika argument kedua di isi `false`, hasilnya berupa array dimana 1 paragraf = 1 element array.
- `paragraph(3)` mengembalikan 3 paragraf acak. Karena tidak memiliki tambahan argument kedua maka akan ada variasi jumlah paragraf (plus-minus 1 paragraf).
- `paragraph(3, false)` mengembalikan 3 paragraf acak. Karena argument kedua bernilai `false`, maka jumlah paragraf akan selalu 3 (tidak ada variasi).
- `paragraphs(2, false)` mengembalikan 2 kumpulan paragraf. Satu kumpulan paragraf bisa terdiri dari 3 – 5 paragraf. Karena argument kedua bernilai `false`, maka hasil akhir berbentuk array, dimana 1 kumpulan paragraf = 1 element array. Jika argument kedua diisi `true` maka hasilnya akan menjadi string.

Untuk penulisan method, perhatikan bahwa hasil dari method `singular` akan berbeda dengan method `plural`. Maksudnya, method `sentence()` tidak sama dengan `sentences()`, begitu pula antara method `paragraph()` yang menampilkan hasil berbeda dengan method `paragraphs()`.

Name Format

Faker menyediakan format yang cukup detail untuk men-generate nama serta komponen yang biasa ada di nama:

`routes\web.php`

```

1  <?php
2  ...
3  Route::get('/faker-name', function () {
4      $faker = Faker::create('id_ID');
5      echo $faker->title;           echo "<hr>";
6      echo $faker->titleMale;       echo "<hr>";
7      echo $faker->titleFemale;     echo "<hr>";
8      echo $faker->suffix;          echo "<hr>";
9      echo $faker->name;            echo "<hr>";
10     echo $faker->firstName;       echo "<hr>";

```

Faker

```
11 echo $faker->firstNameMale;      echo "<hr>";
12 echo $faker->firstNameFemale;    echo "<hr>";
13 echo $faker->lastName;          echo "<hr>";
14 echo $faker->lastNameMale;       echo "<hr>";
15 echo $faker->lastNameFemale;     echo "<hr>";
16});
```

Hasil kode program:

```
H.
Ir.
dr.
M.TI.
Julia Lailasari
Rahmi
Asmianto
Tania
Anggriawan
Permadi
Yulianti
```

Nama dari setiap property setidaknya sudah menggambarkan fungsi dari property tersebut. Diantaranya titel awalan laki-laki (`titleMale`), titel awalan perempuan (`titleFemale`), nama depan laki-laki (`firstNameMale`) dan nama belakang perempuan (`lastNameFemale`).

Address Format

Address format dipakai untuk men-generate komponen alamat. Sama seperti nama, Faker menyediakan kode untuk hampir setiap komponen alamat:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-address', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->buildingNumber;      echo "<hr>";
6     echo $faker->streetName;         echo "<hr>";
7     echo $faker->streetAddress;      echo "<hr>";
8     echo $faker->postcode;           echo "<hr>";
9     echo $faker->city;              echo "<hr>";
10    echo $faker->stateAbhr;        echo "<hr>";
11    echo $faker->state;             echo "<hr>";
12    echo $faker->address;           echo "<hr>";
13    echo $faker->country;          echo "<hr>";
14});
```

Hasil kode program:

```
875
Kartini
Psr. Baan No. 604
```

Faker

```
26003
Administrasi Jakarta Timur
KalBar
Sulawesi Tenggara
Jln. Raden No. 640, Madiun 25050, Banten
Nikaragua
```

Hasil alamat ini sangat dipengaruhi oleh localization. Jika Faker di instansiasi tanpa menyertakan localization, secara default menggunakan 'en_US' sehingga yang tampil adalah alamat di Amerika Serikat.

Phone dan Company Format

Nomor telepon dan nama perusahaan juga bisa di-generate dengan kode berikut:

```
routes\web.php
```

```
1 <?php
2 ...
3 Route::get('/faker-phone', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->phoneNumber;      echo "<hr>";
6     echo $faker->company;         echo "<hr>";
7 });

});
```

Hasil kode program:

```
0893 175 250
PT Hasanah
```

Untuk nomor telepon, Faker akan men-generate dalam berbagai format seperti (+62) 249 1446 9237, 0361 7809 026, atau (+62) 212 6977 727. Jika data-data ini akan diinput ke dalam database, maka kolom tersebut harus bisa menerima karakter (,) dan +.

Internet Format

Internet format adalah format untuk men-generate data yang berhubungan dengan teknologi seperti email, URL atau IP address:

```
routes\web.php
```

```
1 <?php
2 ...
3 Route::get('/faker-internet', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->email;           echo "<hr>";
6     echo $faker->safeEmail;       echo "<hr>";
7     echo $faker->freeEmail;        echo "<hr>";
8     echo $faker->companyEmail;    echo "<hr>";
9     echo $faker->freeEmailDomain; echo "<hr>";
10    echo $faker->safeEmailDomain; echo "<hr>";
```

Faker

```
11 echo $faker->userName;           echo "<hr>";
12 echo $faker->password;          echo "<hr>";
13 echo $faker->domainName;        echo "<hr>";
14 echo $faker->domainWord;        echo "<hr>";
15 echo $faker->tld;              echo "<hr>";
16 echo $faker->url;              echo "<hr>";
17 echo $faker->slug;             echo "<hr>";
18 echo $faker->ipv4;             echo "<hr>";
19 echo $faker->localIpv4;        echo "<hr>";
20 echo $faker->ipv6;             echo "<hr>";
21 echo $faker->macAddress;       echo "<hr>";
22 });

});
```

Hasil kode program:

```
tnasyiah@gmail.com
darijan.nainggolan@example.org
sihombing.gambira@gmail.co.id
titin17@sitompul.sch.id
yahoo.co.id
example.net
permata.yani
108]["Ekb@UC
widiastuti.go.id
pertwi
com
https://damanik.com/est-placeat-impedit-sint-voluptatem-quas-magnam.html
et-alias-doloremque-qui-aut-commodi-ut
221.255.77.245
192.168.184.213
88b8:4e31:6ebf:2fc5:ddb:e32a:8fc6:7db
67:99:F5:99:B9:E9
```

Fungsi dari sebagian besar property sudah bisa langsung kita tebak.

Khusus untuk email hadir dalam 2 'rasa': versi normal yang terdiri dari `email` dan `freeEmailDomain` serta versi 'safe' dengan format `safeEmail` dan `safeEmailDomain`.

Property `email` dengan tambahan 'safe' merupakan versi aman yang bisa dipastikan tidak ada pemiliknya. Jika kita men-generate email menggunakan perintah `email`, bisa jadi email tersebut sudah dimiliki oleh seseorang. Untuk contoh-contoh yang sensitif, lebih baik pakai `safeEmail()`.

Domain seperti `example.com`. `example.net` dan `example.org` sengaja di siapkan oleh IANA (badan pengatur domain internasional) sebagai domain contoh.

Date Format

Berikut beberapa property untuk membuat format tanggal menggunakan Faker:

Faker

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-date', function () {
4     $faker = Faker::create('id_ID');
5     echo $faker->unixTime;                                echo "<hr>";
6     echo $faker->date;                                    echo "<hr>";
7     echo $faker->date('d/m/Y');                           echo "<hr>";
8     echo $faker->date('d/m/Y', '1990-01-01');           echo "<hr>";
9     echo $faker->time;                                    echo "<hr>";
10    echo $faker->dayOfMonth;                            echo "<hr>";
11    echo $faker->dayOfWeek;                            echo "<hr>";
12    echo $faker->month;                                 echo "<hr>";
13    echo $faker->monthName;                            echo "<hr>";
14    echo $faker->year;                                 echo "<hr>";
15});
```

Hasil kode program:

```
1589855946
1975-05-05
03/04/2013
09/03/1980
07:58:52
15
Tuesday
08
April
1988
```

Method date() bisa menerima argument berupa pola tanggal seperti yang biasa kita input ke dalam date() function bawaan PHP.

Meskipun sudah menggunakan localization 'id_ID', nama bulan dan nama hari tetap tampil dalam bahasa inggris. Bagaimana cara membuat versi bahasa Indonesia? Kita bisa minta bantuan Carbon:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-date-carbon', function () {
4     $faker = Faker::create('id_ID');
5     $date = \Carbon\Carbon::createFromTimestamp($faker->unixTime)->locale('id');
6     echo $date->dayName;
7     echo "<hr>";
8     echo $date->monthName;
9});
```

Hasil kode program:

```
Minggu
```

Faker

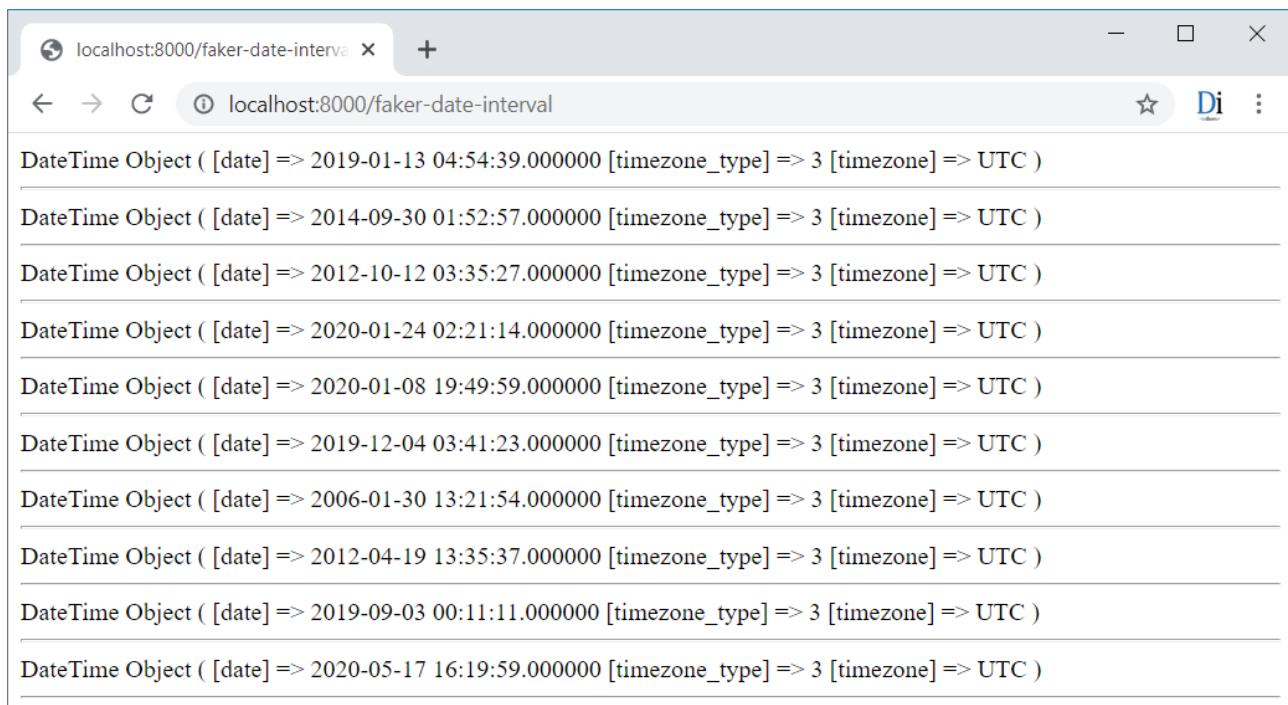
Februari

Teknik yang saya pakai adalah men-generate angka `unixTime()` random dari Faker sebagai nilai input untuk Carbon melalui method `createFromTimestamp()`. Selanjutnya cukup akses property `dayName` dan `monthName` dari object carbon tersebut.

Faker juga menyediakan format khusus untuk men-generate tanggal dalam jangka waktu tertentu:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-date-interval', function () {
4     $faker = Faker::create('id_ID');
5
6     print_r($faker->dateTimeBetween('-10 years'));
7     print_r($faker->dateTimeBetween('-10 years', '-5 years'));
8     print_r($faker->dateTimeBetween('2010-01-01', '2015-01-01'));
9     print_r($faker->dateTimeBetween('2020-01-01', '2020-01-31'));
10
11    print_r($faker->dateTimeInInterval('2020-01-01', '+ 30 days'));
12    print_r($faker->dateTimeInInterval('2020-01-01', '- 1 years'));
13
14    print_r($faker->dateTimeThisCentury());
15    print_r($faker->dateTimeThisDecade());
16    print_r($faker->dateTimeThisYear());
17    print_r($faker->dateTimeThisMonth());
18});
```



Gambar: Hasil generate tanggal acak menggunakan Faker

Hasil akhir dari semua method ini berbentuk **DateTime** object, sehingga nantinya bisa kita input ke dalam Carbon object.

Method `dateTimeBetween()` bisa menerima 2 buah argument opsional, yakni tanggal awal dan tanggal akhir. Tanggal ini bisa berbentuk string sebagai penentu waktu, misalnya `dateTimeBetween(' -10 years')` akan men-generate tanggal acak dalam jangka waktu 10 tahun sebelum tanggal hari ini. Atau `dateTimeBetween(' -10 years', ' -5 years')` akan men-generate tanggal acak dalam jangka waktu 10 tahun sampai 5 tahun sebelum tanggal hari ini.

Method `dateTimeInInterval()` juga menerima 2 argument, namun argument kedua berupa interval yang relatif terhadap tanggal pertama. Sebagai contoh, `dateTimeInInterval('2020-01-01', '+ 30 days')` akan men-generate tanggal acak antara tanggal 01 Januari 2020 hingga 30 hari sesudahnya.

Sesuai dengan namanya, method `dateTimeThisCentury()` akan men-generate satu tanggal acak dalam abad ini, begitu juga dengan method `dateTimeThisDecade()`, `dateTimeThisYear()` dan `dateTimeThisMonth()` yang akan men-generate tanggal acak dalam satu dekade, satu tahun, dan dalam satu bulan ini.

NIK Format

Yang cukup unik dan akan sangat berguna, Faker juga menyediakan format NIK (Nomor Induk Kependudukan) seperti yang ada di KTP Indonesia. Ini hanya bisa dilakukan jika Faker object di instansiasi dengan localization '`id_ID`':

```
routes\web.php

1 <?php
2 ...
3 Route::get('/faker-nik', function () {
4     $faker = Faker::create('id_ID');
5
6     echo $faker->nik();           echo "<br>";
7     echo $faker->nik('male');    echo "<br>";
8     echo $faker->nik('female');  echo "<br>";
9
10    $tgl_lahir = $faker->dateTimeInInterval('1990-01-01', '+5 years');
11
12    echo $faker->nik('male', $tgl_lahir);           echo "<br>";
13    echo $faker->nik('female', $tgl_lahir);         echo "<br>";
14});
```

Hasil kode program:

```
7108743001024341
1301080505079049
7301035106101896
7312440610911145
1109514610913717
```

NIK yang ada di KTP terdiri dari 16 digit angka dengan aturan sebagai berikut:

2 digit kode provinsi + 2 digit kode kota/kabupaten+ 2 digit kode kecamatan + 6 digit tanggal lahir dalam format ddmmyy (untuk perempuan tanggal lahir di tambah 40) + 4 digit nomor urut yang dimulai dari 0001.

Faker menyediakan method `nik()`, `nik('male')`, dan `nik('female')`. Ketiganya dipakai untuk men-generate NIK laki-laki atau perempuan, NIK laki-laki saja, dan NIK perempuan saja.

Method NIK juga bisa menerima argument kedua berupa string tanggal lahir. Di baris 10 saya mengisi variabel `$tanggal_lahir` dengan DateTime object untuk tanggal 01 Januari 1990 hingga 5 tahun sesudahnya. Variabel `$tanggal_lahir` ini kemudian menjadi argument kedua dalam method `nik()` di baris 13 dan 14.

Hasilnya, method `nik('male',$tgl_lahir)` dan `nik('female',$tgl_lahir)` akan men-generate nomor NIK untuk penduduk yang lahir antara 1990 sampai 1995.

Materi tentang NIK ini menutup pembahasan tentang Faker format. Meskipun sudah banyak yang kita bahas, Faker masih menyediakan berbagai format lain. Untuk lebih detail bisa mengunjungi dokumentasi Faker di github.com/fzaninotto/Faker.

6.4. Faker Seed

Ketika menjalankan kode yang sama beberapa kali, Faker akan men-generate data random (selalu berubah). Jika kita butuh data yang selalu konsisten, bisa menjalankan method `$faker->seed()` sebelum mengakses Faker format:

```
routes\web.php

1 <?php
2 ...
3 Route::get('/faker-seed', function () {
4     $faker = Faker::create('id_ID');
5     $faker->seed(999);
6
7     echo $faker->nik();      echo "<hr>";
8     echo $faker->name();    echo "<hr>";
9     echo $faker->address(); echo "<hr>";
10    echo $faker->email();   echo "<hr>";
11    echo $faker->company(); echo "<hr>";
12});
```

Hasil kode program:

```
1605154604068785
Iriana Wahyuni S.Gz
Psr. Laswi No. 756, Sorong 72529, DIY
maimunah.saragih@anggriawan.info
```

Di baris 5 saya menginput method `seed()` dengan argument 999. Maka hasil data yang di generate oleh Faker akan selalu sama setiap kali halaman di refresh, termasuk jika anda coba jalankan kode di atas. Jika ingin variasi hasil lain, silahkan isi method `seed()` dengan angka berbeda, misalnya 100, 9, atau 1234.

6.5. Faker Modifiers

Modifier terdiri dari beberapa method tambahan untuk mengatur hasil akhir Faker format.

Unique Modifiers

Method `unique()` berfungsi untuk memastikan angka yang di generate tidak berulang dalam sekali tampil. Ini akan lebih mudah dijelaskan dengan contoh praktek.

Silahkan pelajari sebentar kode program berikut:

```
routes\web.php

1 <?php
2 ...
3 Route::get('/faker-loop', function () {
4     $faker = Faker::create('id_ID');
5     for ($i=0; $i<10; $i++) {
6         echo $faker->randomDigit . '# ';
7     }
8});
```

Hasil kode program:

```
9 # 8 # 7 # 5 # 2 # 8 # 2 # 1 # 0 # 8 #
```

Di sini saya membuat perulangan sebanyak 10 kali, dimana dalam setiap iterasi akan dihasilkan 1 digit angka acak dengan perintah `$faker->randomDigit`.

Terlihat bahwa digit yang tampil bersifat random dan tidak berhubungan satu sama lain. Maksudnya jika angka pertama muncul 9, dalam iterasi selanjutnya bisa keluar angka 9 lagi. Hasil dari setiap pemanggilan perintah `$faker->randomDigit` memiliki peluang tetap antara 0 - 9.

Bagaimana jika saya ingin agar dalam 10 kali perulangan tidak boleh terdapat angka ganda? Inilah fungsi dari method `unique()`:

```
routes\web.php

1 <?php
2 ...
3 Route::get('/faker-unique', function () {
```

Faker

```
4 $faker = Faker::create('id_ID');
5 for ($i=0; $i<10; $i++) {
6     echo $faker->unique()->randomDigit . ' # ';
7 }
8});
```

Hasil kode program:

```
7 # 8 # 6 # 2 # 0 # 9 # 3 # 5 # 4 # 1 #
```

Kali ini angka yang dihasilkan bersifat unik, yakni tidak ada angka sama yang boleh muncul lebih dari 1 kali. Ini ibarat kita memiliki 10 angka dalam 1 kantong. Ketika 1 angka sudah diambil, isi kantong akan berkurang dan angka tersebut tidak bisa diambil lagi.

Namun logika di atas juga memiliki batasan. Bisakah anda menebak kenapa kode berikut menghasilkan error?

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-unique-error', function () {
4     $faker = Faker::create('id_ID');
5     for ($i=0; $i<11; $i++) {
6         echo $faker->unique()->randomDigit . ' # ';
7     }
8});
```

Hasil kode program:

```
OverflowException
Maximum retries of 10000 reached without finding a unique value
```

Error terjadi karena saya mengulang method `$faker->unique()->randomDigit` sebanyak 11 kali. Padahal angka unik yang bisa di-generate oleh `randomDigit` maksimal hanya 10 buah.

Method `unique()` juga sering digabung dengan method `randomElement()`. Method `randomElement()` sendiri sudah pernah kita bahas pada materi Alphanumeric Format, yang berfungsi untuk mengambil 1 element acak dari sebuah array:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/faker-random-array', function () {
4     $faker = Faker::create('id_ID');
5     $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
6
7     for ($i=0; $i<5; $i++) {
8         echo $faker->randomElement($jurusan) . ' # ';
9     }
10});
```

Hasil kode program:

```
Ilmu Komputer # Teknik Informatika # Ilmu Komputer # Teknik Informatika #
Sistem Informasi #
```

Di baris 5 saya mengisi variabel `$jurusan` dengan array yang terdiri dari 3 element: "Ilmu Komputer", "Teknik Informatika", dan "Sistem Informasi". Kemudian menjalankan method `$faker->randomElement($jurusan)` sebanyak 5 kali menggunakan `for` loop. Hasilnya, muncul 5 kali string jurusan yang diambil dari array `$jurusan` secara acak.

Dengan memadukan method `randomElement()` dengan `unique()`, kita bisa buat pembatasan string jurusan yang hanya boleh tampil sekali saja:

`routes\web.php`

```
1 <?php
2 ...
3 Route::get('/faker-unique-array', function () {
4     $faker = Faker::create('id_ID');
5     $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
6
7     for ($i=0; $i<3; $i++) {
8         echo $faker->unique()->randomElement($jurusan). ' # ';
9     }
10});
```

Hasil kode program:

```
Ilmu Komputer # Teknik Informatika # Sistem Informasi #
```

Dengan cara ini kita bisa mengatur cara pengambilan element dari sebuah array.

Optional Modifiers

Method `optional()` bisa dipakai agar data yang di-generate boleh menghasilkan nilai NULL atau kosong. Berikut contoh penggunaannya:

`routes\web.php`

```
1 <?php
2 ...
3 Route::get('/faker-optional-1', function () {
4     $faker = Faker::create('id_ID');
5     $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
6
7     for ($i=0; $i<10; $i++) {
8         var_dump($faker->optional()->randomElement($jurusan)); echo "<br>";
9     }
10});
```

Hasil kode program:

```
NULL
NULL
NULL
string(13) "Ilmu Komputer"
string(16) "Sistem Informasi"
NULL
NULL
NULL
NULL
string(13) "Ilmu Komputer"
```

Kode di atas mirip seperti sebelumnya, tapi kali ini saya melakukan perulangan sebanyak 10 kali. Selain itu terdapat tambahan method `optional()` di baris 8. Dengan kode ini maka perintah `randomElement($jurusan)` bisa mengembalikan nilai `NULL`.

Secara default peluang kemunculan nilai `NULL` adalah 50%. Kita bisa mengatur peluang ini dengan menginput angka pecahan sebagai argument method `optional()`:

```
routes\web.php

1 <?php
2 ...
3 Route::get('/faker-optimal-2', function () {
4     $faker = Faker::create('id_ID');
5     $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
6
7     for ($i=0; $i<10; $i++) {
8         var_dump($faker->optional(0.8)->randomElement($jurusan)); echo "<br>";
9     }
10});
```

Hasil kode program:

```
string(13) "Ilmu Komputer"
string(13) "Ilmu Komputer"
NULL
string(13) "Ilmu Komputer"
string(13) "Ilmu Komputer"
string(16) "Sistem Informasi"
string(13) "Ilmu Komputer"
string(16) "Sistem Informasi"
string(18) "Teknik Informatika"
NULL
```

Sekarang saya menambah argument `0.8` ke dalam method `optional()`. Ini akan men-set peluang kemunculan string menjadi 80%, sedangkan peluang kemunculan `NULL` hanya 20%. Dengan kata lain jika dijalankan 10 kali, nilai `NULL` cenderung akan muncul 2 kali saja.

Method `optional()` juga menerima argument kedua berupa string yang tampil sebagai pengganti nilai `NULL`. Berikut contohnya:

```
routes\web.php
```

```

1  <?php
2  ...
3  Route::get('/faker-optimal-3', function () {
4      $faker = Faker::create('id_ID');
5      $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
6
7      for ($i=0; $i<10; $i++) {
8          echo $faker->optional(0.2,"Tidak memilih Jurusan")->randomElement($jurusan);
9          echo "<br>";
10     }
11 });

```

Hasil kode program:

```

Tidak memilih Jurusan
Tidak memilih Jurusan
Sistem Informasi
Tidak memilih Jurusan
Tidak memilih Jurusan
Tidak memilih Jurusan
Ilmu Komputer
Tidak memilih Jurusan
Tidak memilih Jurusan
Tidak memilih Jurusan

```

Dengan menulis `optional(0.2,"Tidak memilih Jurusan")`, maka string "Tidak memilih Jurusan" akan tampil sebagai pengganti nilai NULL. Dan karena saya menginput argument pertama dengan nilai 0.2, maka peluang kemunculan string jurusan sekarang hanya 20% saja, sedangkan kemunculan string "Tidak memilih Jurusan" ada di 80%.

6.6. Faker dan Eloquent

Saatnya kita coba terapkan library Faker untuk kode program yang lebih real. Misalnya dengan men-generate data mahasiswa menggunakan Faker yang kemudian diinput ke dalam tabel `mahasiswa`.

Tabel `mahasiswa` butuh 4 nilai awal untuk kolom **nim**, **nama**, **tanggal_lahir** dan **ipk**. Berikut kode Faker yang bisa dipakai:

```
routes\web.php
```

```

1  <?php
2  ...
3  Route::get('/faker-mahasiswa', function () {
4      $faker = Faker::create('id_ID');
5
6      echo $faker->numerify('10#####');           echo "<br>";
7      echo $faker->name;                          echo "<br>";
8
9      $tgl_lahir = $faker->dateTimeInInterval('1999-01-01', '+1 years');

```

Faker

```
10 echo \Carbon\Carbon::parse($tgl_lahir)->isoFormat('D-M-YYYY'); echo "<br>";
11
12 echo $faker->randomFloat(2, 2, 4);           echo "<br>";
13});
```

Hasil kode program:

```
10578637
Estiawan Budiman
8-11-1999
2.37
```

Untuk men-generate nilai nim saya menggunakan format `$faker->numerify('10#####')`. Angka 10 di awal hanya agar seragam saja, dimana setiap nim mahasiswa akan tampil dalam format 10xxxxxx.

Di baris 7 nilai nama mahasiswa akan di generate menggunakan format `$faker->name`.

Yang agak panjang adalah proses generate nilai tanggal lahir. Supaya datanya terasa lebih asli, saya ingin tanggal lahir mahasiswa antara tahun 1999 – 2000. Batasan ini bisa di dapat dari perintah `$faker->dateTimeInInterval('1999-01-01', '+1 years')`. Hasilnya kemudian diinput ke dalam Carbon agar menghasilkan format D-M-YYYY.

Terakhir saya men-generate nilai ipk dengan format `$faker->randomFloat(2, 2, 4)`. Ini akan menghasilkan angka pecahan dengan 2 tempat desimal dan digit angka bulat antara 2 – 4.

Sekarang saatnya input data ini ke database. Sebelum itu, silahkan buat tabel `mahasiswas` jika belum tersedia. Berikut perintah yang diperlukan untuk membuat model Mahasiswa beserta file migration:

```
php artisan make:model Mahasiswa -m
```

Lalu buka file migration `mahasiswas` dan tambah pendefinisian 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->date('tanggal_lahir');
8         $table->decimal('ipk',3,2)->default(1.00);
9         $table->timestamps();
10    });
11 }
```

Jalankan migration dengan perintah:

```
php artisan migrate
```

Agar tidak bermasalah dengan proses *mass assignment*, tambah property `protected $guarded = []` ke dalam model Mahasiswa:

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 }
```

Tabel mahasiswas sudah tersedia, saatnya jalankan proses input dari Faker:

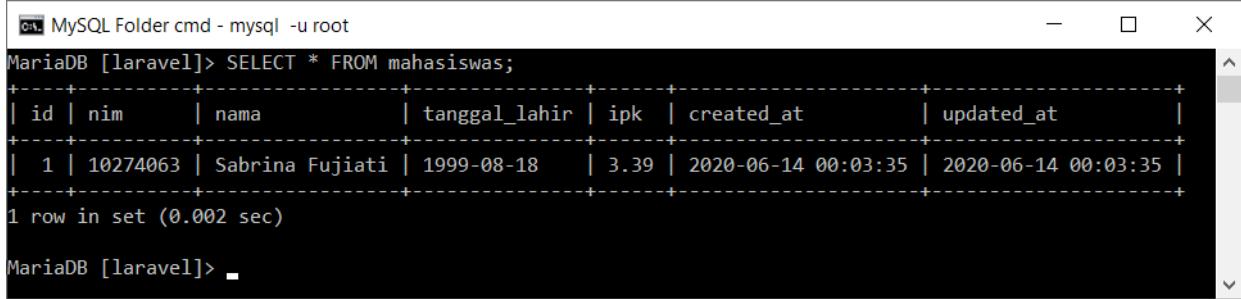
routes\web.php

```

1 <?php
2 ...
3 Route::get('/faker-mahasiswa-db', function () {
4     $faker = Faker::create('id_ID');
5
6     App\Models\Mahasiswa::create(
7         [
8             'nim' => $faker->numerify('10#####'),
9             'nama' => $faker->name,
10            'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01', '+ 3 years'),
11            'ipk' => $faker->randomFloat(2, 2, 4),
12        ]
13    );
14
15    return "Data mahasiswa berhasil ditambah";
16});
```

Untuk kolom `tanggal_lahir` kita bisa langsung input DateTime object hasil dari `$faker->dateTimeInInterval()`, tidak perlu dikonversi lagi menggunakan Carbon.

Silahkan akses URL <http://localhost:8000/faker-mahasiswa-db>, jika tampil teks "Data mahasiswa berhasil ditambah", artinya kode di atas sukses berjalan. Cek ke database untuk memastikan:



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 10274063 | Sabrina Fujiati | 1999-08-18 | 3.39 | 2020-06-14 00:03:35 | 2020-06-14 00:03:35 |
+----+-----+-----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]>
```

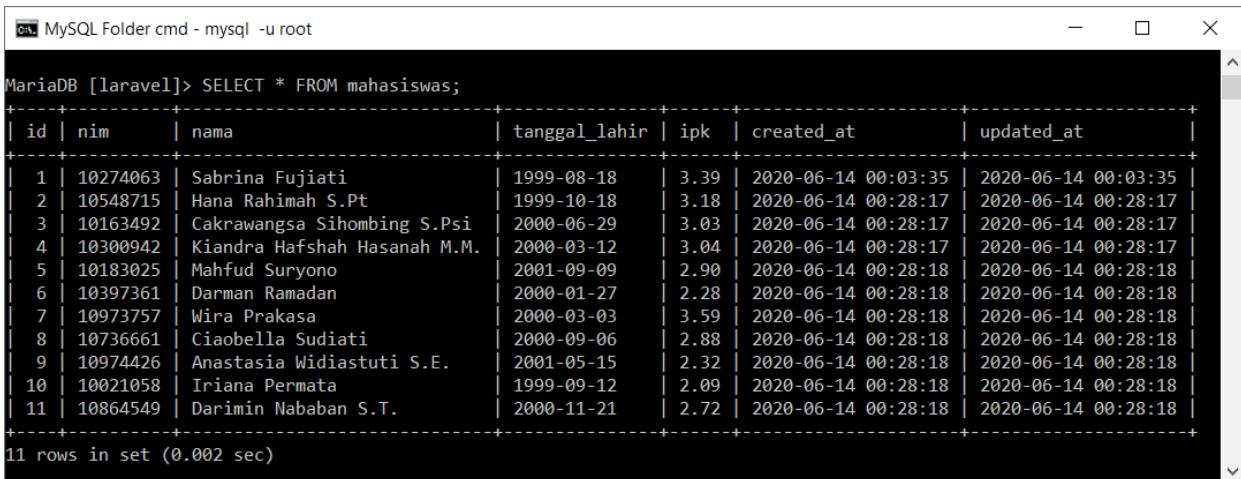
Gambar: Isi tabel mahasiswa

Sip, satu data sample sudah berhasil masuk ke tabel `mahasiswa`. Bagaimana jika ingin menambah hingga 10 data? Tidak masalah, tinggal lakukan perulangan sebanyak 10 kali:

`routes\web.php`

```
1 <?php
2 ...
3 Route::get('/faker-mahasiswa-db-loop', function () {
4     $faker = Faker::create('id_ID');
5
6     for ($i=0; $i<10; $i++) {
7         App\Models\Mahasiswa::create(
8             [
9                 'nim' => $faker->numerify('10#####'),
10                'nama' => $faker->name,
11                'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01', '+ 3 years'),
12                'ipk' => $faker->randomFloat(2, 2, 4),
13            ]
14        );
15    }
16
17    return "Data mahasiswa berhasil ditambah";
18});
```

Akses URL `http://localhost:8000/faker-mahasiswa-db-loop` di web browser untuk menjalankan kode di atas, lalu cek ke database:



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 10274063 | Sabrina Fujiati | 1999-08-18 | 3.39 | 2020-06-14 00:03:35 | 2020-06-14 00:03:35 |
| 2 | 10548715 | Hana Rahimah S.Pt | 1999-10-18 | 3.18 | 2020-06-14 00:28:17 | 2020-06-14 00:28:17 |
| 3 | 10163492 | Cakrawangsa Sihombing S.Psi | 2000-06-29 | 3.03 | 2020-06-14 00:28:17 | 2020-06-14 00:28:17 |
| 4 | 10300942 | Kiandra Hafshah Hasanah M.M. | 2000-03-12 | 3.04 | 2020-06-14 00:28:17 | 2020-06-14 00:28:17 |
| 5 | 10183025 | Mahfud Suryono | 2001-09-09 | 2.90 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
| 6 | 10397361 | Darman Ramadan | 2000-01-27 | 2.28 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
| 7 | 10973757 | Wira Prakasa | 2000-03-03 | 3.59 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
| 8 | 10736661 | Ciaobella Sudiaty | 2000-09-06 | 2.88 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
| 9 | 10974426 | Anastasia Widiasutti S.E. | 2001-05-15 | 2.32 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
| 10 | 10021058 | Iriana Permata | 1999-09-12 | 2.09 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
| 11 | 10864549 | Darimin Nababan S.T. | 2000-11-21 | 2.72 | 2020-06-14 00:28:18 | 2020-06-14 00:28:18 |
+----+-----+-----+-----+-----+-----+
11 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa setelah proses loop

Dengan bantuan Faker, kita bisa generate ratusan bahkan ribuan data dengan mudah.

Namun ada sedikit data yang masih kurang pas, yakni bagian nama yang ternyata memiliki titel seperti S.Pt, S.Psi, atau M.M. Seharusnya nama mahasiswa tidak memiliki titel seperti ini, meskipun tidak menutup kemungkinan ada sarjana yang ingin kuliah kembali.

Supaya tidak memiliki titel, saya akan ganti format `$faker->name` dengan gabungan `$faker->firstName + $faker->lastName`. Silahkan reset tabel `mahasiswas` dengan perintah `php artisan migrate:fresh`, lalu jalankan kode berikut:

`routes\web.php`

```

1  <?php
2  ...
3  Route::get('/faker-mahasiswa-db-name', function () {
4      $faker = Faker::create('id_ID');
5
6      for ($i=0; $i<100; $i++) {
7          App\Models\Mahasiswa::create(
8              [
9                  'nim' => $faker->numerify('10#####'),
10                 'nama' => $faker->firstName." ".$faker->lastName,
11                 'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01', '+ 3 years'),
12                 'ipk' => $faker->randomFloat(2, 2, 4),
13             ]
14         );
15     }
16
17     return "Data mahasiswa berhasil ditambah";
18 });

```

Ketika URL `http://localhost:8000/faker-mahasiswa-db-name` di akses, akan butuh waktu yang sedikit lebih lama (antara 5 – 10 detik), ini karena saya men-generate 100 mahasiswa:

91	10088475	Abyasa Suryono	2001-01-30	3.69	2020-06-14 00:39:46	2020-06-14 00:39:46		
92	10704310	Gambira Suwarno	2001-09-09	2.26	2020-06-14 00:39:46	2020-06-14 00:39:46		
93	10791902	Zahra Novitasari	2001-03-31	3.42	2020-06-14 00:39:46	2020-06-14 00:39:46		
94	10738523	Novi Sihombing	2001-07-18	2.83	2020-06-14 00:39:46	2020-06-14 00:39:46		
95	10376960	Gambira Saputra	2001-01-18	2.01	2020-06-14 00:39:46	2020-06-14 00:39:46		
96	10673118	Fathonah Damaniik	2000-09-04	3.93	2020-06-14 00:39:46	2020-06-14 00:39:46		
97	10268938	Makuta Utami	1999-08-27	3.23	2020-06-14 00:39:46	2020-06-14 00:39:46		
98	10111780	Kasiran Gunarto	2001-08-01	2.22	2020-06-14 00:39:46	2020-06-14 00:39:46		
99	10981091	Galiono Anggriawan	1999-06-04	2.50	2020-06-14 00:39:46	2020-06-14 00:39:46		
100	10725041	Najib Fujiati	1999-05-28	2.91	2020-06-14 00:39:46	2020-06-14 00:39:46		

100 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswas setelah proses loop 100 kali

Sekarang nama mahasiswa sudah tidak lagi memiliki titel.

Namun jika diperhatikan dengan seksama kita memiliki masalah baru. Karena nama depan dan nama belakang di generate terpisah, ada kemungkinan nama depan laki-laki disambung

dengan nama belakang perempuan.

Ini bisa diakali dengan men-generate nama secara berpasangan, yakni `$faker->firstNameMale." ".$faker->lastNameMale` dan `$faker->firstNameFemale." ".$faker->lastNameFemale`, lalu dijalankan bergantian. Namun saya merasa ini tidak terlalu masalah, silahkan jika anda ingin buat yang seperti ini.

Untuk data yang lebih valid lagi, bisa juga mengubah nilai kolom `created_at` dan `update_at` agar mencerminkan data yang diisi secara berkala.

Sepanjang bab ini kita telah membahas Faker Library yang menawarkan cara cepat untuk men-generate data acak. Seperti yang terlihat pada percobaan terakhir, data ini sangat pas sebagai *sample* awal di database.

Namun di manakah kode untuk proses generate ini sebaiknya di tempatkan? Laravel menjawabnya dengan fitur yang disebut sebagai **Seeder**. Inilah bahasan kita 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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

7. Seeder

Pada beberapa bab sebelum ini, kita menempatkan proses generate tabel `mahasiswa` di route dan dijalankan dari web browser (dengan mengetik alamat URL). Laravel sebenarnya sudah menyediakan fitur khusus untuk keperluan ini, yang disebut sebagai **Seeder**.

Dalam bab kali ini kita akan bahas apa itu Seeder dan bagaimana cara penggunaannya.

Sebagai bahan praktek, boleh dilanjutkan dari bab sebelumnya (Faker). Namun agar tidak terdapat kode yang bentrok, silahkan hapus semua isi file `route\web.php` lalu jalankan perintah `php artisan migrate:fresh` agar tabel `mahasiswa` kembali kosong.

7.1. Pengertian Seeder

Seeder (atau di dokumentasi Laravel disebut juga sebagai **Database: Seeding**), adalah sebuah file khusus yang berguna untuk proses pengisian data awal ke tabel database. Jika `migration` dipakai untuk proses generate struktur tabel, maka `seeder` adalah tempat men-generate data tabel tersebut.

Proses generate data atau seeder bisa ditulis di 2 tempat:

1. File 'master seeder' di `database\seeders\DatabaseSeeder.php`.
2. File seeder terpisah.

Kita akan bahas kedua cara ini.

7.2. Menjalankan Seeder

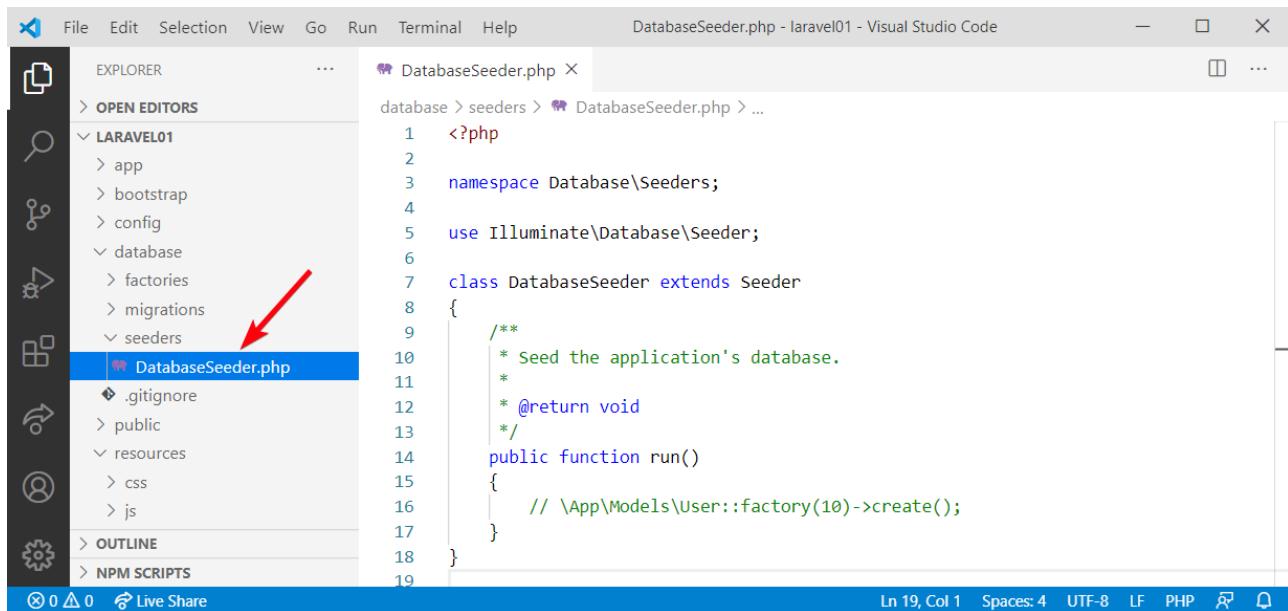
Cara pertama untuk menjalankan seeder adalah dari file master seeder yang berada di `database\seeders\DatabaseSeeder.php`. File `DatabaseSeeder.php` sudah tersedia pada saat proses instalasi Laravel, silahkan buka file ini:

`database\seeders\DatabaseSeeder.php`

```
1 <?php  
2  
3 namespace Database\Seeders;  
4  
5 use Illuminate\Database\Seeder;  
6
```

Seeder

```
7 class DatabaseSeeder extends Seeder
8 {
9     /**
10      * Seed the application's database.
11      *
12      * @return void
13      */
14     public function run()
15     {
16         // \App\Models\User::factory(10)->create();
17     }
18 }
```



Gambar: Isi file DatabaseSeeder.php

Isi file DatabaseSeeder.php tidak terlalu panjang, terdiri dari perintah `namespace Database\Seeders` di baris 3, `use Illuminate\Database\Seeder` di baris 5, lalu pembuatan `DatabaseSeeder` class yang men-extends `Seeder` class.

Di dalam class `DatabaseSeeder` terdapat satu method bernama `run()` yang hanya memiliki 1 baris perintah. Perintah itu pun di non-aktifkan menjadi komentar (baris 16).

Sekarang silahkan modifikasi isi file `DatabaseSeeder.php` menjadi sebagai berikut:

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use App\Models\Mahasiswa;
7
8 class DatabaseSeeder extends Seeder
9 {
```

Seeder

```
10     public function run()
11     {
12         Mahasiswa::create(
13             [
14                 'nim' => '19003036',
15                 'nama' => 'Sari Citra Lestari',
16                 'tanggal_lahir' => '2001-12-31',
17                 'ipk' => 3.62,
18             ]
19         );
20     }
21 }
```

Tambahan kode program ada di baris 6 dan 12 – 19. Di baris 6 berupa perintah import `use App\Models\Mahasiswa` yang diperlukan untuk mempersingkat penulisan Eloquent ketika mengakses model Mahasiswa.

Kemudian di baris 12 – 19 terdapat kode eloquent untuk proses input 1 data mahasiswa bernama 'Sari Citra Lestari' ke dalam model Mahasiswa. Perintah *mass assignment* ini sudah sering kita pakai, namun selama ini berada di route atau controller.

Save file di atas dan saatnya jalankan file seeder. Sebelum itu pastikan tabel `mahasiswas` sudah tersedia dan model `Mahasiswa` juga sudah ada. Jika perlu, kosongkan isi tabel `mahasiswas` terlebih dahulu.

Berikut perintah untuk menjalankan file seeder:

```
php artisan db:seed
```

```
C:\xampp\htdocs\laravel01>php artisan db:seed
Database seeding completed successfully.

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

Gambar: Menjalankan seeder

Jika tampil pesan "Database seeding completed successfully", artinya proses seeder sudah sukses berjalan. Untuk memastikan, cek isi tabel `mahasiswas`:

```
MySQL [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+
| id | nim   | nama        | tanggal_lahir | ipk   | created_at      | updated_at      |
+----+-----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.62 | 2020-06-16 07:56:08 | 2020-06-16 07:56:08 |
+----+-----+-----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi tabel mahasiswas

Sip, data 'Sari Citra Lestari' sudah berhasil di input ke database.

Khusus untuk proses *mass assignment* yang dilakukan di seeder, kita tidak perlu menambah property `$fillable` atau `$guarded` ke dalam Model.

Selanjutnya, bagaimana jika ingin men-generate banyak data sekaligus? Tidak masalah, tinggal tulis perintah eloquent yang diperlukan ke dalam method `run()`:

database\seeders\DatabaseSeeder.php

```

1  <?php
2
3  namespace Database\Seeders;
4  use Illuminate\Database\Seeder;
5  use App\Models\Mahasiswa;
6  use App\Models\User;
7  use Illuminate\Support\Facades\Hash;
8
9  class DatabaseSeeder extends Seeder
10 {
11     public function run()
12     {
13         Mahasiswa::create(
14             [
15                 'nim' => '19003036',
16                 'nama' => 'Sari Citra Lestari',
17                 'tanggal_lahir' => '2001-12-31',
18                 'ipk' => 3.62,
19             ]
20         );
21
22         Mahasiswa::create(
23             [
24                 'nim' => '19021044',
25                 'nama' => 'Rudi Permana',
26                 'tanggal_lahir' => '2000-08-22',
27                 'ipk' => 2.99,
28             ],
29         );
30         Mahasiswa::create(
31             [
32                 'nim' => '19002032',
33                 'nama' => 'Rina Kumala Sari',
34                 'tanggal_lahir' => '2000-06-28',
35                 'ipk' => 3.82,
36             ],
37         );
38         Mahasiswa::create(
39             [
40                 'nim' => '18012012',
41                 'nama' => 'James Situmorang',
42                 'tanggal_lahir' => '1999-04-02',
43                 'ipk' => 2.74,
44             ]

```

Seeder

```
45     );
46
47     User::create(
48         [
49             'name' => 'Admin',
50             'email' => 'admin@gmail.com',
51             'password' => Hash::make('qwerty'),
52         ]
53     );
54 }
55 }
```

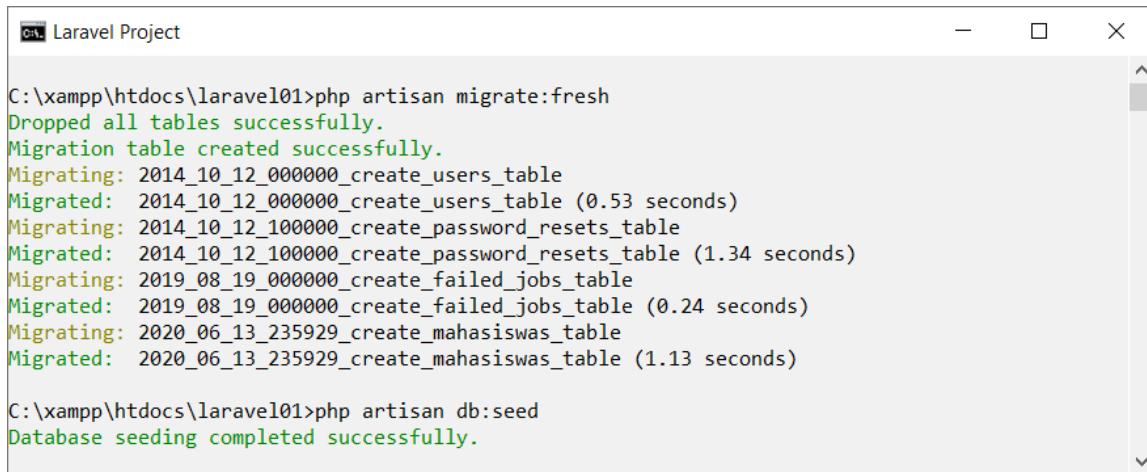
Dalam kode program ini saya memodifikasi file `DatabaseSeeder.php` dengan menambah beberapa perintah `insert eloquent`, termasuk untuk tabel `users` bawaan Laravel. Karena terdapat proses generate data tabel `users`, perlu tambahan perintah import `use App\Models\User` di baris 6 serta import facade class `Illuminate\Support\Facades\Hash` di baris 7.

Jika anda masih ingat, perintah ini sama persis seperti materi Persiapan Awal di bab **Accessor dan Mutator**, hanya saja sekarang di pindah dari route ke dalam seeder.

Silahkan buka cmd dan jalankan 2 perintah berikut untuk mengosongkan tabel dan menjalankan ulang seeder:

```
php artisan migrate:fresh
```

```
php artisan db:seed
```



```
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 (0.53 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (1.34 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.24 seconds)
Migrating: 2020_06_13_235929_create_mahasiswa_table
Migrated: 2020_06_13_235929_create_mahasiswa_table (1.13 seconds)

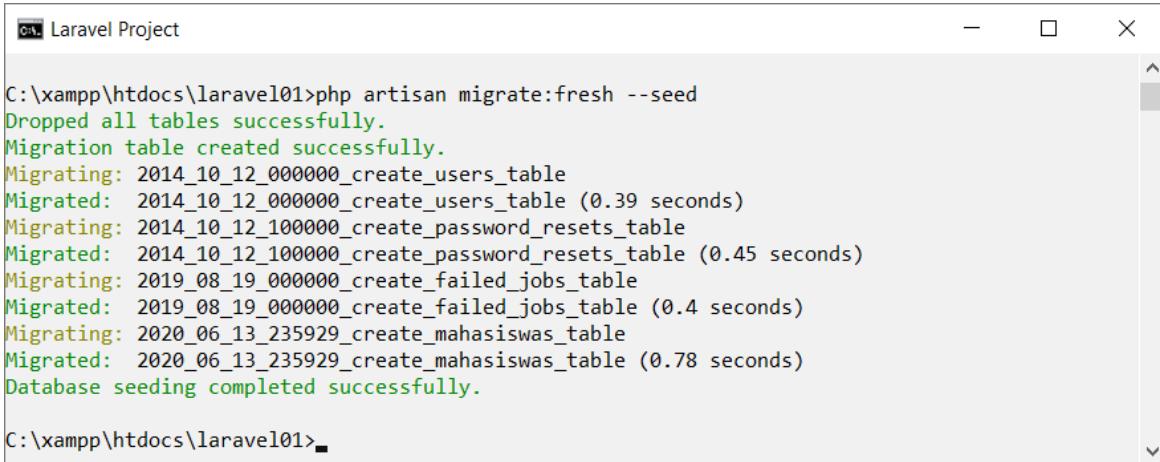
C:\xampp\htdocs\laravel01>php artisan db:seed
Database seeding completed successfully.
```

Gambar: Proses refresh tabel dan menjalankan seeder

Proses `migrate` dan `\Seeder` ini cukup sering dipakai bersamaan. Oleh karena itu Laravel menyediakan perintah singkat untuk menjalankan keduanya sekaligus, yakni dengan menambah flag `--seed` ke dalam perintah `php artisan migrate`:

```
php artisan migrate:fresh --seed
```

Seeder

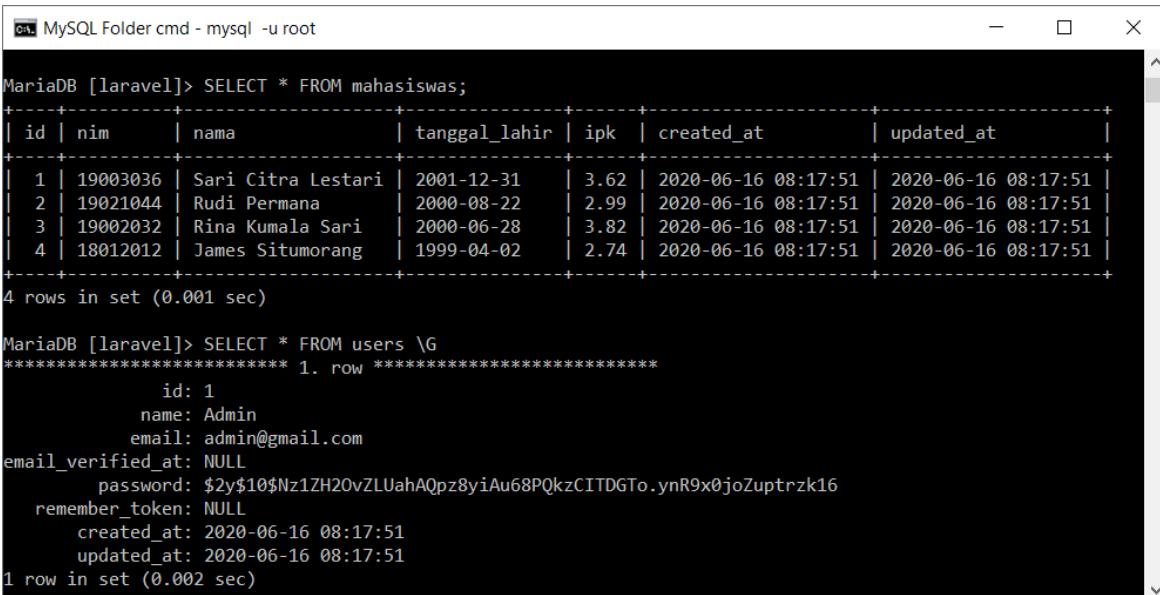


```
C:\xampp\htdocs\laravel01>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.39 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.45 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.4 seconds)
Migrating: 2020_06_13_235929_create_mahasiswa_table
Migrated: 2020_06_13_235929_create_mahasiswa_table (0.78 seconds)
Database seeding completed successfully.

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

Gambar: Perintah gabungan refresh tabel dan menjalankan seeder sekaligus

Hasilnya, di dalam tabel `mahasiswa` akan terdapat 4 data, serta 1 data baru untuk tabel `user`.



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim  | nama | tanggal_lahir | ipk | created_at   | updated_at   |
+----+-----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2001-12-31 | 3.62 | 2020-06-16 08:17:51 | 2020-06-16 08:17:51 |
| 2  | 19021044 | Rudi Permana | 2000-08-22 | 2.99 | 2020-06-16 08:17:51 | 2020-06-16 08:17:51 |
| 3  | 19002032 | Rina Kumala Sari | 2000-06-28 | 3.82 | 2020-06-16 08:17:51 | 2020-06-16 08:17:51 |
| 4  | 18012012 | James Situmorang | 1999-04-02 | 2.74 | 2020-06-16 08:17:51 | 2020-06-16 08:17:51 |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.001 sec)

MariaDB [laravel]> SELECT * FROM users \G
***** 1. row *****
      id: 1
      name: Admin
      email: admin@gmail.com
email_verified_at: NULL
      password: $2y$10$Nz1ZH20vZLUahAQpz8yiAu68PQkzCITDGTo.ynR9x0joZuptrzk16
remember_token: NULL
      created_at: 2020-06-16 08:17:51
      updated_at: 2020-06-16 08:17:51
1 row in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa dan tabel users

Tambahan `--seed` juga bisa dipakai untuk perintah `migrate` lain. Misalnya jika database belum di generate, maka perintah pembuatan tabel + seed adalah:

```
php artisan migrate --seed
```

7.3. Seeder dan Faker

Kita sudah berhasil menjalankan file seeder untuk proses generate tabel `mahasiswa` dan tabel `users`, namun semua data masih ditulis manual. Akan lebih menarik jika data tersebut dibuat secara random menggunakan Faker.

Berikut modifikasi seeder dengan tambahan faker:

Seeder

database\seeders\DatabaseSeeder.php

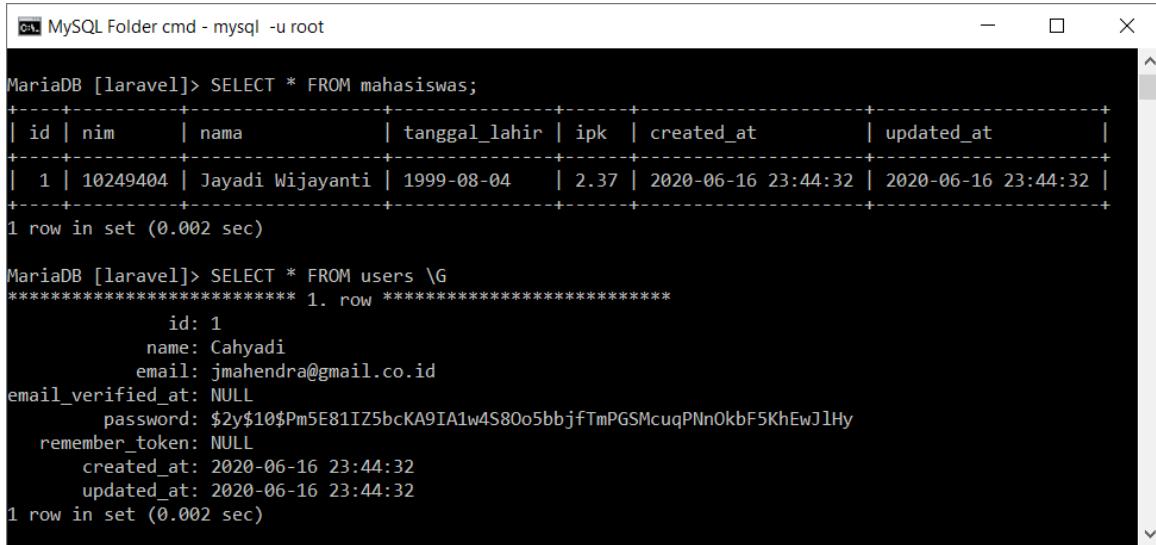
```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use App\Models\Mahasiswa;
7 use App\Models\User;
8 use Illuminate\Support\Facades\Hash;
9 use Faker\Factory as Faker;
10
11 class DatabaseSeeder extends Seeder
12 {
13     public function run()
14     {
15         $faker = Faker::create('id_ID');
16
17         Mahasiswa::create(
18             [
19                 'nim' => $faker->numerify('10#####'),
20                 'nama' => $faker->firstName." ".$faker->lastName,
21                 'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01',
22                                         '+ 3 years'),
23                 'ipk' => $faker->randomFloat(2, 2, 4),
24             ]
25         );
26
27         User::create(
28             [
29                 'name' => $faker->firstName,
30                 'email' => $faker->email,
31                 'password' => Hash::make('qwerty'),
32             ]
33         );
34     }
35 }
36 }
```

Karena akan menggunakan Faker class, maka di baris 9 saya butuh tambahan import namespace `use Faker\Factory as Faker;`. Selanjutnya antara baris 15 – 33 terdapat perintah insert eloquent yang datanya di generate dari Faker.

Proses generate mahasiswa tidak ada masalah karena sudah pernah kita praktekkan di bab Faker. Untuk generate data user, saya menggunakan `$faker->firstName` untuk nama, serta `$faker->email` untuk email. Sedangkan kolom password tetap diinput manual sebagai `Hash::make('qwerty')`.

Jalankan seeder dengan perintah `php artisan migrate:fresh --seed` dan periksa isi database:

Seeder



```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+-----+
| id | nim | nama | tanggal_lahir | ipk | created_at | updated_at |
+----+-----+-----+-----+-----+-----+-----+
| 1 | 10249404 | Jayadi Wijayanti | 1999-08-04 | 2.37 | 2020-06-16 23:44:32 | 2020-06-16 23:44:32 |
+----+-----+-----+-----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM users \G
***** 1. row *****
    id: 1
    name: Cahyadi
    email: jmahendra@gmail.co.id
email_verified_at: NULL
    password: $2y$10$Pm5E81IZ5bcKA9IA1w4S80o5bbjfTmPGSMcuqPNn0kbF5KhEwJlHy
remember_token: NULL
    created_at: 2020-06-16 23:44:32
    updated_at: 2020-06-16 23:44:32
1 row in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa dan users yang di generate dari faker

Oke, data yang berasal dari seeder dan faker sudah berhasil kita input.

Sekarang bagaimana jika test generate 100 data mahasiswa dan 100 data user? Tidak masalah, cukup buat perulangan for ke dalam file seeder:

database\seeders\DatabaseSeeder.php

```
1 <?php
2 ...
3 class DatabaseSeeder extends Seeder
4 {
5     public function run()
6     {
7         $faker = Faker::create('id_ID');
8
9         for ($i=0; $i<100; $i++) {
10
11             Mahasiswa::create(
12                 [
13                     'nim' => $faker->numerify('10#####'),
14                     'nama' => $faker->firstName." ".$faker->lastName,
15                     'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01',
16                                         '+ 3 years'),
17                     'ipk' => $faker->randomFloat(2, 2, 4),
18                 ]
19             );
20
21             User::create(
22                 [
23                     'name' => $faker->firstName,
24                     'email' => $faker->unique()->email,
25                     'password' => Hash::make('qwerty'),
26                 ]
27             );
28     }
}
```

Seeder

```
29
30      }
31 }
```

Kembali jalankan perintah `php artisan migrate:fresh --seed`.

MySQL Folder cmd - mysql -u root					
85	Mulya	aslijan33@yahoo.co.id	NULL	\$2y\$10\$TDmisfpW5yPE38R3U5wA	
86	Juli	zulaikha.permata@yahoo.com	NULL	\$2y\$10\$XlKP1Lp7Ypc6RVHsmY4U	
87	Ibun	puput13@nasyidah.desa.id	NULL	\$2y\$10\$88ZAFw0U4H5Dqpu1c8Cs	
88	Eka	dewi97@yahoo.co.id	NULL	\$2y\$10\$f/7BRWlufbcj2P3uMA71o	
89	Belinda	uchita85@gmail.co.id	NULL	\$2y\$10\$T9jk1ibFhajlfzXcvGpy	
90	Yuni	xpuspasari@yahoo.com	NULL	\$2y\$10\$1dGxL3MK8AHzkiV5WkPT	
91	Paulin	puspasari.pangestu@ wahyuni.go.id	NULL	\$2y\$10\$gI.b0ycbhIN.X5zWw0/k	
92	Talia	asmuni56@gmail.com	NULL	\$2y\$10\$Y5lUT9FYUT9RDxI9vW8j	
93	Wahyu	mila75@rajata.desa.id	NULL	\$2y\$10\$2lhY/.jR0qlqDPnsm2lu	
94	Lasmanto	wandriani@yahoo.co.id	NULL	\$2y\$10\$Jw0CuaWFN1vQbkYPaCK	
95	Estiono	banana30@puspasari.sch.id	NULL	\$2y\$10\$Oq0P5xbmGoHzD509bJe2	
96	Edison	novitasari.bakijan@agustina.asia	NULL	\$2y\$10\$IZndu7yYnberKCI29mba	
97	Edison	rahmawati.farhunnisa@hasanah.sch.id	NULL	\$2y\$10\$GVn.ROXA5zcmLRjetR1u	
98	Lasmanto	wzulkarnain@gmail.co.id	NULL	\$2y\$10\$zDxvHbw6sB.EGUH8wtkh	
99	Balijan	aisyah69@uliarti.ac.id	NULL	\$2y\$10\$HYJ.LciocD8h6bUm7a0	
100	Darijan	paris.damanik@hidayat.info	NULL	\$2y\$10\$VGJGvC7GgUblBSDQ3Yvs	

Gambar: Isi tabel users

Proses seeding butuh waktu beberapa saat (sekitar 10 – 15 detik), ini karena kita men-generate 200 data sekaligus. Cara ini sangat praktis dan cocok untuk membuat data awal dari aplikasi yang sedang dirancang.

7.4. Membuat File Seeder

Apa yang sudah kita praktekkan sebenarnya sudah cukup untuk membuat seeder sederhana. Namun jika terdapat banyak tabel yang harus di generate, Laravel menyediakan cara membuat file seeder terpisah.

Untuk membuat file seeder, jalankan perintah `php artisan` dengan format berikut:

```
php artisan make:seeder <namaFileSeeder>
```

Nama file seeder boleh bebas, namun agar lebih jelas dan mengikuti kebiasaan programmer Laravel, bisa menggunakan format: `<NamaTabel>TableSeeder`.

Nama tabel boleh ditulis menggunakan versi singular tanpa tambahan 's' di akhir kata, atau versi plural dengan tambahan 's'. Misalnya untuk tabel `mahasiswa`, nama file seeder menjadi `MahasiswaTableSeeder` atau `MahasiswaTableSeeder`.

Mari kita coba, silahkan ketik perintah berikut di cmd:

```
php artisan make:seeder MahasiswaTableSeeder
```

Seeder

```
C:\xampp\htdocs\laravel01>php artisan make:seeder MahasiswaTableSeeder
Seeder created successfully.

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

Gambar: membuat file seeder

Setelah menjalankan perintah ini, file bernama `MahasiswaTableSeeder.php` akan muncul di folder `database\seeders\`:

```
File Edit Selection View Go Run Terminal Help MahasiswaTableSeeder.php - laravel01 - Visual Studio Code
EXPLORER
OPEN EDITORS
LARAVEL01
app
bootstrap
config
database
factories
migrations
seeders
DatabaseSeeder.php
MahasiswaTableSeeder.php
.gitignore
public
resources
outline
NPM SCRIPTS
MahasiswaTableSeeder.php
```

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class MahasiswaTableSeeder extends Seeder
8 {
9
10    /**
11     * Run the database seeds.
12     *
13     * @return void
14     */
15    public function run()
16    {
17        //
18    }
}
```

Gambar: Isi file `MahasiswaTableSeeder.php`

Isi file `MahasiswaTableSeeder.php` sangat mirip seperti file master seeder, yakni perintah `namespace Database\Seeders` di baris awal, kemudian import class `Illuminate\Database\Seeder`, serta class `MahasiswaTableSeeder` yang men-extends `Seeder` class.

Di dalam class `MahasiswaTableSeeder` ini juga terdapat method `run()` tempat kita menulis kode seeder.

Sebagai bahan praktek, pindahkan kode generate tabel `mahasiswas` dari `DatabaseSeeder.php` ke dalam file `MahasiswaTableSeeder.php` ini:

`database\seeders\MahasiswaTableSeeder.php`

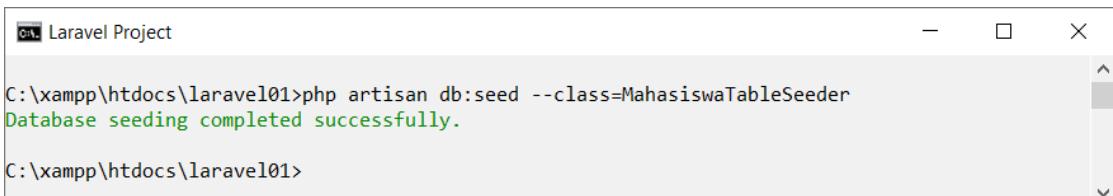
```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use App\Models\Mahasiswa;
7 use Faker\Factory as Faker;
8
9 class MahasiswaTableSeeder extends Seeder
```

Seeder

```
10  {
11      /**
12      * Run the database seeds.
13      *
14      * @return void
15      */
16     public function run()
17     {
18         $faker = Faker::create('id_ID');
19
20         for ($i=0; $i<10; $i++) {
21             Mahasiswa::create(
22                 [
23                     'nim' => $faker->numerify('10#####'),
24                     'nama' => $faker->firstName." ".$faker->lastName,
25                     'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01',
26                                         '+ 3 years'),
27                     'ipk' => $faker->randomFloat(2, 2, 4),
28                 ]
29             );
30         }
31     }
32 }
```

Agar tidak perlu menunggu terlalu lama, perulangan for di baris 20 saya ubah menjadi 10 kali saja. Save file di atas dan jalankan dengan perintah berikut:

```
php artisan db:seed --class=MahasiswaTableSeeder
```



Gambar: Menjalankan seed MahasiswaTableSeeder

Hasilnya, isi tabel mahasiswas akan bertambah 10 baris yang berasal dari file `MahasiswaTableSeeder.php`:

104	10677335	Fitria Novitasari	1999-10-16	3.01	2020-06-18 03:40:56	2020-06-18 03:40:56				
105	10347502	Daryani Saragih	2001-04-04	3.42	2020-06-18 03:40:56	2020-06-18 03:40:56				
106	10815345	Darimin Maryati	2001-10-11	3.81	2020-06-18 03:40:56	2020-06-18 03:40:56				
107	10549853	Cahyo Haryanti	2000-08-16	2.80	2020-06-18 03:40:56	2020-06-18 03:40:56				
108	10918653	Radit Fujiati	1999-10-03	3.71	2020-06-18 03:40:56	2020-06-18 03:40:56				
109	10437687	Gatra Namaga	1999-11-25	2.71	2020-06-18 03:40:56	2020-06-18 03:40:56				
110	10553124	Karman Yuniar	2001-07-31	3.64	2020-06-18 03:40:56	2020-06-18 03:40:56				

Gambar: Isi tabel mahasiswas jadi 110 baris

Exercise

Sebagai latihan, bisakah anda buat file seeder terpisah untuk men-generate 5 data baru untuk tabel `users`? Silahkan coba sebentar.

Answer

Pertama, buat file `UserTableSeeder` dengan perintah berikut:

```
php artisan make:seeder UserTableSeeder
```

Setelah itu tulis (atau copy) kode seeder seperti yang ada di file `DatabaseSeeder.php` sebelumnya:

```
database\seeders\UserTableSeeder.php
```

```

1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use App\Models\User;
7  use Illuminate\Support\Facades\Hash;
8  use Faker\Factory as Faker;
9
10 class UserTableSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds.
14      *
15      * @return void
16      */
17     public function run()
18     {
19         $faker = Faker::create('id_ID');
20
21         for ($i=0; $i<5; $i++) {
22             User::create(
23                 [
24                     'name' => $faker->firstName,
25                     'email' => $faker->unique()->email,
26                     'password' => Hash::make('qwerty'),
27                 ]
28             );
29         }
30     }
31 }
```

Jalankan file seeder dengan perintah:

```
php artisan db:seed --class=UserTableSeeder
```

Hasilnya, isi tabel users akan bertambah 5 dengan data baru.

7.5. Menjalankan Banyak File Seeder

Jika aplikasi yang kita buat terdiri dari 5 tabel atau lebih, maka idealnya akan ada 5 file seeder untuk setiap tabel. Jika dibuat terpisah seperti ini, akan cukup repot menjalankan 5 kali perintah seeder (satu untuk setiap tabel).

Sebagai solusi, kita bisa jadikan file `DatabaseSeeder.php` sebagai 'file master' yang akan memanggil kelima file seeder tersebut. Dan sebenarnya inilah fungsi utama dari file `DatabaseSeeder.php`, yakni sebagai tempat untuk memanggil file-file seeder lain.

Saat ini kita sudah memiliki 2 buah file seeder terpisah, yakni `MahasiswaTableSeeder.php` dan `UserTableSeeder.php` (dari exercise). Untuk menjalankan keduanya, silahkan modifikasi file `DatabaseSeeder.php` sebagai berikut:

`database\seeders\DatabaseSeeder.php`

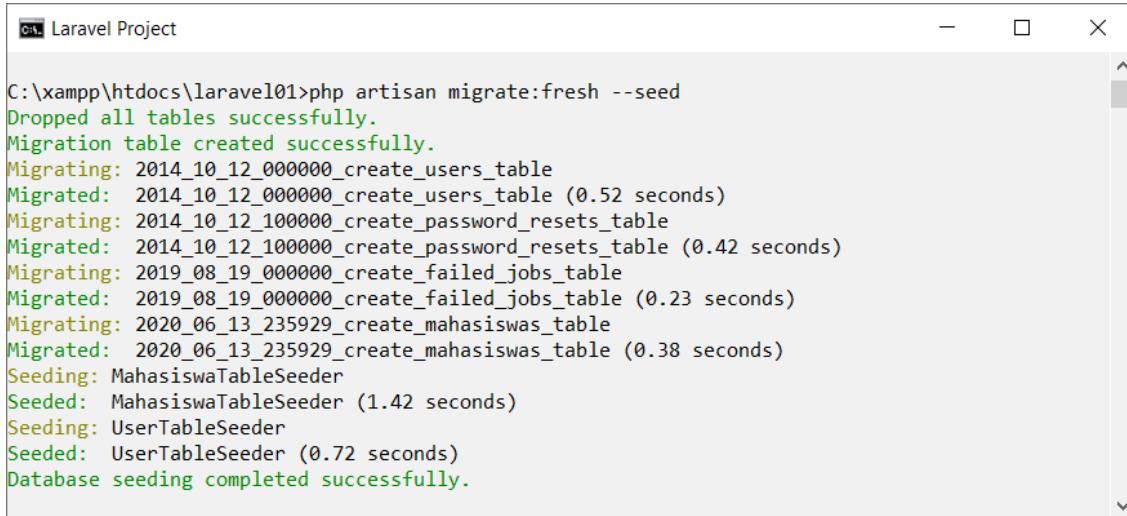
```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class DatabaseSeeder extends Seeder
8 {
9     public function run()
10    {
11        $this->call(MahasiswaTableSeeder::class);
12        $this->call(UserTableSeeder::class);
13    }
14 }
```

Isi file `DatabaseSeeder.php` sekarang jauh lebih sederhana, hanya perlu 2 buah baris di dalam method `run()`, yakni `$this->call(MahasiswaTableSeeder::class)` untuk memanggil seeder tabel `mahasiswa`, dan `$this->call(UserTableSeeder::class)` untuk memanggil file seeder tabel `users`.

Jika nantinya kita punya file seeder ketiga bernama `DosenTableSeeder`, maka tinggal input dengan perintah `$this->call(DosenTableSeeder::class)`.

Sekarang dengan mengetik perintah `php artisan db:seed` atau `php artisan migrate:fresh --seed` maka proses seeder untuk tabel `mahasiswa` dan `users` otomatis akan dijalankan.

Seeder



```
C:\xampp\htdocs\laravel01>php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.52 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.42 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.23 seconds)
Migrating: 2020_06_13_235929_create_mahasiswa_table
Migrated: 2020_06_13_235929_create_mahasiswa_table (0.38 seconds)
Seeding: MahasiswaTableSeeder
Seeded: MahasiswaTableSeeder (1.42 seconds)
Seeding: UserTableSeeder
Seeded: UserTableSeeder (0.72 seconds)
Database seeding completed successfully.
```

Gambar: Menjalankan seeder

Jadi, apakah lebih baik tulis file seeder terpisah seperti ini atau langsung di file `DatabaseSeeder.php` seperti contoh pada awal bab?

Cara yang lebih disarankan adalah buat file seeder terpisah untuk setiap tabel, lalu gabung di file `DatabaseSeeder.php`. Selain lebih rapi, ini akan memudahkan kolaborasi antara programmer (jika project dibuat oleh tim). Programmer lain akan langsung paham bahwa seeder untuk tabel **barang** berada di file `BarangTableSeeder.php`.

Namun untuk project sederhana dan dikerjakan sendiri, saya lebih sering buat langsung kode seeder di file `DatabaseSeeder.php` karena lebih praktis.

7.6. Seeder dari File CSV

Materi kali ini lebih ke tambahan, namun saya rasa cukup bermanfaat dan bisa membantu dalam situasi tertentu.

Ketika kita butuh data awal yang benar-benar asli (bukan data acak yang berasal dari Faker), maka terpaksa harus dibuat manual. Jika data tersebut cukup banyak, biasanya dibuat menggunakan aplikasi *spreadsheet* seperti Microsoft Excel, lalu di-import ke dalam database MySQL.

Proses import data ke MySQL akan lebih mudah jika data asal sudah dalam format `.csv`. Format CSV (singkatan dari *Comma Separated Values*) sebenarnya berbentuk file teks biasa dimana setiap data dipisah dengan karakter koma `,`. Hampir semua aplikasi *spreadsheet* memiliki menu export ke format CSV.

Proses import ke MySQL bisa dilakukan melalui aplikasi phpMyAdmin atau langsung mengetik perintah query di MySQL Client, yakni menggunakan query `LOAD DATA INFILE` seperti yang pernah di bahas pada buku **MySQL Uncover**.

Bagaimana jika proses import dari file CSV ini bisa di jalankan dari seeder? Tentu akan sangat memudahkan karena isi tabel bisa di generate ulang dengan 1 perintah sederhana.

Menyiapkan File CSV

Langkah pertama yang harus kita lakukan adalah menyiapkan file CSV. Ini bisa dibuat dari aplikasi Microsoft Excel atau aplikasi spreadsheet lain. Agar seragam, saya akan pakai aplikasi **Google Spreadsheet** yang bisa diakses gratis (selama terkoneksi ke internet). Silahkan buka web browser, login ke akun Google lalu buka <https://docs.google.com/spreadsheets>.

Sebagai bahan percobaan, saya ingin melakukan proses import ke dalam tabel `mahasiswa`. Maka kita butuh membuat tabel dengan kolom `id`, `nim`, `nama`, `tanggal_lahir` dan `ipk`.

	A	B	C	D	E	F	G	H
1	id	nim	nama	tanggal_lahir	ipk			
2		1 19003036	Sari Citra Lestari	2001-12-31	3.62			
3		2 19021044	Rudi Permana	2000-08-22	2.99			
4		3 19002032	Rina Kumala Sar	2000-06-28	3.82			
5		4 18012012	James Situmorang	1999-04-02	2.74			
6								
7								

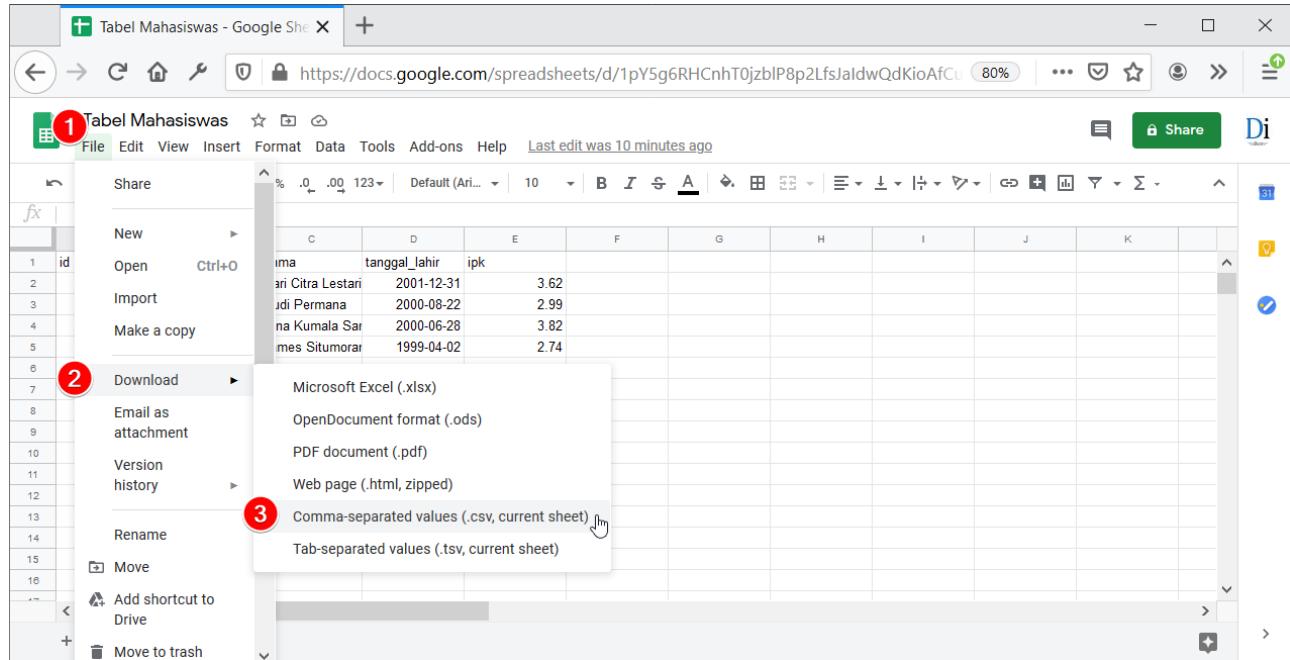
Gambar: Tampilan Google Spreadsheets

Agar tidak bermasalah dengan proses import nantinya, nama kolom di baris 1 harus sama dengan nama kolom di tabel MySQL.

Khusus untuk kolom `id`, bersifat opsional (boleh ditulis, boleh tidak). Data tanggal dan angka juga harus ditulis dalam format yang di dukung MySQL, yakni format YYYY-MM-DD untuk tanggal dan tanda titik sebagai penanda angka desimal, bukan tanda koma.

Setelah selesai, export file ini ke dalam format CSV dengan memilih menu **File** (1), pilih **Download** (2) dan klik **Comma-separated values (.csv, current sheet)** (3).

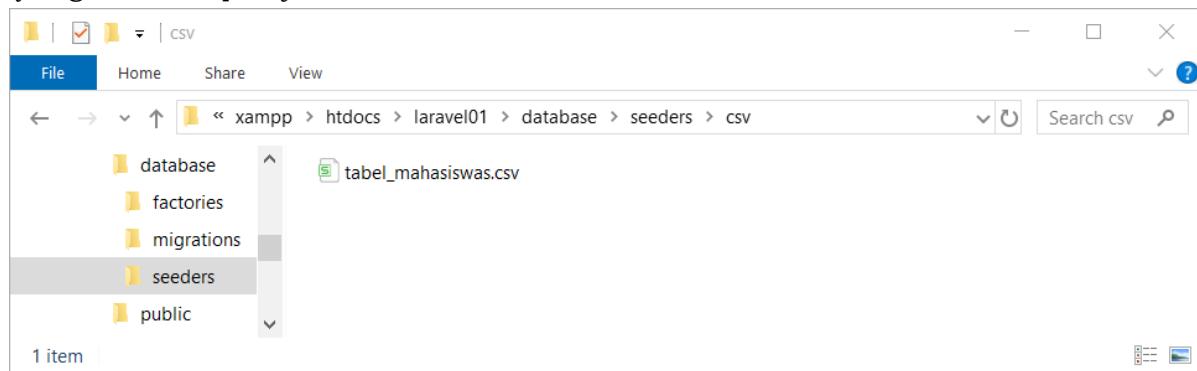
Seeder



Gambar: Proses konversi ke dalam format CSV

Selanjutnya akan tampil jendela download. Simpan file ini ke folder `database\seeders\csv\` di folder instalasi Laravel. Folder '`csv`' ini memang belum ada sebelumnya, maka silahkan buat terlebih dahulu.

Setelah disimpan, rename nama file menjadi `tabel_mahasiswa.csv`. Ini tidak harus dilakukan, hanya agar lebih rapi saja.



Gambar: Lokasi file `tabel_mahasiswa.csv`

Sebelum masuk ke kode program, periksa sekilas isi `tabel_mahasiswa.csv` dengan membukanya menggunakan aplikasi **Notepad++** atau aplikasi teks editor lain seperti **VS Code** atau **Notepad** bawaan Windows:

Seeder



C:\xampp\htdocs\laravel01\database\seeds\csv\tabel_mahasiswa.csv - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
tabel_mahasiswa.csv
1 id,nim,nama,tanggal_lahir,ipk
2 1,19003036,Sari Citra Lestari,2001-12-31,3.62
3 2,19021044,Rudi Permana,2000-08-22,2.99
4 3,19002032,Rina Kumala Sari,2000-06-28,3.82
5 4,18012012,James Situmorang,1999-04-02,2.74
Normal text file length : 207 lines : 5 Ln : 1 Col : 30 Sel : 0 | 0 Windows (CR LF) ANSI INS

Gambar: Isi file tabel_mahasiswa.csv

Terlihat setiap data dipisah dengan tanda koma. Inilah ciri khas dari format file CSV. Apabila diperlukan, bisa tambah baris baru selama mengikuti format yang ada.

Download Library CSV Seeder

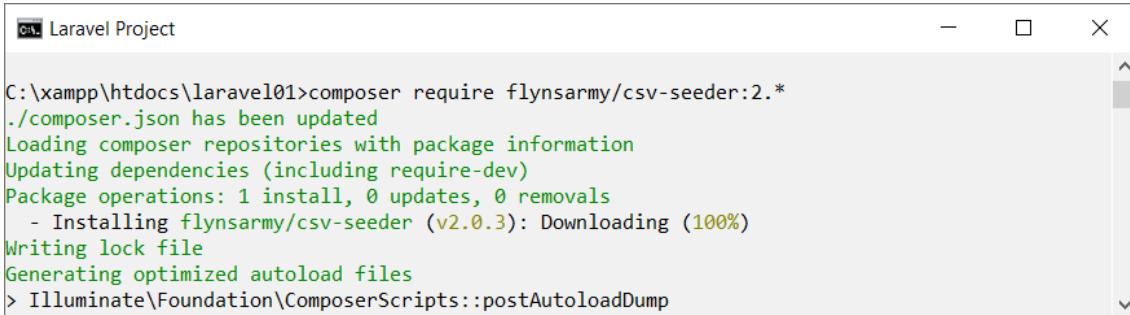
Masuk ke kode Laravel, kita perlu menginstall sebuah library tambahan. Cukup banyak library yang bisa dipakai, kali saya menggunakan **Laravel CSV Seeder** yang beralamat di <https://github.com/Flynsarmy/laravel-csv-seeder>.

Untuk menginstallnya, buka cmd dan masuk ke folder instalasi laravel. Jika anda menggunakan PHP 7.4 atau lebih, jalankan perintah:

```
composer require flynsarmy/csv-seeder:2.*
```

Atau jika menggunakan PHP di bawah versi 7.4, jalankan perintah:

```
composer require flynsarmy/csv-seeder:1.*
```



```
C:\xampp\htdocs\laravel01>composer require flynsarmy/csv-seeder:2.*  
.composer.json has been updated  
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
Package operations: 1 install, 0 updates, 0 removals  
- Installing flynsarmy/csv-seeder (v2.0.3): Downloading (100%)  
Writing lock file  
Generating optimized autoload files  
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
```

Gambar: Proses instalasi library flynsarmy/csv-seeder

Setelah berhasil di install, buka file `MahasiswaTableSeeder.php`, yakni file seeder untuk tabel `mahasiswa`, lalu modifikasi sebagai berikut:

```
database\seeders\MahasiswaTableSeeder.php
```

```
1 <?php  
2  
3 namespace Database\Seeders;
```

Seeder

```
4
5  use Illuminate\Database\Seeder;
6  use Flynsarmy\CsvSeeder\CsvSeeder;
7
8  class MahasiswaTableSeeder extends CsvSeeder
9  {
10    public function __construct()
11    {
12      $this->table = 'mahasiswas';
13      $this->filename = base_path().'/database/seeders/csv/tabel_mahasiswas.csv';
14      $this->should_trim = true;
15      $this->timestamps = true;
16      $this->created_at = now();
17      $this->updated_at = now();
18    }
19
20    public function run()
21    {
22      parent::run();
23    }
24 }
```

Perubahan pertama ada di baris 6, dimana kita butuh perintah import namespace `use Flynsarmy\CsvSeeder\CsvSeeder`. Ini diperlukan agar library CSV Seeder bisa diakses dari dalam file seeder. Selanjutnya di baris 8 class `MahasiswaTableSeeder` tidak lagi men-extends **Seeder** class, tapi **CsvSeeder** class.

Kemudian antara baris 10 – 18 terdapat sebuah constructor dengan beberapa pengaturan property. Berikut penjelasan dari property-property ini:

- `$this->table`: Nama tabel database yang akan diisi.
- `$this->filename`: Path ke file csv. Jika file CSV di simpan pada tempat lain, sesuaikan alamat pathnya di sini.
- `$this->should_trim`: Apakah data akan di-trim (menghapus tambahan spasi di awal dan akhir data). Nilai `true` berarti data akan di trim,
- `$this->timestamps`: Apakah akan ditambah data `timestamps` (untuk kolom `created_at` dan `updated_at`). Nilai `true` berarti kedua kolom tersebut akan diisi.
- `$this->created_at`: Nilai untuk kolom `created_at`. Karena diisi dengan nilai `now()` maka akan mengambil tanggal saat ini.
- `$this->updated_at`: Nilai untuk kolom `updated_at`. Karena diisi dengan nilai `now()` maka akan mengambil tanggal saat ini.

Selain beberapa property di atas, library CSV Seeder masih memiliki beberapa pengaturan lain. Daftar lengkapnya bisa dipelajari ke <https://github.com/Flynsarmy/laravel-csv-seeder>. Tapi untuk keperluan kita, pengaturan di atas sudah mencukupi.

Seeder

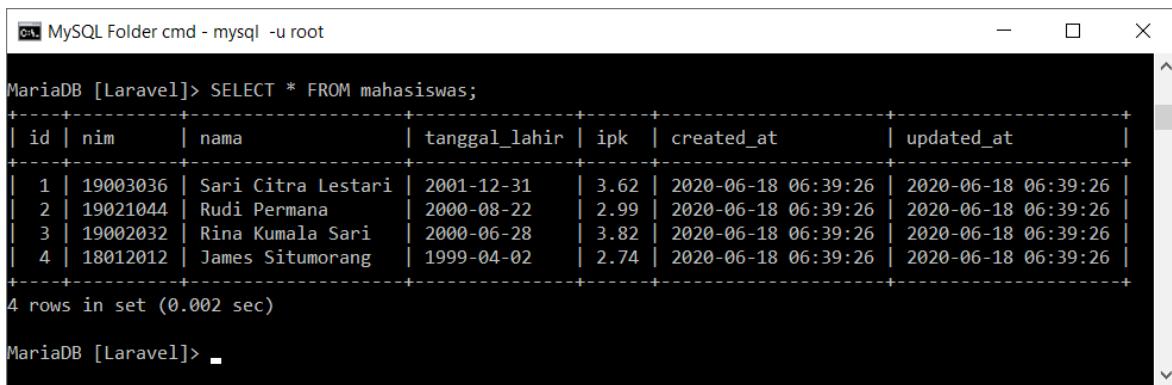
Terakhir, jalankan file seeder seperti biasa:

```
php artisan migrate:fresh --seed
```

Perintah di atas akan menjalankan ulang migration (tabel kembali kosong), lalu memproses file master seeder. Di dalam file master seeder sendiri sudah terdapat kode untuk menjalankan file `MahasiswaTableSeeder.php` hasil dari praktik kita sebelumnya. Atau jika ingin menjalankan langsung file `MahasiswaTableSeeder.php`, bisa menggunakan perintah:

```
php artisan db:seed --class=MahasiswaTableSeeder
```

Jika tidak terdapat error, maka seharusnya tabel mahasiswa sudah berisi data yang berasal dari file `tabel_mahasiswas.csv`.



The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM mahasiswas;". The output displays four rows of data from the table:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.62	2020-06-18 06:39:26	2020-06-18 06:39:26
2	19021044	Rudi Permana	2000-08-22	2.99	2020-06-18 06:39:26	2020-06-18 06:39:26
3	19002032	Rina Kumala Sari	2000-06-28	3.82	2020-06-18 06:39:26	2020-06-18 06:39:26
4	18012012	James Situmorang	1999-04-02	2.74	2020-06-18 06:39:26	2020-06-18 06:39:26

4 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswa yang berasal dari file tabel_mahasiswas.csv

Seeder merupakan salah satu fitur yang sangat praktis untuk men-generate data awal ke dalam database. Hampir dalam setiap project kita perlu data sample. Dengan gabungan fitur Seeder + Faker, proses generate data ini bisa dilakukan dengan mudah dan bisa di ulang kapan saja.

Satu lagi fitur Laravel yang berguna untuk men-generate data adalah **Factory**. Inilah yang menjadi materi bahasan kita selanjutnya.

8. Factory

Factory yang akan kita bahas kali ini masih berhubungan dengan proses generate data. Dalam penerapannya, factory juga bisa digabung dengan seeder dan faker. Kali ini kita akan bahas secara mendalam apa fungsi dari factory dan bagaimana cara penggunaannya.

Sebagai bahan praktek, boleh dilanjutkan dari bab sebelumnya (**Seeder**). Namun agar tidak terdapat kode yang bentrok, silahkan hapus semua isi file `route\web.php` lalu jalankan perintah `php artisan migrate:fresh` agar tabel `mahasiswa` kembali kosong.

8.1. Pengertian Factory

Factory adalah sebuah class di dalam Laravel yang dipakai untuk men-generate instance dari Model class. Instance ini akan berisi data yang seolah-olah berasal dari database.

Misalnya kita memiliki model `Mahasiswa`, maka factory bisa dipakai untuk men-generate 1 data mahasiswa. Data mahasiswa tersebut tidak diambil dari database, tapi langsung di generate dari kode program. Nantinya data hasil factory juga bisa diinput ke dalam database (berfungsi mirip seperti seeder).

Sebagai pengingat, Model pada dasarnya adalah sebuah class. Berikut isi dari file `app\Models\Mahasiswa.php` yang sudah sangat sering kita akses:

`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 }
```

Di baris 8 terdapat kode untuk membuat class **Mahasiswa** yang men-extends class **Model**. Mulai dari Laravel 8, setiap model sekarang memiliki trait atau method tambahan yang berasal

dari import class **HasFactory**. Dengan factory, nantinya kita bisa menjalankan kode seperti ini :

```
$foo = Mahasiswa::factory()->make();
```

Hasil akhirnya, \$foo akan berisi object (*instance*) dari class Mahasiswa. Variabel \$foo memiliki data lengkap mahasiswa seperti nim, nama, tanggal_lahir dan ipk, yang seolah-olah berasal dari database.

Sampai di sini mungkin anda masih bingung apa fungsi sebenarnya dari factory, kita akan bahas bertahap.

Factory merupakan salah satu fitur yang mendapat perubahan besar di Laravel 8. Sekarang factory bisa langsung dipanggil dari class Model, tidak lagi dari method `$factory` seperti di Laravel 7 ke bawah.

8.2. Membuat File Factory

Dalam penerapannya, sebuah factory akan berpasangan dengan satu model. Misalnya untuk model Mahasiswa akan berpasangan dengan "Mahasiswa Factory", begitu juga untuk model User yang akan berpasangan dengan "User Factory".

Pada saat membuat file factory, seharusnya sudah tersedia model yang akan kita akses. Atau bisa juga model tersebut dibuat bersamaan dengan file factory. Berikut format dasar perintah **php artisan** untuk membuat file factory:

```
php artisan make:factory <NamaFactory>
```

Nama factory boleh bebas tapi disarankan memakai format `<NamaModel>+Factory`. Misalnya jika ingin membuat factory untuk model Mahasiswa, maka nama file menjadi `MahasiswaFactory`. Atau factory untuk model User bisa ditulis sebagai `UserFactory`.

Mari masuk ke praktik dengan membuat file `MahasiswaFactory`. Silahkan buka cmd, masuk ke folder instalasi laravel lalu jalankan perintah berikut:

```
php artisan make:factory MahasiswaFactory
```

```
C:\xampp\htdocs\laravel01>php artisan make:factory MahasiswaFactory
Factory created successfully.

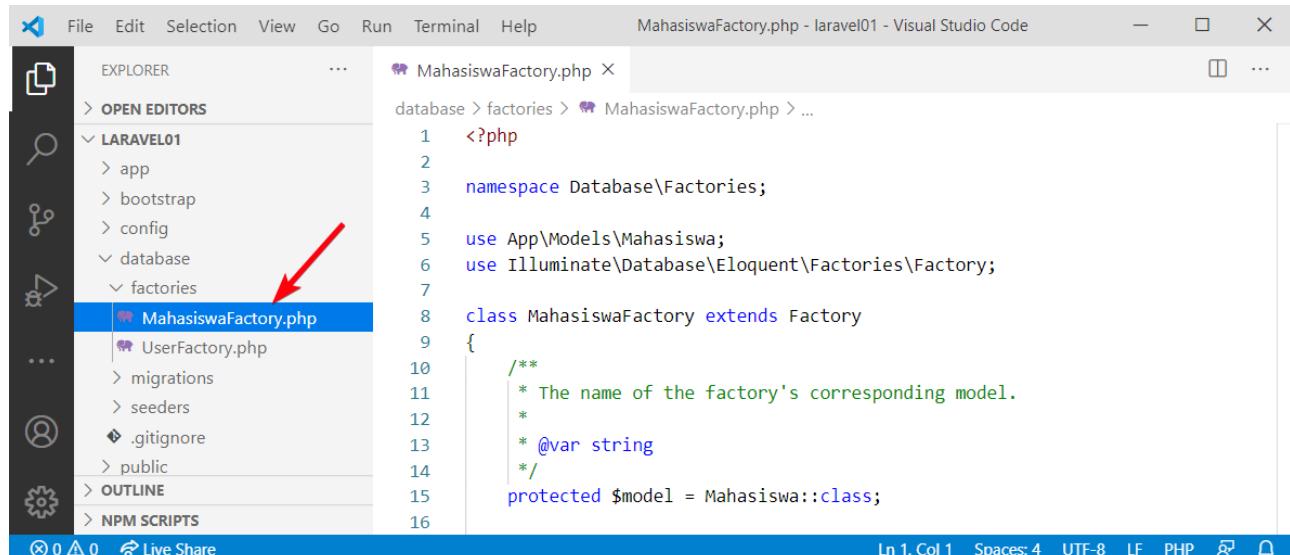
C:\xampp\htdocs\laravel01>
```

Gambar: Membuat file MahasiswaFactory

Jika tampil pesan `Factory created successfully.`, maka artinya file `MahasiswaFactory` sudah berhasil dibuat. Laravel menyimpan file factory di folder `database\factories\`. Silahkan buka

Factory

folder ini dan lihat isi file `MahasiswaFactory.php`:



Gambar: Lokasi file `MahasiswaFactory.php`

`database\factories\MahasiswaFactory.php`

```
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Mahasiswa;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class MahasiswaFactory extends Factory
9 {
10     protected $model = Mahasiswa::class;
11
12     public function definition()
13     {
14         return [
15             // ...
16         ];
17     }
18 }
```

Inilah kode awal dari file factory. Untuk menyederhanakan pembahasan, saya menghapus baris komentar yang ada.

Seperti class Laravel pada umumnya, di baris 3 terdapat pendefinisian `namespace Database\Factories`, kemudian diikuti dengan import class model **Mahasiswa** di baris 5 dan class **Factory** di baris 6.

Karena saat ini kita sudah memiliki model bernama `Mahasiswa`, Laravel langsung mengisi baris 5 dengan perintah `use App\Models\Mahasiswa`. Atau jika tidak ada pasangan model, baris ini akan berisi perintah `use App\Models\Model`.

Factory

Di baris 8, class **MahasiswaFactory** men-extends parent class **Factory**. Di dalam class inilah kita akan membuat kode untuk men-generate factory. Property `$model` di baris 10 sudah berisi pemanggilan `Mahasiswa::class`, yakni model class yang akan kita akses.

Mengenai data yang akan di generate, nantinya di tulis dalam array `return` mulai baris 15. Data tersebut bisa saja berbentuk data statis, yang artinya setiap kali factory di akses akan menghasilkan data yang sama. Atau lebih pas, buat data ini menggunakan Faker

Sekarang kita akan coba generate beberapa data mahasiswa menggunakan factory. Silahkan modifikasi file `MahasiswaFactory.php` sebagai berikut:

database\factories\MahasiswaFactory.php

```
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Mahasiswa;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class MahasiswaFactory extends Factory
9 {
10     protected $model = Mahasiswa::class;
11
12     public function definition()
13     {
14         return [
15             'nim' => $this->faker->unique()->numerify('10#####'),
16             'nama' => $this->faker->firstName()." ".$this->faker->lastName,
17             'tanggal_lahir' => $this->faker->dateTimeInInterval('1999-01-01',
18                                         '+ 3 years'),
19             'ipk' => $this->faker->randomFloat(2, 2, 4),
20         ];
21     }
22 }
```

Kode di baris 10 – 13 sama seperti kode untuk men-generate data tabel mahasiswas di bab seeder. Bisa juga disebut bahwa di sini kita memindahkan proses generate data mahasiswa dari seeder ke dalam factory. Perhatikan bahwa di dalam file Factory, kita bisa langsung mengakses faker dari `$this->faker`, tidak perlu lagi membuat proses inisialisasi Faker.

Save file `MahasiswaFactory.php`, dan saatnya kita coba jalankan.

8.3. Menjalankan Factory

Kembali ke pengertian, Factory dipakai untuk membuat *instance* dari class Model. Proses pembuatan *instance* ini bisa dilakukan berbagai tempat, terutama di controller dan juga di seeder. Agar lebih sederhana, praktek untuk controller akan saya pindahkan ke route saja (sama seperti bab-bab sebelumnya).

Factory

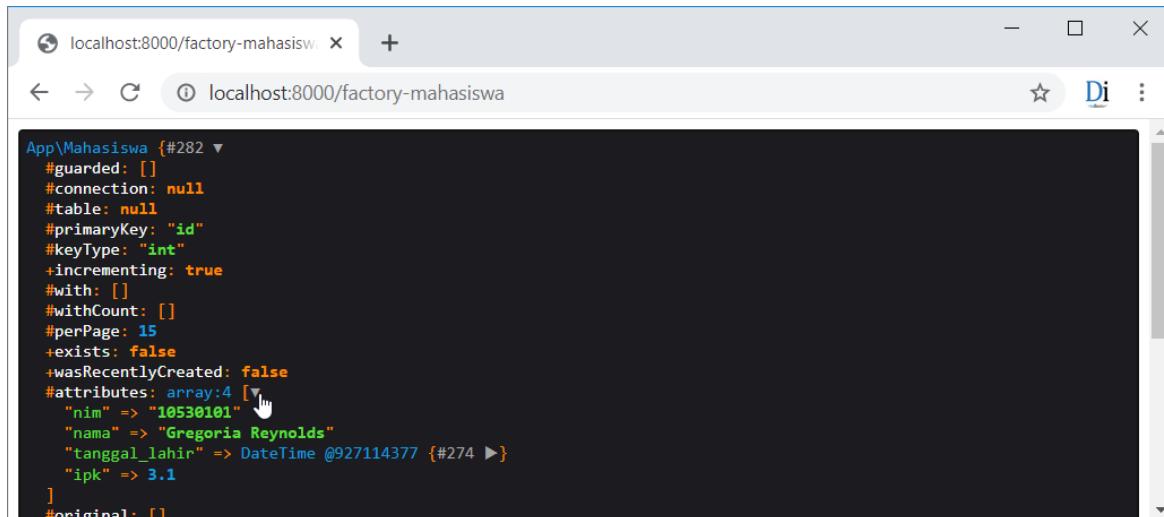
Berikut contoh pertama dari pengaksesan factory:

```
routes\web.php
```

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Models\Mahasiswa;
5
6 Route::get('/factory-mahasiswa', function () {
7     $mahasiswa = Mahasiswa::factory()->make();
8     dump ($mahasiswa);
9 });
10
11
```

Route '/factory-mahasiswa' sebenarnya hanya terdiri dari 1 kode utama, yakni pembuatan instance dari model Mahasiswa di baris 7.

Perintah yang digunakan adalah `Mahasiswa::factory()->make()`. Hasil dari perintah ini disimpan ke dalam variabel `$mahasiswa` untuk selanjutnya di `dump()` pada baris 8. Silahkan akses URL `localhost:8000/factory-mahasiswa` di web browser:



Gambar: Tampilan dump() factory mahasiswa

Terlihat variabel `$mahasiswa` sudah berisi object dari `App\Models\Mahasiswa`. Jika anda membuka tab `#attributes`, di sana terlihat data mahasiswa yang di generate oleh Faker.

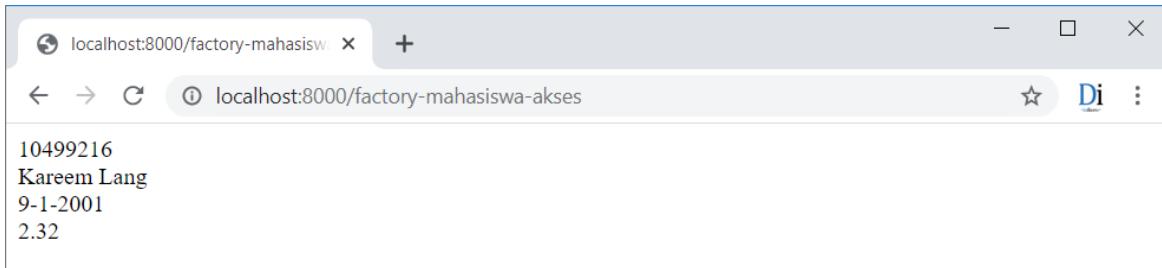
Variabel `$mahasiswa` ini bisa diproses sebagaimana layaknya data yang berasal dari database (hasil eloquent). Mari kita coba akses setiap kolom data mahasiswa:

```
routes\web.php
```

```
1 <?php
2 ...
3 Route::get('/factory-mahasiswa-akses', function () {
4
```

Factory

```
5 $mahasiswa = Mahasiswa::factory()->make();
6
7 echo $mahasiswa->nim. '<br>';
8 echo $mahasiswa->nama. '<br>';
9 echo \Carbon\Carbon::parse($mahasiswa->tanggal_lahir)->isoFormat('D-M-YYYY');
10 echo "<br>";
11 echo $mahasiswa->ipk. '<br>';
12
13});
```



Gambar: Mengakses data factory

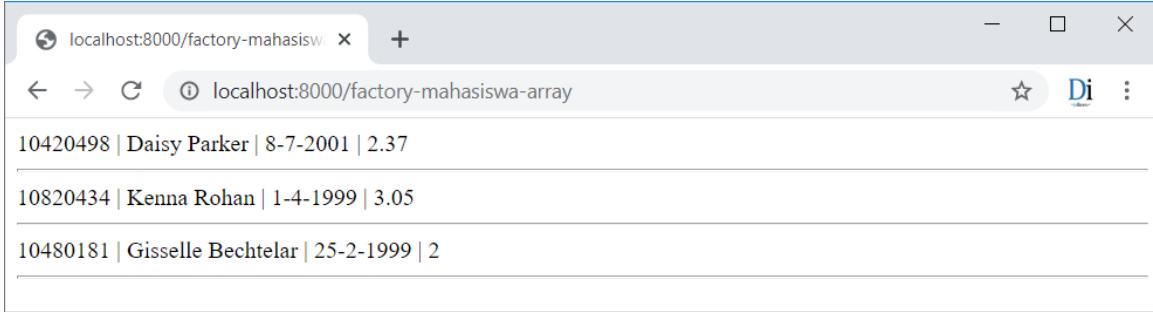
Cara yang dipakai sama persis seperti mengakses hasil eloquent, yakni `$mahasiswa->nim`, `$mahasiswa->nama`, dan juga `$mahasiswa->ipk`. Khusus untuk kolom `tanggal_lahir`, saya menggunakan Carbon agar lebih mudah di format.

`Mahasiswa::factory()` juga memiliki berbagai method tambahan, misalnya `count()` yang bisa dipakai untuk menentukan jumlah *instance object*. Misalnya jika dijalankan `Mahasiswa::factory()->count(3)->make()`, maka akan mengembalikan 3 buah instance object Mahasiswa dalam bentuk array. Berikut contoh penggunaannya:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/factory-mahasiswa-array', function () {
4     $mahasiswas = Mahasiswa::factory()->count(3)->make();
5
6     foreach ($mahasiswas as $mahasiswa) {
7         echo $mahasiswa->nim. ' | ';
8         echo $mahasiswa->nama. ' | ';
9         $carbon_tgl_lahir = \Carbon\Carbon::parse($mahasiswa->tanggal_lahir);
10        echo $carbon_tgl_lahir->isoFormat('D-M-YYYY'). " | ";
11        echo $mahasiswa->ipk. '<br>';
12    }
13});
```

Factory



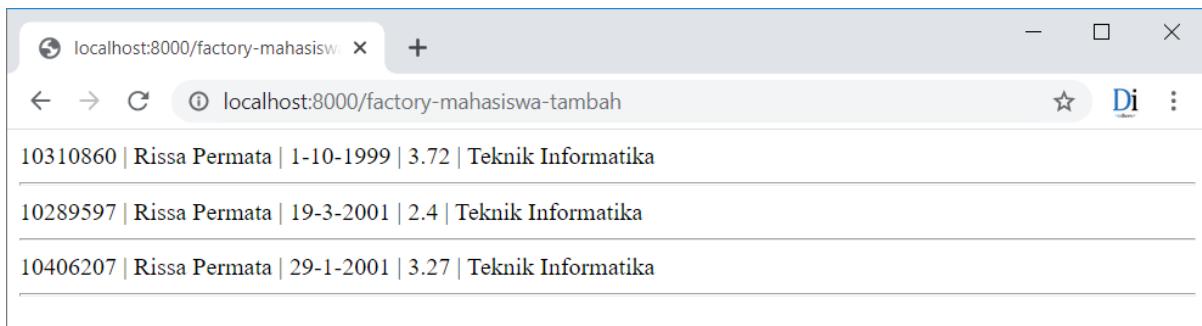
Gambar: Hasil generate 3 instance mahasiswa

Perintah `Mahasiswa:: factory()->count(3)->make()` mengembalikan 3 buah *instance* dalam bentuk array. Oleh karena itu perlu perulangan `foreach` antara baris 7 – 13 untuk membuka isi array. Kembali, ini sama persis seperti yang biasa kita lakukan di eloquent. Khusus untuk kolom `tanggal_lahir` terpaksa saya pecah ke dalam 2 baris karena keterbatasan lebar buku.

Apabila diperlukan, kita bisa menimpa atau menambah nilai baru yang di generate oleh `factory`. Caranya, tulis array ke dalam method `make()` seperti contoh berikut:

`routes\web.php`

```
1 <?php
2 ...
3 Route::get('/factory-mahasiswa-tambah', function () {
4     $mahasiswas = Mahasiswa::factory()->count(3)->make([
5         'nama' => 'Rissa Permata',
6         'jurusan' => 'Teknik Informatika',
7     ]);
8     foreach ($mahasiswas as $mahasiswa) {
9         echo $mahasiswa->nim. ' | ';
10        echo $mahasiswa->nama. ' | ';
11        $carbon_tgl_lahir = \Carbon\Carbon::parse($mahasiswa->tanggal_lahir);
12        echo $carbon_tgl_lahir->isoFormat('D-M-YYYY'). " | ";
13        echo $mahasiswa->ipk. " | ";
14        echo $mahasiswa->jurusan. '<hr>';
15    }
16});
```



Gambar: Mengubah dan menambah hasil factory

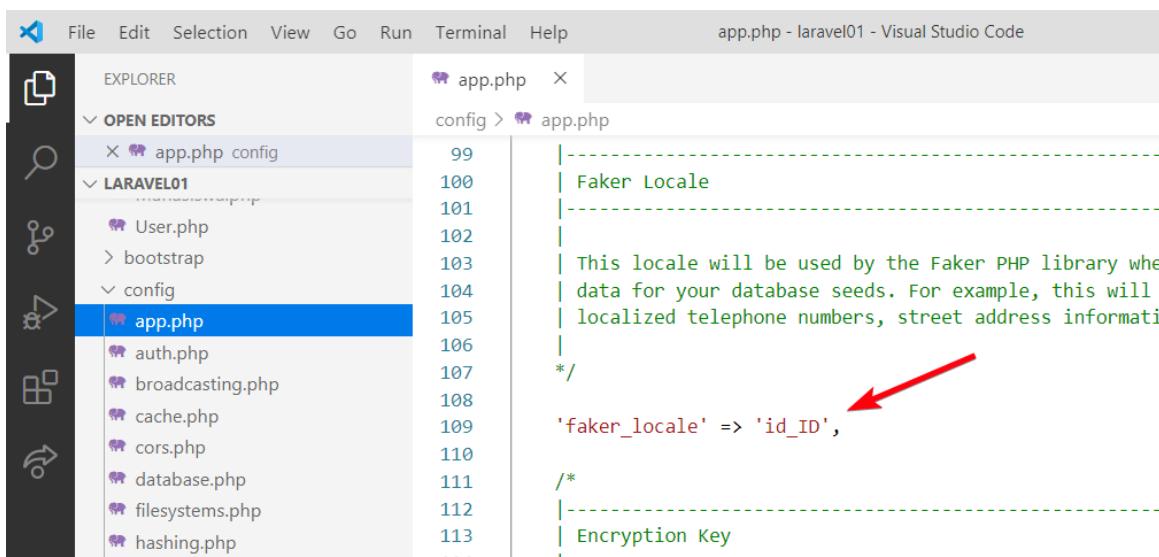
Pada saat menulis perintah `factory`, saya menambah array 2 element ke dalam method `make()` di baris 5 – 6. Nilai nama yang diinput akan menimpa kolom `nama` yang di generate oleh

factory. Sedangkan nilai jurusan akan menjadi nilai baru karena di factory mahasiswa kita tidak men-set kolom ini.

8.4. Factory (Faker) Localization

Dari beberapa percobaan yang sudah kita lakukan, nilai yang di generate oleh factory masih dalam bahasa inggris. Ini karena secara default localization Faker di set sebagai 'en_US'.

Untuk mengubahnya ke dalam Bahasa Indonesia, silahkan buka file config\app.php dan cari baris '`'faker_locale' => 'en_US'`', kemudian ganti menjadi '`'faker_locale' => 'id_ID'`'.



```

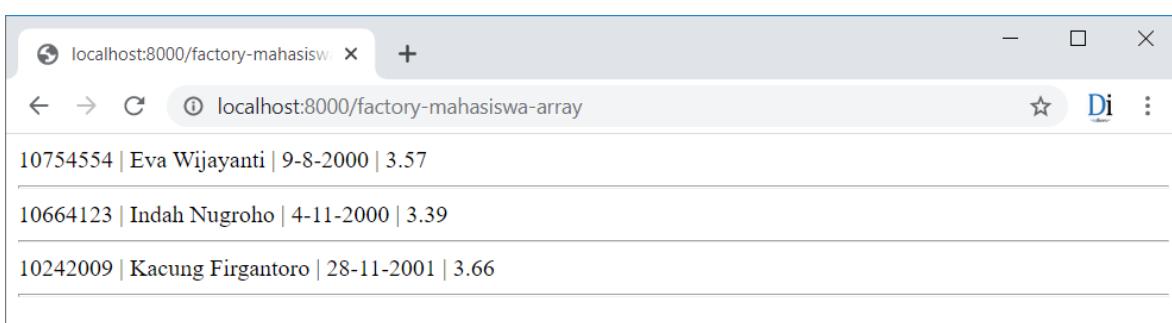
File Edit Selection View Go Run Terminal Help
app.php - laravel01 - Visual Studio Code

EXPLORER app.php
OPEN EDITORS config > app.php
LARAVEL01
User.php
> bootstrap
config
app.php
auth.php
broadcasting.php
cache.php
cors.php
database.php
filesystems.php
hashing.php
99 | -
100 | FAKER Locale
101 |
102 |
103 | This locale will be used by the Faker PHP library whe
104 | data for your database seeds. For example, this will
105 | localized telephone numbers, street address informati
106 |
107 */
108
109 'faker_locale' => 'id_ID', -----^
110 /*
111 |
112 |
113 |
114

```

Gambar: Mengganti pengaturan localization Faker

Save file config\app.php lalu test akses kembali localhost:8000/factory-mahasiswa-array, sekarang data yang di generate factory sudah dalam versi bahasa Indonesia.



Gambar: Hasil factory Mahasiswa sudah dalam bahasa Indonesia

Pengaturan '`'faker_locale'` ini bersifat global, yang artinya ketika kita mengakses perintah Faker di factory lain, secara otomatis juga memakai localization '`'id_ID'`'.

Jika kita ingin membuat locale untuk factory tertentu saja, salah satu caranya dengan mengakses langsung Faker class, tidak menggunakan `$this->faker` bawaan Factory. Sebagai

Factory

contoh untuk men-generate data faker dalam bahasa spanyol (es_ES), bisa menggunakan kode berikut:

database\factories\MahasiswaFactory.php

```
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Mahasiswa;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7 use Faker\Factory as Faker;
8
9 class MahasiswaFactory extends Factory
10 {
11     protected $model = Mahasiswa::class;
12
13     public function definition()
14     {
15         $faker = Faker::create('es_ES');
16         return [
17             'nim' => $faker->unique()->numerify('10#####'),
18             'nama' => $faker->firstName() . $faker->lastName,
19             'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01',
20                                         '+ 3 years'),
21             'ipk' => $faker->randomFloat(2, 2, 4),
22         ];
23     }
24 }
```

Di baris 7 saya mengimpor class **Faker** yang akan dipakai untuk proses instansiasi Faker object. Sebagai argument di baris 15, menggunakan string locale bahasa Spanyol, yakni 'es_ES'. Kemudian dalam array **return**, proses generate data dilakukan dari **\$faker** object, bukan lagi **\$this->faker**.



Gambar: Hasil factory dalam bahasa Spanyol

Dengan cara ini, setiap file factory bisa menggunakan versi locale yang berbeda-beda.

8.5. Input Factory ke Database

Sampai di sini mudah-mudahan anda sudah bisa memahami fungsi utama dari Factory, yakni

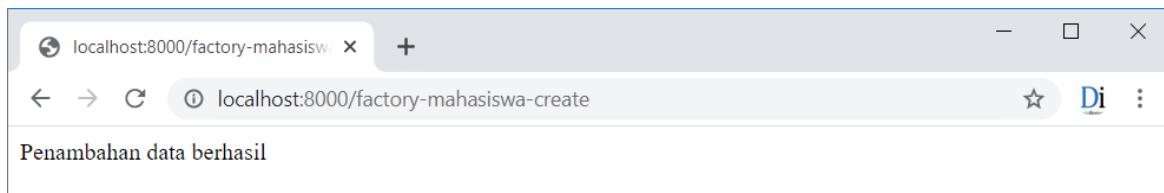
Factory

men-generate data Model. Sebelumnya, data ini langsung kita simpan ke dalam variabel dan menjadi *instance* dari Model tersebut.

Selain mengembalikan *instance* Model, hasil data dari Factory juga bisa di input langsung ke dalam tabel database. Ini menjadikan fungsi factory mirip seperti seeder. Caranya, ganti pemanggilan method `Mahasiswa::factory()->make()` menjadi `Mahasiswa::factory()->create()` seperti contoh berikut:

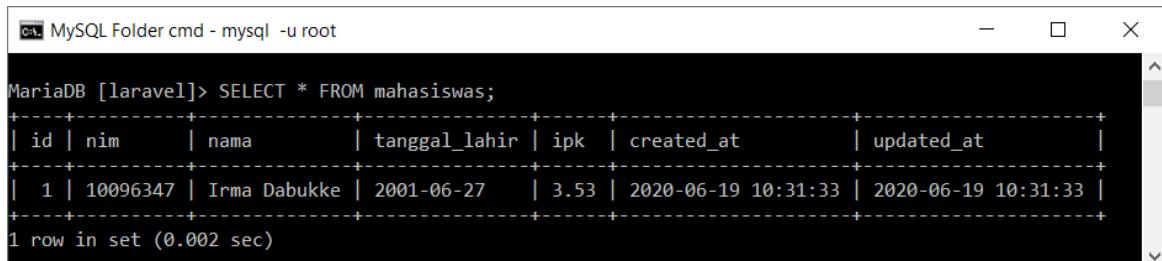
routes\web.php

```
1 <?php
2 ...
3 Route::get('/factory-mahasiswa-create', function () {
4     Mahasiswa::factory()->create();
5     return "Penambahan data berhasil";
6});
```



Gambar: Menginput hasil factory ke database

Sekarang pada saat alamat `localhost:8000/factory-mahasiswa-create` diakses, perintah `Mahasiswa::factory()->create()` akan di jalankan. Hasilnya, terdapat penambahan 1 data baru ke tabel `mahasiswas`:



Gambar: Isi tabel mahasiswas yang berasal dari factory

Apabila halaman di refresh beberapa kali, data akan terus ditambahkan ke dalam tabel `mahasiswas`.

Jika kita ingin menginput banyak data sekaligus, tambah argument kedua ke dalam function `factory()`:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/factory-mahasiswa-create-10', function () {
4     Mahasiswa::factory()->count(10)->create();
```

Factory

```
5         return "Penambahan data berhasil";
6     });
7
```

Di baris 4 saya menginput angka 10 sebagai argument kedua function factory(). Ketika di akses, kode di atas akan menginput 10 data mahasiswa:

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 11 rows, each representing a student record with columns: id, nim, nama, tanggal_lahir, ipk, created_at, and updated_at. The data includes names like Irma Dabukke, Putri Nasyiah, Jessica Pertiwi, etc., with various birth dates and GPAs.

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	10096347	Irma Dabukke	2001-06-27	3.53	2020-06-19 10:31:33	2020-06-19 10:31:33
2	10321260	Putri Nasyiah	1999-06-29	2.59	2020-06-19 10:40:53	2020-06-19 10:40:53
3	10911909	Jessica Pertiwi	2000-01-16	2.65	2020-06-19 10:40:53	2020-06-19 10:40:53
4	10685493	Catur Halim	1999-07-06	2.73	2020-06-19 10:40:53	2020-06-19 10:40:53
5	10326528	Paris Astuti	2000-07-31	2.58	2020-06-19 10:40:53	2020-06-19 10:40:53
6	10189762	Indra Purnawati	2000-07-31	2.33	2020-06-19 10:40:53	2020-06-19 10:40:53
7	10137904	Dwi Siregar	1999-05-20	3.12	2020-06-19 10:40:53	2020-06-19 10:40:53
8	10034171	Faizah Irawan	2001-09-16	3.85	2020-06-19 10:40:54	2020-06-19 10:40:54
9	10405271	Maya Zulaika	2000-02-24	2.18	2020-06-19 10:40:54	2020-06-19 10:40:54
10	10052629	Maria Susanti	1999-06-04	3.13	2020-06-19 10:40:54	2020-06-19 10:40:54
11	10634754	Novi Natsir	2000-06-14	3.82	2020-06-19 10:40:54	2020-06-19 10:40:54

11 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswa yang berasal dari factory

Dengan teknik ini, kita bisa menginput data baru ke dalam database. Ini sebenarnya mirip seperti seeder, tapi kali ini diakses langsung dari kode program.

8.6. File UserFactory.php

Anda mungkin sudah memperhatikan hal ini sebelumnya, bahwa di dalam folder `database\factories\` sudah terdapat 1 file factory bawaan Laravel, yakni `UserFactory.php`. Yup, dilihat dari namanya ini adalah file factory untuk `User` model.

The screenshot shows a Visual Studio Code interface with the file `UserFactory.php` open. The code defines a factory for the `User` model, using the `Faker` library to generate fake data. A red arrow points to the file in the Explorer sidebar, which lists the `LARAVEL01` project structure, including `app`, `bootstrap`, `config`, `database` (with `factories` subfolder containing `MahasiswaFactory.php` and `UserFactory.php`), `migrations`, `seeders`, `.gitignore`, and `NPM SCRIPTS`.

```
21 * @return array
22 */
23 public function definition()
24 {
25     return [
26         'name' => $this->faker->name,
27         'email' => $this->faker->unique()->safeEmail,
28         'email_verified_at' => now(),
29         'password' => '$2y$10$92IXUNpkj00rQ5byMi.Ye4oKoEa3RqallC/.og/at2.uh
30         'remember_token' => Str::random(10),
31     ];
32 }
33 }
34 }
```

Gambar: File UserFactory.php

Dengan menghapus baris komentar, berikut isi dari file `UserFactory.php`:

Factory

```
database\factories\UserFactory.php

1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\User;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7 use Illuminate\Support\Str;
8
9 class UserFactory extends Factory
10 {
11     protected $model = User::class;
12
13     public function definition()
14     {
15         return [
16             'name' => $this->faker->name,
17             'email' => $this->faker->unique()->safeEmail,
18             'email_verified_at' => now(),
19             'password' => '$2y$10$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/
20                                     .og/at2.uheWG/igi', // password
21             'remember_token' => Str::random(10),
22         ];
23     }
24 }
```

Isi file ini sangat mirip seperti MahasiswaFactory, hanya saja sekarang menggunakan model User. File UserFactory berisi 5 data yang di generate pada baris 9 - 14. Bermodalkan materi tentang Faker, kita sudah bisa menebak nilai untuk kolom name, email dan email_verified_at.

Untuk kolom password, Laravel langsung mengisinya dengan sebuah string acak. Ini adalah hasil hash dari kata 'secret'. Jadi jika nantinya jika kita ingin login sebagai user yang berasal dari Factory, passwordnya adalah 'secret'.

Terakhir, kolom remember_token diisi dengan 10 karakter string acak dari perintah facade Str::random(10).

Pada saat melakukan riset untuk materi ini, terdapat pertanyaan menarik di [stackover flow](#), yakni mengapa Laravel langsung menulis karakter hasil hashing sebagai nilai kolom password. Kenapa tidak di generate saja dengan perintah Hash::make('secret')?

Alasannya semata-mata untuk efisiensi. Proses hashing butuh sedikit waktu, dan karena UserFactory ditujukan untuk membuat data sample, akan lebih efisien jika password langsung dituliskan dalam bentuk hasil hashing saja. Dengan demikian PHP tidak perlu melakukan pemrosesan mencari hasil hashing.

Mari kita test jalankan UserFactory dari route. Silahkan tambah kode berikut:

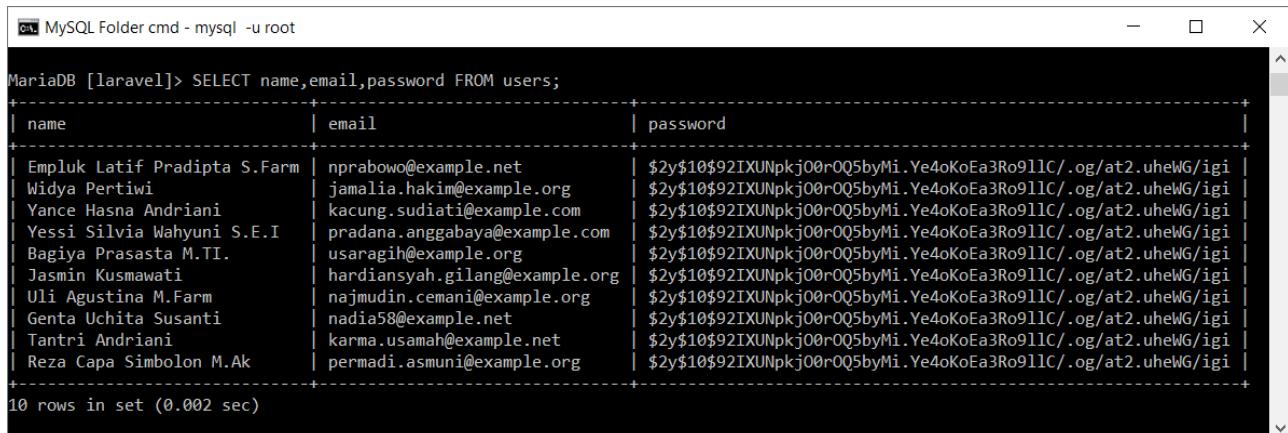
Factory

routes\web.php

```
1 <?php
2 ...
3 Route::get('/factory-mahasiswa-user', function () {
4     Mahasiswa::factory()->count(10)->create();
5     \App\Models\User::factory()->count(10)->create();
6     return "Penambahan data berhasil";
7 });
```

Agar tidak perlu menulis perintah import di bagian atas route, saya langsung mengakses namespace User model sebagai App\Models\User di baris 5.

Ketika URL localhost:8000/factory-mahasiswa-user diakses, akan menambah 10 data mahasiswa dan juga 10 data user ke dalam database:



name	email	password
Empluk Latif Pradipta S.Farm	nprabowo@example.net	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Widya Pertiwi	jamalia.hakim@example.org	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Yance Hasna Andriani	kacung.sudiatip@example.com	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Yessi Silvia Wahyuni S.E.I	pradana.anggabaya@example.com	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Bagiya Prasasta M.TI.	usaragih@example.org	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Jasmin Kusmawati	hardiansyah.gilang@example.org	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Uli Agustina M.Farm	najmudin.cemani@example.org	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Genta Uchita Susanti	nadia58@example.net	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Tantri Andriani	karma.usamah@example.net	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi
Reza Capa Simbolon M.Ak	permadi.asmuni@example.org	\$2y\$10\$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi

Gambar: Isi tabel users yang di generate dari UserFactory

8.7. Mengakses Factory dari Seeder

File factory sudah berisi data lengkap untuk men-generate Model. Pada saat membuat seeder, kita juga bisa memanggil factory ini agar tidak perlu membuat ulang kode yang sama. Silahkan buka file master seeder database\seeders\DatabaseSeeder.php, lalu modifikasi sebagai berikut:

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class DatabaseSeeder extends Seeder
8 {
9     public function run()
10    {
11        \App\Models\Mahasiswa::factory()->count(10)->create();
```

Factory

```
12         \App\Models\User::factory()->count(10)->create();  
13     }  
14 }
```

Artinya ketika perintah seeder di akses, baris 11 dan 12 akan men-generate data factory Mahasiswa dan User, kemudian langsung menginputnya ke database. Silahkan test dengan menjalankan perintah `php artisan db:seed`, lalu cek ke database.

```
Laravel Project  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\xampp\htdocs\laravel01>php artisan db:seed  
Database seeding completed successfully.  
C:\xampp\htdocs\laravel01>
```

Gambar: Menjalankan seeder yang berisi factory

Dengan mengakses factory, kita tidak perlu membuat perulangan `for` saat ingin menginput banyak data ke database, tapi cukup mengubah nilai argument method `count()`.

Apabila diperlukan, pemanggilan seeder bisa juga di tulis pada file seeder terpisah, sehingga urutan pembuatan seeder menjadi:

1. Buat file Factory `MahasiswaFactory.php`
2. Buat file Seeder `MahasiswaTableSeeder.php`, lalu akses factory dari dalam file ini, yakni dari perintah `\App\Models\Mahasiswa::factory()->count(10)->create()` ke dalam method `run()`.
3. Input perintah `$this->call(MahasiswaTableSeeder::class)` ke dalam file master seeder `DatabaseSeeder.php`.
4. Jalankan seeder dari cmd dengan perintah `php artisan db:seed`.

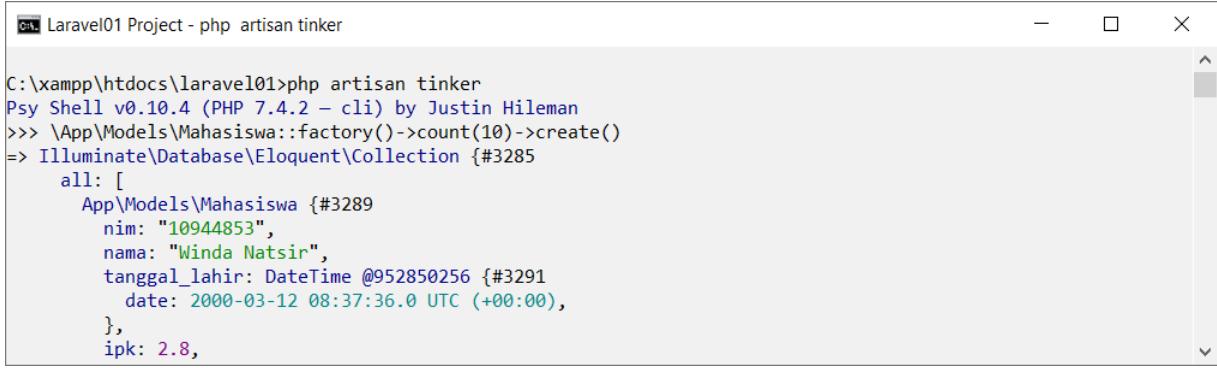
Empat langkah ini adalah proses pembuatan seeder yang paling ideal. Namun untuk project sederhana, saya kadang melompati langkah ke-2, dimana pemanggilan factory langsung di lakukan dari file `DatabaseSeeder.php` saja.

8.8. Mengakses Factory dari Tinker

Teknik ini sangat praktis dan sering dilakukan. Jika kita sudah memiliki file factory, proses generate data bisa dijalankan langsung dari Tinker.

Caranya, buka Tinker lalu langsung saja ketik perintah `\App\Models\Mahasiswa::factory()->count(10)->create()` untuk men-generate 10 data mahasiswa:

Factory



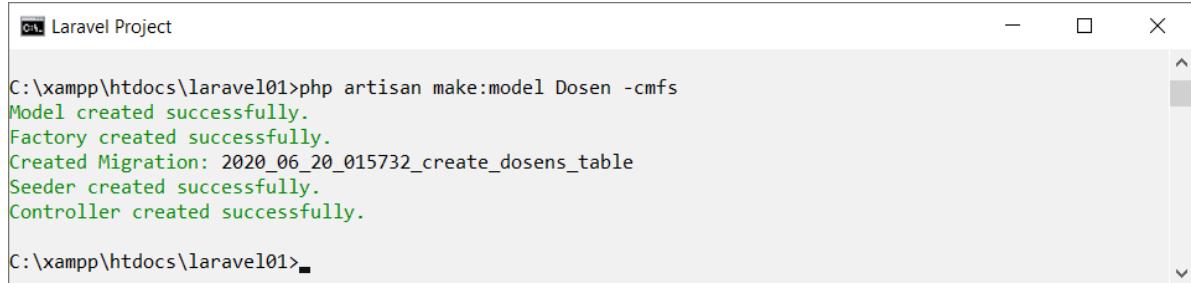
```
C:\xampp\htdocs\laravel01>php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.2 - cli) by Justin Hileman
>>> \App\Models\Mahasiswa::factory()->count(10)->create()
=> Illuminate\Database\Eloquent\Collection {#3285
    all: [
        App\Models\Mahasiswa {#3289
            nim: "10944853",
            nama: "Winda Natsir",
            tanggal_lahir: DateTime @952850256 {#3291
                date: 2000-03-12 08:37:36.0 UTC (+00:00),
            },
            ipk: 2.8,
        },
    ],
}
```

Gambar: Mengakses factory dari Tinker

8.9. Shortcut untuk Generate File Factory

Umumnya file **factory** di buat bersamaan dengan file **model**, **migration**, **seeder** dan juga **controller**. Karena itulah Laravel menyediakan shortcut khusus untuk membuat kelima file ini sekaligus. Caranya, tambah flag `-cmfs` ke dalam perintah `php artisan make:model` seperti contoh berikut:

```
php artisan make:model Dosen -cmfs
```



```
C:\xampp\htdocs\laravel01>php artisan make:model Dosen -cmfs
Model created successfully.
Factory created successfully.
Created Migration: 2020_06_20_015732_create_dosens_table
Seeder created successfully.
Controller created successfully.

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

Gambar: Hasil perintah `php artisan make:model Dosen -cmfs`

Perintah di atas akan membuat 5 file:

1. File model: `app\Models\Dosen.php`
2. File controller: `app\Http\Controllers\DosenController.php`
3. File migration: `database\migrations\<timestamp>create_dosens_table.php`
4. File factory: `database\factories\DosenFactory.php`
5. File seeder: `database\seeders\DosenSeeder.php`

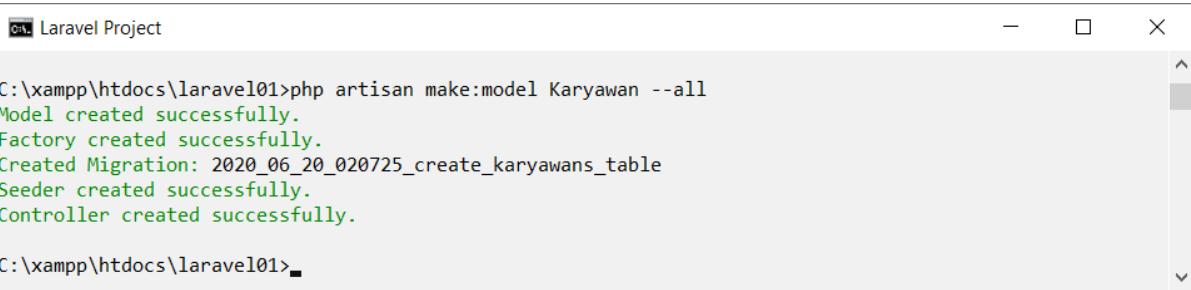
Flag `-cmfs` merupakan singkatan dari komponen yang ingin dibuat, yakni **controller**, **migration**, **factory** dan **seeder**. Jika kita tidak butuh seeder, maka bisa menjalankan:

```
php artisan make:model Dosen -cmf
```

Lebih jauh lagi, Laravel menyediakan flag `--all` untuk membuat file **model**, **resource controller**, **migration**, **factory** dan **seeder**:

Factory

```
php artisan make:model Karyawan --all
```



```
C:\xampp\htdocs\laravel01>php artisan make:model Karyawan --all
Model created successfully.
Factory created successfully.
Created Migration: 2020_06_20_020725_create_karyawans_table
Seeder created successfully.
Controller created successfully.

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

Gambar: Hasil perintah php artisan make:model Karyawan --all

Perbedaan dari sebelumnya, kali ini controller sudah langsung berbentuk *resource controller* yang siap di pakai untuk membuat aplikasi CRUD.

Shortcut perintah php artisan ini akan mempercepat pembuatan aplikasi menggunakan Laravel. Daftar semua shortcut ini juga bisa dilihat dengan mengetik `php artisan help make:model`.

Sepanjang bab ini kita telah melihat cara penggunaan Factory, yakni untuk membuat instance dari model serta juga bisa dipakai untuk menginput data sample ke dalam database.

Biasanya materi Faker, Seeder dan Factory sering dibahas bersamaan karena saling berhubungan satu sama lain. Namun dengan memecahnya menjadi bab tersendiri, kita bisa mempelajari setiap komponen dengan lebih detail.

Berikutnya, kita akan rehat sejenak dari pembahasan materi Laravel dan masuk ke mini project.

9. Case Study: Generating Data

Studi kasus kali ini sebenarnya belum bisa disebut sebagai sebuah "project asli". Tapi lebih ke latihan implementasi materi *generating data* yang sudah kita pelajari sejak awal buku, terutama tentang **faker + seeder + factory**. Ketiga fitur ini cukup sering dipakai pada saat membuat project baru di Laravel.

Selain itu apa yang kita praktekkan juga sebagai persiapan untuk bab berikutnya.

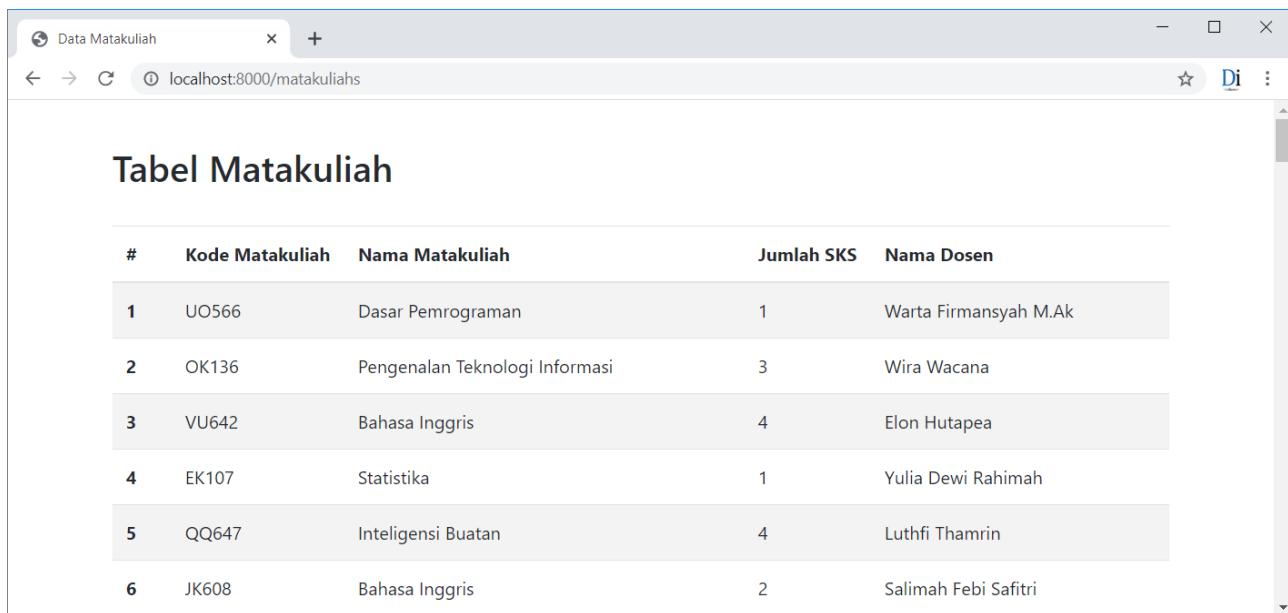
Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

9.1. Data Mata Kuliah

Studi kasus yang akan kita buat adalah menampilkan data mata kuliah, yakni mulai dari proses pembuatan tabel, generate *data sample*, hingga menampilkannya di web browser. Berikut tampilan akhir dari project ini:



The screenshot shows a web browser window titled "Data Matakuliah". The address bar indicates the URL is "localhost:8000/matakuliahs". The main content area displays a table titled "Tabel Matakuliah". The table has six columns: #, Kode Matakuliah, Nama Matakuliah, Jumlah SKS, and Nama Dosen. There are six rows of data, each with a different course code, name, credit hours, and professor name. The table is styled with alternating row colors.

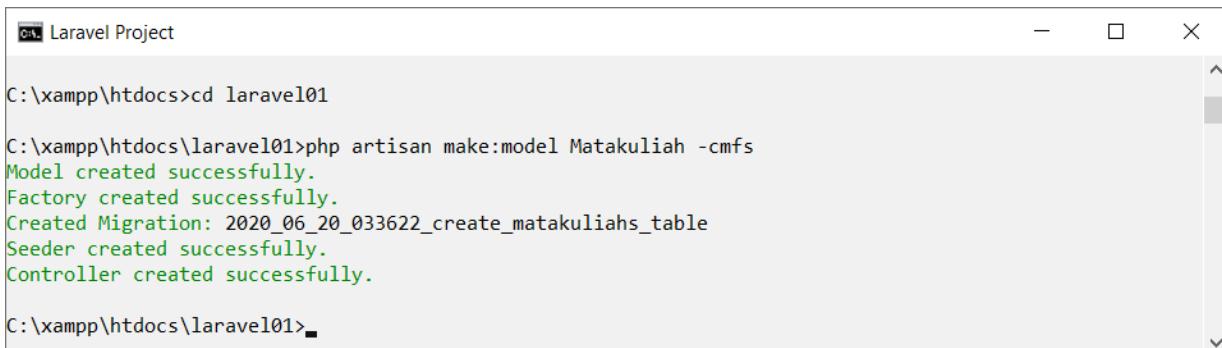
#	Kode Matakuliah	Nama Matakuliah	Jumlah SKS	Nama Dosen
1	UO566	Dasar Pemrograman	1	Warta Firmansyah M.Ak
2	OK136	Pengenalan Teknologi Informasi	3	Wira Wacana
3	VU642	Bahasa Inggris	4	Elon Hutapea
4	EK107	Statistika	1	Yulia Dewi Rahimah
5	QQ647	Inteligensi Buatan	4	Luthfi Thamrin
6	JK608	Bahasa Inggris	2	Salimah Febi Safitri

Gambar: Tampilan tabel Matakuliah

9.2. Generating File

Langkah pertama adalah men-generate file-file yang diperlukan, yakni model, controller, migration, factory dan seeder. Saya memilih nama model **Matakuliah** untuk proyek ini. Berikut perintah yang bisa dipakai untuk membuat semua file:

```
php artisan make:model Matakuliah -cmfs
```



```
C:\xampp\htdocs>cd laravel01
C:\xampp\htdocs\laravel01>php artisan make:model Matakuliah -cmfs
Model created successfully.
Factory created successfully.
Created Migration: 2020_06_20_033622_create_matakuliah_table
Seeder created successfully.
Controller created successfully.

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

Gambar: Hasil perintah php artisan make:model Matakuliah -cmfs

Perintah di atas akan men-generate 5 file berikut:

1. File model: app\Models\Matakuliah.php
2. File controller: app\Http\Controllers\MatakuliahController.php
3. File migration: database\migrations\<timestamp>create_matakuliah_table.php
4. File factory: database\factories\MatakuliahFactory.php
5. File seeder: database\seeders\MatakuliahSeeder.php

Baik, lanjut ke tahap berikutnya.

9.3. Isi File Migration

Tabel matakuliah akan memiliki 7 kolom: `id`, `kode_matakuliah`, `nama_matakuliah`, `jumlah_sks`, `nama_dosen`, `created_at` dan `update_at`. Berikut kode yang diperlukan untuk file migration:

```
database\migrations\<timestamp>create_matakuliah_table.php
```

```
1  public function up()
2  {
3      Schema::create('matakuliah', function (Blueprint $table) {
4          $table->id();
5          $table->char('kode_matakuliah',5)->unique();
6          $table->string('nama_matakuliah');
7          $table->integer('jumlah_sks');
8          $table->string('nama_dosen');
9          $table->timestamps();
10     });
}
```

```
11     }
```

Saya membuat tipe data kolom `kode_matakuliah` sebagai **char** dengan 5 karakter. Kolom ini memiliki tambahan key `unique()` sehingga tidak boleh terdapat data `kode_matakuliah` yang berulang.

Kemudian untuk kolom `nama_matakuliah` di set sebagai **string**, kolom `jumlah_sks` sebagai **integer** dan kolom `nama_dosen` juga sebagai **string**.

Lanjut dengan menjalankan proses migration:

```
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 (648.96ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (484.22ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (566.43ms)
Migrating: 2020_10_09_095756_create_matakuliahs_table
Migrated: 2020_10_09_095756_create_matakuliahs_table (1,306.32ms)

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

Gambar: Menjalankan migration

Jika tidak ada error, artinya tabel `matakuliahs` sudah berhasil dibuat.

9.4. Isi File Factory

Untuk mengisi tabel `matakuliahs`, kita berangkat dari file Factory. Namun agar mudah di uji coba, saya akan test generate data dari route terlebih dahulu. Jika datanya dirasa sudah sesuai, baru nanti dipindah ke file `MatakuliahFactory.php`.

Berikut kode program untuk proses generate data mata kuliah (menggunakan Faker):

```
routes\web.php
```

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/test-faker', function () {
6
7     $daftar_matakuliah= [
8         "Matematika", "Fisika Dasar", "Kimia Dasar", "Pengantar Rekayasa & Desain",
9         "Pengenalan Teknologi Informasi", "Bahasa Inggris", "Olah Raga",
10        "Pengantar Rekayasa & Desain", "Tata Tulis Karya Ilmiah",
11        "Pengantar Analisis Rangkaian", "Algoritma & Struktur Data",
12        "Matematika Diskrit", "Logika Informatika", "Probabilitas & Statistika",
13        "Aljabar Geometri", "Organisasi & Arsitektur Komputer",
```

```

14 "Pemrograman Berorientasi Objek", "Strategi Algoritma",
15 "Teori Bahasa Formal dan Otomata", "Sistem Operasi", "Basis Data",
16 "Dasar Rekayasa Perangkat Lunak", "Pengembangan Aplikasi Berbasis Web",
17 "Pengembangan Aplikasi pada Platform Khusus", "Jaringan Komputer",
18 "Manajemen Proyek Perangkat Lunak", "Manajemen Basis Data",
19 "Interaksi Manusia & Komputer", "Inteligensi Buatan", "Agama dan Etika",
20 "Sistem Paralel dan Terdistribusi", "Sistem Informasi", "Riset Operasi",
21 "Grafika Komputer", "Socio-Informatika dan Profesionalisme",
22 "Wawasan Teknologi & Komunikasi Ilmiah", "Algoritma & Struktur Data",
23 "Bahasa Inggris", "Pengantar Sistem Operasi", "Arsitektur SI/TI Perusahaan",
24 "Kepemimpinan & Ketrampilan Interpersonal", "Statistika",
25 "Desain & Manajemen Proses Bisnis", "Desain & Manajemen Jaringan Komputer",
26 "Dasar-Dasar Pengembangan Perangkat Lunak", "Pengantar Basis Data",
27 "Pemrograman Berorientasi Objek", "Analisis & Desain Perangkat Lunak",
28 "Interaksi Manusia & Komputer", "Keamanan Aset Informasi",
29 "Desain Basis Data", "Pemrograman Berbasis Web", "Sistem Cerdas",
30 "Konstruksi & Pengujian Perangkat Lunak", "Tata Tulis Ilmiah",
31 "Manajemen Proyek TI", "Simulasi Sistem", "Tata Kelola TI",
32 "Perencanaan Sumber Daya Perusahaan", "Manajemen Layanan TI",
33 "Dasar Pemrograman", "Proyek Perangkat Lunak", "Manajemen Resiko TI"];
34
35 $faker = \Faker\Factory::create('id_ID');
36
37 echo $faker->unique()->bothify('??###')."<br>";
38 echo $faker->randomElement($daftar_matakuliah)."<br>";
39 echo $faker->numberBetween(1,4)."<br>";
40 echo $faker->name()."<br>";
41 });

```

Di baris 7 – 33 saya membuat array \$daftar_matakuliah yang cukup panjang. Ini diperlukan karena faker belum menyediakan data spesifik seperti ini. Dengan menyiapkan array mata kuliah, data yang di generate akan terasa lebih nyata. Bagi yang pernah (atau sedang) kuliah di jurusan komputer, tentu tidak asing dengan daftar mata kuliah di atas.

Proses generate file sendiri di lakukan dengan kode program antara baris 37 – 40.

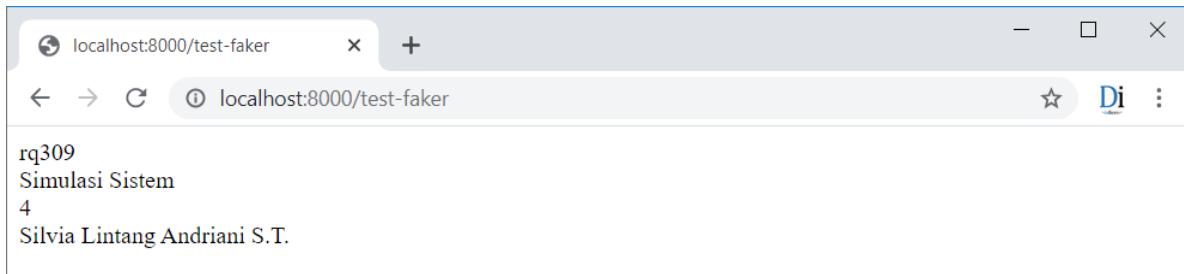
Perintah \$faker->unique()->bothify('??###') di siapkan untuk kolom kode_matakuliah. Nantinya data yang di generate terdiri dari 2 karakter huruf acak, kemudian disambung dengan 3 digit angka, seperti ax783 atau nv003. Tambahan method unique() memastikan tidak ada kode mata kuliah yang berulang.

Perintah \$faker->randomElement(\$daftar_matakuliah) di siapkan untuk kolom nama_matakuliah, yang nilainya akan diambil dari salah satu element array \$daftar_matakuliah. Karena tidak menggunakan method unique(), terdapat kemungkinan mata kuliah yang berulang. Ini tidak masalah karena kode mata kuliah pasti sudah berbeda. Selain itu dalam kenyataannya, satu mata kuliah bisa di ajar oleh beberapa dosen berbeda.

Perintah \$faker->numberBetween(1,4) disiapkan untuk kolom jumlah_sks. Nilai yang di generate berbentuk angka bulat antara 1 sampai 4.

Terakhir, perintah \$faker->name dipakai untuk mengisi kolom nama_dosen.

Silahkan akses route `localhost:8000/test-faker` di web browser dan cek apakah hasilnya sudah sesuai dengan keinginan kita. Saya sangat sarankan untuk selalu melakukan langkah ini setiap kali akan membuat Factory. Alasannya karena akan lebih sulit melihat hasil akhir di file factory daripada di route.



Gambar: Test generate data matakuliah di route

Baik, karena data yang ditampilkan saya anggap sudah pas, saatnya pindahkan kode ini ke `MatakuliahFactory.php`:

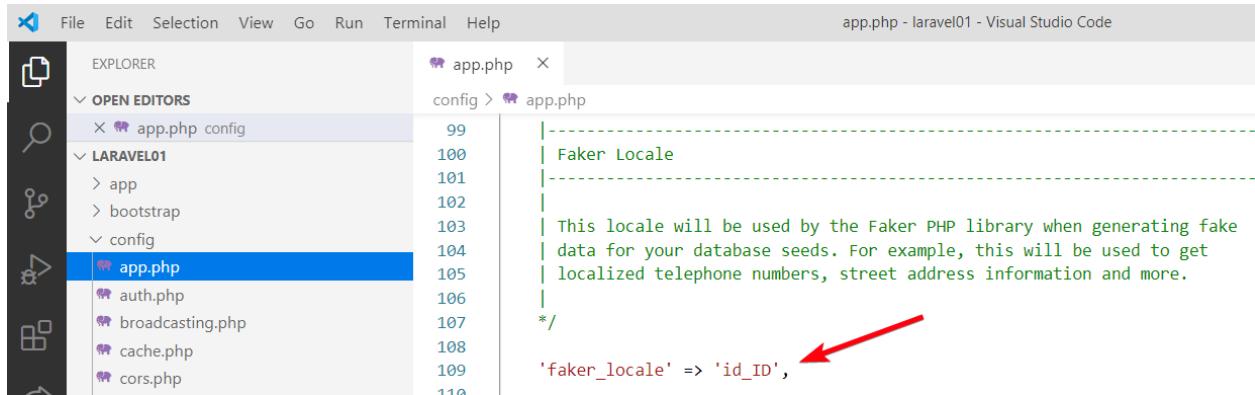
`database\factories\MatakuliahFactory.php`

```
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Matakuliah;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class MatakuliahFactory extends Factory
9 {
10     protected $model = Matakuliah::class;
11
12     public function definition()
13     {
14         $daftar_matakuliah= [
15             "Matematika", "Fisika Dasar", "Kimia Dasar", "Pengantar Rekayasa & Desain",
16             ...
17             ...
18             "Dasar Pemrograman", "Proyek Perangkat Lunak", "Manajemen Resiko TI"];
19
20         return [
21             'kode_matakuliah' => $this->faker->unique()->bothify('??###'),
22             'nama_matakuliah' => $this->faker->randomElement($daftar_matakuliah),
23             'jumlah_sks' => $this->faker->numberBetween(1,4),
24             'nama_dosen' => $this->faker->name(),
25         ];
26     }
27 }
```

Di dalam method `definition()` saya kembali mendefinisikan array `$daftar_matakuliah` (baris 14 - 18). Sama seperti di route, array ini berisi semua daftar mata kuliah. Yang harus diperhatikan, array `$daftar_matakuliah` ini ditulis sebelum perintah `return`.

Isi dari perintah `return` (baris 21 - 24) mirip seperti kode yang kita pakai di route, hanya saja sekarang menggunakan `$this->faker`, bukan lagi `$faker`. Setiap key array akan berpasangan dengan kolom tabel `matakuliah`.

Dan tidak lupa, agar nama dosen yang di generate Faker tampil dalam versi bahasa Indonesia, silahkan edit pengaturan '`faker_locale`' dari '`en_US`' menjadi '`id_ID`' di file `config/app.php`.



Gambar: Mengatur localization Faker

9.5. Isi File Seeder

Isi kode Factory sudah selesai, selanjutnya akan kita akses dari seeder. Silahkan buka file `database\seeders\MatakuliahSeeder.php` lalu tambah kode program berikut:

`database\seeders\MatakuliahSeeder.php`

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class MatakuliahSeeder extends Seeder
8 {
9     /**
10      * Run the database seeds.
11      *
12      * @return void
13      */
14     public function run()
15     {
16         \App\Models\Matakuliah::factory()->count(100)->create();
17     }
18 }
```

Tambahan kode hanya berupa 1 perintah di baris 16. Kode tersebut akan men-generate data mata kuliah sebanyak 100 baris. Kemudian kita harus daftarkan juga file `MatakuliahSeeder.php` ini ke file master seeder, yakni file `DatabaseSeeder.php`:

Case Study: Generating Data

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class DatabaseSeeder extends Seeder
8 {
9     /**
10      * Seed the application's database.
11      *
12      * @return void
13      */
14     public function run()
15     {
16         $this->call(MatakuliahSeeder::class);
17     }
18 }
```

Dalam file ini, kode yang perlu ditambah juga hanya 1 baris, yakni pemanggilan class MatakuliahSeeder di baris 16.

Persiapan untuk seeder sudah selesai. Saatnya generate data dengan menjalankan perintah:

```
php artisan db:seed
```

```
C:\xampp\htdocs\laravel01>php artisan db:seed
Seeding: MatakuliahSeeder
Seeded: MatakuliahSeeder (6.61 seconds)
Database seeding completed successfully.

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

Gambar: Jalankan file seeder

Siip, tidak ada masalah. Untuk memastikan, saya akan cek langsung ke dalam database:

matakuliah		dosen
id	kode_matakuliah	nama_dosen
1	uo566	Warta Firmansyah M.Ak
2	ok136	Wira Wacana
3	vu642	Elon Hutapea
4	ek107	Yulia Dewi Rahimah
5	qq647	Luthfi Thamrin
6	jk608	Salimah Febi Safitri
7	kw095	Diana Hassanah
8	fd026	Prabu Ramadan
9	oc648	Kunthara Kuswoyo

Gambar: Isi tabel matakuliah

Terlihat data hasil seeder sudah ada di dalam database.

9.6. Isi File Controller

Proses generate data mata kuliah sudah selesai. Berikutnya adalah menampilkan data tersebut ke web browser.

Sama seperti bab-bab sebelumnya, proses mengambil dan menampilkan data dari database bisa saja kita tulis langsung di route. Namun kali ini saya ingin menggunakan cara 'ideal', yakni menulis route untuk mengakses controller, dan dari controller ini data akan dikirim ke View.

Berikut route yang diperlukan:

```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MatakuliahController;
5
6 Route::get('/matakuliahs', [MatakuliahController::class, 'index']);
```

Dengan perintah route di baris 6, maka ketika URL '/matakuliahs' diakses, method `index()` yang terdapat di dalam file `MatakuliahController` akan dijalankan.

Jika sebelumnya anda menggunakan Laravel 7, maka penulisan route di atas terlihat sedikit berbeda. Di Laravel 8, penulisan route yang mengakses controller harus ditulis dalam bentuk class dan berada di dalam array.

Berikut kode program untuk method `index()` tersebut:

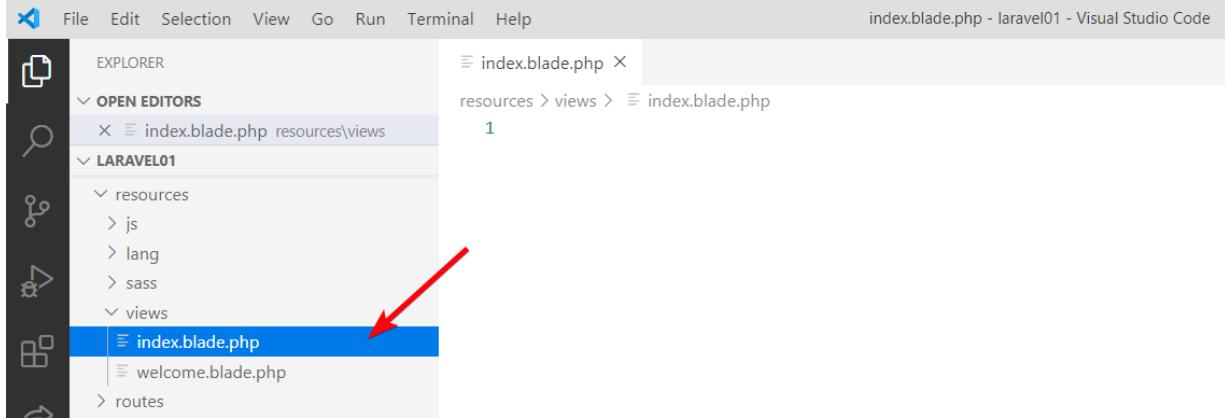
```
app\Http\Controllers\MatakuliahController.php

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MatakuliahController extends Controller
8 {
9     public function index()
10    {
11        $matakuliahs = \App\Models\Matakuliah::all();
12        return view('index',['matakuliahs' => $matakuliahs]);
13    }
14 }
```

Perintah `\App\Models\Matakuliah::all()` di baris 11 akan mengambil semua data mata kuliah yang ada di tabel `matakuliahs`. Hasilnya berbentuk collection yang kemudian disimpan ke dalam variabel `$matakuliahs` untuk selanjutnya dikirim ke view 'index' di baris 12.

9.7. Membuat View

File terakhir yang akan kita tulis adalah file view. Silahkan buat file view bernama `index.blade.php` di folder `resources\views\`.

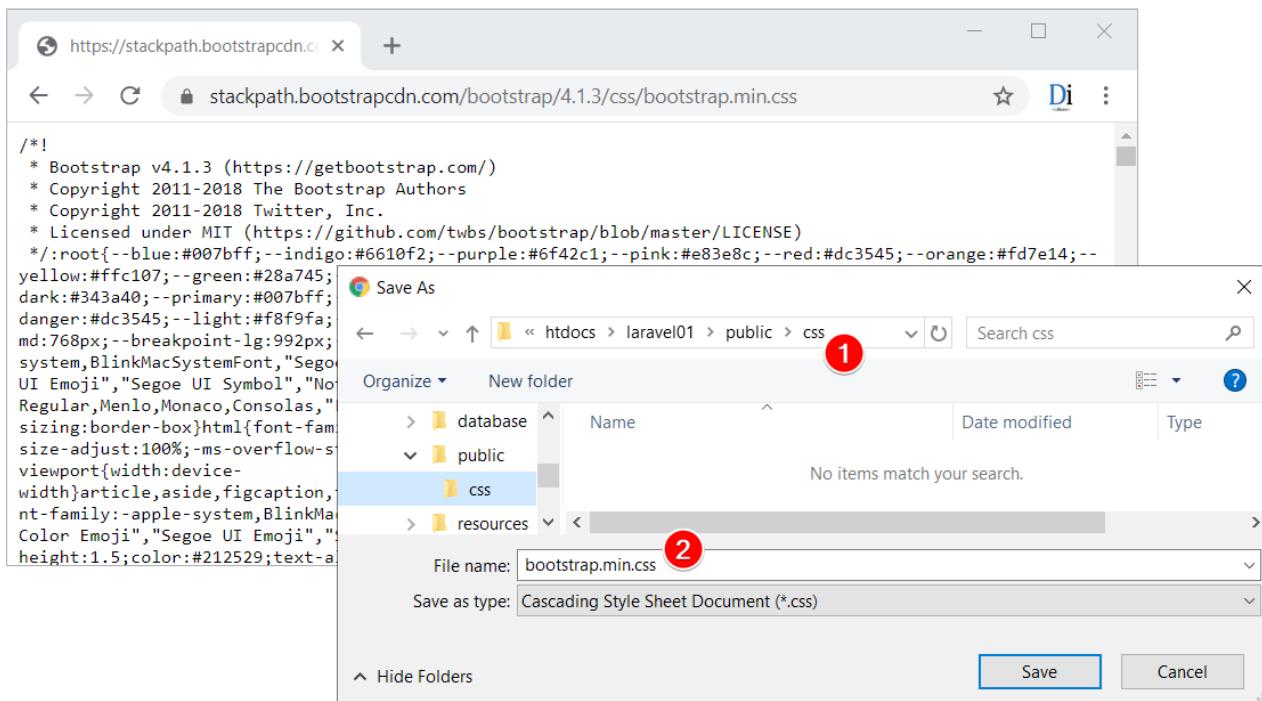


Gambar: Buat file view index.blade.php di folder resources\views\

Agar tampilan tabel lebih menarik, saya akan pakai bantuan framework CSS Bootstrap.

Silahkan download file CSS Bootstrap 4 dari <https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css>. Atau bisa juga akses dari file kode program di Google Drive.

Jika anda mengakses alamat file Bootstrap di atas, tekan tombol CRTL + S, lalu simpan di folder `laravel01\public\css` (1) sebagai `bootstrap.min.css` (2). Folder `css` ini memang belum tersedia di folder `public`, jadi silahkan buat terlebih dahulu.



Gambar: Cara download file bootstrap.min.css dan simpan ke laravel01\public\css.

Alternatif lain jika tidak mau repot mendownload, file CSS Bootstrap juga bisa langsung di akses dari CDN tersebut.

Kembali ke view blade `index.blade.php`, isi dengan kode program berikut:

`resources\views\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 Matakuliah</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 Matakuliah</h2>
18             </div>
19
20             <table class="table table-striped">
21                 <thead>
22                     <tr>
23                         <th>ID</th>
24                         <th>Kode Matakuliah</th>
25                         <th>Nama Matakuliah</th>
26                         <th>Jumlah SKS</th>
27                         <th>Nama Dosen</th>
28                     </tr>
29                 </thead>
30                 <tbody>
31                     @forelse ($matakuliah as $matakuliah)
32                         <tr>
33                             <th>{{$matakuliah->id}}</th>
34                             <td>{{$matakuliah->kode_matakuliah}}</td>
35                             <td>{{$matakuliah->nama_matakuliah}}</td>
36                             <td>{{$matakuliah->jumlah_sks}}</td>
37                             <td>{{$matakuliah->nama_dosen}}</td>
38                         </tr>
39                     @empty
40                         <td colspan="6" class="text-center">Tidak ada data...</td>
41                     @endforelse
42                 </tbody>
43             </table>
44         </div>
45     </div>
46 </div>
```

```
47  
48 </body>  
49 </html>
```

Di sini saya menampilkan data mata kuliah ke dalam tabel HTML. Data tersebut di proses menggunakan perulangan @forelse antara baris 31 – 41. Teknik ini sudah sering kita pakai di buku **Laravel Uncover**.

Silahkan akses alamat <http://localhost:8000/matakuliahs> untuk melihat tampilan view ini:

The screenshot shows a Microsoft Edge browser window with the title 'Data Matakuliah'. The address bar shows 'localhost:8000/matakuliahs'. The main content is a table titled 'Tabel Matakuliah' with the following data:

ID	Kode Matakuliah	Nama Matakuliah	Jumlah SKS	Nama Dosen
1	uo566	Dasar Pemrograman	1	Warta Firmansyah M.Ak
2	ok136	Pengenalan Teknologi Informasi	3	Wira Wacana
3	vu642	Bahasa Inggris	4	Elon Hutapea
4	ek107	Statistika	1	Yulia Dewi Rahimah
5	qq647	Inteligensi Buatan	4	Luthfi Thamrin
6	jk608	Bahasa Inggris	2	Salimah Febi Safitri

Gambar: Tampilan data mata kuliah

9.8. Menambah Accessor

Setelah melihat tampilan akhir, saya merasa kolom Kode Matakuliah akan lebih menarik jika tampil dalam huruf besar semua, seperti U0566 atau OK136.

Terdapat beberapa solusi untuk masalah ini. Yang paling mudah, terutama jika anda sudah cukup familiar dengan CSS, adalah menggunakan property `text-transform: uppercase` ke dalam tag `<td>`. Dengan demikian kode looping tabel menjadi sebagai berikut:

```
resources\views\index.blade.php
```

```
1 ...
2 @forelse ($matakuliahs as $matakuliah)
3   <tr>
4     <th>{{ $matakuliah->id }}</th>
5     <td style="text-transform: uppercase">{{ $matakuliah->kode_matakuliah }}</td>
6     <td>{{ $matakuliah->nama_matakuliah }}</td>
7     <td>{{ $matakuliah->jumlah_sks }}</td>
8     <td>{{ $matakuliah->nama_dosen }}</td>
9   </tr>
```

Case Study: Generating Data

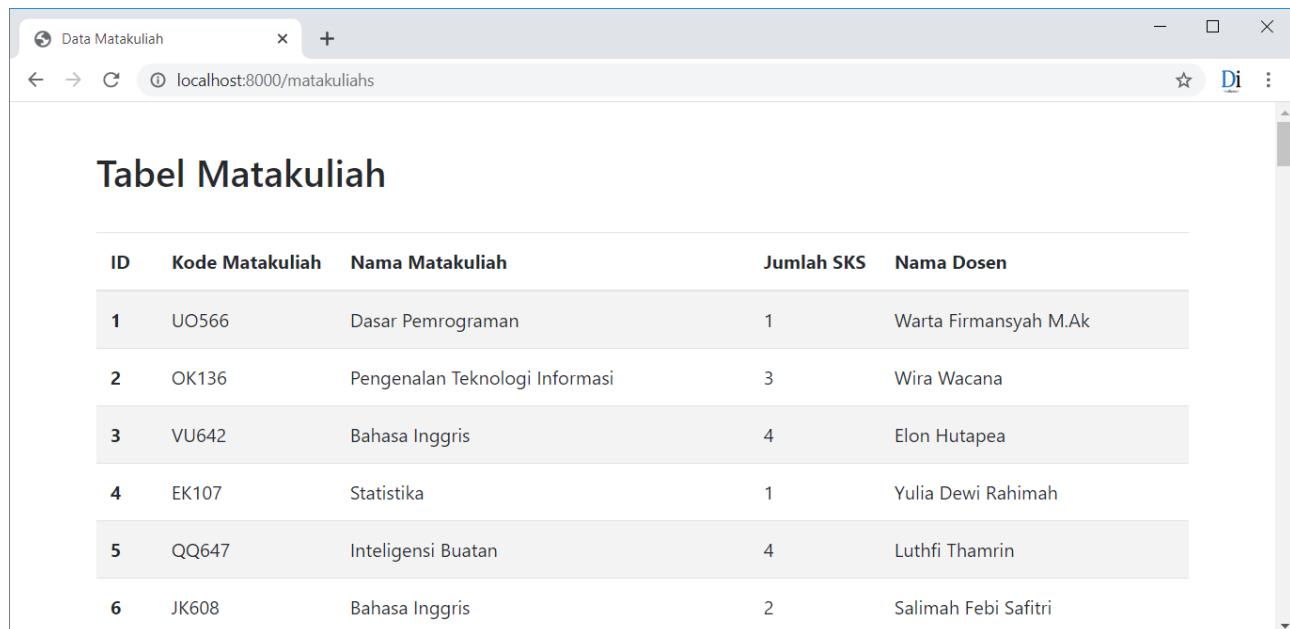
```
10 @empty
11     <td colspan="6" class="text-center">Tidak ada data...</td>
12 @endforelse
```

Cara kedua, menggunakan accessor agar ketika diakses, kolom kode_matakuliah langsung tampil dalam huruf besar. Silahkan tambah kode berikut ke dalam file app\Models\Matakuliah.php:

app\Models\Matakuliah.php

```
1 <?php
2
3 namespace App;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Matakuliah extends Model
8 {
9     public function getKodeMatakuliahAttribute($value)
10    {
11        return strtoupper($value);
12    }
13 }
```

Sekarang semua pemanggilan kolom kode_matakuliah dari eloquent akan tampil dengan huruf besar.



ID	Kode Matakuliah	Nama Matakuliah	Jumlah SKS	Nama Dosen
1	UO566	Dasar Pemrograman	1	Warta Firmansyah M.Ak
2	OK136	Pengenalan Teknologi Informasi	3	Wira Wacana
3	VU642	Bahasa Inggris	4	Elon Hutapea
4	EK107	Statistika	1	Yulia Dewi Rahimah
5	QQ647	Inteligensi Buatan	4	Luthfi Thamrin
6	JK608	Bahasa Inggris	2	Salimah Febi Safitri

Gambar: Tampilan data mata kuliah dengan accessor

Alternatif ketiga, bisa juga dengan mengubah langsung kode generate data di Factory, lalu jalankan ulang seeder:

database\factories\MatakuliahFactory.php

```
1 ...  
2 return [  
3     'kode_matakuliah' => strtoupper($this->faker->unique()->bothify('??###')),  
4     ...  
5 ];
```

Dari ketiga teknik di atas anda bisa pilih salah satu, yakni apakah menambah kode CSS, menggunakan accessor, atau dari factory.

Sering kali ada banyak banyak solusi untuk memecahkan satu masalah. Semakin banyak pengalaman membuat kode program, semakin siap kita dalam menghadapi hal-hal baru.

Dalam bab ini kita telah membuat studi kasus sederhana yang berfokus kepada proses generate data menggunakan Faker, Factory dan Seeder. Kemudian mengakses data tersebut dari controller dan menampilkannya di view. Cara seperti ini akan sering kita lakukan ketika membuat project baru.

Berikutnya, kita akan coba atasi salah satu masalah yang selalu muncul ketika menampilkan banyak data, yakni **Pagination**.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

10. Pagination

Ketika ingin menampilkan data dalam jumlah banyak, kurang efisien jika data itu langsung tampil dalam 1 halaman atau tabel yang panjang. Untuk 10 data tidak ada masalah, 100 data mungkin juga masih masuk akal. Tapi kalau 1000 baris tabel dalam 1 halaman tentu tidak lagi *user friendly*.

Salah satu solusi untuk mengatasinya adalah dengan **Pagination**. Dalam bab kali ini kita akan membahas apa itu pagination dan bagaimana cara membuatnya di Laravel.

Sebagai bahan praktik, saya akan sambung hasil bab **Case Study: Generating Data** sebelumnya. Dengan demikian kita sudah memiliki 100 data mata kuliah dan juga file-file dasar seperti model di `Matakuliah.php`, controller di `MatakuliahController.php`, serta file view `index.blade.php`.

10.1. Pengertian Pagination

Dalam web programming, **pagination** atau **paging** adalah teknik menampilkan banyak data dengan cara memecahnya menjadi beberapa halaman. Karena data tampil hanya sebagian, nanti terdapat tombol atau link sebagai mekanisme untuk pindah ke halaman berikutnya.

Pagination sering dipakai untuk memecah data yang disajikan dalam bentuk tabel atau pada web yang memiliki banyak artikel (seperti web forum dan blog) agar pengunjung bisa melihat artikel sebelum/selanjutnya.



Gambar: Tombol yang biasanya ada di bagian bawah sebuah pagination

Jika menggunakan PHP native, proses pembuatan pagination ini cukup rumit. Kita harus memecah data tabel menggunakan query `LIMIT`, menentukan berapa banyak data yang ingin ditampilkan, kemudian memproses query string sebagai patokan urutan data. Sebagai contoh, jika di buka halaman `http://localhost/index.php?halaman=5`, maka tampilkan halaman ke 5 dari urutan data.

Menggunakan Laravel, proses pagination ini bisa buat dengan mudah karena sudah menjadi fitur bawaan dari Query Builder dan Eloquent.

10.2. Membuat Pagination

Jika anda mengikuti praktek dari bab sebelumnya, maka kita sudah memiliki 100 data mata kuliah di dalam database. Ketika halaman `localhost:8000/matakuliahs` diakses, semua data ini langsung tampil dalam satu tabel yang cukup panjang. Supaya lebih mudah diakses, saya ingin membuat pagination yang menampilkan 10 data pada setiap halaman.

Proses pembuatan pagination di Laravel cukup sederhana, yakni panggil method `paginate()` di controller, lalu tambah 1 perintah `links()` ke dalam file blade. Mari kita coba.

Pertama, ganti cara pengaksesan data mata kuliah di `MatakuliahController` dari sebelumnya \App\Models\Matakuliah::all() menjadi \App\Models\Matakuliah::paginate(10):

`app\Http\Controllers\MatakuliahController.php`

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class MatakuliahController extends Controller
8 {
9     public function index()
10    {
11        $matakuliahs = \App\Models\Matakuliah::paginate(10);
12        return view('index', ['matakuliahs' => $matakuliahs]);
13    }
14 }
```

Angka 10 yang diinput sebagai argument dipakai untuk menentukan jumlah data yang diambil. Jika kita ingin menampilkan 15 data untuk setiap halaman, maka bisa ganti ke `paginate(15)`.

Setelah perubahan ini, langsung test buka halaman `localhost:8000/matakuliahs`.

ID	Kode	Nama Mata Kuliah	Skor	Dosen Pengajar
7	KW095	Manajemen Proyek Perangkat Lunak	2	Diana Hassannah
8	FD026	Manajemen Proyek Perangkat Lunak	2	Prabu Ramadan
9	OC648	Inteligensi Buatan	2	Kunthara Kuswoyo
10	TU413	Strategi Algoritma	2	Eva Rahayu

Gambar: Tampilan halaman `localhost:8000/matakuliahs`

Sekarang isi tabel yang tampil hanya 10, tidak lagi 100 baris. Inilah efek dari penggunaan method `paginate(10)`.

Pagination

Untuk menampilkan tombol navigasi di bagian bawah tabel, kita butuh edit file blade sesaat lagi. Atau bisa juga dengan menulis langsung query string ?page=<angka> setelah nama halaman. Sebagai contoh, silahkan ketik alamat `localhost:8000/matakuliah?page=5` di web browser lalu tekan Enter.

Halaman akan *reload* dan menampilkan data yang berbeda, yakni mata kuliah urutan 40 – 50 atau halaman ke 5 dari pagination dengan 10 data per halaman.

47	OQ314	Aljabar Geometri	3	Victoria Laksita S.Gz
48	IS357	Pemrograman Berorientasi Objek	3	Agus Sirait
49	PU413	Teori Bahasa Formal dan Otomata	3	Harjo Sihombing S.Farm
50	RL192	Basis Data	3	Upik Adiarja Mandala S.E.

Gambar: Tampilan halaman `localhost:8000/matakuliah?page=5`

Sekarang buka file `index.blade.php`, kita akan tambah sedikit kode program untuk menampilkan tombol navigasi pagination:

`resources\views\index.blade.php`

```
1 ...  
2 ...  
3  
4 <div class="container mt-3">  
5   <div class="row">  
6     <div class="col-12">  
7  
8       <div class="py-4">  
9         <h2>Tabel Matakuliah</h2>  
10      </div>  
11  
12      <table class="table table-striped">  
13        <thead>  
14          <tr>  
15            <th>#</th>  
16            <th>Kode Matakuliah</th>  
17            <th>Nama Matakuliah</th>  
18            <th>Jumlah SKS</th>  
19            <th>Nama Dosen</th>  
20          </tr>  
21        </thead>  
22        <tbody>  
23          @forelse ($matakuliah as $matakuliah)  
24            <tr>  
25              <th>{{$matakuliah->id}}</th>  
26              <td>{{$matakuliah->kode_matakuliah}}</td>
```

Pagination

```
27      <td>{{$matakuliah->nama_matakuliah}}</td>
28      <td>{{$matakuliah->jumlah_sks}}</td>
29      <td>{{$matakuliah->nama_dosen}}</td>
30  </tr>
31  @empty
32      <td colspan="6" class="text-center">Tidak ada data...</td>
33  @endforelse
34  </tbody>
35  </table>
36  </div>
37</div>
38
39<div class="row">
40  <div class="mx-auto mt-3">
41    {{ $matakulahs->links() }}
42  </div>
43</div>
44
45</div>
```

Untuk menampilkan tombol pagination di view, yang di perlukan hanyalah 1 baris perintah, yakni `{{ $matakulahs->links() }}` seperti di baris 41. Variabel `$matakulahs` di sini merujuk ke variabel collection yang dikirim dari controller.

Tambahan dua buah tag `<div>` di baris 39 dan 40 hanya untuk merapikan tampilan tombol, yakni supaya tampil rata tengah dan memiliki sedikit margin top dengan tabel di atasnya.

Akan tetapi jika anda me-refresh halaman `localhost:8000/matakulahs`, tampilan pagination terlihat berantakan. Ini karena di Laravel 8, secara default sudah menggunakan framework **Tailwind CSS**, bukan lagi **Bootstrap** sebagaimana di Laravel 7 ke bawah.

Namun tidak masalah, kita tetap bisa mengubahnya kembali ke Bootstrap. Caranya, buka file `app\Providers\AppServiceProvider.php`, lalu modifikasi sebagai berikut:

```
app\Providers\AppServiceProvider.php

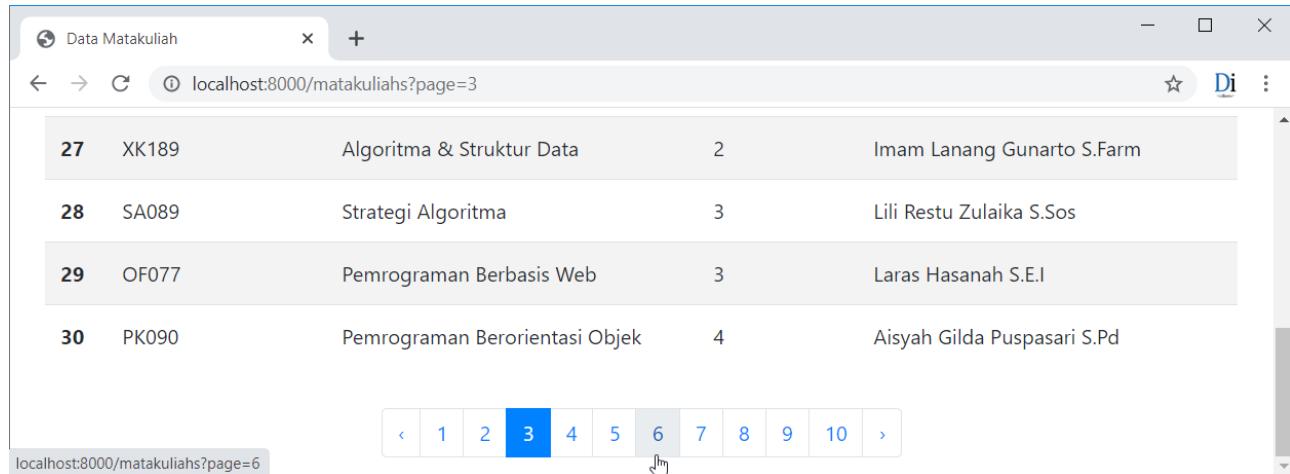
1  <?php
2
3  namespace App\Providers;
4
5  use Illuminate\Support\ServiceProvider;
6  use Illuminate\Pagination\Paginator;
7
8  class AppServiceProvider extends ServiceProvider
9  {
10      /**
11       * Register any application services.
12       *
13       * @return void
14       */
15      public function register()
16      {
17          //
```

Pagination

```
18     }
19
20     /**
21      * Bootstrap any application services.
22      *
23      * @return void
24      */
25     public function boot()
26     {
27         Paginator::useBootstrap();
28     }
29 }
```

Tambahannya hanya 2 baris, yakni `use Illuminate\Pagination\Paginator` di baris 6, serta `Paginator::useBootstrap()` di baris 27.

Save file `AppServiceProvider.php`, lalu buka kembali halaman `localhost:8000/matakuliahs`:



Gambar: Tampilan tombol navigasi pagination

Sekarang tampilan tombol pagination sudah sempurna. Setiap tombol navigasi ini sudah langsung berfungsi, termasuk tombol < dan > yang ada di bagian paling kiri dan paling kiri.

Laravel memang sering berganti-ganti teknologi terutama untuk library front-end. Ini memang jadi tantangan bagi kita karena terpaksa harus mempelajari library tersebut. Misalnya dalam kasus ini, secara default style untuk pagination sudah berganti dari **Bootstrap ke Tailwind CSS**.

Ini tidak otomatis menjadi bukti bahwa Tailwind CSS lebih baik daripada Bootstrap. Kebetulan saja, tim di balik framework Laravel ingin selalu mencoba sesuatu yang baru dan sedang populer.

Tapi apakah itu wajib dipakai? Tidak juga. Kuncinya sesuaikan dengan skill dan kebutuhan kita, terlebih Laravel juga masih menyediakan cara praktis jika ingin tetap menggunakan Bootstrap.

10.3. Pengaturan Pagination di Controller

Pagination yang kita buat sudah bisa langsung dipakai, namun kadang ini masih belum cukup. Dalam situasi tertentu, kita butuh fleksibilitas untuk mengatur tampilan pagination.

Terdapat 2 tempat untuk mengatur pagination, yakni di controller dan di file view. Pengaturan di controller biasanya berhubungan dengan data yang ingin ditampilkan. Sedangkan pengaturan di file view lebih ke mengubah bentuk tombol navigasi pagination.

Kita akan bahas pengaturan di controller terlebih dahulu.

Sebelumnya telah dibahas bahwa method `paginate()` bisa menerima 1 argument untuk menentukan jumlah data pada setiap halaman. Method ini juga bisa digabung dengan berbagai method Eloquent lain, misalnya seperti contoh berikut:

app\Http\Controllers\MatakuliahController.php

```

1 <?php
2 ...
3 class MatakuliahController extends Controller
4 {
5     public function index()
6     {
7         $matakuliahs = \App\Models\Matakuliah::where('jumlah_sks', 4)->paginate(5);
8         return view('index',['matakuliahs' => $matakuliahs]);
9     }
10 }
```

Di baris 7 saya menjalankan method `paginate(5)` ke dalam hasil `where('jumlah_sks', 4)`.

Dengan pemanggilan ini, data yang tampil dibatasi hanya untuk mata kuliah dengan 4 sks saja.

Atau bisa juga dijalankan sebagai berikut:

app\Http\Controllers\MatakuliahController.php

```

1 <?php
2 ...
3 class MatakuliahController extends Controller
4 {
5     public function index()
6     {
7         $matakuliahs = \App\Models\Matakuliah::where('jumlah_sks', 4)
8             ->orderBy('nama_matakuliah')
9             ->paginate(5);
10        return view('index',['matakuliahs' => $matakuliahs]);
11    }
12 }
```

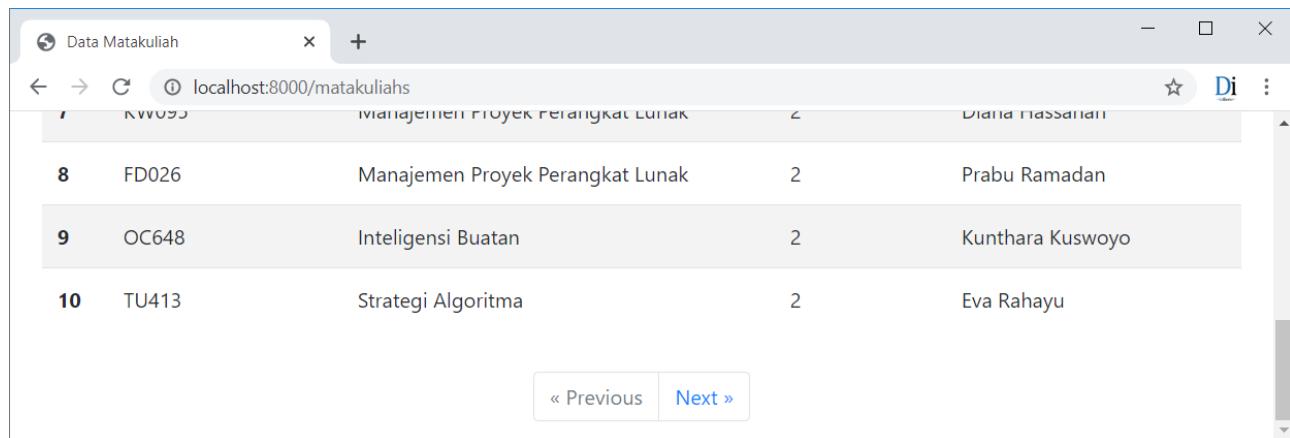
Sekarang selain membatasi jumlah sks, data juga di urutkan berdasarkan kolom `nama_matakuliah`.

Pagination

Selain method `paginate()`, terdapat juga method `simplePaginate()` untuk membuat tampilan pagination yang lebih sederhana:

app\Http\Controllers\MatakuliahController.php

```
1 <?php
2 ...
3 class MatakuliahController extends Controller
4 {
5     public function index()
6     {
7         $matakuliahs = \App\Models\Matakuliah::simplePaginate(10);
8         return view('index',['matakuliahs' => $matakuliahs]);
9     }
10 }
```



Gambar: Tampilan tombol navigasi untuk simple pagination

Seperti yang terlihat, tampilan tombol berubah menjadi hanya **Previous** dan **Next** saja, tidak lagi berbentuk angka-angka halaman.

10.4. Pengaturan Pagination di View

Pengaturan tampilan tombol navigasi pagination bisa kita lakukan dari file view. Perintah `$matakuliahs->links()` akan men-generate tampilan tombol berupa angka-angka, kecuali jika dari controller dipanggil `simplePaginate()`, maka tombol pagination akan berubah menjadi **Previous** dan **Next** saja.

Terdapat beberapa pengaturan lain yang akan kita bahas. Sebelum itu saya ingin mengubah data yang dikirim dari controller menjadi `paginate(5)`:

app\Http\Controllers\MatakuliahController.php

```
1 <?php
2 ...
3 class MatakuliahController extends Controller
4 {
```

Pagination

```
5     public function index()
6     {
7         $matakuliah = \App\Models\Matakuliah::paginate(5);
8         return view('index', ['matakuliah' => $matakuliah]);
9     }
10 }
```

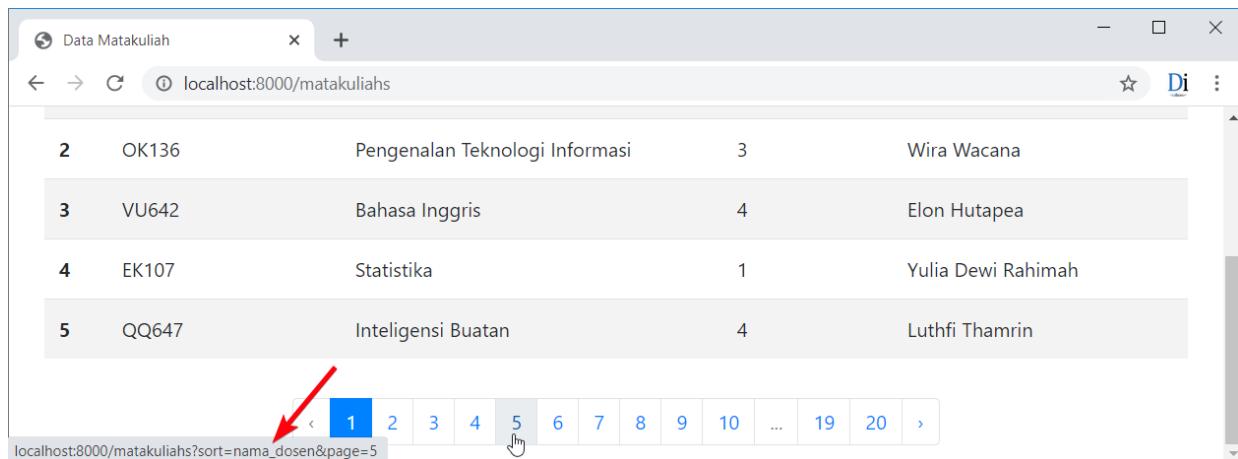
Dengan perintah `\App\Models\Matakuliah::paginate(5)`, maka setiap halaman sekarang akan menampilkan 5 data mata kuliah saja. Otomatis jumlah angka link navigasi juga semakin banyak, total menjadi $100/5 = 20$ tombol.

Menambah Query String (appends)

Jika kita ingin membuat fitur ke tambahan pagination, besar kemungkinan butuh penambahan query string. Untuk menambah query string ke dalam link navigasi, jalankan method `appends()` seperti contoh berikut:

`resources\views\index.blade.php`

```
1 ...
2     <div class="row">
3         <div class="mx-auto mt-3">
4             {{ $matakuliah->appends(['sort' => 'nama_dosen'])->links() }}
5         </div>
6     </div>
7 ...
```

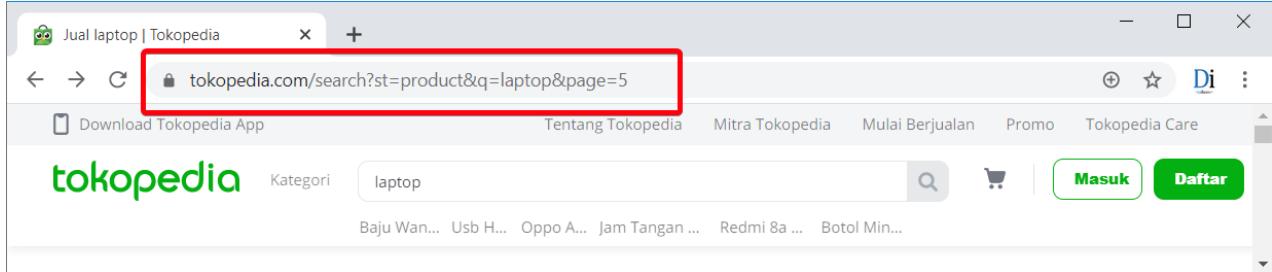


Gambar: Hasil penambahan perintah appends()

Sekarang semua link tombol navigasi akan memiliki tambahan `sort=nama_dosen` di bagian URL. Tambahan query string ini nantinya bisa diproses dari route atau dari dalam controller.

Query string biasa dipakai untuk membuat semacam fitur filter ke dalam pagination. Berikut contoh penerapan query string di halaman pagination Tokopedia:

Pagination



Gambar: Tampilan URL saat melakukan pencarian di tokopedia

Pada gambar di atas saya melakukan pencarian di website Tokopedia. Bisa terlihat di bagian URL ada 3 query string, yakni `st=product`, `q=laptop` dan `page=5`.

Fungsi dari setiap query string ini tergantung dari programmer Tokopedia, tapi bisa di tebak bahwa `st=product` dipakai untuk menentukan jenis kategori pencarian, `q=laptop` untuk keyword pencarian dan `page=5` untuk halaman pagination ke 5.

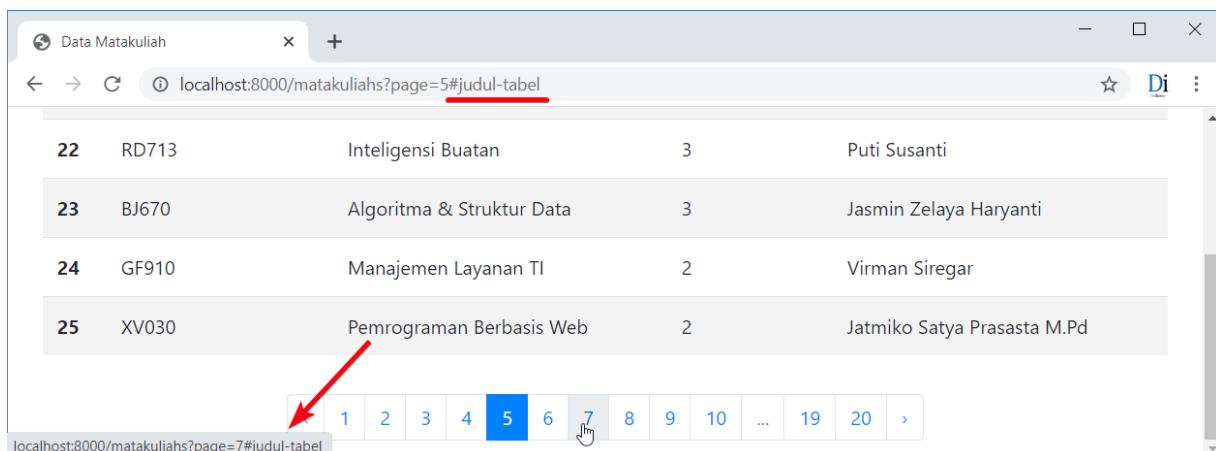
Penambahan fitur filter di halaman pagination Laravel tidak akan kita bahas karena sedikit kompleks (melibatkan pemrosesan form). Tapi setidaknya Laravel menyediakan cara mudah untuk menambahkan query string ke link navigasi pagination.

Menambah Hash Fragment (fragment)

Hash fragment adalah sebutan untuk karakter `#` di akhir URL. Untuk membuatnya, bisa memakai perintah `fragment()` seperti contoh berikut:

`resources\views\index.blade.php`

```
1 ...  
2     <div class="row">  
3         <div class="mx-auto mt-3">  
4             {{ $matakuliahs->fragment('judul-tabel')->links() }}  
5         </div>  
6     </div>  
7 ...
```



Gambar: Hasil penambahan perintah `fragment()`

Pagination

Terdapat tambahan `#judul-tabel` pada akhir link pagination. Hash fragment sendiri sering dipakai untuk membuat link internal dalam sebuah halaman.

Sebagai contoh praktik, saya akan manfaatkan fitur hash fragment ini untuk "menahan" tampilan tabel supaya tidak lompat ke bagian atas pada saat link pagination di klik. Silahkan modifikasi tag `<h2>Tabel Mata Kuliah</h2>` di bagian atas file view, dari sebelumnya:

`resources\views\index.blade.php`

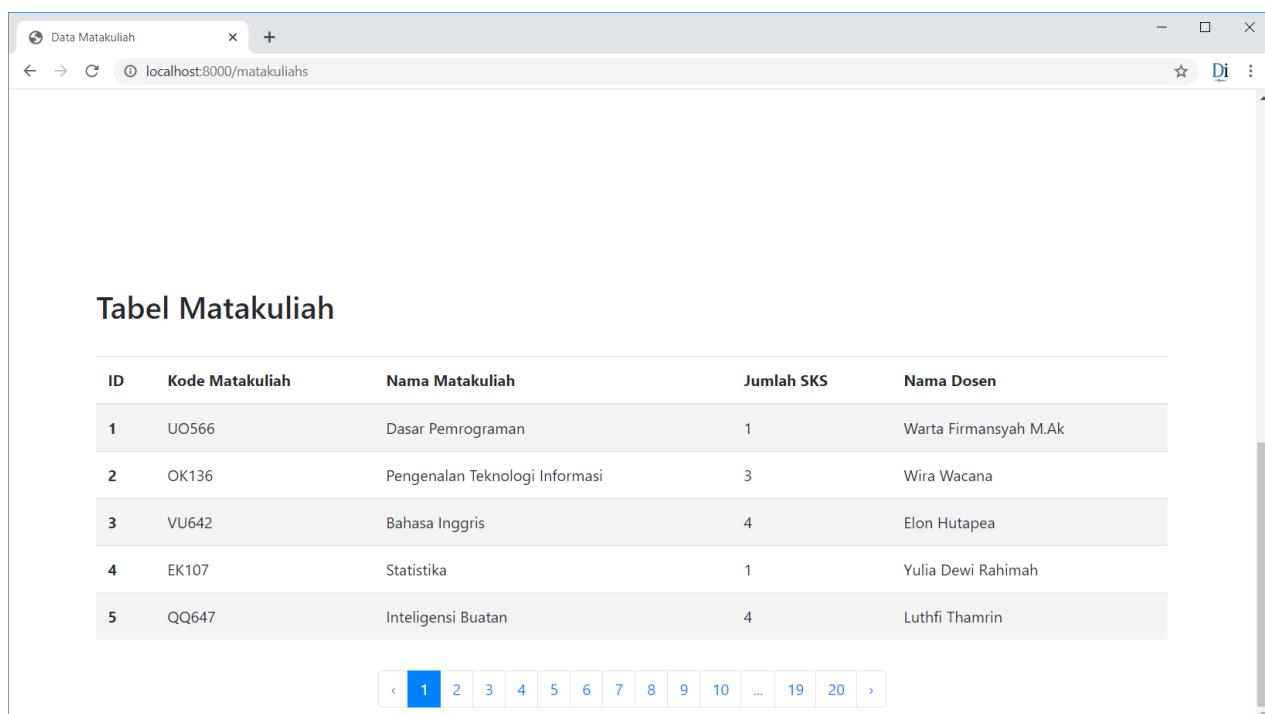
```
1 ...  
2 <div class="py-4">  
3   <h2>Tabel Mata Kuliah</h2>  
4 </div>  
5 ...
```

Menjadi:

```
1 ...  
2 <div class="py-4">  
3   <h2 style="margin-top:1000px" id="judul-tabel">Tabel Mata Kuliah</h2>  
4 </div>  
5 ...
```

Di baris 3 terdapat tambahan atribut `style="margin-top:1000px"` dan `id="judul-tabel"`. Kode CSS `margin-top:1000px` akan membuat tabel terdorong cukup jauh ke bagian bawah. Kemudian atribut `id="judul-tabel"` dipakai sebagai target `anchor` dari `#judul-tabel`.

Silahkan buka halaman `localhost:8000/matakuliah`, maka tabel mata kuliah sekarang ada di bagian bawah halaman web (harus di scroll untuk melihatnya):



Gambar: Tabel mata kuliah ada di bagian bawah halaman

Pagination

Teks klik salah satu link pagination. Akan terbuka halaman baru dimana tampilan web browser langsung menuju tabel mata kuliah yang ada di bagian bawah. Inilah efek dari link internal yang dibuat dengan *hash fragment #judul-tabel*.

Di HTML, jika sebuah URL memiliki tambahan `#judul-tabel`, itu akan ter-link ke satu element yang memiliki atribut `id="judul-tabel"`. Dalam contoh kita, atribut tersebut adalah tag `<h2>Tabel Mata Kuliah</h2>`.

Untuk membuktikan, silahkan ganti perintah `$matakuliah->fragment('judul-tabel')->links()` menjadi `$matakuliah->links()`. Sekarang setiap kali link pagination di klik, halaman reload dan menampilkan bagian atas saja.

Sebelum lanjut, hapus kembali atribut `style="margin-top:1000px"` dan `id="judul-tabel"` untuk tag `<h2>Tabel Mata Kuliah</h2>` supaya posisi tabel mata kuliah kembali ke normal.

Mengatur Pagination Link Window (onEachSide)

Sebelumnya kita sudah mengubah pemanggilan pagination di controller menjadi `paginate(5)`. Ini menghasilkan cukup banyak link pagination, total 20 buah untuk 20 halaman. Silahkan klik nomor urut 10:



Gambar: Tampilan tombol pagination ketika link ke 10 di klik

Laravel berusaha menampilkan semua tombol halaman, namun karena total halaman cukup banyak, maka tidak semua tombol bisa terlihat. Pada saat link halaman 10 di klik, di sisi kiri dan kanan terdapat 3 nomor halaman lain, yakni 7, 8, 9 di sisi kiri dan 11, 12, 13 di sisi kanan.

Kita bisa mengatur jumlah link di sisi kanan dan kiri ini dengan menambah perintah `onEachSide()` seperti contoh berikut:

`resources\views\index.blade.php`

```
1 ...  
2 <div class="row">  
3   <div class="mx-auto mt-3">  
4     {{ $matakuliah->onEachSide(1)->links() }}  
5   </div>  
6 </div>  
7 ...
```

Pagination



Gambar: Tampilan tombol pagination ketika link ke 10 di klik

Method `onEachSide()` menerima 1 buah argument berupa jumlah link yang akan tampil di kedua sisi link. Dalam kode di atas saya menjalankan `$matakuliahs->onEachSide(1)->links()`, sehingga terdapat 1 link untuk setiap sisi.

Generate Template Link Pagination

Bawaan Laravel, tombol pagination sudah langsung memiliki kode Tailwind atau Bootstrap. Bagaimana jika kita ingin memakai framework CSS lain atau ingin mengubah tampilan tombol-tombol dengan kode CSS sendiri?

Ini bisa dilakukan dengan mengedit langsung file template tombol pagination. Namun file template harus di generate terlebih dahulu menggunakan perintah **php artisan** berikut:

```
php artisan vendor:publish --tag=laravel-pagination
```

```
Laravel Project
C:\xampp\htdocs\laravel01>php artisan vendor:publish --tag=laravel-pagination
Copied Directory [vendor\laravel\framework\src\Illuminate\Pagination\resources\views]
To [\resources\views\vendor\pagination]
Publishing complete.

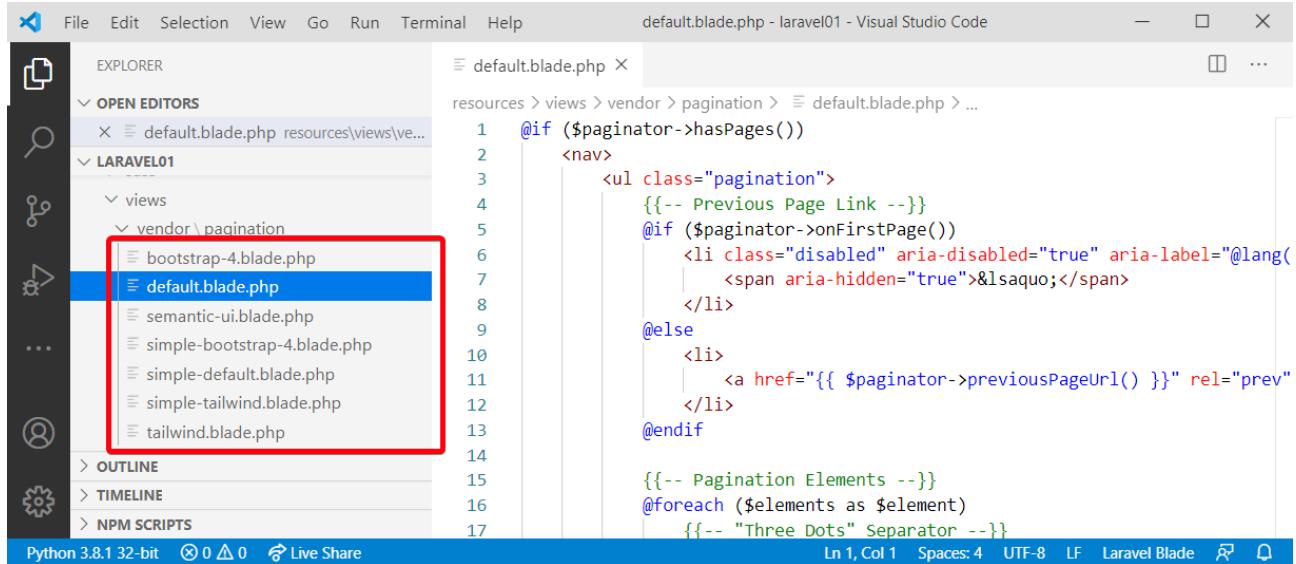
C:\xampp\htdocs\laravel01>
```

Gambar: Proses generate file template pagination

Setelah menjalankan perintah di atas, akan muncul folder baru di `\resources\views\vendor\pagination\`. Terdapat 7 file template bawaan Laravel 8. Jumlah template ini mungkin akan bertambah tergantung update dari tim Laravel:

- ◆ `bootstrap-4.blade.php`
- ◆ `default.blade.php`
- ◆ `semantic-ui.blade.php`
- ◆ `simple-bootstrap-4.blade.php`
- ◆ `simple-default.blade.php`
- ◆ `simple-tailwind.blade.php`
- ◆ `tailwind.blade.php`

Pagination



```
default.blade.php - laravel01 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
resources > views > vendor > pagination > default.blade.php ...
1 @if ($paginator->hasPages())
2   <nav>
3     <ul class="pagination">
4       {{-- Previous Page Link --}}
5       @if ($paginator->onFirstPage())
6         <li class="disabled" aria-disabled="true" aria-label="@lang('...
7           | <span aria-hidden="true">&lsquo;</span>
8         </li>
9       @else
10        <li>
11          | <a href="{{ $paginator->previousPageUrl() }}" rel="prev"
12        </li>
13      @endif
14
15      {{-- Pagination Elements --}}
16      @foreach ($elements as $element)
17        {{-- "Three Dots" Separator --}}
Ln 1, Col 1 Spaces: 4 UTF-8 LF Laravel Blade ⚡ 🔍
```

Python 3.8.1 32-bit 0 Δ 0 Live Share

Gambar: File template pagination

Berdasarkan nama file, kita bisa menebak jenis framework CSS yang disediakan Laravel:

Bootstrap 4 (`bootstrap-4.blade.php`), **Semantic UI** (`semantic-ui.blade.php`) dan **Tailwind** (`tailwind.blade.php`).

Khusus untuk **Bootstrap** dan **Tailwind**, juga hadir dalam versi *simple*, yakni `simple-bootstrap-4.blade.php` dan `simple-tailwind.blade.php`.

Terakhir terdapat 2 file default: `default.blade.php` dan `simple-default.blade.php`. Kedua file inilah yang secara tidak langsung kita gunakan sejak awal bab. Jika dari controller dipanggil method `paginate()`, template yang dipakai adalah `default.blade.php`. Sedangkan jika dipanggil `simplePaginate()`, template yang dipakai adalah `simple-default.blade.php`.

Untuk mengubah tampilan tombol link pagination, input salah satu nama file template sebagai argument dari method `links()`. Sebagai contoh, jika ingin memakai template `semantic-ui.blade.php`, jalankan method `$matakuliahs->links('vendor.pagination.semantic-ui')`.

Berikut prakteknya:

`resources\views\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    <link rel="stylesheet"
9      href="https://cdn.jsdelivr.net/npm/semantic-ui@2.4.2/dist/semantic.min.css">
10
11  <title>Data Matakuliah</title>
12 </head>
```

Pagination

```
13 <body>
14 ...
15 ...
16   <div class="row">
17     <div class="mx-auto mt-3">
18       {{ $matakuliah->links('vendor.pagination.semantic-ui') }}
19     </div>
20   </div>
21 ...
22 </body>
23 </html>
```

The screenshot shows a web browser window titled "Data Matakuliah". The URL is "localhost:8000/matakuliah". The page contains a table titled "Tabel Matakuliah" with the following data:

ID	Kode Matakuliah	Nama Matakuliah	Jumlah SKS	Nama Dosen
1	UO566	Dasar Pemrograman	1	Warta Firmansyah M.Ak
2	OK136	Pengenalan Teknologi Informasi	3	Wira Wacana
3	VU642	Bahasa Inggris	4	Elon Hutapea
4	EK107	Statistika	1	Yulia Dewi Rahimah
5	QQ647	Inteligensi Buatan	4	Luthfi Thamrin

At the bottom of the table, there is a navigation bar with the following buttons: < (left arrow), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ..., 19, 20, > (right arrow).

Gambar: Tombol pagination dengan style dari Semantic UI

Di baris 8-9 terdapat tag `<link>` untuk mengakses file `semantic.min.css` yang berada di CDN. Ini di perlukan agar tampilan tombol sesuai dengan desain [Semantic UI](#). Karena file CSS berada di CDN, maka halaman blade harus terhubung ke internet pada saat di jalankan.

Di baris 18, terdapat perintah `$matakuliah->links('vendor.pagination.semantic-ui')`. Artinya, design tombol pagination akan memakai kode yang terdapat di file `semantic-ui.blade.php`.

Membuat Template Link Pagination

Apabila design template pagination masih kurang sesuai, kita bisa edit langsung file template tersebut, atau bisa juga membuat file template yang baru. Caranya, buat sebuah file blade di folder `vendor/pagination/`, lalu akses file tersebut menggunakan method `links()`.

Saya akan buat file baru bernama `matakuliah.blade.php` di dalam folder `vendor/pagination/`. Agar tidak terlalu lama merancang kode HTML dan CSS, saya akan copy isi file `simple-bootstrap-4.blade.php`. Lalu ganti 2 kode berikut:

Pagination

- ◆ `@lang('pagination.previous')` menjadi Mundur di baris 7 dan 11
- ◆ `@lang('pagination.next')` menjadi Maju di baris 18 dan 22

```
matakuliah.blade.php
resources > views > vendor > pagination > matakuliah.blade.php > ...
1  @if ($paginator->hasPages())
2    <nav>
3      <ul class="pagination">
4        {{-- Previous Page Link --}}
5        @if ($paginator->onFirstPage())
6          <li class="page-item disabled" aria-disabled="true">
7            <span class="page-link">@lang('pagination.previous')</span>
8          </li>
9        @else
10          <li class="page-item">
11            <a class="page-link" href="{{ $paginator->previousPageUrl() }}" rel="prev">@lang('pagination.previous')
12          </li>
13        @endif
14
15        {{-- Next Page Link --}}
16        @if ($paginator->hasMorePages())
17          <li class="page-item">
18            <a class="page-link" href="{{ $paginator->nextPageUrl() }}" rel="next">@lang('pagination.next')
19          </li>
20        @else
21          <li class="page-item disabled" aria-disabled="true">
22            <span class="page-link">@lang('pagination.next')</span>
23          </li>
24        @endif
25      </ul>
26    </nav>
27  @endif
28
```

Gambar: Isi kode template pagination yang berasal dari simple-bootstrap-4.blade.php

Berikut kode program final dari file `matakuliah.blade.php`:

`resources\views\vendor\pagination\matakuliah.blade.php`

```
1  @if ($paginator->hasPages())
2    <nav>
3      <ul class="pagination">
4        {{-- Previous Page Link --}}
5        @if ($paginator->onFirstPage())
6          <li class="page-item disabled" aria-disabled="true">
7            <span class="page-link">Mundur</span>
8          </li>
9        @else
10          <li class="page-item">
11            <a class="page-link" href="{{ $paginator->previousPageUrl() }}" rel="prev">Mundur</a>
12          </li>
13        @endif
14
15        {{-- Next Page Link --}}
16        @if ($paginator->hasMorePages())
17          <li class="page-item">
18            <a class="page-link" href="{{ $paginator->nextPageUrl() }}" rel="next">Maju</a>
19          </li>
20        @else
21          <li class="page-item disabled" aria-disabled="true">
22            <span class="page-link">Maju</span>
23          </li>
24        @endif
25      </ul>
26    </nav>
27  @endif
28
```

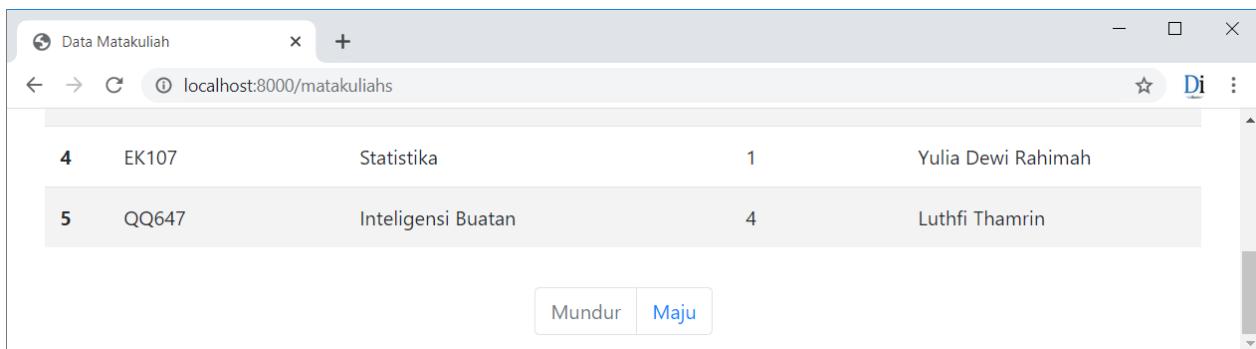
Pagination

```
23      <li class="page-item disabled" aria-disabled="true">
24          <span class="page-link">Maju</span>
25      </li>
26  @endif
27  </ul>
28  </nav>
29 @endif
```

Setelah itu jalankan method `$matakuliah->links('vendor.pagination.matakuliah')` dari file `index.blade.php`:

`resources\views\index.blade.php`

```
1 ...
2  <div class="row">
3      <div class="mx-auto mt-3">
4          {{ $matakuliah->links('vendor.pagination.matakuliah') }}
5      </div>
6  </div>
7 ...
```



Gambar: Tombol pagination yang berasal dari file template matakuliah.blade.php

Terlihat tampilan dari tombol pagination sudah berubah menjadi Mundur dan Maju, sesuai dengan kode yang ada di file `matakuliah.blade.php`.

10.5. Paginator Instance Methods

Dalam praktek sebelum ini, kita sudah pelajari cara membuat file template baru untuk mendesain tombol pagination.

Jika anda mempelajari file template pagination bawaan Laravel, terlihat cukup banyak perintah blade tambahan, terutama yang diakses dari **\$paginator object** seperti `$paginator->onFirstPage()`, `$paginator->hasMorePages()`, atau `$paginator->currentPage()`.

Di dalam template pagination, `$paginator object` ini berisi berbagai info tentang proses pagination, misalnya apakah masih ada halaman berikutnya, berapa jumlah data per halaman atau berapa jumlah total sisa halaman. Informasi seperti ini sangat berguna untuk merancang tampilan pagination.

Pagination

Laravel menyediakan cukup banyak method yang bisa diakses dari \$paginator object, kita akan lihat sekilas beberapa diantaranya. Sebagai bahan praktek, silahkan buat file route baru:

routes\web.php

```
1 <?php
2 use App\Http\Controllers\MatakuliahController;
3
4 Route::get('/matakuliahs', [MatakuliahController::class, 'index']);
5 Route::get('/test-paginate', [MatakuliahController::class, 'testPaginate']);
```

Lalu isi method testPaginate() di MatakuliahController dengan kode berikut:

app\Http\Controllers\MatakuliahController.php

```
1 <?php
2 ...
3     public function testPaginate()
4     {
5         $matakuliahs = \App\Models\Matakuliah::paginate(5);
6         return view('test',[ 'matakuliahs' => $matakuliahs]);
7     }
```

Method ini akan mengakses view bernama test.blade.php dengan mengirim data pagination \$matakuliahs. Berikut isi dari file view test.blade.php:

resources\views\test.blade.php

```
{{ $matakuliahs->links('vendor.pagination.test-paginate') }}
```

Yup, hanya 1 baris saja. Yakni perintah link() yang akan mengakses file test-paginate.blade.php di folder vendor\pagination\. Mari buat file ini dengan kode berikut:

resources\views\vendor\pagination\test-paginate.blade.php

```
1 <?php
2
3 echo "count() = ". $paginator->count(); echo "<br>";
4 echo "currentPage() = ". $paginator->currentPage(); echo "<br>";
5 echo "firstItem() = ". $paginator->firstItem(); echo "<br>";
6 echo "hasPages() = ". $paginator->hasPages(); echo "<br>";
7 echo "hasMorePages() = ". $paginator->hasMorePages(); echo "<br>";
8 echo "lastItem() = ". $paginator->lastItem(); echo "<br>";
9 echo "lastPage() = ". $paginator->lastPage(); echo "<br>";
10 echo "nextPageUrl() = ". $paginator->nextPageUrl(); echo "<br>";
11 echo "onFirstPage() = ". $paginator->onFirstPage(); echo "<br>";
12 echo "previousPageUrl() = ". $paginator->previousPageUrl(); echo "<br>";
13 echo "total() = ". $paginator->total(); echo "<br>";
14 echo "getPageName() = ". $paginator->getPageName(); echo "<br>";
15 echo "url(5) = ". $paginator->url(5); echo "<br>";
16 echo "getOptions() ="; dump($paginator->getOptions()); echo "<br>";
17 echo "items() ="; dump($paginator->items()); echo "<br>";
18 echo "getUrlRange(2,4) ="; dump($paginator->getUrlRange(2, 4)); echo "<br>";
```

Pagination

```
19 echo "\$elements = ";           dump($elements);           echo "<hr>";
```

Hasil kode program:

```
count() = 5
currentPage() = 1
firstItem() = 1
hasPages() = 1
hasMorePages() = 1
lastItem() = 5
lastPage() = 20
nextPageUrl() = http://localhost:8000/test-paginate?page=2
onFirstPage() = 1
previousPageUrl() =
total() = 100
getPageName() = page
url(5) = http://localhost:8000/test-paginate?page=5
getOptions() =
array:2 [▼
    "path" => "http://localhost:8000/test-paginate"
    "pageName" => "page"
]
items() =
array:5 [▼
    0 => App\Models\Matakuliah {#1187 ▶}
    1 => App\Models\Matakuliah {#1188 ▶}
    2 => App\Models\Matakuliah {#1189 ▶}
    3 => App\Models\Matakuliah {#1190 ▶}
    4 => App\Models\Matakuliah {#1191 ▶}
]
getUrlRange(2,4) =
array:3 [▼
    2 => "http://localhost:8000/test-paginate?page=2"
    3 => "http://localhost:8000/test-paginate?page=3"
    4 => "http://localhost:8000/test-paginate?page=4"
]
$elements =
array:3 [▼
    0 => array:10 [▶]
    3 => ...
    4 => array:2 [▶]
]
```

Dalam file `test-paginate.blade.php` saya tidak membuat tampilan tombol pagination, tapi langsung memeriksa berbagai method dari `$paginator object`. Berikut penjelasan dari setiap method:

- `$paginator->count()`: Tampilkan jumlah data untuk halaman saat ini.
- `$paginator->currentPage()`: Tampilkan urutan halaman saat ini.
- `$paginator->firstItem()`: Tampilkan urutan dari data pertama halaman saat ini.
- `$paginator->hasPages()`: Apakah total data bisa dipecah ke dalam beberapa halaman.

- `$paginator->hasMorePages()`: Apakah masih ada data berikutnya yang bisa ditampilkan.
- `$paginator->lastItem()`: Tampilkan urutan dari data terakhir halaman saat ini
- `$paginator->lastPage()`: Tampilkan angka untuk halaman terakhir (tidak tersedia jika menggunakan simplePaginate).
- `$paginator->nextPageUrl()`: Tampilkan alamat URL untuk halaman berikutnya.
- `$paginator->onFirstPage()`: Apakah halaman saat ini merupakan halaman pertama.
- `$paginator->previousPageUrl()`: Tampilkan alamat URL untuk halaman sebelumnya.
- `$paginator->total()`: Tampilkan jumlah total data (tidak tersedia jika menggunakan simplePaginate).
- `$paginator->getPageName()`: Tampilan nama query string yang dipakai untuk menyimpan urutan halaman.
- `$paginator->url($page)`: Tampilkan URL halaman dari urutan halaman yang diinput sebagai argument.
- `$paginator->getOptions()`: Tampilkan array dari *paginator options*.
- `$paginator->items()`: Tampilkan array dari data untuk halaman saat ini.
- `$paginator->getUrlRange($start, $end)`: Generate array daftar nama halaman dari angka yang diinput ke dalam argument.
- `$elements`: Array yang berisi daftar nama halaman,

Bermodalkan informasi dari method-method ini, kita bisa merancang tampilan tombol pagination dari nol. Jika butuh contoh penerapan langsung, silahkan buku file template pagination bawaan Laravel. Namun memang ini lebih ditujukan untuk level advanced, dalam pembuatan project pada umumnya, pagination bawaan Laravel sudah mencukupi.

Dalam bab ini kita telah mempelajari salah satu fitur yang hampir selalu kita butuhkan ketika menampilkan data dalam jumlah banyak. Proses pembuatan pagination di Laravel sangat mudah, namun tetap menyediakan cara jika ingin membuat custom pagination dari nol.

Lanjut, kita akan masuk ke bab **Eloquent Lanjutan**.

11. Eloquent Lanjutan

Materi **Eloquent** sebenarnya sudah kita pelajari di buku **Laravel Uncover**. Namun masih ada beberapa fitur tambahan yang belum sempat di bahas, misalnya tentang cara mengubah nama tabel default untuk model, serta berbagai method yang dipakai untuk menampilkan data tabel. Dalam bab ini akan dibahas beberapa materi Eloquent lanjutan yang sayang untuk dilewati.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel101
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

11.1. Mengubah Nama Tabel

Secara bawaan, Eloquent mencari nama tabel berdasarkan versi plural (jamak) dari nama model. Sebagai contoh, jika kita memiliki model `Mahasiswa`, maka nama tabel yang akan diakses adalah `mahasiswas`. Atau untuk model `Dosen`, nama tabel yang akan diakses adalah `dosens`.

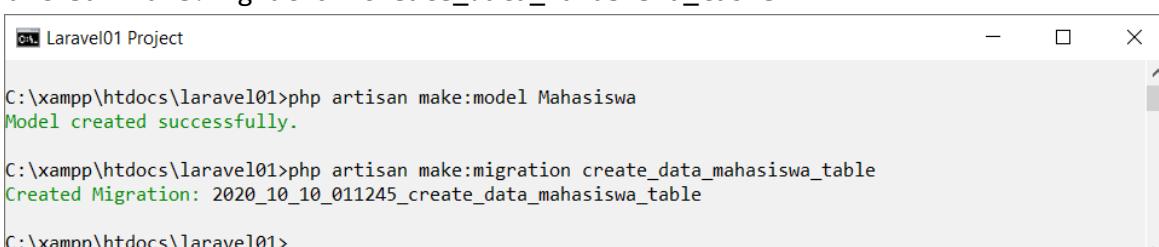
Aturan penamaan ini juga otomatis dipakai pada saat men-generate model dengan tambahan flag `-m`, misalnya: `php artisan make:model Mahasiswa -m`. Flag `-m` ini akan langsung membuat file migration dengan nama tabel `mahasiswas`. Bagi kita yang menggunakan bahasa indonesia, tambahan 's' di akhir nama tabel ini terkesan cukup aneh, seperti `barang`, `bukus`, atau `gurus`.

Solusi yang banyak dipakai oleh programmer Laravel di Indonesia adalah tetap menggunakan kata bahasa inggris sebagai nama tabel dan model. Jadi daripada membuat model `Barang`, ganti menjadi `Item`, model `Buku` menjadi `Book`, dan `Guru` menjadi `Teacher`. Dengan demikian tidak terasa aneh jika nanti tabel bernama `items`, `books` dan `teachers`.

Solusi kedua adalah mengganti nama tabel default Eloquent dari dalam model. Caranya cukup mudah, yakni dengan menambah sebuah property `$table` ke dalam model. Mari kita coba.

Sebagai bahan praktek, saya akan buat tabel `data_mahasiswa` yang nantinya diakses oleh model `Mahasiswa`. Karena nama tabel tidak standar, maka kita harus membuat file model dan migration secara terpisah:

```
php artisan make:model Mahasiswa  
php artisan make:migration create_data_mahasiswa_table
```



```
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa  
Model created successfully.  
C:\xampp\htdocs\laravel01>php artisan make:migration create_data_mahasiswa_table  
Created Migration: 2020_10_011245_create_data_mahasiswa_table  
C:\xampp\htdocs\laravel01>
```

Gambar: Membuat file Model dan migration

Kita akan mulai dari file migration terlebih dahulu. Berikut struktur tabel `data_mahasiswa` yang sebenarnya masih sama persis seperti tabel `mahasiswa` selama ini:

```
database\migrations\<timestamp>create_data_mahasiswa_table.php
```

```
1 public function up()  
2 {  
3     Schema::create('data_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    });  
11 }
```

Lanjut dengan menjalankan proses migration:

```
php artisan migrate
```

Sampai di sini, tabel `data_mahasiswa` sudah siap untuk diakses. Sekarang buka file model `Mahasiswa.php`, lalu tambah satu property `$table` dengan hak akses `protected`:

```
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 $table = 'data_mahasiswa';  
12 }
```

Di baris 11 saya menginput string `data_mahasiswa` ke dalam property `protected $table`.

Dengan perintah ini, maka setiap kali perintah Eloquent di akses, model Mahasiswa akan terhubung ke tabel `data_mahasiswa`, tidak lagi mencari tabel `mahasiswas`.

Untuk uji coba, test input satu data baru menggunakan Eloquent dari route:

`routes\web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Models\Mahasiswa;
5
6 Route::get('/insert-mahasiswa-1', function () {
7     $mahasiswa = new Mahasiswa;
8     $mahasiswa->nim = '19003036';
9     $mahasiswa->nama = 'Sari Citra Lestari';
10    $mahasiswa->tanggal_lahir = '2001-12-31';
11    $mahasiswa->ipk = 3.62;
12    $mahasiswa->save();
13
14    return "Penambahan mahasiswa berhasil";
15});
```



Gambar: Insert 1 data mahasiswa

A screenshot of a terminal window titled 'MySQL Folder cmd - mysql -u root'. It shows the output of a SQL query: 'MariaDB [laravel]> SELECT * FROM data_mahasiswa;'. The result is a table with one row:

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	19003036	Sari Citra Lestari	2001-12-31	3.62	2020-06-26 06:26:35	2020-06-26 06:26:35

1 row in set (0.002 sec)

Gambar: Isi tabel data_mahasiswa

Sip, tidak ada masalah. Data yang diinput dari route sukses masuk ke tabel `data_mahasiswa`, termasuk jika menjalankan perintah-perintah Eloquent lain.

11.2. Menonaktifkan Timestamp

Selain mencari tabel dengan nama *plural*, perilaku default lain dari Model adalah otomatis meng-update tanggal **timestamp**. Tanggal timestamp yang dimaksud adalah pasangan kolom `update_at` dan `created_at`.

Pada saat kita melakukan proses insert atau update menggunakan Eloquent, kolom `update_at` dan `created_at` otomatis diisi nilai tanggal saat ini. Fitur ini sebenarnya sangat bermanfaat. Namun bagaimana jika karena sesuatu hal kita tidak ingin kedua kolom terisi secara otomatis?

Caranya, tambah property `public $timestamps` dengan nilai `false` ke dalam Model:

app\Models\Mahasiswa.php

```
1 <?php
2 ...
3
4 class Mahasiswa extends Model
5 {
6     use HasFactory;
7     protected $table = 'data_mahasiswa';
8
9     public $timestamps = false;
10 }
```

Sekarang ketika perintah insert atau update di jalankan dari Eloquent, kolom `update_at` dan `created_at` akan berisi nilai `NULL`. Berikut percobaannya:

routes\web.php

```
1 <?php
2 ...
3 Route::get('/insert-mahasiswa-2', function () {
4     $mahasiswa = new Mahasiswa;
5     $mahasiswa->nim = '19021044';
6     $mahasiswa->nama = 'Rudi Permana';
7     $mahasiswa->tanggal_lahir = '2000-08-22';
8     $mahasiswa->ipk = 2.99;
9     $mahasiswa->save();
10
11     return "Penambahan mahasiswa berhasil";
12 });

});
```

The screenshot shows two windows. The top window is a browser displaying the URL `localhost:8000/insert-mahasiswa-2`, with the page content "Penambahan mahasiswa berhasil". The bottom window is a terminal window titled "MySQL Folder cmd - mysql -u root" showing the output of a MySQL query:

```
MariaDB [laravel]> SELECT * FROM data_mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim      | nama        | tanggal_lahir | ipk    | created_at          | updated_at          |
+----+-----+-----+-----+-----+-----+
| 1  | 19003036 | Sari Citra Lestari | 2001-12-31   | 3.62   | 2020-06-26 06:26:35 | 2020-06-26 06:26:35 |
| 2  | 19021044 | Rudi Permana   | 2000-08-22   | 2.99   | NULL                | NULL                |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.002 sec)
```

Gambar: Kolom `created_at` dan `update_at` berisi `NULL`

11.3. Membuat Nilai Default

Fitur lain yang bisa kita tambah ke dalam Model adalah membuat nilai default. Maksudnya, jika pada saat proses insert terdapat kolom yang tidak diisi, akan diambil dari nilai default ini.

Caranya, tambah property `$attributes` ke dalam class model. Isi dari property `$attributes` berupa array pasangan nama kolom dan nilai default yang diinginkan. Berikut contoh penggunaannya:

app\Models\Mahasiswa.php

```

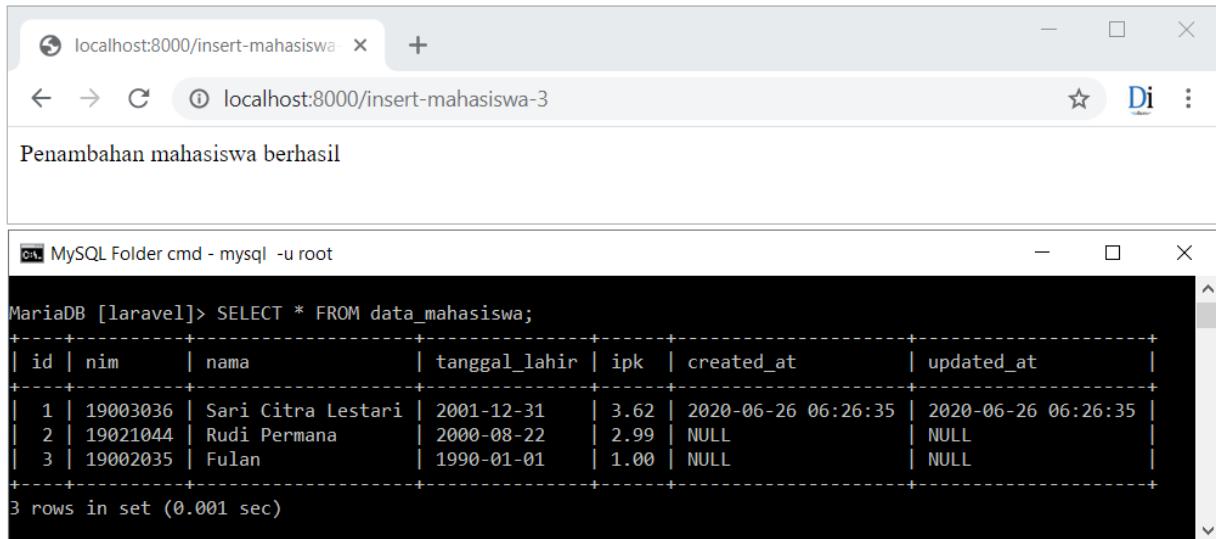
1  <?php
2  ...
3
4  class Mahasiswa extends Model
5  {
6      use HasFactory;
7      protected $table = 'data_mahasiswa';
8
9      public $timestamps = false;
10
11     protected $attributes = [
12         'nama' => 'Fulan',
13         'tanggal_lahir' => '1990-01-01',
14     ];
15 }
```

Di baris 12 – 15 saya mengisi nilai default untuk kolom `nama` dengan string 'Fulan' dan kolom `tanggal_lahir` dengan string `1990-01-01`. Jika nilai kedua kolom tidak diinput pada saat proses insert, nilai inilah yang akan dipakai:

routes\web.php

```

1  <?php
2  ...
3  Route::get('/insert-mahasiswa-3', function () {
4      $mahasiswa = new Mahasiswa;
5      $mahasiswa->nim = '19002035';
6      $mahasiswa->save();
7
8      return "Penambahan mahasiswa berhasil";
9  });
```



Gambar: Kolom nama dan tanggal_lahir menggunakan nilai default

Dalam contoh route ini saya hanya mengisi kolom `nim` saja ke dalam tabel `data_mahasiswa`. Karena terdapat nilai default, kolom `nama` dan `tanggal_lahir` otomatis berisi angka yang berasal dari property `$attributes`. Sedangkan kolom `ipk` sudah memiliki nilai default yang diinput ke dalam struktur tabel pada saat pendefinisian migration.

Jadi, nilai default untuk Eloquent bisa dibuat di sisi PHP menggunakan property `$attributes` atau langsung ke tabel MySQL pada saat pendefinisian migration.

11.4. Menampilkan Data Tabel

Menampilkan data tabel menjadi salah satu materi paling penting dalam Eloquent. Di buku **Laravel Uncover** kita telah praktekkan beberapa method seperti `all()`, `get()`, `first()`, `find()`, `latest()`, serta method *modifier* seperti `where()`, `orderBy()`, `limit()`, `skip()`, `take()` dan `findOrFail()`.

Selain method-method tersebut, masih ada beberapa method lain yang belum sempat di bahas. Kali ini kita akan pelajari dengan lebih detail method apa saja yang bisa dipakai untuk menampilkan data. Ini juga sebagai "materi pemanasan" sebelum masuk ke materi **Eloquent Relationship** pada bab berikutnya.

Sebagai bahan praktek, saya lanjut memakai tabel `data_mahasiswa`. Tabel ini nantinya tetap diakses dari model `Mahasiswa`.

Untuk data *sample*, akan di-generate menggunakan **seeder**. Supaya lebih praktis, saya tidak membuat factory maupun file seeder terpisah, tapi cukup generate langsung dari file master seeder di `DatabaseSeeder.php`. Silahkan buka file ini dan modifikasi sebagai berikut:

database\seeders\DatabaseSeeder.php

```

1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7 use App\Models\Mahasiswa;
8
9 class DatabaseSeeder extends Seeder
10 {
11     public function run()
12     {
13         $faker = Faker::create('id_ID');
14         $faker->seed(123);
15
16         for ($i=0; $i<10; $i++) {
17             Mahasiswa::create(
18                 [
19                     'nim' => $faker->unique()->numerify('10#####'),
20                     'nama' => $faker->firstName() . $faker->lastName(),
21                     'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01', '+ 3 years'),
22                     'ipk' => $faker->randomFloat(2, 2, 4),
23                     'created_at' => $faker->dateTimeBetween('-10 days', '-5 days'),
24                     'updated_at' => $faker->dateTimeBetween('-3 days'),
25                 ]
26             );
27         }
28     }
29 }
```

Kode seeder yang dipakai masih mirip seperti sebelumnya. Sedikit penambahan ada di baris 14 berupa perintah `$faker->seed(123)` agar data yang di generate Faker sama persis dengan contoh dalam buku ini.

Selain itu kolom `created_at` dan `updated_at` juga di input dengan tanggal acak agar data terasa lebih asli (baris 23 dan 24). Kolom `created_at` akan diisi tanggal antara 10 hingga 5 hari sebelum tanggal sekarang. Sedangkan kolom `update_at` akan berisi tanggal antara 3 hari sampai dengan tanggal hari ini.

File model `Mahasiswa` juga perlu sedikit modifikasi, terutama karena kita sudah mematikan fungsi `timestamp`. Berikut tampilan file `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 $table = 'data_mahasiswa';
12
13     //public $timestamps = false;
14
15     protected $attributes = [
16         'nama' => 'Fulan',
17         'tanggal_lahir' => '1990-01-01',
18     ];
19
20     protected $dates = [
21         'tanggal_lahir',
22     ];
23
24     protected $guarded = [];
25 }

```

Property `$timestamps = false` di baris 13 saya non-aktifkan (atau bisa juga dihapus) supaya kolom `created_at` dan `updated_at` bisa diinput dari seeder.

Kemudian terdapat tambahan property `protected $dates` di baris 20 untuk kolom `tanggal_lahir`. Ini diperlukan agar nantinya hasil dari kolom `tanggal_lahir` sudah langsung dalam bentuk Carbon object. Mengenai hal ini sudah pernah kita bahas di akhir bab Carbon.

Terakhir saya menambah property `protected $guarded = []` di baris 24 sebagai persiapan untuk perintah `mass assignment` yang akan digunakan nantinya. Lanjut, jalankan seeder dengan kode berikut:

```
php artisan migrate:fresh --seed
```

Lalu cek ke database apakah data sudah berhasil di generate:

The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM data_mahasiswa;". The output displays 10 rows of data from the table, each containing values for id, nim, nama, tanggal_lahir, ipk, created_at, and updated_at. The columns are separated by vertical lines and have horizontal dashed lines above and below them. The data includes various student names like Mustofa Simanjuntak, Shania Pertiwi, Queen Suryatmi, etc., along with their respective IDs, NIMs, birth dates, GPAs, and timestamps for creation and update.

id	nim	nama	tanggal_lahir	ipk	created_at	updated_at
1	10300234	Mustofa Simanjuntak	1999-12-02	2.90	2020-06-21 10:35:55	2020-06-29 05:19:42
2	10512421	Shania Pertiwi	1999-12-17	3.16	2020-06-22 16:07:21	2020-06-27 11:36:11
3	10605319	Queen Suryatmi	2001-01-08	2.88	2020-06-23 19:38:34	2020-06-29 21:32:15
4	10694755	Kemal Padmasari	2001-06-10	2.88	2020-06-24 04:11:40	2020-06-27 07:33:18
5	10531551	Kambali Mulyani	1999-11-23	2.31	2020-06-22 15:08:09	2020-06-27 21:44:38
6	10262023	Putri Natsir	1999-12-20	2.70	2020-06-21 20:56:30	2020-06-28 22:51:02
7	10228263	Mahdi Rajata	2001-04-19	2.18	2020-06-21 04:00:28	2020-06-28 18:28:03
8	10975558	Eka Hasanah	2000-09-23	2.15	2020-06-23 12:41:08	2020-06-29 17:35:51
9	10426351	Clara Wijaya	2000-06-21	3.00	2020-06-22 09:14:21	2020-06-28 04:51:26
10	10568430	Erik Pudjiastuti	2000-03-30	2.63	2020-06-20 18:04:57	2020-06-27 07:13:01

10 rows in set (0.001 sec)

Gambar: Isi tabel data_mahasiswa

Sip, data untuk tabel `data_mahasiswa` sudah berhasil di generate. Perhatikan nilai kolom `created_at` dan `updated_at` sudah berisi tanggal acak sesuai dengan jangkauan waktu yang di

buat dari Faker.

Menampilkan Eloquent Collection

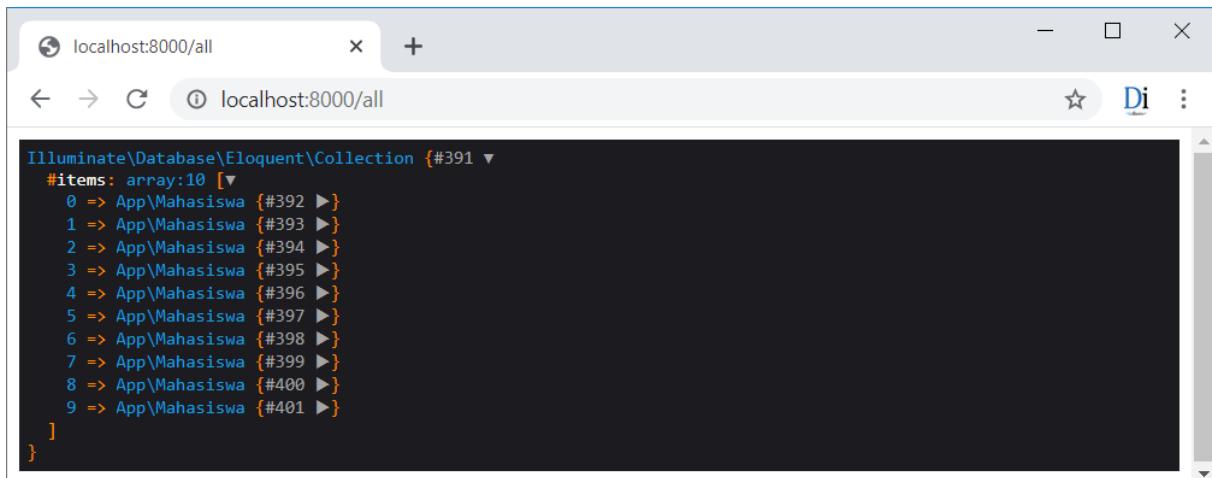
Sebagaimana yang kita ketahui (terutama sepanjang bahasan di buku **Laravel Uncover**), hasil dari method Eloquent bisa berbentuk satu data atau banyak data. Jika perintah Eloquent mengembalikan banyak data, maka hasilnya berbentuk sebuah array, atau lebih tepatnya **collection** dari *model object*.

Berikut percobaannya:

routes\web.php

```
1 <?php
2 ...
3 Route::get('all', function () {
4     $mahasiswa = Mahasiswa::all();
5     dump($mahasiswa);
6 });


```



Gambar: Hasil dump() dari Mahasiswa::all()

Di dalam Laravel, collection bisa disebut sebagai "array on steroid" atau array dengan *superpower*. Selain berperilaku sebagaimana layaknya array, collection juga memiliki banyak method tambahan. Terkait tentang collection, juga sudah kita bahas di buku **Laravel Uncover**.

Setiap element yang ada di dalam collection hasil kode program diatas merupakan sebuah **model object**. Model object sendiri tidak hanya berisi data tabel saja, tapi juga berbagai atribut lain yang kadang tidak kita perlukan.

Dalam contoh di atas, untuk melihat isi satu baris data tabel kita harus klik tanda panah di **items** (1), klik tanda panah di element 0 (2) dan masuk ke tab **attributes** (3):

```

Illuminate\Database\Eloquent\Collection {#391
    #items: array:10 [▼
        0 => App\Mahasiswa {#392 ▼
            #table: "data_mahasiswa"
            #dates: array:1 [▶]
            #attributes: array:7 [▼
                "id" => 1
                "nim" => "10300234"
                "nama" => "Mustofa Simanjuntak"
                "tanggal_lahir" => "1999-12-02"
                "ipk" => "2.90"
                "created_at" => "2020-06-21 10:35:55"
                "updated_at" => "2020-06-29 05:19:42"
            ]
            #guarded: []
            #connection: "mysql"
        }
    ]
}

```

Gambar: Klik tab attributes untuk melihat data kolom

Ini cukup merepotkan, apalagi jika kita hanya ingin melihat semua isi tabel saja (tidak perlu info lain). Trik yang sering saya pakai adalah menjalankan method `toArray()` ke hasil eloquent collection:

`routes\web.php`

```

1 <?php
2 ...
3 Route::get('all', function () {
4     $mahasiswas = Mahasiswa::all();
5     dump($mahasiswas->toArray());
6 });

```

Method `toArray()` akan mengkonversi sebuah collection menjadi array biasa. Tampilan array ini menjadi lebih ringkas:

```

array:10 [▼
    0 => array:7 [▼
        "id" => 1
        "nim" => "10300234"
        "nama" => "Mustofa Simanjuntak"
        "tanggal_lahir" => "1999-12-02T00:00:00.000000Z"
        "ipk" => "2.90"
        "created_at" => "2020-06-21T10:35:55.000000Z"
        "updated_at" => "2020-06-29T05:19:42.000000Z"
    ]
    1 => array:7 [▼
        "id" => 2
        "nim" => "10512421"
        "nama" => "Shania Pertiwi"
        "tanggal_lahir" => "1999-12-17T00:00:00.000000Z"
        "ipk" => "3.16"
    ]
}

```

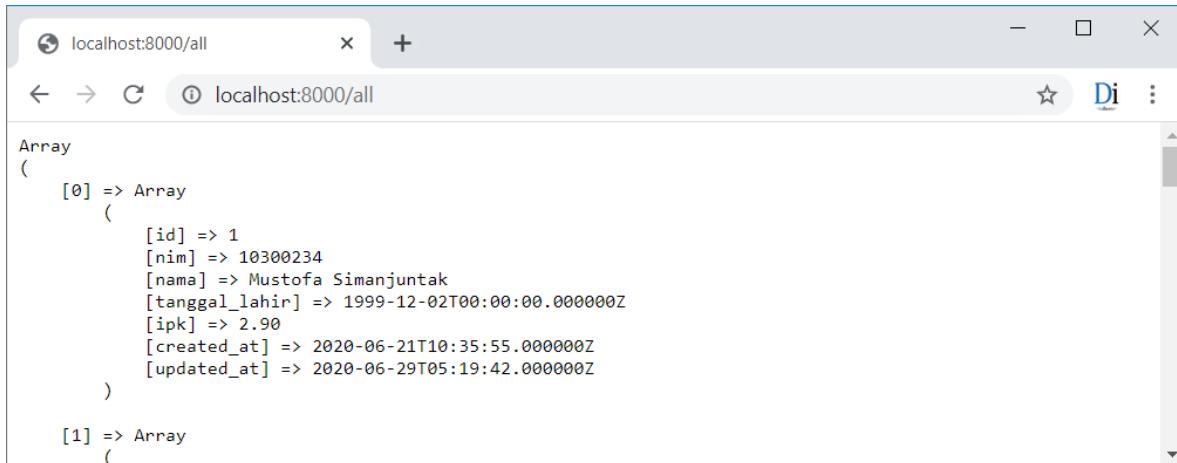
Gambar: Hasil dari method `toArray()`

Sekarang untuk melihat isi setiap baris tabel tinggal men-klik element array. Atau jika ingin menampilkan semua data array, tahan tombol CTRL lalu klik tanda panah di baris paling atas.

Alternatif lain, juga bisa menggunakan cara "tradisional", yakni dengan perintah echo "<pre>" serta perintah print_r() dari hasil method toArray():

routes\web.php

```
1 <?php
2 ...
3 Route::get('all', function () {
4     $mahasiswa = Mahasiswa::all();
5     echo "<pre>";
6     print_r($mahasiswa->toArray());
7});
```



Gambar: Hasil menampilkan array dengan print_r()

Sekarang semua data tabel langsung tampil tanpa perlu klik tombol apapun.

Itulah beberapa cara yang bisa dipakai untuk melihat isi tabel yang diakses dari Eloquent. Namun semua ini hanya cocok untuk proses *debugging* saja. Untuk aplikasi asli, kita tetap harus menampilkannya dari view.

Berikut cara yang akan umum di gunakan untuk mengirim data hasil eloquent ke dalam view:

routes\web.php

```
1 <?php
2 ...
3 Route::get('all', function () {
4     $mahasiswa = Mahasiswa::all();
5     return view('mahasiswa', compact('mahasiswa'));
6});
```

Route di atas akan memanggil file view bernama `mahasiswa.blade.php` dengan mengirim data `$mahasiswa` yang sudah dalam bentuk collection. Berikut kode file view tersebut:

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 <div class="container mt-3">
13   <div class="row">
14     <div class="col-12">
15
16       <div class="py-4">
17         <h2>Data Mahasiswa</h2>
18       </div>
19
20       <table class="table table-striped">
21         <thead>
22           <tr>
23             <th>ID</th>
24             <th>NIM</th>
25             <th>Nama</th>
26             <th>Tanggal Lahir</th>
27             <th>IPK</th>
28             <th>Created At</th>
29             <th>Updated At</th>
30           </tr>
31         </thead>
32         <tbody>
33           @forelse ($mahasiswas as $mahasiswa)
34             <tr>
35               <th>{{$mahasiswa->id}}</th>
36               <td>{{$mahasiswa->nim}}</td>
37               <td>{{$mahasiswa->nama}}</td>
38               <td>{{$mahasiswa->tanggal_lahir->isoFormat('DD-MM-Y')}}</td>
39               <td>{{$mahasiswa->ipk}}</td>
40               <td>{{$mahasiswa->created_at->isoFormat('DD-MM-Y')}}</td>
41               <td>{{$mahasiswa->updated_at->isoFormat('DD-MM-Y')}}</td>
42             </tr>
43           @empty
44             <td colspan="7" class="text-center">Tidak ada data...</td>
45           @endforelse
46         </tbody>
47       </table>
48     </div>
49   </div>
50
51 </div>
52
53 </body>
54 </html>
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020

Gambar: Tampilan isi tabel data mahasiswa di view

Isi file view `mahasiswas.blade.php` mirip seperti contoh kita sebelumnya (bab pagination), dimana saya menampilkan isi tabel menggunakan perulangan `@forelse` antara baris 33 – 45. Khusus untuk tipe data tanggal seperti `tanggal_lahir`, `created_at` dan `updated_at`, ditampilkan dengan bantuan method `isoFormat('DD-MM-Y')` bawaan Carbon.

Karena dalam kode ini saya mengakses file `/css/bootstrap.min.css`, maka silahkan copy file CSS Bootstrap ke folder `public`. File Bootstrap ini bisa di download langsung dari web resminya atau bisa juga ambil dari Google Drive.

Menampilkan 1 Data Eloquent

Tidak semua hasil eloquent berbentuk collection. Beberapa perintah hanya mengembalikan 1 data saja seperti contoh berikut:

```
routes\web.php
```

```
1 Route::get('first', function () {
2     $mahasiswa = Mahasiswa::first();
3     dump($mahasiswa);
4 });
```

```
App\Mahasiswa {#386 ▾
  #table: "data_mahasiswa"
  #dates: array:1 [▶]
  #attributes: array:7 [▼
    "id" => 1
    "nim" => "10300234"
    "nama" => "Mustofa Simanjuntak"
    "tanggal_lahir" => "1999-12-02"
    "ipk" => "2.90"
    "created_at" => "2020-06-21 10:35:55"
    "updated_at" => "2020-06-29 05:19:42"
  ]}
```

Gambar: Hasil dump() dari Mahasiswa::first()

Method `Mahasiswa::first()` mengembalikan 1 data dari tabel `data_mahasiswa` yang ada di urutan pertama. Karena tidak terdapat tambahan modifier lain seperti `orderBy()`, maka yang diambil adalah data mahasiswa dengan kolom `id = 1`.

Dari hasil `dump()` terlihat method `Mahasiswa::first()` mengembalikan data bukan berbentuk collection, tapi berupa satu object model `App\Models\Mahasiswa`. Akibatnya, kita tidak perlu melakukan perulangan atau looping di dalam file view (malah akan error jika tetap dilakukan). Oleh karena itu saya perlu membuat file view terpisah. Berikut route yang diperlukan:

`routes\web.php`

```
1 Route::get('first', function () {
2     $mahasiswa = Mahasiswa::first();
3     return view('mahasiswa', compact('mahasiswa'));
4 });
```

Route ini memanggil file view bernama `mahasiswa.blade.php` dengan mengirim data `$mahasiswa` yang berisi 1 data mahasiswa. Berikut kode untuk file view tersebut:

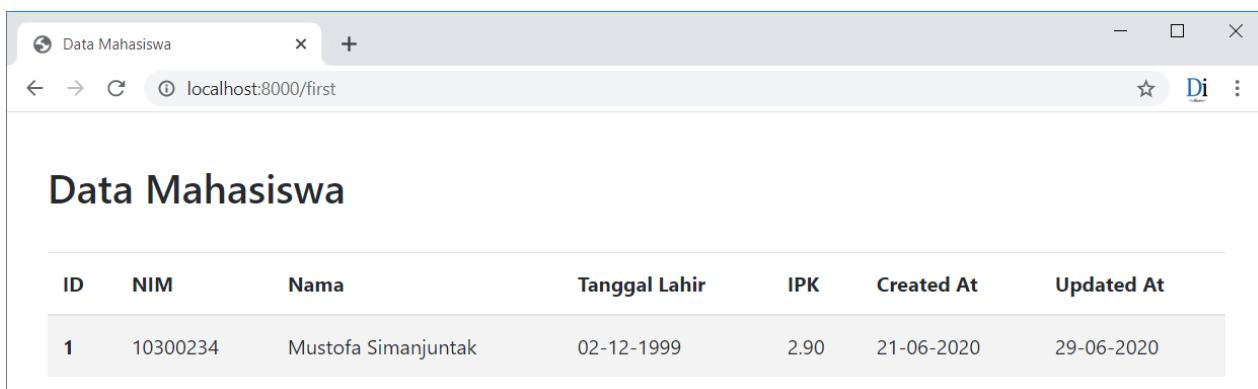
`resources\views\mahasiswas.blade.php`

```
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>Data Mahasiswa</h2>
18             </div>
19
20             <table class="table table-striped">
21                 <thead>
22                     <tr>
23                         <th>ID</th>
24                         <th>NIM</th>
25                         <th>Nama</th>
26                         <th>Tanggal Lahir</th>
27                         <th>IPK</th>
28                         <th>Created At</th>
29                         <th>Updated At</th>
30                     </tr>
31             </thead>
```

```

32     <tbody>
33         @isset($mahasiswa)
34             <tr>
35                 <th>{{$mahasiswa->id}}</th>
36                 <td>{{$mahasiswa->nim}}</td>
37                 <td>{{$mahasiswa->nama}}</td>
38                 <td>{{$mahasiswa->tanggal_lahir->isoFormat('DD-MM-Y')}}</td>
39                 <td>{{$mahasiswa->ipk}}</td>
40                 <td>{{$mahasiswa->created_at->isoFormat('DD-MM-Y')}}</td>
41                 <td>{{$mahasiswa->updated_at->isoFormat('DD-MM-Y')}}</td>
42             </tr>
43         @else
44             <td colspan="7" class="text-center">Tidak ada data...</td>
45         @endisset
46     </tbody>
47 </table>
48 </div>
49 </div>
50
51 </div>
52
53 </body>
54 </html>

```



The screenshot shows a web browser window with the title 'Data Mahasiswa'. The URL in the address bar is 'localhost:8000/first'. The page content is a table with the following data:

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020

Gambar: Tampilan isi tabel data mahasiswa di view

Kode ini pada dasarnya sama seperti sebelumnya, hanya saja sekarang tidak menggunakan perulangan `@forelse`, tapi hanya `@isset($mahasiswa)` untuk memeriksa apakah variabel `$mahasiswa` berisi sesuatu atau tidak. Jika ada, maka tampilkan data untuk 1 baris tabel tersebut.

Sampai di sini kita memiliki 2 buah file view: `mahasiswas.blade.php` dan `mahasiswa.blade.php`. Nama kedua file ini memang sangat mirip, hanya dibedakan dengan tambahan 's'. Ini sesuai dengan fungsinya, dimana jika data hasil eloquent berupa collection (jamak/plural), maka kirim ke `mahasiswas.blade.php`. Tapi jika data tersebut hanya terdiri dari 1 data saja (tunggal/singular), maka kirim ke `mahasiswa.blade.php`.

Lanjut, kita akan bahas satu per satu method Eloquent untuk menampilkan data tabel.

Dalam dokumentasi Laravel, method-method ini berada di beberapa tempat, antara lain di bagian [Eloquent: Getting Started](#), [Eloquent: Collections](#) serta [Database: Query Builder](#). Silahkan buka dokumentasi ini juga anda butuh referensi lebih lanjut.

Method all() dan get()

Kedua method ini sudah sangat sering kita pakai. Jika dijalankan tanpa tambahan method lain, keduanya akan menampilkan seluruh kolom untuk semua baris data yang ada di tabel:

routes\web.php

```

1 Route::get('all', function () {
2     $mahasiswa = Mahasiswa::all();
3     return view('mahasiswa', compact('mahasiswa'));
4 });
5
6 Route::get('get', function () {
7     $mahasiswa = Mahasiswa::get();
8     return view('mahasiswa', compact('mahasiswa'));
9 });

```



The screenshot shows a browser window with the title 'Data Mahasiswa'. The address bar displays 'localhost:8000/get'. The page content is a table with the following data:

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020

Gambar: Hasil method Mahasiswa::get()

Hasil dari method all() dan get() berbentuk collection yang saya kirim ke view mahasiswa.blade.php. Kedua method ini juga bisa menerima argument berupa nama kolom yang ingin diambil:

routes\web.php

```

1 Route::get('all-column', function () {
2     $result = Mahasiswa::all(['nama', 'ipk']);
3     dump($result->toArray());
4 });
5
6 Route::get('get-column', function () {
7     $result = Mahasiswa::get(['nama', 'ipk']);

```

```
8     dump($result->toArray());
9 );
```

```
array:10 [▼
  0 => array:2 [▼
    "nama" => "Mustofa Simanjuntak"
    "ipk" => "2.90"
  ]
  1 => array:2 [▼
    "nama" => "Shania Pertwiwi"
    "ipk" => "3.16"
  ]
  2 => array:2 [▼
    "nama" => "Queen Suryatmi"
    "ipk" => "2.88"
  ]
  3 => array:2 [▼
```

Gambar: Hasil method Mahasiswa::all(['nama','ipk'])

Di baris 2 dan 7 saya menginput array ['nama', 'ipk'] ke dalam method `get()` dan `all()`. Hasilnya, collection hanya berisi 2 kolom saja, yakni kolom `nama` dan `ipk`. Karena tidak berisi data untuk kolom lain, hasil ini tidak bisa dikirim ke file view `mahasiswas.blade.php`.

Perbedaan mendasar antara `get()` dan `all()` adalah, method `get()` bisa dipakai untuk mengakses hasil method lain seperti `where()`, `orderB()`, dll. Sedangkan `all()` tidak bisa.

Lebih teknis lagi, method `get()` adalah milik **Eloquent\Builder** object, sedangkan method `all()` merupakan milik **Eloquent\Model** object. Secara internal, di dalam pendefinisian method `all()` terdapat pemanggilan ke method `get()`.

Method select()

Method `select()` bisa dipakai untuk memilih kolom apa saja yang ingin ditampilkan. Nama kolom diinput sebagai argument seperti contoh berikut:

routes\web.php

```
1 Route::get('select', function () {
2     $result = Mahasiswa::select('nama', 'ipk', 'nim')->get();
3     dump($result->toArray());
4});
```

```
array:10 [▼
  0 => array:3 [▼
    "nama" => "Mustofa Simanjuntak"
    "ipk" => "2.90"
    "nim" => "10300234"
  ]
  1 => array:3 [▼
    "nama" => "Shania Pertwiwi"
    "ipk" => "3.16"
    "nim" => "10512421"
  ]
```

Gambar: Hasil method Mahasiswa::select('nama','ipk','nim')->get()

Hasil akhir dari method `select()` berbentuk collection. Hasil yang sama sebenarnya juga bisa di dapat dengan menjalankan method `Mahasiswa::get(['nama', 'ipk', 'nim'])` atau `Mahasiswa::all(['nama', 'ipk', 'nim'])`.

Method `orderBy()`, `orderByDesc()`, dan `inRandomOrder()`

Method `orderBy()` berfungsi sama seperti query `ORDER BY` di SQL, yakni untuk mengurutkan baris tabel berdasarkan kolom tertentu. Nama kolom yang dipakai sebagai patokan pengurutan diinput sebagai argument:

`routes\web.php`

```
1 Route::get('order-by-1', function () {
2     $mahasiswas = Mahasiswa::orderBy('ipk')->get();
3     return view('mahasiswas', compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
8	10975558	Eka Hasanah	23-09-2000	2.15	23-06-2020	29-06-2020
7	10228263	Mahdi Rajata	19-04-2001	2.18	21-06-2020	28-06-2020
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020

Gambar: Hasil method `Mahasiswa::orderBy('ipk')->get()`

Secara default, method `orderBy()` terurut menaik, yakni dari nilai paling rendah ke nilai yang paling tinggi.

Jika kita ingin menampilkan data secara menurun, yakni dari nilai tinggi ke rendah, tambah string `'desc'` sebagai argument kedua dari method `orderBy()`:

`routes\web.php`

```
1 Route::get('order-by-2', function () {
2     $mahasiswas = Mahasiswa::orderBy('ipk', 'desc')->get();
3     return view('mahasiswas', compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020

Gambar: Hasil method Mahasiswa::orderBy('ipk','desc')->get()

Alternatif penulisan bisa juga dengan method `orderByDesc()`, seperti contoh berikut:

routes\web.php

```
1 Route::get('order-by-desc', function () {
2     $mahasiswas = Mahasiswa::orderByDesc('ipk')->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
```

Hasil yang di dapat akan sama seperti `orderBy('ipk','desc')`, dimana data tabel akan diurutkan secara menurun berdasarkan kolom `ipk`.

Masih tentang pengurutan, Laravel juga menyediakan method yang cukup unik, yakni `inRandomOrder()`. Method ini akan mengacak urutan data tabel dalam setiap pemanggilan:

routes\web.php

```
1 Route::get('in-random-order', function () {
2     $mahasiswas = Mahasiswa::inRandomOrder()->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
```

Hasilnya, urutan data tabel `data_mahasiswa` akan terus berganti-ganti setiap halaman di load.

Method `where()`

Method `where()` mewakili query `WHERE` di dalam SQL. Fungsinya untuk membatasi tampilan data tabel berdasarkan kondisi tertentu. Kondisi `WHERE` ini sangat beragam, oleh karena itu pula Laravel menyediakan banyak variasi dari method `where()`.

Dalam bentuk paling sederhana, method `where()` bisa diisi dengan 2 buah argument berupa nama kolom sebagai argument pertama, dan nilai yang akan dicari sebagai argument kedua:

routes\web.php

```
1 Route::get('where-1', function () {
2     $mahasiswa = Mahasiswa::where('ipk', 3)->get();
3     return view('mahasiswa', compact('mahasiswa'));
4});
```

The screenshot shows a browser window with the title 'Data Mahasiswa'. The URL in the address bar is 'localhost:8000/where-1'. The page displays a table with the following columns: ID, NIM, Nama, Tanggal Lahir, IPK, Created At, and Updated At. There is one row of data:

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020

Gambar: Hasil method Mahasiswa::where('ipk',3)->get()

Method `Mahasiswa::where('ipk', 3)->get()` mengembalikan collection dari semua mahasiswa dengan ipk 3.

Jika kita ingin membuat kondisi yang bukan "sama dengan", bisa menjalankan method `where()` dengan 3 argument. Dalam format ini, argument pertama tetap diisi nama kolom, argument kedua diisi dengan operator perbandingan, serta argument ketiga diisi nilai yang dicari:

routes\web.php

```
1 Route::get('where-2', function () {
2     $mahasiswa = Mahasiswa::where('ipk', '>=', 3)->get();
3     return view('mahasiswa', compact('mahasiswa'));
4});
```

The screenshot shows a browser window with the title 'Data Mahasiswa'. The URL in the address bar is 'localhost:8000/where-2'. The page displays a table with the following columns: ID, NIM, Nama, Tanggal Lahir, IPK, Created At, and Updated At. There are two rows of data:

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020

Gambar: Hasil method Mahasiswa::where('ipk','>=',3)->get()

Method `Mahasiswa::where('ipk', '>=', 3)->get()` mengembalikan collection dari semua mahasiswa dengan ipk lebih atau sama dengan 3.

Operator yang digunakan juga termasuk query SQL lain seperti `LIKE`:

routes\web.php

```
1 Route::get('where-3', function () {
2     $mahasiswa = Mahasiswa::where('nama', 'LIKE', '%i')->get();
3     return view('mahasiswa', compact('mahasiswa'));
4});
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020
10	10568430	Erik Pudjiastuti	30-03-2000	2.63	20-06-2020	27-06-2020

Gambar: Hasil method Mahasiswa::where('nama','LIKE','%i')->get()

Method `Mahasiswa::where('nama', 'LIKE', '%i')->get()` mengembalikan collection dari semua mahasiswa dengan nilai kolom nama yang diakhiri dengan huruf i.

Hasil dari method `where()` juga bisa digabung dengan method lain seperti `orderBy()`:

routes\web.php

```
1 Route::get('where-order', function () {
2     $mahasiswa = Mahasiswa::where('nama', 'LIKE', '%i')
3         ->orderBy('tanggal_lahir', 'desc')->get();
4     return view('mahasiswa', compact('mahasiswa'));
5});
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
10	10568430	Erik Pudjiastuti	30-03-2000	2.63	20-06-2020	27-06-2020
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020

Gambar: Hasil method Mahasiswa::where('nama','LIKE','%i')->orderBy('tanggal_lahir','desc')->get()

Method `Mahasiswa::where('nama','LIKE','%i')->orderBy('tanggal_lahir','desc')->get()` mengembalikan collection dari semua mahasiswa dengan nilai kolom nama yang diakhiri dengan huruf i, kemudian diurutkan berdasarkan kolom `tanggal_lahir` secara menurun.

Eloquent juga mendukung penggabungan beberapa kondisi WHERE. Misalnya kita ingin menampilkan semua mahasiswa yang namanya diakhiri dengan huruf i, **dan** memiliki ipk di atas 2.7, bisa menggunakan kode berikut:

routes\web.php

```
1 Route::get('and-where-1', function () {
2     $mahasiswas = Mahasiswa::where('nama','LIKE','%i')
3         ->where('ipk','>',2.7)->get();
4     return view('mahasiswas',compact('mahasiswas'));
5});
```

Di sini saya men-chaining atau menyambung penulisan 2 method where, yakni `Mahasiswa::where('nama','LIKE','%i')->where('ipk','>',2.7)->get()`.

Alternatif penulisan lain adalah dengan menginput sebuah array ke dalam argument method `where()`. Setiap element array mewakili sebuah kondisi WHERE. Query di atas juga bisa ditulis sebagai berikut:

routes\web.php

```
1 Route::get('and-where-2', function () {
2     $mahasiswas = Mahasiswa::where([
3         ['nama','LIKE','%i'],
4         ['ipk','>',2.7]
5     ])->get();
6     return view('mahasiswas',compact('mahasiswas'));
7});
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020

Gambar: Hasil method Mahasiswa::where(['nama'.'LIKE'.'%i'], ['ipk'.'>'.'2.7'])->get()

Di baris 3 dan 4 saya menulis dua buah array untuk kondisi WHERE. Kondisi ini sama seperti sebelumnya, yakni untuk mencari semua mahasiswa yang namanya diakhiri dengan huruf i, **dan** memiliki ipk di atas 2.7. Jika butuh tambahan kondisi lain, tinggal tambah element array berikutnya.

Yang perlu diperhatikan, kondisi ini merupakan sebuah nested array. Di akhir baris 2 dan di awal baris 5 terdapat satu lagi tanda kurung siku.

Method orWhere()

Selanjutnya, bagaimana jika yang diinginkan adalah gabungan 2 buah query WHERE tetapi dengan logika **or?** Laravel menyediakan method orWhere() untuk keperluan ini:

routes\web.php

```

1 Route::get('or-where', function () {
2     $mahasiswas = Mahasiswa::where('nama', 'LIKE', '%i')
3         ->orWhere('ipk', '>', 2.7)->get();
4     return view('mahasiswas', compact('mahasiswas'));
5 });

```

Kode program di atas akan mencari semua mahasiswa yang namanya diakhiri dengan huruf i **atau** memiliki ipk di atas 2.7. Dengan logika ini, meskipun nama mahasiswa tidak diakhiri dengan huruf i, tetap bisa tampil jika mahasiswa tersebut memiliki ipk di atas 2.7 (cukup memenuhi salah satu syarat saja).

Kita juga bisa menggabung 2 kondisi WHERE atau lebih seperti contoh berikut:

routes\web.php

```

1 Route::get('where-chaining-1', function () {
2     $mahasiswas = Mahasiswa::where('nama', 'LIKE', '%i')
3         ->orWhere('ipk', '>', 2.7)
4         ->Where('ipk', '<', 3.0)

```

```

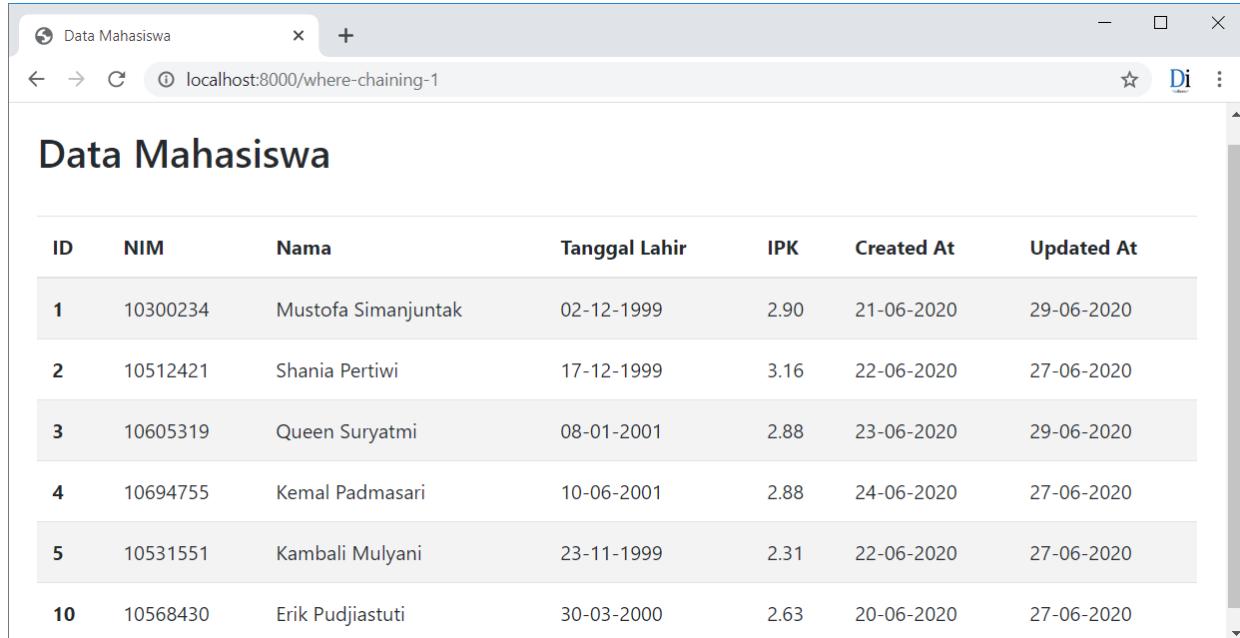
5           ->get();
6   return view('mahasiswa',compact('mahasiswa'));
7 });

```

Kali ini saya menjalankan 3 buah kondisi WHERE untuk mencari semua mahasiswa yang namanya diakhiri dengan huruf i **atau** memiliki ipk di atas 2.7 **dan** ipk di bawah 3.0.

Perintah ini sama artinya dengan query MySQL berikut:

```
SELECT * FROM data_mahasiswa WHERE nama LIKE '%i' OR ipk > 2.7 AND ipk < 3.0;
```



ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020
10	10568430	Erik Pudjiastuti	30-03-2000	2.63	20-06-2020	27-06-2020

Gambar: Hasil method Mahasiswa::where('nama','LIKE','%i')->orWhere('ipk','>',2.7)->Where('ipk','<',3.0)->get()

Eloquent juga menyediakan alternatif penulisan yang sekilas tampak lebih rumit tapi cocok untuk query WHERE yang kompleks. Caranya adalah dengan menulis sebuah closure atau *anonymous function* sebagai argument dari method `where()` atau method `orWhere()`.

Berikut contoh penggunaannya:

routes\web.php

```

1 Route::get('where-chaining-2', function () {
2     $mahasiswa = Mahasiswa::where('nama','LIKE','%i')
3             ->orWhere(function($query) {
4                 $query->where('ipk','>',2.7)->where('ipk','<',3.0);
5             }
6         )->get();
7     return view('mahasiswa',compact('mahasiswa'));
8 });

```

Di baris 3-5, saya mengisi sebuah function ke dalam argument method `orWhere()`. Function ini menerima 1 buah argument bernama `$query`. Di baris 4, variabel `$query` akan dipakai untuk menyambung penulisan kondisi WHERE.

Perintah di atas sama artinya dengan query MySQL berikut:

```
SELECT * FROM data_mahasiswa WHERE nama LIKE '%i' OR (ipk > 2.7 AND ipk < 3.0);
```

Hasilnya sama persis seperti contoh kita sebelumnya, hanya saja sekarang terdapat tambahan tanda kurung di antara `OR (ipk > 2.7 AND ipk < 3.0)` karena function closure di baris 3-5 menjadi sebuah kesatuan.

Method `whereBetween()` dan `whereNotBetween()`

Kedua method ini mewakili query `BETWEEN` dan `NOT BETWEEN` milik SQL. Keduanya dipakai untuk membatasi tampilan data berdasarkan sebuah *range* atau jangkauan tertentu.

Method `whereBetween()` dan `whereNotBetween()` butuh 2 buah argument. Argument pertama diisi dengan nama kolom yang akan dibatasi, serta argument kedua diisi dengan 2 element array yang menentukan awal dan akhir jangkauan.

Berikut contoh penggunaan dari method `whereBetween()`:

routes\web.php

```
1 Route::get('where-between', function () {
2     $mahasiswas = Mahasiswa::whereBetween('ipk',[2.7,3.0])->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020
6	10262023	Putri Natsir	20-12-1999	2.70	21-06-2020	28-06-2020
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020

Gambar: Hasil method `Mahasiswa::whereBetween('ipk',[2.7,3.0])->get()`

Method `Mahasiswa::whereBetween('ipk',[2.7,3.0])->get()` akan mencari semua mahasiswa yang memiliki ipk antara 2.7 sampai 3.0.

Hasil yang sama juga bisa didapat dari query MySQL berikut:

```
SELECT * FROM data_mahasiswa WHERE ipk BETWEEN 2.7 AND 3.0;
```

Sedangkan method `whereNotBetween()` dipakai untuk men-negasi-kan atau membalik logika `WHERE BETWEEN`. Jika saya ingin menampilkan semua data mahasiswa dengan ipk **selain** 2.7 sampai 3.0, bisa menggunakan perintah berikut:

`routes\web.php`

```
1 Route::get('where-not-between', function () {
2     $mahasiswas = Mahasiswa::whereNotBetween('ipk',[2.7,3.0])->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020
7	10228263	Mahdi Rajata	19-04-2001	2.18	21-06-2020	28-06-2020
8	10975558	Eka Hasanah	23-09-2000	2.15	23-06-2020	29-06-2020
10	10568430	Erik Pudjiastuti	30-03-2000	2.63	20-06-2020	27-06-2020

Gambar: Hasil method `Mahasiswa::whereNotBetween('ipk',[2.7,3.0])->get()`

Jika menggunakan MySQL client, hasil yang sama bisa didapat dengan query berikut:

```
SELECT * FROM data_mahasiswa WHERE ipk NOT BETWEEN 2.7 AND 3.0;
```

Method `whereIn()` dan `whereNotIn()`

Method `whereIn()` dan `whereNotIn()` dipakai untuk mencari data berdasarkan beberapa nilai tertentu yang persis sama. Kedua method ini butuh 2 buah argument, yakni nama kolom sebagai argument pertama, serta array yang berisi element yang akan dicari.

Berikut contoh penggunaan method `whereIn()`:

`routes\web.php`

```
1 Route::get('where-in', function () {
2     $mahasiswas = Mahasiswa::whereIn('ipk',[2.7,3.0])->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
6	10262023	Putri Natsir	20-12-1999	2.70	21-06-2020	28-06-2020
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020

Gambar: Hasil method Mahasiswa::whereIn('ipk',[2.7,3.0])->get()

Method `Mahasiswa::whereIn('ipk',[2.7,3.0])->get()` akan menampilkan seluruh data mahasiswa dengan ipk 2.7 dan 3.0. Mahasiswa yang tidak memiliki ipk persis seperti ini tidak akan ditampilkan.

Hasil yang sama juga bisa didapat dengan query MySQL berikut:

```
SELECT * FROM data_mahasiswa WHERE ipk IN (2.7,3.0);
```

Sedangkan untuk method `whereNotIn()`, itu akan membalik logika WHERE IN. Sebagai contoh, jika saya ingin menampilkan data mahasiswa **selain** id 1, 2, 3, 4 dan 5, bisa menggunakan perintah berikut:

routes\web.php

```
1 Route::get('where-not-in', function () {
2     $mahasiswas = Mahasiswa::whereNotIn('id',[1,2,3,4,5])->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
6	10262023	Putri Natsir	20-12-1999	2.70	21-06-2020	28-06-2020
7	10228263	Mahdi Rajata	19-04-2001	2.18	21-06-2020	28-06-2020
8	10975558	Eka Hasanah	23-09-2000	2.15	23-06-2020	29-06-2020
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020
10	10568430	Erik Pudjiastuti	30-03-2000	2.63	20-06-2020	27-06-2020

Gambar: Hasil method Mahasiswa::whereIn('ipk',[2.7,3.0])->get()

Jika menggunakan MySQL client, hasil yang sama bisa didapat dengan query berikut:

```
SELECT * FROM data_mahasiswa WHERE id NOT IN (1,2,3,4,5);
```

Kita juga bisa menggabung beberapa pemanggilan method `whereIn()` dan `whereNotIn()` sekaligus:

routes\web.php

```
1 Route::get('where-in-chaining-1', function () {
2     $mahasiswas = Mahasiswa::whereIn('ipk',[2.7,3.0])
3         ->whereNotIn('id',[7,8,9])->get() ;
4     return view('mahasiswas',compact('mahasiswas'));
5 });
```

Perintah di atas akan menampilkan seluruh data mahasiswa yang memiliki ipk 2.7 serta 3.0 **dan** tidak memiliki id 7, 8. serta 9.

Jika ingin menggabung kondisi dengan logika **or**, Laravel menyediakan method `orWhereIn()` dan `orWhereNotIn()`. Misalnya jika ingin menampilkan seluruh data mahasiswa yang memiliki ipk 2.7 serta 3.0 **atau** tidak memiliki id 7, 8. serta 9, maka perintahnya adalah:

routes\web.php

```
1 Route::get('where-in-chaining-2', function () {
2     $mahasiswas = Mahasiswa::whereIn('ipk',[2.7,3.0])
3         ->orWhereNotIn('id',[7,8,9])->get() ;
4     return view('mahasiswas',compact('mahasiswas'));
5 });
```

Jika menggunakan MySQL client, hasil yang sama bisa didapat dengan query berikut:

```
SELECT * FROM data_mahasiswa WHERE ipk IN (2.7,3.0) OR id NOT IN (7,8,9);
```

Method `whereNull()` dan `whereNotNull()`

Method `whereNull()` dan `whereNotNull()` dipakai untuk menampilkan data berdasarkan apakah terdapat kolom yang bernilai NULL atau tidak. Kedua method ini butuh 1 argument berupa nama kolom yang ingin diperiksa.

Berikut contoh penggunaannya:

routes\web.php

```
1 Route::get('where-null', function () {
2     $mahasiswas = Mahasiswa::whereNull('tanggal_lahir')->get();
3     return view('mahasiswas',compact('mahasiswas'));
4 });
5
6 Route::get('where-not-null', function () {
7     $mahasiswas = Mahasiswa::whereNotNull('tanggal_lahir')->get();
```

```
8     return view('mahasiswa', compact('mahasiswa'));
9 );
```

Hasil method `Mahasiswa::whereNull('tanggal_lahir')->get()` tidak akan menampilkan data apa pun karena kita tidak memiliki nilai NULL untuk kolom `tanggal_lahir`.

Sedangkan hasil dari method `Mahasiswa::whereNotNull('tanggal_lahir')->get()` akan menampilkan seluruh baris tabel karena semuanya memenuhi syarat `whereNotNull()`.

Jika menggunakan MySQL, perintah di atas sama artinya dengan query berikut:

```
SELECT * FROM data_mahasiswa WHERE tanggal_lahir IS NULL;
SELECT * FROM data_mahasiswa WHERE tanggal_lahir IS NOT NULL;
```

Laravel juga menyediakan method `orWhereNull()` serta `orWhereNotNull()` untuk membuat kondisi WHERE gabungan (*chaining*) dengan logika `or`.

Method `whereDate()`, `whereMonth()`, `whereDay()`, `whereYear()` dan `whereTime()`

Sesuai dengan namanya, kelima method ini dipakai untuk proses seleksi kolom bertipe tanggal dan waktu (DATE, TIME atau DATETIME).

Sama seperti penulisan method `where()`, kelima method bisa diisi dengan 2 atau 3 argument. Jika kita ingin mencari data tanggal yang sama persis (memakai operator perbandingan 'sama dengan') maka cukup menggunakan 2 argument. Atau jika ingin menulis manual operator perbandingan, bisa dijalankan dengan 3 argument.

Berikut contoh penggunaan kelima method ini:

`routes\web.php`

```
1 Route::get('where-date', function () {
2     $mahasiswa = Mahasiswa::whereDate('tanggal_lahir', '2000-09-23')->get();
3     return view('mahasiswa', compact('mahasiswa'));
4 });
5
6 Route::get('where-month', function () {
7     $mahasiswa = Mahasiswa::whereMonth('tanggal_lahir', '12')->get();
8     return view('mahasiswa', compact('mahasiswa'));
9 });
10
11 Route::get('where-day', function () {
12     $mahasiswa = Mahasiswa::whereDay('tanggal_lahir', '>', '20')->get();
13     return view('mahasiswa', compact('mahasiswa'));
14 });
15
16 Route::get('where-year', function () {
17     $mahasiswa = Mahasiswa::whereYear('tanggal_lahir', '2000')->get();
18     return view('mahasiswa', compact('mahasiswa'));
19 });
```

```

20
21 Route::get('where-time', function () {
22     $mahasiswa = Mahasiswa::whereTime('created_at', '>', '20:00:00')->get();
23     return view('mahasiswa', compact('mahasiswa'));
24 });

```

Karena kita sudah cukup sering menggunakan variasi method `where()`, saya yakin anda bisa memahami perintah yang ada.

Jika menggunakan MySQL client, hasil yang sama bisa didapat dari query berikut:

```

SELECT * FROM data_mahasiswa WHERE tanggal_lahir = '2000-09-23';
SELECT * FROM data_mahasiswa WHERE MONTH(tanggal_lahir) = '12';
SELECT * FROM data_mahasiswa WHERE DAY(tanggal_lahir) > '20';
SELECT * FROM data_mahasiswa WHERE YEAR(tanggal_lahir) = '2000';
SELECT * FROM data_mahasiswa WHERE TIME(created_at) > '20:00:00';

```

Method limit()

Method `limit()` dipakai untuk membatasi data yang tampil berdasarkan urutan baris. Method ini butuh 1 argument untuk menentukan jumlah data yang akan diambil. Biasanya method `limit()` dipadukan dengan `orderBy()` sebagai penentu urutan baris.

Berikut contoh penggunaannya:

`routes\web.php`

```

1 Route::get('limit', function () {
2     $mahasiswa = Mahasiswa::orderBy('ipk', 'desc')->limit(3)->get();
3     return view('mahasiswa', compact('mahasiswa'));
4 });

```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
9	10426351	Clara Wijaya	21-06-2000	3.00	22-06-2020	28-06-2020
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020

Gambar: Hasil method `Mahasiswa::orderBy('ipk','desc')->limit(3)->get()`

Method `Mahasiswa::orderBy('ipk', 'desc')->limit(3)->get()` akan menampilkan 3 mahasiswa teratas dengan ipk paling tinggi.

Method skip() dan take()

Kedua method ini juga berfungsi untuk mengambil sebagian data berdasarkan urutan baris. Method `skip()` dipakai untuk melompati baris, sedangkan `take()` untuk mengambil baris. Masing-masing method butuh satu argument berupa jumlah baris yang dilompati atau diambil.

Berikut contoh penggunaannya:

`routes\web.php`

```
1 Route::get('skip-take', function () {
2     $mahasiswa = Mahasiswa::orderBy('ipk','desc')->skip(2)->take(3)->get();
3     return view('mahasiswa',compact('mahasiswa'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020

Gambar: Hasil method `Mahasiswa::orderBy('ipk','desc')->skip(2)->take(3)->get()`

Method `Mahasiswa::orderBy('ipk','desc')->skip(2)->take(3)->get()` akan mengurutkan data mahasiswa berdasarkan kolom `ipk` dari paling tinggi ke paling rendah, kemudian lewati 2 baris pertama, lalu ambil 3 baris berikutnya.

Atau bisa juga disimpulkan bahwa data yang tampil adalah mahasiswa dengan ipk tertinggi di urutan 3 – 5.

Method first() dan firstWhere()

Fungsi dari method `first()` cukup sederhana, yakni untuk mengambil 1 data yang berada di urutan paling atas. Karena data yang dikembalikan hanya satu, maka hasil akhir dari method `first()` bukan berbentuk collection, tapi langsung 1 data model mahasiswa. Berikut contoh penggunaannya:

`routes\web.php`

```
1 Route::get('first', function () {
2     $mahasiswa = Mahasiswa::first();
3     return view('mahasiswa',compact('mahasiswa'));
```

```
4});
```



The screenshot shows a browser window with the title 'Data Mahasiswa'. The URL in the address bar is 'localhost:8000/first'. The page content is a table with the following data:

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
1	10300234	Mustofa Simanjuntak	02-12-1999	2.90	21-06-2020	29-06-2020

Gambar: Hasil method Mahasiswa::first()

Ketika dipanggil tanpa method tambahan, data di tabel `data_mahasiswa` akan di terurut berdasarkan kolom `id`. Sehingga perintah `Mahasiswa::first()` menampilkan mahasiswa yang memiliki `id = 1`.

Umumnya kita perlu mengurutkan data berdasarkan kolom tertentu, kemudian baru mengambil 1 data paling atas:

routes\web.php

```
1 Route::get('order-by-first', function () {
2     $mahasiswa = Mahasiswa::orderBy('ipk','desc')->first();
3     return view('mahasiswa',compact('mahasiswa'));
4});
```

Kode ini akan menampilkan 1 data mahasiswa dengan ipk paling tinggi.

Method `first()` juga sering didahului dengan method `where()`,= seperti contoh berikut:

routes\web.php

```
1 Route::get('where-first', function () {
2     $mahasiswa = Mahasiswa::where('ipk','<=',3)->first();
3     return view('mahasiswa',compact('mahasiswa'));
4});
```

Sekarang hasilnya adalah satu data mahasiswa dengan ipk kurang atau sama dengan 3. Jika tanpa method `first()`, besar kemungkinan ada banyak mahasiswa yang memenuhi kriteria ini.

Karena perintah gabungan `first()` dan `where()` cukup sering dipakai, Laravel menyediakan penulisan singkat, yakni `firstWhere()`:

routes\web.php

```
1 Route::get('first-where', function () {
2     $mahasiswa = Mahasiswa::firstWhere('ipk','<=',3);
3     return view('mahasiswa',compact('mahasiswa'));
4});
```

Method `Mahasiswa::firstWhere('ipk', '<=' , 3)` akan mengembalikan 1 data mahasiswa di urutan paling atas yang memenuhi kondisi ipk \leq 3.

Method latest()

Sekilas, method `latest()` terlihat sebagai kebalikan dari method `first()`, tapi ternyata tidak persis seperti itu. Method `latest()` dipakai untuk menampilkan semua data tabel yang telah terurut berdasarkan kolom `created_at` mulai dari data terbaru (paling akhir di `create`).

Berikut contoh penggunaannya:

`routes\web.php`

```
1 Route::get('latest', function () {
2     $mahasiswas = Mahasiswa::latest()->get();
3     return view('mahasiswas', compact('mahasiswas'));
4 });
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020
3	10605319	Queen Suryatmi	08-01-2001	2.88	23-06-2020	29-06-2020
8	10975558	Eka Hasanah	23-09-2000	2.15	23-06-2020	29-06-2020
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020

Gambar: Hasil method `Mahasiswa::latest()->get()`

Terlihat data langsung terurut berdasarkan kolom `created_at`. Bisa juga disebut bahwa method `Mahasiswa::latest()->get()` adalah penulisan singkat dari `Mahasiswa::orderBy('created_at', 'desc')->get()`.

Jika kita hanya ingin mengambil 1 data paling akhir, bisa dengan menggabung method `latest()` dan `first()` seperti contoh berikut:

`routes\web.php`

```
1 Route::get('latest-first', function () {
2     $mahasiswa = Mahasiswa::latest()->first();
3     return view('mahasiswa', compact('mahasiswa'));
4 });
```



The screenshot shows a browser window with the title 'Data Mahasiswa'. The URL in the address bar is 'localhost:8000/latest-first'. The page content is a table with the following data:

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
4	10694755	Kemal Padmasari	10-06-2001	2.88	24-06-2020	27-06-2020

Gambar: Hasil method Mahasiswa::latest()->first()

Method find() dan findOrFail()

Method `find()` berfungsi untuk mengambil data tabel berdasarkan kolom `id`. Dalam bentuk yang paling sederhana, method `find()` butuh 1 argument berupa nomor `id` yang ingin diambil:

routes\web.php

```
1 Route::get('find', function () {
2     $mahasiswa = Mahasiswa::find(5);
3     return view('mahasiswa', compact('mahasiswa'));
4 });
```

Perintah di atas akan menampilkan 1 data mahasiswa yang memiliki `id` = 5.

Jika kita ingin mengambil beberapa data mahasiswa dengan `id` yang berbeda-beda, bisa dengan menginput nomor `id` tersebut sebagai sebuah array:

routes\web.php

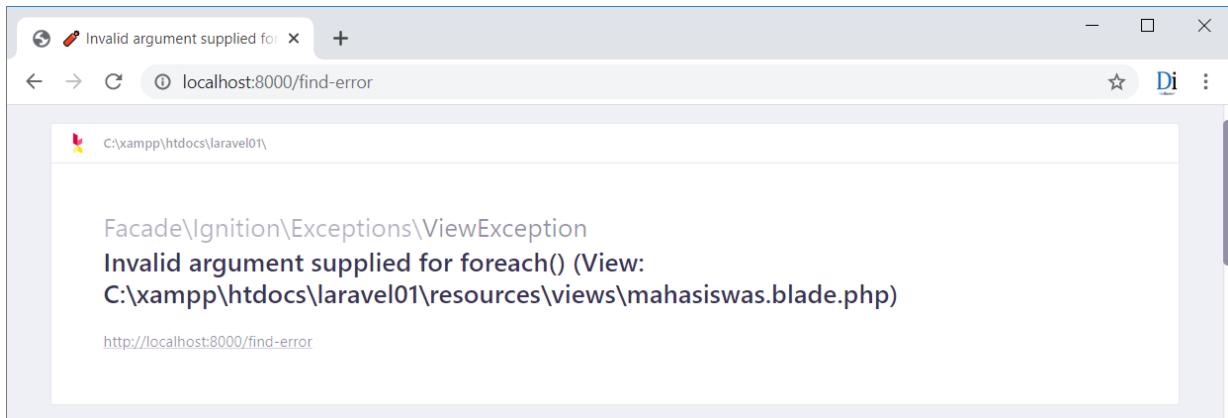
```
1 Route::get('find-array', function () {
2     $mahasiswas = Mahasiswa::find([1,4,6,7]);
3     return view('mahasiswas', compact('mahasiswas'));
4 });
```

Namun perhatikan bahwa jika dijalankan seperti ini, hasil akhir dari method `find()` sudah berbentuk `collection` karena terdiri dari beberapa data mahasiswa. Sedangkan jika method `find()` dipakai untuk mengambil 1 data mahasiswa saja seperti contoh yang pertama, akan mengembalikan satu object model mahasiswa.

Laravel juga menyediakan method `findOrFail()` untuk mengantisipasi error seandainya `id` yang ingin dicari ternyata tidak ditemukan. Sebagai contoh, jalankan kode program berikut:

routes\web.php

```
1 Route::get('find-error', function () {
2     $mahasiswas = Mahasiswa::find(99);
3     return view('mahasiswas', compact('mahasiswas'));
4 });
```

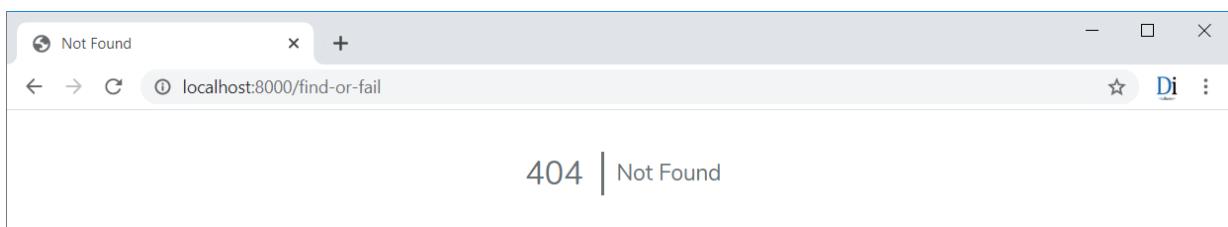


Gambar: Error karena id 99 tidak ditemukan

Halaman error ini tampil karena mahasiswa dengan id 99 tidak ada di dalam tabel `data_mahasiswa`. Untuk menghindari tampilan error, method `find(99)` bisa diganti menjadi `findOrFail(99)`:

`routes\web.php`

```
1 Route::get('find-or-fail', function () {
2     $mahasiswa = Mahasiswa::findOrFail(99);
3     return view('mahasiswa', compact('mahasiswa'));
4 });
```



Gambar: Halaman 404

Sekarang jika id 99 tidak ditemukan, yang tampil adalah halaman 404 | Not Found.

Method value()

Method `value()` bisa dipakai untuk mengambil satu nilai kolom dari sebuah baris data. Nama kolom tersebut diinput sebagai argument ke dalam method `value()`. Berikut contoh penggunaannya:

`routes\web.php`

```
1 Route::get('value-1', function () {
2     $result = Mahasiswa::value('nama');
3     echo $result; // Mustofa Simanjuntak
4 });
```

Dalam contoh di atas, method `Mahasiswa::value('nama')` mengembalikan nilai kolom `nama` dari baris pertama tabel `data_mahasiswa`, yakni "Mustofa Simanjuntak". Hasil akhir dari

method `value()` langsung berbentuk string, bukan lagi model object sehingga tidak bisa kita teruskan ke view `mahasiswa.blade.php`.

Umumnya sebelum menjalankan method `value()`, perlu menyeleksi data yang akan diambil, misalnya menggunakan method `where()`:

routes\web.php

```
1 Route::get('value-2', function () {
2     $result = Mahasiswa::where('nama', 'Clara Wijaya')->value('ipk');
3     echo $result;    // 3.00
4});
```

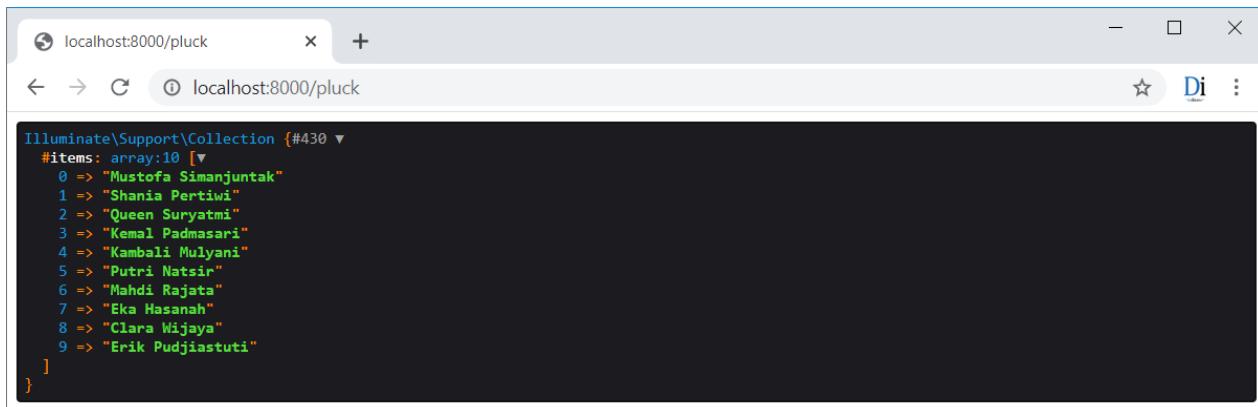
Method `Mahasiswa::where('nama', 'Clara Wijaya')->value('ipk')` artinya cari mahasiswa bernama "Clara Wijaya", lalu ambil nilai ipk dari mahasiswa tersebut.

Method `pluck()`

Method `pluck()` bisa dipakai untuk mengambil semua nilai data untuk satu kolom. Kolom yang ingin diambil diinput sebagai argument. Hasil akhir dari method ini berupa sebuah collection dari kumpulan string:

routes\web.php

```
1 Route::get('pluck', function () {
2     $result = Mahasiswa::pluck('nama');
3     dump($result);
4});
```



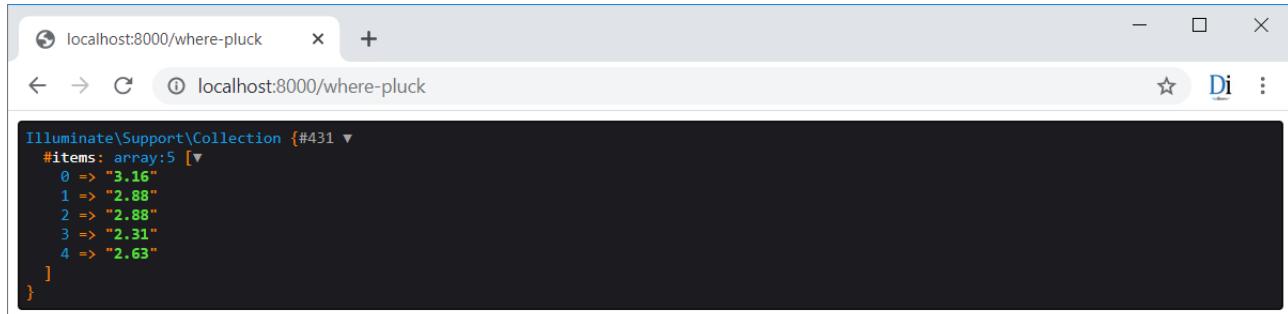
Gambar: Hasil method `Mahasiswa::pluck('nama')`

Method `Mahasiswa::pluck('nama')` akan mengembalikan nilai kolom nama dari semua baris. Kembali, kita tidak bisa menampilkannya lewat view `mahasiswa.blade.php` karena data yang dihasilkan hanya untuk satu kolom nama saja.

Method `pluck()` juga bisa dipadukan dengan method Eloquent lain, misalnya `where()`:

routes\web.php

```
1 Route::get('where-pluck', function () {
2     $result = Mahasiswa::where('nama', 'LIKE', '%i')->pluck('ipk');
3     dump($result);
4 });
```



Gambar: Hasil method Mahasiswa::where('nama','LIKE','%i')->pluck('ipk')

Dalam kode ini saya ingin menampilkan semua nilai ipk untuk mahasiswa yang namanya diakhiri dengan huruf i.

Method exists() dan doesntExist()

Dalam beberapa kesempatan, kita hanya ingin memeriksa apakah sebuah data ada di dalam database atau tidak. Untuk keperluan ini, bisa menggunakan method `exists()` atau `doesntExist()`. Kedua method mengembalikan nilai boolean `true` atau `false` tergantung apakah data di temukan atau tidak:

routes\web.php

```
1 Route::get('exists', function () {
2     $result = Mahasiswa::where('ipk', '>', 3.6)->exists();
3     dump($result); // false
4
5     $result = Mahasiswa::where('nama', 'LIKE', '%a')->exists();
6     dump($result); // true
7 });
8
9 Route::get('doesnt-exist', function () {
10    $result = Mahasiswa::where('ipk', '>', 3.6)->doesntExist();
11    dump($result); // true
12
13    $result = Mahasiswa::where('nama', 'LIKE', '%a')->doesntExist();
14    dump($result); // false
15});
```

Dengan beberapa percobaan ini bisa di simpulkan bahwa dalam tabel `data_mahasiswa` tidak ditemukan mahasiswa yang memiliki ipk di atas 3.6. Namun terdapat setidaknya satu mahasiswa yang memiliki nama dengan akhiran huruf a.

Method count(), max(), min() dan avg()

Method `count()`, `max()`, `min()` dan `avg()` berfungsi untuk mencari jumlah, nilai minimum, nilai maksimum dan nilai rata-rata dari sebuah kolom. Dalam SQL, fungsi seperti ini disebut sebagai *aggregate function*.

Keempat method butuh sebuah argument berupa nama tabel yang akan dihitung. Berikut contoh penggunaannya:

`routes\web.php`

```

1  Route::get('aggregate', function () {
2      $result = Mahasiswa::all()->count();
3      dump($result); // 10
4
5      $result = Mahasiswa::where('nama', 'LIKE', '%e%')->count();
6      dump($result); // 5
7
8      $result = Mahasiswa::all()->max('ipk');
9      dump($result); // 3.16
10
11     $result = Mahasiswa::where('nama', 'LIKE', '%e%')->max('tanggal_lahir');
12     dump($result); // 2001-06-10
13
14     $result = Mahasiswa::all()->min('ipk');
15     dump($result); // 2.15
16
17     $result = Mahasiswa::all()->avg('ipk');
18     dump($result); // 2.679
19 });

```

Dari kode di atas, bisa disimpulkan bahwa:

- ◆ Terdapat 10 mahasiswa di dalam tabel.
- ◆ Terdapat 5 mahasiswa yang memiliki minimal 1 huruf e di dalam namanya.
- ◆ Nilai ipk paling tinggi dari semua mahasiswa adalah 3.16.
- ◆ Dari semua mahasiswa dengan minimal 1 huruf e di dalam nama, yang paling muda lahir di 2001-06-10.
- ◆ Nilai ipk paling rendah dari semua mahasiswa adalah 2.15.
- ◆ Rata-rata nilai ipk semua mahasiswa adalah 2.679.

Method *aggregate* ini cocok dipakai untuk membuat laporan atau logika program yang butuh data spesifik.

Method firstOrCreate()

Method `firstOrCreate()` merupakan gabungan dari method `first()` dan `create()`. Jika data yang dicari ada di dalam tabel, method ini mengembalikan satu eloquent model, atau sama

seperti hasil method `first()`. Namun jika tidak ditemukan, sebuah data baru akan dibuat, atau sama seperti fungsi method `create()`.

Method `firstOrCreate()` butuh minimal satu argument berbentuk array yang berisi pasangan nama kolom dan nilai yang dicari. Berikut contoh penggunaannya:

`routes\web.php`

```
1 Route::get('first-or-create', function () {
2     $mahasiswa = Mahasiswa::firstOrCreate(['nama' => 'Shania Pertiwi']);
3     return view('mahasiswa', compact('mahasiswa'));
4});
```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
2	10512421	Shania Pertiwi	17-12-1999	3.16	22-06-2020	27-06-2020

Gambar: Hasil method `Mahasiswa::firstOrCreate(['nama' => 'Shania Pertiwi'])`

Method `Mahasiswa::firstOrCreate(['nama' => 'Shania Pertiwi'])` akan memeriksa apakah mahasiswa dengan nama "Shania Pertiwi" ada di dalam tabel atau tidak. Jika ditemukan, data mahasiswa tersebut diambil dan disimpan ke dalam variabel `$mahasiswa`.

Sekarang mari kita tukar dengan nama yang tidak ada di dalam tabel `data_mahasiswa`:

`routes\web.php`

```
1 Route::get('first-or-create', function () {
2     $mahasiswa = Mahasiswa::firstOrCreate(['nama' => 'Nova Pertiwi']);
3     return view('mahasiswa', compact('mahasiswa'));
4});
```

Illuminate\Database\QueryException
SQLSTATE[HY000]: General error: 1364 Field 'nim' doesn't have a default value (SQL: insert into `data_mahasiswa` (`nama`, `tanggal_lahir`, `updated_at`, `created_at`) values ('Nova Pertiwi', 1990-01-01, 2020-07-11 02:35:58, 2020-07-11 02:35:58))

Gambar: Error karena gagal proses insert

Ketika mahasiswa "Nova Pertiwi" tidak ditemukan, method `firstOrCreate(['nama' => 'Nova Pertiwi'])` akan mencoba membuat data baru untuk mahasiswa tersebut.

Masalahnya, proses pembuatan data mahasiswa butuh info lain seperti `nim`, `tanggal_lahir`, dan `ipk`. Data-data ini tidak ditemukan sehingga terjadi error di atas. Solusinya, kita bisa tambah argument kedua ke dalam method `firstOrCreate()` untuk mengisi info tersebut:

`routes\web.php`

```

1 Route::get('first-or-create-input', function () {
2     $mahasiswa = Mahasiswa::firstOrCreate(
3         ['nama' => 'Nova Pertiwi'],
4         [
5             'nim' => '10512007',
6             'ipk' => 3.4,
7         ]
8     );
9     return view('mahasiswa', compact('mahasiswa'));
10 });

```

Argument kedua untuk method `firstOrCreate()` terdapat di baris 4 – 7. Di sini saya mengisi data `nim` dengan '10512007' dan 'ipk' dengan 3.4.

Tapi bagaimana dengan kolom `tanggal_lahir`? Nilai untuk kolom `tanggal_lahir` sebenarnya juga harus di tulis, akan tetapi di dalam file model `Mahasiswa.php` kita masih memiliki nilai default berupa '1990-01-01'. Dengan demikian, kolom `tanggal_lahir` tidak akan error jika nilainya tidak ditulis, karena akan menggunakan nilai default yang ada di `Mahasiswa.php`.

Pada saat URL `localhost:8000/first-or-create-input` di akses, mahasiswa dengan nama 'Nova Pertiwi' beserta data yang terdapat di argument kedua akan diinput terlebih dahulu ke dalam database. Setelah itu, sebuah model object dari mahasiswa tersebut akan dikembalikan ke dalam variabel `$mahasiswa` untuk selanjutnya di teruskan ke dalam view.

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
11	10512007	Nova Pertiwi	01-01-1990	3.4	11-07-2020	11-07-2020

Gambar: Proses input berhasil di tambahkan ke dalam tabel

Perhatikan nilai kolom ID, yakni 11. Ini berarti terdapat tambahan satu data baru ke dalam tabel `data_mahasiswa` karena sebelumnya kita hanya memiliki 10 baris data.

Agar lebih fleksibel, nilai-nilai kolom juga bisa ditulis menggunakan Faker:

routes\web.php

```

1 Route::get('first-or-create-faker', function () {
2     $mahasiswa = Mahasiswa::firstOrCreate(
3         ['nama' => 'Nova Pertiwi'],
4         [
5             'nim' => Faker\Factory::create()->unique()->numerify('10#####'),
6             'ipk' => Faker\Factory::create()->randomFloat(2, 2, 4),
7         ]
8     );
9     return view('mahasiswa', compact('mahasiswa'));
10 });

```

Sekarang, nilai untuk kolom 'nim' dan 'ipk' akan di generate oleh Faker dan akan selalu berubah-ubah saat menginput data baru.

Method firstOrNew()

Method `firstOrNew()` sebenarnya mirip seperti `firstOrCreate()`. Bedanya, jika data yang dicari tidak ditemukan, method ini tidak langsung menginput data baru ke dalam database, tapi hanya mengembalikan sebuah model object.

Berikut contoh penggunaannya:

routes\web.php

```

1 Route::get('first-or-new', function () {
2     $faker = \Faker\Factory::create('id_ID');
3     $mahasiswa = Mahasiswa::firstOrNew(
4         ['nama' => 'Lisa Pertiwi'],
5         [
6             'id' => $faker->numberBetween(100, 199),
7             'nim' => $faker->numerify('10#####'),
8             'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01', '+3 years'),
9             'ipk' => $faker->randomFloat(2, 2, 4),
10            'created_at' => $faker->dateTimeBetween('-10 days', '-5 days'),
11            'updated_at' => $faker->dateTimeBetween('-3 days'),
12        ]
13    );
14    return view('mahasiswa', compact('mahasiswa'));
15 });

```

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
192	10240676	Lisa Pertiwi	09-11-2000	2.53	05-07-2020	10-07-2020

Gambar: Hasil method Mahasiswa::firstOrNew(...)

Ketika di jalankan, jika terdapat mahasiswa bernama 'Lisa Pertiwi' di dalam database, maka method `firstOrNew()` akan mengembalikan object untuk model tersebut (diambil dari dalam database). Namun jika tidak ditemukan, sebuah object Mahasiswa baru akan dibuat berdasarkan nilai yang ditulis pada argument kedua.

Sebagai argument kedua pada method `Mahasiswa::firstOrNew()` di atas, saya harus menulis seluruh nilai untuk kolom tabel (baris 6-11). Ini diperlukan karena model object langsung dibuat dari kode program, bukan diambil dari database. Hal ini mirip seperti praktik method `make()` pada bab **Factory**.

Akan tetapi, jika nantinya kita tetap ingin menyimpan data ini ke dalam database, bisa memanggil method `save()` dari variabel `$mahasiswa`:

`routes\web.php`

```

1  Route::get('first-or-new-save', function () {
2      $faker = \Faker\Factory::create('id_ID');
3      $mahasiswa = Mahasiswa::firstOrNew(
4          ['nama' => 'Lisa Pertiwi'],
5          [
6              'id' => $faker->numberBetween(100,199),
7              'nim' => $faker->numerify('10#####'),
8              'tanggal_lahir' => $faker->dateTimeInInterval('1999-01-01', '+3 years'),
9              'ipk' => $faker->randomFloat(2, 2, 4),
10             'created_at' => $faker->dateTimeBetween('-10 days', '-5 days'),
11             'updated_at' => $faker->dateTimeBetween('-3 days'),
12         ]
13     );
14     // save data mahasiswa ke database
15     $mahasiswa->save();
16     return view('mahasiswa', compact('mahasiswa'));
17 });

```

Dengan penambahan perintah `$mahasiswa->save` di baris 15, maka setelah object Mahasiswa dibuat, selanjutnya akan disimpan ke dalam database.

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
198	10627764	Lisa Pertiwi	01-12-2000	2.97	06-07-2020	09-07-2020

Gambar: Hasil method `Mahasiswa::firstOrNew(...)` + `save()`

Sebagai perbandingan dengan method `firstOrCreate()`, alur yang terjadi untuk method `firstOrNew() + save()` adalah, object Mahasiswa di buat terlebih dahulu baru kemudian di

simpan ke dalam database. Sedangkan untuk method `firstOrCreate()`, data mahasiswa diinput ke database terlebih dahulu, baru diambil kembali untuk menjadi model object.

Method `firstOr()`

Method `firstOr()` merupakan versi generik atau versi umum dari gabungan method `first()` dan "sesuatu". Maksud "sesuatu" di sini adalah sebuah function yang bisa kita input sebagai argument ke dalam method `firstOr()`. Pengertian ini akan lebih jelas dengan contoh praktik:

`routes\web.php`

```
1 Route::get('first-or', function () {
2     $faker = \Faker\Factory::create('id_ID');
3     $mahasiswa = Mahasiswa::where('nama', 'Dewi Pertiwi')->firstOr(
4         function () {
5             return Mahasiswa::find(5);
6         }
7     );
8     return view('mahasiswa', compact('mahasiswa'));
9 });
```

Di baris 3 saya menggabung method `where()` dengan `firstOr()`. Cara membacanya adalah, cari apakah terdapat mahasiswa bernama 'Dewi Pertiwi' di dalam tabel. Jika ada, kembalikan model object dari mahasiswa tersebut.

Akan tetapi jika tidak ditemukan, maka function di baris 4 – 6 akan di eksekusi. Dalam contoh ini, perintah tersebut adalah `return Mahasiswa::find(5)`, yakni kembalikan model object untuk mahasiswa dengan id 5.

ID	NIM	Nama	Tanggal Lahir	IPK	Created At	Updated At
5	10531551	Kambali Mulyani	23-11-1999	2.31	22-06-2020	27-06-2020

Gambar: Hasil method `Mahasiswa::where('nama','Dewi Pertiwi')->firstOr(...)`

Isi function untuk method `firstOr()` bisa kode apa saja, namun biasanya mengembalikan sebuah model object. Jika kita punya file factory untuk model `Mahasiswa`, itu juga bisa dijalankan seperti contoh berikut:

`routes\web.php`

```
1 Route::get('first-or', function () {
2     $faker = \Faker\Factory::create('id_ID');
3     $mahasiswa = Mahasiswa::where('nama', 'Dewi Pertiwi')->firstOr(
4         function () {
```

```
5      // jika ada factory, bisa dipanggil disini
6      return factory(Mahasiswa::class)->make();
7  }
8  );
9  return view('mahasiswa',compact('mahasiswa'));
10 );
```

Karena kita tidak menyiapkan file factory Mahasiswa, kode program di atas akan error. Namun setidaknya inilah gambaran prinsip kerja dari method `firstOr()`.

Method `updateOrCreate()`

Di lihat dari namanya, bisa di tebak bahwa method ini adalah gabungan dari proses `update()` dan `create()`. Jika data yang dicari sudah ada di dalam tabel, maka lakukan proses `update()`. Namun jika data tersebut tidak ditemukan, lakukan proses `create()`.

Berikut contoh penggunaan dari method `updateOrCreate()`:

routes\web.php

```
1 Route::get('update-or-create', function () {
2     $mahasiswa = Mahasiswa::updateOrCreate(
3         ['nama' => 'Naira Pertiwi'],
4         [
5             'nim' => Faker\Factory::create()->unique()->numerify('10#####'),
6             'ipk' => 3.99,
7         ]
8     );
9     return view('mahasiswa',compact('mahasiswa'));
10});
```

Argument yang diperlukan mirip seperti method gabungan sebelumnya, yakni data yang dicari ada di argument pertama, dan data untuk proses update atau create ada di argument kedua.

Jika mahasiswa bernama 'Naira Pertiwi' di temukan, maka update kolom `nim` dan `ipk` sesuai data yang ada di argument kedua. Namun jika tidak ditemukan, tambah 1 mahasiswa baru dengan nama 'Naira Pertiwi' beserta nilai `nim` dan `ipk` yang ada di argument kedua.

Dalam bab ini kita telah membahas materi tambahan Eloquent, terutama method yang dipakai untuk menampilkan data tabel. Meskipun sudah cukup banyak, method-method ini baru mengakses satu tabel saja.

Bagaimana dengan menampilkan data dari 2 tabel yang saling terhubung? Inilah topik bahasan kita berikutnya, yakni tentang **Eloquent Relationship**. Total terdapat 8 jenis relationship yang akan di bahas mulai dari bab berikutnya.

12. Eloquent Relationship: One to One

Akhirnya kita sampai ke bahasan utama buku ini, yakni **Eloquent Relationship**. Sebagai materi pertama akan dibahas tentang hubungan *one to one*, dimana satu data di tabel utama akan terhubung ke satu data di tabel kedua.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

12.1. Pengertian One to One Relationship

Eloquent Relationship adalah sebutan untuk penerapan Eloquent ke dalam hubungan (*relationship*) antar tabel di dalam database.

Dalam teori database, terdapat cukup banyak relationship yang bisa dibuat antara satu tabel dengan tabel lain. Kali ini kita akan bahas *relationship one to one* terlebih dahulu, nantinya juga terdapat *relationship one to many*, dan juga *relationship many to many*.

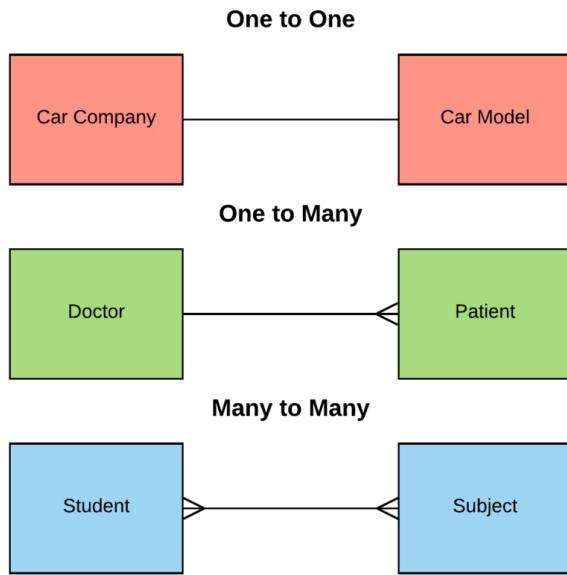
Maksud dari **one to one relationship** adalah, satu baris data di tabel utama terhubung dengan satu baris data di tabel kedua. Dan begitu juga sebaliknya, satu baris data di tabel kedua, terhubung dengan satu baris data di tabel utama.

Contoh dari konsep ini seperti satu mahasiswa yang memiliki satu nilai ipk, atau satu perusahaan memiliki satu nomor telepon.

Sebenarnya, *relationship one to one* tidak terlalu sering di pakai, karena pada dasarnya jika satu baris data hanya berhubungan dengan satu data lain (dan begitu juga sebaliknya), maka kedua tabel bisa digabung menjadi satu tabel panjang.

Alasan utama pembuatan *relationship one to one* lebih ke performa, karena bisa jadi hanya beberapa kolom saja yang sering di akses, sedangkan kolom lain lebih ke data tambahan. Data tambahan inilah yang bisa dipecah menjadi tabel kedua.

Dalam diagram **ERD** (Entity Relationship Diagram), *relationship one to one* digambarkan dengan satu garis yang tidak bercabang:



Gambar: Diagram ERD dari 3 jenis relationship (sumber gambar: <https://commons.wikimedia.org>)

Jika menggunakan perintah SQL biasa, proses penggabungan tabel biasanya dibuat dengan query `SELECT ... JOIN`. Penulisan query `JOIN` kadang bisa jadi sangat rumit apalagi jika sudah melibatkan banyak tabel yang saling terhubung.

Dengan eloquent relationship, kita tidak perlu menulis query `JOIN` lagi, tapi cukup memanggil beberapa method singkat saja.

Praktek tentang eloquent relationship butuh pemahaman dasar teori database seperti *primary key*, *foreign key*, serta *referential integrity*. Konsep teori ini cukup luas dan di luar materi Laravel.

Dalam beberapa kesempatan saya akan coba jelaskan secara singkat tentang istilah yang ada, namun lebih ideal jika anda sudah pernah mendalami database dan paham query dasar MySQL, misalnya dari buku **MySQL Uncover** DuniaIlkom.

12.2. Persiapan Awal

Sebagai bahan praktek dari penerapan eloquent relationship *one to one*, kita akan buat dua buah tabel, yakni tabel `mahasiswa` dan tabel `nilai`. Kedua tabel ini berpasangan dengan model **Mahasiswa** dan model **Nilai**.

Tabel `mahasiswa` berisi 6 kolom: `id`, `nim`, `nama`, `jurusan`, `created_at` dan `updated_at`.

Sedangkan tabel `nilai` berisi 7 kolom: `id`, `sem_1`, `sem_2`, `sem_3`, `mahasiswa_id`, `created_at` dan `updated_at`.

Eloquent Relationship: One to One

Berikut struktur tabel `mahasiswa` dan tabel `nilais` yang akan kita buat:

The screenshot shows the MySQL command-line interface with two queries run against the 'laravel' database.

```
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim  | nama | jurusan | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1  | 10300234 | Mustofa Simanjuntak | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 2  | 10451982 | Marsito Purnawati | Sistem Informasi | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 3  | 10980764 | Ika Puspasari | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 4  | 10605319 | Queen Suryatmi | Sistem Informasi | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 5  | 10438572 | Yuliana Nurdyanti | Teknik Informatika | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 6  | 10737995 | Tiara Siregar | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 7  | 10531551 | Kambali Mulyani | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 8  | 10155818 | Hesti Ramadan | Teknik Informatika | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 9  | 10274992 | Galang Maryadi | Teknik Informatika | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 10 | 10228263 | Mahdi Rajata | Sistem Informasi | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
+----+-----+-----+-----+-----+-----+
10 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM nilais;
+----+-----+-----+-----+-----+-----+
| id | sem_1 | sem_2 | sem_3 | mahasiswa_id | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1  | 2.18  | 2.44  | 3.14  | 9            | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 2  | 3.13  | 2.35  | 3.01  | 1            | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 3  | 3.15  | 2.15  | 3.39  | 8            | 2020-07-14 02:36:43 | 2020-07-14 02:36:43 |
| 4  | 2.85  | 3.50  | 2.20  | 3            | 2020-07-14 02:36:43 | 2020-07-14 02:36:43 |
| 5  | 3.89  | 2.98  | 3.00  | 4            | 2020-07-14 02:36:43 | 2020-07-14 02:36:43 |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Tabel mahasiswa dan tabel nilais

Hubungan yang akan di rancang untuk kedua tabel adalah **one to one**, dimana 1 baris data di tabel `mahasiswa` akan terhubung dengan 1 baris data di tabel `nilais`.

Tabel `nilais` dipakai untuk menampung nilai IP (Indeks Prestasi) setiap mahasiswa. Kolom `sem_1`, `sem_2`, dan `sem_3` disiapkan untuk nilai IP semester 1, semester 2 dan semester 3.

Idealnya, kita perlu menulis hingga semester 12 atau lebih agar sesuai dengan jumlah maksimal semester di sebuah jurusan S-1. Namun untuk menyederhanakan pembahasan, saya batasi sampai semester 3 saja.

Pengertian Primary Key dan Foreign Key

Jika anda sudah pernah belajar teori database, tentu sudah familiar dengan istilah *primary key* dan *foreign key*. Kedua komponen inilah yang menjadi "lem penghubung" antara satu tabel dengan tabel lain.

Primary key adalah sebuah kolom (atau beberapa kolom) yang bisa mengidentifikasi setiap baris di dalam sebuah tabel. Jika tabel di generate menggunakan *migration*, secara otomatis Laravel sudah men-set kolom `id` sebagai *primary key*.

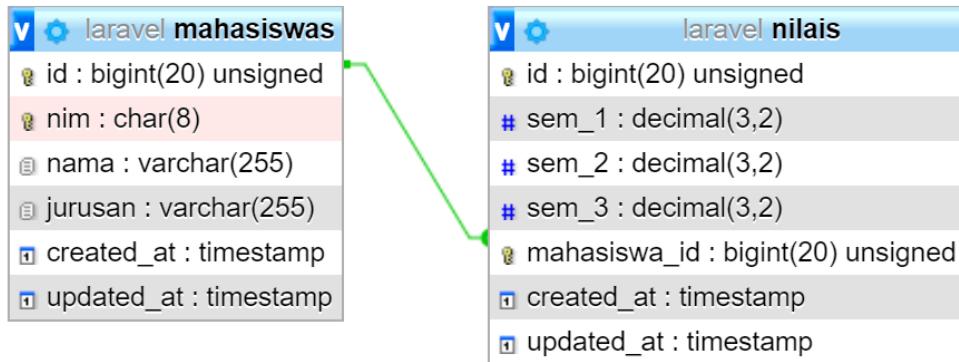
Sedangkan **foreign key** adalah sebutan untuk kolom yang nilainya merujuk ke *primary key* dari tabel lain. Kolom yang bertindak sebagai *foreign key* akan dipakai untuk proses penggabungan tabel (pembuatan *relationship*).

Dalam contoh kita, *foreign key* ini ada di kolom `mahasiswa_id` di tabel `nilais`. Meskipun nama

kolom bisa dibuat dengan bebas, kebiasaan programmer Laravel menulis nama kolom *foreign key* dengan format <nama_tabel_primary>_id.

Penulisan *nama_tabel_primary* menggunakan versi *singular* tanpa akhiran 's'. Sehingga nama kolom *foreign key* yang akan menampung *id* mahasiswa di tabel *nilais* akan bernama *mahasiswa_id*.

Jika digambar ke dalam diagram **ERD**, berikut hubungan antara tabel *mahasiswas* dan tabel *nilais*:



Gambar: Hubungan antara tabel mahasiswas dengan tabel nilais

Terlihat garis antara kolom *id* di tabel *mahasiswas* dengan kolom *mahasiswa_id* di tabel *nilais*. Inilah hubungan *primary key* dengan *foreign key* dalam konsep *relationship one to one*.

Dengan hubungan seperti ini, maka nilai kolom *mahasiswa_id* di tabel *nilais* akan berpasangan dengan kolom *id* di tabel *mahasiswas*.

Nantinya, kita juga bisa menerapkan konsep *referential integrity* untuk menjaga konsistensi data antar tabel. Misalnya proses input ke tabel *nilais* hanya bisa dilakukan jika *id* mahasiswa sudah ada di tabel *mahasiswas*.

Jika anda sedikit bingung dengan istilah *relationship*, *primary key*, *foreign key* dan *referential integrity*, bisa coba cari buku seputar teori database.

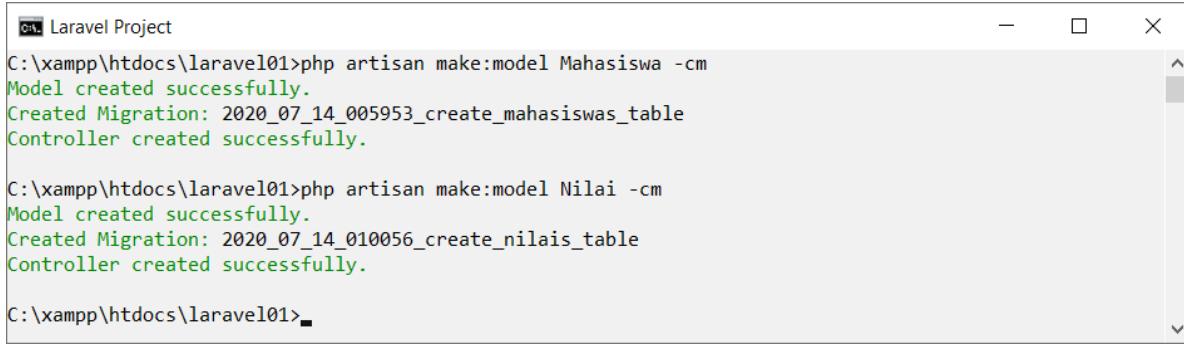
Generate Data Sample

Cukup dengan teori, saatnya masuk ke praktik pembuatan tabel. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```
php artisan make:model Mahasiswa -cm  
php artisan make:model Nilai -cm
```

Kode di atas akan membuat file **model**, **controller** serta **migration** untuk tabel *mahasiswas* dan *nilais*.

Eloquent Relationship: One to One



```
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa -cm
Model created successfully.
Created Migration: 2020_07_14_005953_create_mahasiswas_table
Controller created successfully.

C:\xampp\htdocs\laravel01>php artisan make:model Nilai -cm
Model created successfully.
Created Migration: 2020_07_14_010056_create_nilais_table
Controller created successfully.

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

Gambar: Pembuatan model, controller serta migration

Selanjutnya buka file migration tabel `mahasiswas` lalu rancang struktur tabel berikut:

database\migrations\<timestamp>_create_mahasiswas_table.php

```
11 public function up()
12 {
13     Schema::create('mahasiswas', function (Blueprint $table) {
14         $table->id();
15         $table->char('nim',8)->unique();
16         $table->string('nama');
17         $table->string('jurusan');
18         $table->timestamps();
19     });
20 }
```

Tidak ada sesuatu yang baru di sini. Kolom `nim` saya set dengan char 8 dengan tambahan method `unique()`, kemudian kolom `nama` dan `jurusan` di set sebagai string.

Lanjut, buka file migration untuk tabel `nilais` dan rancang struktur tabel sebagai berikut:

database\migrations\<timestamp>_create_nilais_table.php

```
1 public function up()
2 {
3     Schema::create('nilais', function (Blueprint $table) {
4         $table->id();
5         $table->decimal('sem_1',3,2)->nullable();
6         $table->decimal('sem_2',3,2)->nullable();
7         $table->decimal('sem_3',3,2)->nullable();
8         $table->unsignedBigInteger('mahasiswa_id')->unique();
9         $table->timestamps();
10
11         $table->foreign('mahasiswa_id')->references('id')->on('mahasiswas');
12     });
13 }
```

Setelah pembuatan kolom `id`, saya mendefinisikan kolom `sem_1`, `sem_2`, dan `sem_3` sebagai `decimal`, lalu menambah method `nullable()` agar ketiga kolom boleh berisi nilai `NIL` (boleh tidak diisi).

Di baris 8 terdapat pendefinisian kolom `mahasiswa_id` dengan tipe data `unsignedBigInteger`. Inilah kolom yang di siapkan sebagai *foreign key*. Tipe data `unsignedBigInteger` merupakan tipe data yang juga dipakai oleh setiap kolom `id` dari migration. Karena kolom ini nantinya merujuk ke kolom `id` di tabel `mahasiswas`, maka tipe datanya juga harus sama.

Tambahan method `unique()` akan membuat nilai untuk kolom `mahasiswa_id` tidak boleh berulang. Ini menjadi syarat dari *relationship one to one*, karena jika terdapat beberapa nilai yang merujuk ke satu mahasiswa yang sama, maka itu sudah bukan hubungan *one to one* lagi.

Pembuatan hubungan antara kolom `mahasiswa_id` dengan kolom `id` milik tabel `mahasiswas` dilakukan dengan perintah pada baris 11. Perintah ini bisa dibaca: "Buat *foreign key* untuk kolom `mahasiswa_id` yang merujuk ke kolom `id` milik tabel `mahasiswas`".

Sejak Laravel 7, terdapat perintah tambahan untuk menyederhanakan pembuatan *foreign key*. Sebagai alternatif, kita bisa membuat struktur migration untuk tabel `nilais` dengan kode berikut:

```
database\migrations\<timestamp>_create_nilais_table.php
```

```
1 public function up()
2 {
3     Schema::create('nilais', function (Blueprint $table) {
4         $table->id();
5         $table->decimal('sem_1',3,2)->nullable();
6         $table->decimal('sem_2',3,2)->nullable();
7         $table->decimal('sem_3',3,2)->nullable();
8         $table->foreignId('mahasiswa_id')->unique()->constrained();
9         $table->timestamps();
10    });
11 }
```

Perubahan ada di baris 8. Sekarang pendefinisian kolom `mahasiswa_id` menggunakan tipe data `foreignId` yang merupakan alias atau penulisan lain dari `unsignedBigInteger`. Meskipun pada dasarnya sama saja, tapi method `foreignId()` langsung menegaskan bahwa kolom ini merupakan sebuah *foreign key*.

Kemudian di akhir baris terdapat tambahan method `constrained()`. Ini juga penulisan singkat untuk membuat hubungan antar tabel. Method `constrained()` menggantikan penulisan kode di baris 11 sebelumnya. Syarat dari penulisan singkat ini adalah, nama kolom *foreign key* harus ditulis dalam format `<nama_tabel_primary>_id`. Ini karena Laravel akan mencoba menebak nama tabel asal *primary key* berdasarkan nama kolom.

Kita bisa menggunakan versi yang mana saja untuk membuat *foreign key*, apakah menggunakan cara penulisan lengkap sebagai berikut:

```
$table->unsignedBigInteger('mahasiswa_id')->unique();
$table->foreign('mahasiswa_id')->references('id')->on('mahasiswas');
```

Atau menggunakan versi singkat:

```
$table->foreignId('mahasiswa_id')->unique()->constrained();
```

Buat kedua tabel dengan menjalankan proses migration:

```
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 (489.89ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (1,248.79ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (539.47ms)
Migrating: 2020_10_10_022554_create_mahasiswa_table
Migrated: 2020_10_10_022554_create_mahasiswa_table (493.55ms)
Migrating: 2020_10_10_022605_create_nilais_table
Migrated: 2020_10_10_022605_create_nilais_table (2,322.16ms)

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

Gambar: Jalankan migration untuk pembuatan tabel mahasiswa dan nilai

Langkah berikutnya adalah membuat beberapa data sample. Kali ini saya kembali menggunakan bantuan **Seeder** dan **Faker**. Silahkan buka file `DatabaseSeeder.php`, lalu modifikasi sebagai berikut:

```
database\seeders\DatabaseSeeder.php
```

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7 use App\Models\Mahasiswa;
8 use App\Models\Nilai;
9
10 class DatabaseSeeder extends Seeder
11 {
12     public function run()
13     {
14         $faker = Faker::create('id_ID');
15         $faker->seed(123);
16         $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
17
18         for ($i=0; $i<10; $i++) {
19             Mahasiswa::create(
20                 [
21                     'nim' => $faker->unique()->numerify('10#####'),
22                     'nama' => $faker->firstName." ".$faker->lastName,
```

Eloquent Relationship: One to One

```
23     'jurusan' => $faker->randomElement($jurusan),
24   ]
25 );
26 }
27
28 for ($i=0; $i<5; $i++) {
29   Nilai::create(
30   [
31     'sem_1' => $faker->randomFloat(2, 2, 4),
32     'sem_2' => $faker->randomFloat(2, 2, 4),
33     'sem_3' => $faker->randomFloat(2, 2, 4),
34     'mahasiswa_id' => $faker->unique()->randomDigit,
35   ]
36 );
37 }
38 }
39 }
```

Kode ini akan men-generate 10 data mahasiswa serta 5 data nilai. Jalankan dengan perintah
php artisan db:seed :

```
C:\ Laravel Project
C:\xampp\htdocs\laravel01>php artisan db:seed
Database seeding completed successfully.

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

Gambar: Menjalankan Seeder

Untuk memastikan, bisa cek isi kedua tabel menggunakan MySQL client atau phpMyAdmin:

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | jurusan | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 10300234 | Mustofa Simanjuntak | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 2 | 10451982 | Marsito Purnawati | Sistem Informasi | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 3 | 10980764 | Ika Puspasari | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 4 | 10605319 | Queen Suryatmi | Sistem Informasi | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 5 | 10438572 | Yuliana Nurdyanti | Teknik Informatika | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 6 | 10737995 | Tiara Siregar | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 7 | 10531551 | Kambali Mulyani | Ilmu Komputer | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 8 | 10155818 | Hesti Ramadan | Teknik Informatika | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 9 | 10274992 | Galang Maryadi | Teknik Informatika | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 10 | 10228263 | Mahdi Rajata | Sistem Informasi | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
+----+-----+-----+-----+-----+-----+
10 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM nilais;
+----+-----+-----+-----+-----+-----+
| id | sem_1 | sem_2 | sem_3 | mahasiswa_id | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 2.18 | 2.44 | 3.14 | 9 | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 2 | 3.13 | 2.35 | 3.01 | 1 | 2020-07-14 02:36:42 | 2020-07-14 02:36:42 |
| 3 | 3.15 | 2.15 | 3.39 | 8 | 2020-07-14 02:36:43 | 2020-07-14 02:36:43 |
| 4 | 2.85 | 3.50 | 2.20 | 3 | 2020-07-14 02:36:43 | 2020-07-14 02:36:43 |
| 5 | 3.89 | 2.98 | 3.00 | 4 | 2020-07-14 02:36:43 | 2020-07-14 02:36:43 |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Data tabel mahasiswas dan nilais yang di generate Seeder

Persiapan sudah selesai, saatnya masuk ke materi penggabungan data.

12.3. Membuat Join dengan DB Facade (Raw SQL Queries)

Sebelum masuk ke eloquent relationship, saya ingin membahas cara penggabungan tabel dengan menulis query MySQL secara manual. Ini agar nantinya bisa kita bandingkan dengan solusi yang disediakan eloquent. Selain itu untuk tabel yang kompleks, kadang menulis langsung query MySQL akan lebih efisien dibandingkan memakai eloquent relationship.

Di buku Laravel Uncover kita sudah bahas cara menjalankan query MySQL secara manual, yakni bab tentang **Raw SQL Queries** yang diakses dari **DB Facade**.

Kode program *raw query* ini akan saya buat di file `MahasiswaController.php`, dan berikut daftar route yang diperlukan:

`routes\web.php`

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::get('/mahasiswa/all', [MahasiswaController::class, 'all']);
7 Route::get('/mahasiswa/gabung-1', [MahasiswaController::class, 'gabung1']);
8 Route::get('/mahasiswa/gabung-2', [MahasiswaController::class, 'gabung2']);
9 Route::get('/mahasiswa/gabung-join-1', [MahasiswaController::class, 'gabungJoin1']);
10 Route::get('/mahasiswa/gabung-join-2', [MahasiswaController::class, 'gabungJoin2']);
11 Route::get('/mahasiswa/gabung-join-3', [MahasiswaController::class, 'gabungJoin3']);
```

Route pertama dipakai untuk testing mengakses semua data mahasiswa:

`app\Http\Controllers\MahasiswaController.php`

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use Illuminate\Http\Request;
5 use Illuminate\Support\Facades\DB;
6
7 class MahasiswaController extends Controller
8 {
9     public function all()
10    {
11        $mahasiswas = DB::select('SELECT * FROM mahasiswas');
12        foreach ($mahasiswas as $mahasiswa) {
13            echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan <br>";
14        }
15    }
16 }
```

localhost:8000/mahasiswa/all	
←	→
C	localhost:8000/mahasiswa/all
⋮	Di
10300234 Mustofa Simanjuntak Ilmu Komputer	
10451982 Marsito Purnawati Sistem Informasi	
10980764 Ika Puspasari Ilmu Komputer	
10605319 Queen Suryatmi Sistem Informasi	
10438572 Yuliana Nurdyanti Teknik Informatika	
10737995 Tiara Siregar Ilmu Komputer	
10531551 Kambali Mulyani Ilmu Komputer	
10155818 Hesti Ramadan Teknik Informatika	
10274992 Galang Maryadi Teknik Informatika	
10228263 Mahdi Rajata Sistem Informasi	

Gambar: Hasil query DB::select('SELECT * FROM mahasiswa')

Penulisan *raw query* butuh facade class **DB**, oleh karena itu perlu tambahan perintah import `use Illuminate\Support\Facades\DB` di baris 5.

Selanjutnya di baris 11 saya menjalankan query `SELECT * FROM mahasiswa` dari method `DB::select()`. Query ini akan mengambil semua data dari tabel `mahasiswa` untuk disimpan ke dalam variabel `$mahasiswa`.

Isi variabel `$mahasiswa` kemudian masuk ke perulangan `foreach` di baris 12 – 14 untuk menampilkan kolom `nim`, `nama` dan `jurusan`. Praktek seperti ini sudah pernah kita bahas di buku **Laravel Uncover**.

Lanjut, mari coba akses hasil gabungan tabel `mahasiswa` dan tabel `nilais`.

Di dalam bahasa SQL, proses penggabungan tabel bisa dilakukan dengan beberapa cara, diantaranya menggunakan query `WHERE` atau menggunakan query `JOIN`.

Berikut contoh penggabungan tabel `mahasiswa` dan tabel `nilais` dengan query `WHERE`:

app\Http\Controllers\MahasiswaController.php

```
1 public function gabung1()
2 {
3     $mahasiswa = DB::select('SELECT * FROM mahasiswa, nilais WHERE
4                             mahasiswa.id = nilais.mahasiswa_id');
5     dump($mahasiswa);
6 }
```

Query yang dijalankan adalah `SELECT * FROM mahasiswa, nilais WHERE mahasiswa.id = nilais.mahasiswa_id`. Ini bisa di terjemahkan sebagai "ambil semua kolom yang ada di tabel `mahasiswa` dan tabel `nilais`, dengan syarat data kolom `id` di tabel `mahasiswa` harus sama dengan kolom `mahasiswa_id` di tabel `nilais`".

Karena pada saat proses generate data seeder saya membuat 5 baris untuk tabel `nilais`, maka hanya terdapat 5 mahasiswa yang memenuhi syarat penggabungan ini:

Eloquent Relationship: One to One

MariaDB [laravel]> SELECT * FROM nilais;

id	sem_1	sem_2	sem_3	mahasiswa_id	created_at	updated_at
1	2.18	2.44	3.14	9	2020-07-14 02:36:42	2020-07-14 02:36:42
2	3.13	2.35	3.01	1	2020-07-14 02:36:42	2020-07-14 02:36:42
3	3.15	2.15	3.39	8	2020-07-14 02:36:43	2020-07-14 02:36:43
4	2.85	3.50	2.20	3	2020-07-14 02:36:43	2020-07-14 02:36:43
5	3.89	2.98	3.00	4	2020-07-14 02:36:43	2020-07-14 02:36:43

5 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM mahasiswa;

id	nim	nama	jurusan	created_at	updated_at
1	10300234	Mustofa Simanjuntak	Ilmu Komputer	2020-07-14 02:36:42	2020-07-14 02:36:42
2	10451982	Marsito Purnawati	Sistem Informasi	2020-07-14 02:36:42	2020-07-14 02:36:42
3	10980764	Ika Puspasari	Ilmu Komputer	2020-07-14 02:36:42	2020-07-14 02:36:42
4	10605319	Queen Suryatmi	Sistem Informasi	2020-07-14 02:36:42	2020-07-14 02:36:42
5	10438572	Yuliana Nurdyanti	Teknik Informatika	2020-07-14 02:36:42	2020-07-14 02:36:42
6	10737995	Tiara Siregar	Ilmu Komputer	2020-07-14 02:36:42	2020-07-14 02:36:42
7	10531551	Kambali Mulyani	Ilmu Komputer	2020-07-14 02:36:42	2020-07-14 02:36:42
8	10155818	Hesti Ramadhan	Teknik Informatika	2020-07-14 02:36:42	2020-07-14 02:36:42
9	10274992	Galang Maryadi	Teknik Informatika	2020-07-14 02:36:42	2020-07-14 02:36:42
10	10228263	Mahdi Rajata	Sistem Informasi	2020-07-14 02:36:42	2020-07-14 02:36:42

Gambar: Terdapat 5 data mahasiswa yang memiliki nilai

Dari perintah `dump($mahasiswa)` terlihat bahwa hasil akhir query ini berbentuk array:

```
array:5 [▼
  0 => {#259 ▼
    +"id": 1
    +"nim": "10274992"
    +"nama": "Galang Maryadi"
    +"jurusan": "Teknik Informatika"
    +"created_at": "2020-07-14 02:36:42"
    +"updated_at": "2020-07-14 02:36:42"
    +"sem_1": "2.18"
    +"sem_2": "2.44"
    +"sem_3": "3.14"
    +"mahasiswa_id": 9
  }
  1 => {#261 ▶}
  2 => {#262 ▶}
  3 => {#263 ▶}
  4 => {#264 ▶}
]
```

Gambar: Hasil penggabungan tabel mahasiswa dan tabel nilais

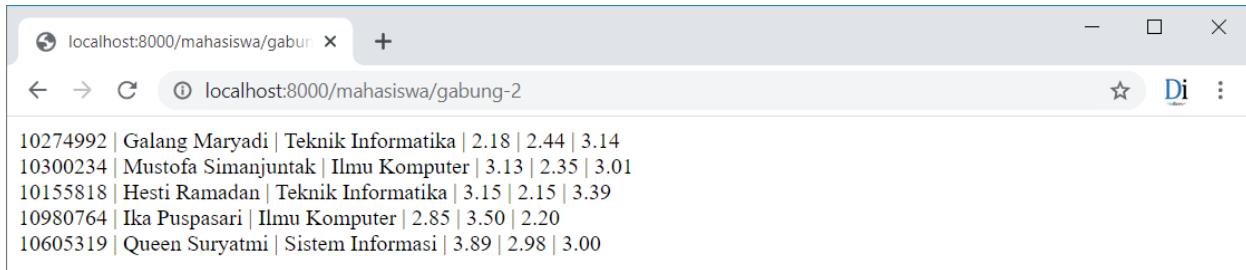
Jika ingin mengakses setiap data secara terpisah, bisa langsung di echo sebagai berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function gabung2()
2 {
3     $mahasiswa = DB::select('SELECT * FROM mahasiswa, nilais WHERE
4                             mahasiswa.id = nilais.mahasiswa_id');
5
6     foreach ($mahasiswa as $mahasiswa) {
7         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
8         echo "$mahasiswa->sem_1 | $mahasiswa->sem_2 | $mahasiswa->sem_3 <br> ";
9     }
10 }
```

Eloquent Relationship: One to One

```
9     }
10 }
```



Gambar: Hasil penggabungan tabel mahasiswas dan tabel nilais

Terlihat bahwa untuk menampilkan data hasil proses gabungan tabel, sama seperti cara biasa yang kita pakai, yakni dengan format `$variabel->nama_kolom_tabel`.

Bagaimana dengan proses gabungan dengan query JOIN? Berikut prakteknya:

```
app\Http\Controllers\MahasiswaController.php
```

```
1 public function gabungJoin1()
2 {
3     $mahasiswas = DB::select('SELECT * FROM mahasiswas JOIN nilais ON
4                             mahasiswas.id = nilais.mahasiswa_id');
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
7         echo "$mahasiswa->sem_1 | $mahasiswa->sem_2 | $mahasiswa->sem_3 <br> ";
8     }
9 }
```

Perintah query di baris 3 - 4 bisa dibaca: "ambil semua kolom di tabel mahasiswas yang di-join dengan tabel nilais, dengan syarat data kolom id di tabel mahasiswas harus sama dengan kolom `mahasiswa_id` di tabel nilais". Hasilnya akan sama seperti versi query WHERE.

Kita juga bisa menjalankan query yang lebih kompleks seperti contoh berikut:

```
app\Http\Controllers\MahasiswaController.php
```

```
1 public function gabungJoin2()
2 {
3     $mahasiswas = DB::select('SELECT * FROM mahasiswas JOIN nilais ON
4                             mahasiswas.id = nilais.mahasiswa_id WHERE jurusan = "Ilmu Komputer"');
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
7         echo "$mahasiswa->sem_1 | $mahasiswa->sem_2 | $mahasiswa->sem_3 <br> ";
8     }
9 }
```

Hasil kode program:

```
10300234 | Mustofa Simanjuntak | Ilmu Komputer | 3.13 | 2.35 | 3.01
10980764 | Ika Puspasari | Ilmu Komputer | 2.85 | 3.50 | 2.20
```

Perintah query di baris 3 – 4 bisa dibaca: "ambil semua kolom di tabel mahasiswa yang di-join dengan tabel nilais, dengan syarat data kolom id di tabel mahasiswa harus sama dengan kolom mahasiswa_id di tabel nilais. Kemudian filter hanya untuk mahasiswa dengan jurusan 'Ilmu Komputer' saja".

Hasilnya, terdapat 2 mahasiswa yang memenuhi kriteria, yakni Mustofa Simanjuntak dan Ika Puspasari.

Syarat gabungan ini juga bisa ditulis berdasarkan data tabel `nilais`:

app\Http\Controllers\MahasiswaController.php

```
1 public function gabungJoin3()
2 {
3     $mahasiswa = DB::select('SELECT * FROM mahasiswa JOIN nilais ON
4                               mahasiswa.id = nilais.mahasiswa_id WHERE nilais.sem_2 > 3');
5
6     foreach ($mahasiswa as $mahasiswa) {
7         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
8         echo "$mahasiswa->sem_1 | $mahasiswa->sem_2 | $mahasiswa->sem_3 <br> ";
9     }
10 }
```

Hasil kode program:

10980764 | Ika Puspasari | Ilmu Komputer | 2.85 | 3.50 | 2.20

Perintah query di baris 3 – 4 bisa dibaca: "ambil semua kolom di tabel mahasiswa yang di-join dengan tabel nilais, dengan syarat data kolom id di tabel mahasiswa harus sama dengan kolom mahasiswa_id di tabel nilais. Kemudian filter hanya untuk mahasiswa dengan nilai sem_2 di atas 3".

Terlihat, hanya ada 1 mahasiswa yang memenuhi kriteria, yakni "Ika Puspasari".

12.4. Menginstall Laravel Debugbar

Dengan eloquent relationship, proses penggabungan tabel nantinya akan lebih praktis dan lebih singkat dibandingkan menulis query JOIN secara manual (*raw query*). Namun kadang kita butuh melihat langsung apa query yang dijalankan oleh Laravel. **Laravel Debugbar** bisa dipakai untuk keperluan ini.

Sesuai namanya, **Laravel Debugbar** adalah sebuah package yang berguna untuk proses debugging atau pencarian kesalahan. Selain melihat query, kita juga bisa melihat isi dari variabel session dan cookie, route yang sedang diakses, file view yang dipakai, serta jumlah memory yang digunakan sebuah halaman.

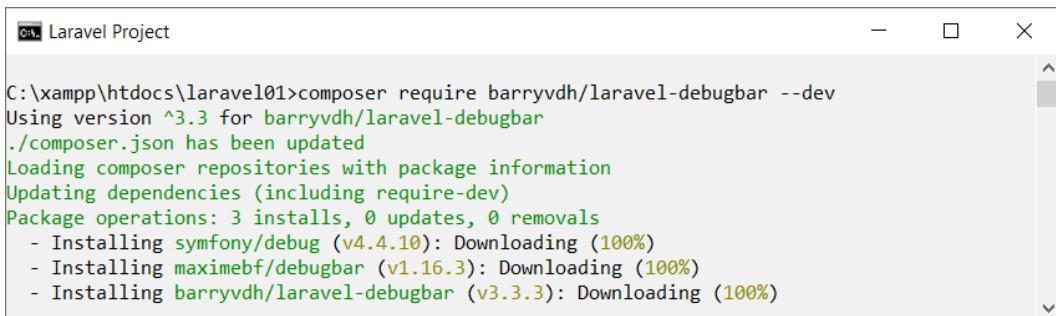
Informasi seperti ini sangat berguna, tidak saja ketika menangani error, tapi juga untuk mengoptimalkan kode yang sedang dibuat. Cukup banyak programmer Laravel yang menjadikan

Eloquent Relationship: One to One

laravel debugbar sebagai package wajib pada saat membuat sebuah project baru.

Cara instalasi laravel debugbar juga sangat mudah. Silahkan buka **cmd**, masuk ke folder instalasi laravel, lalu ketik perintah berikut dan tekan Enter:

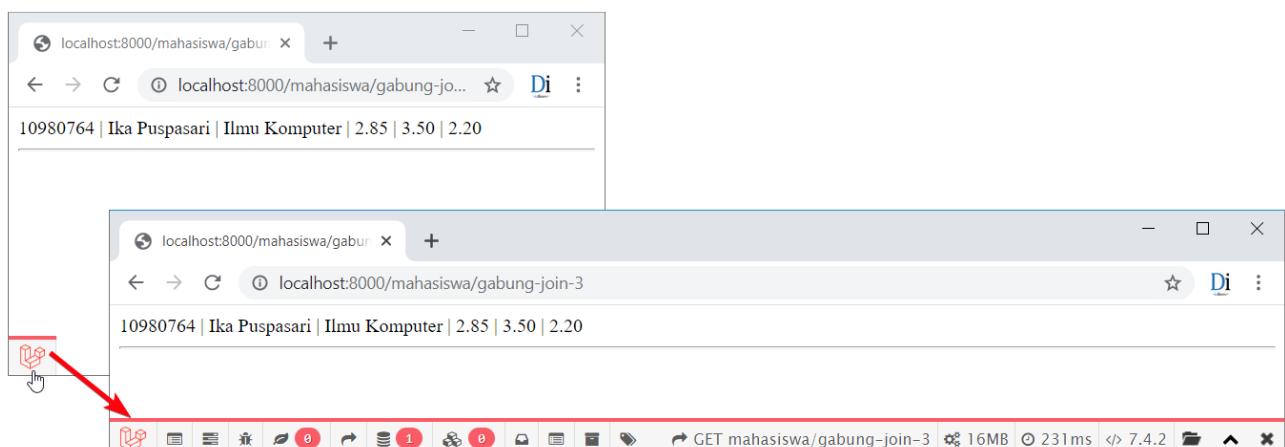
```
composer require barryvdh/laravel-debugbar --dev
```



```
C:\xampp\htdocs\laravel01>composer require barryvdh/laravel-debugbar --dev
Using version ^3.3 for barryvdh/laravel-debugbar
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 3 installs, 0 updates, 0 removals
- Installing symfony/debug (v4.4.10): Downloading (100%)
- Installing maximebf/debugbar (v1.16.3): Downloading (100%)
- Installing barryvdh/laravel-debugbar (v3.3.3): Downloading (100%)
```

Gambar: Proses instalasi Laravel Debugbar

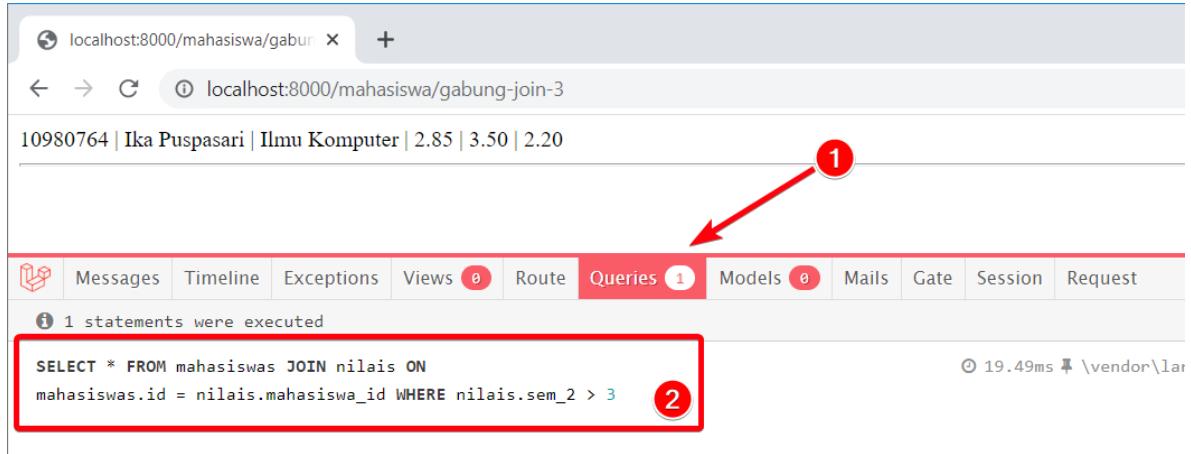
Proses instalasi akan berlangsung beberapa saat. Setelah selesai, buka web browser dan akses salah satu route yang sudah kita tulis sebelumnya. Di sudut kiri bawah akan muncul icon Laravel. Klik icon ini dan jendela laravel debugbar akan tampil memanjang di bagian bawah web browser:



Gambar: Membuka tab Laravel Debugbar

Menu laravel debugbar cukup banyak, tapi fokus utama kita kali ini adalah melihat query yang sedang berjalan. Silahkan klik icon disk di bagian tengah, atau teks 'Queries' (1) jika jendela web browser cukup lebar. Di bagian bawah (2), akan terlihat query yang dijalankan Laravel ke MySQL Server:

Eloquent Relationship: One to One

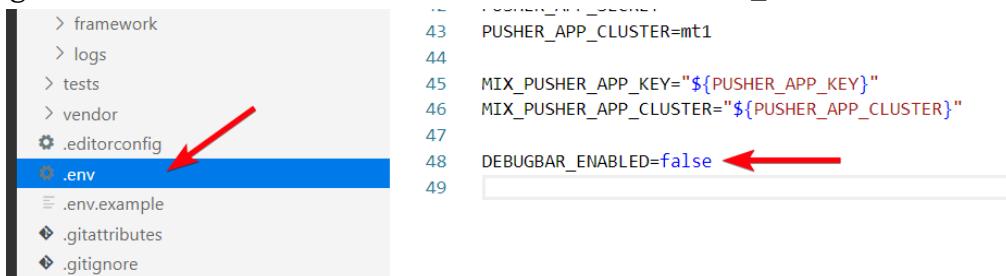


Gambar: Membuka tab Queries di Laravel Debugbar

Dalam contoh di atas saya mengakses URL `localhost:8000/mahasiswa/gabung-join-3` yang menggunakan *raw query*. Tentunya ini kurang berguna karena kita sudah tau query yang dijalankan. Namun jika menggunakan eloquent, tab Queries ini perlu diperhatikan karena menjadi kunci dari pengaksesan database.

Setelah menginstall Laravel Debugbar, maka di bagian bawah akan selalu hadir tab tambahan ini. Jika terasa mengganggu, bisa klik icon 'x' di sisi paling kanan agar tab Laravel Debugbar kembali menjadi icon kecil.

Atau jika ingin menonaktifkan sama sekali, tambah baris `DEBUGBAR_ENABLED=false` ke file `.env`:



Gambar: Cara non-aktifkan Laravel Debugbar

Alternatifnya bisa juga mengubah pengaturan `APP_DEBUG` menjadi `false` di file `.env`. Tapi ini akan ikut me-nonaktifkan seluruh tampilan error Laravel.

12.5. Eloquent Relationship `hasOne()`

Sebelumnya kita telah lihat cara penggabungan tabel dengan menulis query secara manual (*raw query*). Sekarang saatnya beralih ke eloquent *relationship*.

Hubungan antara 2 tabel sebenarnya berlaku timbal balik, yakni dari tabel utama (tempat *primary key* berada) ke tabel kedua (tempat *foreign key* berada), dan begitu juga sebaliknya, dari tabel kedua ke tabel utama.

Dalam konsep hubungan *one to one*, satu data di tabel utama **memiliki satu** (bahasa inggris: **has one**) pasangan data di tabel kedua. Sebaliknya, satu data di tabel kedua **dimiliki** (bahasa inggris: **belongs to**) satu data di tabel utama.

Untuk praktek kita, hubungan yang terjadi adalah, satu data di tabel `mahasiswa` **has one** data di tabel `nilais`. Kemudian, satu data di tabel `nilais` **belongs to** satu data di tabel `mahasiswa`.

Konsep antara *has one* dan *belongs to* ini sangat penting dipahami karena akan berhubungan dengan method yang harus ditulis ke dalam file model di Laravel. Method tersebut bernama `hasOne()` dan `belongsTo()`.

Untuk membuat hubungan "mahasiswa *has one* nilai", silahkan buka file `Mahasiswa.php`, lalu modifikasi dengan menambah kode 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     public function nilai()
12     {
13         return $this->hasOne('App\Models\Nilai');
14     }
15 }
```

Tambahan kode program ada di baris 11 – 14, yakni sebuah method bernama `nilai()` yang berisi perintah `return $this->hasOne('App\Models\Nilai');`. Inilah cara kita memberitahu Laravel apa jenis hubungan dari tabel `mahasiswa` dengan tabel `nilais`, yakni mahasiswa **has one** nilai.

Nama method `nilai()` menggunakan kata *singular* tanpa tambahan 's' karena hubungan yang terjadi adalah *one to one*. Sebagai argument dari method `$this->hasOne()`, berisi namespace file model yang terhubung, yakni '`App\Models\Nilai`'.

Dengan tambahan satu method ini, kita sudah bisa menjalankan perintah *eloquent relationship one to one* dari tabel `mahasiswa` ke tabel `nilais`.

Saat ini belum terdapat hubungan nilai *belongs to* mahasiswa, itu akan kita buat secara terpisah sesaat lagi.

Mari masuk ke praktek. Berikut daftar route yang sudah saya siapkan:

Eloquent Relationship: One to One

routes\web.php

```
1 Route::prefix('/mahasiswa')->group(function () {
2     Route::get('/find', [MahasiswaController::class, 'find']);
3     Route::get('/where', [MahasiswaController::class, 'where']);
4     Route::get('/where-chaining', [MahasiswaController::class, 'whereChaining']);
5     Route::get('/all-join', [MahasiswaController::class, 'allJoin']);
6     Route::get('/has', [MahasiswaController::class, 'has']);
7     Route::get('/where-has', [MahasiswaController::class, 'whereHas']);
8     Route::get('/doesnt-have', [MahasiswaController::class, 'doesntHave']);
9     Route::get('/where-doesnt-have', [MahasiswaController::class, 'whereDoesntHave']);
10
11    Route::get('/insert-save', [MahasiswaController::class, 'insertSave']);
12    Route::get('/insert-create', [MahasiswaController::class, 'insertCreate']);
13
14    Route::get('/update', [MahasiswaController::class, 'update']);
15    Route::get('/update-push', [MahasiswaController::class, 'updatePush']);
16    Route::get('/update-push-where', [MahasiswaController::class, 'updatePushWhere']);
17
18    Route::get('/delete-find', [MahasiswaController::class, 'deleteFind']);
19    Route::get('/delete-where', [MahasiswaController::class, 'deleteWhere']);
20    Route::get('/delete-if', [MahasiswaController::class, 'deleteIf']);
21    Route::get('/delete-cascade', [MahasiswaController::class, 'deleteCascade']);
22});
```

Terdapat 17 route yang akan di bahas. Jumlahnya memang cukup banyak karena mencakup fitur CRUD yang tersedia di *relationship one to one*.

Semua route berada di dalam *route prefixes '/mahasiswa'* di baris 1. Dengan ini, maka alamat URL akan memiliki tambahan '/mahasiswa' di bagian awal seperti /mahasiswa/find, /mahasiswa/where, dst. Ini semata-mata agar lebih rapi saja dan tidak berhubungan dengan materi relationship.

Menampilkan Satu Gabungan Data

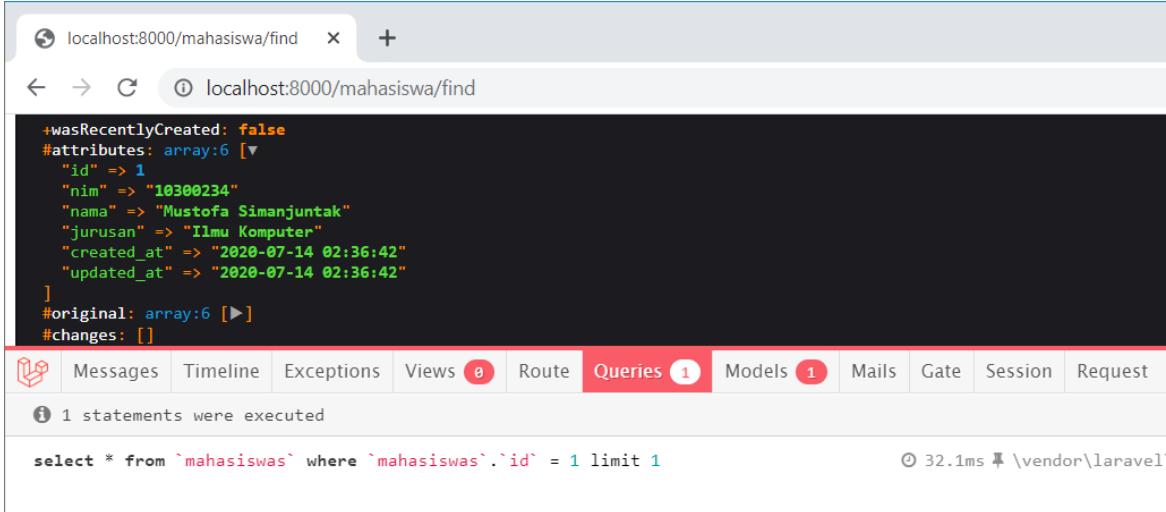
Dalam beberapa route pertama, kita akan bahas bagaimana cara menampilkan satu gabungan data antara tabel `mahasiswas` dengan tabel `nilais`.

Silahkan buka file `Mahasiswa Controller.php`, lalu tambah method berikut:

app\Http\Controllers\MahasiswaController.php

```
1 <?php
2 ...
3 use App\Models\Mahasiswa;
4 ...
5
6 public function find()
7 {
8     $mahasiswa = Mahasiswa::find(1);
9     dump($mahasiswa);
10}
```

Eloquent Relationship: One to One



```
+wasRecentlyCreated: false
#attributes: array:6 [▼
  "id" => 1
  "nim" => "10300234"
  "nama" => "Mustofa Simanjuntak"
  "jurusan" => "Ilmu Komputer"
  "created_at" => "2020-07-14 02:36:42"
  "updated_at" => "2020-07-14 02:36:42"
]
#original: array:6 [▶]
#changes: []
```

Queries 1 Models 1 Mails Gate Session Request

1 statements were executed

```
select * from `mahasiswas` where `mahasiswas`.`id` = 1 limit 1
```

Gambar: Hasil perintah dump(\$mahasiswa)

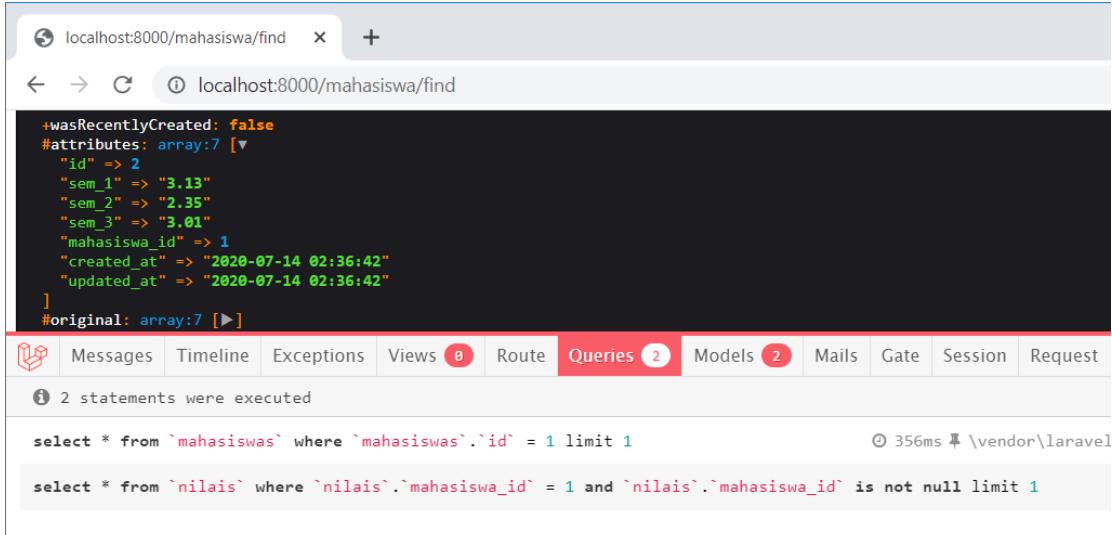
Kode di atas berisi perintah eloquent biasa dimana method `Mahasiswa::find(1)` akan mengembalikan satu object model mahasiswa dengan `id = 1`. Isi variabel `$mahasiswa` selanjutnya di `dump()` di baris 9. Isi tabel `mahasiswas` bisa terlihat di bagian `#attributes`.

Bisa diperhatikan juga dari tab Laravel Debugbar bahwa query yang sebenarnya di jalankan oleh eloquent adalah `"select * from `mahasiswas` where `mahasiswas`.`id` = 1 limit 1"`

Karena kita sudah mendefinisikan *relationship* antara model Mahasiswa dengan model Nilai menggunakan method `nilai()`, maka bisa menjalankan kode berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function find()
2 {
3     $mahasiswa = Mahasiswa::find(1);
4     dump($mahasiswa->nilai);
5 }
```



```
+wasRecentlyCreated: false
#attributes: array:7 [▼
  "id" => 2
  "sem_1" => "3.13"
  "sem_2" => "2.35"
  "sem_3" => "3.01"
  "mahasiswa_id" => 1
  "created_at" => "2020-07-14 02:36:42"
  "updated_at" => "2020-07-14 02:36:42"
]
#original: array:7 [▶]
```

Queries 2 Models 2 Mails Gate Session Request

2 statements were executed

```
select * from `mahasiswas` where `mahasiswas`.`id` = 1 limit 1
select * from `nilais` where `nilais`.`mahasiswa_id` = 1 and `nilais`.`mahasiswa_id` is not null limit 1
```

Gambar: Hasil perintah dump(\$mahasiswa->nilai)

Eloquent Relationship: One to One

Perintah `$mahasiswa->nilai` seolah-olah ingin mengakses kolom `nilai` milik tabel `mahasiswas`. Akan tetapi karena di dalam tabel `mahasiswas` tidak ditemukan kolom `nilai`, Laravel akan coba mencari apakah terdapat method `nilai()` di file model `Mahasiswa`.

Ternyata isi method `nilai()` mengembalikan perintah `return $this->hasOne('App\Models\Nilai')`, yakni sebuah perintah *relationship*. Maka Laravel akan mengembalikan satu model object `Nilai` yang terhubung dengan model `Mahasiswa` saat ini.

Silahkan buka baris `#attributes` dari hasil `dump($mahasiswa->nilai)`, itu adalah nilai milik mahasiswa dengan `id = 1`. Eloquent relationship sudah langsung menghubungkan keduanya.

Dari Laravel debugbar bisa juga terlihat 2 query yang dipakai Laravel. Query pertama untuk mengambil satu data mahasiswa dengan `id = 1`, yakni untuk perintah `Mahasiswa::find(1)`.

Sedangkan untuk perintah `$mahasiswa->nilai` di dapat dari query berikut:

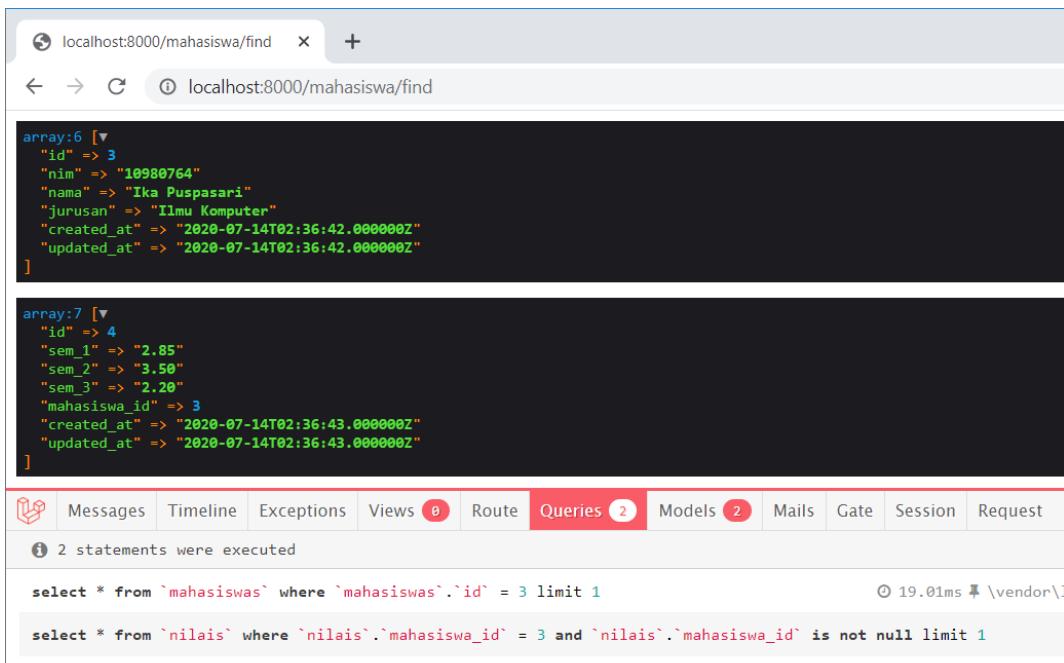
```
select * from `nilais` where `nilais`.`mahasiswa_id` = 1 and
`nilais`.`mahasiswa_id` is not null limit 1
```

Query inilah yang dipakai Laravel untuk mengambil data nilai untuk mahasiswa saat ini.

Mari kita coba untuk data mahasiswa lain:

app\Http\Controllers\MahasiswaController.php

```
1 public function find()
2 {
3     $mahasiswa = Mahasiswa::find(3);
4     dump($mahasiswa->toArray());
5     dump($mahasiswa->nilai->toArray());
6 }
```



Gambar: Hasil perintah `dump($mahasiswa->toArray())` dan `dump($mahasiswa->nilai->toArray())`;

Eloquent Relationship: One to One

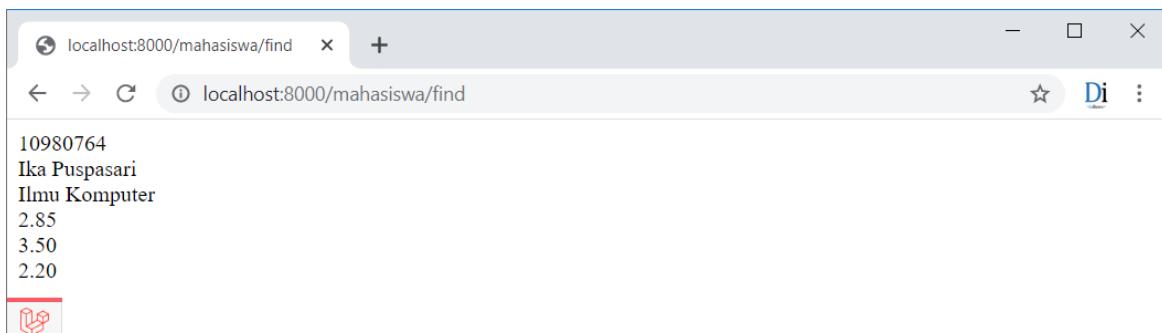
Method `Mahasiswa::find(3)` akan mencari mahasiswa dengan id = 3. Karena kita sudah mendefinisikan hubungan relationship `has one`, maka perintah `$mahasiswa->nilai` akan berisi nilai dari mahasiswa dengan id 3 tersebut.

Tambahan method `toArray()` di baris 4 dan 5 saya pakai agar tampilan data kolom langsung dalam bentuk array, tidak perlu klik baris `#attributes` lagi.

Baik, kita sudah bisa menampilkan data yang saling berhubungan dalam bentuk array. Tapi bagaimana menampilkan data per kolom? Berikut caranya:

app\Http\Controllers\MahasiswaController.php

```
1 public function find()
2 {
3     $mahasiswa = Mahasiswa::find(3);
4
5     echo $mahasiswa->nim."<br>";
6     echo $mahasiswa->nama."<br>";
7     echo $mahasiswa->jurusan."<br>";
8     echo $mahasiswa->nilai->sem_1."<br>";
9     echo $mahasiswa->nilai->sem_2."<br>";
10    echo $mahasiswa->nilai->sem_3."<br>";
11 }
```



Gambar: Menampilkan data mahasiswas dan nilais

Kode di baris 5 – 7 tidak ada masalah, itu merupakan perintah eloquent yang sering kita pakai. Yang menarik ada di baris 8 – 10, inilah cara untuk menampilkan nilai kolom yang ber-relasi. Semua kolom milik tabel `nilais` bisa diakses dari object `$mahasiswa->nilai`. Kembali, `nilai` di sini merujuk ke nama method `nilai()` yang ada di dalam model `Mahasiswa`.

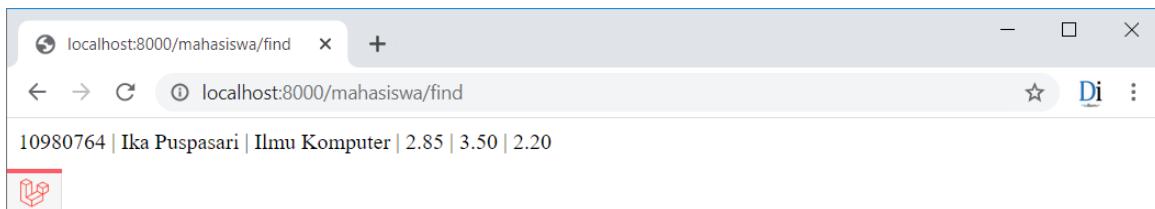
Agar lebih rapi, saya bisa tampilkan semua data dalam 1 baris saja:

app\Http\Controllers\MahasiswaController.php

```
1 public function find()
2 {
3     $mahasiswa = Mahasiswa::find(3);
4
5     echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
6     echo "{$mahasiswa->nilai->sem_1} | {$mahasiswa->nilai->sem_2} | ";
7     echo "{$mahasiswa->nilai->sem_3}";
```

Eloquent Relationship: One to One

```
8 }
```

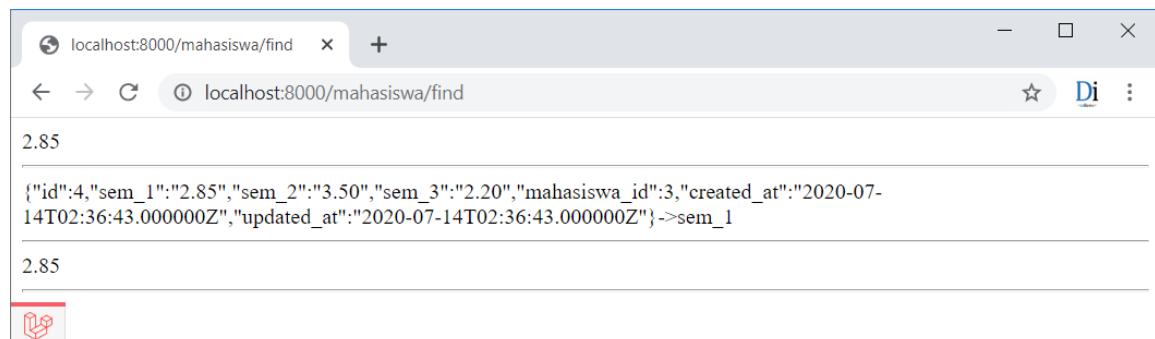


Gambar: Menampilkan hasil kolom mahasiswas dan nilai dalam satu baris

Jika ditulis seperti ini, tanda kurung kurawal di setiap komponen tabel `nilais` harus di tulis agar tidak terjadi efek *variable interpolation* (baris 6 dan 7). Berikut masalah yang dimaksud:

```
app\Http\Controllers\MahasiswaController.php
```

```
1 public function find()
2 {
3     $mahasiswa = Mahasiswa::find(3);
4
5     echo $mahasiswa->nilai->sem_1;      echo "<hr>";
6     echo "$mahasiswa->nilai->sem_1";    echo "<hr>";
7     echo "{$mahasiswa->nilai->sem_1}"; echo "<hr>";
8 }
```



Gambar: Masalah dengan efek variable interpolation

Perhatikan hasil perintah di baris 6 yang tidak menggunakan tanda kurung kurawal. Inilah efek *variable interpolation*, sebab yang akan di proses PHP adalah `"{$mahasiswa->nilai}->sem_1"`. Masalah seperti ini sering membuat bingung karena sepintas tidak ada yang salah.

Lanjut, kita juga bisa menulis berbagai perintah eloquent untuk mencari mahasiswa yang diinginkan, misalnya dengan method `where()`:

```
app\Http\Controllers\MahasiswaController.php
```

```
1 public function where()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Hesti Ramadan')->first();
4
5     echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
6     echo "{$mahasiswa->nilai->sem_1}, | {$mahasiswa->nilai->sem_2} | ";
7     echo "{$mahasiswa->nilai->sem_3}";
```

```
8 }
```

Hasil kode program:

```
10155818 | Hesti Ramadan | Teknik Informatika | 3.15, | 2.15 | 3.39
```

Dalam kode di atas saya ingin mencari mahasiswa bernama 'Hesti Ramadan', kemudian tampilkan semua data termasuk kolom yang berasal dari tabel `nilais`.

Proses penulisan method eloquent juga bisa di *chaining*. Misalnya untuk langsung mencari nilai ip semester 2 dari mahasiswa bernama 'Queen Suryatmi', bisa dengan kode berikut:

```
app\Http\Controllers\MahasiswaController.php
```

```
1 public function whereChaining()
2 {
3     $nilai = Mahasiswa::where('nama', 'Queen Suryatmi')->first()->nilai->sem_2;
4     echo $nilai; // 2.98
5 }
```

Di baris 3, hasil dari method `first()` langsung disambung dengan pengaksesan `nilai->sem_2`.

Menampilkan Banyak Gabungan Data

Apa yang kita praktekkan baru menampilkan satu gabungan data, yakni mahasiswa dengan id atau nama tertentu. Sekarang bagaimana jika ingin menampilkan semua nilai mahasiswa sekaligus?

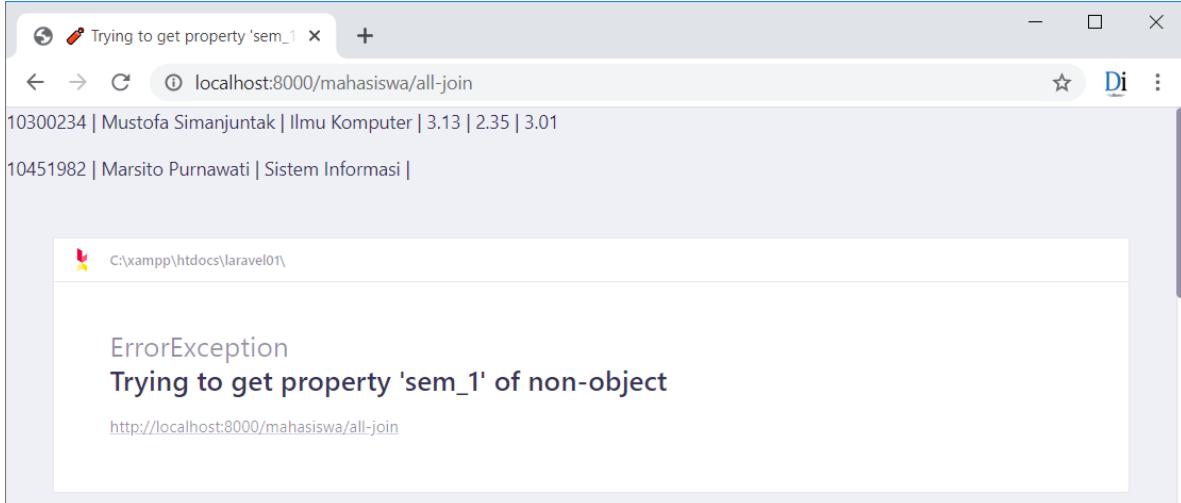
Ketika menampilkan banyak gabungan data, terdapat kemungkinan ada baris yang tidak mempunyai pasangan di tabel kedua. Dalam contoh kita, hanya ada 5 mahasiswa yang memiliki nilai, sedangkan 5 mahasiswa lain belum memiliki nilai.

Perhatikan contoh kode berikut:

```
app\Http\Controllers\MahasiswaController.php
```

```
1 public function allJoin()
2 {
3     $mahasiswas = Mahasiswa::all();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
7         echo "{$mahasiswa->nilai->sem_1} | {$mahasiswa->nilai->sem_2} | ";
8         echo "{$mahasiswa->nilai->sem_3} <hr>";
9     }
10 }
```

Eloquent Relationship: One to One



Gambar: Error pada saat mengakses property sem_1

Di baris 3 saya mengambil semua data tabel `mahasiswas` dengan perintah `Mahasiswa::all()`, lalu melakukan perulangan untuk mengakses setiap kolom tabel `mahasiswas` dan juga kolom di tabel `nilais`. Perulangan `foreach` di perlukan karena hasil dari `Mahasiswa::all()` sudah berbentuk `collection`, bukan satu data lagi.

Namun terjadi error karena tidak semua baris di tabel `mahasiswas` memiliki pasangan data di tabel `nilais`. Apa yang dihasilkan dari kode tersebut sama dengan query `LEFT JOIN` berikut:

```
SELECT nim,nama,jurusan,sem_1,sem_2,sem_3,mahasiswa_id FROM mahasiswas LEFT JOIN nilais ON mahasiswas.id = nilais.mahasiswa_id;
```

A screenshot of a MySQL command-line interface titled "MySQL Folder cmd - mysql -u root". The query executed is "SELECT nim,nama,jurusan,sem_1,sem_2,sem_3,mahasiswa_id FROM mahasiswas LEFT JOIN nilais ON mahasiswas.id = nilais.mahasiswa_id;". The result is a table with 10 rows. The columns are nim, nama, jurusan, sem_1, sem_2, sem_3, and mahasiswa_id. The data includes various student names, their majors, and their scores across three semesters. Some students have NULL values in the sem_1, sem_2, and sem_3 columns, indicating they don't have scores for those semesters.

Gambar: Hasil query LEFT JOIN

Dalam bahasa SQL, query `LEFT JOIN` berfungsi untuk menampilkan semua data gabungan meskipun tidak memiliki pasangan di tabel kedua. Nilai `NULL` yang terdapat di kolom `sem_1`, `sem_2`, `sem_3`, dan `mahasiswa_id` menjelaskan bahwa tidak ada nilai untuk mahasiswa tersebut.

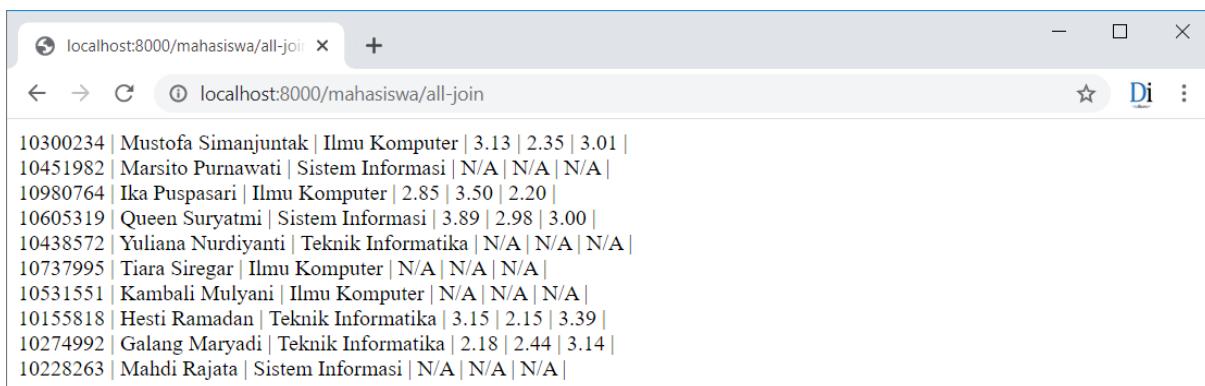
Inilah yang menyebabkan error dalam kode program kita, karena dalam perulangan `foreach` tidak semua mahasiswa memiliki nilai `$mahasiswa->nilai->sem_1`. Salah satu solusi dari

Eloquent Relationship: One to One

masalah ini adalah dengan membuat kondisi pemeriksaan nilai seperti contoh berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function allJoin()
2 {
3     $mahasiswas = Mahasiswa::all();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
7         echo $mahasiswa->nilai->sem_1 ?? 'N/A'; echo " | ";
8         echo $mahasiswa->nilai->sem_2 ?? 'N/A'; echo " | ";
9         echo $mahasiswa->nilai->sem_3 ?? 'N/A'; echo " | ";
10        echo "<br>";
11    }
12 }
```



Gambar: Tampilkan semua data mahasiswa dan nilainya

Di baris 7 – 9 saya menambah sebuah null coalescing operator menggunakan tanda tanya dua kali ' ?? '.

Perintah `$mahasiswa->nilai->sem_1 ?? 'N/A'` bisa dibaca: Jika `$mahasiswa->nilai->sem_1` berisi sesuatu, tampilkan isi dari variabel tersebut, tapi jika berisi nilai NULL maka tampilkan string 'N/A'.

Lazy Loading vs Eager Loading

Masih melanjutkan contoh sebelumnya, jika tab Queries dari Laravel Debugbar dibuka, akan terlihat tampilan berikut:

NIM	Nama	Jurusan	Nilai Semester 1	Nilai Semester 2	Nilai Semester 3
10300234	Mustofa Simanjuntak	Ilmu Komputer	3.13	2.35	3.01
10451982	Marsito Purnawati	Sistem Informasi	N/A	N/A	N/A
10980764	Ika Puspasari	Ilmu Komputer	2.85	3.50	2.20

Queries 11 | Models 15 | Mails | Gate | Session | Request

11 statements were executed

```

select * from `mahasiswas`

select * from `nilais` where `nilais`.`mahasiswa_id` = 1 and `nilais`.`mahasiswa_id` is not null limit 1

select * from `nilais` where `nilais`.`mahasiswa_id` = 2 and `nilais`.`mahasiswa_id` is not null limit 1

select * from `nilais` where `nilais`.`mahasiswa_id` = 3 and `nilais`.`mahasiswa_id` is not null limit 1

select * from `nilais` where `nilais`.`mahasiswa_id` = 4 and `nilais`.`mahasiswa_id` is not null limit 1

```

Gambar: Query yang dijalankan oleh eloquent relationship

Ternyata Laravel tidak mengambil semua data sekaligus, tapi satu per satu. Proses ini butuh 11 buah query, terdiri dari 1 query untuk mengambil semua data mahasiswa, dan 10 query untuk mengambil data `nilai` (satu untuk setiap mahasiswa).

Konsep ini dikenal sebagai **Lazy Loading**, yakni ketika perintah `Mahasiswa::all()` di jalankan, eloquent tidak akan mengambil data apapun dari tabel yang ber-relasi. Barulah saat dijalankan perintah `$mahasiswa->nilai`, satu data diambil dari tabel `nilais`. Karena di tabel mahasiswa terdapat 10 baris, maka perlu 10 buah query untuk mengambil semua pasangan nilainya.

Lazy loading sebenarnya akan menghemat resources, terutama jika kita hanya mengakses beberapa data saja. Akan tetapi jika di awal perlu mengambil banyak data yang ber-relasi, maka teknik *lazy loading* tidak lagi efisien.

Bayangkan jika terdapat 1000 mahasiswa dan 1000 nilai, maka proses ini perlu menjalankan $1 + 1000 = 1001$ query! Masalah ini dikenal sebagai **N+1 problem** dan umum terjadi di sistem ORM (Object-Relational Mapping) seperti Eloquent.

Solusinya adalah dengan memaksa eloquent untuk mengambil semua data yang ber-relasi langsung di awal. Ini dilakukan dengan menambah method `with()` seperti contoh berikut:

app\Http\Controllers\MahasiswaController.php

```

1 public function allJoin()
2 {
3     $mahasiswas = Mahasiswa::with('nilai')->get();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
7         echo $mahasiswa->nilai->sem_1 ?? 'N/A'; echo " | ";
8         echo $mahasiswa->nilai->sem_2 ?? 'N/A'; echo " | ";
9         echo $mahasiswa->nilai->sem_3 ?? 'N/A'; echo " | ";
10        echo "<br>";

```

Eloquent Relationship: One to One

```
11      }
12 }
```

The screenshot shows a browser window with the URL `localhost:8000/mahasiswa/all-join`. Below the URL is a Laravel Debugbar header with tabs: Messages, Timeline, Exceptions, Views (0), Route, **Queries (2)**, Models (15), Mails, Gate, Session, Request. The Queries tab is active, showing two statements executed:

```
select * from `mahasiswas`
select * from `nilais` where `nilais`.`mahasiswa_id` in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Gambar: Hasil dari perintah `Mahasiswa::with('nilai')->get()`

Method `with()` butuh 1 argument berupa nama method yang dipakai untuk membuat relasi. Di dalam model `Mahasiswa.php`, terdapat method `nilai()` yang berisi pendefinisian relasi `has one`. Maka string '`nilai`' inilah yang menjadi argument untuk method `with()`.

Dari laravel debugbar, terlihat jumlah query yang dijalankan sekarang hanya 2 buah, yakni satu untuk mengambil semua data dari tabel `mahasiswas`, lalu satu untuk mengambil semua data dari tabel `nilais`. Berapa pun jumlah data di tabel `nilais`, query yang diperlukan tetap hanya 2. Solusi ini dikenal dengan istilah **Eager Loading**.

Sepintas *eager loading* terlihat lebih efisien, namun jika kita hanya perlu mengakses 1 atau 2 data di tabel `nilais`, malah jadi boros resources. Jika terdapat 1000 data nilai, maka di memory server butuh ruang untuk menampung seluruh nilai-nilai ini.

Lazy loading dan *eager loading* harus di pakai tergantung situasi. Secara default, semua perintah eloquent menggunakan *lazy loading*. Jika ingin menggunakan *eager loading*, harus ditambah dengan method `with()`. Nantinya kita akan lihat beberapa contoh kasus yang pas untuk kedua konsep ini.

Menampilkan Batasan `has()` dan `whereHas()`

Kembali ke pembahasan eloquent relationship one to one. Sebelumnya kita sudah lihat bagaimana cara menampilkan data yang ber-relasi.

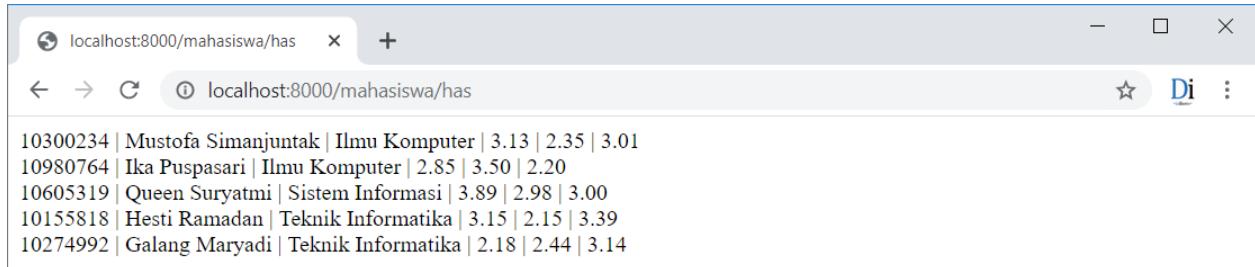
Masalah di mana terdapat baris tabel yang tidak memiliki pasangan cukup umum terjadi, karena itu Laravel menyediakan method khusus untuk membatasi hal tersebut, yakni method `has()` dan `whereHas()`.

Sebagai contoh, jika kita ingin menampilkan semua data `mahasiswas` yang memiliki pasangan di tabel `nilais`, atau dengan kata lain ingin menampilkan "data mahasiswa has nilai", bisa menggunakan perintah berikut:

Eloquent Relationship: One to One

app\Http\Controllers\MahasiswaController.php

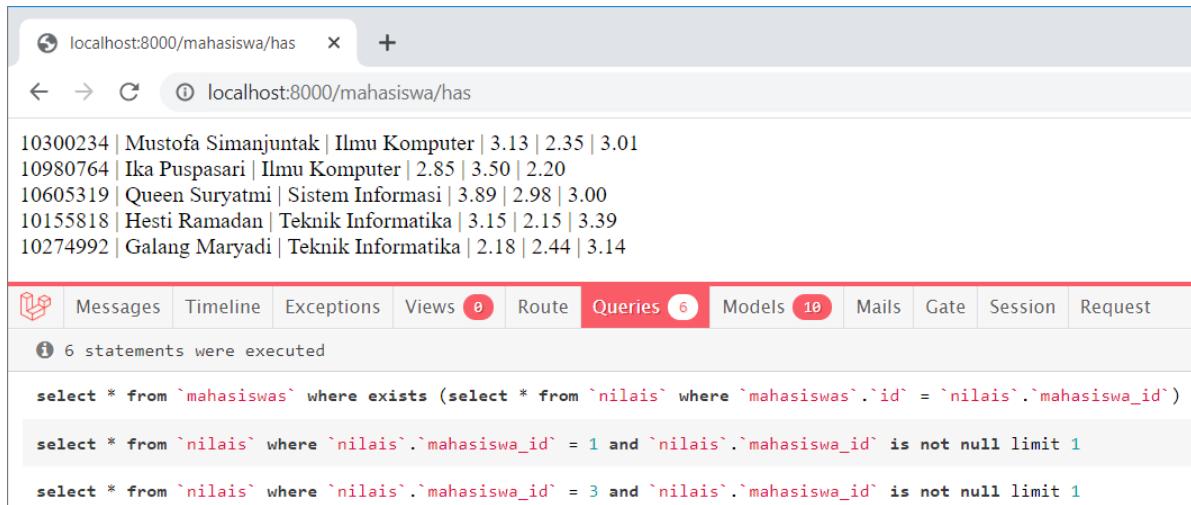
```
1 public function has()
2 {
3     $mahasiswas = Mahasiswa::has('nilai')->get();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
7         echo "{$mahasiswa->nilai->sem_1} | {$mahasiswa->nilai->sem_2} | ";
8         echo "{$mahasiswa->nilai->sem_3} <br>";
9     }
10 }
```



Gambar: Hasil perintah Mahasiswa::has('nilai')->get()

Perintah `Mahasiswa::has('nilai')->get()` langsung memberikan data mahasiswa yang memiliki pasangan di tabel `nilais`. Jika ada mahasiswa yang tidak memiliki nilai, maka tidak akan tampil.

Dengan membuka Laravel debugbar, terlihat query yang dijalankan oleh Eloquent:



Gambar: Query yang dijalankan Eloquent

Laravel mengambil hasil di atas dengan konsep himpunan melalui sub query EXIST. Selain itu perhatikan N+1 problem yang muncul, butuh 6 buah query untuk mendapatkan hasil di atas. Jika ingin menggunakan versi *eager loading*, bisa dengan perintah berikut:

Eloquent Relationship: One to One

app\Http\Controllers\MahasiswaController.php

```
1 public function has()
2 {
3     $mahasiswas = Mahasiswa::with('nilai')->has('nilai')->get();
4     foreach ($mahasiswas as $mahasiswa) {
5         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
6         echo "{$mahasiswa->nilai->sem_1} | {$mahasiswa->nilai->sem_2} | ";
7         echo "{$mahasiswa->nilai->sem_3} <br>";
8     }
9 }
```

The screenshot shows a browser window with the URL `localhost:8000/mahasiswa/has`. The page content lists student records:

ID	NIM	Nama	Jurusan	Nilai Semester 1	Nilai Semester 2	Nilai Semester 3
10300234	Mustofa Simanjuntak	Ilmu Komputer	3.13 2.35 3.01			
10980764	Ika Puspasari	Ilmu Komputer	2.85 3.50 2.20			
10605319	Queen Suryatmi	Sistem Informasi	3.89 2.98 3.00			
10155818	Hesti Ramadan	Teknik Informatika	3.15 2.15 3.39			
10274992	Galang Maryadi	Teknik Informatika	2.18 2.44 3.14			

Below the list, the query log shows:

```
select * from `mahasiswas` where exists (select * from `nilais` where `mahasiswas`.`id` = `nilais`.`mahasiswa_id`)
select * from `nilais` where `nilais`.`mahasiswa_id` in (1, 3, 4, 8, 9)
```

Gambar: Hasil penggunaan eager loading

Sekarang query yang dijalankan cukup 2 buah, tidak 6 query seperti sebelumnya.

Dalam beberapa situasi, kadang kita ingin membuat syarat tambahan, misalnya "tampilkan daftar mahasiswa yang nilai semester 1 di atas 3". Untuk membuat kondisi seperti ini, bisa menggunakan method `whereHas()`:

app\Http\Controllers\MahasiswaController.php

```
1 public function whereHas()
2 {
3     $mahasiswas = Mahasiswa::whereHas('nilai', function ($query) {
4         $query->where('sem_1', '>=', 3);
5     })->get();
6
7     foreach ($mahasiswas as $mahasiswa) {
8         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan | ";
9         echo "{$mahasiswa->nilai->sem_1} | {$mahasiswa->nilai->sem_2} | ";
10        echo "{$mahasiswa->nilai->sem_3} <br>";
11    }
12 }
```

NIM	Nama	Jurusan	Nilai Rata-Rata	Nilai Keterlambatan	Total Nilai
10300234	Mustofa Simanjuntak	Ilmu Komputer	3.13	2.35	3.01
10155818	Hesti Ramadan	Teknik Informatika	3.15	2.15	3.39
10605319	Queen Suryatmi	Sistem Informasi	3.89	2.98	3.00

```
select * from `mahasiswa` where exists (select * from `nilais` where `mahasiswa`.`id` = `nilais`.`mahasiswa_id` and `sem_1` >= 3)

select * from `nilais` where `nilais`.`mahasiswa_id` = 1 and `nilais`.`mahasiswa_id` is not null limit 1

select * from `nilais` where `nilais`.`mahasiswa_id` = 8 and `nilais`.`mahasiswa_id` is not null limit 1
```

Gambar: Hasil perintah Mahasiswa::whereHas(...)

Method `whereHas()` butuh 2 buah argument. Argument pertama diisi dengan nama method relationship, yang dalam contoh kita adalah string '`nilai`'.

Kemudian argument kedua berupa *anonymous function* berisi batasan kondisi `where()` yang harus dipenuhi oleh tabel `nilais`. Penulisan seperti ini pernah kita buat saat membahas method `orWhere()` di bab sebelumnya.

Jika ingin menggunakan versi eager loading, bisa dengan kode berikut:

```
1 $mahasiswa = Mahasiswa::with('nilai')->whereHas('nilai', function ($query) {
2     $query->where('sem_1', '>=', 3);
3 })->get();
```

Sebagai tambahan, Eloquent juga memiliki method `orWhereHas()` untuk membuat logika `or` di dalam himpunan `EXIST`.

Menampilkan Batasan `doesntHave()` dan `whereDoesntHave()`

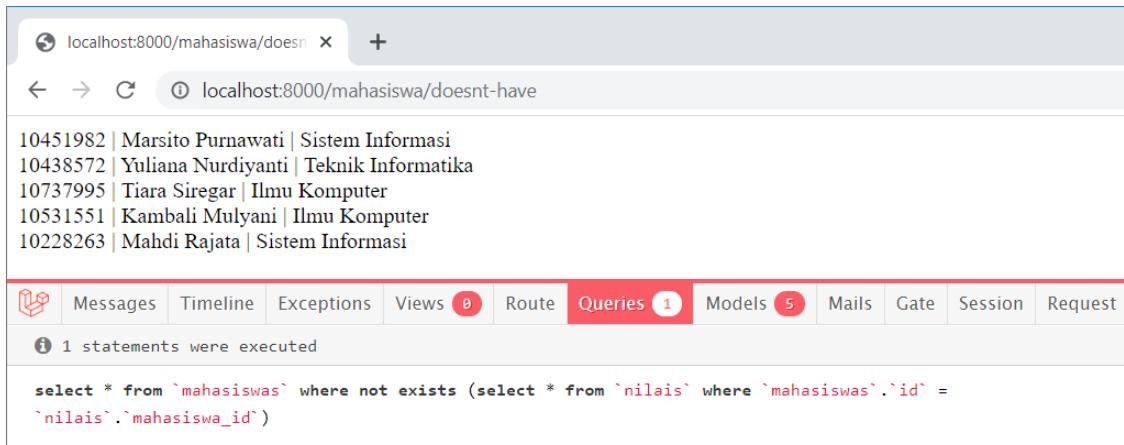
Kedua method ini merupakan kebalikan dari method `has()` dan `whereHas()`. Method `doesntHave()` dan `whereDoesntHave()` dipakai untuk menampilkan data yang **tidak** memiliki pasangan di tabel kedua.

Sebagai contoh, untuk menampilkan semua mahasiswa yang tidak memiliki nilai, bisa dengan kode berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function doesntHave()
2 {
3     $mahasiswa = Mahasiswa::doesntHave('nilai')->get();
4
5     foreach ($mahasiswa as $mahasiswa) {
6         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan <br>";
7     }
8 }
```

Eloquent Relationship: One to One



The screenshot shows the Laravel Artisan log at `localhost:8000/mahasiswa/doesnt-have`. It displays a list of student records and the executed SQL query.

```
10451982 | Marsito Purnawati | Sistem Informasi
10438572 | Yuliana Nurdyanti | Teknik Informatika
10737995 | Tiara Siregar | Ilmu Komputer
10531551 | Kambali Mulyani | Ilmu Komputer
10228263 | Mahdi Rajata | Sistem Informasi

1 statements were executed
select * from `mahasiswa` where not exists (select * from `nilais` where `mahasiswa`.`id` =
`nilais`.`mahasiswa_id`)
```

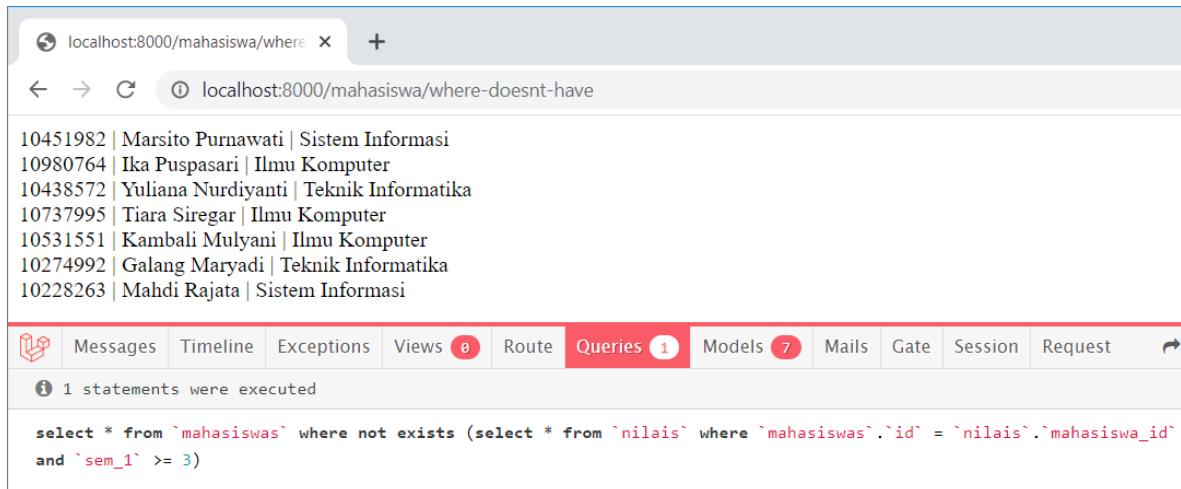
Gambar: Hasil perintah `Mahasiswa::doesntHave('nilai')->get()`

Terlihat query yang dijalankan Laravel adalah `select * from `mahasiswa` where not exists (...).`

Untuk membuat pembatasan lanjutan, tersedia juga method `whereDoesntHave()`. Sebagai contoh jika saya ingin menampilkan semua mahasiswa yang tidak memiliki nilai **dan** juga tidak memiliki nilai ipk semester 1 di atas 3, bisa menggunakan perintah berikut:

`app\Http\Controllers\MahasiswaController.php`

```
1 public function whereDoesntHave()
2 {
3     $mahasiswa = Mahasiswa::whereDoesntHave('nilai', function ($query) {
4         $query->where('sem_1', '>=', 3);
5     })->get();
6
7     foreach ($mahasiswa as $mahasiswa) {
8         echo "$mahasiswa->nim | $mahasiswa->nama | $mahasiswa->jurusan <br>";
9     }
10 }
```



The screenshot shows the Laravel Artisan log at `localhost:8000/mahasiswa/where-doesnt-have`. It displays a list of student records and the executed SQL query.

```
10451982 | Marsito Purnawati | Sistem Informasi
10980764 | Ika Puspasari | Ilmu Komputer
10438572 | Yuliana Nurdyanti | Teknik Informatika
10737995 | Tiara Siregar | Ilmu Komputer
10531551 | Kambali Mulyani | Ilmu Komputer
10274992 | Galang Maryadi | Teknik Informatika
10228263 | Mahdi Rajata | Sistem Informasi

1 statements were executed
select * from `mahasiswa` where not exists (select * from `nilais` where `mahasiswa`.`id` =
`nilais`.`mahasiswa_id` and `sem_1` >= 3)
```

Gambar: Hasil perintah `Mahasiswa::whereDoesntHave('nilai')->get()`

Cara penggunaan method `whereDoesntHave()` ini mirip seperti method `whereHas()`, yakni dengan menggunakan *anonymous function*. Jika kita ingin membuat gabungan logika **or**, tersedia juga method `orWhereDoesntHave()`.

Menginput Data Relationship

Sekarang kita masuk ke cara menginput data ke dalam tabel yang saling ber-relasi.

Sebenarnya kedua tabel tetap bisa di input terpisah. Tabel `mahasiswa` tidak ada masalah, kita bebas menginput mahasiswa baru. Namun untuk tabel `nilais`, data baru hanya bisa diinput dengan syarat kolom `mahasiswa_id` harus merujuk ke `id` mahasiswa yang ada di tabel `mahasiswa`.

Karena terdapat hubungan *relationship one to one*, maka data baru untuk tabel `nilais` hanya bisa diinput untuk satu mahasiswa saja. Kita tidak bisa menginput lebih dari satu nilai untuk mahasiswa yang sama.

Yang sering dilakukan adalah menginput data mahasiswa baru beserta nilainya sekaligus. Berikut cara yang bisa dipakai:

app\Http\Controllers\MahasiswaController.php

```
1 public function insertSave()
2 {
3     $mahasiswa = new Mahasiswa;
4     $mahasiswa->nim = '19005011';
5     $mahasiswa->nama = 'Riana Putria';
6     $mahasiswa->jurusan = 'Ilmu Komputer';
7     $mahasiswa->save();
8
9     $nilai = new \App\Models\Nilai;
10    $nilai->sem_1 = 3.12;
11    $nilai->sem_2 = 3.23;
12    $nilai->sem_3 = 3.34;
13
14    $mahasiswa->nilai()->save($nilai);
15    echo "Penambahan $mahasiswa->nama ke database berhasil";
16 }
```

Di baris 3 saya membuat sebuah object model Mahasiswa yang disimpan ke variabel `$mahasiswa`. Ke dalamnya di input nilai `nim`, `nama` dan `jurusan` antara baris 4 – 6. Mahasiswa ini kemudian disimpan ke dalam database dengan perintah `$mahasiswa->save()`.

Setelah data mahasiswa tersimpan, antara baris 9 – 12 saya juga membuat object model Nilai ke dalam variabel `$nilai` dan mengisi data kolom `sem_1`, `sem_2` dan `sem_3`. Idenya adalah, isi variabel `$nilai` akan menjadi nilai dari mahasiswa 'Riana Putria' yang baru saja kita buat.

Pertanyaannya, bagaimana cara menghubungkan nilai ini dengan mahasiswa tersebut? Secara teori, kita perlu menginput kolom `id_mahasiswa` milik variabel `$nilai` seperti ini:

Eloquent Relationship: One to One

```
$nilai->mahasiswa_id = <id mahasiswa 'Riana Putria'>;
```

Masalahnya, kita tidak tau berapa `id` mahasiswa 'Riana Putria'. Kolom `id` untuk tabel `mahasiswas` di generate secara otomatis ketika menjalankan perintah `$mahasiswa->save()`.

Di sinilah *eloquent relationship* menjalankan fungsinya. Cukup dengan menulis perintah `$mahasiswa->nilai()->save($nilai)` seperti di baris 14. Perintah ini secara otomatis mengisi kolom `mahasiswa_id` dengan `id` yang tersimpan di object `$mahasiswa`.

Perhatikan bahwa pemanggilan method diawali dari variabel `$mahasiswa`, lalu di sambung dengan method `nilai()` dan diakhiri oleh method `save($nilai)`.

Silahkan akses URL `localhost:8000/mahasiswa/insert-save` dan lihat isi dari tab Queries laravel debugbar:

The screenshot shows the Laravel Debugbar interface. The top navigation bar includes links for Messages, Timeline, Exceptions, Views (0), Route, **Queries (2)**, Models (0), Mails, Gate, Session, and Request. The 'Queries' tab is active, displaying the following log:

```
2 statements were executed
insert into `mahasiswas` (`nim`, `nama`, `jurusan`, `updated_at`, `created_at`) values ('19005011', 'Riana Putria', 'Ilmu Komputer', '2020-07-23 05:09:29', '2020-07-23 05:09:29')
insert into `nilais` (`sem_1`, `sem_2`, `sem_3`, `mahasiswa_id`, `updated_at`, `created_at`) values (3.12, 3.23, 3.34, 11, '2020-07-23 05:09:30', '2020-07-23 05:09:30')
```

A red arrow points to the second query, which inserts data into the 'nilais' table.

Gambar: Proses insert data

Ada 2 buah query `INSERT` yang dijalankan Eloquent. Pertama untuk menginput data mahasiswa, dan kedua untuk menginput data nilai. Dengan menggunakan perintah `$mahasiswa->nilai()->save($nilai)`, secara otomatis Laravel akan mengisi kolom `id_mahasiswa` di tabel `nilais` sesuai dengan `id` mahasiswa yang baru saja di input, yakni 11.

Untuk memastikan, bisa cek langsung ke dalam database:

```
MariaDB [laravel]> SELECT * FROM mahasiswas WHERE id = 11;
+----+-----+-----+-----+-----+
| id | nim   | nama    | jurusan | created_at      | updated_at      |
+----+-----+-----+-----+-----+
| 11 | 19005011 | Riana Putria | Ilmu Komputer | 2020-07-23 05:09:29 | 2020-07-23 05:09:29 |
+----+-----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM nilais WHERE mahasiswa_id = 11;
+----+-----+-----+-----+-----+
| id | sem_1 | sem_2 | sem_3 | mahasiswa_id | created_at      | updated_at      |
+----+-----+-----+-----+-----+
| 6  | 3.12  | 3.23  | 3.34  | 11          | 2020-07-23 05:09:30 | 2020-07-23 05:09:30 |
+----+-----+-----+-----+-----+
1 row in set (0.002 sec)
```

Gambar: Isi tabel mahasiswas dan nilais untuk mahasiswa dengan id = 11

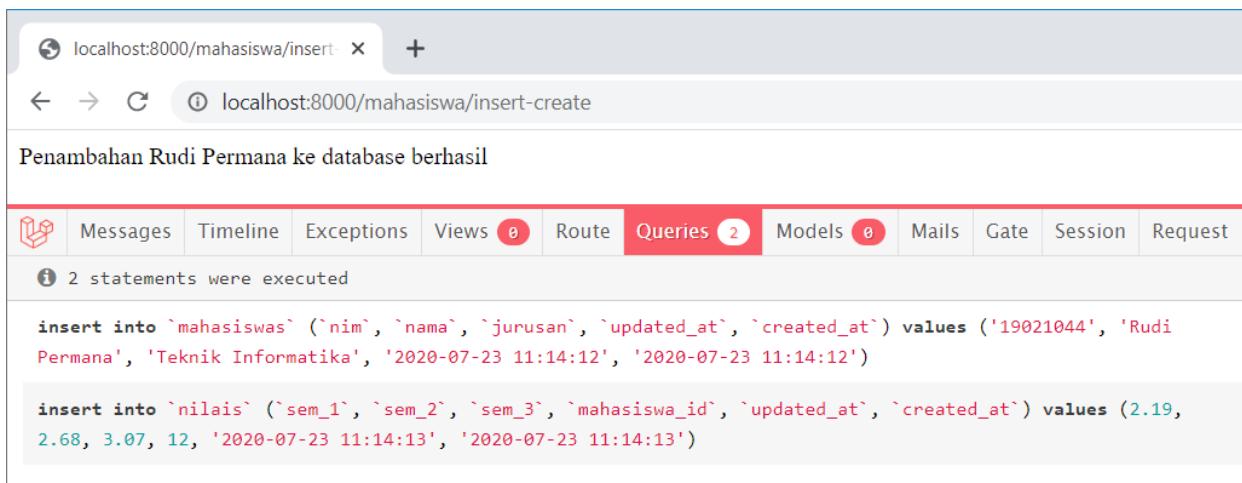
Eloquent Relationship: One to One

Terlihat kolom `mahasiswa_id` di tabel `nilais` berisi angka yang sama dengan kolom `id` dari mahasiswa 'Riana Putria'.

Cara lain untuk menambah data baru ke kedua tabel adalah menggunakan teknik `mass assignment`. Agar bisa berjalan, silahkan tambah baris `protected $guarded = []` ke dalam model Mahasiswa dan model Nilai. Setelah itu jalankan kode berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function insertCreate()
2 {
3     $mahasiswa = Mahasiswa::create(
4         [
5             'nim' => '19021044',
6             'nama' => 'Rudi Permana',
7             'jurusan' => 'Teknik Informatika',
8         ]
9     );
10
11    $mahasiswa->nilai()->create([
12        'sem_1' => 2.19,
13        'sem_2' => 2.68,
14        'sem_3' => 3.07,
15    ]);
16
17    echo "Penambahan $mahasiswa->nama ke database berhasil";
18 }
```



Gambar: Proses insert data dengan mass assignment

Pada dasarnya cara yang kita pakai sama seperti sebelumnya, hanya saja sekarang tidak lagi menggunakan method `save()`, tapi `create()`. Semua nilai kolom diinput sebagai array ke dalam method `create()`.

Agar nilai kolom `mahasiswa_id` langsung terisi, method yang dijalankan adalah `$mahasiswa->nilai()->create(...)` seperti yang ada di baris 11 – 15.

Untuk memastikan, silahkan cek isi tabel `mahasiswas` dan tabel `nilais`, pastikan kolom

Eloquent Relationship: One to One

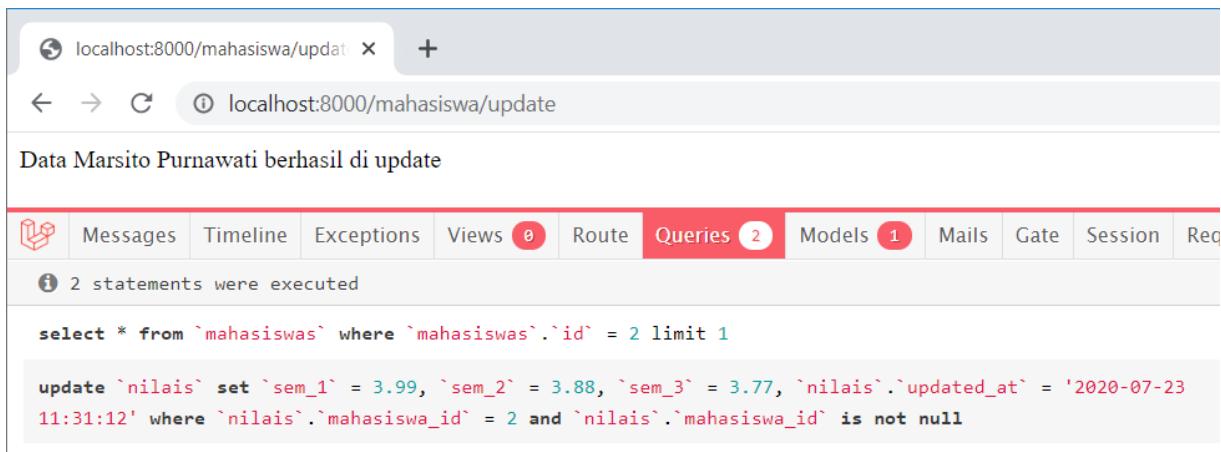
`mahasiswa_id` di tabel `nilais` merujuk ke `id` dari mahasiswa 'Rudi Permana'.

Mengupdate Data Relationship

Eloquent relationship juga menyediakan beberapa cara untuk mengupdate data yang saling terhubung. Cara pertama bisa lewat method `update()` seperti contoh berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function updateMassAssignment()
2 {
3     $mahasiswa = Mahasiswa::find(2);
4
5     $mahasiswa->nilai()->update([
6         'sem_1' => 3.99,
7         'sem_2' => 3.88,
8         'sem_3' => 3.77,
9     ]);
10 }
```



Gambar: Proses update data dengan method `update()`

Di baris 3 saya mencari data yang ingin di update, yakni mahasiswa dengan `id` = 2. Hasil dari perintah `Mahasiswa::find(2)` kemudian disimpan ke dalam variabel `$mahasiswa`. Di sini variabel `$mahasiswa` sudah berisi satu object model `Mahasiswa`.

Proses update sendiri mirip seperti pada pemanggilan method `create()` sebelumnya, yakni diakses dari perintah `$mahasiswa->nilai()->update([...])`. Nilai yang akan di update ditulis dalam bentuk array.

Dengan perintah ini, isi tabel `nilais` untuk mahasiswa dengan `id` = 2 sudah berubah, sesuai dengan array yang ada di baris 6 – 8.

Alternatif cara update kedua adalah menggunakan method `push()`. Berikut contoh penggunaannya:

Eloquent Relationship: One to One

app\Http\Controllers\MahasiswaController.php

```
1 public function updatePush()
2 {
3     $mahasiswa = Mahasiswa::find(9);
4
5     $mahasiswa->nilai->sem_1 = 2.44;
6     $mahasiswa->nilai->sem_2 = 2.55;
7     $mahasiswa->nilai->sem_3 = 2.66;
8
9     $mahasiswa->push();
10    echo "Update nilai $mahasiswa->nama berhasil";
11 }
```



Gambar: Proses update data dengan method push()

Mirip seperti sebelumnya, di baris 3 saya kembali mengisi variabel \$mahasiswa dengan object model Mahasiswa yang memiliki id = 9.

Pada baris 5 – 7, proses update untuk tabel nilai langsung diakses dari variabel \$mahasiswa ini dengan cara mengisi \$mahasiswa->nilai->sem_1, \$mahasiswa->nilai->sem_2 dan \$mahasiswa->nilai->sem_3. Agar semua nilai di update ke database, jalankan perintah \$mahasiswa->push() seperti di baris 9.

Berikut contoh lain dari penggunaan method push():

app\Http\Controllers\MahasiswaController.php

```
1 public function updatePushWhere()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Hesti Ramadan')->first();
4     $mahasiswa->nilai->sem_1 = 2.44;
5     $mahasiswa->nilai->sem_2 = 2.55;
6     $mahasiswa->nilai->sem_3 = 2.66;
7
8     $mahasiswa->push();
9     echo "Data $mahasiswa->nama berhasil di update";
10 }
```

```
localhost:8000/mahasiswa/update-push-where
Data Hesti Ramadan berhasil di update

Messages Timeline Exceptions Views 0 Route Queries 3 Models 2 Mails Gate Session Request
3 statements were executed
select * from `mahasiswas` where `nama` = 'Hesti Ramadan' limit 1
select * from `nilais` where `nilais`.`mahasiswa_id` = 8 and `nilais`.`mahasiswa_id` is not null limit 1
update `nilais` set `sem_1` = 2.44, `sem_2` = 2.55, `sem_3` = 2.66, `nilais`.`updated_at` = '2020-07-24 00:15:26' where `id` = 3
```

Gambar: Proses update data dengan method push()

Perbedaan dari contoh sebelumnya hanya di baris 3, yakni dari cara mencari mahasiswa yang ingin di update. Kali ini saya ingin mengupdate nilai untuk mahasiswa yang bernama 'Hesti Ramadan'.

Menghapus Data Relationship

Dalam teori database, terdapat konsep yang disebut *referential integrity*. **Referential integrity** adalah sekumpulan aturan untuk menjaga konsistensi data yang saling berelasi. Aturan ini secara tidak langsung sudah kita lihat dari praktek sebelumnya, yakni:

- Satu mahasiswa hanya bisa memiliki satu nilai, dan satu nilai hanya bisa dimiliki oleh satu mahasiswa (aturan *one to one relationship*).
- Sebuah data nilai tidak bisa dibuat tanpa mengisi kolom `id_mahasiswa`. Dengan kata lain, setiap nilai harus 'melekat' ke satu mahasiswa.

Ketika akan menghapus data, maka terdapat aturan lain:

- Data mahasiswa hanya bisa dihapus jika tidak ada nilai yang 'melekat' ke mahasiswa tersebut. Jika ingin menghapus satu data mahasiswa, nilainya harus dihapus terlebih dahulu. Namun tidak masalah jika ingin menghapus data nilai tanpa harus menghapus pasangan mahasiswanya.

Mari kita uji aturan ini:

app\Http\Controllers\MahasiswaController.php

```
1 public function deleteFind()
2 {
3     $mahasiswa = Mahasiswa::find(1);
4     $mahasiswa->delete();
5 }
```

Eloquent Relationship: One to One



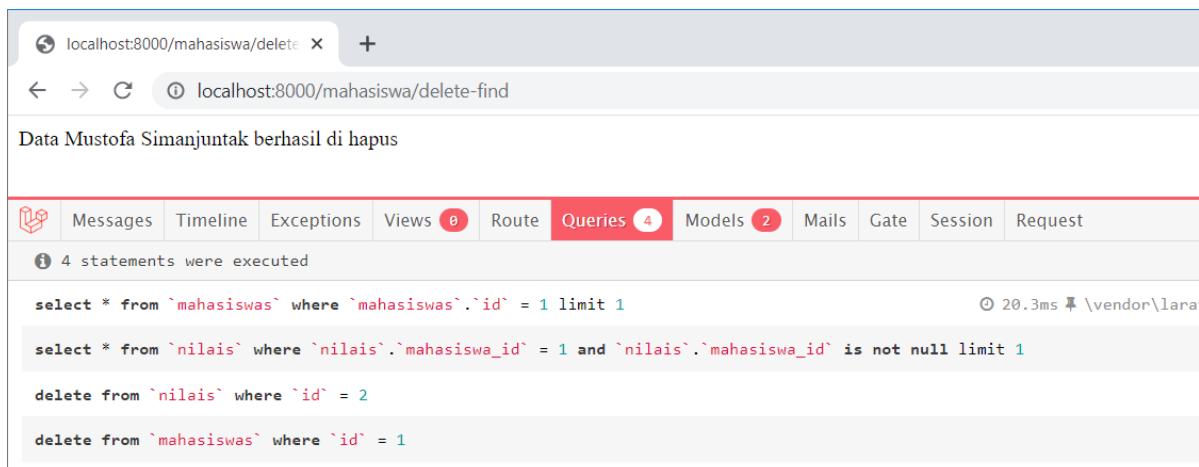
Gambar: Error integrity constraint violation

Di sini saya coba menghapus mahasiswa dengan id = 1. Hasilnya terdapat error dengan pesan "Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails". Ini merupakan pesan error dari MySQL karena menyalahi aturan referential integrity.

Mahasiswa dengan id = 1 masih memiliki data di tabel `nilais`. Maka untuk bisa menghapus mahasiswa ini, nilainya harus dihapus terlebih dahulu:

app\Http\Controllers\MahasiswaController.php

```
1 public function deleteFind()
2 {
3     $mahasiswa = Mahasiswa::find(1);
4     $mahasiswa->nilai->delete();
5     $mahasiswa->delete();
6
7     echo "Data $mahasiswa->nama berhasil di hapus";
8 }
```



Gambar: Proses penghapusan data mahasiswa berhasil

Sebelum menghapus mahasiswa dengan perintah `$mahasiswa->delete()`, saya menghapus

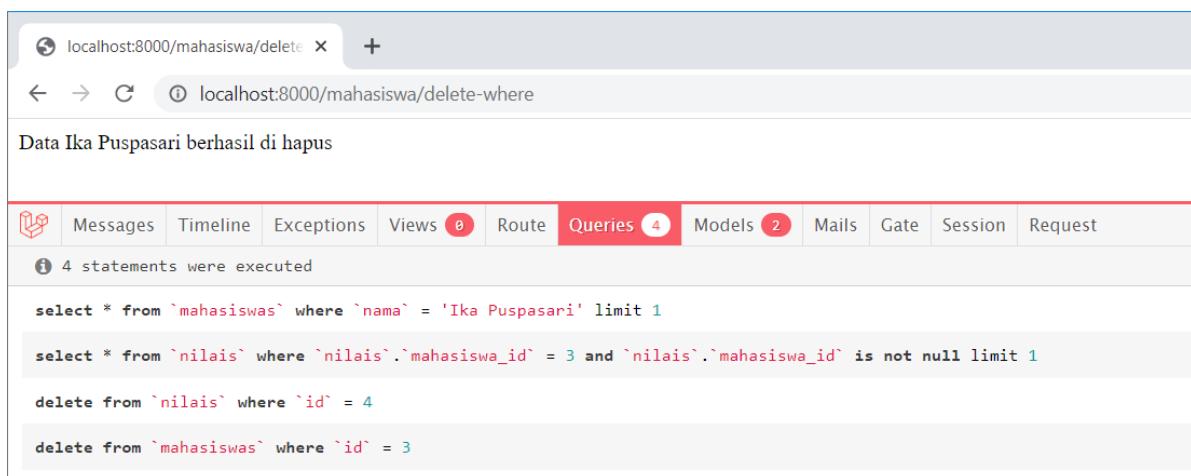
Eloquent Relationship: One to One

nilai dari mahasiswa tersebut dengan perintah `$mahasiswa->nilai->delete()` di baris 4. Dengan urutan seperti ini, data mahasiswa sukses dihapus dan tidak terjadi error *referential integrity*.

Jika kita ingin menghapus mahasiswa berdasarkan nama, bisa tambah method `where()` untuk mencari mahasiswa yang diinginkan:

app\Http\Controllers\MahasiswaController.php

```
1 public function deleteWhere()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Ika Puspasari')->firstOrFail();
4     $mahasiswa->nilai->delete();
5     $mahasiswa->delete();
6
7     echo "Data $mahasiswa->nama berhasil di hapus";
8 }
```



Gambar: Menghapus mahasiswa berdasarkan nama

Selain menambah method `where()`, saya juga mengganti method `first()` dengan `firstOrFail()`. Ini dipakai sebagai antisipasi jika nama tidak ditemukan.



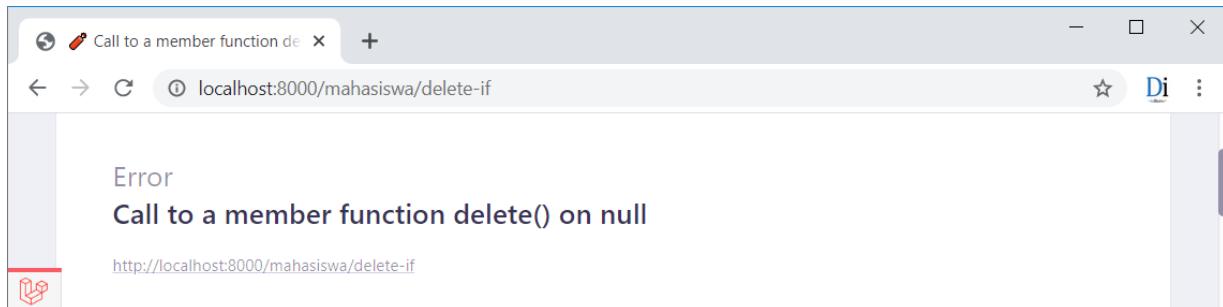
Gambar: Halaman 404 karena mahasiswa tidak ditemukan

Dengan method `firstOrFail()`, maka jika nama mahasiswa tidak ditemukan akan tampil halaman 404, bukan lagi halaman error jika menggunakan method `first()` saja.

Kita sudah bisa menghapus data mahasiswa dan nilai. Akan tetapi sebenarnya masih ada satu lagi masalah yang muncul. Bisakah anda tebak apa penyebab error dari kode program berikut?

app\Http\Controllers\MahasiswaController.php

```
1 public function deleteIf()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Yuliana Nurdiyanti')->firstOrFail();
4     $mahasiswa->nilai->delete();
5     $mahasiswa->delete();
6
7     echo "Data $mahasiswa->nama berhasil di hapus";
8 }
```



Gambar: Error Call to a member function delete() on null

Saya bisa pastikan mahasiswa 'Yuliana Nurdiyanti' ada di dalam tabel `mahasiswas`. Karena jika tidak ada, yang tampil adalah halaman 404 hasil dari `firstOrFail()`.

Sumber masalah ada di baris 4, yakni proses penghapusan tabel `nilais`. Mahasiswa 'Yuliana Nurdiyanti' ternyata tidak memiliki nilai. Akibatnya, perintah `$mahasiswa->nilai->delete()` akan mencoba menghapus sesuatu dari nilai `NULL`.

Untuk mengatasi masalah ini, kita bisa buat sebuah pemeriksaan kondisi if terlebih dahulu. Jika mahasiswa yang ingin dihapus memiliki nilai, hapus nilai tersebut. Namun jika mahasiswa tidak memiliki nilai, tidak perlu lakukan proses penghapusan.

Berikut modifikasi kode program sebelumnya dengan penambahan kondisi if:

app\Http\Controllers\MahasiswaController.php

```
1 public function deleteIf()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Yuliana Nurdiyanti')->firstOrFail();
4     if (!empty($mahasiswa->nilai)){
5         $mahasiswa->nilai->delete();
6     }
7
8     $mahasiswa->delete();
9
10    echo "Data $mahasiswa->nama berhasil di hapus";
11 }
```

Eloquent Relationship: One to One

```
localhost:8000/mahasiswa/delete x +
← → ⌂ ⓘ localhost:8000/mahasiswa/delete-if
Data Yuliana Nurdyanti berhasil dihapus
Messages Timeline Exceptions Views 0 Route Queries 3 Models 1 Mails Gate Session Request
3 statements were executed
select * from `mahasiswas` where `nama` = 'Yuliana Nurdyanti' limit 1
select * from `nilais` where `nilais`.`mahasiswa_id` = 5 and `nilais`.`mahasiswa_id` is not null limit 1
delete from `mahasiswas` where `id` = 5
```

Gambar: Menghapus data mahasiswa yang tidak memiliki nilai

Perintah `$mahasiswa->nilai->delete()` sekarang berada di dalam blok `if (!empty($mahasiswa->nilai))`. Artinya, proses penghapusan nilai hanya dilakukan jika `$mahasiswa->nilai` berisi sesuatu.

Alternatif cara lain untuk mempermudah proses penghapusan adalah dengan mengaktifkan mode `ON DELETE CASCADE` milik MySQL. Jika fitur ini aktif, MySQL otomatis akan menghapus data di tabel nilai jika dibutuhkan.

Untuk mengaktifkan fitur ini, kita harus modifikasi ulang struktur tabel `nilais`, yakni dengan menambah method `onDelete('cascade')` ke dalam pendefinisian kolom `mahasiswa_id`. Silahkan buka kembali file migration tabel `nilais`, lalu modifikasi dari sebelumnya:

database\migrations\<timestamp>_create_nilais_table.php

```
1 public function up()
2 {
3     Schema::create('nilais', function (Blueprint $table) {
4         ...
5         $table->foreignId('mahasiswa_id')->unique()->constrained();
6         ...
7     });
8 }
```

Menjadi:

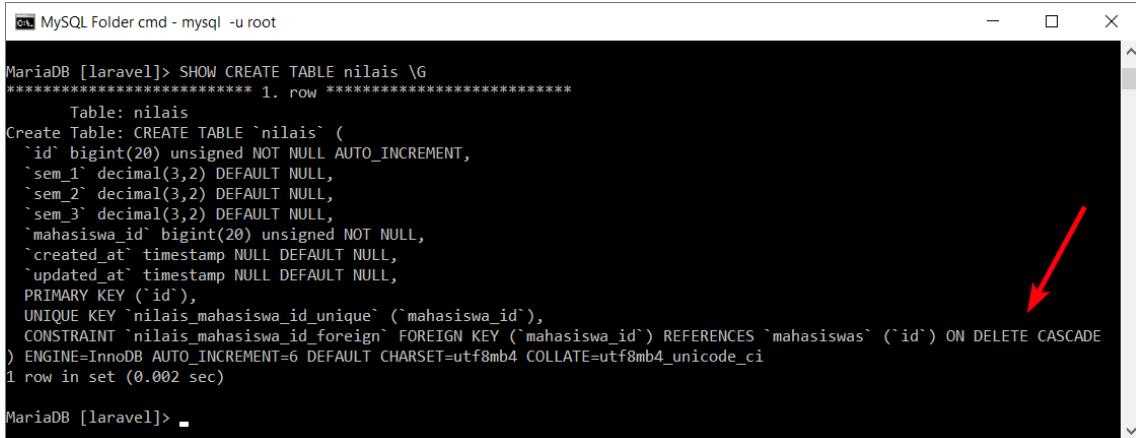
database\migrations\<timestamp>_create_nilais_table.php

```
1 public function up()
2 {
3     Schema::create('nilais', function (Blueprint $table) {
4         ...
5         $table->foreignId('mahasiswa_id')->unique()->constrained()
6             ->onDelete('cascade');
7         ...
8     });
9 }
```

Eloquent Relationship: One to One

Agar tambahan ini bisa efektif, generate ulang file migration dan juga seeder dengan perintah
php artisan migrate:fresh --seed.

Untuk memastikan fitur *on delete cascade* sudah aktif, jalankan query SHOW CREATE TABLE nilais \G di MySQL client, seharusnya akan terlihat tambahan "ON DELETE CASCADE" pada baris CONSTRAINT:



```
MariaDB [laravel]> SHOW CREATE TABLE nilais \G
***** 1. row *****
  Table: nilais
Create Table: CREATE TABLE `nilais` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `sem_1` decimal(3,2) DEFAULT NULL,
  `sem_2` decimal(3,2) DEFAULT NULL,
  `sem_3` decimal(3,2) DEFAULT NULL,
  `mahasiswa_id` bigint(20) unsigned NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `nilais_mahasiswa_id_unique` (`mahasiswa_id`),
  CONSTRAINT `nilais_mahasiswa_id_foreign` FOREIGN KEY (`mahasiswa_id`) REFERENCES `mahasiswa` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci
1 row in set (0.002 sec)

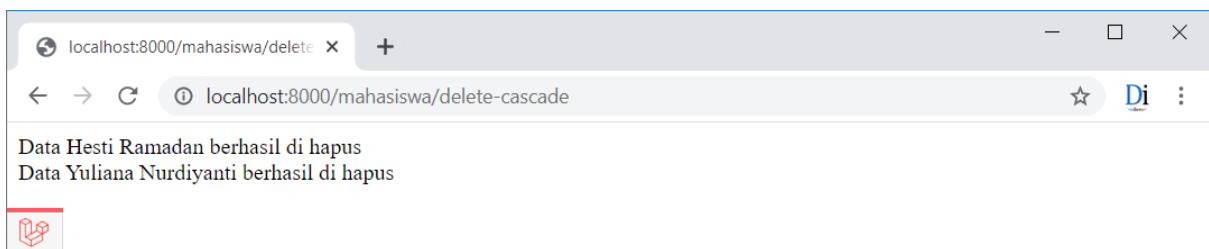
MariaDB [laravel]>
```

Gambar: Mode ON DELETE CASCADE sudah aktif

Untuk membuktikan, saya akan coba hapus beberapa data mahasiswa:

app\Http\Controllers\MahasiswaController.php

```
1 public function deleteCascade()
2 {
3
4     $mahasiswa = Mahasiswa::where('nama', 'Hesti Ramadan')->firstOrFail();
5     $mahasiswa->delete();
6     echo "Data $mahasiswa->nama berhasil dihapus <br>";
7
8     $mahasiswa = Mahasiswa::where('nama', 'Yuliana Nurdiyanti')->firstOrFail();
9     $mahasiswa->delete();
10    echo "Data $mahasiswa->nama berhasil dihapus";
11 }
```



Gambar: Menghapus dua mahasiswa

Dalam kode program ini saya menghapus 2 mahasiswa, Hesti Ramadan dan Yuliana Nurdiyanti. Khusus untuk Hesti Ramadan, mahasiswa tersebut memiliki data di tabel `nilais`. Tapi karena efek `ON DELETE CASCADE`, maka MySQL secara otomatis menghapus data nilai tersebut dari tabel `nilais`.

Perlu menjadi catatan bahwa fitur `ON DELETE CASCADE` ini adalah milik MySQL, bukan bawaan Laravel. Kita hanya mengaktifkannya di file migration saja.

Penjelasan lebih lanjut tentang `ON DELETE CASCADE` dan juga fitur lain seperti `ON DELETE SET NULL` atau `ON DELETE RESTRICT` menjadi jatah buku **MySQL Uncover** DuniaIlkom.

12.6. Eloquent Relationship `belongsTo()`

Dalam materi sebelum ini, kita fokus ke hubungan dari tabel `mahasiswa` ke tabel `nilais`, yakni relationship mahasiswa *has* nilai.

Sekarang saatnya kita masuk ke hubungan dari tabel `nilais` ke tabel `mahasiswa`. Dalam *relationship one to one*, hubungan yang terjadi adalah nilai **dimiliki oleh** mahasiswa, atau dalam bahasa inggris, nilai **belongs to** mahasiswa.

Untuk membuat hubungan seperti ini, silahkan buka file model `Nilai`, lalu tambah satu method bernama `mahasiswa()`:

App\Models\Nilai.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 Nilai extends Model
9 {
10     use HasFactory;
11     protected $guarded = [];
12
13     public function mahasiswa()
14     {
15         return $this->belongsTo('App\Models\Mahasiswa');
16     }
17 }
```

Tambahan kode program ada di baris 13 – 16, yakni sebuah method bernama `mahasiswa()` yang berisi perintah `return $this->belongsTo('App\Models\Mahasiswa')`. Inilah cara kita memberitahu Laravel apa jenis hubungan dari tabel `nilais` dengan tabel `mahasiswa`, yakni nilai **belongs to** mahasiswa.

Nama method `mahasiswa()` menggunakan kata *singular* tanpa tambahan 's' karena hubungan yang terjadi adalah *one to one*. Sebagai argument dari method `$this->belongsTo()`, berisi namespace file model yang terhubung, yakni '`App\Models\Mahasiswa`'.

Dengan tambahan method ini, kita sudah bisa menjalankan perintah eloquent *relationship one*

Eloquent Relationship: One to One

to one dari tabel `nilais` ke tabel `mahasiswa`.

Berikut daftar route yang akan di bahas:

`routes\web.php`

```
1 <?php
2 ...
3
4 use App\Http\Controllers\NilaiController;
5
6 Route::prefix('/nilai')->group(function () {
7     Route::get('/find', [NilaiController::class, 'find']);
8     Route::get('/where', [NilaiController::class, 'where']);
9     Route::get('/where-chaining', [NilaiController::class, 'whereChaining']);
10    Route::get('/has', [NilaiController::class, 'has']);
11    Route::get('/has-eager', [NilaiController::class, 'hasEager']);
12
13   Route::get('/test-input-1', [NilaiController::class, 'testInput1']);
14   Route::get('/test-input-2', [NilaiController::class, 'testInput2']);
15   Route::get('/test-input-3', [NilaiController::class, 'testInput3']);
16   Route::get('/test-input-4', [NilaiController::class, 'testInput4']);
17
18   Route::get('/associate-new', [NilaiController::class, 'associateNew']);
19   Route::get('/associate-find', [NilaiController::class, 'associateFind']);
20
21   Route::get('/delete', [NilaiController::class, 'delete']);
22   Route::get('/delete-mahasiswa', [NilaiController::class, 'deleteMahasiswa']);
23});
```

Sama seperti route untuk mahasiswa, saya menggunakan *route prefixes* '/nilai' di baris 6.

Dengan ini, maka alamat URL akan memiliki tambahan '/nilai' di bagian awal seperti /nilai/find, /nilai/where, dst.

Lalu silahkan refresh kedua tabel dan generate ulang seeder dengan perintah `php artisan migrate:fresh --seed`.

Urutan pembahasan juga sama seperti pada praktik mahasiswa has one nilai, namun karena konsep dasarnya sudah banyak kita pelajari, saya hanya akan membahas dengan lebih singkat.

Menampilkan Satu Gabungan Data

Sekarang kita membahas konsep nilai *belongs to* mahasiswa, maka semua perintah akan berangkat dari object model Nilai, tidak lagi dari model Mahasiswa seperti sebelumnya.

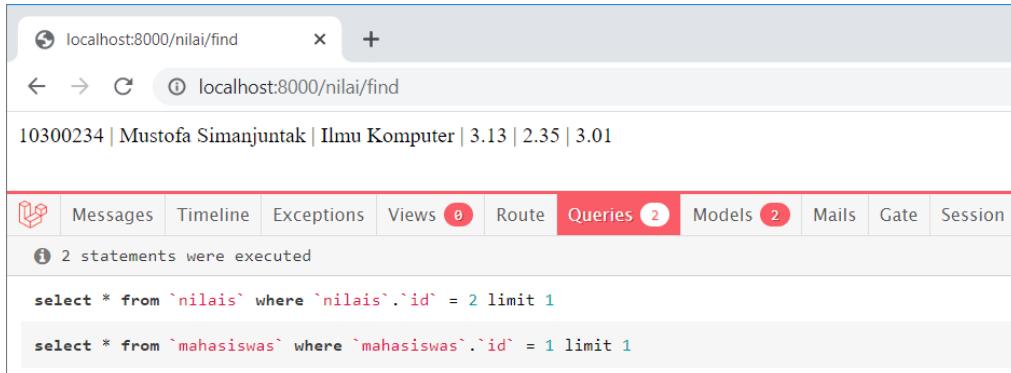
Berikut contoh cara mengakses nama mahasiswa dari object nilai:

`app\Http\Controllers\NilaiController.php`

```
1 <?php
2 ...
3 use App\Models\Nilai;
4 ...
```

Eloquent Relationship: One to One

```
5
6  public function find()
7  {
8      $nilai = Nilai::find(2);
9
10     echo "{$nilai->mahasiswa->nim} | ";
11     echo "{$nilai->mahasiswa->nama} | ";
12     echo "{$nilai->mahasiswa->jurusan} | ";
13     echo "$nilai->sem_1 | $nilai->sem_2 | $nilai->sem_3";
14 }
```



Gambar: Menampilkan data mahasiswa yang diakses dari model Nilai

Di baris 8 saya mencari nilai dengan id = 2, lalu menyimpannya ke dalam variabel \$nilai.

Untuk mengakses data mahasiswa dari variabel \$nilai, bisa menggunakan perintah \$nilai->mahasiswa->nim, \$nilai->mahasiswa->nama dan \$nilai->mahasiswa->jurusan. Tanda kurung kurawal " { " dan " } " diperlukan untuk menghindari efek *variable interpolation*.

Method `where()` juga bisa dipakai untuk mengubah kondisi pencarian:

app\Http\Controllers\NilaiController.php

```
1  public function where()
2  {
3      $nilai = Nilai::where('sem_3','>',2)->first();
4
5      echo "{$nilai->mahasiswa->nim} | ";
6      echo "{$nilai->mahasiswa->nama} | ";
7      echo "{$nilai->mahasiswa->jurusan} | ";
8      echo "$nilai->sem_1 | $nilai->sem_2 | $nilai->sem_3";
9  }
```

Hasil kode program:

10274992 | Galang Maryadi | Teknik Informatika | 2.18 | 2.44 | 3.14

Di baris 3 saya ingin mencari satu data dari tabel `nilais` yang nilai semester 3 di atas 2. Dari variabel \$nilai, bisa diakses nim, nama, serta jurusan milik mahasiswa tersebut.

Pencarian data ini juga bisa di-*chaining* menjadi 1 perintah panjang:

Eloquent Relationship: One to One

app\Http\Controllers\NilaiController.php

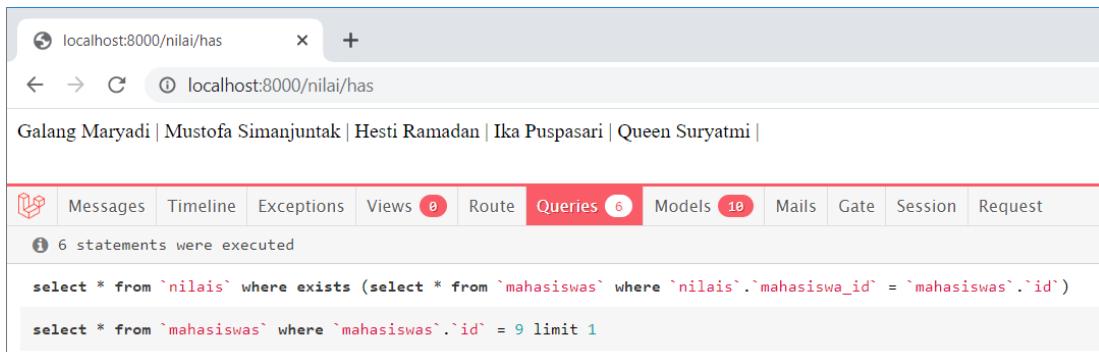
```
1 public function whereChaining()
2 {
3     echo Nilai::where('sem_3','>',2)->first()->mahasiswa->nama; // Galang Maryadi
4 }
```

Menampilkan Batasan has()

Dari object model Nilai, kita juga bisa menjalankan method `has()`. Karena dijalankan dari object nilai, maka ini akan menampilkan semua nilai yang dimiliki oleh mahasiswa:

app\Http\Controllers\NilaiController.php

```
1 public function has()
2 {
3     $nilais = Nilai::has('mahasiswa')->get();
4     foreach ($nilais as $nilai) {
5         echo $nilai->mahasiswa->nama." | ";
6     }
7 }
```



Gambar: Hasil dari perintah `Nilai::has('mahasiswa')->get()`

Logika yang terjadi memang agak aneh, karena dalam konsep *referential integrity*, semua nilai wajib di miliki oleh mahasiswa.

Perhatikan efek dari *lazy loading*, dimana terdapat 6 query yang dijalankan Laravel. Untuk versi *eager loading*, silahkan tambah method `with()` seperti contoh berikut:

app\Http\Controllers\NilaiController.php

```
1 public function hasEager()
2 {
3     // Tampilkan semua siswa yang memiliki nilai, versi eager loading
4     $nilais = Nilai::with('mahasiswa')->has('mahasiswa')->get();
5     foreach ($nilais as $nilai) {
6         echo $nilai->mahasiswa->nama." | ";
7     }
8 }
```

```
localhost:8000/nilai/has-eager
localhost:8000/nilai/has-eager
Galang Maryadi | Mustofa Simanjuntak | Hesti Ramadan | Ika Puspasari | Queen Suryatmi |
Messages Timeline Exceptions Views 0 Route Queries 2 Models 10 Mails Gate Session Request
2 statements were executed
select * from `nilais` where exists (select * from `mahasiswa` where `nilais`.`mahasiswa_id` = `mahasiswa`.`id`)
select * from `mahasiswa` where `mahasiswa`.`id` in (1, 3, 4, 8, 9)
```

Gambar: Hasil dari perintah Nilai::with('mahasiswa')->has('mahasiswa')->get()

Sekarang untuk menampilkan semua nama mahasiswa hanya butuh 2 buah query.

Untuk object model Nilai ini, kita juga bisa menjalankan method `whereHas()`, `doesntHave()` serta `whereDoesntHave()`. Namun untuk mempersingkat materi tidak akan saya bahas lagi. Terlebih logika ini akan lebih pas dijalankan dari model Mahasiswa, bukan dari model Nilai.

Menginput Data Relationship

Batasan *referential integrity* pada *one to one relationship* membuat kita tidak bisa menambah isi tabel `nilais` tanpa menghubungkannya dengan tabel mahasiswa. Termasuk juga jika mengisi kolom `mahasiswa_id` dengan mahasiswa yang belum ada, atau mengisinya dengan mahasiswa yang sudah memiliki nilai.

Ketiga percobaan berikut akan menghasilkan error:

app\Http\Controllers\NilaiController.php

```
1 public function testInput1()
2 {
3     // Test input nilai tanpa mengisi kolom mahasiswa_id:
4     $nilai = new Nilai;
5     $nilai->sem_1 = 3.12;
6     $nilai->sem_2 = 3.23;
7     $nilai->sem_3 = 3.34;
8     $nilai->save();
9
10    echo "Penambahan nilai ke database berhasil";
11    // SQLSTATE[HY000]: General error: 1364
12    // Field 'mahasiswa_id' doesn't have a default value
13 }
14
15 public function testInput2()
16 {
17     // Test input nilai dengan mengisi mahasiswa yang belum ada
18     $nilai = new Nilai;
19     $nilai->sem_1 = 3.12;
20     $nilai->sem_2 = 3.23;
21     $nilai->sem_3 = 3.34;
22     $nilai->mahasiswa_id = 20;
```

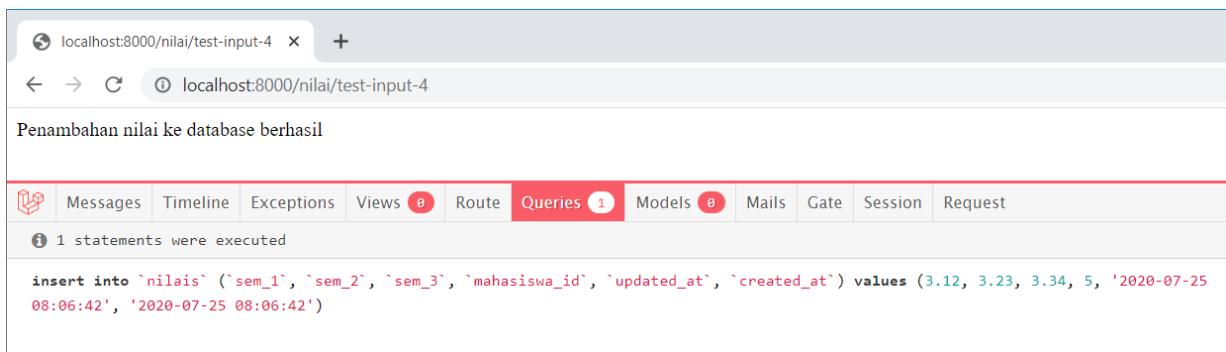
Eloquent Relationship: One to One

```
23     $nilai->save();
24
25     echo "Penambahan nilai ke database berhasil";
26     // SQLSTATE[23000]: Integrity constraint violation: 1452
27     // Cannot add or update a child row: a foreign key constraint fails
28 }
29
30 public function testInput3()
31 {
32     // Test input nilai dengan mengisi mahasiswa yang sudah memiliki nilai
33     $nilai = new Nilai;
34     $nilai->sem_1 = 3.12;
35     $nilai->sem_2 = 3.23;
36     $nilai->sem_3 = 3.34;
37     $nilai->mahasiswa_id = 3;
38     $nilai->save();
39
40     echo "Penambahan nilai ke database berhasil";
41     // SQLSTATE[23000]: Integrity constraint violation: 1062
42     // Duplicate entry '3' for key 'nilais_mahasiswa_id_unique'
43 }
```

Kita hanya bisa mengisi nilai baru untuk mahasiswa yang belum memiliki nilai sebelumnya:

app\Http\Controllers\NilaiController.php

```
1 public function testInput4()
2 {
3     // Ini bisa karena mahasiswa dengan id 5 sudah ada dan belum memiliki nilai
4     $nilai = new Nilai;
5     $nilai->sem_1 = 3.12;
6     $nilai->sem_2 = 3.23;
7     $nilai->sem_3 = 3.34;
8     $nilai->mahasiswa_id = 5;
9     $nilai->save();
10
11    echo "Penambahan nilai ke database berhasil";
12 }
```



Gambar: Penambahan data nilai berhasil

Cara lain adalah membuat mahasiswa baru, lalu memberikan nilai kepada mahasiswa tersebut. Ini sudah pernah kita jalankan dari model Mahasiswa. Namun juga bisa dilakukan dari model

Eloquent Relationship: One to One

Nilai menggunakan method `associate()`. Berikut contohnya:

app\Http\Controllers\NilaiController.php

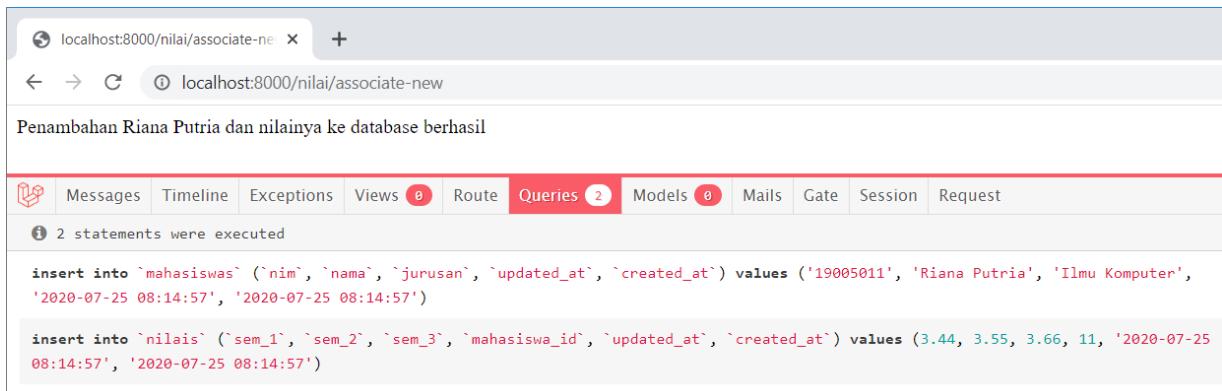
```
1 public function associateNew()
2 {
3     $mahasiswa = new \App\Models\Mahasiswa;
4     $mahasiswa->nim = '19005011';
5     $mahasiswa->nama = 'Riana Putria';
6     $mahasiswa->jurusan = 'Ilmu Komputer';
7     $mahasiswa->save();
8
9     $nilai = new Nilai;
10    $nilai->sem_1 = 3.44;
11    $nilai->sem_2 = 3.55;
12    $nilai->sem_3 = 3.66;
13
14    $nilai->mahasiswa()->associate($mahasiswa);
15    $nilai->save();
16
17    echo "Penambahan $mahasiswa->nama dan nilainya ke database berhasil";
18 }
```

Di baris 3 – 7 saya membuat object mahasiswa baru, mengisi property `nim`, `nama` dan `jurusan` kemudian langsung menyimpannya ke dalam database.

Di baris 9 – 12 saya juga membuat object nilai dan mengisi property `sem_1`, `sem_2`, dan `sem_3`. Seharusnya, property `mahasiswa_id` diisi dengan `id` dari object `$mahasiswa`. Akan tetapi dari mana nilai id ini diketahui?

Eloquent menyediakan method `associate()` untuk proses ini. Perintah `$nilai->mahasiswa()->associate($mahasiswa)` akan 'menghubungkan' object `$nilai` dengan `$mahasiswa`. Method `associate()` butuh sebuah argument yang diisi dengan object yang akan dihubungkan.

Setelah itu, simpan object nilai dengan perintah `$nilai->save()`.



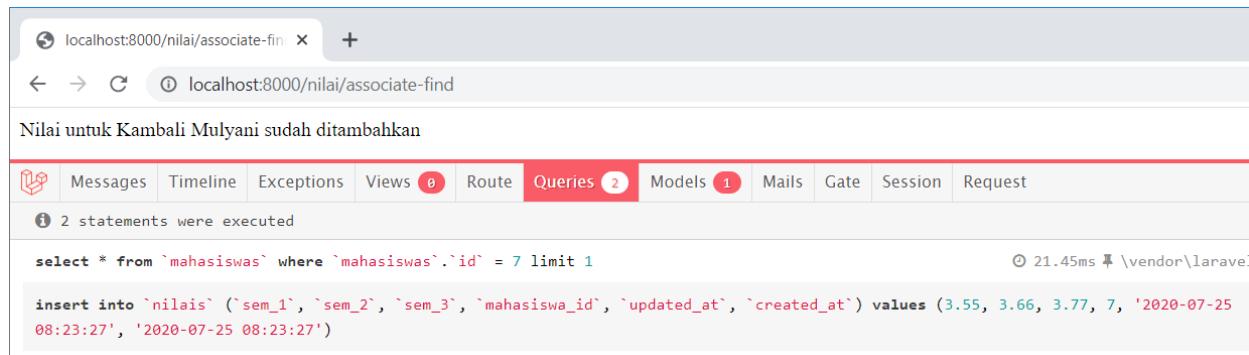
Gambar: Proses menyimpan data nilai dengan bantuan `associate()`

Method `associate()` ini tidak hanya untuk object mahasiswa yang baru, tapi bisa juga untuk object mahasiswa yang sudah ada di dalam database:

Eloquent Relationship: One to One

app\Http\Controllers\NilaiController.php

```
1 public function associateFind()
2 {
3     $mahasiswa = \App\Models\Mahasiswa::find(7);
4
5     $nilai = new Nilai;
6     $nilai->sem_1 = 3.55;
7     $nilai->sem_2 = 3.66;
8     $nilai->sem_3 = 3.77;
9
10    $nilai->mahasiswa()->associate($mahasiswa);
11    $nilai->save();
12
13    echo "Nilai untuk $mahasiswa->nama sudah ditambahkan ";
14 }
```



Gambar: Proses menyimpan data nilai dengan bantuan associate()

Kali ini saya mencari mahasiswa dengan id 7, lalu mengisi nilai untuk mahasiswa tersebut.

Sebenarnya Laravel juga menyediakan method `dissociate()` untuk 'memutus' hubungan nilai dengan mahasiswa. Tapi ini tidak bisa kita pakai karena kolom `mahasiswa_id` di tabel `nilais` tidak boleh kosong (tidak di set sebagai `nullable`).

Sebagai pengingat, proses input data bisa dilakukan dari 2 object model, apakah dari model Mahasiswa, atau model Nilai.

Jika dilakukan dari object mahasiswa, maka kita bisa menjalankan perintah berikut:

```
$mahasiswa->nilai()->save($nilai)
```

Sedangkan jika dilakukan dari object Nilai, maka bisa menggunakan perintah berikut:

```
$nilai->mahasiswa()->associate($mahasiswa);
$nilai->save();
```

Keduanya sama-sama bisa dipakai, tergantung situasi dan kebutuhan.

Menghapus Data Relationship

Tabel `nilais` bisa dihapus tanpa ada batasan *referential integrity*. Misalnya untuk menghapus 1 data nilai dengan id 3, bisa menggunakan kode berikut:

app\Http\Controllers\NilaiController.php

```
1 public function delete()
2 {
3     $nilai = Nilai::find(3);
4     $nilai->delete();
5     echo "Nilai berhasil di hapus";
6 }
```



Gambar: Menghapus 1 data nilai

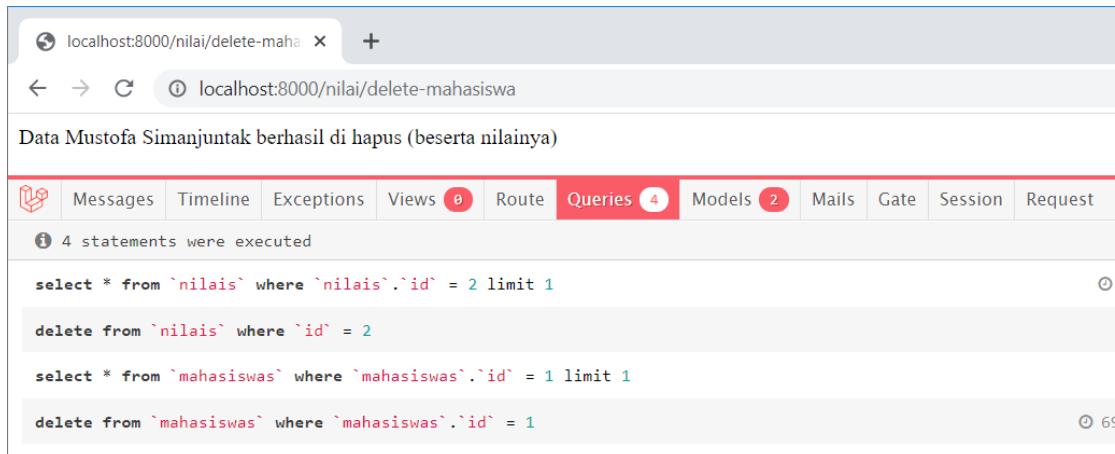
Dalam kode program ini saya menghapus nilai dengan id = 3. Ini merupakan kode eloquent biasa tanpa menggunakan relationship.

Jika kita ingin ikut menghapus mahasiswa yang terhubung ke nilai tersebut, bisa disambung seperti kode berikut:

app\Http\Controllers\NilaiController.php

```
1 public function deleteMahasiswa()
2 {
3     $nilai = Nilai::find(2);
4     $nilai->delete();
5
6     $nama_mahasiswa = $nilai->mahasiswa->nama;
7     $nilai->mahasiswa()->delete();
8
9     echo "Data $nama_mahasiswa berhasil di hapus (beserta nilainya)";
10 }
```

Eloquent Relationship: One to One



The screenshot shows a browser window with the URL `localhost:8000/nilai/delete-mahasiswa`. Below the URL bar, the page title is "Data Mustofa Simanjuntak berhasil dihapus (beserta nilainya)". The page content displays a success message: "4 statements were executed". Below this, four SQL queries are listed:

```
select * from `nilais` where `nilais`.`id` = 2 limit 1
delete from `nilais` where `id` = 2
select * from `mahasiswa` where `mahasiswa`.`id` = 1 limit 1
delete from `mahasiswa` where `mahasiswa`.`id` = 1
```

Gambar: Menghapus 1 data nilai beserta mahasiswa yang terhubung

Setelah menghapus nilai yang memiliki id 2 dari baris 4, di baris 6 saya mengambil nama mahasiswa yang terhubung ke nilai tersebut. Ini dipakai untuk menampilkan pesan echo di baris 9. Karena setelah mahasiswa yang memiliki nilai di hapus di baris 8, property `$nilai->mahasiswa->nama` tidak bisa lagi di akses (berisi NULL).

Inilah cara untuk menghapus data mahasiswa yang terhubung dengan nilai saat ini, yakni dengan perintah `$nilai->mahasiswa()->delete()`.

Dalam bab ini kita telah membahas cukup panjang tentang *relationship one to one* dalam Laravel. Materi utama ada di pembuatan *relationship* "mahasiswa **hasOne** nilai", serta "nilai **belongsTo** mahasiswa".

Dengan hubungan ini, kita bisa mengakses nilai dari tabel lain seolah-olah sebagai property dari object sekarang, seperti `$mahasiswa->nilai->sem_1` atau `$nilai->mahasiswa->nama`. Penulisan seperti ini sangat praktis daripada harus menulis manual query JOIN.

Berikutnya, kita akan masuk ke **Eloquent Relationship One to Many**.

13. Eloquent Relationship: One to Many

Eloquent Relationship One to Many artinya satu data di tabel utama terhubung ke banyak data di tabel kedua. Hubungan seperti ini cukup banyak dipakai dan akan kita bahas dengan lebih detail.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

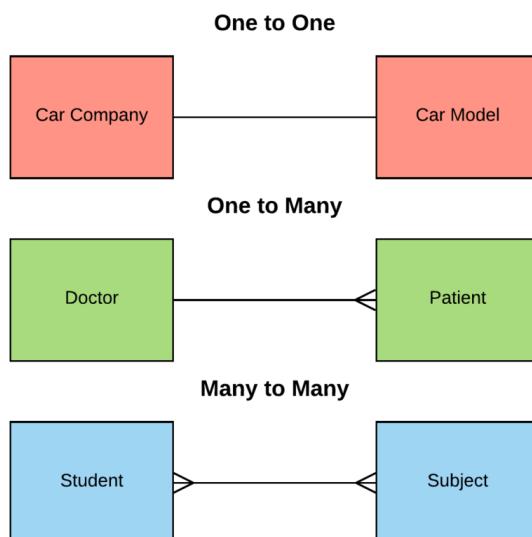
```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

13.1. Pengertian One to Many Relationship

Maksud dari **one to many relationship** adalah, satu baris data di tabel utama terhubung dengan satu atau lebih data di tabel kedua. Sebaliknya, satu baris data di tabel kedua, hanya terhubung ke satu baris data di tabel utama.

Dalam diagram **ERD** (Entity Relationship Diagram), *relationship one to many* digambarkan dengan satu garis yang bercabang di tabel kedua:



Gambar: Diagram ERD dari 3 jenis relationship (sumber gambar: <https://commons.wikimedia.org>)

Contoh dari konsep ini seperti satu jurusan yang memiliki banyak mahasiswa, atau satu perusahaan memiliki banyak karyawan. Akan tetapi satu mahasiswa hanya bisa memiliki satu jurusan, serta satu karyawan hanya bisa memiliki satu perusahaan.

13.2. Persiapan Awal

Sebagai bahan praktik dari penerapan *eloquent relationship one to many*, kita akan buat dua buah tabel, yakni tabel `jurusans` dan tabel `mahasiswas`. Kedua tabel ini berpasangan dengan model **Jurusan** dan model **Mahasiswa**.

Tabel `jurusans` berisi 6 kolom: `id`, `nama`, `kepala_jurusan`, `daya_tampung`, `created_at` dan `updated_at`. Tabel `mahasiswas` juga berisi 6 kolom: `id`, `nim`, `nama`, `jurusan_id`, `created_at` dan `updated_at`.

Berikut struktur tabel tabel `jurusans` dan `mahasiswas` yang akan di buat:

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM jurusans;
+----+-----+-----+-----+-----+-----+
| id | nama | kepala_jurusan | daya_tampung | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | Ilmu Komputer | Dr. Syahrial, M.Kom | 120 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 2 | Teknik Informatika | Prof. Mulyono | 250 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 3 | Sistem Informasi | Dr. Umar Agustinus, M.Sc. | 90 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 4 | Teknik Komputer | Prof. Gunarto, M.T | 60 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.976 sec)

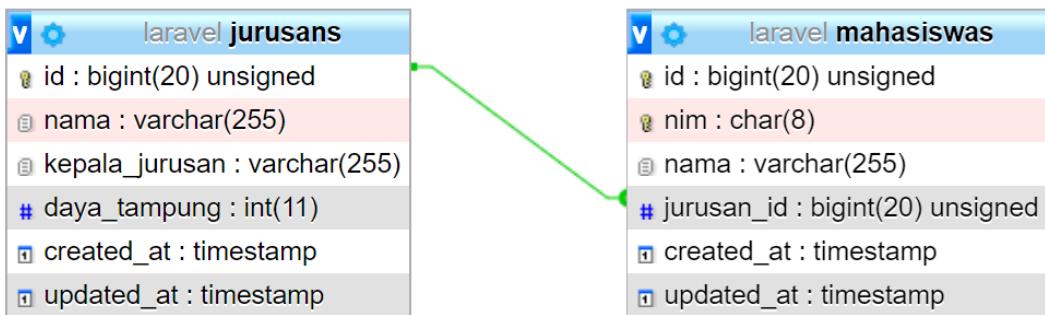
MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+
| id | nim | nama | jurusan_id | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | 10300234 | Mustofa Simanjuntak | 1 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 2 | 10451982 | Marsito Purnawati | 3 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 3 | 10980764 | Ika Puspasari | 1 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 4 | 10605319 | Queen Suryatmi | 3 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 5 | 10438572 | Yuliana Nurdyanti | 2 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 6 | 10737995 | Tiara Siregar | 1 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 7 | 10531551 | Kambali Mulyani | 1 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 8 | 10155818 | Hesti Ramadan | 2 | 2020-07-27 07:06:49 | 2020-07-27 07:06:49 |
| 9 | 10274992 | Galang Maryadi | 2 | 2020-07-27 07:06:49 | 2020-07-27 07:06:49 |
| 10 | 10228263 | Mahdi Rajata | 3 | 2020-07-27 07:06:49 | 2020-07-27 07:06:49 |
+----+-----+-----+-----+-----+-----+
10 rows in set (0.002 sec)
```

Gambar: Tabel `jurusans` dan tabel `mahasiswas`

Hubungan untuk kedua tabel adalah **one to many**, dimana 1 jurusan bisa memiliki banyak mahasiswa, tapi 1 mahasiswa hanya bisa terdaftar di 1 jurusan saja. Mirip seperti *relationship one to one*, kolom `jurusan_id` di tabel `mahasiswas` akan bertindak sebagai *foreign key* yang menghubungkan kedua tabel.

Jika digambarkan ke dalam diagram **ERD**, berikut hubungan antara tabel `jurusans` dan tabel `mahasiswas`:

Eloquent Relationship: One to Many



Gambar: Hubungan antara tabel jurusans dengan tabel mahasiswa

Terlihat garis antara kolom `id` di tabel `jurusans` dengan kolom `jurusan_id` di tabel `mahasiswa`. Inilah hubungan *primary key* dengan *foreign key* dalam konsep *relationship one to many*.

Generate Data Sample

Kita akan buat kedua tabel melalui migration serta beberapa file tambahan. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```
php artisan make:model Jurusan -cm  
php artisan make:model Mahasiswa -cm
```

Kode di atas akan membuat file **model**, **controller** serta **migration** untuk tabel `jurusans` dan `mahasiswa`.

```
Laravel Project  
C:\xampp\htdocs\laravel01>php artisan make:model Jurusan -cm  
Model created successfully.  
Created Migration: 2020_07_27_061930_create_jurusans_table  
Controller created successfully.  
  
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa -cm  
Model created successfully.  
Created Migration: 2020_07_27_062003_create_mahasiswa_table  
Controller created successfully.  
  
C:\xampp\htdocs\laravel01>
```

Gambar: Pembuatan model, controller serta migration

Selanjutnya buka file migration tabel `jurusans` lalu rancang struktur tabel berikut:

database\migrations\<timestamp>_create_jurusans_table.php

```
1 public function up()  
2 {  
3     Schema::create('jurusans', function (Blueprint $table) {  
4         $table->id();  
5         $table->string('nama');  
6         $table->string('kepala_jurusan');  
7         $table->integer('daya_tampung');  
8         $table->timestamps();  
9     });  
10 }
```

Eloquent Relationship: One to Many

Tambahan untuk file migration ada di baris 5-7, yakni pembuatan kolom `nama` dan `kepala_jurusan` bertipe string, serta kolom `daya_tampung` dengan tipe data integer.

Lanjut, buka file migration untuk tabel `mahasiswa` dan rancang struktur tabel sebagai berikut:

database\migrations\<timestamp>_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->foreignId('jurusan_id')->constrained()->onDelete('cascade');
8         $table->timestamps();
9     });
10 }
```

Kolom `nim` dan `nama` sudah sering kita pakai. Tambahan sekarang ada di kolom `jurusans_id` yang berfungsi sebagai *foreign key* ke tabel `jurusans`.

Selain itu saya juga langsung menambah method `onDelete('cascade')` yang berfungsi untuk mengaktifkan fitur `ON DELETE CASCADE` milik MySQL. Efeknya, jika sebuah jurusan dihapus, maka semua mahasiswa yang memilih jurusan tersebut juga akan ikut terhapus.

Penggunaan method `onDelete('cascade')` ini tidak wajib, tapi bergantung ke desain program yang dibuat. Dalam beberapa kasus, efeknya malah tidak diinginkan. Bisa saja kita mewajibkan admin aplikasi untuk mengosongkan tabel mahasiswa terlebih dahulu sebelum menghapus jurusan.

Jika di bandingkan bab sebelumnya, saya tidak lagi menambah method `unique()` ke penulisan *foreign key*. Ini karena nantinya kolom `jurusans_id` boleh berisi angka yang sama beberapa kali (*one to many*).

Buat kedua tabel dengan menjalankan proses migration: `php artisan migrate`.

Langkah berikutnya adalah membuat beberapa data sample. Silahkan buka file `DatabaseSeeder.php` lalu modifikasi sebagai berikut:

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7 use App\Models\Jurusan;
8 use App\Models\Mahasiswa;
9
10 class DatabaseSeeder extends Seeder
```

Eloquent Relationship: One to Many

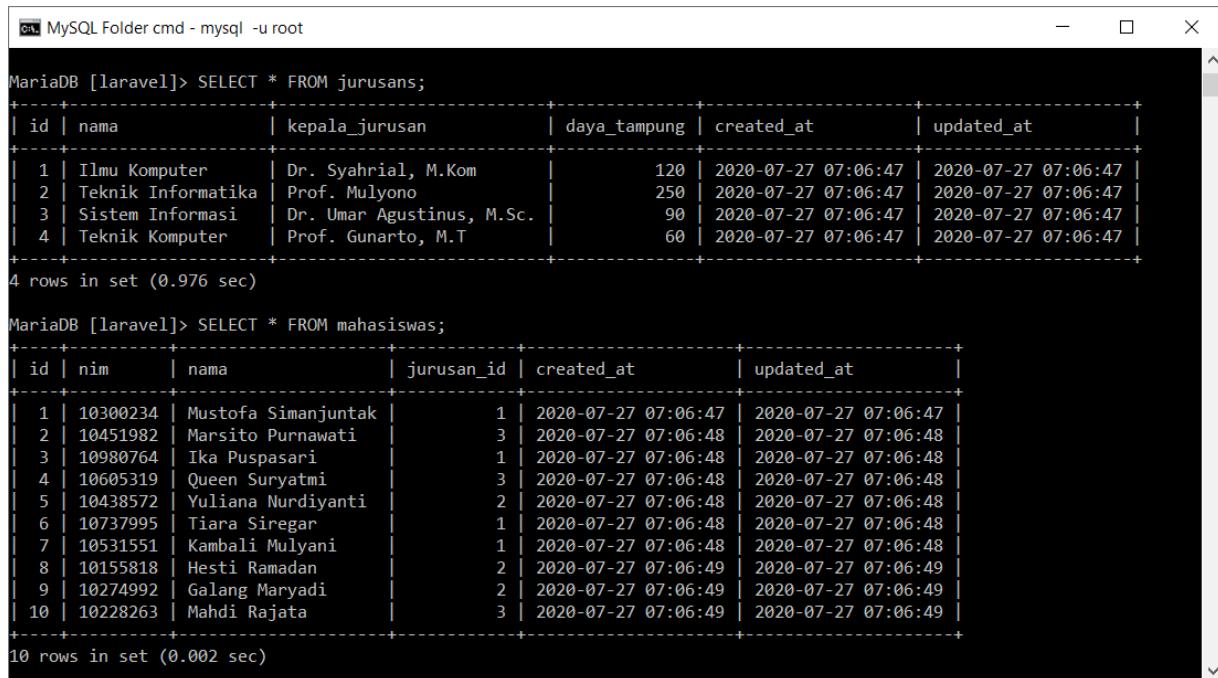
```
11  {
12      public function run()
13      {
14          Jurusan::create(
15              [
16                  'nama' => 'Ilmu Komputer',
17                  'kepala_jurusan' => 'Dr. Syahrial, M.Kom',
18                  'daya_tampung' => 120,
19              ]
20          );
21
22          Jurusan::create(
23              [
24                  'nama' => 'Teknik Informatika',
25                  'kepala_jurusan' => 'Prof. Mulyono',
26                  'daya_tampung' => 250,
27              ]
28          );
29
30          Jurusan::create(
31              [
32                  'nama' => 'Sistem Informasi',
33                  'kepala_jurusan' => 'Dr. Umar Agustinus, M.Sc.',
34                  'daya_tampung' => 90,
35              ]
36          );
37
38          Jurusan::create(
39              [
40                  'nama' => 'Teknik Komputer',
41                  'kepala_jurusan' => 'Prof. Gunarto, M.T',
42                  'daya_tampung' => 60,
43              ]
44          );
45
46          $faker = Faker::create('id_ID');
47          $faker->seed(123);
48
49          for ($i=0; $i<10; $i++) {
50              App\Models\Mahasiswa::create(
51                  [
52                      'nim' => $faker->unique()->numerify('10#####'),
53                      'nama' => $faker->firstName." ".$faker->lastName,
54                      'jurusan_id' => $faker->numberBetween(1,3),
55                  ]
56              );
57          }
58      }
59 }
```

Di baris 14 – 44 saya men-generate isi tabel jurusans secara manual. Terdapat 4 jurusan yang dibuat, yakni Ilmu Komputer, Teknik Informatika, Sistem Informasi dan Teknik Komputer. Laravel akan mengisi kolom id untuk keempat jurusan ini secara berurutan dari 1 – 4.

Eloquent Relationship: One to Many

Kemudian isi tabel `mahasiswa` di generate menggunakan Faker di baris 46 – 57. Khusus untuk kolom `jurusan_id` saya isi dengan nilai 1 – 3 saja. Ini agar kita memiliki 1 jurusan yang tidak memiliki mahasiswa, yakni jurusan Teknik Komputer dengan id 4.

Jalankan seeder dengan perintah `php artisan db:seed`. Berikut data lengkap dari kedua tabel:



```
MySQL [laravel] > SELECT * FROM jurusans;
+----+-----+-----+-----+-----+
| id | nama | kepala_jurusan | daya_tampung | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1 | Ilmu Komputer | Dr. Syahrial, M.Kom | 120 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 2 | Teknik Informatika | Prof. Mulyono | 250 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 3 | Sistem Informasi | Dr. Umar Agustinus, M.Sc. | 90 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 4 | Teknik Komputer | Prof. Gunarto, M.T | 60 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
+----+-----+-----+-----+-----+
4 rows in set (0.976 sec)

MySQL [laravel] > SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+
| id | nim | nama | jurusan_id | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1 | 10300234 | Mustofa Simanjuntak | 1 | 2020-07-27 07:06:47 | 2020-07-27 07:06:47 |
| 2 | 10451982 | Marsito Purnawati | 3 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 3 | 10980764 | Ika Puspasari | 1 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 4 | 10605319 | Queen Suryatmi | 3 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 5 | 10438572 | Yuliana Nurdyanti | 2 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 6 | 10737995 | Tiara Siregar | 1 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 7 | 10531551 | Kambali Mulyani | 1 | 2020-07-27 07:06:48 | 2020-07-27 07:06:48 |
| 8 | 10155818 | Hesti Ramadan | 2 | 2020-07-27 07:06:49 | 2020-07-27 07:06:49 |
| 9 | 10274992 | Galang Maryadi | 2 | 2020-07-27 07:06:49 | 2020-07-27 07:06:49 |
| 10 | 10228263 | Mahdi Rajata | 3 | 2020-07-27 07:06:49 | 2020-07-27 07:06:49 |
+----+-----+-----+-----+-----+
10 rows in set (0.002 sec)
```

Gambar: Isi tabel jurusans dan mahasiswa

Bisa terlihat bahwa jurusan setiap mahasiswa di tentukan dari nilai kolom `jurusan_id`.

Sebagai contoh, Mustofa Simanjuntak terdaftar di jurusan Ilmu Komputer (`jurusan_id = 1`), serta Marsito Purnawati terdaftar di jurusan Sistem Informasi (`jurusan_id = 3`). Hubungan seperti inilah yang bisa kita proses menggunakan *eloquent relationship*.

13.3. Membuat Join dengan DB Facade (Raw SQL Queries)

Sebagai pemanasan sebelum masuk ke *eloquent relationship*, saya kembali mengajak untuk menulis sedikit query SQL secara manual.

Kode program *raw query* ini akan dibuat pada file `JurusUserController.php`, dan berikut daftar route yang diperlukan:

```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\JurusUserController;
5
6 Route::get('/jurusan/all', [JurusUserController::class, 'all']);
```

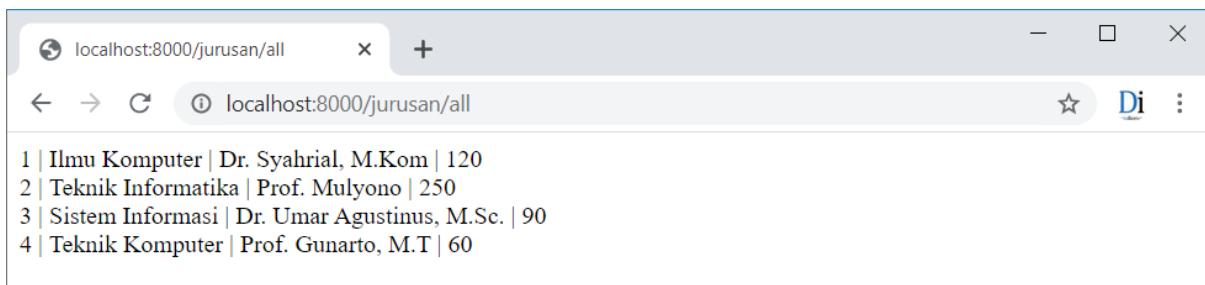
Eloquent Relationship: One to Many

```
7 Route::get('/jurusan/gabung', [JurusanController::class, 'gabung']);  
8 Route::get('/jurusan/gabung-join', [JurusanController::class, 'gabungJoin']);
```

Route pertama dipakai untuk testing mengakses semua data jurusan:

app\Http\Controllers\JurusanController.php

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4 use Illuminate\Support\Facades\DB;  
5  
6 class JurusanController extends Controller  
7 {  
8     public function all()  
9     {  
10         $jurusans = DB::select('SELECT * FROM jurusans');  
11         foreach ($jurusans as $jurusan) {  
12             echo "$jurusan->id | $jurusan->nama | $jurusan->kepala_jurusan | ";  
13             echo "$jurusan->daya_tampung <br>";  
14         }  
15     }  
16 }
```



Gambar: Menampilkan semua isi tabel jurusans

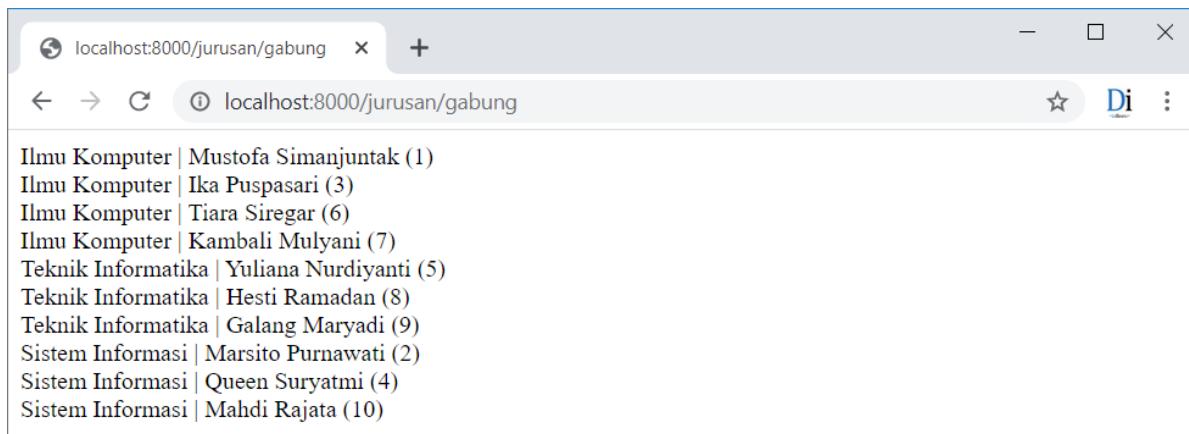
Di sini saya menjalankan query 'SELECT * FROM jurusans', lalu menampilkan isi setiap kolom.

Lanjut, berikut kode program untuk menampilkan gabungan tabel jurusans dan mahasiswa:

app\Http\Controllers\JurusanController.php

```
1 public function gabung()  
2 {  
3     $result = DB::select(  
4         'SELECT  
5             jurusans.nama as nama_jurusan,  
6             mahasiswa.id as id_mahasiswa,  
7             mahasiswa.nama as nama_mahasiswa  
8             FROM jurusans, mahasiswa  
9             WHERE jurusans.id = mahasiswa.jurusan_id'  
10        );  
11  
12    foreach ($result as $row) {  
13        echo "$row->nama_jurusan | $row->nama_mahasiswa ($row->id_mahasiswa) <br>";  
14    }
```

15 }



Gambar: Menampilkan semua isi tabel jurusans dan mahasiswa

Untuk menampilkan gabungan data tabel `jurusans` dan `mahasiswa`, saya menulis query yang cukup panjang di baris 4 – 9. Sebenarnya ini lebih karena pembuatan perintah `as` (alias) kolom `nama` dan `id`.

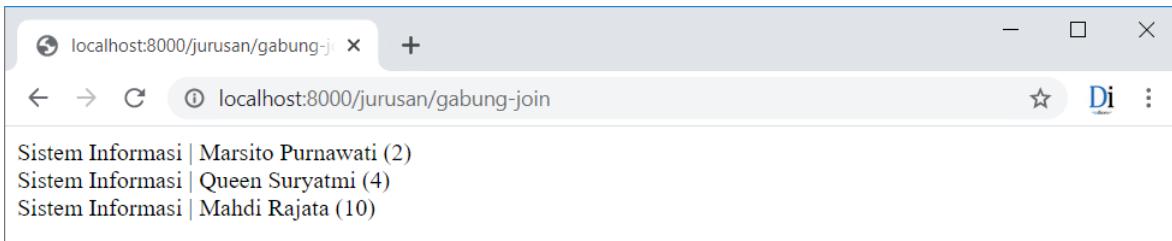
Ketika menulis query penggabungan tabel, MySQL akan 'bingung' jika terdapat nama kolom yang sama di kedua tabel. Jika tidak error, maka nilai kolom yang sama ini akan tertimpa oleh kolom lain.

Di dalam tabel `jurusans` dan `mahasiswa` sama-sama terdapat kolom `nama` dan `id`. Akibatnya, query '`SELECT * FROM jurusans, mahasiswa WHERE jurusans.id = mahasiswa.jurusan_id`' tidak bisa dipakai.

Sebagai alternatif penggabungan, bisa juga menggunakan query `JOIN`:

app\Http\Controllers\JurusanController.php

```
1 public function gabungJoin()
2 {
3     $result = DB::select(
4         'SELECT
5             jurusans.nama as nama_jurusan,
6             mahasiswa.id as id_mahasiswa,
7             mahasiswa.nama as nama_mahasiswa
8             FROM jurusans JOIN mahasiswa
9             ON jurusans.id = mahasiswa.jurusan_id
10            WHERE jurusans.nama = "Sistem Informasi"
11            ORDER BY mahasiswa.id'
12     );
13
14     foreach ($result as $row) {
15         echo "$row->nama_jurusan | $row->nama_mahasiswa ($row->id_mahasiswa) <br>";
16     }
17 }
```



Gambar: Menampilkan semua isi tabel jurusans dan mahasiswa dengan query JOIN

Selain query JOIN, saya juga membuat batasan WHERE untuk mahasiswa yang memilih jurusan Sistem Informasi saja. Selain itu data akan terurut berdasarkan kolom id tabel mahasiswa.

Inilah cara penulisan *raw query* untuk menampilkan data dari kedua tabel. Menggunakan eloquent relationship, perintah yang digunakan akan lebih sederhana dan lebih singkat.

13.4. Eloquent Relationship hasMany()

Sekarang saatnya masuk ke praktik penggabungan tabel dengan eloquent. Hal pertama yang harus dilakukan adalah mendefinisikan seperti apa hubungan atau *relationship* antara tabel jurusans dengan tabel mahasiswa.

Dalam konsep ini saya akan merancang agar satu baris data di tabel jurusans terhubung ke banyak data di tabel mahasiswa, atau bisa juga disebut satu jurusan *memiliki banyak* mahasiswa (jurusan **has many** mahasiswa).

Sebaliknya, satu baris data di tabel mahasiswa hanya bisa terhubung ke satu data di tabel jurusans. Kali ini hubungan yang terjadi adalah satu mahasiswa *dimiliki oleh* satu jurusan (mahasiswa **belongs to** jurusan).

Berdasarkan praktik dari bab eloquent relationship one to one, bisa ditebak method yang akan kita gunakan adalah `hasMany()` dan `belongsTo()`.

Method `hasMany()` akan berada di model Jurusan, dan nantinya method `belongsTo()` di model Mahasiswa. Kita akan bahas method `hasMany()` terlebih dahulu.

Untuk membuat hubungan "jurusan has many mahasiswa", silahkan buka file `Jurusan.php`, lalu modifikasi dengan menambah kode berikut:

app\Models\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 {
```

Eloquent Relationship: One to Many

```
10     use HasFactory;
11
12     public function mahasiswa()
13     {
14         return $this->hasMany('App\Models\Mahasiswa');
15     }
16 }
```

Tambahan kode ada di baris 12 – 15, yakni sebuah method bernama `mahasiswa()` yang berisi perintah `return $this->hasMany('App\Models\Mahasiswa');`. Inilah cara kita memberitahu Laravel jenis hubungan dari tabel `jurusans` ke tabel `mahasiswa`, yaitu jurusan *has many* mahasiswa.

Nama method `mahasiswa()` menggunakan kata plural dengan tambahan 's' karena hubungan yang terjadi adalah *one to many*. Sebagai argument dari method `$this->hasMany()`, berisi namespace file model yang terhubung, yakni '`App\Models\Mahasiswa`'.

Dengan tambahan method ini, kita sudah bisa menjalankan berbagai perintah eloquent relationship *one to many* dari tabel jurusan ke tabel `mahasiswa`.

Berikut daftar route yang akan di bahas:

`routes\web.php`

```
1 Route::prefix('/jurusan')->group(function () {
2     Route::get('/find', [JurusanController::class, 'find']);
3     Route::get('/where', [JurusanController::class, 'where']);
4     Route::get('/all-join', [JurusanController::class, 'allJoin']);
5
6     Route::get('/has', [JurusanController::class, 'has']);
7     Route::get('/where-has', [JurusanController::class, 'whereHas']);
8     Route::get('/doesnt-have', [JurusanController::class, 'doesntHave']);
9
10    Route::get('/with-count', [JurusanController::class, 'withCount']);
11    Route::get('/load-count', [JurusanController::class, 'loadCount']);
12
13    Route::get('/insert-save', [JurusanController::class, 'insertSave']);
14    Route::get('/insert-create', [JurusanController::class, 'insertCreate']);
15    Route::get('/insert-create-many', [JurusanController::class, 'insertCreateMany']);
16
17    Route::get('/update', [JurusanController::class, 'update']);
18    Route::get('/update-push', [JurusanController::class, 'updatePush']);
19
20    Route::get('/delete', [JurusanController::class, 'delete']);
21});
```

Urutan route sama seperti praktek relationship *one to one*, yakni mulai dari cara menampilkan data, proses insert, update dan delete.

Sebelum membuat kode program, saya juga akan menginstall Laravel Debugbar dengan perintah `composer require barryvdh/laravel-debugbar --dev`.

Menampilkan Satu Gabungan Data

Berikut kode program untuk menampilkan 1 baris data dari tabel jurusans:

app\Http\Controllers\JurusanController.php

```
1 <?php
2 ...
3 use App\Models\Jurusan;
4 use App\Models\Mahasiswa;
5
6     public function find()
7     {
8         $jurusan = Jurusan::find(1);
9
10        echo "$jurusan->id | $jurusan->nama | $jurusan->kepala_jurusan | ";
11        echo "$jurusan->daya_tampung <br>";
12    }
```

Hasil kode program:

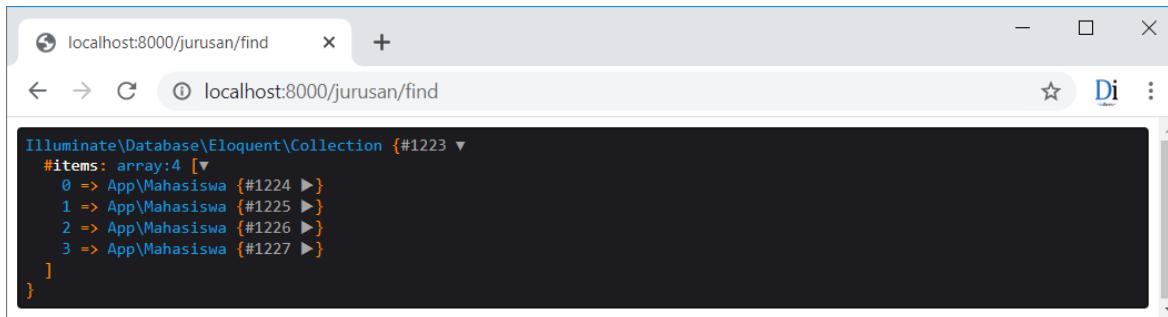
```
1 | Ilmu Komputer | Dr. Syahrial, M.Kom | 120
```

Dalam kode di atas saya mencari jurusan yang memiliki id 1 kemudian menampilkan data kolom `id`, `nama`, `kepala_jurusan` dan `daya_tampung`. Tidak ada hal yang baru karena ini hanya perintah eloquent biasa.

Lanjut, bagaimana cara menampilkan daftar mahasiswa yang terhubung ke jurusan tersebut? Kita bisa akses dari perintah `$jurusan->mahasiswas`:

app\Http\Controllers\JurusanController.php

```
1 public function find()
2 {
3     $jurusan = Jurusan::find(1);
4     dump($jurusan->mahasiswas);
5 }
```



Gambar: Hasil perintah `dump($jurusan->mahasiswas)`

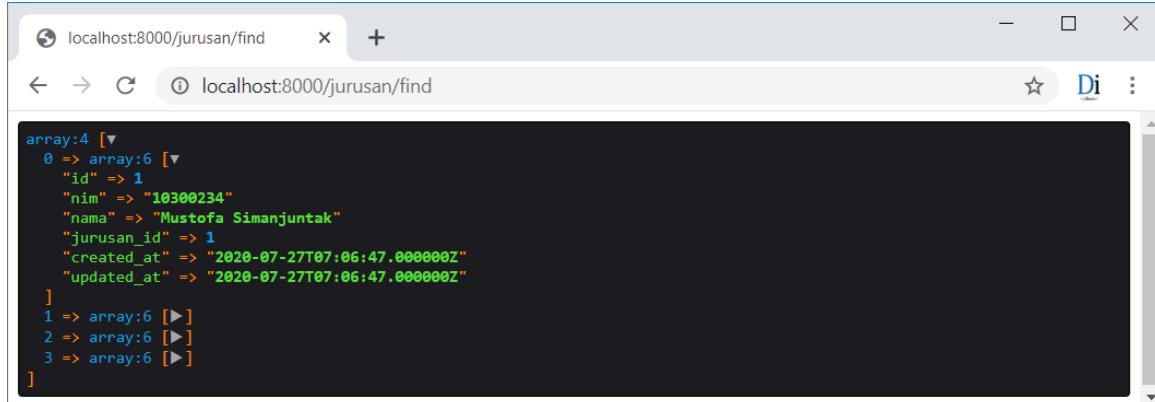
Karena ini adalah hubungan *one to many*, maka perintah `$jurusan->mahasiswas` akan mengembalikan *collection* dari beberapa object `Mahasiswa`. Secara logika tentu saja kita berharap bahwa satu jurusan akan memiliki banyak mahasiswa.

Eloquent Relationship: One to Many

Dalam contoh di atas terdapat 4 object Mahasiswa. Untuk melihat data dari setiap object, bisa buka baris `#attribute` atau tambah method `toArray()` ke perintah `$jurusan->mahasiswa`:

app\Http\Controllers\JurusanController.php

```
1 public function find()
2 {
3     $jurusan = Jurusan::find(1);
4     dump($jurusan->mahasiswa->toArray());
5 }
```



```
array:4 [▼
  0 => array:6 [▼
    "id" => 1
    "nim" => "10300234"
    "nama" => "Mustofa Simanjuntak"
    "jurusan_id" => 1
    "created_at" => "2020-07-27T07:06:47.000000Z"
    "updated_at" => "2020-07-27T07:06:47.000000Z"
  ]
  1 => array:6 [▶]
  2 => array:6 [▶]
  3 => array:6 [▶]
]
```

Gambar: Hasil perintah `dump($jurusan->mahasiswa->toArray())`

Sekarang data dari setiap mahasiswa langsung terlihat karena sudah di konversi menjadi array.

Sebagai pengingat, data mahasiswa ini bisa diakses dari perintah `$jurusan->mahasiswa` karena di dalam model Jurusan kita sudah membuat method `mahasiswa()`.

Method `mahasiswa()` berisi perintah `return $this->hasMany('App\Models\Mahasiswa');`. Hasilnya, setiap kali property `$jurusan->mahasiswa` diakses, itu akan mengembalikan collection dari semua mahasiswa yang ber-relasi.

Sekarang bagaimana cara mengakses setiap data mahasiswa? Karena hasil dari `$jurusan->mahasiswa` berbentuk collection, solusinya adalah perulangan `foreach`:

app\Http\Controllers\JurusanController.php

```
1 public function find()
2 {
3     $jurusan = Jurusan::find(1);
4     echo "Jurusan $jurusan->nama <br>";
5     echo "Nama Kepala Jurusan : $jurusan->kepala_jurusan <br>";
6     echo "Daya Tampung       : $jurusan->daya_tampung orang <br>";
7
8     echo "## Daftar Mahasiswa ##";
9     echo "<br><br>";
10    foreach ($jurusan->mahasiswa as $mahasiswa){
11        echo "$mahasiswa->nama ($mahasiswa->nim) <br>";
12    }
13 }
```

Eloquent Relationship: One to Many

The screenshot shows a browser window with the URL `localhost:8000/jurusan/find`. The page displays information about a Jurusan object with ID 1, specifically 'Ilmu Komputer'. It includes the head teacher's name ('Dr. Syahrial, M.Kom') and the number of students ('120 orang'). Below this, a section titled '## Daftar Mahasiswa ##' lists student names: Mustofa Simanjuntak (10300234), Ika Puspasari (10980764), Tiara Siregar (10737995), and Kambali Mulyani (10531551). At the bottom of the page, the Laravel Debugbar shows two database queries:

```
select * from `jurusans` where `jurusans`.`id` = 1 limit 1
select * from `mahasiswa` where `mahasiswa`.`jurusan_id` = 1 and `mahasiswa`.`jurusan_id` is not null
```

The Debugbar also indicates 19.34ms execution time.

Gambar: Menampilkan daftar mahasiswa untuk jurusan dengan id = 1

Di baris 3, perintah `Jurusan::find(1)` akan mengembalikan object model Jurusan yang memiliki id = 1. Object ini kemudian disimpan ke dalam variabel `$jurusan`. Antara baris 4 – 6 saya menampilkan 3 nilai milik tabel jurusan, yakni `nama`, `kepala_jurusan`, dan `daya_tampung`.

Kemudian di baris 10 terdapat perulangan `foreach ($jurusan->mahasiswa as $mahasiswa)`. Kode ini akan mengakses setiap object model Mahasiswa yang tersimpan di dalam `$jurusan->mahasiswa`. Dalam setiap perulangan, kolom `nama` dan `nim` diakses dari perintah `$mahasiswa->nama` (`$mahasiswa->nim`).

Penjelasan di atas memang sedikit rumit karena perlu pemahaman mendalam tentang object Model, collection, serta cara kerja perulangan foreach. Mudah-mudahan bisa dipahami karena ini menjadi kunci dari cara menampilkan data menggunakan eloquent relationship one to many.

Dari tampilan Laravel debugbar terlihat bahwa meskipun kita menggunakan perulangan foreach, tapi query yang dijalankan hanya 2 buah. Semua isi collection object Mahasiswa langsung diambil ketika menjalankan perintah `$jurusan->mahasiswa`, tidak satu per satu.

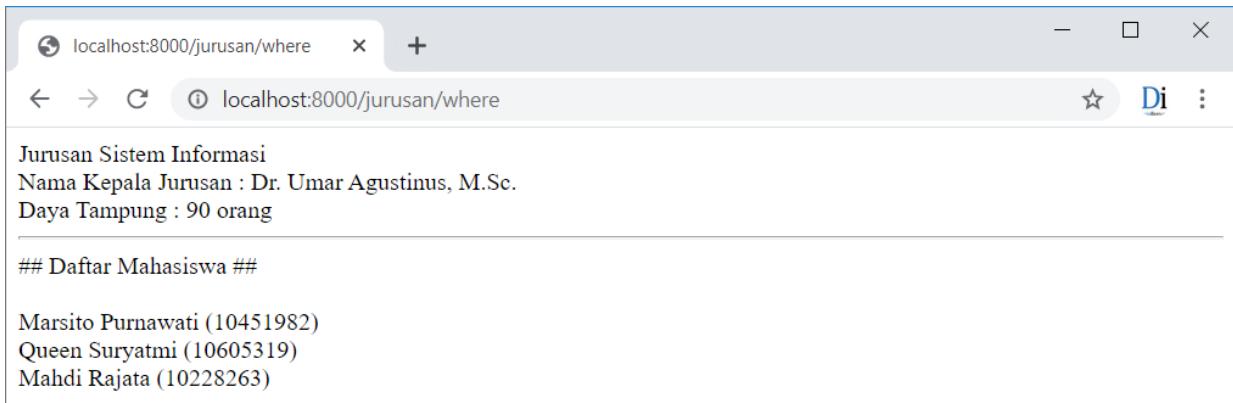
Sebagai contoh lain, saya ingin menampilkan daftar mahasiswa yang jurusannya di pimpin oleh Dr. Umar Agustinus, M.Sc.:

app\Http\Controllers\JurusanController.php

```
1 public function relationshipWhere()
2 {
3     $jurusan = Jurusan::where('kepala_jurusan', 'Dr. Umar Agustinus, M.Sc.')
4         ->first();
5
6     echo "Jurusan $jurusan->nama <br>";
7     echo "Nama Kepala Jurusan : $jurusan->kepala_jurusan <br>";
8     echo "Daya Tampung       : $jurusan->daya_tampung orang <br>";
9 }
```

Eloquent Relationship: One to Many

```
10     echo "## Daftar Mahasiswa ##";
11     echo "<br><br>";
12     foreach ($jurusan->mahasiswas as $mahasiswa){
13         echo "$mahasiswa->nama ($mahasiswa->nim) <br>";
14     }
15 }
```



Gambar: Menampilkan daftar mahasiswa untuk jurusan yang di kepala oleh Dr. Umar Agustinus, M.Sc.

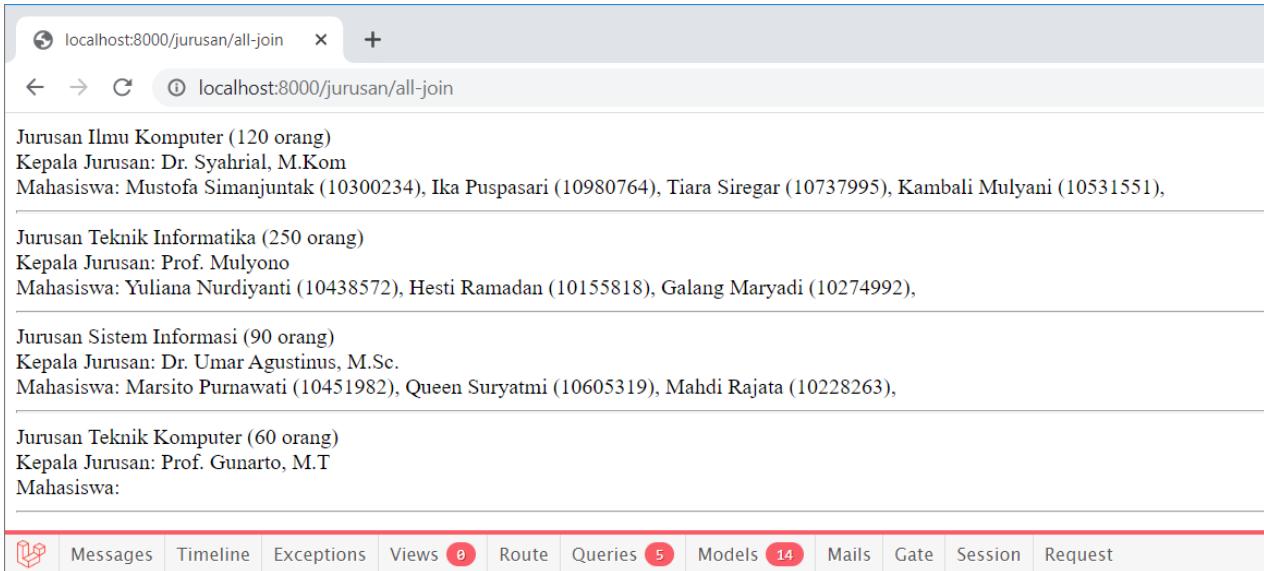
Perbedaan dari kode sebelumnya hanya di proses pencarian jurusan. Kali ini menggunakan method `where()->find()` untuk mencari jurusan yang di kepala Dr. Umar Agustinus, M.Sc.

Menampilkan Banyak Gabungan Data

Dalam praktek sebelumnya kita sudah menampilkan banyak data mahasiswa, tapi itu hanya untuk 1 jurusan. "Banyak gabungan data" yang dimaksud di sini adalah untuk tabel Jurusan.

Berdasarkan apa yang kita pelajari sejauh ini, bisakah anda rancang kode program yang akan menampilkan semua jurusan beserta nama mahasiswa dari setiap jurusan tersebut?

Kita berangkat dari perintah `$jurusans = Jurusan::all()`, dan hasil akhir yang saya inginkan adalah seperti tampilan di bawah ini:



Gambar: Menampilkan seluruh jurusan beserta seluruh mahasiswa

Ketika menjalankan perintah `$jurusans = Jurusan::all()`, variabel `$jurusan` akan berisi collection dari semua object model Jurusan.

Setiap kali mendapat data dalam bentuk collection, kuncinya adalah **perulangan foreach**. Kita perlu melakukan perulangan untuk menampilkan setiap data dari collection ini:

```
1 $jurusans = Jurusan::all();
2
3 foreach ($jurusans as $jurusan){
4     echo "Jurusan $jurusan->nama ($jurusan->daya_tampung_orang)<br> ";
5     echo "Kepala Jurusan: $jurusan->kepala_jurusan <br> ";
6     echo "<hr>";
7 }
```

Dengan kode program di atas, data `nama`, `daya_tampung` dan `nama_kepala_jurusan` sudah tampil di web browser. Sekarang giliran menampilkan daftar mahasiswa yang terdaftar untuk jurusan tersebut.

Dari praktek sebelumnya, daftar mahasiswa bisa diakses dari `$jurusan->mahasiswas`. Setiap object Jurusan akan memiliki property ini. Hasilnya berupa collection dari object Mahasiswa. Kembali, karena kita berhadapan dengan collection, kuncinya perulangan **foreach**:

app\Http\Controllers\JurusanController.php

```
1 public function allJoin()
2 {
3     $jurusans = Jurusan::all();
4
5     foreach ($jurusans as $jurusan){
6         echo "Jurusan $jurusan->nama ($jurusan->daya_tampung_orang)<br> ";
7         echo "Kepala Jurusan: $jurusan->kepala_jurusan <br> ";
8         echo "Mahasiswa: ";
9         foreach ($jurusan->mahasiswas as $mahasiswa){
10            echo "$mahasiswa->nama ($mahasiswa->nim), ";
11        }
12        echo "<hr>";
13    }
14 }
```

Di sini data semua mahasiswa saya tampilkan dengan perulangan `foreach` di baris 9 – 11.

Kode di atas cukup singkat sekaligus kompleks karena terdapat *nested foreach*, yakni perulangan `foreach` di dalam perulangan `foreach`. Saya sangat sarankan anda luangkan waktu sejenak untuk memahami alur kode program yang terjadi.

Jika melihat Laravel debugbar, kode di atas butuh 5 buah query (*lazy loading*). Jika ingin menggunakan versi *eager loading*, bisa ditambah dengan method `with()` seperti contoh berikut:

Eloquent Relationship: One to Many

app\Http\Controllers\JurusanController.php

```
1 public function allJoin()
2 {
3     $jurusans = Jurusan::with('mahasiswa')->get();
4
5     foreach ($jurusans as $jurusan){
6         echo "Jurusan $jurusan->nama ($jurusan->daya_tampung_orang)<br> ";
7         echo "Kepala Jurusan: $jurusan->kepala_jurusan <br> ";
8         echo "Mahasiswa: ";
9         foreach ($jurusan->mahasiswa as $mahasiswa){
10            echo "$mahasiswa->nama ($mahasiswa->nim), ";
11        }
12        echo "<br>";
13    }
14 }
```

Jurusan Ilmu Komputer (120 orang)
Kepala Jurusan: Dr. Syahrial, M.Kom
Mahasiswa: Mustofa Simanjuntak (10300234), Ika Puspasari (10980764), Tiara Siregar (10737995), Kambali Mulyani (10531551),

Jurusan Teknik Informatika (250 orang)
Kepala Jurusan: Prof. Mulyono
Mahasiswa: Yuliana Nurdyanti (10438572), Hesti Ramadan (10155818), Galang Maryadi (10274992),

Queries 2

```
select * from `jurusans`
select * from `mahasiswa` where `mahasiswa`.`jurusan_id` in (1, 2, 3, 4)
```

Gambar: Menampilkan seluruh jurusan beserta seluruh mahasiswa (versi eager loading)

Hasilnya akan sama seperti sebelumnya, tapi sekarang hanya butuh 2 buah query.

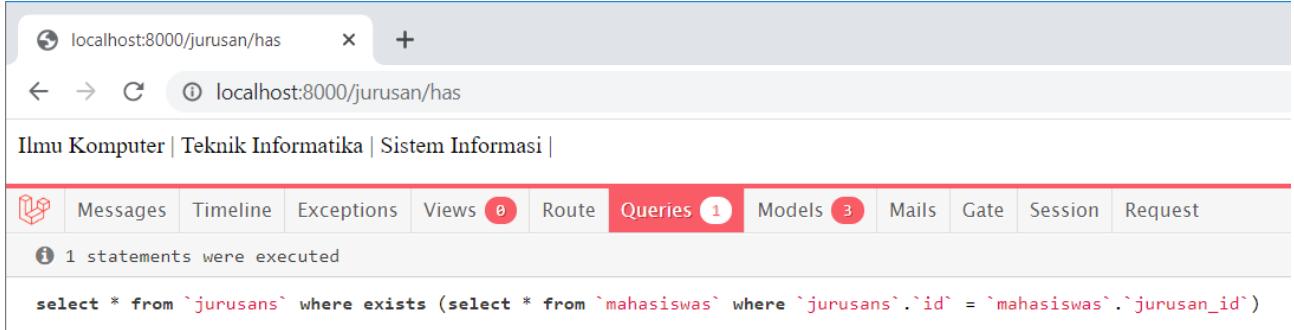
Menampilkan Batasan has() dan whereHas()

Method `has()` dan `whereHas()` juga bisa dipakai pada *relationship one to many*. Misalnya untuk mencari info apa saja jurusan yang minimal punya 1 mahasiswa:

app\Http\Controllers\JurusanController.php

```
1 public function has()
2 {
3     $jurusans = Jurusan::has('mahasiswa')->get();
4
5     foreach ($jurusans as $jurusan) {
6         echo "$jurusan->nama | ";
7     }
8 }
```

Eloquent Relationship: One to Many



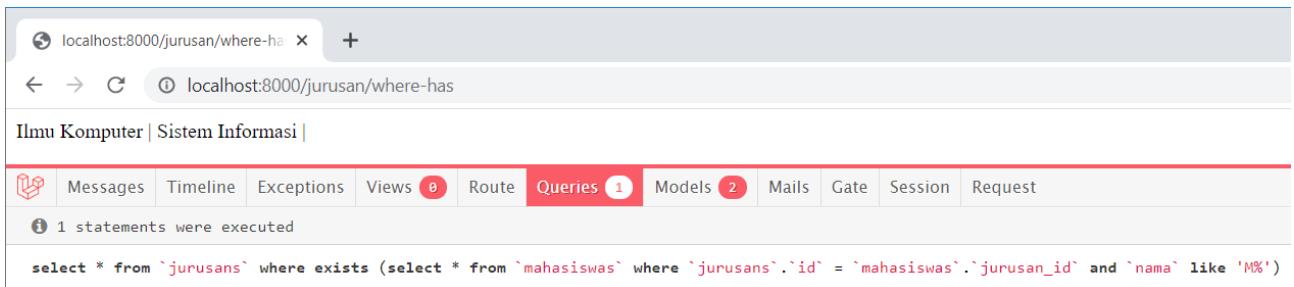
Gambar: Daftar jurusan yang memiliki mahasiswa

Dari hasil di atas terlihat ada 3 jurusan yang memiliki minimal 1 mahasiswa. Perintah `Jurusan::has ('mahasiswa')->get()` akan mengembalikan collection, sehingga harus di proses menggunakan perulangan foreach untuk menampilkan data setiap jurusan.

Jika ingin membuat batasan yang lebih kompleks, bisa menggunakan method `whereHas()` seperti contoh berikut:

app\Http\Controllers\JurusanController.php

```
1 public function whereHas()
2 {
3     $jurusans = Jurusan::whereHas('mahasiswa', function ($query) {
4         $query->where('nama', 'like', 'M%');
5     })->get();
6
7     foreach ($jurusans as $jurusan) {
8         echo "$jurusan->nama | ";
9     }
10 }
```



Gambar: Daftar jurusan yang memiliki mahasiswa dengan nama diawali huruf R

Maksud dari perintah di atas adalah saya ingin mencari daftar jurusan yang memiliki minimal 1 mahasiswa, dan di antara mahasiswa tersebut harus ada yang namanya diawali huruf 'R'.

Ternyata hanya ada 2 buah jurusan, yakni Ilmu Komputer dan Sistem Informasi. Artinya jurusan Teknik Informatika tidak memiliki mahasiswa yang namanya di awali dengan huruf 'R'.

Menampilkan Batasan doesntHave()

Method `doesntHave()` bisa dipakai untuk menampilkan data yang **tidak punya pasangan** di tabel kedua. Misalnya saya ingin menampilkan semua jurusan yang tidak memiliki mahasiswa:

app\Http\Controllers\JurusanController.php

```
1 public function doesntHave()
2 {
3     $jurusans = Jurusan::doesntHave('mahasiswa')->get();
4
5     foreach ($jurusans as $jurusan) {
6         echo "$jurusan->nama | ";
7     }
8 }
```



Gambar: Tampilkan semua jurusan yang tidak memiliki mahasiswa

Untuk kondisi yang lebih kompleks, tersedia juga method `whereDoesntHave()` dan `orWhereDoesntHave()`.

Menghitung Data dengan withCount() dan loadCount()

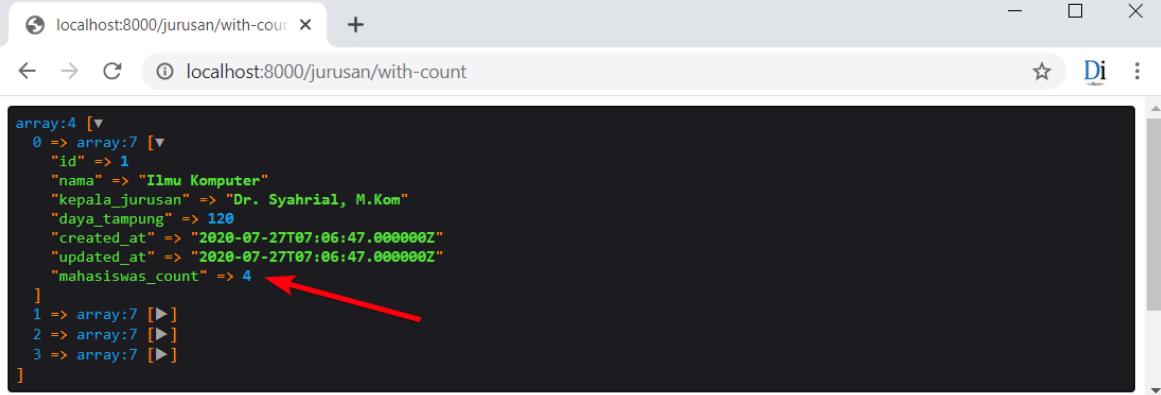
Ketika memproses data *one to many*, kadang kala kita butuh info jumlah baris yang terhubung ke tabel utama. Untuk keperluan ini tersedia method `withCount()` dan `loadCount()`.

Perbedaan dari kedua method ini ada di posisi pemanggilan. Jika di awal kita sudah ingin menampilkan jumlah mahasiswa, maka bisa pakai method `withCount()` seperti contoh berikut:

app\Http\Controllers\JurusanController.php

```
1 public function withcount()
2 {
3     $jurusans = Jurusan::withCount('mahasiswa')->get();
4     dump($jurusans->toArray());
5 }
```

Eloquent Relationship: One to Many



```
array:4 [▼
  0 => array:7 [▼
    "id" => 1
    "nama" => "Ilmu Komputer"
    "kepala_jurusan" => "Dr. Syahrial, M.Kom"
    "daya_tampung" => 120
    "created_at" => "2020-07-27T07:06:47.000000Z"
    "updated_at" => "2020-07-27T07:06:47.000000Z"
    "mahasiswa_count" => 4
  ] 1 => array:7 [▶]
  2 => array:7 [▶]
  3 => array:7 [▶]
]
```

Gambar: Hasil penggunaan method withCount()

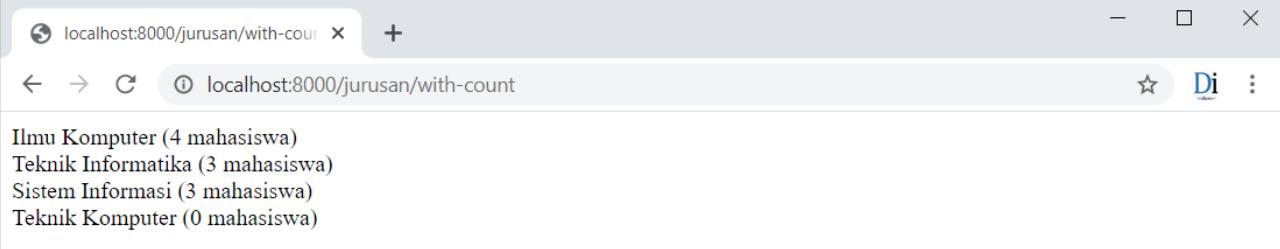
Method `withCount()` butuh satu argument berupa string dari nama method relasi. Dalam contoh ini, method relasi tersebut adalah `mahasiswa()` yang ada di model Jurusan.

Dengan pemanggilan `Jurusan::withCount('mahasiswa')->get()`, maka akan muncul satu property tambahan bernama `mahasiswa_count` di dalam setiap object Jurusan. Property ini berisi angka dari jumlah mahasiswa yang terhubung.

Untuk menampilkannya, bisa diakses dengan cara biasa:

app\Http\Controllers\JurusanController.php

```
1 public function withcount()
2 {
3     $jurusans = Jurusan::withCount('mahasiswa')->get();
4     foreach ($jurusans as $jurusan) {
5         echo "$jurusan->nama ($jurusan->mahasiswa_count mahasiswa) <br> ";
6     }
7 }
```



```
Ilmu Komputer (4 mahasiswa)
Teknik Informatika (3 mahasiswa)
Sistem Informasi (3 mahasiswa)
Teknik Komputer (0 mahasiswa)
```

Gambar: Menampilkan hasil withCount()

Perulangan `foreach` di perlukan karena method `Jurusan::withCount('mahasiswa')->get()` mengembalikan sebuah collection.

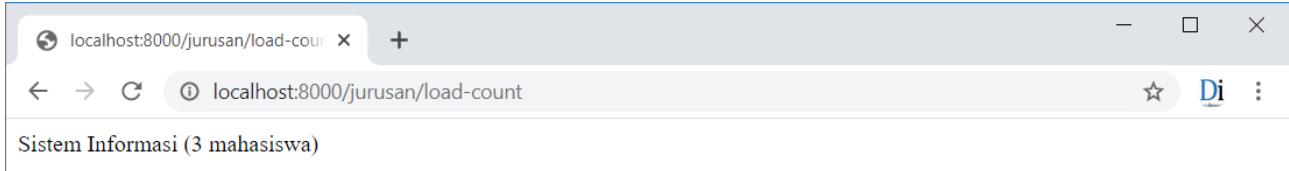
Untuk method `loadCount()`, dipakai jika object Jurusan sudah tersedia sebelumnya:

app\Http\Controllers\JurusanController.php

```
1 public function loadCount()
2 {
```

Eloquent Relationship: One to Many

```
3 $jurusan = Jurusan::where('kepala_jurusan', 'Dr. Umar Agustinus, M.Sc.')
4     ->first();
5 $jurusan->loadCount('mahasiswa');
6 echo "$jurusan->nama ($jurusan->mahasiswa_count mahasiswa) <br> ";
7 }
```



Gambar: Menampilkan hasil loadCount()

Di baris 3 saya mencari jurusan yang di kepala oleh 'Dr. Umar Agustinus, M.Sc.'. Hasilnya disimpan ke dalam variabel \$jurusan.

Sampai di sini, object Jurusan di variabel \$jurusan belum memiliki informasi terkait jumlah mahasiswa. Barulah ketika perintah \$jurusan->loadCount('mahasiswa') di baris 5 di jalankan, property mahasiswa_count bisa diakses dari object \$jurusan.

Hasil di atas tidak perlu perulangan foreach karena variabel \$jurusan tidak berisi collection.

Menginput Data Relationship

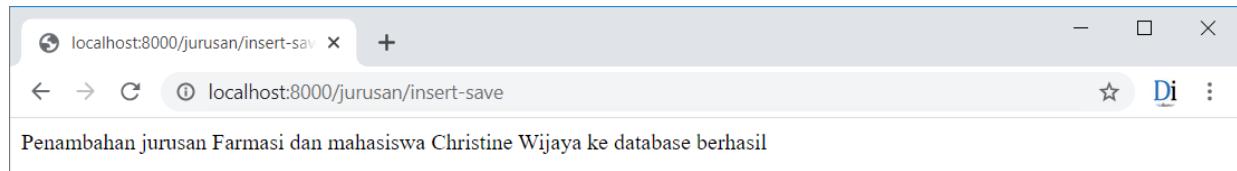
Proses input untuk tabel jurusans dan mahasiswa bisa dilakukan sekaligus maupun terpisah. Syaratnya, kita tidak bisa menginput data mahasiswa ke jurusan yang belum tersedia. Setiap kali ingin menginput data mahasiswa, kolom jurusan_id harus diisi dengan id yang ada di tabel jurusans.

Jika ingin menginput kedua tabel pada saat yang bersamaan, bisa menggunakan method save() yang mirip seperti di relationship one to one:

app\Http\Controllers\JurusanController.php

```
1 public function insertSave()
2 {
3     $jurusan = new Jurusan;
4     $jurusan->nama = 'Farmasi';
5     $jurusan->kepala_jurusan = 'Prof. Silvia Nst, M.Farm';
6     $jurusan->daya_tampung = 125;
7     $jurusan->save();
8
9     $mahasiswa = new Mahasiswa;
10    $mahasiswa->nim = '19001516';
11    $mahasiswa->nama = 'Christine Wijaya';
12
13    $jurusan->mahasiswa()->save($mahasiswa);
14
15    echo "Penambahan jurusan $jurusan->nama dan
16        mahasiswa $mahasiswa->nama ke database berhasil";
17 }
```

Eloquent Relationship: One to Many



Gambar: Proses insert dengan method save()

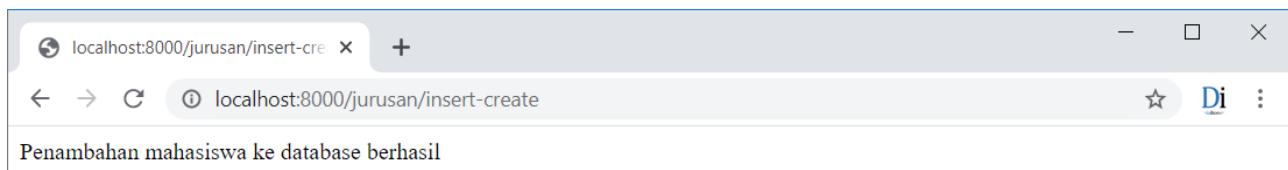
Di baris 3 – 7 saya membuat jurusan baru dan langsung menyimpannya ke tabel jurusans. Kemudian di baris 9 – 11 membuat object mahasiswa baru.

Agar mahasiswa ini langsung terhubung ke jurusan Farmasi yang baru saja dibuat, bisa menjalankan perintah `$jurusan->mahasiswa()->save($mahasiswa)` seperti di baris 13. Secara otomatis, kolom `jurusan_id` di tabel `mahasiswa` akan terisi `id` dari jurusan Farmasi.

Alternatif cara insert lain adalah menggunakan teknik *mass assignment* dengan method `create()`. Supaya bisa berjalan, tambah baris `protected $guarded = []` ke dalam model Jurusan dan Mahasiswa, setelah itu jalankan kode berikut:

app\Http\Controllers\JurusanController.php

```
1 public function insertCreate()
2 {
3     $jurusan = Jurusan::where('nama', 'Ilmu Komputer')->first();
4
5     $jurusan->mahasiswa()->create([
6         'nim' => '19001912',
7         'nama' => 'Bobby Permana',
8     ]);
9
10    echo "Penambahan mahasiswa ke database berhasil";
11 }
```



Gambar: Proses insert dengan method create()

Kode di atas akan menambah satu mahasiswa baru ke dalam tabel `mahasiswa`, dan mahasiswa tersebut sudah langsung terdaftar ke jurusan Ilmu Komputer.

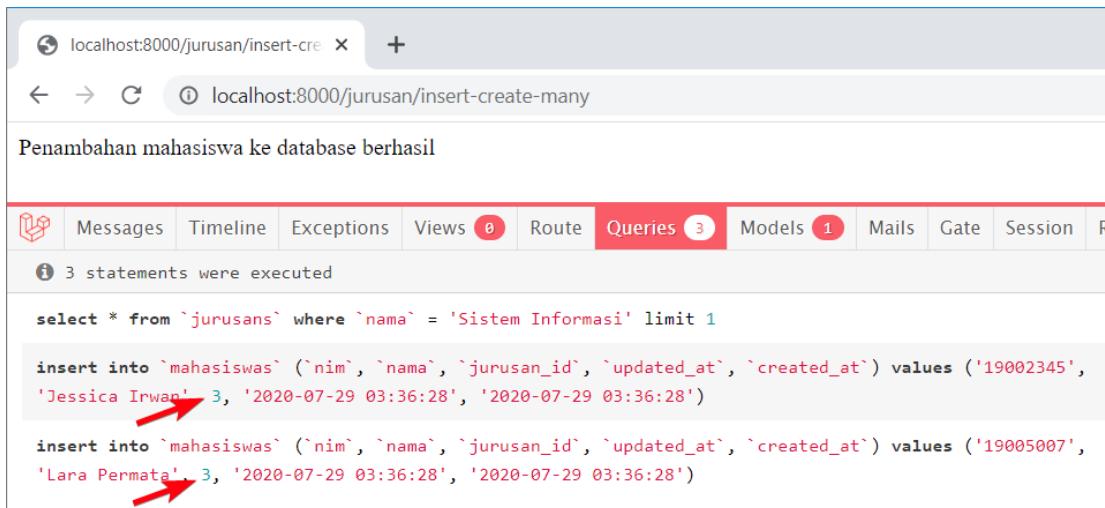
Dalam *relationship one to many*, kita akan sering menginput banyak data baru tabel kedua. Misalnya menginput 10 data mahasiswa sekaligus ke satu jurusan yang sama. Untuk keperluan ini Laravel menyediakan method `createMany()`. Berikut contoh penggunaannya:

app\Http\Controllers\JurusanController.php

```
1 public function insertCreateMany()
2 {
3     $jurusan = Jurusan::where('nama', 'Sistem Informasi')->first();
```

Eloquent Relationship: One to Many

```
4
5     $jurusan->mahasiswa->createMany([
6         [
7             'nim' => '19002345',
8             'nama' => 'Jessica Irwan'
9         ],
10        [
11            'nim' => '19005007',
12            'nama' => 'Lara Permata'
13        ]
14    ]);
15
16    echo "Penambahan mahasiswa ke database berhasil";
17 }
```



Gambar: Proses insert dengan method createMany()

Cara kerja dari method `createMany()` mirip seperti contoh `create()`. Hanya saja sekarang argument untuk method `createMany()` berbentuk nested array dimana setiap data mahasiswa disimpan ke dalam element array terpisah.

Mengupdate Data Relationship

Juga sama seperti *relationship one to one*, proses update untuk *one to many* juga bisa dilakukan lewat method `update()` dan `push()`. Berikut contoh untuk method `update()`:

app\Http\Controllers\JurusanController.php

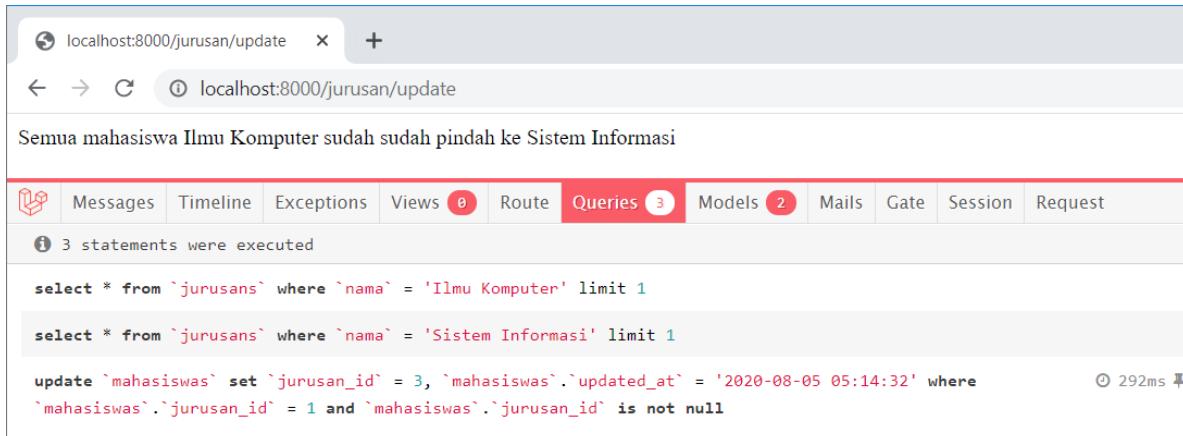
```
1  public function update()
2  {
3      $jurusan_ilkom = Jurusan::where('nama', 'Ilmu Komputer')->first();
4      $jurusan_si = Jurusan::where('nama', 'Sistem Informasi')->first();
5
6      $jurusan_ilkom->mahasiswa->update([
7          'jurusan_id' => $jurusan_si->id,
8      ]);
9 }
```

Eloquent Relationship: One to Many

```
10     echo "Semua mahasiswa $jurusan_ilkom->nama sudah sudah pindah  
11         ke $jurusan_si->nama";  
12 }
```

Di baris 3 dan 4 saya menyimpan object dari jurusan Ilmu Komputer ke variabel \$jurusan_ilkom serta jurusan Sistem Informasi ke variabel \$jurusan_si.

Kemudian di baris 6 mengakses semua mahasiswa dari \$jurusan_ilkom, lalu mengupdate kolom 'jurusan_id' agar berisi nilai \$jurusan_si->id. Artinya, saya ingin mengubah jurusan dari semua mahasiswa Ilmu Komputer ke jurusan Sistem Informasi.



Gambar: Proses update dengan method update()

Dari hasil tab Laravel debugbar bisa dipelajari lebih lanjut apa query yang sebenarnya dijalankan oleh Laravel.

Cara update kedua adalah menggunakan method push() seperti contoh berikut:

app\Http\Controllers\JurusanController.php

```
1 public function updatePush()  
2 {  
3     $jurusan = Jurusan::where('nama', 'Sistem Informasi')->first();  
4  
5     foreach ($jurusan->mahasiswa as $mahasiswa) {  
6         $mahasiswa->nama = $mahasiswa->nama . " S.Kom";  
7         $mahasiswa->push();  
8         echo "Berhasil update nama mahasiswa menjadi  
9             $mahasiswa->nama <br>";  
10    }  
11 }
```

Di baris 3 saya kembali mengambil satu object jurusan Sistem Informasi dan menyimpannya ke variabel \$jurusan.

Kemudian di baris 5 terdapat perulangan foreach yang mengakses setiap mahasiswa jurusan Sistem Informasi. Dalam setiap iterasi, saya mengupdate kolom nama dengan menambah string "S.Kom" di belakang nama yang ada saat ini. Artinya, semua mahasiswa Sistem Informasi

Eloquent Relationship: One to Many

sudah menjadi sarjana!

Agar perubahan ini bisa sampai ke database, jalankan method `$mahasiswa->push()` seperti di baris 7.



```
Berhasil update nama mahasiswa menjadi Mustofa Simanjuntak S.Kom  
Berhasil update nama mahasiswa menjadi Marsito Purnawati S.Kom  
Berhasil update nama mahasiswa menjadi Ika Puspasari S.Kom  
Berhasil update nama mahasiswa menjadi Queen Suryatmi S.Kom  
Berhasil update nama mahasiswa menjadi Tiara Siregar S.Kom  
Berhasil update nama mahasiswa menjadi Kambali Mulyani S.Kom  
Berhasil update nama mahasiswa menjadi Mahdi Rajata S.Kom  
Berhasil update nama mahasiswa menjadi Bobby Permana S.Kom  
Berhasil update nama mahasiswa menjadi Jessica Irwan S.Kom  
Berhasil update nama mahasiswa menjadi Lara Permata S.Kom
```

Gambar: Proses update dengan method push()

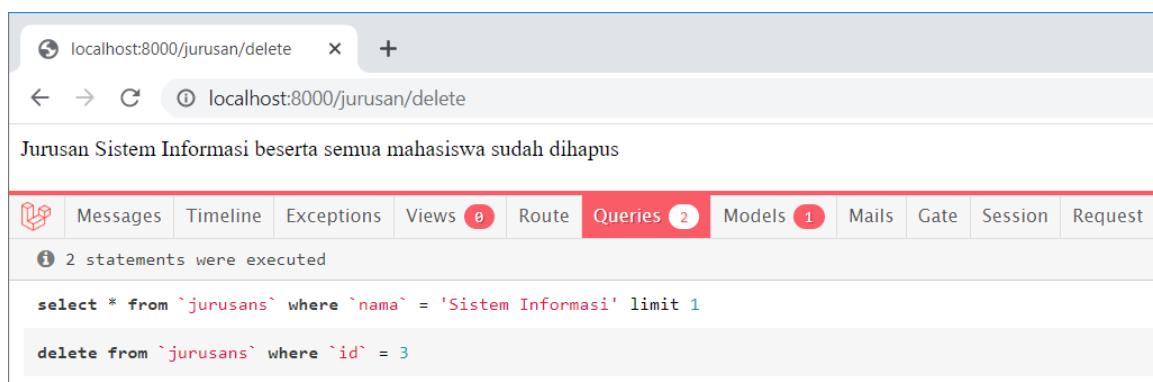
Menghapus Data Relationship

Sebagaimana biasa, proses penghapusan data bisa dilakukan dengan method `delete()`. Efek referential integrity juga harus kita terapkan di sini. Misalnya sebuah jurusan hanya bisa dihapus jika sudah tidak ada mahasiswa yang terdaftar di jurusan tersebut.

Pada saat pendefinisian tabel `mahasiswas`, saya sudah menambah `onDelete('cascade')` sehingga jika satu jurusan di hapus, maka semua mahasiswa dari jurusan tersebut juga akan ikut terhapus. Berikut contoh percobaan menghapus jurusan Sistem Informasi:

app\Http\Controllers\JurusanController.php

```
1 public function delete()  
2 {  
3     $jurusan = Jurusan::where('nama', 'Sistem Informasi')->firstOrFail();  
4     $jurusan->delete();  
5     echo "Jurusan $jurusan->nama beserta semua mahasiswa sudah dihapus";  
6 }
```



```
Jurusan Sistem Informasi beserta semua mahasiswa sudah dihapus
```

Queries 2

```
select * from `jurusans` where `nama` = 'Sistem Informasi' limit 1  
delete from `jurusans` where `id` = 3
```

Gambar: Proses penghapusan data dengan method delete()

Dari query di Laravel debugbar hanya terlihat 1 perintah delete jurusan dengan id = 3. Proses penghapusan data `mahasiswa` untuk jurusan tersebut di proses secara otomatis oleh MySQL.

13.5. Eloquent Relationship `belongsTo()`

Sekarang kita masuk ke hubungan antara tabel `mahasiswa` ke tabel `jurusans`. Dalam *relationship one to many*, hubungan yang terjadi adalah satu mahasiswa **dimiliki oleh** satu jurusan, atau dalam bahasa inggris: mahasiswa **belongs to** jurusan.

Yup, hubungan ini sama persis seperti antara tabel `nilais` dengan tabel `mahasiswa` di bab *relationship one to one*.

Dalam *relationship one to many*, hubungan antara tabel kedua ke tabel utama tetap **one to one**. Yang *one to many* adalah hubungan antara tabel utama ke tabel kedua.

Jika kita ingin membuat hubungan tabel kedua ke tabel utama juga *one to many*, maka itu akan menjadi *relationship many to many* (akan di bahas dalam bab selanjutnya).

Untuk membuat relationship mahasiswa **belongs to** jurusan, silahkan buka file model `Mahasiswa` lalu tambah satu method `jurusan()`:

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
13     public function jurusan()
14     {
15         return $this->belongsTo('App\Models\Jurusan');
16     }
17 }
```

Tambahan kode program ada di baris 13 – 16, yakni sebuah method bernama `jurusan()` yang berisi perintah `return $this->belongsTo('App\Models\Jurusan')`. Inilah cara kita memberitahu Laravel apa jenis hubungan dari tabel `mahasiswa` dengan tabel `jurusans`, yakni mahasiswa **belongs to** jurusan.

Nama method `jurusan()` menggunakan kata *singular* tanpa tambahan 's' karena hubungan yang terjadi adalah *one to one*. Sebagai argument dari method `$this->belongsTo()`, berisi

Eloquent Relationship: One to Many

namespace file model yang terhubung, yakni 'App\Models\Jurusan'.

Dengan tambahan method ini, kita sudah bisa menjalankan perintah *eloquent relationship one to one* dari tabel `mahasiswa` ke tabel `jurusans`.

Berikut daftar route yang akan di bahas:

routes\web.php

```
1 <?php
2 ...
3
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::prefix('/mahasiswa')->group(function () {
7     Route::get('/find', [MahasiswaController::class, 'find']);
8     Route::get('/where', [MahasiswaController::class, 'where']);
9     Route::get('/where-chaining', [MahasiswaController::class, 'whereChaining']);
10    Route::get('/has', [MahasiswaController::class, 'has']);
11    Route::get('/where-has', [MahasiswaController::class, 'whereHas']);
12    Route::get('/doesnt-have', [MahasiswaController::class, 'doesntHave']);
13
14    Route::get('/associate', [MahasiswaController::class, 'associate']);
15
16    Route::get('/associate-update', [MahasiswaController::class, 'associateUpdate']);
17
18    Route::get('/delete', [MahasiswaController::class, 'delete']);
19    Route::get('/dissociate', [MahasiswaController::class, 'dissociate']);
20});
```

Semua route akan mengakses `MahasiswaController`, sehingga kode program akan nantinya kita buat di file `app\Http\Controllers\MahasiswaController.php`.

Sebagai data awal, silahkan refresh tabel `jurusans` dan `mahasiswa` beserta seeder dengan perintah `php artisan migrate:fresh --seed`.

Menampilkan Satu Gabungan Data

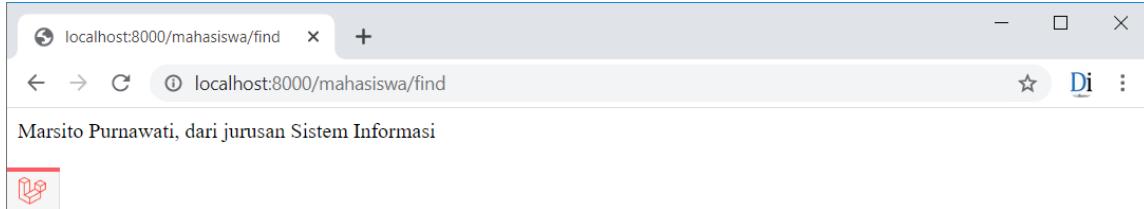
Karena kita sudah mendefinisikan relationship, maka setiap object `Mahasiswa` akan memiliki property `jurusan` yang bisa diakses. Berikut contohnya:

app\Http\Controllers\MahasiswaController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use App\Models\Jurusan;
5 use App\Models\Mahasiswa;
6
7 class MahasiswaController extends Controller
8 {
9     public function find()
10    {
```

Eloquent Relationship: One to Many

```
11     $mahasiswa = Mahasiswa::find(2);
12
13     echo "$mahasiswa->nama, dari jurusan {$mahasiswa->jurusan->nama}";
14 }
15 }
```



Gambar: Menampilkan data mahasiswa

Pada baris 11 saya mengisi variabel \$mahasiswa dengan object Mahasiswa yang memiliki id = 2. Lalu di baris 13 nama mahasiswa di echo beserta jurusannya. Informasi mengenai nama jurusan bisa diakses dari property \$mahasiswa->jurusan->nama.

Berikut contoh lain:

app\Http\Controllers\MahasiswaController.php

```
1 public function where()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'like', 'M%')
4             ->orderBy('nama', 'desc')->firstOrFail();
5
6     echo "$mahasiswa->nama, dari jurusan {$mahasiswa->jurusan->nama}";
7     // Mustofa Simanjuntak, dari jurusan Ilmu Komputer
8 }
```

Sekarang di baris 3 saya mencari satu mahasiswa yang namanya diawali dengan huruf M, lalu di urutkan berdasarkan nama secara menurun. Informasi mengenai nama jurusan bisa diakses dari \$mahasiswa->jurusan->nama.

Nilai kolom milik tabel jurusans juga bisa langsung diakses dalam satu baris (*chaining*):

app\Http\Controllers\MahasiswaController.php

```
1 public function whereChaining()
2 {
3     echo Mahasiswa::where('nama', 'Mahdi Rajata')->firstOrFail()->jurusan->nama;
4     // Sistem Informasi
5 }
```

Dalam kode program ini saya langsung men-echo nama jurusan dari mahasiswa Mahdi Rajata.

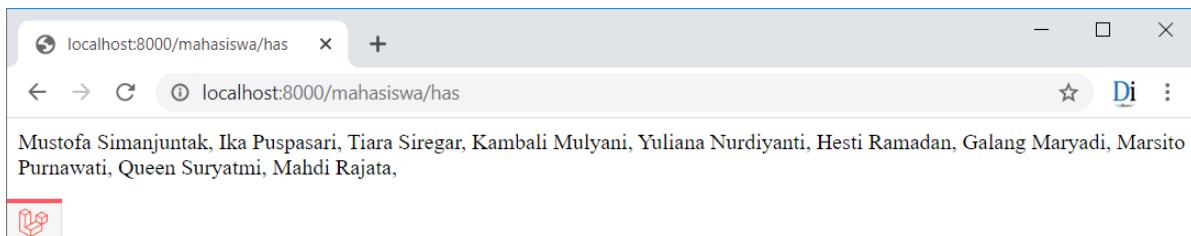
Menampilkan Batasan has() dan whereHas()

Method has() bisa dipakai untuk menampilkan semua mahasiswa yang memiliki nilai di tabel jurusans:

Eloquent Relationship: One to Many

app\Http\Controllers\MahasiswaController.php

```
1 public function has()
2 {
3     $mahasiswas = Mahasiswa::has('jurusan')->get();
4     foreach ($mahasiswas as $mahasiswa) {
5         echo "$mahasiswa->nama, ";
6     }
7 }
```



Gambar: Menampilkan data mahasiswa yang memiliki jurusan

Karena syarat *referential integrity*, kode program di atas akan menampilkan semua mahasiswa yang ada di tabel `mahasiswas`. Dalam tabel `mahasiswas`, kita tidak mengizinkan ada mahasiswa yang tidak memiliki jurusan.

Untuk pembatasan yang lebih kompleks, bisa menggunakan method `whereHas()`:

app\Http\Controllers\MahasiswaController.php

```
1 public function whereHas()
2 {
3     $mahasiswas = Mahasiswa::whereHas('jurusan', function ($query) {
4         $query->where('nama', 'Sistem Informasi');
5     })->get();
6     foreach ($mahasiswas as $mahasiswa) {
7         echo "$mahasiswa->nama, ";
8     }
9     // Marsito Purnawati, Queen Suryatmi, Mahdi Rajata,
10 }
```

Kode di atas akan menampilkan semua mahasiswa yang memilih jurusan Sistem Informasi. Batasan ini sebenarnya lebih pas jika dilakukan dari object Jurusan.

Menampilkan Batasan `doesntHave()`

Method `doesntHave()` bisa dipakai untuk menampilkan semua mahasiswa yang **tidak memiliki** jurusan. Berikut contoh penggunaannya:

app\Http\Controllers\MahasiswaController.php

```
1 public function doesntHave()
2 {
3     $mahasiswas = Mahasiswa::doesntHave('jurusan')->get();
4     foreach ($mahasiswas as $mahasiswa) {
5         echo $mahasiswa->nama.", ";
```

Eloquent Relationship: One to Many

```
6     }
7 }
```

Kode program di atas tidak akan menampilkan data apapun karena kita tidak mengizinkan mahasiswa tanpa jurusan.

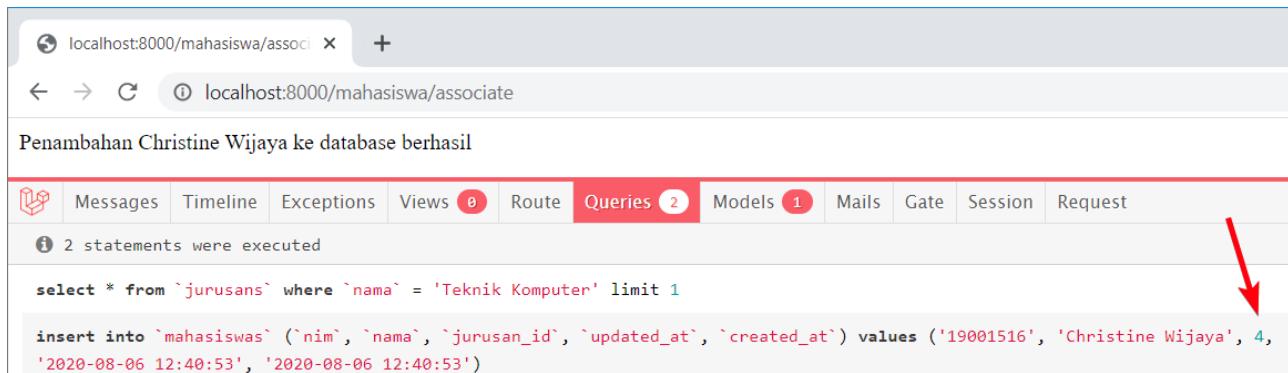
Menginput Data Relationship

Kita bisa menginput data baru ke dalam tabel `mahasiswa` sepanjang kolom `jurusan_id` diisi dengan `id` jurusan yang sudah ada sebelumnya.

Sama seperti pada praktik *one to one relationship*, method `associate()` bisa dipakai untuk menghubungkan kedua tabel. Berikut contoh prakteknya:

app\Http\Controllers\MahasiswaController.php

```
1 public function associate()
2 {
3     $jurusan = Jurusan::where('nama', 'Teknik Komputer')->first();
4
5     $mahasiswa = new Mahasiswa;
6     $mahasiswa->nim = '19001516';
7     $mahasiswa->nama = 'Christine Wijaya';
8
9     $mahasiswa->jurusan()->associate($jurusan);
10    $mahasiswa->save();
11
12    echo "Penambahan $mahasiswa->nama ke database berhasil";
13 }
```



Gambar: Menginput mahasiswa yang dihubungkan dengan method associate()

Di baris 3, variabel `$jurusan` akan berisi object dari jurusan Teknik Komputer. Lalu di baris 5 – 7 saya membuat sebuah object mahasiswa baru yang disimpan ke dalam variabel `$mahasiswa`. Sampai di sini kita tidak bisa langsung menyimpan `$mahasiswa` karena perlu mengisi kolom `jurusan_id`.

Perintah `$mahasiswa->jurusan()->associate($jurusan)` pada baris 9 di pakai untuk menghubungkan object `$mahasiswa` dengan `$jurusan`. Dengan perintah ini, property `jurusan_id` milik `$mahasiswa` akan langsung berisi `id` dari object `$jurusan`.

Eloquent Relationship: One to Many

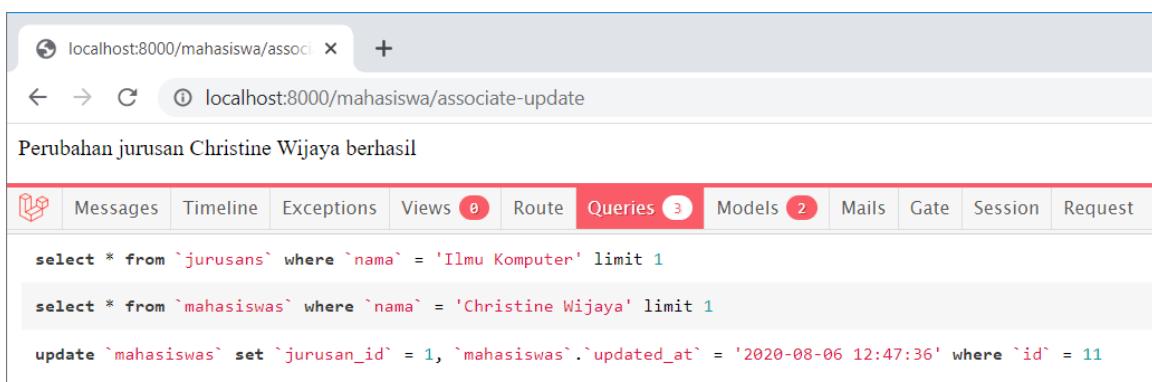
Setelah itu simpan mahasiswa dengan menjalankan perintah `$mahasiswa->save()`. Dari tampilan Laravel debugbar bisa terlihat kolom `jurusan_id` untuk tabel `mahasiswa` berisi angka 4 yang berasal dari object `$jurusan`.

Mengupdate Data Relationship

Method `associate()` + `save()` yang baru saja kita coba juga bisa dipakai untuk proses update:

app\Http\Controllers\MahasiswaController.php

```
1 public function associateUpdate()
2 {
3     $jurusan = Jurusan::where('nama', 'Ilmu Komputer')->first();
4     $mahasiswa = Mahasiswa::where('nama', 'Christine Wijaya')->first();
5
6     $mahasiswa->jurusan()->associate($jurusan);
7     $mahasiswa->save();
8
9     echo "Perubahan jurusan $mahasiswa->nama berhasil";
10 }
```



Gambar: Mengupdate mahasiswa yang dihubungkan dengan method `associate()`

Perbedaan dengan contoh sebelumnya ada di baris 4, dimana kali ini saya mencari mahasiswa yang sudah ada di tabel `misiswas`. Pemanggilan method `associate()` di baris 6 akan menghubungkan `$mahasiswa` dengan `$jurusan`. Hasilnya, jurusan `Christine Wijaya` akan berubah dari Teknik Komputer ke Ilmu Komputer.

Menghapus Data Relationship

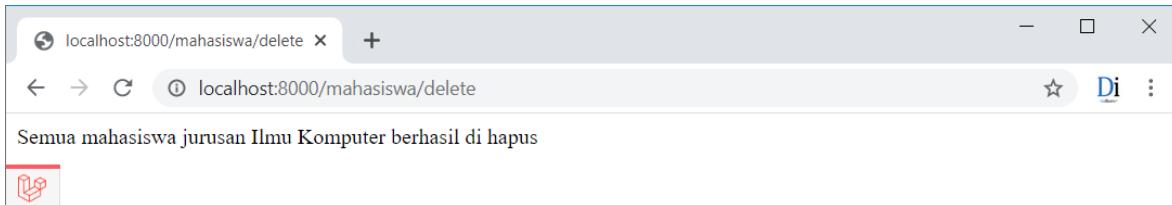
Isi tabel `misiswas` bisa dihapus dengan method `delete()` tanpa terpengaruh batasan *referential integrity*. Berikut contoh penggunaannya:

app\Http\Controllers\MahasiswaController.php

```
1 public function delete()
2 {
3     $misiswas = Mahasiswa::whereHas('jurusan', function ($query) {
4         $query->where('nama', 'Ilmu Komputer');
5     })->get();
```

Eloquent Relationship: One to Many

```
6      foreach ($mahasiswa as $mahasiswa) {
7          $mahasiswa->delete();
8      }
9
10     echo "Semua mahasiswa jurusan Ilmu Komputer berhasil di hapus";
11 }
12 }
```



Gambar: Menghapus semua mahasiswa yang memilih jurusan Ilmu Komputer

Method `Mahasiswa::whereHas()` di baris 3 akan mengembalikan collection dari semua mahasiswa yang memilih jurusan Ilmu Komputer. Kemudian perulangan `foreach` di baris 7 – 8 akan menghapus semua data mahasiswa menggunakan perintah `$mahasiswa->delete()`.

Memutus Hubungan Relationship

Laravel menyediakan method `dissociate()` yang bisa dipakai untuk memutus hubungan antara tabel utama dengan tabel kedua. Maksud dari "memutus hubungan" di sini adalah mengosongkan isi kolom *foreign key* di tabel kedua. Dalam praktek kita, kolom yang dimaksud adalah kolom `jurusan_id` milik tabel `mahasiswa`.

Berikut contoh penggunaan method `dissociate()`:

app\Http\Controllers\MahasiswaController.php

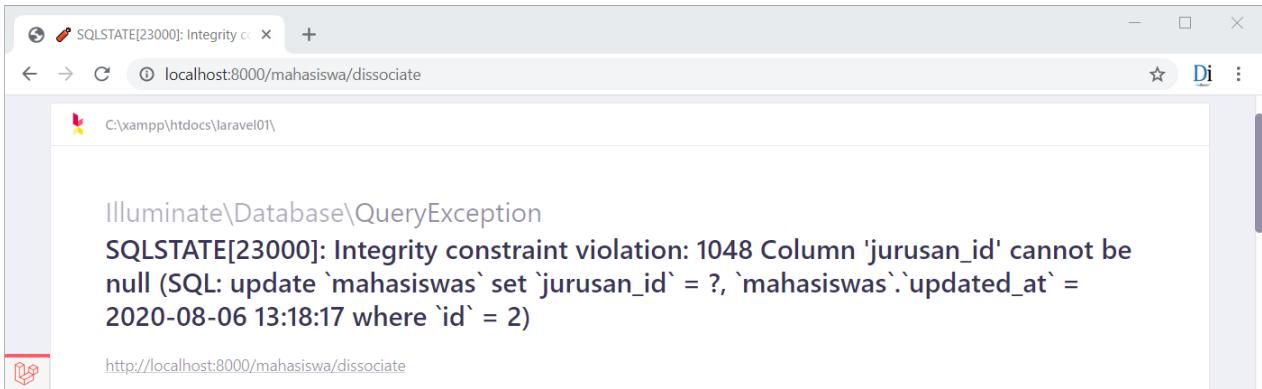
```
1  public function dissociate()
2  {
3      $mahasiswa = Mahasiswa::whereHas('jurusan', function ($query) {
4          $query->where('nama', 'Sistem Informasi');
5      })->get();
6
7      foreach ($mahasiswa as $mahasiswa) {
8          $mahasiswa->jurusan()->dissociate();
9          $mahasiswa->save();
10         echo "Pengosongan jurusan untuk $mahasiswa->nama_mahasiswa berhasil <br>";
11     }
12 }
```

Perintah di baris 3 – 5 saya pakai untuk mencari semua mahasiswa yang memilih jurusan Sistem Informasi. Hasilnya berbentuk collection yang terdiri dari beberapa object Mahasiswa.

Karena berbentuk collection, kita perlu perulangan `foreach`. Untuk setiap object Mahasiswa, jalankan method `$mahasiswa->jurusan()->dissociate()` yang akan mengosongkan property `jurusan_id`. Kemudian simpan perubahan ini dengan method `$mahasiswa->save()`.

Eloquent Relationship: One to Many

Berikut hasil kode program di atas:



Gambar: Error method dissociate()

Error SQLSTATE[23000]: Integrity constraint violation sebenarnya sudah pernah kita temui sebelumnya. Error ini terjadi karena pelanggaran terhadap aturan *referential integrity*. MySQL tidak mengizinkan kolom `jurusan_id` kosong, namun inilah yang coba dilakukan oleh method `dissociate()`.

Agar kode di atas bisa berjalan, kita harus mengubah pendefinisian kolom `jurusan_id` di file migration. Silahkan buka file migration tabel `mahasiswa` lalu tukar satu baris berikut:

database\migrations\<timestamp>_create_mahasiswa_table.php

```
1 public function up()
2 {
3     Schema::create('mahasiswa', function (Blueprint $table) {
4         ...
5         $table->foreignId('jurusan_id')->constrained()->onDelete('cascade');
6         ...
7     });
8 }
```

Menjadi:

database\migrations\<timestamp>_create_mahasiswa_table.php

```
1 public function up()
2 {
3     Schema::create('mahasiswa', function (Blueprint $table) {
4         ...
5         $table->foreignId('jurusan_id')->nullable()->constrained()
6             ->onDelete('cascade');
7         ...
8     });
9 }
```

Terdapat tambahan perintah `nullable()` ke kolom `jurusan_id`. Ini dipakai agar kolom `jurusan_id` boleh diisi dengan nilai `NULL` (tidak berisi data apa-apa).

Eloquent Relationship: One to Many

Agar perubahan ini efektif, generate ulang file migration dan juga seeder dengan perintah `php artisan migrate:fresh --seed` lalu jalankan kembali kode sebelumnya:

```
localhost:8000/mahasiswa/dissoc
localhost:8000/mahasiswa/dissociate

Pengosongan jurusan untuk Marsito Purnawati berhasil
Pengosongan jurusan untuk Queen Suryatmi berhasil
Pengosongan jurusan untuk Mahdi Rajata berhasil

Messages Timeline Exceptions Views 0 Route Queries 4 Models 3 Mails Gate Session Request
4 statements were executed

select * from `mahasiswa` where exists (select * from `jurusans` where `mahasiswa`.`jurusan_id` = `jurusans`.`id` and `nama` = 'Sistem Informasi')

update `mahasiswa` set `jurusan_id` = '', `mahasiswa`.`updated_at` = '2020-08-08 04:52:11' where `id` = 2

update `mahasiswa` set `jurusan_id` = '', `mahasiswa`.`updated_at` = '2020-08-08 04:52:11' where `id` = 4

update `mahasiswa` set `jurusan_id` = '', `mahasiswa`.`updated_at` = '2020-08-08 04:52:11' where `id` = 10
```

Gambar: Memutus hubungan tabel mahasiswa dengan jurusans

Sip, tidak ada error. Ketika diperiksa ke dalam tabel `mahasiswa`, kolom `jurusan_id` untuk ketiga mahasiswa tersebut sudah berganti dengan nilai `NULL`:

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswa;
+----+-----+-----+-----+-----+-----+
| id | nim   | nama            | jurusan_id | created_at    | updated_at   |
+----+-----+-----+-----+-----+-----+
| 1  | 10300234 | Mustofa Simanjuntak | 1          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 2  | 10451982 | Marsito Purnawati | NULL       | 2020-08-06 13:38:47 | 2020-08-06 13:38:53 |
| 3  | 10980764 | Ika Puspasari     | 1          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 4  | 10605319 | Queen Suryatmi    | NULL       | 2020-08-06 13:38:47 | 2020-08-06 13:38:53 |
| 5  | 10438572 | Yuliana Nurdyanti | 2          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 6  | 10737995 | Tiara Siregar      | 1          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 7  | 10531551 | Kambali Mulyani    | 1          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 8  | 10155818 | Hesti Ramadan      | 2          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 9  | 10274992 | Galang Maryadi     | 2          | 2020-08-06 13:38:47 | 2020-08-06 13:38:47 |
| 10 | 10228263 | Mahdi Rajata       | NULL       | 2020-08-06 13:38:47 | 2020-08-06 13:38:53 |
+---+-----+-----+-----+-----+-----+
10 rows in set (0.001 sec)
```

Gambar: Tiga mahasiswa memiliki nilai `NULL` untuk kolom `jurusan_id`

Inilah efek dari method `dissociate()`, yakni memutus hubungan relationship dengan cara menghapus kolom *foreign key* dari tabel kedua. Agar efek ini bisa berjalan, kita harus set kolom *foreign key* supaya bisa menerima nilai `NULL` menggunakan method `nullable()`.

Mengizinkan nilai `NULL` ke *foreign key* bukanlah desain database yang baik. Seharusnya kita tetap mengikuti konsep *referential integrity*, dimana kolom *foreign key* hanya bisa diisi dengan `id` dari kolom utama dan tidak boleh kosong (tidak boleh berisi `NULL`).

Dalam bab ini kita telah membahas *relationship one to many*. Jenis relationship ini menjadi salah satu yang paling sering dipakai. Lanjut, kita akan masuk ke **Eloquent Relationship Many to Many**.

14. Eloquent Relationship: Many to Many

Kita masuk ke jenis ketiga dari eloquent relationship, yakni **Relationship Many to Many**. Relationship many to many memang lebih kompleks dibandingkan relationship one to one dan relationship one to many, namun sangat menarik untuk dibahas.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

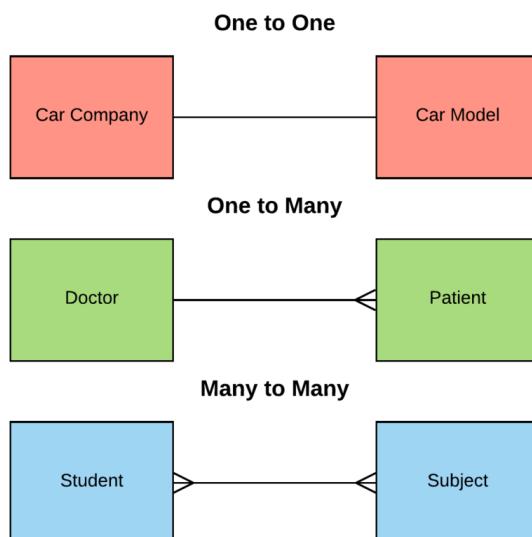
```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

14.1. Pengertian Many to Many Relationship

Maksud dari **many to many relationship** adalah, satu baris data di tabel utama bisa terhubung dengan satu atau lebih data di tabel kedua. Sebaliknya, satu baris data di tabel kedua, juga bisa terhubung dengan satu atau lebih data di tabel utama.

Dalam diagram **ERD** (Entity Relationship Diagram), relationship many to many digambarkan dengan garis bercabang di kedua tabel:



Gambar: Diagram ERD dari 3 jenis relationship (sumber gambar: <https://commons.wikimedia.org>)

Contoh dari konsep ini adalah satu mahasiswa bisa mengambil banyak mata kuliah. Sebaliknya, satu mata kuliah juga bisa diambil oleh banyak mahasiswa. Atau seorang penulis bisa mengarang banyak judul buku, kemudian satu judul buku juga bisa dikarang oleh banyak penulis.

14.2. Persiapan Awal

Sebagai bahan praktek dari penerapan *eloquent relationship many to many*, saya akan buat hubungan antara mahasiswa dengan mata kuliah. Artinya kita akan butuh tabel `mahasiswa`s dan tabel `matakuliah`s.

Akan tetapi *relationship many to many* juga butuh satu tabel bantu yang dipakai sebagai "penghubung" dari tabel `mahasiswa`s dan tabel `matakuliah`s. Tabel bantu ini sering disebut sebagai **tabel pivot** (*pivot table*).

Tabel pivot diperlukan karena baik tabel `mahasiswa`s maupun tabel `matakuliah`s sama-sama bertindak sebagai tabel utama. Di kedua tabel terdapat *primary key* namun tanpa *foreign key*. Kolom *foreign key* inilah yang akan ditempatkan pada tabel pivot.

Pemberian nama tabel pivot juga punya aturan tersendiri (meskipun tidak wajib), yakni gabungan dari nama plural dari kedua tabel utama yang diurutkan berdasarkan abjad. Karena tabel utama kita adalah `mahasiswa`s dan `matakuliah`s, maka nama tabel pivot menjadi `mahasiswa_matakuliah`.

Nama tabel pivot bukan `matakuliah_mahasiswa` karena secara abjad, "**mahasiswa**" lebih dahulu dibandingkan "**matakuliah**".

Berikut struktur tabel yang akan kita buat:

- ◆ Tabel **mahasiswa**: `id`, `nim`, `nama`, `jurusan_id`, `created_at` dan `updated_at`.
- ◆ Tabel **matakuliah**: `id`, `kode`, `nama`, `jumlah_sks`, `created_at` dan `updated_at`.
- ◆ Tabel **mahasiswa_matakuliah**: `id`, `mahasiswa_id`, `matakuliah_id`, `created_at` dan `updated_at`.

Struktur tabel `mahasiswa`s sama persis seperti contoh praktek *relationship one to many*.

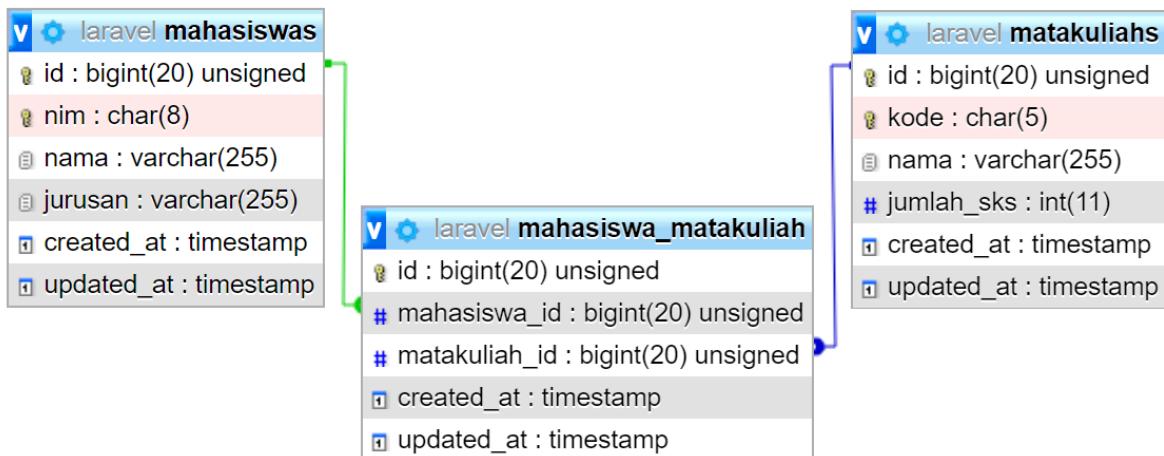
Tabel `matakuliah`s terdiri dari 4 kolom utama, yakni `id`, lalu kolom `kode` yang berisi kode unik dari sebuah mata kuliah, `nama` mata kuliah dan `jumlah_sks` dari sebuah mata kuliah.

Yang menarik ada di struktur tabel `mahasiswa_matakuliah`. Tabel pivot ini punya kolom `mahasiswa_id` yang berfungsi sebagai *foreign key* dari kolom `id` di tabel `mahasiswa`s. Serta punya kolom `matakuliah_id` yang berfungsi sebagai *foreign key* dari kolom `id` di tabel `matakuliah`s.

Jika digambarkan ke dalam diagram **ERD**, berikut hubungan antara tabel `mahasiswa`, tabel

Eloquent Relationship: Many to Many

matakuliah, serta tabel mahasiswa_matakuliah:



Gambar: Hubungan antara tabel mahasiswa, matakuliah, serta mahasiswa_matakuliah

Terlihat garis antara kolom `id` di tabel `mahasiswa` dengan kolom `mahasiswa_id` di tabel `mahasiswa_matakuliah`. Serta garis antara kolom `id` di tabel `matakuliah` dengan kolom `matakuliah_id` di tabel `mahasiswa_matakuliah`. Inilah hubungan *primary key* dengan *foreign key* dalam konsep *relationship many to many*.

Generate Data Sample

Kita akan buat ketiga tabel melalui migration serta beberapa file tambahan. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```
php artisan make:model Mahasiswa -cm  
php artisan make:model Matakuliah -cm
```

Kode di atas akan membuat file **model**, **controller** serta **migration** untuk tabel `mahasiswa` dan `matakuliah`.

Untuk tabel `mahasiswa_matakuliah` tidak perlu model dan controller, cukup migration saja:

```
php artisan make:migration create_mahasiswa_matakuliah_table
```

```
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa -cm  
Model created successfully.  
Created Migration: 2020_10_10_060759_create_mahasiswa_table  
Controller created successfully.  
  
C:\xampp\htdocs\laravel01>php artisan make:model Matakuliah -cm  
Model created successfully.  
Created Migration: 2020_10_10_060842_create_matakuliah_table  
Controller created successfully.  
  
C:\xampp\htdocs\laravel01>php artisan make:migration create_mahasiswa_matakuliah_table  
Created Migration: 2020_10_10_060901_create_mahasiswa_matakuliah_table
```

Gambar: Pembuatan model, controller serta migration

Eloquent Relationship: Many to Many

Selanjutnya buka file migration tabel `mahasiswa`s lalu rancang struktur tabel berikut:

database\migrations\<timestamp>_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('jurusan');
8         $table->timestamps();
9     });
10 }
```

Struktur tabel ini sama persis seperti praktik di bab *relationship one to one*. Lanjut, buka file migration untuk tabel `matakuliah`s dan rancang struktur tabel berikut:

database\migrations\<timestamp>_create_matakuliah_table.php

```
1 public function up()
2 {
3     Schema::create('matakuliah', function (Blueprint $table) {
4         $table->id();
5         $table->char('kode',5)->unique();
6         $table->string('nama');
7         $table->integer('jumlah_sks');
8         $table->timestamps();
9     });
10 }
```

Tidak ada sesuatu yang baru di sini. Kolom `kode` menggunakan tipe data char serta terdapat tambahan method `unique()` untuk menghindari kode yang berulang. Kolom `nama` menggunakan tipe data string, dan `jumlah_sks` dengan tipe data integer.

Serta berikut struktur file migration untuk tabel `masiswa_matakuliah`:

database\migrations\<timestamp>_create_masiswa_matakuliah_table.php

```
1 public function up()
2 {
3     Schema::create('masiswa_matakuliah', function (Blueprint $table) {
4         $table->id();
5         $table->foreignId('masiswa_id')->constrained()->onDelete('cascade');
6         $table->foreignId('matakuliah_id')->constrained()->onDelete('cascade');
7         $table->timestamps();
8     });
9 }
```

Seperti yang pernah kita bahas sebelumnya, tabel `masiswa_matakuliah` berisi 2 kolom utama, yakni `masiswa_id` sebagai *foreign key* dari tabel `masiswa`s, serta kolom `matakuliah_id` sebagai *foreign key* dari tabel `matakuliah`s.

Eloquent Relationship: Many to Many

Buat ketiga tabel dengan menjalankan proses migration: `php artisan migrate`.

Langkah selanjutnya adalah membuat beberapa data sample. Silahkan buka file `DatabaseSeeder.php` lalu modifikasi sebagai berikut:

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7 use App\Models\Mahasiswa;
8 use App\Models\Matakuliah;
9
10 class DatabaseSeeder extends Seeder
11 {
12     public function run()
13     {
14         $faker = Faker::create('id_ID');
15         $faker->seed(123);
16         $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
17
18         for ($i=0; $i<10; $i++) {
19             Mahasiswa::create(
20                 [
21                     'nim' => $faker->unique()->numerify('10#####'),
22                     'nama' => $faker->firstName()." ".$faker->lastName(),
23                     'jurusan' => $faker->randomElement($jurusan),
24                 ]
25             );
26         }
27
28         Matakuliah::create(
29             [
30                 'kode' => 'AP001',
31                 'nama' => 'Algoritma dan Pemrograman',
32                 'jumlah_sks' => 2,
33             ]
34         );
35
36         Matakuliah::create(
37             [
38                 'kode' => 'AL002',
39                 'nama' => 'Aljabar Linear',
40                 'jumlah_sks' => 2,
41             ]
42         );
43
44         Matakuliah::create(
45             [
46                 'kode' => 'KG001',
47                 'nama' => 'Kriptografi',
```

Eloquent Relationship: Many to Many

```
48         'jumlah_sks' => 2,
49     ],
50 );
51
52     Matakuliah::create(
53     [
54         'kode' => 'KD004',
55         'nama' => 'Kalkulus Dasar',
56         'jumlah_sks' => 4,
57     ]
58 );
59
60     Matakuliah::create(
61     [
62         'kode' => 'PB012',
63         'nama' => 'Pemrograman Berorientasi Objek',
64         'jumlah_sks' => 3,
65     ]
66 );
67
68 }
69 }
```

Di baris 14 – 26 saya men-generate 10 data mahasiswa untuk tabel `mahasiswas` menggunakan Faker. Kemudian di baris 28 – 66 meng-generate 5 buah data matakuliah secara manual. Jalankan seeder dengan perintah `php artisan db:seed`.

Berikut isi data di tabel `mahasiswas` dan `matakuliah`:

The screenshot shows a terminal window titled "Select MySQL Folder cmd - mysql -u root". It contains two SQL queries:

```
MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+-----+
| id | nim  | nama   | jurusan | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1  | 10300234 | Mustofa Simanjuntak | Ilmu Komputer | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 2  | 10451982 | Marsito Purnawati | Sistem Informasi | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 3  | 10980764 | Ika Puspasari | Ilmu Komputer | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 4  | 10605319 | Queen Suryatmi | Sistem Informasi | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 5  | 10438572 | Yuliana Nurdiyanti | Teknik Informatika | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 6  | 10737995 | Tiara Siregar | Ilmu Komputer | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 7  | 10531551 | Kambali Mulyani | Ilmu Komputer | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 8  | 10155818 | Hesti Ramadan | Teknik Informatika | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 9  | 10274992 | Galang Maryadi | Teknik Informatika | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 10 | 10228263 | Mahdi Rajata | Sistem Informasi | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
+----+-----+-----+-----+-----+-----+
10 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM matakuliah;
+----+-----+-----+-----+-----+
| id | kode  | nama   | jumlah_sks | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1  | AP001 | Algoritma dan Pemrograman | 2 | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 2  | AL002 | Aljabar Linear | 2 | 2020-08-09 13:39:48 | 2020-08-09 13:39:48 |
| 3  | KG001 | Kriptografi | 2 | 2020-08-09 13:39:49 | 2020-08-09 13:39:49 |
| 4  | KD004 | Kalkulus Dasar | 4 | 2020-08-09 13:39:49 | 2020-08-09 13:39:49 |
| 5  | PB012 | Pemrograman Berorientasi Objek | 3 | 2020-08-09 13:39:49 | 2020-08-09 13:39:49 |
+----+-----+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswas dan matakuliah

Tabel `mahasiswa_matakuliah` saat ini masih kosong dan akan kita isi menggunakan perintah eloquent sesaat lagi.

14.3. Eloquent Relationship belongsToMany()

Hubungan *relationship* antara tabel `mahasiswa` dengan tabel `matakuliah`s adalah **many to many**. Dalam Eloquent, hubungan ini di definisikan sebagai *mahasiswa belongs to many matakuliah*. Untuk membuatnya, kita akan pakai method `belongsToMany()`.

Silahkan buka file model `Mahasiswa.php` lalu tambah satu method 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     public function matakuliah()
12     {
13         return $this->belongsToMany('App\Models\Matakuliah');
14     }
15 }
```

Tambahan kode ada di baris 11 – 14, yakni sebuah method bernama `matakuliah()` yang berisi perintah `return $this->belongsToMany('App\Models\Matakuliah')`. Inilah cara kita memberitahu Laravel jenis hubungan dari tabel `mahasiswa` ke tabel `matakuliah`, yaitu *mahasiswa belongs to many matakuliah*.

Nama method `matakuliah()` menggunakan kata *plural* dengan tambahan 's' karena hubungan yang terjadi adalah *many to many*. Sebagai argument dari method `$this->belongsToMany()`, berisi namespace file model yang terhubung, yakni '`App\Models\Matakuliah`'.

Dengan tambahan method tersebut, kita sudah bisa menjalankan berbagai perintah *eloquent relationship many to many* dari tabel `mahasiswa` ke tabel `matakuliah`.

Berikut daftar route yang akan di bahas:

routes\web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\MahasiswaController;
5
6 Route::prefix('/mahasiswa')->group(function () {
7
8     Route::get('/all', [MahasiswaController::class, 'all']);
9     Route::get('/attach', [MahasiswaController::class, 'attach']);

```

Eloquent Relationship: Many to Many

```
10 Route::get('/attach-array', [MahasiswaController::class, 'attachArray']);
11 Route::get('/attach-where', [MahasiswaController::class, 'attachWhere']);
12 Route::get('/tampil', [MahasiswaController::class, 'tampil']);
13 Route::get('/relationship-count', [MahasiswaController::class, 'relationshipCount']);
14 Route::get('/detach', [MahasiswaController::class, 'detach']);
15 Route::get('/sync', [MahasiswaController::class, 'sync']);
16 Route::get('/sync-lagi', [MahasiswaController::class, 'syncLagi']);
17 Route::get('/sync-chaining', [MahasiswaController::class, 'syncChaining']);
18 Route::get('/sync-without', [MahasiswaController::class, 'syncWithout']);
19 Route::get('/toggle', [MahasiswaController::class, 'toggle']);
20 Route::get('/delete', [MahasiswaController::class, 'delete']);
21
22 });
```

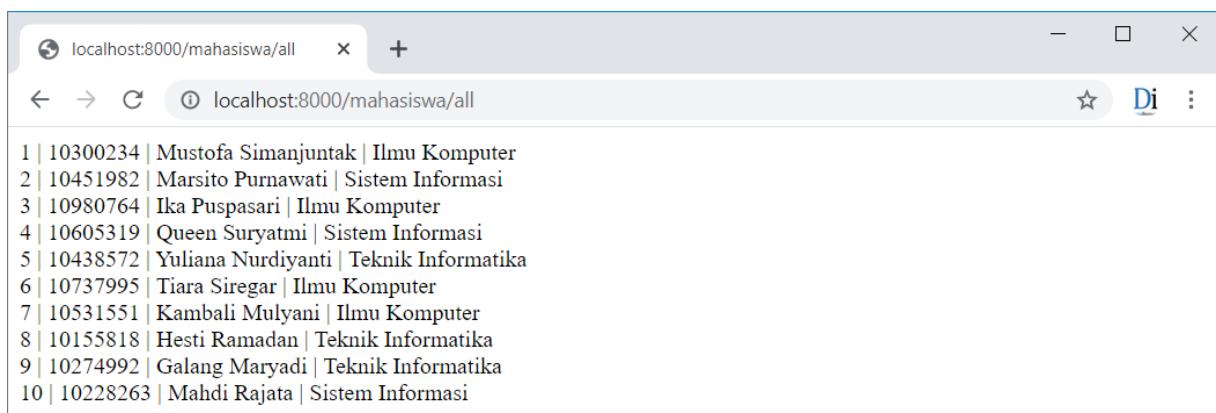
Sebelum membuat kode program, saya juga akan menginstall Laravel Debugbar dengan perintah `composer require barryvdh/laravel-debugbar --dev`.

Menampilkan Semua Data Mahasiswa

Route pertama di siapkan untuk menampilkan semua data yang ada di tabel `mahasiswas`:

app\Http\Controllers\MahasiswaController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use App\Models\Mahasiswa;
5 use App\Models\Matakuliah;
6
7 class MahasiswaController extends Controller
8 {
9     public function all()
10    {
11        $mahasiswas = Mahasiswa::all();
12        foreach ($mahasiswas as $mahasiswa) {
13            echo "$mahasiswa->id | $mahasiswa->nim | ";
14            echo "$mahasiswa->nama | $mahasiswa->jurusan <br>";
15        }
16    }
17 }
```



Gambar: Menampilkan isi tabel mahasiswas

Ini merupakan kode eloquent biasa tanpa menggunakan relationship.

Menginput Data Relationship

Kita belum bisa menampilkan data relationship karena tabel `mahasiswa_matakuliah` belum berisi nilai apapun. Idenya adalah, jika ada mahasiswa yang ingin mengambil matakuliah, `id` mahasiswa tersebut dan `id` matakuliah akan dicatat ke dalam tabel `mahasiswa_matakuliah`.

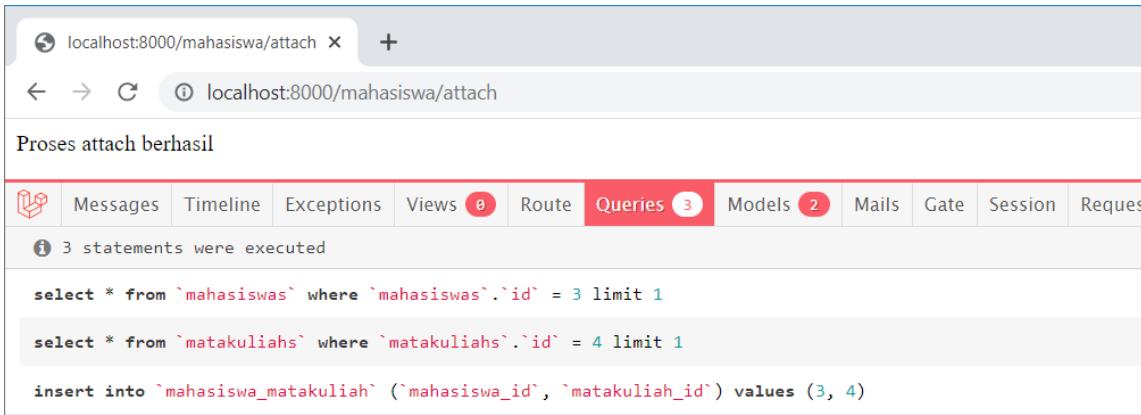
Eloquent menyediakan method `attach()` untuk keperluan ini:

app\Http\Controllers\MahasiswaController.php

```
1 public function attach()
2 {
3     $mahasiswa = Mahasiswa::find(3);
4     $matakuliah = Matakuliah::find(4);
5
6     // Lakukan proses attach, yakni hubungkan mahasiswa dan matakuliah
7     $mahasiswa->matakuliahhs()->attach($matakuliah);
8
9     echo "Proses attach berhasil";
10 }
```

Di baris 3 dan 4 saya mencari satu object mahasiswa dan satu object matakuliah. Kode ini bisa saja lebih kompleks misalnya mencari mahasiswa atau matakuliah dengan batasan `where()`. Sepanjang variabel `$mahasiswa` dan `$matakuliah` berisi satu object, maka tidak ada masalah.

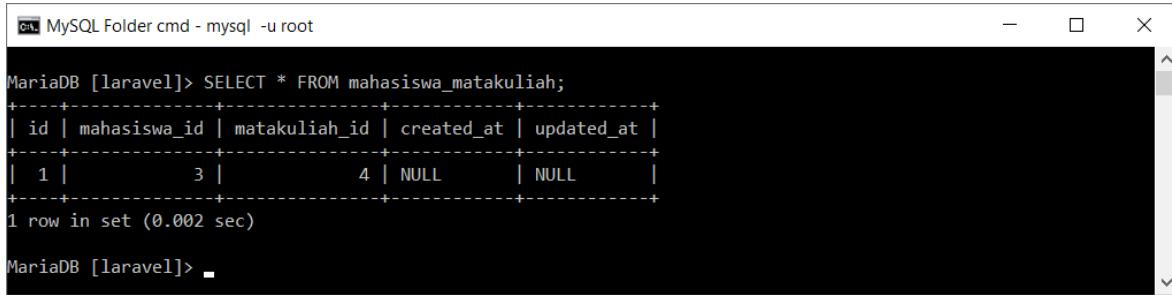
Untuk menghubungkan `$mahasiswa` dengan `$matakuliah`, jalankan method `attach()` seperti di baris 7. Ini artinya mahasiswa dengan id 3 mengambil satu matakuliah yang memiliki id 4.



Gambar: Proses attach satu mahasiswa dengan satu matakuliah

Dari laravel Debugbar bisa dipelajari query apa yang dijalankan oleh Eloquent. Agar lebih jelas, berikut isi tabel `mahasiswa_matakuliah` setelah menjalankan method `attach()`:

Eloquent Relationship: Many to Many



The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". The command "SELECT * FROM mahasiswa_matakuliah;" is run, resulting in the following output:

id	mahasiswa_id	matakuliah_id	created_at	updated_at
1	3	4	NULL	NULL

1 row in set (0.002 sec)

MariaDB [laravel]> .

Gambar: Isi tabel mahasiswa_matakuliah setelah di attach

Terlihat 1 baris data yang berisi pasangan `mahasiswa_id` 3 dengan `matakuliah_id` 4. Inilah fungsi dari tabel pivot `mahasiswa_matakuliah`, yakni sebagai tempat untuk mencatat semua *relationship* yang terjadi.

Menambah Method `withTimestamps()`

Dari tampilan tabel `mahasiswa_matakuliah` sebelumnya, kolom `created_at` dan `update_at` ternyata masih berisi nilai `NULL`. Ini memang bawaan Eloquent dimana ketika melakukan proses `attach()`, kolom *timestamp* tidak langsung terisi.

Jika kita ingin agar Eloquent juga mengisi kolom `created_at` dan `update_at`, maka perlu menambah method `withTimestamps()` ke dalam definisi *relationship*. Caranya, buka kembali file model `Mahasiswa.php` lalu ubah definisi hubungan dari sebelumnya:

app\Models\Mahasiswa.php

```
1 public function matakuliahs()
2 {
3     return $this->belongsToMany('App\Models\Matakuliah');
4 }
```

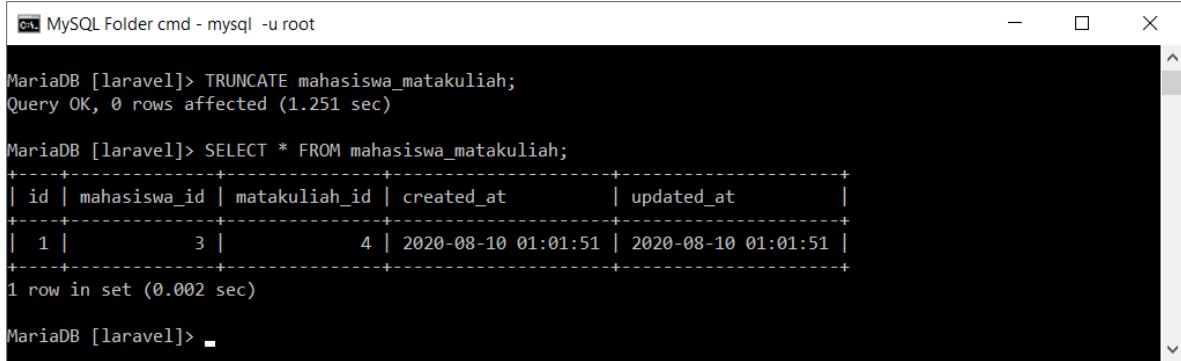
Menjadi:

app\Models\Mahasiswa.php

```
1 public function matakuliahs()
2 {
3     return $this->belongsToMany('App\Models\Matakuliah')->withTimestamps();
4 }
```

Save file model `Mahasiswa.php`, kosongkan tabel `mahasiswa_matakuliah` (bisa dengan menjalankan perintah `TRUNCATE` dari cmd MySQL client), lalu akses kembali URL `localhost:8000/mahasiswa/attach`.

Setelah itu periksa ulang ke dalam tabel:



The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". It displays the following SQL session:

```
MariaDB [laravel]> TRUNCATE mahasiswa_matakuliah;
Query OK, 0 rows affected (1.251 sec)

MariaDB [laravel]> SELECT * FROM mahasiswa_matakuliah;
+----+-----+-----+-----+
| id | mahasiswa_id | matakuliah_id | created_at           | updated_at           |
+----+-----+-----+-----+
| 1  |         3    |        4    | 2020-08-10 01:01:51 | 2020-08-10 01:01:51 |
+----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi tabel mahasiswa_matakuliah setelah di attach + timestamp

Hasilnya, kolom `created_at` dan `update_at` sudah langsung terisi.

Fitur ini memang tidak wajib tapi bisa dipakai dalam beberapa situasi, misalnya jika kita ingin menampilkan informasi tanggal berapa seorang mahasiswa memilih mata kuliah tertentu.

Menginput Banyak Data Relationship

Method `attach()` sebenarnya juga bisa menerima argument berbentuk collection. Dengan demikian, banyak data bisa dihubungkan sekaligus:

app\Http\Controllers\MahasiswaController.php

```
1 public function attachArray()
2 {
3
4     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
5     $matakuliah = Matakuliah::find([1, 2, 3]);
6
7     $mahasiswa->matakuliah->attach($matakuliah);
8
9     echo "Proses attach berhasil";
10 }
```

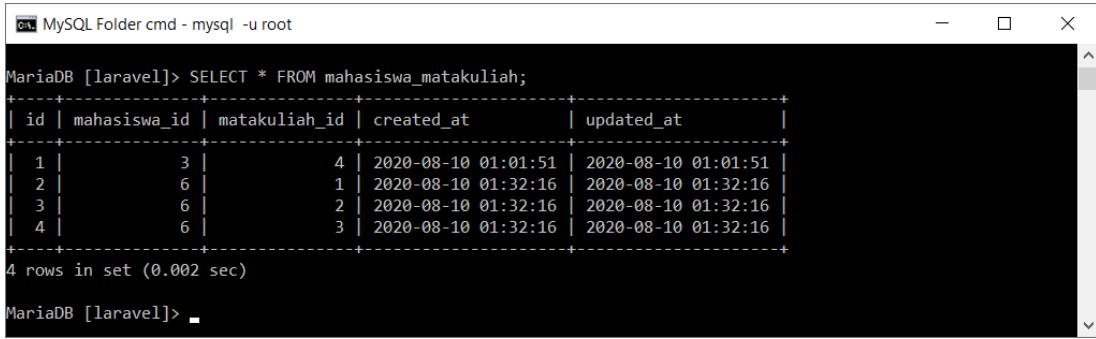
Di baris 4 saya mencari mahasiswa bernama Tiara Siregar, lalu menyimpannya ke dalam variabel `$mahasiswa`.

Kemudian di baris 5 mengambil 3 object Matakuliah yang memiliki id 1, 2 dan 3. Karena terdiri dari 3 object, maka variabel `$matakuliah` akan berbentuk collection.

Proses `attach()` dilakukan dengan cara yang sama seperti sebelumnya, yakni dengan perintah `$mahasiswa->matakuliah->attach($matakuliah)`.

Silahkan akses URL `localhost:8000/mahasiswa/attach-array` di web browser, lalu periksa kembali isi tabel `mahasiswa_matakuliah`:

Eloquent Relationship: Many to Many



The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM mahasiswa_matakuliah;". The output is a table with the following data:

id	mahasiswa_id	matakuliah_id	created_at	updated_at
1	3	4	2020-08-10 01:01:51	2020-08-10 01:01:51
2	6	1	2020-08-10 01:32:16	2020-08-10 01:32:16
3	6	2	2020-08-10 01:32:16	2020-08-10 01:32:16
4	6	3	2020-08-10 01:32:16	2020-08-10 01:32:16

4 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswa_matakuliah setelah attach collection

Dari tabel ini bisa terlihat mahasiswa dengan id 6 mengambil 3 buah mata kuliah, yakni matakuliah dengan id 1, 2 dan 3.

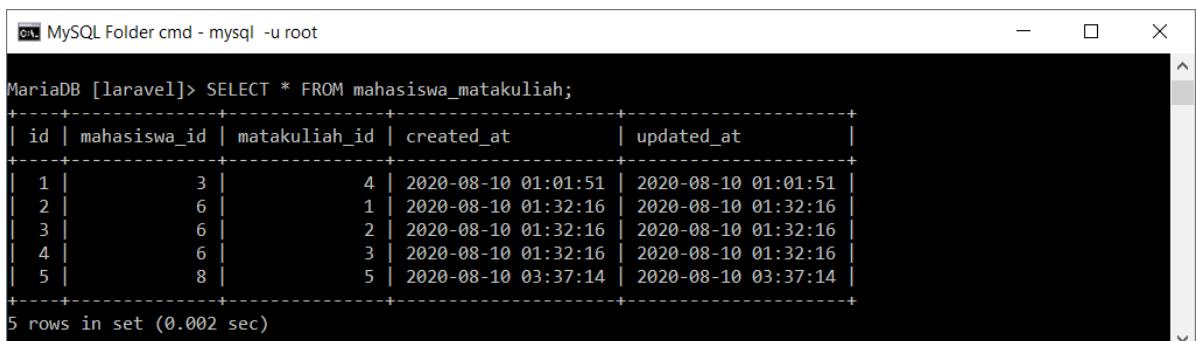
Berikut contoh lain dari proses attach():

app\Http\Controllers\MahasiswaController.php

```
1 public function attachWhere()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Hesti Ramadan')->first();
4     $matakuliahs = Matakuliah::where('jumlah_sks', 3)->get();
5
6     $mahasiswa->matakuliahs()->attach($matakuliahs);
7
8     echo "Proses attach berhasil";
9 }
```

Kode ini bisa dibaca: Hesti Ramadan ingin mengambil semua mata kuliah yang memiliki jumlah sks sebanyak 3.

Akses URL `localhost:8000/mahasiswa/attach-where` lalu cek kembali ke database:



The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM mahasiswa_matakuliah;". The output is a table with the following data:

id	mahasiswa_id	matakuliah_id	created_at	updated_at
1	3	4	2020-08-10 01:01:51	2020-08-10 01:01:51
2	6	1	2020-08-10 01:32:16	2020-08-10 01:32:16
3	6	2	2020-08-10 01:32:16	2020-08-10 01:32:16
4	6	3	2020-08-10 01:32:16	2020-08-10 01:32:16
5	8	5	2020-08-10 03:37:14	2020-08-10 03:37:14

5 rows in set (0.002 sec)

Gambar: Isi tabel mahasiswa_matakuliah setelah attach where

Terdapat 1 tambahan data di baris terakhir, dimana ternyata hanya ada 1 mata kuliah yang memiliki jumlah sks 3.

Menampilkan Data Relationship

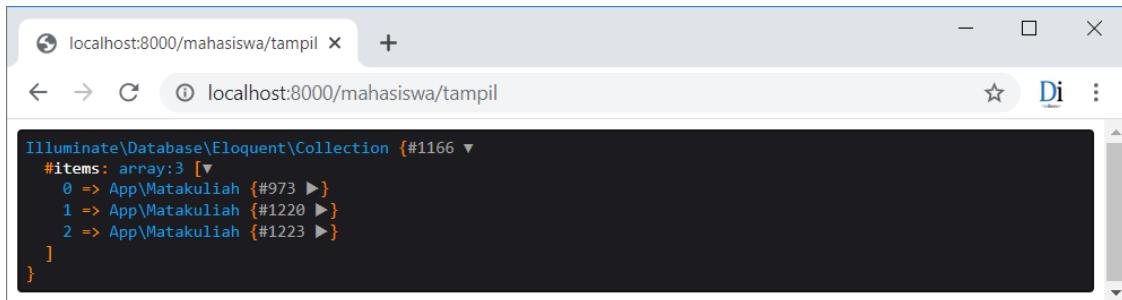
Tabel `mahasiswa_matakuliah` sudah berisi beberapa data. Sekarang saatnya menampilkan

data-data ini. Sebagai contoh, bagaimana cara menampilkan semua mata kuliah yang sudah diambil oleh Tiara Siregar?

Cara yang dipakai sebenarnya sama seperti *eloquent relationship* biasa, yakni dengan mengakses property `matakuliahs` dari object Mahasiswa:

app\Http\Controllers\MahasiswaController.php

```
1 public function tampil()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4     dump($mahasiswa->matakuliahs);
5 }
```



Gambar: Hasil `dump($mahasiswa->matakuliahs)`

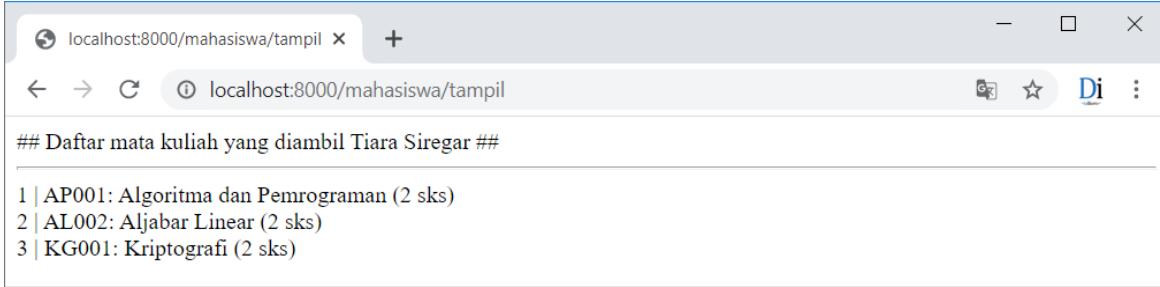
Yang perlu diperhatikan adalah, karena kita memakai *relationship many to many*, maka property `$mahasiswa->matakuliahs` akan selalu berisi collection meskipun itu hanya 1 object Matakuliah.

Seperti biasa, jika berurusan dengan collection, perulangan `foreach` adalah kuncinya:

app\Http\Controllers\MahasiswaController.php

```
1 public function tampil()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4
5     echo "## Daftar mata kuliah yang diambil $mahasiswa->nama ## ";
6     echo "<hr>";
7
8     // pakai perulangan untuk menampilkan setiap object matakuliah
9     foreach ($mahasiswa->matakuliahs as $matakuliah)
10    {
11        echo "$matakuliah->id | $matakuliah->kode:
12                      $matakuliah->nama ($matakuliah->jumlah_sks sks) <br>";
13    }
14 }
```

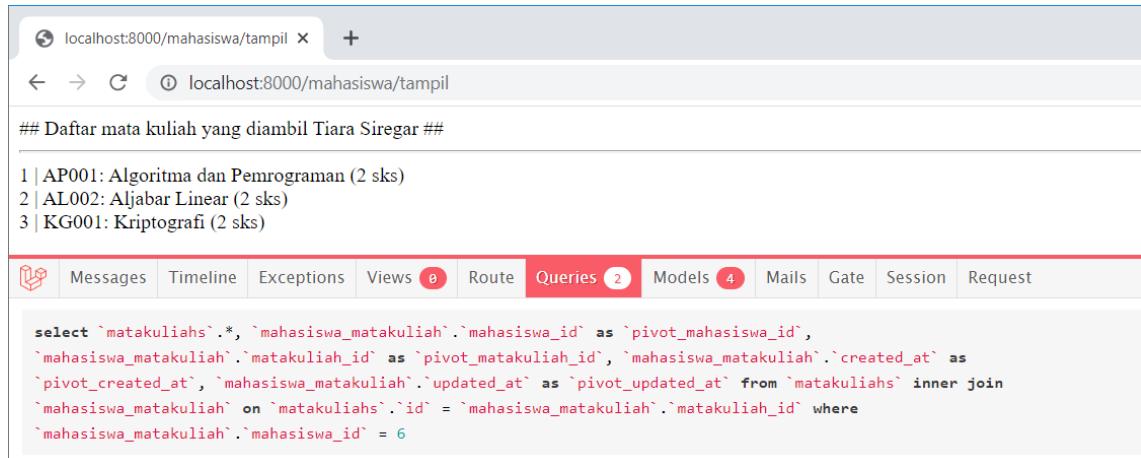
Eloquent Relationship: Many to Many



Gambar: Semua mata kuliah yang diambil oleh Tiara Siregar

Dengan perulangan foreach di baris 9 – 13, saya menampilkan satu per satu `id`, `kode`, `nama` dan `jumlah_sks` setiap object `Matakuliah`. Inilah cara menampilkan data dalam *relationship many to many*.

Jika anda iseng membuka Laravel debugbar, akan terlihat query yang cukup rumit. Dengan eloquent relationship, kita tidak perlu berurusan dengan query-query ini.



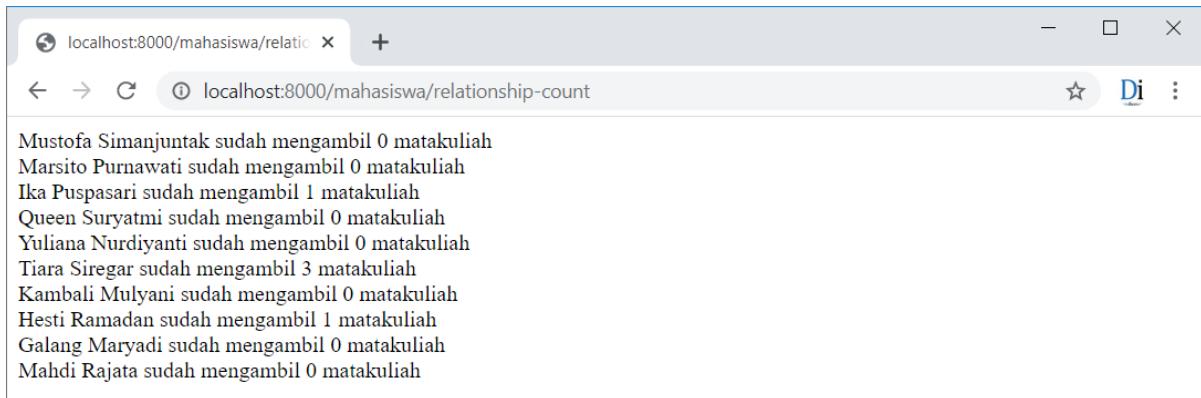
Gambar: Query yang dijalankan oleh eloquent relationship

Menghitung Data dengan `withCount()`

Dalam beberapa situasi, kita ingin mencari info berapa jumlah matakuliah yang diambil oleh setiap mahasiswa. Untuk keperluan ini, bisa menggunakan method `withCount()`:

app\Http\Controllers\MahasiswaController.php

```
1 public function relationshipCount()
2 {
3     $mahasiswas = Mahasiswa::withCount('matakuliah')->get();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nama sudah mengambil
7             $mahasiswa->matakuliah_count matakuliah <br> ";
8     }
9 }
```



Gambar: Mahasiswa dengan tambahan method withCount()

Dengan tambahan method `withCount()` di baris 3, maka dalam setiap object Mahasiswa terdapat property `matakuliah_count` yang bisa diakses. Property ini berisi informasi jumlah id `matakuliah` yang terhubung ke mahasiswa tersebut.

Penjelasan lebih lengkap tentang method `withCount()` dan juga `loadCount()` sudah kita bahas di bab sebelumnya.

Menghapus Data Relationship

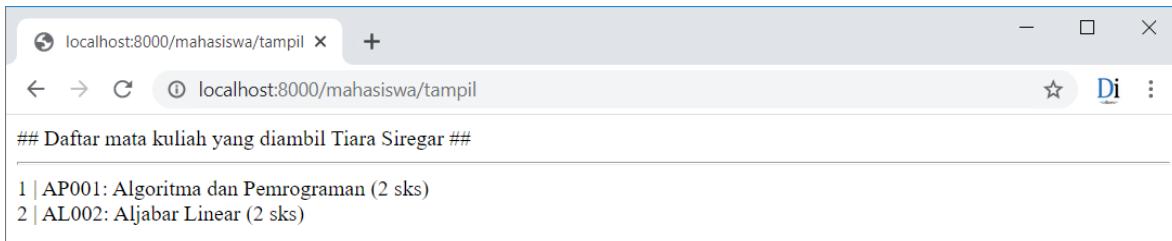
Kita sudah lihat cara penggunaan method `attach()` untuk menghubungkan tabel `mahasiswa` dengan `matakuliah`. Sekarang saatnya membahas method `detach()` yang bisa dipakai untuk memutus hubungan tersebut:

app\Http\Controllers\MahasiswaController.php

```
1 public function detach()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4     $matakuliah = Matakuliah::where('nama', "Kriptografi")->first();
5
6     $mahasiswa->matakuliah()->detach($matakuliah);
7
8     echo "Proses detach berhasil";
9 }
```

Cara penggunaan method `detach()` sama seperti method `attach()`. Dalam kode ini saya mencoba menghapus mata kuliah Kriptografi yang dipilih oleh Tiara Siregar.

Jalankan method di atas dengan mengakses URL `localhost:8000/mahasiswa/detach`, lalu akses `localhost:8000/mahasiswa/tampil` untuk memeriksa daftar mata kuliah yang diambil Tiara Siregar. Sekarang hanya tinggal 2 buah mata kuliah:



The screenshot shows a browser window with the URL `localhost:8000/mahasiswa/tampil`. The page displays a list of course names and their credits: "AP001: Algoritma dan Pemrograman (2 sks)" and "AL002: Aljabar Linear (2 sks)".

```
## Daftar mata kuliah yang diambil Tiara Siregar ##  
1 | AP001: Algoritma dan Pemrograman (2 sks)  
2 | AL002: Aljabar Linear (2 sks)
```

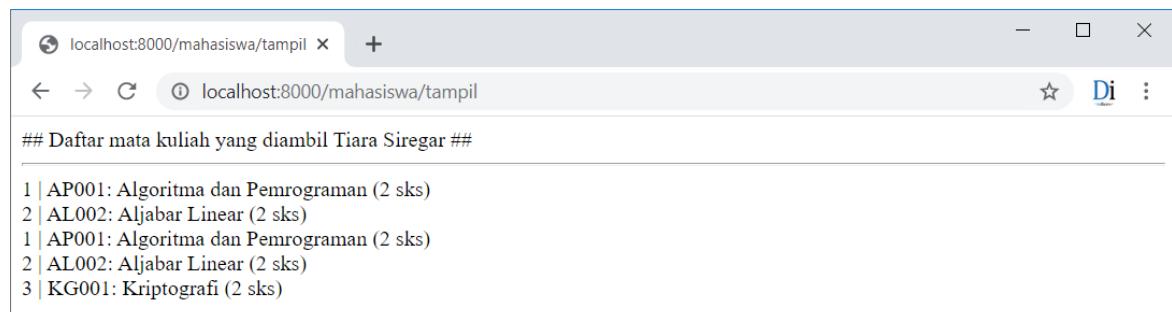
Gambar: Matakuliah Kriptografi sudah tidak ada lagi

Pada dasarnya method `detach()` akan menghapus isi tabel `mahasiswa_matakuliah`. Ketika menjalankan kode di atas, satu baris data akan berkurang dari tabel `mahasiswa_matakuliah`. Method `detach()` juga bisa diisi collection untuk menghapus banyak *relationship* sekaligus.

Menginput Data Relationship dengan sync()

Salah satu kekurangan (atau juga bisa disebut dengan kelebihan) dari penggunaan method `attach()` adalah, terdapat kemungkinan data ganda.

Sebagai contoh, silahkan akses URL `localhost:8000/mahasiswa/attach-array` sekali lagi, maka jurusan yang diambil oleh Tiara Siregar juga ikut berulang:



The screenshot shows a browser window with the URL `localhost:8000/mahasiswa/tampil`. The page displays a list of course names and their credits, including some duplicates: "AP001: Algoritma dan Pemrograman (2 sks)", "AL002: Aljabar Linear (2 sks)", "1 | AP001: Algoritma dan Pemrograman (2 sks)", "2 | AL002: Aljabar Linear (2 sks)", and "3 | KG001: Kriptografi (2 sks)".

```
## Daftar mata kuliah yang diambil Tiara Siregar ##  
1 | AP001: Algoritma dan Pemrograman (2 sks)  
2 | AL002: Aljabar Linear (2 sks)  
1 | AP001: Algoritma dan Pemrograman (2 sks)  
2 | AL002: Aljabar Linear (2 sks)  
3 | KG001: Kriptografi (2 sks)
```

Gambar: Jurusan yang dipilih bisa berulang

Salah satu solusi untuk mencegah hal ini adalah dengan menambah *unique key* ke kolom `mahasiswa_id` dan `matakuliah_id` di tabel `mahasiswa_matakuliah`. Dengan demikian jika eloquent mencoba mengisi data ganda, akan tampil pesan error dari MySQL.

Solusi lain adalah mengganti penggunaan method `attach()` dengan method `sync()`. Method `sync()` juga dipakai untuk menginput hubungan *relationship*, tapi setiap kali method ini dijalankan, hubungan yang sudah ada akan dihapus terlebih dahulu.

Berikut contoh praktek dari method `sync()`:

`app\Http\Controllers\MahasiswaController.php`

```
1 public function sync()  
2 {  
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();  
4     $matakuliah = Matakuliah::find([4,5]);  
5 }
```

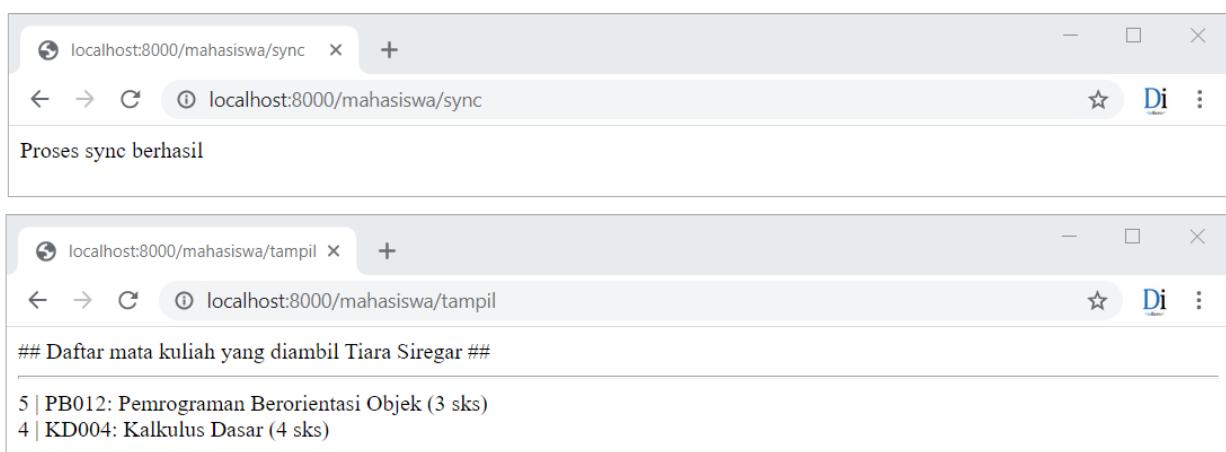
Eloquent Relationship: Many to Many

```
6     $mahasiswa->matakuliahs()->sync($matakuliahs);
7
8     echo "Proses sync berhasil";
9 }
```

Cara penggunaan method `sync()` mirip seperti method `attach()`, termasuk bisa menerima argument dalam bentuk collection atau satu object saja.

Dalam kode program di atas saya menghubungkan matakuliah id 4 dan id 5 ke Tiara Siregar. Sebelum dijalankan, perhatikan bahwa saat ini Tiara Siregar sudah memilih 5 jurusan, yang 2 diantaranya terdapat pengulangan.

Silahkan akses URL `localhost:8000/mahasiswa.sync` untuk menjalankan kode di atas, lalu cek kembali daftar matakuliah Tiara Siregar:



Gambar: Menjalankan method sync()

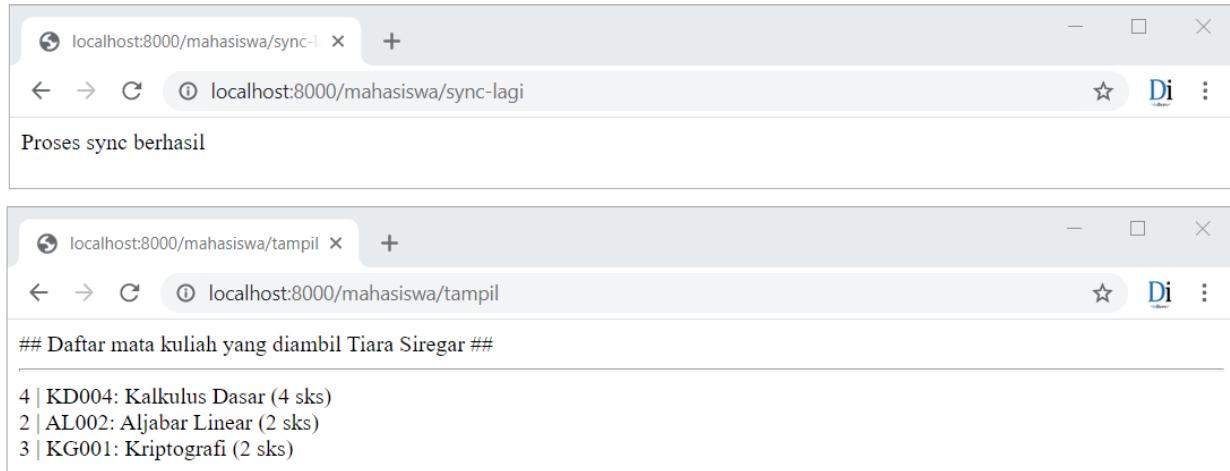
Hasilnya, mata kuliah yang terdaftar hanya ada 2, sesuai dengan yang diinput sebagai argument method `sync()`. Matakuliah yang terdaftar sebelumnya akan dihapus.

Kemudian, saya akan coba ganti lagi daftar mata kuliah Tiara Siregar:

app\Http\Controllers\MahasiswaController.php

```
1 public function syncLagi()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4     $matakuliahs = Matakuliah::find([2,3,4]);
5
6     $mahasiswa->matakuliahs()->sync($matakuliahs);
7     echo "Proses sync berhasil";
8 }
```

Eloquent Relationship: Many to Many



Gambar: Menjalankan method sync()

Terlihat yang sekarang terdaftar hanyalah mata kuliah dengan id 2, 3 dan 4, yakni sesuai data yang diinput ke dalam method sync().

Untuk menyingkat kode program, method sync() juga bisa di-chaining seperti contoh berikut:

app\Http\Controllers\MahasiswaController.php

```
1 public function syncChaining()
2 {
3     Mahasiswa::where('nama', 'Tiara Siregar')->first()->matakuliah()
4         ->sync(Matakuliah::find([1,5]));
5
6     echo "Proses sync berhasil";
7 }
```



Gambar: Menjalankan method sync() yang di-chaining

Meskipun lebih singkat, namun kadang kode seperti ini sedikit susah dipahami.

Nama method sync() merupakan singkatan dari syncronous, yang bisa diartikan sebagai proses sikronisasi (menyamakan kondisi data).

Menginput Data Relationship dengan syncWithoutDetaching()

Nama method ini memang cukup panjang, tapi secara tidak langsung menjelaskan maksud penggunaannya. Method `syncWithoutDetaching()` dipakai untuk proses `sync()`, namun tanpa menghapus data yang sudah ada.

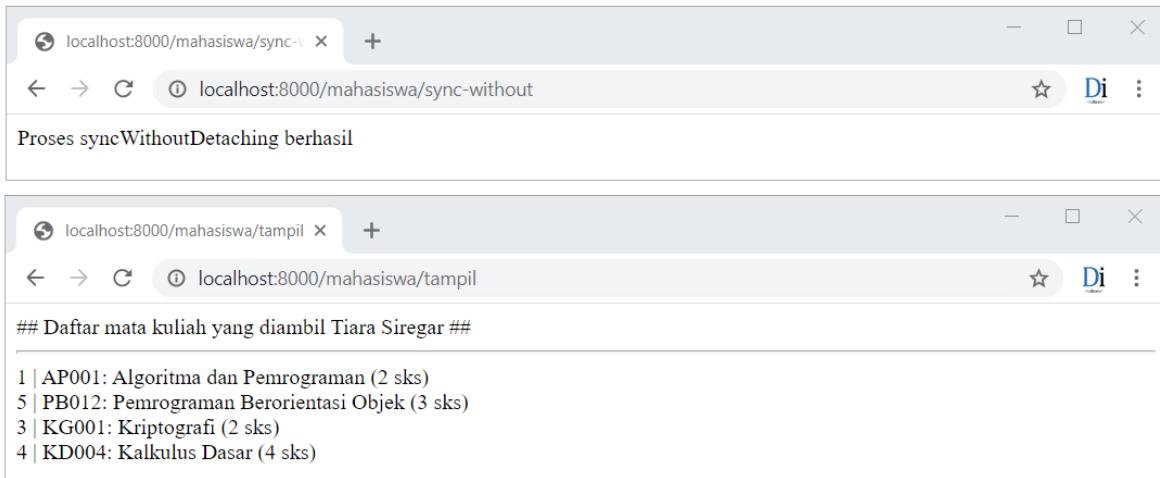
Sebagaimana yang sudah di praktekkan, setiap kali method `sync()` dijalankan, itu akan menghapus data relationship yang sudah ada sebelumnya.

Jika kita ingin menambah 1 data relationship namun tidak ingin menghapus hubungan yang sudah ada, bisa menggunakan method `syncWithoutDetaching()`. Berikut contoh penggunaannya:

app\Http\Controllers\MahasiswaController.php

```
1 public function syncWithout()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4     $matakuliahs = Matakuliah::find([3,4]);
5
6     $mahasiswa->matakuliahs()->syncWithoutDetaching($matakuliahs);
7
8     echo "Proses syncWithoutDetaching berhasil";
9 }
```

Cara penggunaan method `syncWithoutDetaching()` sama seperti method `sync()`. Dalam kode di atas saya ingin menghubungkan mata kuliah dengan id 3 dan id 4 ke Tiara Siregar. Berikut hasilnya:



Gambar: Menjalankan method syncWithoutDetaching()

Pada saat diakses, mata kuliah yang baru akan ditambah, tapi itu tidak menghapus mata kuliah yang sudah diambil.

Tapi kalau seperti ini apa bedanya method `syncWithoutDetaching()` dengan method `attach()`? Silahkan coba akses URL `localhost:8000/mahasiswa/sync-without` beberapa kali,

hasilnya, mata kuliah id 3 dan id 4 tidak akan bertambah sebagaimana jika menggunakan method `attach()`.

Menginput dan Menghapus Data Relationship dengan `toggle()`

Masih membahas cara menginput data relationship, Eloquent juga menyediakan method `toggle()`. Method ini cukup unik karena akan menginput dan menghapus relationship tergantung data yang ada saat ini.

Jika belum ada hubungan antara mahasiswa dengan mata kuliah, method `toggle()` akan menginput hubungan tersebut. Namun jika sudah ada, maka method `toggle()` akan menghapusnya. Berikut contoh praktik dari method `toggle()`:

app\Http\Controllers\MahasiswaController.php

```
1 public function toggle()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4     $matakuliah = Matakuliah::where('nama', 'Kriptografi')->get();
5
6     $mahasiswa->matakuliah()->toggle($matakuliah);
7
8     echo "Proses toggle berhasil";
9 }
```

Untuk menguji hasil dari kode di atas, silahkan akses URL `localhost:8000/mahasiswa/toggle`, lalu lihat daftar mata kuliah yang diambil Tiara Siregar. Kemudian akses kembali, dan lihat lagi. Secara bergantian, mata kuliah Kriptografi akan muncul dan menghilang.

Menghapus Data Relationship

Isi tabel `mahasiswa_matakuliah` bisa dihapus secara manual, dan itu tidak masalah karena tidak ada pembatasan di sini. Ketika data dihapus, artinya kita memutus hubungan antara tabel `mahasiswas` dengan tabel `matakuliah`s.

Namun jika yang dihapus adalah data di tabel `mahasiswas` atau tabel `matakuliah`s, itu akan menyebabkan efek `ON DELETE CASCADE`.

Jika anda lihat kembali file migration tabel `mahasiswa_matakuliah`, saya menambah method `onDelete('cascade')` ke dalam kolom *foreign key* `mahasiswa_id` dan `matakuliah_id`. Ini akan menyebabkan jika data di tabel `mahasiswas` atau tabel `matakuliah`s dihapus, isi tabel `mahasiswa_matakuliah` juga akan dihapus.

Dalam kode program berikut saya akan menghapus mahasiswa Tiara Siregar dari tabel `mahasiswas`:

Eloquent Relationship: Many to Many

app\Http\Controllers\MahasiswaController.php

```
1 public function delete()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();
4     $mahasiswa->delete();
5     echo "Data $mahasiswa->nama berhasil dihapus";
6 }
```

Sebelum di jalankan, pada tabel `mahasiswa_matakuliah` saat ini terdapat 3 data mata kuliah yang diambil oleh Tiara Siregar. Setelah Tiara Siregar dihapus dari tabel `mahasiswas`, ketiga data ini juga akan ikut terhapus.

The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". It contains two SQL queries:

```
MariaDB [laravel]> SELECT * FROM mahasiswa_matakuliah;
+----+-----+-----+-----+-----+
| id | mahasiswa_id | matakuliah_id | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1  | 3          | 4          | 2020-08-10 01:01:51 | 2020-08-10 01:01:51 |
| 5  | 8          | 5          | 2020-08-10 03:37:14 | 2020-08-10 03:37:14 |
| 13 | 6          | 1          | 2020-08-10 13:27:39 | 2020-08-10 13:27:39 |
| 14 | 6          | 5          | 2020-08-10 13:28:05 | 2020-08-10 13:28:05 |
| 16 | 6          | 4          | 2020-08-11 03:18:57 | 2020-08-11 03:18:57 |
+----+-----+-----+-----+-----+
5 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM mahasiswa_matakuliah;
+----+-----+-----+-----+-----+
| id | mahasiswa_id | matakuliah_id | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1  | 3          | 4          | 2020-08-10 01:01:51 | 2020-08-10 01:01:51 |
| 5  | 8          | 5          | 2020-08-10 03:37:14 | 2020-08-10 03:37:14 |
+----+-----+-----+-----+-----+
2 rows in set (0.002 sec)
```

Gambar: Isi tabel mahasiswa_matakuliah sebelum dan setelah Tiara Siregar dihapus

Jika kita tidak menambah method `onDelete('cascade')` ke dalam definisi kolom `mahasiswa_id` dan `matakuliah_id`, maka efek penghapusan otomatis tidak akan terjadi. Dari segi kode program memang tidak akan error, tapi dari segi design database ini bukanlah hal yang baik.

14.4. Eloquent Relationship belongsToMany() (lagi)

Sebagaimana biasa, kita akan pelajari relationship dari dua sisi. Sebelumnya sudah dibahas hubungan tabel `mahasiswas` ke tabel `matakuliahs`, sekarang saatnya membahas hubungan dari tabel `matakuliahs` ke tabel `mahasiswas`.

Yang cukup unik, dalam *relationship many to many* hubungan yang terjadi antara tabel `matakuliahs` dengan tabel `mahasiswas` juga **belongs to many**, yang artinya kita tetap memakai method `belongsToMany()` di dalam model Matakuliah.

Untuk membuat hubungan `matakuliah belongs to many mahasiswa`, silahkan buka file model `Matakuliah.php` lalu tambah satu method berikut:

Eloquent Relationship: Many to Many

app\Models\Matakuliah.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 Matakuliah extends Model
9 {
10     use HasFactory;
11     public function mahasiswa()
12     {
13         return $this->belongsToMany('App\Models\Mahasiswa')->withTimestamps();
14     }
15 }
```

Tambahan kode ada di baris 11 – 14, yakni sebuah method bernama `mahasiswa()` yang berisi perintah `return $this->belongsToMany('App\Models\Mahasiswa')->withTimestamps();`. Inilah cara kita memberitahu Laravel jenis hubungan dari tabel `matakuliah`s ke tabel `masiswa`, yaitu matakuliah *belongs to many* mahasiswa.

Nama method `mahasiswa()` menggunakan kata *plural* dengan tambahan 's' karena hubungan yang terjadi adalah *many to many*. Tambahan method `withTimestamps()` dipakai agar kolom `created_at` dan `updated_at` di tabel `masiswa_matakuliah` langsung terisi ketika relationship dibuat.

Berikut daftar route yang akan di bahas:

routes\web.php

```
1 <?php
2 ...
3
4 use App\Http\Controllers\MatakuliahController;
5
6 Route::prefix('/matakuliah')->group(function () {
7     Route::get('/all', [MatakuliahController::class, 'all']);
8     Route::get('/attach', [MatakuliahController::class, 'attach']);
9     Route::get('/attach-where', [MatakuliahController::class, 'attachWhere']);
10    Route::get('/tampil', [MatakuliahController::class, 'tampil']);
11    Route::get('/detach', [MatakuliahController::class, 'detach']);
12    Route::get('/sync', [MatakuliahController::class, 'sync']);
13    Route::get('/pivot', [MatakuliahController::class, 'pivot']);
14});
```

Sebagai besar materi ini sebenarnya sudah kita pelajari, hanya saja sekarang berangkat dari tabel `matakuliah`s. Silahkan generate ulang tabel dan seeder dengan perintah `php artisan migrate:fresh --seed`.

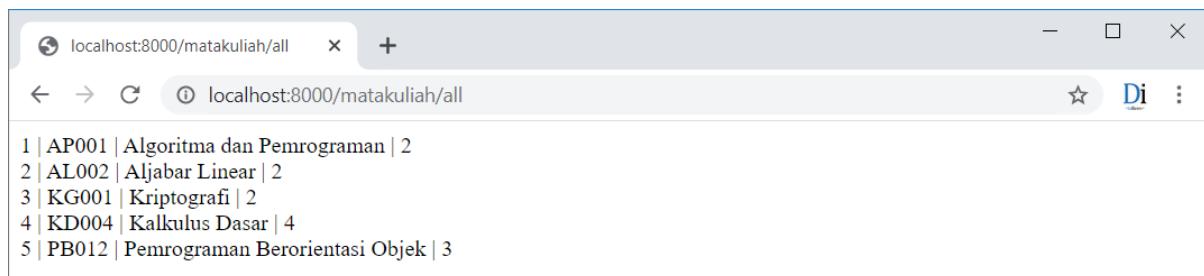
Eloquent Relationship: Many to Many

Menampilkan Semua Data Matakuliah

Route pertama di siapkan untuk testing menampilkan semua data dari tabel `matakuliah`:

app\Http\Controllers\MatakuliahController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use App\Models\Mahasiswa;
5 use App\Models\Matakuliah;
6
7 class MatakuliahController extends Controller
8 {
9     public function all()
10    {
11        $matakuliah = Matakuliah::all();
12        foreach ($matakuliah as $matakuliah) {
13            echo "$matakuliah->id | $matakuliah->kode |
14                $matakuliah->nama | $matakuliah->jumlah_sks <br>";
15        }
16    }
17 }
```



Gambar: Menampilkan isi tabel matakuliah

Ini merupakan kode eloquent biasa tanpa menggunakan relationship.

Menginput Data Relationship

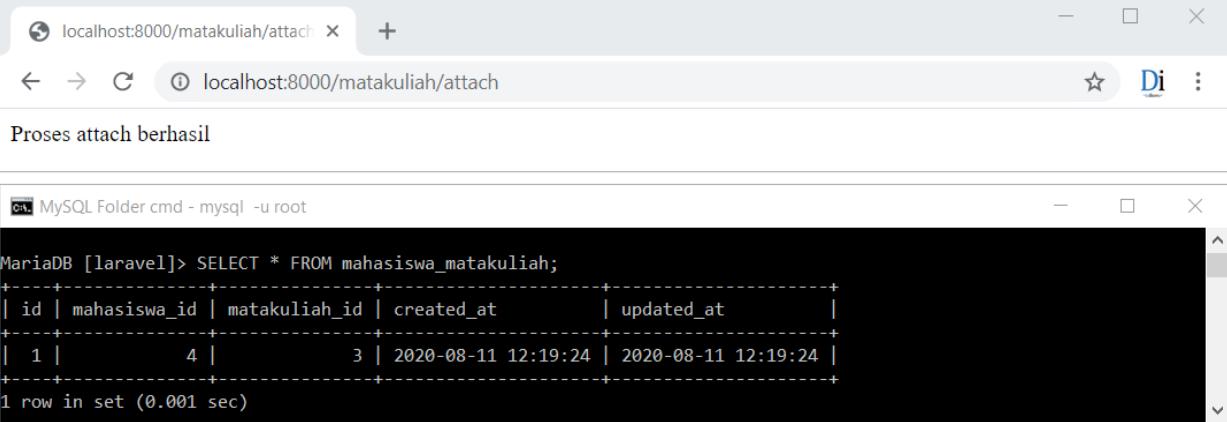
Untuk menginput data relationship dari tabel `matakuliah`, caranya juga sama seperti dari tabel `mahasiswa`. Kita bisa menggunakan method `attach()`, `sync()` maupun `toggle()`.

Berikut contoh dari method `attach()`:

app\Http\Controllers\MatakuliahController.php

```
1 public function attach()
2 {
3     $matakuliah = Matakuliah::find(3);
4     $mahasiswa = Mahasiswa::find(4);
5
6     $matakuliah->mahasiswa()->attach($mahasiswa);
7     echo "Proses attach berhasil";
8 }
```

Eloquent Relationship: Many to Many



The screenshot shows a browser window with the URL `localhost:8000/matakuliah/attach` and a MySQL command-line interface window.

Browser Output:

```
localhost:8000/matakuliah/attach
Proses attach berhasil
```

MySQL Terminal Output:

```
MariaDB [laravel]> SELECT * FROM mahasiswa_matakuliah;
+----+-----+-----+-----+
| id | mahasiswa_id | matakuliah_id | created_at           | updated_at          |
+----+-----+-----+-----+
| 1  |        4    |      3    | 2020-08-11 12:19:24 | 2020-08-11 12:19:24 |
+----+-----+-----+-----+
1 row in set (0.001 sec)
```

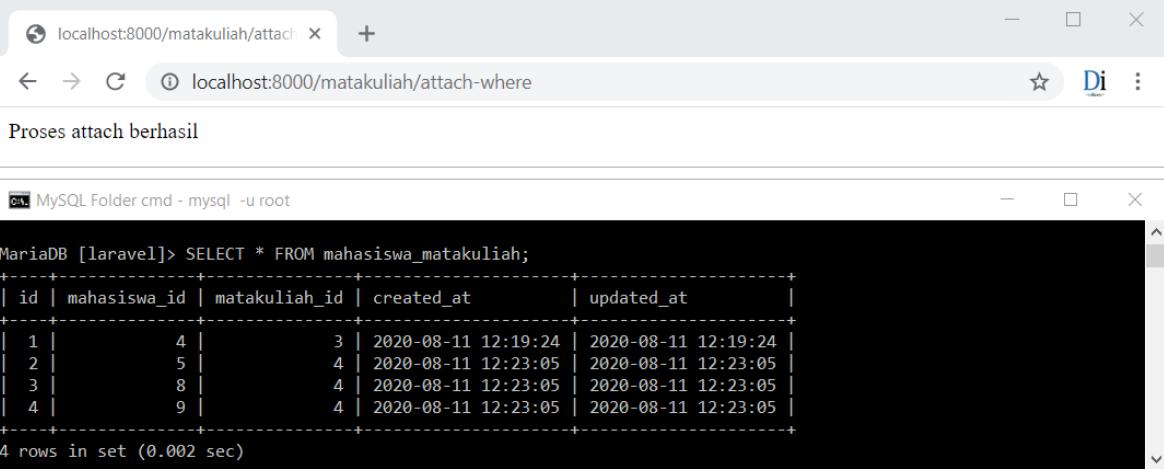
Gambar: Menggunakan method attach() dari tabel matakuliah

Di sini saya mencoba menghubungkan mata kuliah id 3 dengan mahasiswa yang memiliki id 4.

Berikut contoh lain untuk menghubungkan banyak mahasiswa sekaligus:

app\Http\Controllers\MatakuliahController.php

```
1 public function attachWhere()
2 {
3     $matakuliah = Matakuliah::where('nama', 'Kalkulus Dasar')->first();
4     $mahasiswas = Mahasiswa::where('jurusan', 'Teknik Informatika')->get();
5
6     $matakuliah->mahasiswas()->attach($mahasiswas);
7     echo "Proses attach berhasil";
8 }
```



The screenshot shows a browser window with the URL `localhost:8000/matakuliah/attach` and a MySQL command-line interface window.

Browser Output:

```
localhost:8000/matakuliah/attach
Proses attach berhasil
```

MySQL Terminal Output:

```
MariaDB [laravel]> SELECT * FROM mahasiswa_matakuliah;
+----+-----+-----+-----+
| id | mahasiswa_id | matakuliah_id | created_at           | updated_at          |
+----+-----+-----+-----+
| 1  |        4    |      3    | 2020-08-11 12:19:24 | 2020-08-11 12:19:24 |
| 2  |        5    |      3    | 2020-08-11 12:23:05 | 2020-08-11 12:23:05 |
| 3  |        8    |      3    | 2020-08-11 12:23:05 | 2020-08-11 12:23:05 |
| 4  |        9    |      3    | 2020-08-11 12:23:05 | 2020-08-11 12:23:05 |
+----+-----+-----+-----+
4 rows in set (0.002 sec)
```

Gambar: Menggunakan method attach() untuk banyak mahasiswa

Kali ini saya mendaftarkan mata kuliah Kalkulus Dasar ke semua mahasiswa yang mengambil jurusan Teknik Informatika.

Jika dilihat ke dalam tabel `mahasiswa_matakuliah`, akan muncul tambahan 3 data. Artinya, di tabel `mahasiswas` terdapat 3 mahasiswa yang memilih jurusan Teknik Informatika.

Menampilkan Data Relationship

Setelah tabel `mahasiswa_matakuliah` berisi beberapa data, berikut contoh menampilkan daftar mahasiswa yang sudah terdaftar di mata kuliah Kalkulus Dasar:

app\Http\Controllers\MatakuliahController.php

```
1 public function tampil()
2 {
3     $matakuliah = Matakuliah::where('nama', 'Kalkulus Dasar')->first();
4
5     echo "## Daftar mahasiswa yang mengambil mata kuliah $matakuliah->nama ## ";
6     echo "<hr>";
7
8     foreach ($matakuliah->mahasiswas as $mahasiswa)
9     {
10         echo "$mahasiswa->nim: $mahasiswa->nama ($mahasiswa->jurusan) <br>";
11     }
12 }
```



Gambar: Menampilkan semua mahasiswa yang mengambil mata kuliah Kalkulus Dasar

Di baris 3 saya mengambil 1 object mata kuliah Kalkulus Dasar. Menggunakan eloquent relationship, property `$matakuliah->mahasiswas` akan berisi collection dari semua mahasiswa yang terhubung ke jurusan ini.

Menghapus Data Relationship

Untuk menghapus relationship, kita juga bisa menggunakan method `detach()` seperti contoh berikut:

app\Http\Controllers\MatakuliahController.php

```
1 public function detach()
2 {
3     $matakuliah = Matakuliah::where('nama', 'Kalkulus Dasar')->first();
4     $mahasiswa = Mahasiswa::where('nama', 'Galang Maryadi')->first();
5
6     $matakuliah->mahasiswas()->detach($mahasiswa);
7     echo "Proses detach berhasil";
8 }
```

Eloquent Relationship: Many to Many



Gambar: Menghapus relationship dengan method detach()

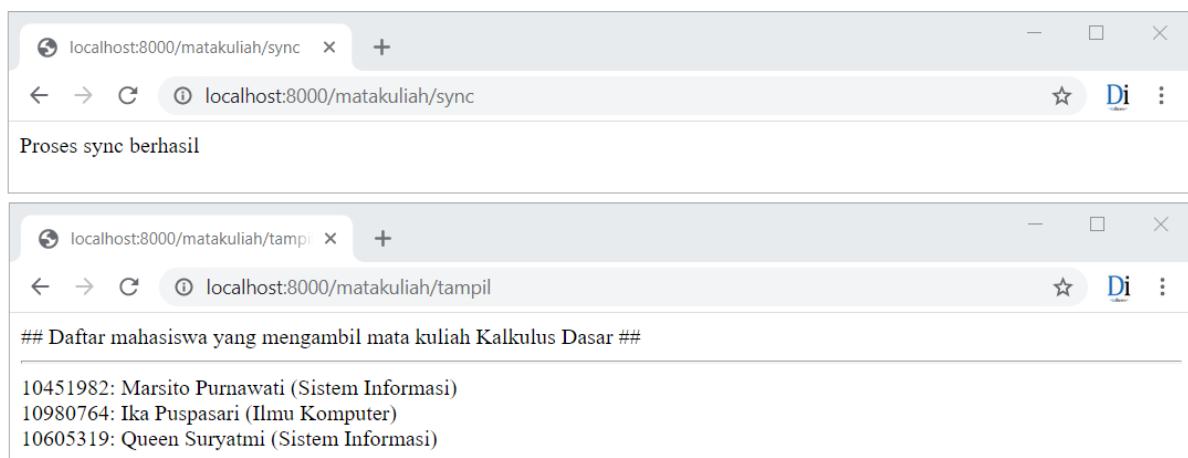
Dalam kode program ini saya menghapus Galang Maryadi dari daftar mahasiswa yang mengambil mata kuliah Kalkulus Dasar.

Menginput Data Relationship dengan sync()

Cara penggunaan method sync() juga sama seperti materi di tabel `mahasiswas`. Berikut contoh penggunaannya jika berangkat dari tabel `matakuliah`:

app\Http\Controllers\MatakuliahController.php

```
1 public function sync()
2 {
3     Matakuliah::where('nama', 'Kalkulus Dasar')->first()->mahasiswas()
4     ->sync(Mahasiswa::find([2,3,4]));
5
6     echo "Proses sync berhasil";
7 }
```



Gambar: Menghubungkan relationship dengan method sync()

Di sini saya mendaftarkan ulang mahasiswa yang memilih mata kuliah Kalkulus Dasar. Karena menggunakan `sync(Mahasiswa::find([2,3,4]))`, maka hanya mahasiswa dengan id 2, 3 dan 4

saja yang terdaftar, data mahasiswa yang sudah terdaftar sebelumnya otomatis akan dihapus.

Cara penggunaan method `syncWithoutDetaching()` dan `toggle()` juga sama seperti contoh tabel `mahasiswas` sehingga tidak perlu kita bahas lagi.

14.5. Mengakses Tabel Pivot

Tabel `mahasiswa_matakuliah` yang bertindak sebagai tabel pivot umumnya tidak diakses secara langsung, kecuali jika kita ingin mencari informasi tentang tanggal `timestamp`, yakni kolom `update_at` dan `created_at` yang ada di tabel tersebut.

Kolom di tabel pivot bisa diakses menggunakan property `pivot` dari setiap object relationship. Berikut contoh penggunaannya:

app\Http\Controllers\MatakuliahController.php

```
1 public function pivot()
2 {
3     $matakuliah = Matakuliah::where('nama', 'Kalkulus Dasar')->first();
4
5     echo "## Daftar mahasiswa yang mengambil mata kuliah $matakuliah->nama ## ";
6     echo "<br>";
7     foreach ($matakuliah->mahasiswas as $mahasiswa)
8     {
9         echo "$mahasiswa->nama ($mahasiswa->jurusan),
10             mengambil mata kuliah pada
11             {$mahasiswa->pivot->created_at->isoFormat('D MMMM Y')} <br>";
12     }
13 }
```



Gambar: Mengakses kolom `created_at` dari tabel pivot

Kode program ini mirip seperti contoh tentang cara menampilkan data relationship, dimana saya menampilkan semua daftar mahasiswa yang mengambil mata kuliah Kalkulus Dasar.

Namun perhatikan tambahan perintah `echo` di baris 11, kode `$mahasiswa->pivot->created_at` dipakai untuk mengakses kolom `created_at` yang ada di tabel `mahasiswa_matakuliah`. Hasil tanggal ini kemudian dilewatkan ke dalam method `isoFormat()` bawaan Carbon.

Jika kita ingin menampilkan kolom lain milik tabel pivot seperti `updated_at`, bisa juga diakses dari `$mahasiswa->pivot->created_at`. Inilah cara untuk mengakses kolom tambahan di tabel

pivot.

14.6. Membuat Composite Unique Key

Ketika membahas tentang masalah data berulang karena penggunaan method `attach()`, saya pernah menyinggung bahwa salah satu solusi adalah dengan mendefinisikan kolom `mahasiswa_id` dan `matakuliah_id` sebagai *unique key*. Ini supaya jika terjadi data berulang, akan tampil pesan error dari MySQL.

Unique key yang dimaksud lebih tepat disebut sebagai **composite unique key**, karena terdiri dari gabungan 2 kolom: `mahasiswa_id` dan `matakuliah_id`. Mari kita coba praktekkan.

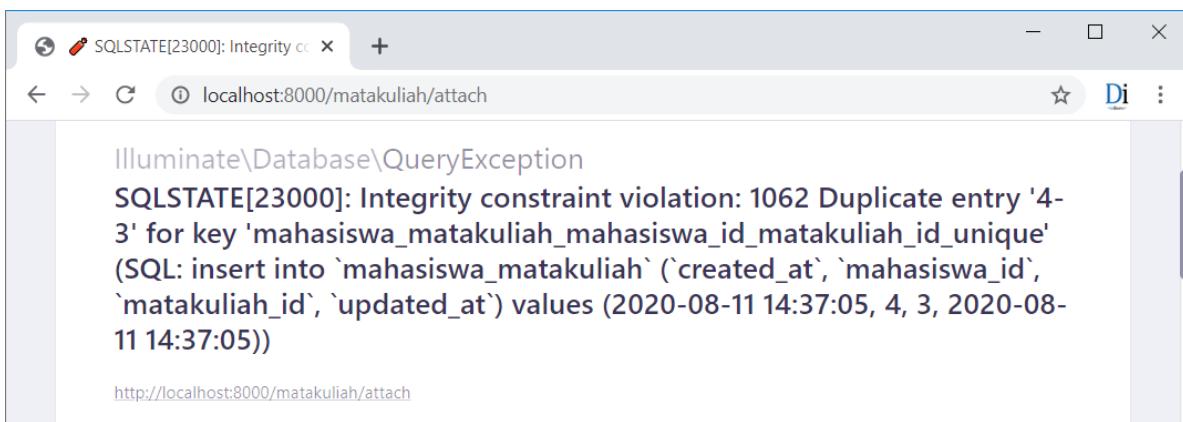
Silahkan buka file migration tabel `mahasiswa_matakuliah` lalu modifikasi sebagai berikut:

database\migrations\<timestamp>_create_mahasiswa_matakuliah_table.php

```
1 public function up()
2 {
3     Schema::create('mahasiswa_matakuliah', function (Blueprint $table) {
4         $table->id();
5         $table->foreignId('mahasiswa_id')->constrained()->onDelete('cascade');
6         $table->foreignId('matakuliah_id')->constrained()->onDelete('cascade');
7         $table->timestamps();
8
9         $table->unique(['mahasiswa_id', 'matakuliah_id']);
10    });
11 }
```

Tambahannya ada di baris 9. Inilah cara membuat composite unique key di Laravel migration.

Generate ulang tabel dan juga seeder dengan perintah `php artisan migrate:fresh --seed`. Setelah itu akses URL `http://localhost:8000/matakuliah/attach` yang berisi pemanggilan method `attach()`. Pada saat dijalankan pertama kali tidak ada masalah, tapi saat diakses untuk kedua kali, akan tampil pesan error berikut:



Gambar: Error karena `attach()` berulang

Pesan error ini berisi keterangan bahwa MySQL melarang kita untuk menginput data ganda ke tabel `mahasiswa_matakuliah`.

Untuk kebanyakan kasus, penggunaan *composite unique key* ini lebih disarankan agar terjadi konsistensi data. Meskipun nantinya kita tidak menggunakan method `attach()`, proteksi data di level tabel seperti ini akan sangat membantu untuk mencegah data berulang.

Jika pada saat testing terjadi error di atas, kita bisa simpulkan ada logika yang salah. Ini jauh lebih baik daripada menemukan bug saat aplikasi sudah rilis nanti.

Akhirnya, kita telah selesai membahas *relationship many to many*. Relationship yang satu ini memerlukan bantuan tabel pivot untuk menghubungkan kedua tabel. Relationship many to many juga cukup sering dipakai.

Berikutnya, kita akan masuk ke **relationship has one through**.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

15. Eloquent Relationship: Has One Through

Relationship has one through merupakan variasi dari relationship one to one. Dalam bab ini kita akan bahas pengertian dan cara penerapannya di Laravel.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel101
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

15.1. Pengertian Has One Through Relationship

Relationship *has one through* adalah relasi atau hubungan *one to one* antara tabel utama dengan tabel ketiga dengan perantara tabel kedua. Jadi dalam prakteknya nanti, relasi *has one through* butuh 3 tabel.

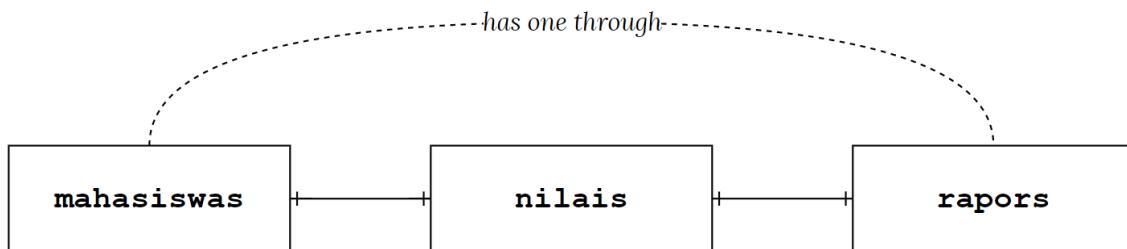
Sebagai contoh, di dalam praktek *relationship one to one* kita sudah memiliki tabel `mahasiswa` dan tabel `nilais`. Hubungan yang terjadi adalah satu mahasiswa terhubung ke satu data nilai, begitu juga sebaliknya satu data nilai terhubung ke satu mahasiswa.

Sekarang saya akan tambah tabel ketiga, yakni tabel `rapors`. Tabel `rapors` berisi kesimpulan nilai mahasiswa. Misal jika di semester 3 si mahasiswa mendapat nilai yang kurang memuaskan, isi rapornya adalah 'Nilai semester 3 sangat turun, harus rajin lagi!'.

Hubungan antara tabel `nilais` dengan tabel `rapors` juga *one to one* karena rapor secara khusus mengomentari satu data nilai. Namun bagaimana hubungan antara tabel `mahasiswa` dengan tabel `rapors`?

Meskipun tabel `mahasiswa` tidak memiliki relationship langsung dengan tabel `rapors`, akan tetapi hubungan itu bisa dilakukan melalui tabel `nilais`. Hubungan seperti inilah yang disebut sebagai **has one through** relationship.

Diagram berikut menggambarkan *relationship* yang terjadi:

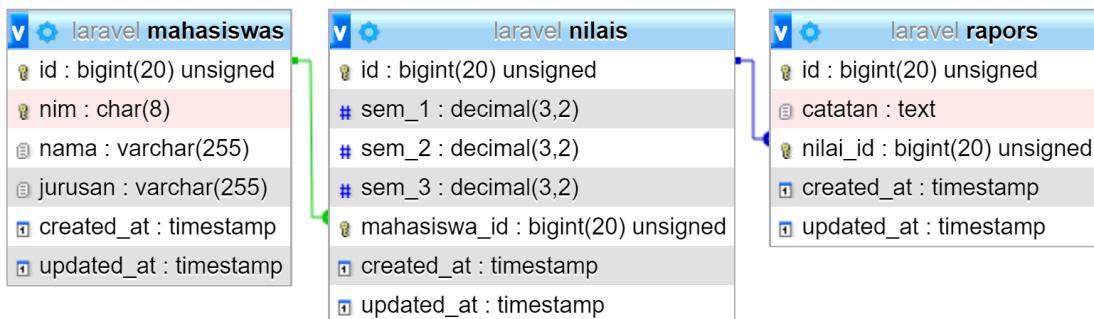


Gambar: Relationship has one through antara tabel mahasiswa dengan tabel rapor

15.2. Persiapan Awal

Sebagai bahan praktik untuk penerapan *has one through*, kita akan buat 3 buah tabel: **mahasiswa**, **nilais** dan **rapors**. Berikut struktur ketiga tabel tersebut:

- ◆ Tabel **mahasiswa**: `id`, `nim`, `nama`, `jurusan`, `created_at` dan `updated_at`.
- ◆ Tabel **nilais**: `id`, `sem_1`, `sem_2`, `sem_3`, `mahasiswa_id`, `created_at` dan `updated_at`.
- ◆ Tabel **rapors**: `id`, `catatan`, `nilai_id`, `created_at` dan `updated_at`.



Gambar: Hubungan antara tabel mahasiswa, matakuliah, serta mahasiswa_matakuliah

Terlihat garis antara kolom `id` di tabel **mahasiswa** dengan kolom `mahasiswa_id` di tabel **nilais**. Serta garis antara kolom `id` di tabel **nilais** dengan kolom `nilai_id` di tabel **rapor**. Kedua garis ini mencerminkan hubungan *one to one*.

Hubungan relationship *has one through* antara tabel **mahasiswa** dengan tabel **rapor** akan kita definisikan di dalam model Laravel.

Generate Data Sample

Kita akan buat ketiga tabel melalui migration. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```

php artisan make:model Mahasiswa -m
php artisan make:model Nilai -m
php artisan make:model Rapor -m
    
```

Kode di atas akan membuat file **model** dan **migration** untuk tabel **mahasiswa**, **nilais**, dan

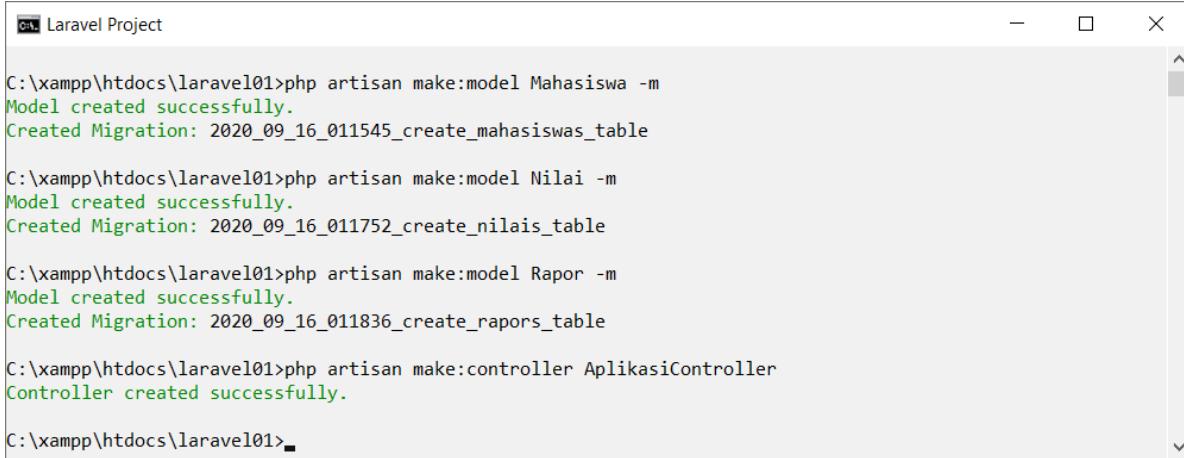
Eloquent Relationship: Has One Through

rapors.

Berikutnya, saya juga butuh satu buah controller sebagai tempat menulis kode program. Jalankan perintah berikut ke dalam cmd:

```
php artisan make:controller AplikasiController
```

Ke dalam **AplikasiController** inilah kita akan menulis berbagai method.



```
C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa -m
Model created successfully.
Created Migration: 2020_09_16_011545_create_mahasiswas_table

C:\xampp\htdocs\laravel01>php artisan make:model Nilai -m
Model created successfully.
Created Migration: 2020_09_16_011752_create_nilais_table

C:\xampp\htdocs\laravel01>php artisan make:model Rapor -m
Model created successfully.
Created Migration: 2020_09_16_011836_create_rapors_table

C:\xampp\htdocs\laravel01>php artisan make:controller AplikasiController
Controller created successfully.

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

Gambar: Pembuatan model, migration dan controller

Setelah semua file tersedia, buka file migration untuk ketiga tabel dan isi dengan kode 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->string('jurusan');
8         $table->timestamps();
9     });
10 }
```

```
database\migrations\<timestamp>_create_nilais_table.php
```

```
1 public function up()
2 {
3     Schema::create('nilais', function (Blueprint $table) {
4         $table->id();
5         $table->decimal('sem_1',3,2)->nullable();
6         $table->decimal('sem_2',3,2)->nullable();
7         $table->decimal('sem_3',3,2)->nullable();
8         $table->foreignId('mahasiswa_id')->unique()->constrained()
9             ->onDelete('cascade');
10        $table->timestamps();
11    });
12 }
```

Eloquent Relationship: Has One Through

```
12 }
```

```
database\migrations\<timestamp>_create_rapors_table.php
```

```
1 public function up()
2 {
3     Schema::create('rapors', function (Blueprint $table) {
4         $table->id();
5         $table->text('catatan')->nullable();
6         $table->foreignId('nilai_id')->unique()->constrained()
7             ->onDelete('cascade');
8         $table->timestamps();
9     });
10 }
```

Buat ketiga tabel dengan menjalankan proses migration: `php artisan migrate`.

Lanjut, kita perlu beberapa data sample. Silahkan buka file `DatabaseSeeder.php` lalu modifikasi sebagai berikut:

```
database\seeders\DatabaseSeeder.php
```

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7 use App\Models\Mahasiswa;
8 use App\Models\Nilai;
9 use App\Models\Rapor;
10
11 class DatabaseSeeder extends Seeder
12 {
13     public function run()
14     {
15         $faker = Faker::create('id_ID');
16         $faker->seed(123);
17         $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
18
19         for ($i=0; $i<10; $i++) {
20             Mahasiswa::create(
21                 [
22                     'nim' => $faker->unique()->numerify('10#####'),
23                     'nama' => $faker->firstName." ".$faker->lastName,
24                     'jurusan' => $faker->randomElement($jurusan),
25                 ]
26             );
27         }
28
29         for ($i=0; $i<5; $i++) {
30             Nilai::create(
31                 [
32                     'sem_1' => $faker->randomFloat(2, 2, 4),
```

Eloquent Relationship: Has One Through

```
33     'sem_2' => $faker->randomFloat(2, 2, 4),
34     'sem_3' => $faker->randomFloat(2, 2, 4),
35     'mahasiswa_id' => $faker->unique()->randomDigit,
36   ]
37 }
38 }
39
40 Rapor::create(
41 [
42   'catatan' => 'Ada peningkatan setiap semester, pertahankan',
43   'nilai_id' => 1
44 ]
45 );
46
47 Rapor::create(
48 [
49   'catatan' => 'Nilai semester 2 sangat turun, harus rajin lagi',
50   'nilai_id' => 2
51 ]
52 );
53
54 Rapor::create(
55 [
56   'catatan' => 'Usahakan agar IP selalu diatas 3',
57   'nilai_id' => 3
58 ]
59 );
60 }
61 }
```

File seeder ini akan men-generate 10 data mahasiswa, 5 data nilai dan 3 data rapor. Jumlah data setiap tabel sengaja di bedakan (tidak semuanya 10 data) agar kita bisa lihat permasalahan yang terjadi jika tanpa memakai *has one through relationship*.

Jalankan seeder dengan perintah `php artisan db:seed`, dan berikut isi data tabel `mahasiswas`, `nilais` dan `rapors` :

Eloquent Relationship: Has One Through

The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". It displays three separate SQL queries:

- Mahasiswa:** A query to select all columns from the "mahasiswa" table. The results show 10 rows of student information, including ID, NIM, name, major, and timestamps for creation and update.
- Nilai:** A query to select all columns from the "nilai" table. The results show 5 rows of grade data, linking each to a specific student ID.
- Rapors:** A query to select all columns from the "rapors" table. The results show 3 rows of notes, each associated with a specific grade ID.

Mahasiswa					
id	nim	nama	jurusan	created_at	updated_at
1	10300234	Mustofa Simanjuntak	Ilmu Komputer	2020-09-16 01:37:07	2020-09-16 01:37:07
2	10451982	Marsito Purnawati	Sistem Informasi	2020-09-16 01:37:07	2020-09-16 01:37:07
3	10980764	Ika Puspasari	Ilmu Komputer	2020-09-16 01:37:07	2020-09-16 01:37:07
4	10605319	Queen Suryatmi	Sistem Informasi	2020-09-16 01:37:07	2020-09-16 01:37:07
5	10438572	Yuliana Nurdyanti	Teknik Informatika	2020-09-16 01:37:07	2020-09-16 01:37:07
6	10737995	Tiara Siregar	Ilmu Komputer	2020-09-16 01:37:07	2020-09-16 01:37:07
7	10531551	Kambali Mulyani	Ilmu Komputer	2020-09-16 01:37:07	2020-09-16 01:37:07
8	10155818	Hesti Ramadhan	Teknik Informatika	2020-09-16 01:37:07	2020-09-16 01:37:07
9	10274992	Galang Maryadi	Teknik Informatika	2020-09-16 01:37:07	2020-09-16 01:37:07
10	10228263	Mahdi Rajata	Sistem Informasi	2020-09-16 01:37:07	2020-09-16 01:37:07

Nilai						
id	sem_1	sem_2	sem_3	mahasiswa_id	created_at	updated_at
1	2.18	2.44	3.14	9	2020-09-16 01:37:07	2020-09-16 01:37:07
2	3.13	2.35	3.01	1	2020-09-16 01:37:07	2020-09-16 01:37:07
3	3.15	2.15	3.39	8	2020-09-16 01:37:07	2020-09-16 01:37:07
4	2.85	3.50	2.20	3	2020-09-16 01:37:08	2020-09-16 01:37:08
5	3.89	2.98	3.00	4	2020-09-16 01:37:08	2020-09-16 01:37:08

Rapors					
id	catatan	nilai_id	created_at	updated_at	
1	Ada peningkatan setiap semester, pertahankan	1	2020-09-16 01:37:08	2020-09-16 01:37:08	
2	Nilai semester 2 sangat turun, harus rajin lagi	2	2020-09-16 01:37:09	2020-09-16 01:37:09	
3	Usahakan agar IP selalu diatas 3	3	2020-09-16 01:37:09	2020-09-16 01:37:09	

Gambar: Isi tabel mahasiswa, nilai dan rapors

15.3. Pendefinisian Relationship One to One

Sebelum kita masuk ke praktik *has one through*, setiap tabel harus dihubungkan terlebih dahulu dengan relationship *one to one*. Yakni antara tabel `mahasiswa` dengan tabel `nilai`, serta tabel `nilai` dengan tabel `rapors`.

Silahkan buka file model `Mahasiswa.php`, `Nilai.php`, dan `Rapor.php` lalu tambah method 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;
```

Eloquent Relationship: Has One Through

```
11     public function nilai()
12     {
13         return $this->hasOne('App\Models\Nilai');
14     }
15 }
```

App\Models\Nilai.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 Nilai extends Model
9 {
10    use HasFactory;
11    public function mahasiswa()
12    {
13        return $this->belongsTo('App\Models\Mahasiswa');
14    }
15
16    public function rapor()
17    {
18        return $this->hasOne('App\Models\Rapor');
19    }
20 }
```

app\Models\Rapor.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 Rapor extends Model
9 {
10    use HasFactory;
11    public function nilai()
12    {
13        return $this->belongsTo('App\Models\Nilai');
14    }
15 }
```

Dengan tambahan kode ini maka setiap tabel sudah terhubung dengan relationship *one to one* satu sama lain, kecuali antara tabel `mahasiswa` dengan tabel `raps`.

Sebagai uji coba, kita akan tes terlebih dahulu apakah semua relationship bisa diakses atau belum. Silahkan tambah 3 route berikut ke file `routes\web.php`:

Eloquent Relationship: Has One Through

routes\web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\ApplikasiController;
5
6 Route::get('/all', [ApplikasiController::class, 'all']);
7
8 Route::get('/relationship-1', [ApplikasiController::class, 'relationship1']);
9 Route::get('/relationship-2', [ApplikasiController::class, 'relationship2']);
```

Langsung masuk ke route pertama, berikut kode program untuk method all() di ApplikasiController.php:

app\Http\Controllers\ApplikasiController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use App\Models\Mahasiswa;
5 use App\Models\Nilai;
6 use App\Models\Rapor;
7
8 class ApplikasiController extends Controller
9 {
10     public function all()
11     {
12         echo "## Mahasiswa ## <br>";
13         $mahasiswas = Mahasiswa::all();
14         foreach ($mahasiswas as $mahasiswa) {
15             echo "$mahasiswa->id | $mahasiswa->nim |
16                   $mahasiswa->nama | $mahasiswa->jurusan <br>";
17         }
18
19         echo "<hr>";
20
21         echo "## Nilai ## <br>";
22         $nilais = Nilai::all();
23         foreach ($nilais as $nilai) {
24             echo "$nilai->id | $nilai->sem_1 | $nilai->sem_2 |
25                   $nilai->sem_3 | $nilai->mahasiswa_id <br>";
26         }
27
28         echo "<hr>";
29
30         echo "## Rapor ## <br>";
31         $rapors = Rapor::all();
32         foreach ($rapors as $rapor) {
33             echo "$rapor->id | $rapor->catatan | $rapor->nilai_id <br>";
34         }
35     }
36 }
```

Eloquent Relationship: Has One Through

```
## Mahasiswa ##
1 | 10300234 | Mustofa Simanjuntak | Ilmu Komputer
2 | 10451982 | Marsito Purnawati | Sistem Informasi
3 | 10980764 | Ika Puspasari | Ilmu Komputer
4 | 10605319 | Queen Suryatmi | Sistem Informasi
5 | 10438572 | Yuliana Nurdyanti | Teknik Informatika
6 | 10737995 | Tiara Siregar | Ilmu Komputer
7 | 10531551 | Kambali Mulyani | Ilmu Komputer
8 | 10155818 | Hesti Ramadan | Teknik Informatika
9 | 10274992 | Galang Maryadi | Teknik Informatika
10 | 10228263 | Mahdi Rajata | Sistem Informasi

## Nilai ##
1 | 2.18 | 2.44 | 3.14 | 9
2 | 3.13 | 2.35 | 3.01 | 1
3 | 3.15 | 2.15 | 3.39 | 8
4 | 2.85 | 3.50 | 2.20 | 3
5 | 3.89 | 2.98 | 3.00 | 4

## Rapor ##
1 | Ada peningkatan setiap semester, pertahankan | 1
2 | Nilai semester 2 sangat turun, harus rajin lagi | 2
3 | Usahakan agar IP selalu diatas 3 | 3
```

Gambar: Menampilkan isi semua tabel mahasiswas, nilais dan rapors

Dalam kode program ini saya menampilkan semua isi tabel `mahasiswas`, `nilais` dan `rapors` menggunakan perintah eloquent biasa (belum memakai relationship). Terlihat semua data sudah berhasil diakses.

Lanjut, masuk ke route kedua yang didefinisikan oleh method `relationship1()`:

app\Http\Controllers\AplikasiController.php

```
1 public function relationship1()
2 {
3     echo "## Mahasiswa - Nilai ## <br>";
4     $mahasiswas = Mahasiswa::has('nilai')->get();
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "$mahasiswa->nama | {$mahasiswa->nilai->sem_1}
7             {$mahasiswa->nilai->sem_2} {$mahasiswa->nilai->sem_3} <br>";
8     }
9
10    echo "<hr>";
11
12    echo "## Nilai - Rapor ## <br>";
13    $nilais = Nilai::has('rapor')->get();
14    foreach ($nilais as $nilai) {
15        echo "$nilai->sem_1 | $nilai->sem_2 | $nilai->sem_3 |
16            {$nilai->rapor->catatan} <br>";
17    }
18 }
```

Eloquent Relationship: Has One Through

```
## Mahasiswa - Nilai ##
Mustofa Simanjuntak | 3.13 2.35 3.01
Ika Puspasari | 2.85 3.50 2.20
Queen Suryatmi | 3.89 2.98 3.00
Hesti Ramadan | 3.15 2.15 3.39
Galang Maryadi | 2.18 2.44 3.14

## Nilai - Rapor ##
2.18 | 2.44 | 3.14 | Ada peningkatan setiap semester, pertahankan
3.13 | 2.35 | 3.01 | Nilai semester 2 sangat turun, harus rajin lagi
3.15 | 2.15 | 3.39 | Usahakan agar IP selalu diatas 3
```

Gambar: Hasil dari halaman <http://localhost:8000/relationship-1>

Kali ini saya sudah menggunakan *eloquent relationship*.

Di baris 4 terdapat perintah `Mahasiswa::has('nilai')->get()` untuk mencari semua mahasiswa yang punya pasangan data di tabel `nilais`. Kemudian saat proses echo, property `$mahasiswa->nilai->sem_1` bisa diakses untuk menampilkan nilai semester 1 mahasiswa.

Variabel `$mahasiswas` otomatis berisi 5 data saja, sesuai dengan jumlah data di tabel `nilais`. Mahasiswa yang tidak memiliki nilai tidak akan di proses. Ini semua merupakan penerapan *relationship one to one* antara tabel `mahasiswas` ke tabel `nilais`.

Begini juga dengan kode program di baris 13, perintah `Nilai::has('rapor')->get()` dipakai untuk mencari semua nilai yang memiliki pasangan data di tabel `rapors`. Variabel `$nilais` hanya berisi 3 data sesuai dengan jumlah data yang ada di tabel `rapors`.

Sekarang, bagaimana cara mengakses tabel `rapors` secara langsung dari tabel `mahasiswas`? Kita bisa coba dengan kode berikut:

app\Http\Controllers\AplikasiController.php

```
1     public function relationship2()
2     {
3         echo "## Mahasiswa - Rapor ## <br>";
4         $mahasiswas = Mahasiswa::has('nilai')->get();
5         foreach ($mahasiswas as $mahasiswa) {
6             echo "$mahasiswa->nama | {$mahasiswa->nilai->rapor->catatan} <br>";
7         }
8     }
```



Gambar: Error saat mengakses rapor

Mirip seperti sebelumnya, di baris 4 terdapat perintah `Mahasiswa::has('nilai')->get()` untuk mencari semua mahasiswa yang memiliki pasangan data di tabel `nilais`. Dari hasil ini, diakses property `$mahasiswa->nilai->rapor->catatan` untuk menampilkan rapor dari mahasiswa tersebut.

Meskipun agak panjang, tidak ada yang salah dari perintah `$mahasiswa->nilai->rapor->catatan`. Ini artinya kita mengakses tabel rapor melalui tabel nilai.

Akan tetapi tampil error karena tidak semua mahasiswa memiliki data rapor. Variabel `$mahasiswas` dalam contoh di atas akan berisi 5 data, sedangkan tabel `raps` hanya berisi 3 data.

15.4. Eloquent Relationship `hasOneThrough()`

Dari percobaan sebelumnya kita mengalami kendala karena tidak bisa mengakses data `raps` dari tabel `mahasiswas`. Seharusnya perintah yang dijalankan adalah `Mahasiswa::has('rapor')`, bukan `Mahasiswa::has('nilai')` karena tujuan akhir adalah mengakses data di tabel `raps`.

Inilah fungsi dari relationship `has one through`, yakni agar tabel `mahasiswas` bisa terhubung ke tabel `raps`. Untuk membuatnya, silahkan tambah method berikut ke file model `Mahasiswa`:

app\Models\Mahasiswa.php

```
1 <?php
2 ...
3     public function rapor()
4     {
5         return $this->hasOneThrough( 'App\Models\Rapor', 'App\Models\Nilai' );
6     }
```

Method `rapor()` berisi satu perintah relationship, yakni `return $this->hasOneThrough('App\Models\Rapor', 'App\Models\Nilai')`.

Eloquent Relationship: Has One Through

Argument pertama dari method `hasOneThrough()` diisi dengan model yang menjadi tujuan *relationship*, yakni '`App\Models\Rapor`'. Sedangkan argument kedua diisi dengan model yang menjadi penghubung, yakni '`App\Models\Nilai`'.

Dengan tambahan ini, maka tabel mahasiswa sudah terhubung ke tabel rapors. Untuk melihat cara penggunaannya, silahkan tambah 4 route ini ke file `routes\web.php`:

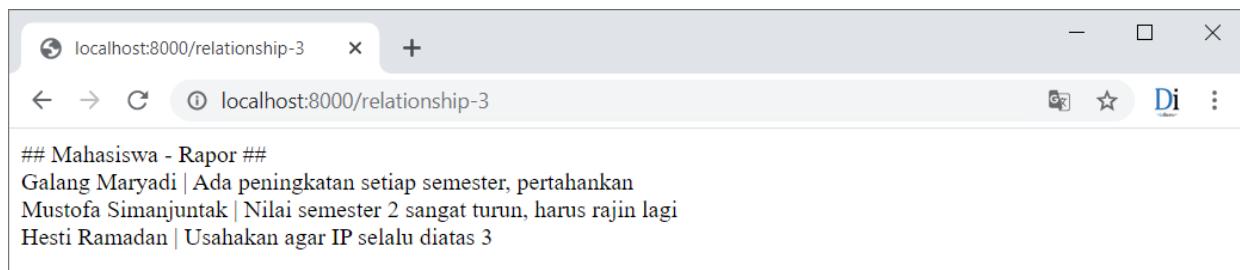
`routes\web.php`

```
1 ...  
2  
3 Route::get('/relationship-3', [ApplikasiController::class, 'relationship3']);  
4  
5 Route::get('/input-1', [ApplikasiController::class, 'input1']);  
6 Route::get('/input-2', [ApplikasiController::class, 'input2']);  
7  
8 Route::get('/delete', [ApplikasiController::class, 'delete']);
```

Berikut kode program untuk route pertama:

`app\Http\Controllers\ApplikasiController.php`

```
1 public function relationship3()  
2 {  
3     echo "## Mahasiswa - Rapor ## <br>";  
4     $mahasiswas = Mahasiswa::has('rapor')->get();  
5     foreach ($mahasiswas as $mahasiswa) {  
6         echo "$mahasiswa->nama | {$mahasiswa->rapor->catatan} <br>";  
7     }  
8 }
```



Gambar: Hasil dari halaman `http://localhost:8000/relationship-3`

Karena *relationship has one through* sudah tersedia, maka kita bisa menjalankan perintah `Mahasiswa::has('rapor')` seperti di baris 4. Perintah ini secara langsung akan men-filter data mahasiswa yang memiliki rapor, sehingga variabel `$mahasiswas` hanya berisi 3 data saja sesuai dengan jumlah data di tabel rapors.

Bagaimana dengan proses input? Mari kita coba:

`app\Http\Controllers\ApplikasiController.php`

```
1 public function input1()  
2 {
```

Eloquent Relationship: Has One Through

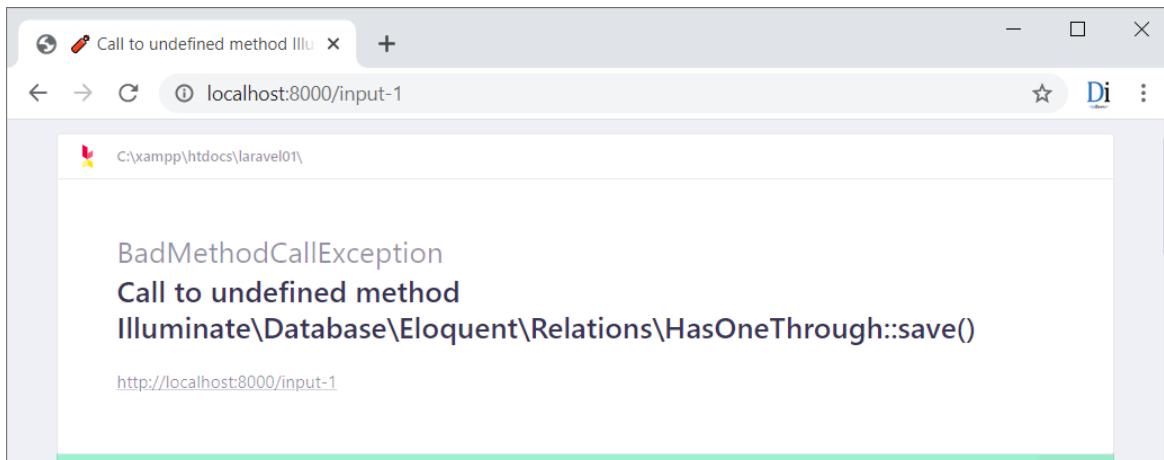
```
3 $mahasiswa = Mahasiswa::where('nama', 'Tiara Siregar')->first();  
4  
5 $nilai = new Nilai;  
6 $nilai->sem_1 = 1.12;  
7 $nilai->sem_2 = 1.23;  
8 $nilai->sem_3 = 1.34;  
9  
10 $mahasiswa->nilai()->save($nilai);  
11  
12 $rapor = new Rapor;  
13 $rapor->catatan = 'Ini serius kuliah ga sih?';  
14  
15 $mahasiswa->rapor()->save($rapor);  
16  
17 echo "Penambahan data $mahasiswa->nama berhasil";  
18 }
```

Di baris 3 saya mengisi variabel `$mahasiswa` dengan object Mahasiswa bernama 'Tiara Siregar'. Data ini sudah tersedia di tabel `mahasiswas`.

Kemudian di baris 5 – 8 saya membuat sebuah object **Nilai** baru. Setelah itu nilai ini diinput ke dalam tabel dengan perintah `$mahasiswa->nilai()->save($nilai)`. Ini merupakan kode eloquent relationship one to one yang sudah pernah kita pelajari.

Selanjutnya di baris 12 – 13 terdapat perintah untuk membuat object **Rapor** baru, lalu coba di input ke tabel rapor dengan perintah `$mahasiswa->rapor()->save($rapor)`. Ini adalah bagian dari relationship *has one through* karena nilai rapor diinput langsung lewat object Mahasiswa.

Berikut hasilnya:



Gambar: Error saat mengakses <http://localhost:8000/input-1>

Ternyata terjadi error karena *has one through* tidak mendukung proses input lewat perintah `$mahasiswa->rapor()->save($rapor)`. Ketika mengakses alamat `localhost:8000/input-1`, data nilai sudah masuk ke database, tapi tidak dengan data rapor.

Solusinya adalah, data rapor mesti diinput melalui object Nilai, bukan dari object Mahasiswa:

Eloquent Relationship: Has One Through

app\Http\Controllers\AplikasiController.php

```
1 public function input2()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Yuliana Nurdiyanti')->first();
4
5     $nilai = new Nilai;
6     $nilai->sem_1 = 1.12;
7     $nilai->sem_2 = 1.23;
8     $nilai->sem_3 = 1.34;
9
10    $mahasiswa->nilai()->save($nilai);
11
12    $rapor = new Rapor;
13    $rapor->catatan = 'Ini serius kuliah ga sih?';
14
15    $nilai->rapor()->save($rapor);
16
17    echo "Penambahan data $mahasiswa->nama berhasil";
18 }
```

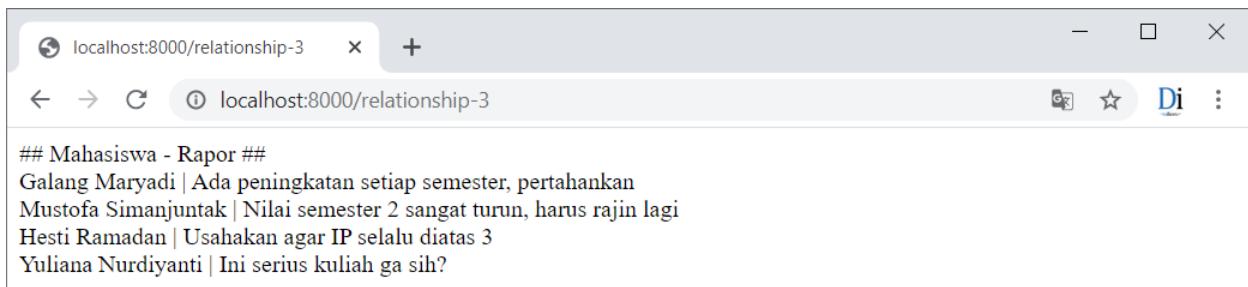


Gambar: Tampilan halaman http://localhost:8000/input-2

Sama seperti sebelumnya, di baris 3 saya mengisi object `$mahasiswa` dengan object `Mahasiswa`, tapi kali ini menggunakan nama yang berbeda agar tidak bentrok dengan data sebelumnya.

Perbedaan lain ada di baris 15. proses input data rapor tidak dilakukan dari variabel `$mahasiswa`, tapi dari variabel `$nilai`. Ini sukses berjalankan karena tabel `nilais` dengan tabel `rapors` memiliki *relationship one to one*.

Jika diakses halaman `localhost:8000/relationship-3`, maka sekarang akan tampil 4 data hasil penambahan di atas:



Gambar: Tampilan halaman http://localhost:8000/relationship-3

Terakhir, kita akan coba hapus data mahasiswa:

Eloquent Relationship: Has One Through

app\Http\Controllers\AplikasiController.php

```
1 public function delete()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Galang Maryadi')->first();
4     $mahasiswa->delete();
5
6     echo "Data $mahasiswa->nama berhasil di hapus";
7 }
```



Gambar: Proses penghapusan data



Gambar: Data 'Galang Maryadi' sudah tidak ada lagi

Perintah di baris 3 akan menghapus data mahasiswa bernama 'Galang Maryadi'. Efeknya, data nilai dan juga data rapor untuk mahasiswa tersebut juga ikut terhapus. Ini berasal dari efek ON DELETE CASCADE pada saat pendefinisian migration.

Sebagai info tambahan, saat ini Laravel tidak menyediakan relationship *belongs to through*, yakni hubungan timbal balik dari tabel `raps` ke tabel `mahasiswas`. Jika ingin membuatnya, bisa memakai library pihak ketiga seperti **staudenmeir/belongs-to-through** (github.com/staudenmeir/belongs-to-through).

Dalam bab ini kita telah membahas relationship *has one through*. Berikutnya akan disambung dengan relationship **has many through**.

16. Eloquent Relationship: Has Many Through

Relationship has many through merupakan variasi dari relationship *one to many*. Hubungan yang akan kita pelajari ini sangat mirip seperti *one many through* tapi untuk versi *one to many*.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel101
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

16.1. Pengertian Has Many Through Relationship

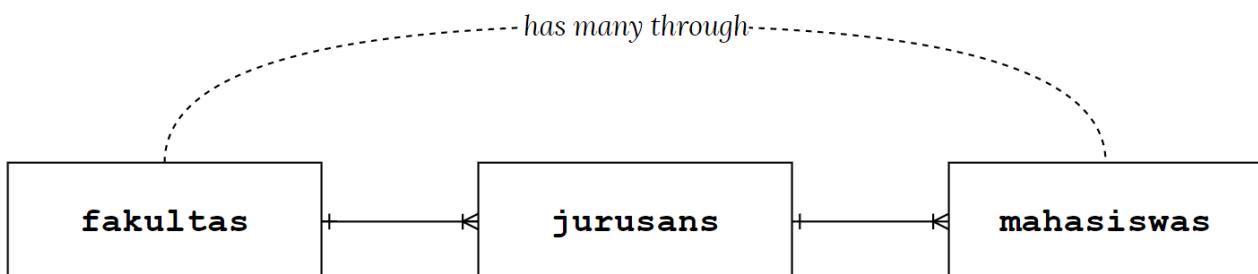
Mirip seperti *has one through*, relationship *has many through* adalah relasi atau hubungan *one to many* antara tabel utama dengan tabel ketiga dengan perantara tabel kedua.

Sebagai contoh, di dalam praktek *relationship one to many* kita sudah memiliki tabel *jurusans* dan tabel *mahasiswa*s. Hubungan yang terjadi adalah satu jurusan bisa memiliki banyak mahasiswa, akan tetapi satu data mahasiswa terhubung ke satu jurusan.

Sekarang saya akan tambah tabel ketiga, yakni tabel *fakultas*. Tabel *fakultas* ini bisa berisi banyak jurusan, namun satu jurusan akan terdaftar di satu fakultas. Jadi hubungan yang terjadi adalah *one to many*.

Dengan tambahan tabel, bagaimana hubungan antara tabel *fakultas* dengan tabel *mahasiswa*? Inilah yang dimaksud dengan **has many through relationship**.

Diagram berikut menggambarkan *relationship* yang terjadi:

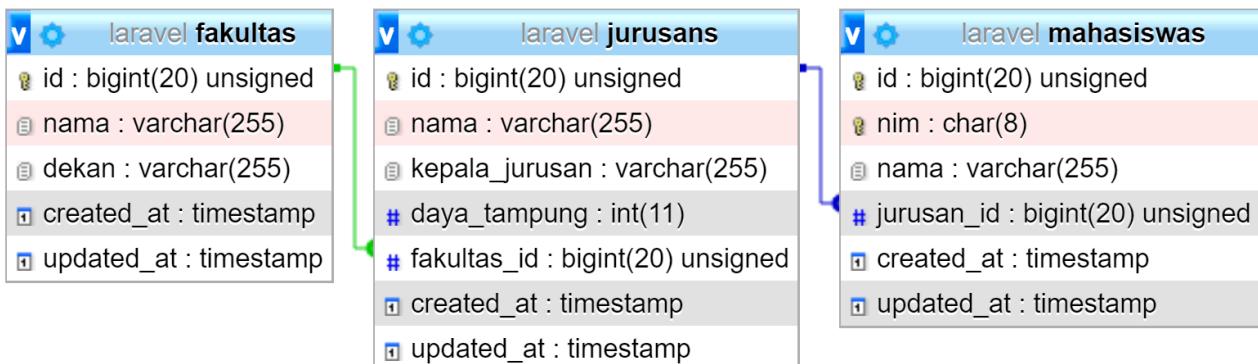


Gambar: Relationship has many through antara tabel fakultas dengan tabel mahasiswa

16.2. Persiapan Awal

Sebagai bahan praktik untuk penerapan *has many through*, kita akan buat 3 buah tabel: **fakultas**, **jurusans** dan **mahasiswa**. Berikut struktur ketiga tabel tersebut:

- ◆ Tabel **fakultas**: id, nama, dekan, created_at dan updated_at.
- ◆ Tabel **jurusans**: id, nama, kepala_jurusan, sayang_tampung, fakultas_id, created_at dan updated_at.
- ◆ Tabel **mahasiswa**: id, nim, nama, jurusan_id, created_at dan updated_at.



Gambar: Hubungan antara tabel mahasiswa, matakuliah, serta mahasiswa_matakuliah

Terlihat garis antara kolom `id` di tabel **fakultas** dengan kolom `fakultas_id` di tabel **jurusans**. Serta garis antara kolom `id` di tabel **jurusans** dengan kolom `jurusan_id` di tabel **mahasiswa**. Kedua garis ini mencerminkan hubungan *one to many*.

Hubungan relationship *has many through* antara tabel **fakultas** dengan tabel **mahasiswa** akan kita definisikan di dalam model Laravel.

Generate Data Sample

Kita akan buat ketiga tabel melalui migration. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```
php artisan make:model Fakultas -m
php artisan make:model Jurusan -m
php artisan make:model Mahasiswa -m
```

Kode di atas akan membuat file **model** dan **migration** untuk tabel **fakultas**, **jurusans**, dan **mahasiswa**.

Sama seperti di bab *has one through*, kita juga butuh satu controller sebagai tempat menulis kode program. Jalankan perintah berikut ke dalam cmd:

```
php artisan make:controller AplikasiController
```

Di dalam **AplikasiController** inilah kita akan menulis berbagai method.

Eloquent Relationship: Has Many Through

```
C:\xampp\htdocs\laravel01>php artisan make:model Fakultas -m
Model created successfully.
Created Migration: 2020_09_17_034547_create_fakultas_table

C:\xampp\htdocs\laravel01>php artisan make:model Jurusan -m
Model created successfully.
Created Migration: 2020_09_17_034604_create_jurusans_table

C:\xampp\htdocs\laravel01>php artisan make:model Mahasiswa -m
Model created successfully.
Created Migration: 2020_09_17_034653_create_mahasiswas_table

C:\xampp\htdocs\laravel01>php artisan make:controller AplikasiController
Controller created successfully.

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

Gambar: Pembuatan model, migration dan controller

Setelah semua file tersedia, buka file migration untuk ketiga tabel dan isi dengan kode berikut:

database\migrations\<timestamp>_create_fakultas_table.php

```
1 public function up()
2 {
3     Schema::create('fakultas', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->string('dekan');
7         $table->timestamps();
8     });
9 }
```

database\migrations\<timestamp>_create_jurusans_table.php

```
1 public function up()
2 {
3     Schema::create('jurusans', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->string('kepala_jurusan');
7         $table->integer('daya_tampung');
8         $table->foreignId('fakultas_id')->constrained()->onDelete('cascade');
9         $table->timestamps();
10    });
11 }
```

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->foreignId('jurusan_id')->constrained()->onDelete('cascade');
```

Eloquent Relationship: Has Many Through

```
8     $table->timestamps();
9 );
10 }
```

Buat ketiga tabel dengan menjalankan perintah migration: `php artisan migrate`.

Dalam contoh ini kita membuat model bernama **Fakultas**, namun tabel yang dihasilkan dari proses migration tetap `fakultas`, bukan menjadi `fakultass` (dengan tambahan 's' untuk plural).

Sepertinya ini berlaku untuk semua model yang diakhiri dengan huruf s, dimana nama tabel tetap sama dengan nama model, tidak lagi ditambah akhiran 's'.

Lanjut, kita perlu beberapa data sample. Silahkan buka file `DatabaseSeeder.php` lalu modifikasi sebagai berikut:

`database\seeders\DatabaseSeeder.php`

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7 use App\Models\Fakultas;
8 use App\Models\Jurusan;
9 use App\Models\Mahasiswa;
10
11 class DatabaseSeeder extends Seeder
12 {
13     public function run()
14     {
15         Fakultas::create(
16             [
17                 'nama' => 'Fakultas Ilmu Komputer',
18                 'dekan' => 'Prof Soerjadi',
19             ]
20         );
21
22         Fakultas::create(
23             [
24                 'nama' => 'Fakultas MIPA',
25                 'dekan' => 'Dr. Nita Sumartini M.Si',
26             ]
27         );
28
29         Fakultas::create(
30             [
31                 'nama' => 'Fakultas Hukum',
32                 'dekan' => 'Prof Chandra Nababan ',
33             ]
34     );
35 }
```

Eloquent Relationship: Has Many Through

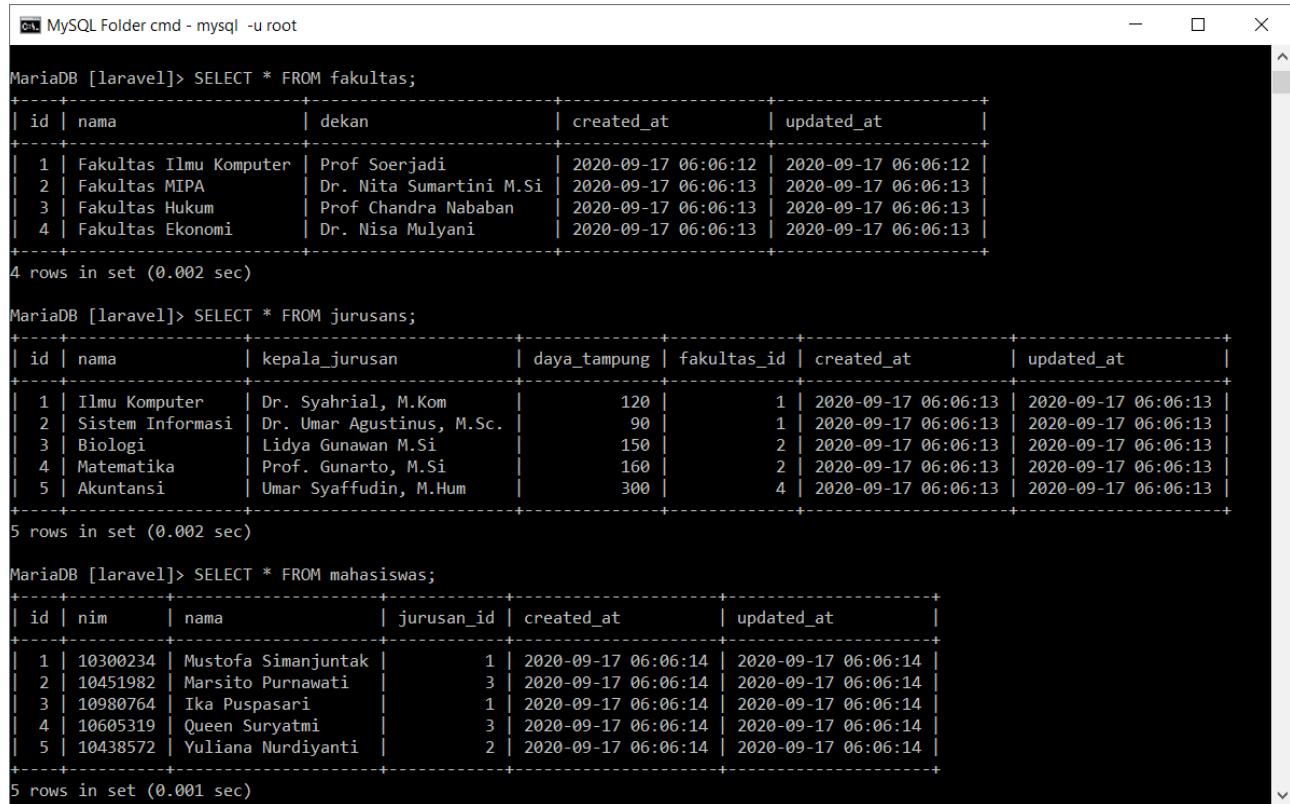
```
35 Fakultas::create(
36     [
37         'nama' => 'Fakultas Ekonomi',
38         'dekan' => 'Dr. Nisa Mulyani',
39     ]
40 );
41
42 Jurusan::create(
43     [
44         'nama' => 'Ilmu Komputer',
45         'kepala_jurusan' => 'Dr. Syahrial, M.Kom',
46         'daya_tampung' => 120,
47         'fakultas_id' => 1,
48     ]
49 );
50
51 Jurusan::create(
52     [
53         'nama' => 'Sistem Informasi',
54         'kepala_jurusan' => 'Dr. Umar Agustinus, M.Sc.',
55         'daya_tampung' => 90,
56         'fakultas_id' => 1,
57     ]
58 );
59
60 Jurusan::create(
61     [
62         'nama' => 'Biologi',
63         'kepala_jurusan' => 'Lidya Gunawan M.Si',
64         'daya_tampung' => 150,
65         'fakultas_id' => 2,
66     ]
67 );
68
69 Jurusan::create(
70     [
71         'nama' => 'Matematika',
72         'kepala_jurusan' => 'Prof. Gunarto, M.Si',
73         'daya_tampung' => 160,
74         'fakultas_id' => 2,
75     ]
76 );
77
78 Jurusan::create(
79     [
80         'nama' => 'Akuntansi',
81         'kepala_jurusan' => 'Umar Syaffudin, M.Hum',
82         'daya_tampung' => 300,
83         'fakultas_id' => 4,
84     ]
85 );
86
87
88 $faker = Faker::create('id_ID');
89 $faker->seed(123);
```

Eloquent Relationship: Has Many Through

```
90
91     for ($i=0; $i<5; $i++) {
92         Mahasiswa::create(
93             [
94                 'nim' => $faker->unique()->numerify('10#####'),
95                 'nama' => $faker->firstName()." ".$faker->lastName,
96                 'jurusan_id' => $faker->numberBetween(1, 3),
97             ]
98         );
99     }
100 }
101 }
102 }
```

File seeder ini akan men-generate 4 data fakultas, 5 data jurusan dan 5 data mahasiswa.

Jalankan seeder dengan perintah `php artisan db:seed`, dan berikut isi data ketiga tabel:



```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM fakultas;
+----+-----+-----+-----+-----+
| id | nama | dekan | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1 | Fakultas Ilmu Komputer | Prof Soerjadi | 2020-09-17 06:06:12 | 2020-09-17 06:06:12 |
| 2 | Fakultas MIPA | Dr. Nita Sumartini M.Si | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
| 3 | Fakultas Hukum | Prof Chandra Nababan | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
| 4 | Fakultas Ekonomi | Dr. Nisa Mulyani | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
+----+-----+-----+-----+-----+
4 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM jurusans;
+----+-----+-----+-----+-----+-----+
| id | nama | kepala_jurusan | daya_tampung | fakultas_id | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1 | Ilmu Komputer | Dr. Syahrial, M.Kom | 120 | 1 | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
| 2 | Sistem Informasi | Dr. Umar Agustinus, M.Sc. | 90 | 1 | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
| 3 | Biologi | Lidya Gunawan M.Si | 150 | 2 | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
| 4 | Matematika | Prof. Gunarto, M.Si | 160 | 2 | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
| 5 | Akuntansi | Umar Syaffudin, M.Hum | 300 | 4 | 2020-09-17 06:06:13 | 2020-09-17 06:06:13 |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+
| id | nim | nama | jurusan_id | created_at | updated_at |
+----+-----+-----+-----+-----+
| 1 | 10300234 | Mustofa Simanjuntak | 1 | 2020-09-17 06:06:14 | 2020-09-17 06:06:14 |
| 2 | 10451982 | Marsito Purnawati | 3 | 2020-09-17 06:06:14 | 2020-09-17 06:06:14 |
| 3 | 10980764 | Ika Puspasari | 1 | 2020-09-17 06:06:14 | 2020-09-17 06:06:14 |
| 4 | 10605319 | Queen Suryatmi | 3 | 2020-09-17 06:06:14 | 2020-09-17 06:06:14 |
| 5 | 10438572 | Yuliana Nurdyanti | 2 | 2020-09-17 06:06:14 | 2020-09-17 06:06:14 |
+----+-----+-----+-----+-----+
5 rows in set (0.001 sec)
```

Gambar: Isi tabel fakultas, jurusans dan mahasiswas

16.3. Pendefinisian Relationship One to Many

Apa yang akan kita lakukan sebenarnya perulangan dari bab *has one through relationship*. Oleh karena itu saya ingin menjadikannya sebagai latihan.

Exercise

Saat ini kita memiliki 3 tabel: `fakultas`, `jurusans` dan `mahasiswa`. Buatlah method di file Model untuk relationship *one to many* antara tabel `fakultas` dengan tabel `jurusans` serta juga relationship *one to many* antara tabel `jurusans` dengan tabel `mahasiswa`.

Answer

Berikut kode yang diperlukan:

app\Models\Fakultas.php

```
1 <?php
2 ...
3 class Fakultas extends Model
4 {
5     public function jurusans()
6     {
7         return $this->hasMany('App\Models\Jurusan');
8     }
9 }
```

app\Models\Jurusan.php

```
1 <?php
2 ...
3 class Jurusan extends Model
4 {
5     public function mahasiswa()
6     {
7         return $this->hasMany('App\Models\Mahasiswa');
8     }
9
10    public function fakultas()
11    {
12        return $this->belongsTo('App\Models\Fakultas');
13    }
14 }
```

app\Models\Mahasiswa.php

```
1 <?php
2 ...
3 class Mahasiswa extends Model
4 {
5     public function jurusan()
6     {
7         return $this->belongsTo('App\Models\Jurusan');
8     }
9 }
```

Exercise

Lanjut ke latihan kedua, buatlah sebuah route yang ketika diakses akan menampilkan semua isi tabel `fakultas`, `jurusans` dan `mahasiswa` secara terpisah. Berikut hasil akhir yang di inginkan:

```
## Fakultas ##
1 | Fakultas Ilmu Komputer | Prof Soerjadi
2 | Fakultas MIPA | Dr. Nita Sumartini M.Si
3 | Fakultas Hukum | Prof Chandra Nababan
4 | Fakultas Ekonomi | Dr. Nisa Mulyani

## Jurusan ##
1 | Ilmu Komputer | Dr. Syahrial, M.Kom | 120 | 1
2 | Sistem Informasi | Dr. Umar Agustinus, M.Sc. | 90 | 1
3 | Biologi | Lidya Gunawan M.Si | 150 | 2
4 | Matematika | Prof. Gunarto, M.Si | 160 | 2
5 | Akuntansi | Umar Syaffudin, M.Hum | 300 | 4

## Mahasiswa ##
1 | 10300234 | Mustofa Simanjuntak | 1
2 | 10451982 | Marsito Purnawati | 3
3 | 10980764 | Ika Puspasari | 1
4 | 10605319 | Queen Suryatmi | 3
5 | 10438572 | Yuliana Nurdyanti | 2
```

Gambar: Menampilkan seluruh data tabel

Answer

Untuk membuat tampilan ini kita perlu sebuah route:

```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\AplikasiController;
5
6 Route::get('/all', [AplikasiController::class, 'All']);
```

Nama method dan alamat route boleh bebas. Dalam contoh ini saya menggunakan route '/all' yang akan mengakses method `all()` di `AplikasiController`:

```
app\Http\Controllers\AplikasiController.php
```

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use App\Models\Fakultas;
```

```
5 use App\Models\Jurusan;
6 use App\Models\Mahasiswa;
7
8 class AplikasiController extends Controller
9 {
10     public function all()
11     {
12         echo "## Fakultas ## <br>";
13         $fa = Fakultas::all();
14         foreach ($fa as $fakultas) {
15             echo "$fakultas->id | $fakultas->nama | $fakultas->dekan <br>";
16         }
17
18         echo "<hr>";
19
20         echo "## Jurusan ## <br>";
21         $jurusans = Jurusan::all();
22         foreach ($jurusans as $jurusan) {
23             echo "$jurusan->id | $jurusan->nama |
24                 $jurusan->kepala_jurusan |
25                 $jurusan->daya_tampung | $jurusan->fakultas_id <br>";
26         }
27
28         echo "<hr>";
29
30         echo "## Mahasiswa ## <br>";
31         $mahasiswas = Mahasiswa::all();
32         foreach ($mahasiswas as $mahasiswa) {
33             echo "$mahasiswa->id | $mahasiswa->nim |
34                 $mahasiswa->nama | $mahasiswa->jurusan_id <br>";
35         }
36     }
37 }
```

Kode yang diperlukan hanya perintah eloquent biasa yang digabung dengan perulangan foreach untuk menampilkan semua data.

Exercise

Latihan ketiga, buatlah sebuah route yang ketika diakses akan menampilkan:

1. Semua jurusan yang ada di 'Fakultas MIPA'
2. Semua mahasiswa yang ada di jurusan 'Ilmu Komputer'

Berikut hasil akhir yang di inginkan:

Eloquent Relationship: Has Many Through



Gambar: Menampilkan jurusan di Fakultas MIPA dan mahasiswa Ilmu Komputer

Answer

Kali ini kita masuk ke materi eloquent relationship. Berikut route yang akan saya pakai:

routes\web.php

```
1 <?php
2 ...
3 use App\Http\Controllers\ApplikasiController;
4
5 Route::get('/relationship-1',[ApplikasiController::class,'relationship1']);
```

Route di atas akan menjalankan method relationship1() di ApplikasiController.php:

app\Http\Controllers\ApplikasiController.php

```
1 public function relationship1()
2 {
3     echo "## Fakultas MIPA ## <br>";
4     $fakultas = Fakultas::where('nama', 'Fakultas MIPA')->first();
5     foreach ($fakultas->jurusans as $jurusan) {
6         echo "$jurusan->nama <br>";
7     }
8
9     echo "<hr>";
10
11    echo "## Jurusan Ilmu Komputer ## <br>";
12    $jurusans = Jurusan::where('nama', 'Ilmu Komputer')->first();
13    foreach ($jurusans->mahasiswas as $mahasiswa) {
14        echo "$mahasiswa->nama <br>";
15    }
16 }
```

Exercise

Latihan ke empat, buatlah sebuah route yang ketika diakses akan menampilkan semua mahasiswa yang ada di Fakultas MIPA. Syarat tambahan: hanya boleh menggunakan 1 perintah eloquent.

Berikut hasil yang diinginkan:



Gambar: Menampilkan seluruh mahasiswa Fakultas MIPA

Answer

Jika anda menebak bahwa ini hanya bisa dilakukan dengan *relationship has many through*, maka itu tidak salah. Kasus ini sangat pas dipecahkan dengan menambah relationship tersebut.

Akan tetapi sebelum ke sana, saya ingin menampilkan sedikit "trik" yang bisa dipakai. Berikut kode programnya:

```
routes\web.php

1 <?php
2 ...
3 use App\Http\Controllers\ApplikasiController;
4
5 Route::get('/relationship-2', [ApplikasiController::class, 'relationship2']);

app\Http\Controllers\ApplikasiController.php

1 public function relationship2()
2 {
3     $fakultas = Fakultas::where('nama', 'Fakultas MIPA')->first();
4     echo "## Mahasiswa $fakultas->nama ## <br>";
5
6     foreach ($fakultas->jurusans as $jurusan) {
7         foreach ($jurusan->mahasiswas as $mahasiswa) {
8             echo "$mahasiswa->nama <br> ";
9         }
10    }
11 }
```

Semua mahasiswa untuk jurusan Fakultas MIPA bisa ditampilkan dengan nested foreach. Perulangan pertama untuk mengambil semua jurusan, dan perulangan kedua untuk menampilkan semua mahasiswa dari setiap jurusan tersebut.

16.4. Eloquent Relationship hasManyThrough()

Dalam latihan sebelum ini, data mahasiswa tetap bisa diakses dari fakultas, akan tetapi kita menggunakan 2 buah perulangan foreach. Cara yang lebih ideal adalah dengan menambah relationship *has many through*, yakni agar tabel `fakultass` bisa langsung terhubung ke tabel `mahasiswas`. Untuk membuatnya, silahkan tambah method berikut ke file model Fakultas:

app\Models\Fakultas.php

```
1 <?php
2 ...
3 public function mahasiswas()
4 {
5     return $this->hasManyThrough('App\Models\Mahasiswa', 'App\Models\Jurusan');
6 }
```

Sama seperti di method `hasOneThrough()`, argument pertama dari method `hasManyThrough()` diisi dengan model yang menjadi tujuan *relationship*, yakni '`App\Models\Mahasiswa`'. Sedangkan argument kedua diisi dengan model yang menjadi penghubung, yakni '`App\Models\Jurusan`'.

Dengan tambahan ini, maka tabel fakultas sudah terhubung ke tabel `mahasiswas`. Berikut kode yang bisa dipakai untuk menampilkan seluruh mahasiswa dari 'fakultas MIPA':

routes\web.php

```
Route::get('/relationship-3', [AplikasiController::class, 'relationship3']);
```

app\Http\Controllers\AplikasiController.php

```
1 public function relationship3()
2 {
3     $fakultas = Fakultas::where('nama', 'Fakultas MIPA')->first();
4     echo "## Mahasiswa $fakultas->nama ## <hr>";
5
6     foreach ($fakultas->mahasiswas as $mahasiswa) {
7         echo "$mahasiswa->nama <br>";
8     }
9 }
```

Eloquent Relationship: Has Many Through



Gambar: Menampilkan seluruh mahasiswa Fakultas MIPA

Kali ini cukup dengan satu perulangan foreach, karena kita bisa menjalankan perintah `$fakultas->mahasiswas` untuk mengakses object Mahasiswa dari variabel `$fakultas`.

Contoh penerapan lain dari relationship *has many through* adalah menghitung jumlah mahasiswa dari fakultas tertentu. Misal, berapa total mahasiswa dari setiap jurusan? Itu bisa dicari dengan kode berikut:

```
routes\web.php  
Route::get('/count', [AplikasiController::class, 'count']);  
  
app\Http\Controllers\AplikasiController.php  
1 public function count()  
2 {  
3     $fa = Fakultas::withCount('mahasiswas')->get();  
4  
5     foreach ($fa as $fakultas) {  
6         echo "$fakultas->nama memiliki  
7             $fakultas->mahasiswas_count mahasiswa <br> ";  
8     }  
9 }
```



Gambar: Total mahasiswa untuk setiap fakultas

Tambahan method `withCount('mahasiswas')` di baris 3 akan menambahkan property `mahasiswas_count` yang berisi jumlah mahasiswa yang ada di setiap fakultas.

Juga sama seperti hubungan *has one through*, proses input mahasiswa tidak bisa dilakukan secara langsung dari object Fakultas, tetapi harus dari object Jurusan yang memiliki hubungan *one to many* dengan tabel `mahasiswas`.

16.5. Haruskah Menggunakan Relationship Has Through?

Dalam 2 bab ini kita sudah membahas tentang *has one through* dan *has many through* relationship. Keduanya dipakai untuk membuat hubungan antara tabel pertama ke tabel ketika.

Tapi sebenarnya ada cara lain untuk menghubungkan tabel pertama ke tabel ketiga, yakni membuat langsung hubungan *one to one* atau *one to many*. Misalnya kode program di model Fakultas bisa saya buat seperti ini:

app\Models\Fakultas.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 Fakultas extends Model
9 {
10     use HasFactory;
11     public function jurusans()
12     {
13         return $this->hasMany('App\Models\Jurusan');
14     }
15
16     public function mahasiswa()
17     {
18         return $this->hasMany('App\Models\Mahasiswa');
19     }
20 }
```

Ini artinya tabel fakultas akan memiliki hubungan *one to many* ke tabel jurusans dan juga *one to many* ke tabel mahasiswa. Dengan teknik ini, kita bisa menampilkan semua mahasiswa untuk satu fakultas tertentu secara langsung, tidak perlu melewati tabel jurusans.

Akan tetapi ada beberapa konsekuensi logika yang terjadi. Pertama, tabel mahasiswa harus ditambah satu kolom baru bernama `fakultas_id`, yang bertindak sebagai *foreign key* dari tabel fakultas.

Dan kedua, tabel mahasiswa sekarang terhubung ke tabel fakultas dan juga tabel jurusans. Ini bisa jadi masalah karena kita harus menjaga keduanya agar selalu sinkron.

Sebagai contoh, Fakultas MIPA memiliki jurusan Biologi. Ketika kita mendaftarkan mahasiswa baru ke jurusan Biologi, maka kolom `fakultas_id` di tabel mahasiswa juga harus diisi dengan nomor id milik fakultas MIPA. Jika tidak, datanya jadi tidak sinkron karena bisa saja mahasiswa jurusan Biologi terdaftar di fakultas Ekonomi.

Keputusan apakah ingin menggunakan relationship *one to many* atau *has many through* perlu

dipikirkan secara matang karena ada dampak logika dari keduanya.

Dalam bab ini kita telah menyelesaikan materi *has many through* relationship. Selanjutnya akan masuk ke relationship yang tidak kalah rumit dan lebih menantang, yakni **one to one polymorphic**.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

17. Eloquent Relationship: One to One Polymorphic

Polymorphic relationship memperkenalkan sebuah konsep lain dalam membuat hubungan antar tabel. Kali ini data di sebuah tabel bisa terhubung tidak hanya ke satu tabel saja, tapi bisa ke banyak tabel sekaligus.

Polymorphic relationship juga hadir dalam 3 "rasa", yakni *one to one*, *one to many*, dan *many to many*. Kita akan bahas mulai dari versi yang paling sederhana: **one to one polymorphic**.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

17.1. Pengertian Polymorphic Relationship

Kata **polymorphic** terasa cukup asing, tapi bagi yang pernah mendalami konsep object oriented programming atau OOP, tentu sudah cukup familiar dengan istilah ini. Secara bahasa, *polymorphic* atau *polymorphism* bisa diartikan sebagai "banyak bentuk".

Dalam konsep OOP, *polymorphic* merujuk ke beberapa object berbeda yang memiliki suatu kesamaan (misalnya memiliki nama method yang sama).

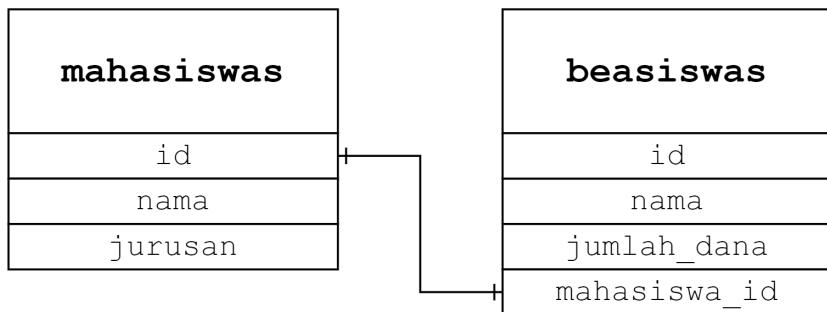
Sedangkan *polymorphic relationship* merujuk ke sebuah tabel yang bisa terhubung ke berbagai tabel berbeda (tidak hanya satu), dan tidak perlu kolom tambahan untuk setiap tabel yang terhubung tersebut.

Dari materi relationship yang sudah kita pelajari sejauh ini, satu tabel pada dasarnya hanya terhubung ke satu tabel lain, yang berbeda hanya di jumlah data saja. Misal dalam contoh *many to many*, tabel `mahasiswa` hanya terhubung ke satu tabel lain, yakni tabel `matakuliah`.

Namun pada *polymorphic relationship*, satu tabel bisa terhubung ke berbagai tabel lain. Mari kita bahas dengan contoh kasus.

Misalkan saat ini kita sedang membuat sistem informasi penerima beasiswa untuk sebuah kampus. Disini perlu tabel `mahasiswa` yang berisi daftar mahasiswa di kampus tersebut, serta tabel `beasiswa` yang berisi daftar beasiswa yang bisa diambil.

Berikut struktur tabel yang bisa digunakan:



Relationship antara tabel mahasiswa dan beasiswa

Keputusan pihak kampus adalah satu mahasiswa hanya bisa mengambil satu beasiswa, dan satu beasiswa hanya bisa diambil oleh satu mahasiswa saja. Ini berarti kita akan pakai relationship one to one, dimana kolom `id` dari tabel `mahasiswa` terhubung dengan kolom `mahasiswa_id` di tabel `beasiswa`.

Sesaat kemudian pihak kampus menghubungi lagi dan ingin beasiswa juga bisa diambil oleh dosen. Bagaimana membuat struktur tabel `beasiswa` dengan tambahan ini?

Solusi paling sederhana adalah membuat 2 buah tabel beasiswa, yakni `beasiswa_mahasiswa`, dan `beasiswa_dosen` agar data beasiswa tidak saling bentrok.

Alternatif lain tetap menggunakan satu tabel `beasiswa` tapi menambah kolom khusus yang berfungsi sebagai *flag* atau penanda jenis beasiswa (apakah itu beasiswa untuk dosen atau beasiswa untuk mahasiswa). Solusi inilah yang disebut sebagai **polymorphic relationship**.

Polymorphic relationship membuat pengelolaan data menjadi lebih efisien karena kita tidak butuh penambahan tabel baru, namun tabel `mahasiswa` perlu di modifikasi.

Pertama kita perlu menambah kolom "jenis" di tabel `beasiswa`. Kolom ini akan berisi informasi "mahasiswa" atau "dosen", yakni nama tabel pemilik beasiswa. Kedua, kolom `mahasiswa_id` juga harus diubah menjadi nama lain karena nantinya bisa berisi `id` mahasiswa dan juga `id` dosen.

Laravel memiliki aturan penamaan untuk kedua kolom tambahan ini:

- ◆ <nama_tabel_singular>"able" + "_id"
- ◆ <nama_tabel_singular>"able" + "_type"

Berikut penerapannya ke dalam tabel `beasiswa`:

- ◆ `beasiswaable_id`
- ◆ `beasiswaable_type`

Betul, nama yang sangat aneh karena kita menggunakan kata bahasa indonesia. Dalam bahasa inggris, tambahan kata "`able`" biasa dipakai untuk menambah makna "*dibolehkan*" atau "*sanggup*". Jadi `beasiswaable` bisa disebut sebagai "bisa mendapat beasiswa".

Pada prakteknya, nama kolom dengan tambahan 'able' ini bisa saja di ganti dengan nama lain, tapi dalam praktek ini saya ingin memakai penamaan sesuai dokumentasi Laravel.

Berikut struktur tabel dengan menerapkan polymorphic relationship:



Gambar: Struktur tabel dengan penerapan one to one polymorphic relationship

Dalam diagram ini, kolom `beasiswaable_id` dan `beasiswaable_type` sama-sama bertindak sebagai *foreign key* karena keduanya harus dipakai bersamaan untuk mencari siapa pemilik beasiswa tersebut.

17.2. Pengertian One to One Polymorphic Relationship

Berdasarkan hubungan antara tabel utama dengan tabel kedua, polymorphic relationship hadir dalam 3 jenis:

- ◆ **One to one polymorphic relationship**
- ◆ **One to many polymorphic relationship**
- ◆ **Many to many polymorphic relationship**

Dalam *one to one polymorphic relationship*, tabel utama hanya bisa memiliki satu data di tabel kedua. Begitu juga tabel kedua hanya bisa terhubung ke satu data di tabel utama.

Tabel utama ini bisa terdiri dari 2 atau banyak tabel. Pada contoh kita, tabel `dosen` dan tabel `mahasiswa` sama-sama bertindak sebagai tabel utama karena di sinilah *primary key* berada. Sedangkan tabel `beasiswa` bertindak sebagai tabel kedua sebagai tempat *foreign key*.

Penerapan *one to one polymorphic relationship* dalam contoh ini berarti setiap dosen atau mahasiswa hanya bisa mengambil 1 beasiswa. Dan 1 beasiswa juga hanya bisa diambil sekali saja.

Untuk *one to many polymorphic relationship*, tabel utama bisa memiliki banyak data di tabel kedua. Sebagai contoh, seorang mahasiswa bisa mengambil banyak beasiswa sekaligus, namun satu beasiswa tetap hanya terhubung ke satu mahasiswa saja.

Sedangkan untuk *many to many polymorphic relationship*, satu data di tabel kedua juga bisa dimiliki oleh banyak data di tabel utama. Dalam contoh beasiswa, ini artinya satu beasiswa bisa diambil oleh banyak mahasiswa, dan satu mahasiswa juga bisa mengambil banyak beasiswa.

Dalam bab ini kita akan bahas *one to one polymorphic relationship* terlebih dahulu.

17.3. Persiapan Awal

Sebagai bahan praktek untuk penerapan *one to one polymorphic relationship*, kita akan buat 3 buah tabel: `dosens`, `mahasiswas` dan `beasiswas`. Berikut struktur ketiga tabel tersebut:

- ◆ Tabel **dosen**: `id`, `nama`, `created_at` dan `updated_at`.
- ◆ Tabel **mahasiswas**: `id`, `nama`, `jurusan`, `created_at` dan `updated_at`.
- ◆ Tabel **beasiswas**: `id`, `nama`, `jumlah_dana`, `beasiswaable_id`, `beasiswaable_type`, `created_at` dan `updated_at`.

Generate Data Sample

Kita akan buat ketiga tabel melalui migration. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```
php artisan make:model Dosen -m  
php artisan make:model Mahasiswa -m  
php artisan make:model Beasiswa -m  
php artisan make:controller AplikasiController
```

Kode di atas akan membuat file **model** dan **migration** serta satu file controller tempat kita akan menulis kode program.

Setelah semua file tersedia, buka file migration untuk ketiga tabel dan isi dengan kode berikut:

```
database\migrations\<timestamp>_create_dosens_table.php
```

```
1 public function up()  
2 {  
3     Schema::create('dosens', function (Blueprint $table) {  
4         $table->id();  
5         $table->string('nama');  
6         $table->timestamps();  
7     });  
8 }
```

```
database\migrations\<timestamp>_create_mahasiswas_table.php
```

```
1 public function up()  
2 {  
3     Schema::create('mahasiswas', function (Blueprint $table) {  
4         $table->id();
```

Eloquent Relationship: One to One Polymorphic

```
5     $table->string('nama');
6     $table->string('jurusan');
7     $table->timestamps();
8 );
9 }
```

database\migrations\<timestamp>_create_beasiswas_table.php

```
1 public function up()
2 {
3     Schema::create('beasiswas', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->integer('jumlah_dana');
7         $table->morphs('beasiswaable');
8         $table->timestamps();
9
10        // Agar tidak ada beasiswa yang diambil berulang
11        $table->unique(['beasiswaable_id', 'beasiswaable_type']);
12    });
13 }
```

Kode migration untuk tabel dosens dan mahasiswa tidak ada masalah karena terdiri dari kolom `nama` dan `jurusan` yang biasa kita buat.

Untuk migration tabel `beasiswas`, terdapat satu perintah baru untuk pembuatan kolom `polymorphic`, yakni `$table->morphs('beasiswaable')` di baris 7. Inilah kode yang akan men-generate kolom `beasiswaable_id` dan `beasiswaable_type`. Apabila diinginkan, nama '`beasiswaable`' bisa diganti dengan nama lain.

Kemudian terdapat tambahan perintah `$table->unique(['beasiswaable_id', 'beasiswaable_type'])` di akhir file migration `beasiswas`. Ini dipakai untuk membuat `unique key` yang terdiri dari 2 kolom: `beasiswaable_id` dan `beasiswaable_type`. Tujuannya supaya ada pembatasan `one to one`, karena kita ingin satu beasiswa hanya bisa diambil oleh satu dosen atau satu mahasiswa saja.

Buat ketiga tabel dengan perintah `php artisan migrate`.

Lanjut, kita perlu beberapa data sample. Silahkan buka file `DatabaseSeeder.php` lalu modifikasi sebagai berikut:

database\seeders\DatabaseSeeder.php

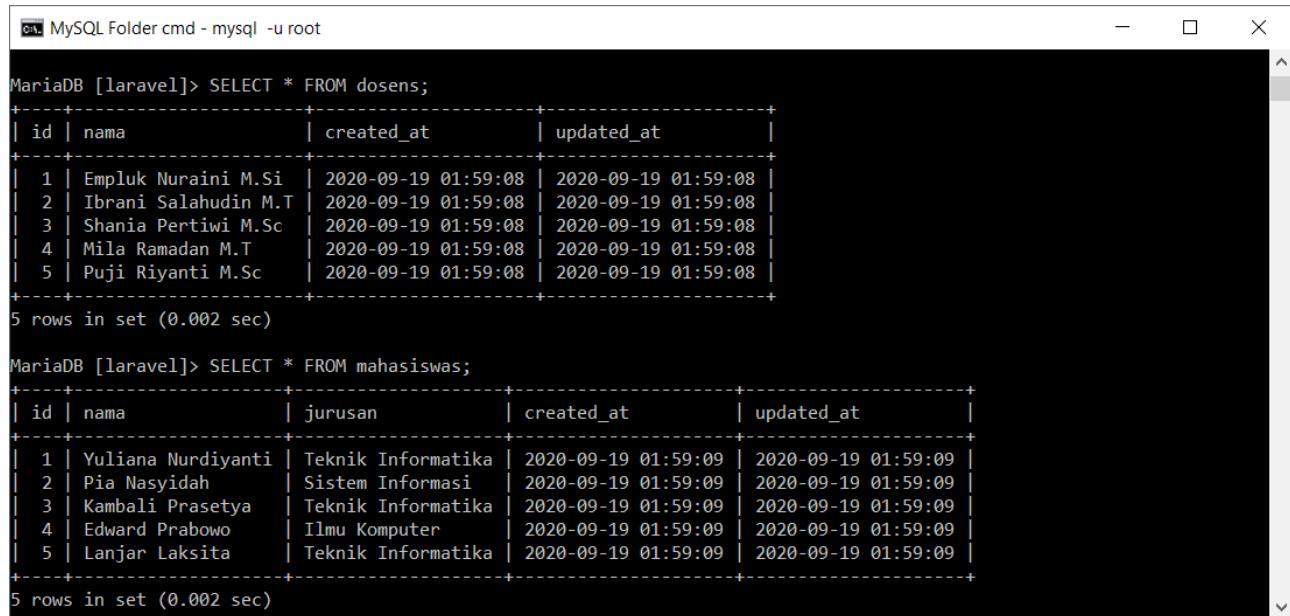
```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7
8 class DatabaseSeeder extends Seeder
```

Eloquent Relationship: One to One Polymorphic

```
9  {
10     public function run()
11     {
12         $faker = Faker::create('id_ID');
13         $faker->seed(123);
14
15         $daftar_titel = ["M.Kom", "M.Sc", "M.T", "M.Si"];
16         for ($i=0; $i<5; $i++) {
17             \App\Models\Dosen::create(
18                 [
19                     'nama' => $faker->firstName." ".$faker->lastName." ".
20                         $faker->randomElement($daftar_titel)
21                 ]
22             );
23         }
24
25         $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
26         for ($i=0; $i<5; $i++) {
27             \App\Models\Mahasiswa::create(
28                 [
29                     'nama' => $faker->firstName." ".$faker->lastName,
30                     'jurusan' => $faker->randomElement($jurusan),
31                 ]
32             );
33         }
34     }
35 }
```

File seeder ini akan men-generate 5 data dosen dan 5 data mahasiswa. Jalankan seeder dengan perintah `php artisan db:seed`.

Berikut hasil data yang sudah di generate:

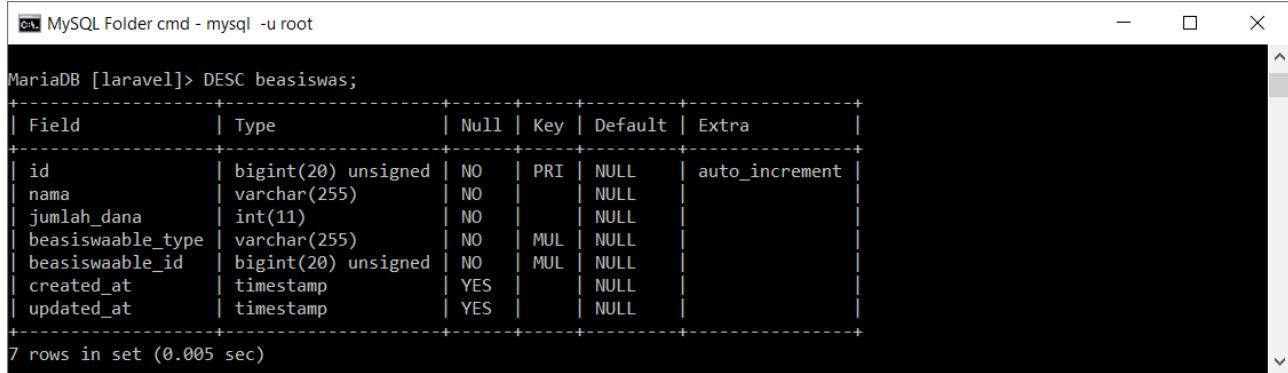


```
MariaDB [laravel]> SELECT * FROM dosens;
+---+-----+-----+-----+
| id | nama           | created_at    | updated_at   |
+---+-----+-----+-----+
| 1 | Empluk Nuraini M.Si | 2020-09-19 01:59:08 | 2020-09-19 01:59:08 |
| 2 | Ibrani Salahudin M.T | 2020-09-19 01:59:08 | 2020-09-19 01:59:08 |
| 3 | Shania Pertwi M.Sc | 2020-09-19 01:59:08 | 2020-09-19 01:59:08 |
| 4 | Mila Ramadan M.T | 2020-09-19 01:59:08 | 2020-09-19 01:59:08 |
| 5 | Puji Riyanti M.Sc | 2020-09-19 01:59:08 | 2020-09-19 01:59:08 |
+---+-----+-----+-----+
5 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM mahasiswa;
+---+-----+-----+-----+
| id | nama           | jurusan       | created_at    | updated_at   |
+---+-----+-----+-----+
| 1 | Yuliana Nurdyanti | Teknik Informatika | 2020-09-19 01:59:09 | 2020-09-19 01:59:09 |
| 2 | Pia Nasyidah | Sistem Informasi | 2020-09-19 01:59:09 | 2020-09-19 01:59:09 |
| 3 | Kambali Prasetya | Teknik Informatika | 2020-09-19 01:59:09 | 2020-09-19 01:59:09 |
| 4 | Edward Prabowo | Ilmu Komputer | 2020-09-19 01:59:09 | 2020-09-19 01:59:09 |
| 5 | Lanjar Laksita | Teknik Informatika | 2020-09-19 01:59:09 | 2020-09-19 01:59:09 |
+---+-----+-----+-----+
5 rows in set (0.002 sec)
```

Gambar: Isi tabel dosens dan mahasiswa

Untuk isi tabel `beasiswa`, akan kita tambah dari kode program sesaat lagi. Namun mari lihat struktur tabel yang digenerate oleh migration:



The screenshot shows a terminal window titled "MySQL Folder cmd - mysql -u root". The command "DESC beasiswa;" is run, and the resulting table structure is displayed. The columns are Field, Type, Null, Key, Default, and Extra. The table has 7 rows.

Field	Type	Null	Key	Default	Extra
<code>id</code>	<code>bigint(20) unsigned</code>	NO	PRI	<code>NULL</code>	<code>auto_increment</code>
<code>nama</code>	<code>varchar(255)</code>	NO		<code>NULL</code>	
<code>jumlah_dana</code>	<code>int(11)</code>	NO		<code>NULL</code>	
<code>beasiswaable_type</code>	<code>varchar(255)</code>	NO	MUL	<code>NULL</code>	
<code>beasiswaable_id</code>	<code>bigint(20) unsigned</code>	NO	MUL	<code>NULL</code>	
<code>created_at</code>	<code>timestamp</code>	YES		<code>NULL</code>	
<code>updated_at</code>	<code>timestamp</code>	YES		<code>NULL</code>	

7 rows in set (0.005 sec)

Gambar: Hasil perintah DESC beasiswa

Terlihat ada tambahan kolom `beasiswaable_type` dan `beasiswaable_id` yang berasal dari perintah `$table->morphs('beasiswa')` pada file migration. Selain itu kedua kolom ini memiliki key `MUL`, yakni key yang terdiri dari gabungan beberapa kolom (multiple key).

17.4. Eloquent Relationship morphOne() dan morphTo()

Dalam Laravel, *one to one polymorphic relationship* bisa dibuat dengan pasangan method `morphOne()` di model utama, serta method `morphTo()` di model kedua.

Silahkan buka file model Dosen, lalu tambah kode berikut:

app\Models\Dosen.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 Dosen extends Model
9 {
10     use HasFactory;
11     public function beasiswa()
12     {
13         return $this->morphOne('App\Models\Beasiswa', 'beasiswaable');
14     }
15 }
```

Di baris 11 terdapat method `beasiswa()` yang nantinya akan kita pakai saat membuat perintah eloquent di Controller. Nama method ini memakai kata *singular* tanpa tambahan 's' karena hubungan yang terjadi adalah one to one.

Pendefinisian relationship ditulis dengan method `morphOne()` seperti di baris 10. Method

Eloquent Relationship: One to One Polymorphic

`morphOne()` butuh 2 buah argument, yakni nama model yang akan dihubungkan, serta nama method penghubung.

Dalam contoh kita, model yang dihubungkan adalah '`App\Models\Beasiswa`', serta nama method penghubung adalah '`beasiswaable`'. Nama method penghubung ini nanti akan definisikan pada model Beasiswa.

Method yang sama juga perlu di tulis ke model Mahasiswa:

`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     public function beasiswa()
12     {
13         return $this->morphOne('App\Models\Beasiswa', 'beasiswaable');
14     }
15 }
```

Kode yang dipakai sama persis karena baik model Dosen maupun model Mahasiswa sama-sama bertindak sebagai tabel utama.

Dan berikut kode untuk model Beasiswa:

`app\Models\Beasiswa.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 Beasiswa extends Model
9 {
10     use HasFactory;
11     public function beasiswaable()
12     {
13         return $this->morphTo();
14     }
15 }
```

Method `beasiswaable()` berfungsi sebagai method penghubung. Isinya terdiri dari satu perintah `return $this->morphTo()` tanpa argument apapun.

Eloquent Relationship: One to One Polymorphic

Inilah cara mendefinisikan *one to one polymorphic relationship* di Laravel.

Sekarang saatnya masuk ke praktik penulisan perintah eloquent. Silahkan tambah beberapa route berikut:

routes\web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\ApplikasiController;
5
6 Route::get('/all', [ApplikasiController::class, 'all']);
7
8 Route::get('/input-beasiswa-1', [ApplikasiController::class, 'inputBeasiswa1']);
9 Route::get('/input-beasiswa-2', [ApplikasiController::class, 'inputBeasiswa2']);
10
11 Route::get('/tampil-beasiswa-1', [ApplikasiController::class, 'tampilBeasiswa1']);
12 Route::get('/tampil-beasiswa-2', [ApplikasiController::class, 'tampilBeasiswa2']);
13 Route::get('/tampil-beasiswa-3', [ApplikasiController::class, 'tampilBeasiswa3']);
14 Route::get('/tampil-beasiswa-4', [ApplikasiController::class, 'tampilBeasiswa4']);
15
16 Route::get('/wheremorph-1', [ApplikasiController::class, 'wheremorph1']);
17 Route::get('/wheremorph-2', [ApplikasiController::class, 'wheremorph2']);
18
19 Route::get('/update-beasiswa-1', [ApplikasiController::class, 'updateBeasiswa1']);
20 Route::get('/update-beasiswa-2', [ApplikasiController::class, 'updateBeasiswa2']);
21
22 Route::get('/delete', [ApplikasiController::class, 'delete']);
```

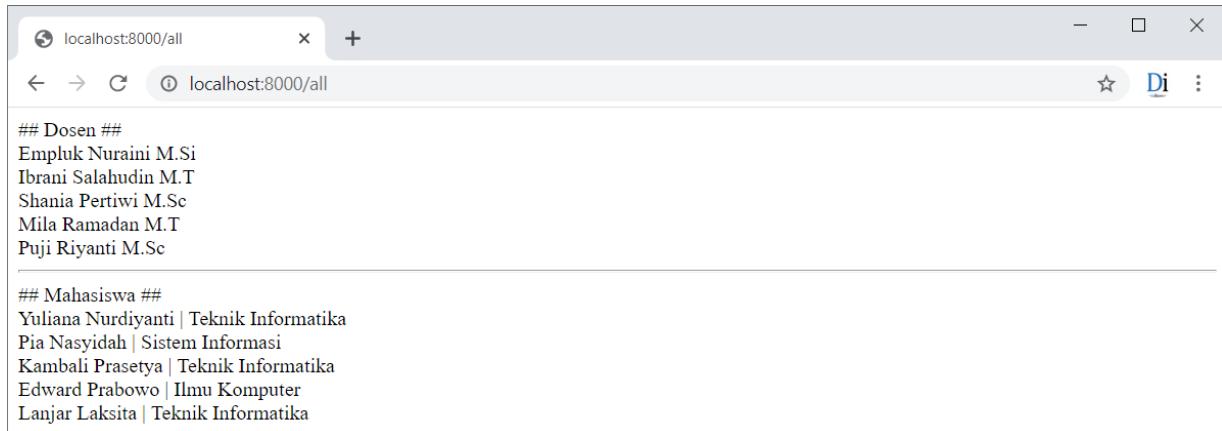
Kita akan bahas satu per satu. Seperti biasa, route pertama dipakai untuk menampilkan semua isi tabel yang ada saat ini:

app\Http\Controllers\ApplikasiController.php

```
1 <?php
2
3 namespace App\Http\Controllers;
4 use App\Models\Dosen;
5 use App\Models\Mahasiswa;
6 use App\Models\Beasiswa;
7 use Illuminate\Http\Request;
8
9 class ApplikasiController extends Controller
10 {
11     public function all()
12     {
13         echo "## Dosen ## <br>";
14         $dosen = Dosen::all();
15         foreach ($dosen as $dosen) {
16             echo "$dosen->nama <br>";
17         }
18         echo "<hr>";
19     }
20 }
```

Eloquent Relationship: One to One Polymorphic

```
20
21     echo "## Mahasiswa ## <br>";
22     $mahasiswas = Mahasiswa::all();
23     foreach ($mahasiswas as $mahasiswa) {
24         echo "$mahasiswa->nama | $mahasiswa->jurusan <br>";
25     }
26 }
27 }
```



Gambar: Menampilkan semua isi tabel

Sip, semua data sudah bisa diakses, kecuali tabel `beasiswa` yang akan kita input sesaat lagi.

Menginput Data Relationship

Saat ini tabel `beasiswa` masih dalam keadaan kosong. Saya akan coba input satu data beasiswa yang langsung terhubung ke satu dosen:

app\Http\Controllers\AplikasiController.php

```
1 public function inputBeasiswa1()
2 {
3     $dosen = Dosen::where('nama', 'Shania Pertwi M.Sc')->first();
4
5     $beasiswa = new Beasiswa;
6     $beasiswa->nama = "Beasiswa Unggulan Dosen Indonesia";
7     $beasiswa->jumlah_dana = 50000000;
8
9     $dosen->beasiswa()->save($beasiswa);
10    echo "$dosen->nama dapat beasiswa $beasiswa->nama";
11 }
```



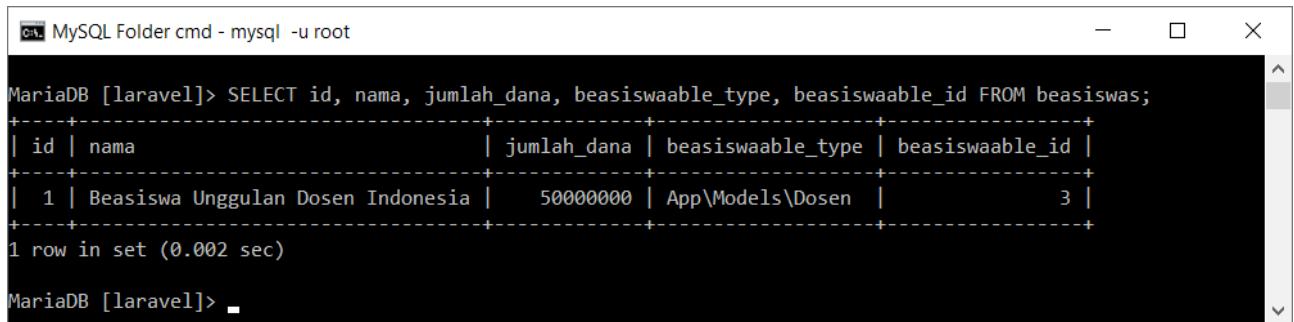
Gambar: Menambah Beasiswa Dosen

Eloquent Relationship: One to One Polymorphic

Di baris 3, variabel \$dosen diisi dengan object Dosen bernama "Shania Pertiwi M.Sc". Kemudian di baris 5 – 7 saya membuat object Beasiswa baru yang disimpan ke dalam variabel \$beasiswa.

Proses input data beasiswa dilakukan dengan perintah \$dosen->beasiswa()->save (\$beasiswa), yang tidak berbeda seperti pada relationship one to one biasa.

Jalankan kode di atas dengan mengakses alamat localhost:8000/input-beasiswa-1, setelah itu mari lihat apa data yang diinput Laravel ke tabel beasiswas:



The screenshot shows a MySQL command-line interface window titled 'MySQL Folder cmd - mysql -u root'. It displays the output of a SQL query:

```
MariaDB [laravel]> SELECT id, nama, jumlah_dana, beasiswaable_type, beasiswaable_id FROM beasiswas;
+----+-----+-----+-----+
| id | nama           | jumlah_dana | beasiswaable_type | beasiswaable_id |
+----+-----+-----+-----+
| 1  | Beasiswa Unggulan Dosen Indonesia | 50000000 | App\Models\Dosen |            3 |
+----+-----+-----+-----+
1 row in set (0.002 sec)

MariaDB [laravel]>
```

Gambar: Isi tabel beasiswas

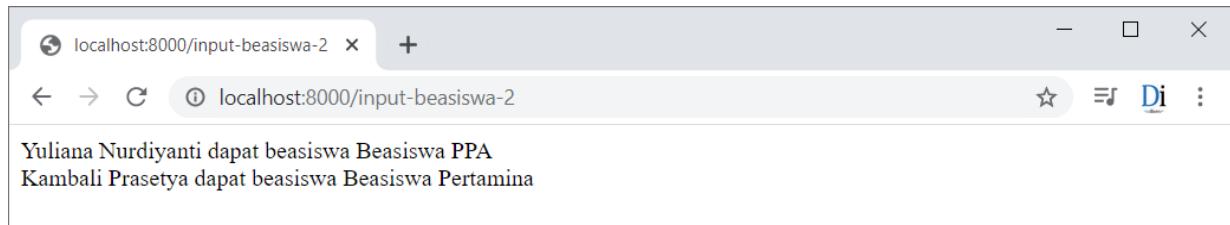
Perhatikan isi kolom beasiswaable_type, Laravel otomatis mengisinya dengan nama model yang terhubung ke beasiswa tersebut yakni 'App\Models\Dosen', sedangkan kolom beasiswaable_id berisi angka 3. Data ini juga bisa dibaca bahwa "Beasiswa Unggulan Dosen Indonesia" sudah terhubung ke dosen yang memiliki id = 3.

Sekarang kita coba input beasiswa untuk mahasiswa:

app\Http\Controllers\AplikasiController.php

```
1 public function inputBeasiswa2()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Yuliana Nurdiyanti')->first();
4
5     $beasiswa = new Beasiswa;
6     $beasiswa->nama = "Beasiswa PPA";
7     $beasiswa->jumlah_dana = 20000000;
8
9     $mahasiswa->beasiswa()->save($beasiswa);
10    echo "$mahasiswa->nama dapat beasiswa $beasiswa->nama <br>";
11
12    $mahasiswa = Mahasiswa::where('nama', 'Kambali Prasetya')->first();
13
14    $beasiswa = new Beasiswa;
15    $beasiswa->nama = "Beasiswa Pertamina";
16    $beasiswa->jumlah_dana = 33000000;
17
18    $mahasiswa->beasiswa()->save($beasiswa);
19    echo "$mahasiswa->nama dapat beasiswa $beasiswa->nama";
20 }
```

Eloquent Relationship: One to One Polymorphic



Gambar: Menambah Beasiswa Mahasiswa

Kali ini saya menginput 2 buah beasiswa ke 2 mahasiswa berbeda. "Beasiswa PPA" menjadi milik "Yuliana Nurdiyanti" sedangkan "Beasiswa Pertamina" kepunyaan "Kambali Prasetya".

Silahkan akses URL `localhost:8000/input-beasiswa-2` untuk menjalankan kode di atas dan cek kembali isi tabel `beasiswas`:

```
MySQL [laravel] > SELECT id, nama, jumlah_dana, beasiswaable_type, beasiswaable_id FROM beasiswas;
+----+-----+-----+-----+-----+
| id | nama           | jumlah_dana | beasiswaable_type | beasiswaable_id |
+----+-----+-----+-----+-----+
| 1  | Beasiswa Unggulan Dosen Indonesia | 50000000 | App\Models\Dosen | 3 |
| 2  | Beasiswa PPA | 20000000 | App\Models\Mahasiswa | 1 |
| 3  | Beasiswa Pertamina | 33000000 | App\Models\Mahasiswa | 3 |
+----+-----+-----+-----+-----+
3 rows in set (0.002 sec)
```

Gambar: Isi tabel beasiswas

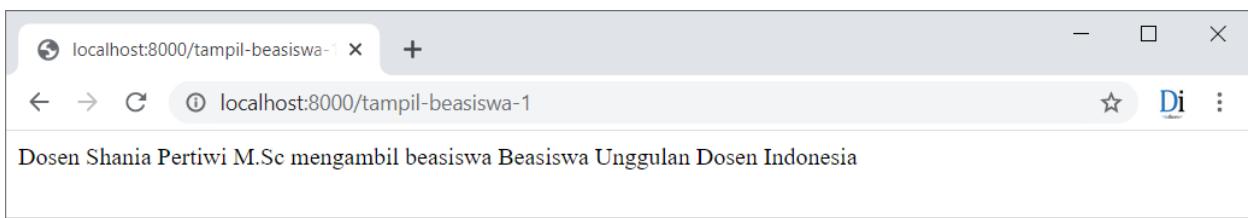
Sekarang kolom `beasiswaable_type` akan berisi "App\Models\Mahasiswa" untuk "Beasiswa PPA" dan "Beasiswa Pertamina". Inilah konsep dasar penyimpanan data pada *polymorphic relationship*.

Menampilkan Data Relationship

Jika berangkat dari tabel utama, perintah yang dipakai untuk menampilkan data *polymorphic* pada dasarnya sama seperti relationship biasa:

`app\Http\Controllers\AplikasiController.php`

```
1 public function tampilBeasiswa1()
2 {
3     $dosen = Dosen::where('nama', 'Shania Pertiwi M.Sc')->first();
4     echo "Dosen $dosen->nama mengambil beasiswa {$dosen->beasiswa->nama}";
5 }
```



Gambar: Menampilkan 1 dosen

Eloquent Relationship: One to One Polymorphic

Perintah `$dosen->beasiswa->nama` di baris 4 bisa dipakai untuk mengambil informasi dari nama beasiswa yang diambil oleh dosen "Shania Pertiwi M.Sc".

Contoh lain adalah menampilkan semua mahasiswa yang mengambil beasiswa:

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa2()
2 {
3     $mahasiswa = Mahasiswa::has('beasiswa')->get();
4     foreach ($mahasiswa as $mahasiswa) {
5         echo "$mahasiswa->nama | {$mahasiswa->beasiswa->nama} <br>";
6     }
7 }
```



Gambar: Menampilkan semua nama mahasiswa yang mengambil beasiswa

Perintah `Mahasiswa::has('beasiswa')->get()` di baris 3 dipakai untuk membatasi mahasiswa yang memiliki beasiswa saja. Perintah ini mengembalikan collection yang selanjutnya diproses dengan perulangan `foreach` pada baris 4 – 6.

Jika berangkat dari model Beasiswa, kita bisa mengakses property `beasiswaable` untuk mencari tau siapa pemilik beasiswa tersebut:

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa3()
2 {
3     $beasiswa = Beasiswa::find(2);
4     echo "Yang dapat $beasiswa->nama adalah {$beasiswa->beasiswaable->nama}";
5 }
```



Gambar: Menampilkan nama yang mengambil beasiswa

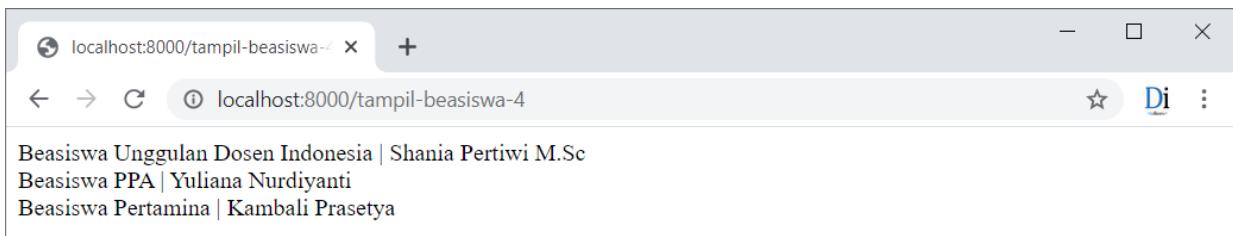
Di baris 3, variabel `$beasiswa` akan berisi object Beasiswa yang memiliki id = 2. Dari object ini, pemilik beasiswa bisa diakses dengan perintah `$beasiswa->beasiswaable->nama`.

Bagaimana jika ingin menampilkan semua beasiswa beserta nama pemiliknya? Bisa dengan kode berikut:

Eloquent Relationship: One to One Polymorphic

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa4()
2 {
3     $beasiswas = Beasiswa::all();
4     foreach ($beasiswas as $beasiswa) {
5         echo "$beasiswa->nama | {$beasiswa->beasiswaable->nama} <br>";
6     }
7 }
```



Gambar: Menampilkan semua penerima beasiswa

Ini merupakan variasi dari kode sebelumnya. Variabel \$beasiswas di baris 3 akan berisi semua object Beasiswa, kemudian perulangan foreach akan mengakses setiap object untuk menampilkan isi property \$beasiswa->nama serta \$beasiswa->beasiswaable->nama.

Menampilkan Batasan whereHasMorph()

Dalam praktek kita saat ini, sebuah beasiswa tidak bisa berdiri sendiri tapi harus punya "pemilik". Kolom beasiswaable_type dan beasiswaable_id tidak boleh kosong karena dalam file migration keduanya tidak di set sebagai nullable().

Untuk situasi dimana kolom beasiswaable_type dan beasiswaable_id boleh di kosongkan, perulangan foreach seperti di tampilBeasiswa4() akan error karena tidak semua beasiswa memiliki property \$beasiswa->beasiswaable->nama.

Sebagai solusi, Laravel menyediakan method whereHasMorph() untuk membatasi pengambilan data hanya untuk model tertentu saja. Berikut contoh penggunaan method ini:

app\Http\Controllers\AplikasiController.php

```
1 public function wherehasmorph1()
2 {
3     $beasiswas = Beasiswa::whereHasMorph('beasiswaable',
4                                         'App\Models\Mahasiswa')->get();
5
6     foreach ($beasiswas as $beasiswa) {
7         echo "$beasiswa->nama | {$beasiswa->beasiswaable->nama} <br>";
8     }
9 }
```

Eloquent Relationship: One to One Polymorphic



Gambar: Hasil dari batasan whereHasMorph('beasiswaable','App\Models\Mahasiswa')

Method `whereHasMorph()` butuh 2 buah argument. Argument pertama diisi dengan nama method relationship yang ada di dalam file Model, yakni '`beasiswaable`'. Sedangkan argument kedua diisi dengan nama model yang ingin dibatasi.

Perintah di baris 3 akan mencari semua beasiswa yang diambil oleh mahasiswa, atau bisa juga disebut bahwa variabel `$beasiswas` akan berisi collection dari semua object Beasiswa yang kolom `beasiswaable_type`-nya berisi 'App\Models\Mahasiswa'.

Jika kita ingin mencari semua beasiswa yang diambil oleh dosen, maka perintah di baris 3 bisa di ganti sebagai berikut:

```
$beasiswas = Beasiswa::whereHasMorph('beasiswaable','App\Models\Dosen')->get();
```

Bagaimana jika ingin menampilkan semua beasiswa yang diambil oleh mahasiswa **dan** dosen? Penulisan argument kedua dari `whereHasMorph()` bisa diganti menjadi array:

app\Http\Controllers\AplikasiController.php

```
1 public function wheremorph2()
2 {
3     $beasiswas = Beasiswa::whereHasMorph('beasiswaable',
4                                     ['App\Models\Mahasiswa', 'App\Models\Dosen'])->get();
5
6     foreach ($beasiswas as $beasiswa) {
7         echo "$beasiswa->nama | {$beasiswa->beasiswaable->nama} <br>";
8     }
9 }
```



Gambar: Menampilkan semua beasiswa untuk 'App\Models\Mahasiswa' dan 'App\Models\Dosen'

Dalam kode di atas, variabel `$beasiswas` akan berisi semua beasiswa yang terhubung ke 'App\Models\Mahasiswa' dan juga 'App\Models\Dosen'.

Jika kita ingin menampilkan data beasiswa untuk **semua** tipe model, argument kedua dari method juga bisa ditulis dengan karakter bintang '*' seperti kode berikut:

Eloquent Relationship: One to One Polymorphic

```
$beasiswa = Beasiswa::whereHasMorph('beasiswaable','*')->get();
```

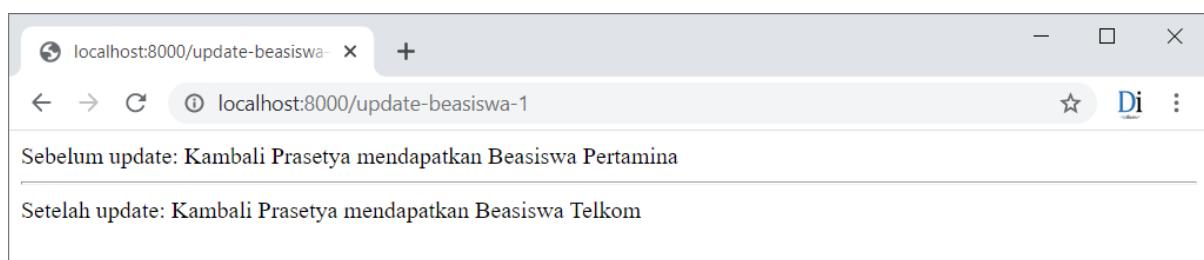
Perintah ini secara tidak langsung menunjukkan bahwa sebenarnya tabel utama yang bisa terhubung secara polymorphic tidak terbatas hanya 2 saja, bisa 3, 4 bahkan lebih. Sebagai contoh, bisa saja kita membuat tabel karyawans yang juga bisa menerima beasiswa. Dengan demikian, kolom `beasiswaable_type` akan berisi 'App\Karyawan'.

Mengupdate Data Relationship

Polymorphic relationship juga mengizinkan kita untuk mengupdate data di tabel yang terhubung. Sebagai contoh, dalam kode program berikut saya ingin menukar nama beasiswa yang diambil oleh mahasiswa "Kambali Prasetya":

app\Http\Controllers\AplikasiController.php

```
1 public function updateBeasiswa1()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Kambali Prasetya')->first();
4     echo "Sebelum update: $mahasiswa->nama mendapatkan
5         {$mahasiswa->beasiswa->nama}";
6
7     echo "<hr>";
8
9     $mahasiswa->beasiswa()->update([
10         'nama' => 'Beasiswa Telkom'
11     ]);
12
13    $mahasiswa = Mahasiswa::where('nama', 'Kambali Prasetya')->first();
14    echo "Setelah update: $mahasiswa->nama mendapatkan
15        {$mahasiswa->beasiswa->nama}";
16 }
```



Gambar: Update data beasiswa

Di awal program, variabel `$mahasiswa` akan berisi object Mahasiswa bernama 'Kambali Prasetya', kemudian saya mengupdate nama beasiswa yang terhubung menggunakan perintah `$mahasiswa->beasiswa()->update(..)`. Hasilnya, nama beasiswa berubah dari "Beasiswa Pertamina" menjadi 'Beasiswa Telkom'.

Cara alternatif bisa juga menggunakan method `push()`. Kode program di baris 9 – 11 juga bisa ditulis sebagai berikut:

```
1 $mahasiswa->beasiswa->nama = 'Beasiswa Pertamina';
```

Eloquent Relationship: One to One Polymorphic

```
2 $mahasiswa->push();
```

Contoh update lain adalah menukar id penerima beasiswa yang di proses dari object Beasiswa:

app\Http\Controllers\AplikasiController.php

```
1 public function updateBeasiswa2()
2 {
3     $beasiswa = Beasiswa::where('nama', 'Beasiswa PPA')->first();
4     echo "Sebelum update: $beasiswa->nama diambil oleh
5         {$beasiswa->beasiswaable->nama}";
6
7     echo "<hr>";
8
9     $mahasiswa = Mahasiswa::where('nama', 'Pia Nasyidah')->first();
10    $beasiswa->beasiswaable_id = $mahasiswa->id;
11    $beasiswa->push();
12
13    $beasiswa = Beasiswa::where('nama', 'Beasiswa PPA')->first();
14    echo "Setelah update: $beasiswa->nama diambil oleh
15        {$beasiswa->beasiswaable->nama}";
16 }
```



Gambar: Update data beasiswa

Kode program di baris 3 akan mengisi variabel \$beasiswa dengan object "Beasiswa PPA". Inilah beasiswa yang id penerimanya akan coba kita tukar. Lalu di baris 9 giliran variabel \$mahasiswa yang di isi object Mahasiswa bernama "Pia Nasyidah".

Proses update dilakukan dengan cara mengisi property \$beasiswa->beasiswaable_id dengan nilai \$mahasiswa->id, kemudian disambung perintah \$beasiswa->push() agar perubahan ini sampai ke database.

Hasilnya, penerima beasiswa PPA akan berubah dari "Yuliana Nurdiyanti" ke "Pia Nasyidah", yakni sesuai dengan nilai id yang tersimpan di kolom beasiswaable_id.

Dari dua contoh proses update ini, terlihat bahwa kita menukar manual nilai kolom di tabel beasiswas. Saat ini Laravel belum menyediakan method khusus seperti sync() atau associate() untuk proses update data *polymorphic relationship*.

Menghapus Data Relationship

Untuk menghapus data beasiswa, kita bisa menggunakan method delete(). Namun karena di

Eloquent Relationship: One to One Polymorphic

tabel mahasiswa tidak terdapat *foreign key*, bisa saja beasiswa tersebut masih terhubung ke dosen atau mahasiswa yang sudah tidak ada.

Agar terjadi konsistensi data, maka ketika data dosen atau mahasiswa dihapus, isi tabel beasiswas juga harus dihapus secara manual:

app\Http\Controllers\AplikasiController.php

```
1 public function delete()
2 {
3     $dosen = Dosen::where('nama', 'Shania Pertiwi M.Sc')->first();
4     $dosen->delete();
5     $dosen->beasiswa()->delete();
6
7     echo "Beasiswa $dosen->nama sudah dihapus";
8 }
```



Gambar: Menghapus data dosen dan juga data beasiswa

Perintah `delete()` di baris 4 akan menghapus data dosen, sedangkan perintah `delete()` di baris 5 akan menghapus data beasiswa untuk dosen tersebut.

Dalam bab ini kita sudah membahas dan *one to one polymorphic relationship*. Berikutnya akan dibahas variasi kedua dari polymorphic relationship, yakni **one to many polymorphic relationship**.

18. Eloquent Relationship: One to Many Polymorphic

Polymorphic relationship one to many merupakan variasi selanjutnya dari polymorphic relationship. Dalam bab ini kita akan bahas apa saja perbedaannya dengan *one to one polymorphic relationship*.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database laravel masih terdapat tabel, silahkan hapus terlebih dahulu.

18.1. Pengertian One to Many Polymorphic Relationship

Polymorphic relationship one to many adalah penerapan relationship *one to many* ke dalam *polymorphic relationship*. Kali ini satu data di tabel utama bisa memiliki beberapa data di tabel kedua.

Agar pembahasan kita lebih cepat dan mudah dipahami, saya akan kembali memakai ilustrasi sistem informasi beasiswa. Dalam *polymorphic relationship one to many*, satu mahasiswa atau satu dosen bisa mengambil lebih dari satu beasiswa sekaligus. Akan tetapi satu beasiswa tetap hanya bisa terhubung ke satu mahasiswa atau satu dosen saja.

Berikut diagram hubungan antar tabel:



Gambar: Struktur tabel dengan penerapan one to many polymorphic relationship

18.2. Generate Data Sample

Sebagai bahan praktek untuk penerapan *one to many polymorphic relationship*, kita akan buat 3 buah tabel: dosens, mahasiswa dan beasiswa. Berikut struktur ketiga tabel tersebut:

- ◆ Tabel **dosen**: id, nama, created_at dan updated_at.
- ◆ Tabel **mahasiswa**: id, nama, jurusan, created_at dan updated_at.
- ◆ Tabel **beasiswa**: id, nama, jumlah_dana, beasiswaable_id, beasiswaable_type, created_at dan updated_at.

Ketiga tabel akan di-generate menggunakan migration. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```
php artisan make:model Dosen -m  
php artisan make:model Mahasiswa -m  
php artisan make:model Beasiswa -m  
php artisan make:controller AplikasiController
```

Kode di atas akan membuat file **model** dan **migration** serta satu file **AplikasiController**.

Setelah semua file tersedia, buka file migration dan isi dengan kode berikut:

```
database\migrations\<timestamp>_create_dosens_table.php
```

```
1 public function up()  
2 {  
3     Schema::create('dosens', function (Blueprint $table) {  
4         $table->id();  
5         $table->string('nama');  
6         $table->timestamps();  
7     });  
8 }
```

```
database\migrations\<timestamp>_create_mahasiswa_table.php
```

```
1 public function up()  
2 {  
3     Schema::create('mahasiswa', function (Blueprint $table) {  
4         $table->id();  
5         $table->string('nama');  
6         $table->string('jurusan');  
7         $table->timestamps();  
8     });  
9 }
```

```
database\migrations\<timestamp>_create_beasiswa_table.php
```

```
1 public function up()  
2 {  
3     Schema::create('beasiswa', function (Blueprint $table) {  
4         $table->id();  
5     });  
6 }
```

Eloquent Relationship: One to Many Polymorphic

```
5     $table->string('nama');
6     $table->integer('jumlah_dana');
7     $table->morphs('beasiswaable');
8     $table->timestamps();
9 }
10 }
```

Struktur tabel ini sama persis seperti pada praktik *one to one polymorphic relationship*. Hanya saja sekarang tidak ada pembatasan *multiple key* untuk kolom `beasiswaable_type` dan `beasiswaable_id`. Tujuannya supaya kedua kolom boleh berisi data yang sama beberapa kali.

Sebagai contoh, jika seorang mahasiswa dengan id = 5 menerima Beasiswa Telkom, maka kolom `beasiswaable_id` di tabel `beasiswas` akan berisi angka 5.

Dalam *one to many relationship*, mahasiswa yang sama bisa saja menerima Beasiswa PPA, sehingga akan kembali dicatat sebagai angka 5 di kolom `beasiswaable_id`. Dalam *polymorphic one to one* sebelumnya proses input seperti ini tidak bisa dilakukan karena ada batasan *multiple key*.

Kembali ke migration, silahkan buat ketiga tabel dengan perintah `php artisan migrate`.

Lanjut, kita perlu beberapa data sample yang juga masih sama seperti praktik *one to one* sebelumnya. Buka file `DatabaseSeeder.php` lalu modifikasi sebagai berikut:

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7
8 class DatabaseSeeder extends Seeder
9 {
10     public function run()
11     {
12         $faker = Faker::create('id_ID');
13         $faker->seed(123);
14
15         $daftar_titel = ["M.Kom", "M.Sc", "M.T", "M.Si"];
16         for ($i=0; $i<5; $i++) {
17             \App\Models\Dosen::create(
18                 [
19                     'nama' => $faker->firstName." ".$faker->lastName." ".
20                               $faker->randomElement($daftar_titel)
21                 ]
22             );
23         }
24
25         $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
26         for ($i=0; $i<5; $i++) {
```

Eloquent Relationship: One to Many Polymorphic

```
27     \App\Models\Mahasiswa::create(
28     [
29         'nama' => $faker->firstName . " " . $faker->lastName,
30         'jurusan' => $faker->randomElement($jurusan),
31     ]
32     );
33 }
34 }
35 }
```

File seeder ini akan men-generate 5 data dosen dan 5 data mahasiswa. Jalankan seeder dengan perintah `php artisan db:seed`.

18.3. Eloquent Relationship `morphMany()` dan `morphTo()`

Untuk *one to many polymorphic relationship* kita butuh pasangan method `morphMany()` di model utama serta method `morphTo()` di model kedua.

Silahkan buka file model Dosen, lalu tambah kode berikut:

app\Models\Dosen.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 Dosen extends Model
9 {
10     use HasFactory;
11     public function beasiswas()
12     {
13         return $this->morphMany('App\Models\Beasiswa', 'beasiswaable');
14     }
15 }
```

Kali ini nama method penghubung adalah `beasiswas()`, menggunakan kata plural dengan akhiran 's' karena satu dosen nantinya bisa memiliki banyak beasiswa.

Untuk method `morphMany()` tetap butuh 2 argument yang sama seperti method `morphOne()`, yakni nama model yang akan dihubungkan sebagai argument pertama, serta nama method penghubung sebagai argument kedua.

Method yang sama juga perlu di tulis ke model Mahasiswa:

app\Models\Mahasiswa.php

```
1 <?php
2
```

Eloquent Relationship: One to Many Polymorphic

```
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     public function beasiswas()
12     {
13         return $this->morphMany('App\Models\Beasiswa', 'beasiswaable');
14     }
15 }
```

Dan berikut kode untuk model Beasiswa:

app\Models\Beasiswa.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 Beasiswa extends Model
9 {
10     use HasFactory;
11     protected $guarded = [];
12
13     public function beasiswaable()
14     {
15         return $this->morphTo();
16     }
17 }
```

Kode program di model Beasiswa masih sama seperti praktek *one to one relationship*, yakni menggunakan method penghubung `beasiswaable()` yang berisi perintah `return $this->morphTo()`.

Tambahan `protected $guarded = []` di baris 11 diperlukan karena kita akan memakai *mass assignment* untuk proses input data.

Sekarang saatnya masuk ke praktek penulisan perintah eloquent. Silahkan tambah beberapa route berikut:

routes\web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\ApplikasiController;
5
```

```
6 Route::get('/input-beasiswa-1', [ApplikasiController::class, 'inputBeasiswa1']);  
7 Route::get('/input-beasiswa-2', [ApplikasiController::class, 'inputBeasiswa2']);  
8  
9 Route::get('/tampil-beasiswa-1',[ApplikasiController::class,'tampilBeasiswa1']);  
10 Route::get('/tampil-beasiswa-2',[ApplikasiController::class,'tampilBeasiswa2']);  
11 Route::get('/tampil-beasiswa-3',[ApplikasiController::class,'tampilBeasiswa3']);  
12 Route::get('/tampil-beasiswa-4',[ApplikasiController::class,'tampilBeasiswa4']);  
13  
14 Route::get('/wherehasmorph', [ApplikasiController::class, 'wherehasmorph']);  
15  
16 Route::get('/update-beasiswa', [ApplikasiController::class, 'updateBeasiswa']);  
17  
18 Route::get('/delete', [ApplikasiController::class, 'delete']);
```

Menginput Data Relationship

Saat ini tabel beasiswas masih dalam keadaan kosong. Pada kode program berikut saya akan coba input dua data beasiswa yang terhubung ke satu dosen:

app\Http\Controllers\ApplikasiController.php

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4 use App\Models\Dosen;  
5 use App\Models\Mahasiswa;  
6 use App\Models\Beasiswa;  
7 use Illuminate\Http\Request;  
8  
9 class ApplikasiController extends Controller  
10 {  
11     public function inputBeasiswa1()  
12     {  
13         $dosen = Dosen::where('nama', 'Puji Riyanti M.Sc')->first();  
14         $dosen->beasiswas()->createMany([  
15             [  
16                 'nama' => "Beasiswa Unggulan Dosen Indonesia",  
17                 'jumlah_dana' => 50000000,  
18             ],  
19             [  
20                 'nama' => "Beasiswa Pendidikan Pascasarjana Dalam Negeri",  
21                 'jumlah_dana' => 100000000,  
22             ]  
23         ]);  
24  
25         echo "$dosen->nama sudah mendapat beasiswa";  
26     }  
27 }
```



Gambar: Menambah Beasiswa Dosen

Di baris 13, variabel \$dosen diisi dengan object Dosen bernama "Puji Riyanti M.Sc". Kemudian di baris 14 – 23 saya mengisi 2 buah beasiswa baru menggunakan teknik *mass assignment*. Karena diakses dengan perintah \$dosen->beasiswas()->createMany([...]), maka data beasiswa sudah langsung terhubung ke "Puji Riyanti M.S".

Jalankan kode di atas dengan mengakses alamat `localhost:8000/input-beasiswa-1`, dan mari periksa isi tabel `beasiswas`:

Select MySQL Folder cmd - mysql -u root					
MariaDB [laravel]>	id	nama	jumlah_dana	beasiswaable_type	beasiswaable_id
+-----+-----+-----+-----+-----+	1 Beasiswa Unggulan Dosen Indonesia 50000000 App\Models\Dosen 5	+-----+-----+-----+-----+-----+			
+-----+-----+-----+-----+-----+	2 Beasiswa Pendidikan Pascasarjana Dalam Negeri 100000000 App\Models\Dosen 5	+-----+-----+-----+-----+-----+			
2 rows in set (0.002 sec)					

Gambar: Isi tabel beasiswas

Terlihat kedua beasiswa yang di input sudah terhubung ke 'App\Models\Dosen' dengan id 5.

Sekarang kita coba input beasiswa untuk mahasiswa:

app\Http\Controllers\AplikasiController.php

```
1 public function inputBeasiswa2()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Pia Nasyidah')->first();
4     $mahasiswa->beasiswas()->createMany([
5         [
6             'nama' => "Beasiswa PPA",
7             'jumlah_dana' => 20000000,
8         ],
9         [
10            'nama' => "Beasiswa Pertamina",
11            'jumlah_dana' => 33000000,
12        ],
13        [
14            'nama' => "Beasiswa LPDP",
15            'jumlah_dana' => 50000000,
16        ]
17    ]);
18
19     echo "$mahasiswa->nama sudah mendapat beasiswa <br>";
20 }
```

Eloquent Relationship: One to Many Polymorphic

```
21 $mahasiswa = Mahasiswa::where('nama', 'Edward Prabowo')->first();
22 $mahasiswa->beasiswas()->createMany([
23     [
24         'nama' => "Beasiswa Telkom",
25         'jumlah_dana' => 44000000,
26     ],
27     [
28         'nama' => "Beasiswa Unggulan Kemendikbud",
29         'jumlah_dana' => 50000000,
30     ]
31 ]);
32
33 echo "$mahasiswa->nama sudah mendapat beasiswa";
34 }
```



Gambar: Menambah Beasiswa Mahasiswa

Kali ini saya menginput 3 buah beasiswa ke "Pia Nasyidah" dan 2 beasiswa ke "Edward Prabowo".

Silahkan akses URL `localhost:8000/input-beasiswa-2` untuk menjalankan kode di atas dan cek kembali isi tabel `beasiswas`:

MariaDB [laravel]> SELECT id, nama, jumlah_dana, beasiswaable_type, beasiswaable_id FROM beasiswas;
+-----+-----+-----+-----+
id nama jumlah_dana beasiswaable_type beasiswaable_id
+-----+-----+-----+-----+
1 Beasiswa Unggulan Dosen Indonesia 50000000 App\Models\Dosen 5
2 Beasiswa Pendidikan Pascasarjana Dalam Negeri 100000000 App\Models\Dosen 5
3 Beasiswa PPA 20000000 App\Models\Mahasiswa 2
4 Beasiswa Pertamina 33000000 App\Models\Mahasiswa 2
5 Beasiswa LPDP 50000000 App\Models\Mahasiswa 2
6 Beasiswa Telkom 44000000 App\Models\Mahasiswa 4
7 Beasiswa Unggulan Kemendikbud 50000000 App\Models\Mahasiswa 4
+-----+-----+-----+-----+
7 rows in set (0.002 sec)

Gambar: Isi tabel beasiswas

Total terdapat tambahan 5 beasiswa baru. Untuk data yang baru saja kita input, kolom `beasiswaable_type` akan berisi "`App\Models\Mahasiswa`", sedangkan kolom `beasiswaable_id` berisi id dari mahasiswa yang terhubung.

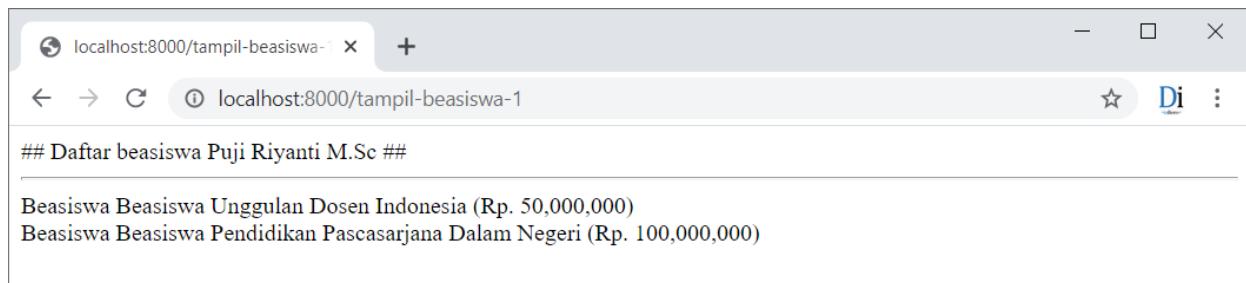
Menampilkan Data Relationship

Jika berangkat dari tabel utama, perintah yang dipakai untuk menampilkan data polymorphic *one to many* pada dasarnya sama seperti *relationship one to many* biasa:

Eloquent Relationship: One to Many Polymorphic

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa1()
2 {
3     $dosen = Dosen::where('nama', 'Puji Riyanti M.Sc')->first();
4
5     echo "## Daftar beasiswa $dosen->nama ##";
6     echo "<hr>";
7     foreach ($dosen->beasiswas as $beasiswa)
8     {
9         echo "Beasiswa $beasiswa->nama
10            (Rp. ".number_format($beasiswa->jumlah_dana).")<br>";
11     }
12 }
```



Gambar: Menampilkan semua beasiswa dari 1 dosen

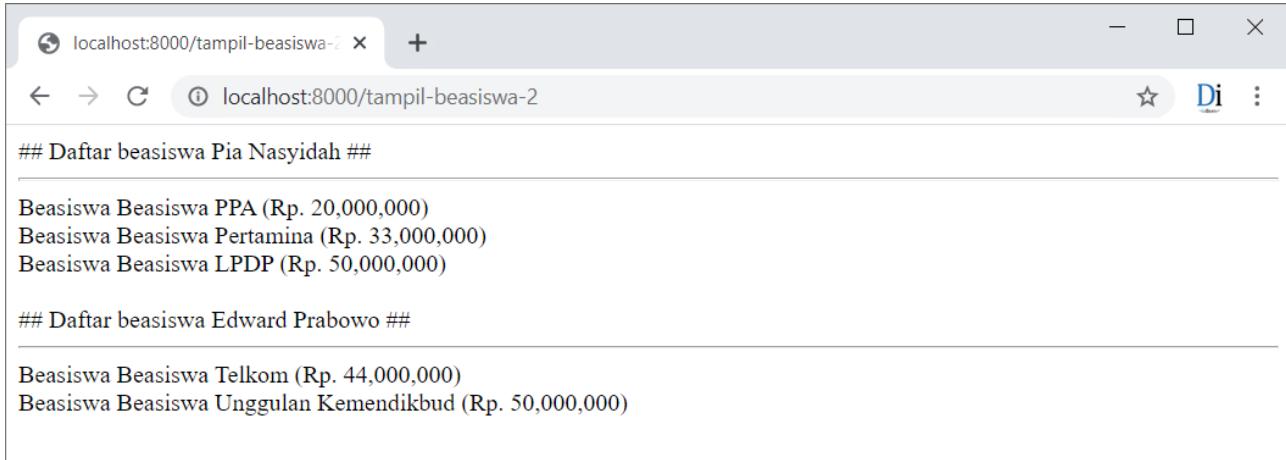
Seorang dosen bisa memiliki banyak beasiswa, sehingga perlu perulangan foreach untuk membuka hasil collection. Dalam contoh di atas, collection beasiswa ini langsung saya akses di dalam perulangan foreach, yakni dengan perintah `foreach ($dosen->beasiswas as $beasiswa)`.

Contoh lain adalah menampilkan semua mahasiswa yang mengambil beasiswa:

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa2()
2 {
3     $mahasiswas = Mahasiswa::has('beasiswas')->get();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "## Daftar beasiswa $mahasiswa->nama ##";
7         echo "<hr>";
8         foreach ($mahasiswa->beasiswas as $beasiswa)
9         {
10             echo "Beasiswa $beasiswa->nama
11                (Rp. ".number_format($beasiswa->jumlah_dana).")<br>";
12         }
13         echo "<br>";
14     }
15 }
```

Eloquent Relationship: One to Many Polymorphic



The screenshot shows a browser window with the URL `localhost:8000/tampil-beasiswa-2`. The page displays two sections of scholarship information:

- ## Daftar beasiswa Pia Nasyidah ##**
 - Beasiswa Beasiswa PPA (Rp. 20,000,000)
 - Beasiswa Beasiswa Pertamina (Rp. 33,000,000)
 - Beasiswa Beasiswa LPDP (Rp. 50,000,000)
- ## Daftar beasiswa Edward Prabowo ##**
 - Beasiswa Beasiswa Telkom (Rp. 44,000,000)
 - Beasiswa Beasiswa Unggulan Kemendikbud (Rp. 50,000,000)

Gambar: Menampilkan semua mahasiswa yang mengambil beasiswa

Pertama, kita perlu mencari siapa saja mahasiswa yang memiliki beasiswa. Ini didapat dari perintah `Mahasiswa::has('beasiswas')->get()` di baris 3. Hasilnya berupa collection dari object Mahasiswa yang kemudian disimpan ke dalam variabel `$mahasiswas`.

Karena variabel `$mahasiswas` berisi collection, perulangan `foreach ($mahasiswas as $mahasiswa)` dipakai untuk mengakses setiap mahasiswa.

Tugas kita selanjutnya adalah menampilkan daftar beasiswa dari masing-masing mahasiswa. Ini bisa diakses dari property `$mahasiswa->beasiswas`, namun karena ini mengembalikan collection, perlu perulangan foreach kembali di baris 8 – 12.

Jika berangkat dari model Beasiswa, property `beasiswaable` bisa diakses untuk mencari tau siapa pemilik beasiswa tersebut:

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa3()
2 {
3     $beasiswa = Beasiswa::where('nama', 'Beasiswa Telkom')->first();
4     echo "Yang dapat $beasiswa->nama adalah {$beasiswa->beasiswaable->nama}";
5 }
```



The screenshot shows a browser window with the URL `localhost:8000/tampil-beasiswa-3`. The page displays the following output:

Yang dapat Beasiswa Telkom adalah Edward Prabowo

Gambar: Menampilkan nama yang mengambil beasiswa

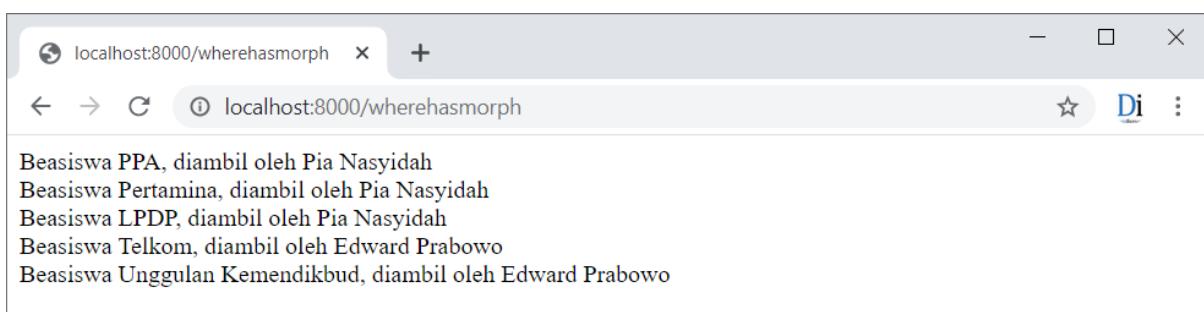
Di baris 3, variabel `$beasiswa` akan berisi object Beasiswa yang memiliki nama "Beasiswa Telkom". Dari object ini, pemilik beasiswa bisa diakses dengan perintah `$beasiswa->beasiswaable->nama`.

Menampilkan Batasan whereHasMorph()

Sama seperti di *relationship polymorphic one to one*, kita juga bisa menggunakan method `whereHasMorph()` untuk men-filter model yang terhubung ke tabel `beasiswa`:

app\Http\Controllers\AplikasiController.php

```
1 public function wherehasmorph()
2 {
3     $beasiswas = Beasiswa::whereHasMorph('beasiswaable',
4                                         'App\Models\Mahasiswa')->get();
5     foreach ($beasiswas as $beasiswa) {
6         echo "$beasiswa->nama, diambil oleh {$beasiswa->beasiswaable->nama} <br>";
7     }
8 }
```



Gambar: Hasil dari batasan `whereHasMorph('beasiswaable','App\Models\Mahasiswa')`

Perintah `Beasiswa::whereHasMorph('beasiswaable','App\Models\Mahasiswa')->get()` di baris 3 akan mencari semua beasiswa yang diambil oleh mahasiswa, lalu di baris 5 – 7 perulangan `foreach` dipakai untuk menampilkan data nama beasiswa serta pemilik dari beasiswa tersebut.

Jika kita ingin mencari semua beasiswa yang diambil oleh dosen, maka perintah di baris 3 bisa di ganti sebagai berikut:

```
$beasiswas = Beasiswa::whereHasMorph('beasiswaable','App\Models\Dosen')->get();
```

Bagaimana jika ingin menampilkan semua beasiswa yang diambil oleh mahasiswa **dan** dosen? Penulisan argument kedua dari `whereHasMorph()` bisa diganti menjadi array `['App\Models\Mahasiswa', 'App\Models\Dosen']` yang contohnya akan sama persis seperti di *polymorphic one to one*.

Mengupdate Data Relationship

Proses update data di relationship one to many juga bisa dilakukan dengan method `update()` atau `push()`. Dalam contoh berikut saya ingin menukar semua beasiswa milik satu mahasiswa ke mahasiswa lain:

Eloquent Relationship: One to Many Polymorphic

app\Http\Controllers\AplikasiController.php

```
1 public function updateBeasiswa()
2 {
3     $mahasiswa1 = Mahasiswa::where('nama', 'Pia Nasyidah')->first();
4     $beasiswas1 = $mahasiswa1->beasiswas->pluck('nama')->toArray();
5
6     $mahasiswa2 = Mahasiswa::where('nama', 'Lanjar Laksita')->first();
7     $beasiswas2 = $mahasiswa2->beasiswas->pluck('nama')->toArray();
8
9     echo "Sebelum Update <hr>";
10    echo "Beasiswa $mahasiswa1->nama: ".implode(", ", $beasiswas1);
11    echo "<br>";
12    echo "Beasiswa $mahasiswa2->nama: ".implode(", ", $beasiswas2);
13
14    echo "<br><br><br>";
15
16    $mahasiswa1->beasiswas()->update([
17        'beasiswaable_id' => $mahasiswa2->id
18    ]);
19
20    $mahasiswa1 = Mahasiswa::where('nama', 'Pia Nasyidah')->first();
21    $beasiswas1 = $mahasiswa1->beasiswas->pluck('nama')->toArray();
22
23    $mahasiswa2 = Mahasiswa::where('nama', 'Lanjar Laksita')->first();
24    $beasiswas2 = $mahasiswa2->beasiswas->pluck('nama')->toArray();
25
26    echo "Setelah Update <hr>";
27    echo "Beasiswa $mahasiswa1->nama: ".implode(", ", $beasiswas1);
28    echo "<br>";
29    echo "Beasiswa $mahasiswa2->nama: ".implode(", ", $beasiswas2);
30 }
```



Gambar: Update data beasiswa

Di awal program, variabel `$mahasiswa1` akan berisi object Mahasiswa bernama 'Pia Nasyidah', Kemudian di baris 2 saya mengisi variabel `$beasiswas1` dengan perintah `$mahasiswa1->beasiswas->pluck('nama')->toArray()`. Perintah ini dipakai untuk mengambil semua nama beasiswa yang dimiliki "Pia Nasyidah".

Property `$mahasiswa1->beasiswas` berisi collection dari semua object beasiswa yang terhubung ke `$mahasiswa1`, lalu method `pluck('nama')` dipakai untuk mengambil property `nama` dari setiap object beasiswa. Terakhir, method `toArray()` akan mengkonversi collection tersebut menjadi array.

Jika ingin memahami lebih dalam cara kerja perintah ini, jalankan `dump()` ke setiap method untuk melihat apa hasil akhir yang didapat. Ketika membuat perintah tersebut, saya juga memakai metode seperti ini untuk memastikan hasilnya sesuai dengan keinginan:

```
dump($mahasiswa1);
dump($mahasiswa1->beasiswas);
dump($mahasiswa1->beasiswas->pluck('nama'));
dump($mahasiswa1->beasiswas->pluck('nama')->toArray());
```

Hasil akhir dari perintah ini, variabel `$beasiswas1` di baris 4 akan berisi array berikut:

```
["Beasiswa PPA", "Beasiswa Pertamina", "Beasiswa LPDP"]
```

Perintah yang sama juga di pakai pada baris 6 dan 7, tapi kali ini variabel `$mahasiswa2` akan berisi object Mahasiswa 'Lanjar Laksita'.

Kenapa harus repot-repot membuat array nama beasiswa? Karena array ini akan saya input ke dalam function `implode()` di baris 10 dan 12. Dengan trik ini, kita tidak butuh perulangan `foreach` untuk menampilkan seluruh nama beasiswa.

Proses update sendiri hanya butuh 1 perintah di baris 16 – 18. Di sini saya mengupdate kolom `beasiswaable_id` untuk semua beasiswa milik `$mahasiswa1` dengan nilai `$mahasiswa2->id`. Artinya, semua beasiswa yang dimiliki oleh 'Pia Nasyidah' akan berganti ke 'Lanjar Laksita'.

Kode di baris 20 – 29 sama persis seperti kode di baris 3 – 12, yakni sekedar menampilkan daftar beasiswa untuk kedua mahasiswa.

Menghapus Data Relationship

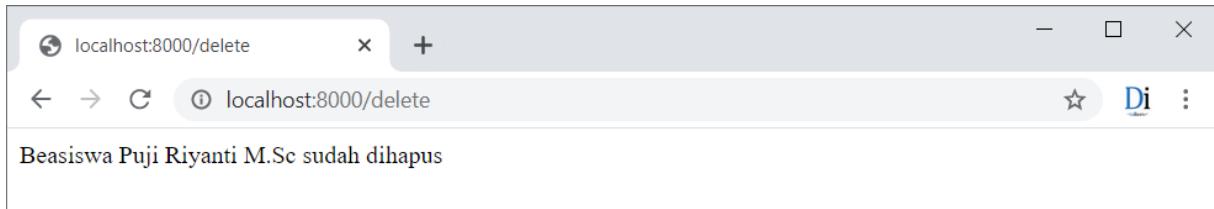
Untuk menghapus data beasiswa, bisa menggunakan method `delete()`. Namun karena di tabel `beasiswas` tidak terdapat `foreign key`, bisa saja beasiswa tersebut masih terhubung ke dosen atau mahasiswa yang sudah tidak ada.

Agar terjadi konsistensi data, maka ketika data dosen atau mahasiswa dihapus, isi tabel `beasiswas` juga harus dihapus secara manual:

```
app\Http\Controllers\AplikasiController.php

1 public function delete()
2 {
3     $dosen = Dosen::where('nama', 'Puji Riyanti M.Sc')->first();
4     $dosen->delete();
5     $dosen->beasiswa()->delete();
6 }
```

```
7     echo "Beasiswa $dosen->nama sudah dihapus";  
8 }
```



Gambar: Menghapus data dosen dan juga data beasiswa

Perintah `delete()` di baris 4 akan menghapus data dosen, sedangkan perintah `delete()` di baris 5 akan menghapus semua data beasiswa untuk dosen tersebut.

Untuk aplikasi yang sensitif, bisa memakai fitur transaction MySQL agar proses hapus kedua tabel bisa berjalan bersamaan atau tidak sama sekali. Alternatif lain juga bisa lewat fitur `trigger` di MySQL untuk membuat otomatisasi penghapusan data beasiswa.

Dalam bab ini kita sudah membahas *one to many polymorphic relationship*. Berikutnya akan masuk ke variasi terakhir dari polymorphic relationship, yakni **many to many polymorphic relationship**.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

19. Eloquent Relationship: Many to Many Polymorphic

Kita sampai ke relationship terakhir sekaligus yang paling kompleks, yakni **polymorphic relationship many to many**. Sesuai dengan namanya, ini adalah versi *many to many* dari *polymorphic relationship*.

Agar seragam dan menghindari error akibat praktek dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

19.1. Pengertian Many to Many Polymorphic Relationship

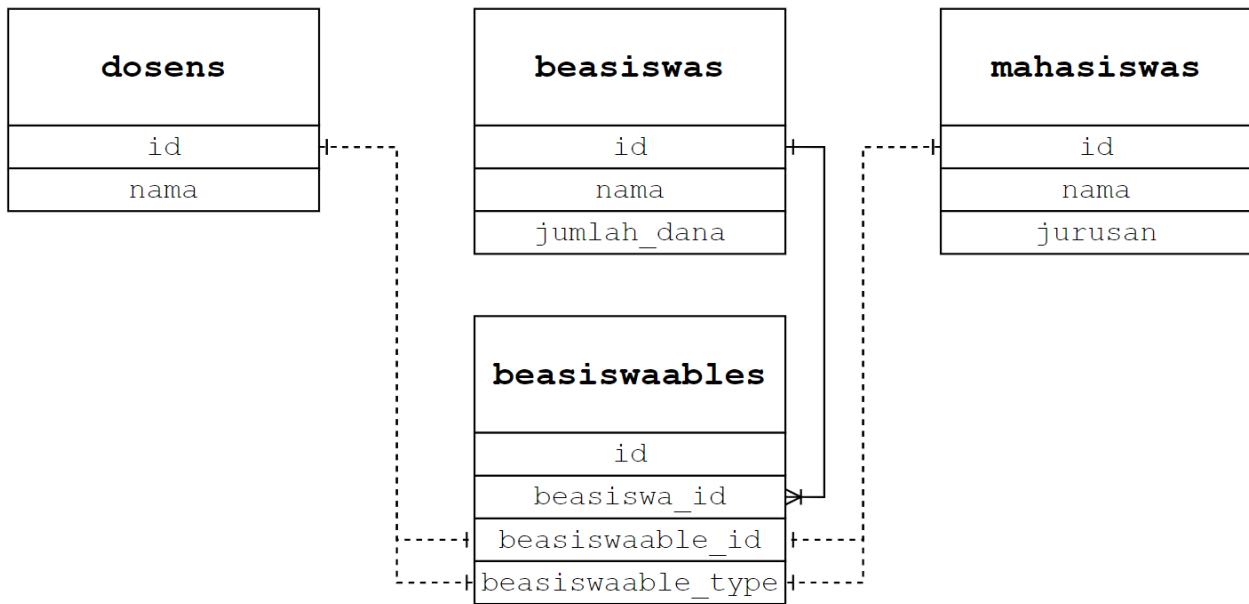
Dalam *many to many polymorphic relationship*, satu data di tabel utama bisa memiliki banyak data di tabel kedua, demikian juga satu data di tabel kedua bisa berpasangan dengan banyak data di tabel utama.

Jika diterapkan ke sistem informasi beasiswa, ini berarti satu mahasiswa atau satu dosen bisa mengambil banyak beasiswa sekaligus. Sebaliknya, satu beasiswa juga bisa diambil oleh banyak dosen atau mahasiswa.

Sama seperti di *many to many relationship*, versi *polymorphic* juga butuh tabel **pivot** untuk mencatat setiap hubungan yang terjadi. Ini membuat struktur tabel menjadi sedikit kompleks. Untuk sistem informasi beasiswa, nantinya butuh 4 buah tabel: tabel `dosens`, tabel `mahasiswa`, tabel `beasiswa`, dan tabel `beasiswaables`.

Tabel terakhir inilah yang akan menjadi tabel pivot. Penamaan tabel pivot pada *polymorphic* ditulis dengan akhiran "ables", yang dalam contoh kita menjadi `beasiswaables`.

Berikut diagram hubungan antar tabel untuk sistem informasi beasiswa versi *many to many polymorphic relationship*:



Gambar: Struktur tabel penerapan many to many polymorphic relationship

Tabel **beasiswa** sekarang berfungsi mencatat nama-nama beasiswa saja. Informasi mengenai siapa yang mengambil beasiswa tersebut, dipindahkan ke tabel pivot **beasiswaables**.

Tabel **beasiswaables** sendiri berisi 3 kolom utama, yakni **beasiswa_id** yang bertindak sebagai *foreign key* dari tabel **beasiswa**, serta kolom **beasiswaable_id** dan kolom **beasiswaable_type**.

Hubungan dari setiap tabel ini akan kita bahas lebih lanjut dengan contoh praktek.

19.2. Generate Data Sample

Berdasarkan diagram sebelumnya, kita perlu membuat 4 buah tabel:

- ◆ Tabel **dosen**: `id`, `nama`, `created_at` dan `updated_at`.
- ◆ Tabel **mahasiswa**: `id`, `nama`, `jurusan`, `created_at` dan `updated_at`.
- ◆ Tabel **beasiswa**: `id`, `nama`, `jumlah_dana`, `created_at` dan `updated_at`.
- ◆ Tabel **beasiswaables**: `id`, `beasiswa_id`, `beasiswaable_id`, `beasiswaable_type`, `created_at` dan `updated_at`.

Kelima empat tabel akan di-generate menggunakan migration. Silahkan buka cmd dan jalankan perintah **php artisan** berikut:

```

php artisan make:model Dosen -m
php artisan make:model Mahasiswa -m
php artisan make:model Beasiswa -m
php artisan make:migration create_beasiswaables_table

php artisan make:controller AplikasiController
  
```

Eloquent Relationship: Many to Many Polymorphic

Kode di atas akan membuat file **model** dan **migration** serta satu file **AplikasiController**.

Setelah semua file tersedia, buka file migration dan isi dengan kode di bawah ini:

database\migrations\<timestamp>_create_dosens_table.php

```
1 public function up()
2 {
3     Schema::create('dosens', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->timestamps();
7     });
8 }
```

database\migrations\<timestamp>_create_mahasiswa_table.php

```
1 public function up()
2 {
3     Schema::create('mahasiswa', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->string('jurusan');
7         $table->timestamps();
8     });
9 }
```

database\migrations\<timestamp>_create_beasiswa_table.php

```
1 public function up()
2 {
3     Schema::create('beasiswa', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->integer('jumlah_dana');
7         $table->timestamps();
8     });
9 }
```

database\migrations\<timestamp>_create_beasiswaables_table.php

```
1 public function up()
2 {
3     Schema::create('beasiswaables', function (Blueprint $table) {
4         $table->id();
5         $table->foreignId('beasiswa_id')->constrained()->onDelete('cascade');
6         $table->morphs('beasiswaable');
7         $table->timestamps();
8     });
9 }
```

Perintah `$table->morphs('beasiswaable')` sekarang ada di tabel `beasiswaables`. Selain itu saya juga membuat batasan `onDelete('cascade')` pada kolom foreign key `beasiswa_id`.

Dengan demikian jika isi tabel `beasiswa` dihapus, data yang terhubung di tabel `beasiswaables` juga akan ikut dihapus. Silahkan buat keempat tabel dengan menjalankan perintah php `artisan migrate`.

Lanjut, kita perlu beberapa data sample yang akan di generate dari file `DatabaseSeeder.php`:

`database\seeders\DatabaseSeeder.php`

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Faker\Factory as Faker;
7
8 class DatabaseSeeder extends Seeder
9 {
10     public function run()
11     {
12         $faker = Faker::create('id_ID');
13         $faker->seed(123);
14
15         $daftar_titel = ["M.Kom", "M.Sc", "M.T", "M.Si"];
16         for ($i=0; $i<5; $i++) {
17             \App\Models\Dosen::create(
18                 [
19                     'nama' => $faker->firstName." ".$faker->lastName." ".
20                             $faker->randomElement($daftar_titel)
21                 ]
22             );
23         }
24
25         $jurusan = ["Ilmu Komputer", "Teknik Informatika", "Sistem Informasi"];
26         for ($i=0; $i<5; $i++) {
27             \App\Models\Mahasiswa::create(
28                 [
29                     'nama' => $faker->firstName." ".$faker->lastName,
30                     'jurusan' => $faker->randomElement($jurusan),
31                 ]
32             );
33         }
34
35         $beasiswa = ["Pertamina", "Telkom", "LPDP", "Kemendikbud", "PPA"];
36         for ($i=0; $i<5; $i++) {
37             \App\Models\Beasiswa::create(
38                 [
39                     'nama'=>"Beasiswa ".$faker->unique()->randomElement($beasiswa),
40                     'jumlah_dana' => $faker->numberBetween(5,12)*1000000,
41                 ]
42             );
43         }
44     }
45 }
46 }
```

Eloquent Relationship: Many to Many Polymorphic

File seeder ini akan men-generate 5 data dosen, 5 data mahasiswa serta 5 data beasiswa. Jalankan seeder dengan perintah `php artisan db:seed`.

19.3. Eloquent Relationship `morphToMany()` & `morphedByMany()`

Untuk *many to many polymorphic relationship*, kita butuh pasangan method `morphToMany()` di model utama, serta method `morphedByMany()` di model kedua.

Silahkan buka file model Dosen, lalu tambah kode berikut:

app\Models\Dosen.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 Dosen extends Model
9 {
10     use HasFactory;
11     public function beasiswas()
12     {
13         return $this->morphToMany('App\Models\Beasiswa', 'beasiswaable')
14             ->withTimestamps();
15     }
16 }
```

Nama method penghubung adalah `beasiswas()`, menggunakan kata plural dengan akhiran 's' karena satu dosen nantinya bisa memiliki banyak beasiswa.

Argument pertama pada method `morphToMany()` diisi dengan nama model yang akan dihubungkan, kemudian sebagai argument kedua berupa nama singular tabel pivot (tanpa akhiran 's'), yang dalam contoh ini bernama 'beasiswaable'.

Method `withTimestamps()` ditambahkan agar tabel pivot ter-update saat terjadi perubahan data. Penjelasan lebih lengkap sudah kita bahas pada materi *relationship many to many*.

Method yang sama juga perlu di tulis ke model Mahasiswa:

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
```

Eloquent Relationship: Many to Many Polymorphic

```
9  {
10     use HasFactory;
11     public function beasiswas()
12     {
13         return $this->morphToMany( 'App\Models\Beasiswa' , 'beasiswaable')
14         ->withTimestamps();
15     }
16 }
```

Dan berikut kode untuk model Beasiswa:

app\Models\Beasiswa.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 Beasiswa extends Model
9  {
10     use HasFactory;
11     public function dosens()
12     {
13         return $this->morphedByMany( 'App\Models\Dosen' , 'beasiswaable');
14     }
15
16     public function mahasiswas()
17     {
18         return $this->morphedByMany( 'App\Models\Mahasiswa' , 'beasiswaable');
19     }
20 }
```

Untuk model Beasiswa sekarang terdapat method dosens() dan mahasiswas() yang menjalankan perintah `return $this->morphedByMany(...)`. Sebagai argument pertama berupa nama model yang terhubung, serta argument kedua berupa nama tabel pivot tanpa akhiran 's'.

Sekarang saatnya masuk ke praktik penulisan perintah eloquent. Silahkan tambah beberapa route berikut:

routes\web.php

```
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\AplikasiController;
5
6  Route::get( '/all' , [AplikasiController::class, 'all']);
7
8  Route::get( '/input-beasiswa-1' , [AplikasiController::class, 'inputBeasiswa1']);
9  Route::get( '/input-beasiswa-2' , [AplikasiController::class, 'inputBeasiswa2']);
10
```

Eloquent Relationship: Many to Many Polymorphic

```
11 Route::get('/tampil-beasiswa-1',[AplikasiController::class,'tampilBeasiswa1']);  
12 Route::get('/tampil-beasiswa-2',[AplikasiController::class,'tampilBeasiswa2']);  
13 Route::get('/tampil-beasiswa-3',[AplikasiController::class,'tampilBeasiswa3']);  
14 Route::get('/tampil-beasiswa-4',[AplikasiController::class,'tampilBeasiswa4']);  
15  
16 Route::get('/with-count', [AplikasiController::class,'withCount']);  
17  
18 Route::get('/detach', [AplikasiController::class,'detach']);  
19  
20 Route::get('/delete-beasiswa', [AplikasiController::class,'deleteBeasiswa']);  
21 Route::get('/delete-mahasiswa', [AplikasiController::class,'deleteMahasiswa']);
```

Kita akan bahas satu per satu. Route pertama dipakai untuk menampilkan semua isi tabel yang ada saat ini:

app\Http\Controllers\AplikasiController.php

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4 use App\Models\Dosen;  
5 use App\Models\Mahasiswa;  
6 use App\Models\Beasiswa;  
7 use Illuminate\Http\Request;  
8  
9 class AplikasiController extends Controller  
10 {  
11     public function all()  
12     {  
13         echo "## Dosen ## <br>";  
14         $dosen = Dosen::all();  
15         foreach ($dosen as $dosen) {  
16             echo "$dosen->nama | ";  
17         }  
18         echo "<hr>";  
19  
20         echo "## Mahasiswa ## <br>";  
21         $mahasiswa = Mahasiswa::all();  
22         foreach ($mahasiswa as $mahasiswa) {  
23             echo "$mahasiswa->nama | $mahasiswa->jurusan <br>";  
24         }  
25         echo "<hr>";  
26  
27         echo "## Beasiswa ## <br>";  
28         $beasiswa = Beasiswa::all();  
29         foreach ($beasiswa as $beasiswa) {  
30             echo "$beasiswa->nama | $beasiswa->jumlah_dana <br>";  
31         }  
32     }  
33 }  
34 }  
35 }
```

Eloquent Relationship: Many to Many Polymorphic

```
## Dosen ##
Empluk Nuraini M.Si | Ibrani Salahudin M.T | Shania Pertiwi M.Sc | Mila Ramadan M.T | Puji Riyanti M.Sc |

## Mahasiswa ##
Juliana Nurdyanti | Teknik Informatika
Pia Nasidah | Sistem Informasi
Kambali Prasetya | Teknik Informatika
Edward Prabowo | Ilmu Komputer
Lanjar Laksita | Teknik Informatika

## Beasiswa ##
Beasiswa Telkom | 7000000
Beasiswa Kemendikbud | 6000000
Beasiswa Pertamina | 10000000
Beasiswa LPDP | 12000000
Beasiswa PPA | 8000000
```

Gambar: Menampilkan semua isi tabel

Sip, semua data sudah bisa diakses.

Menginput Data Relationship

Saat ini belum ada hubungan antara data dosen atau mahasiswa dengan tabel `beasiswas`. Kita akan coba input beberapa data. Dalam kode berikut saya ingin menghubungkan 3 data dosen dengan 2 data beasiswa:

```
app\Http\Controllers\AplikasiController.php

1 public function inputBeasiswa1()
2 {
3     $dosen = Dosen::find([3, 4, 5]);
4     $beasiswas = Beasiswa::where('jumlah_dana', '>=', 10000000)->get();
5
6     foreach ($dosen as $dosen) {
7         $dosen->beasiswas()->attach($beasiswas);
8     }
9
10    echo "Semua dosen sudah mendapat beasiswa";
11 }
```

```
Semua dosen sudah mendapat beasiswa
```

Gambar: Menambah Beasiswa Dosen

Di baris 3, variabel `$dosen` diisi dengan object `Dosen` yang memiliki id 3, 4 dan 5. Artinya akan ada 3 object `dosen` di dalam variabel `$dosen`.

Kemudian pada baris 4, variabel `$beasiswas` diisi dengan semua beasiswa yang jumlah dananya di atas 10.000.000. Melihat ke dalam tabel `beasiswas`, pencarian ini akan cocok di 2

buah beasiswa, yakni "Beasiswa Pertamina" dan "Beasiswa LPDP".

Saya ingin agar ketiga dosen yang terdapat di variabel \$dosens mengambil semua beasiswa yang ada di variabel \$beasiswas. Ini dilakukan dengan mengakses setiap object Dosen menggunakan perintah `foreach ($dosens as $dosen)`, lalu pada setiap perulangan jalankan perintah `$dosen->beasiswas()->attach($beasiswas)`.

Penggunaan method `attach()` mirip seperti yang kita pakai pada relationship many to many biasa. Hasilnya, setiap dosen akan terhubung ke kedua beasiswa. Hubungan ini menambah 6 data baru ke dalam tabel `beasiswaables`:

```
MySQL [laravel] > SELECT * FROM beasiswaables;
+----+-----+-----+-----+-----+
| id | beasiswa_id | beasiswaable_type | beasiswaable_id | created_at      | updated_at      |
+----+-----+-----+-----+-----+
| 1  |      3      | App\Models\Dosen |           3     | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 2  |      4      | App\Models\Dosen |           3     | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 3  |      3      | App\Models\Dosen |           4     | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 4  |      4      | App\Models\Dosen |           4     | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 5  |      3      | App\Models\Dosen |           5     | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 6  |      4      | App\Models\Dosen |           5     | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
+----+-----+-----+-----+-----+
6 rows in set (0.002 sec)
```

Gambar: Isi tabel `beasiswaables`

Sebagai contoh, di baris 1 kita bisa diambil kesimpulan bahwa beasiswa yang ber-id 3 terhubung ke Dosen yang juga ber-id 3. Di baris 2 beasiswa ber-id 4 terhubung ke Dosen yang ber-id 3, dst. Inilah prinsip penyimpanan data dalam *many to many polymorphic relationship*.

Berikutnya, kita akan test untuk data Mahasiswa:

`app\Http\Controllers\AplikasiController.php`

```
1  public function inputBeasiswa2()
2  {
3      $mahasiswa = Mahasiswa::where('jurusan', 'Teknik Informatika')->get();
4      $beasiswas = Beasiswa::whereIn('nama',
5                                     ['Beasiswa Telkom', 'Beasiswa LPDP', 'Beasiswa PPA'])->get();
6
7      foreach ($mahasiswa as $mahasiswa) {
8          $mahasiswa->beasiswas()->sync($beasiswas);
9      }
10
11     // Cari mahasiswa dengan id 4, lalu attach dengan beasiswa id 3
12     Mahasiswa::find(4)->beasiswas()->attach(Beasiswa::find(3));
13
14     echo "Mahasiswa sudah mendapat beasiswa";
15 }
```

Eloquent Relationship: Many to Many Polymorphic



Gambar: Menambah Beasiswa Mahasiswa

Di baris 3, variabel `$mahasiswas` akan berisi semua mahasiswa dari jurusan Teknik Informatika. Jika dilihat ke dalam tabel `mahasiswas`, ini akan cocok dengan 3 orang mahasiswa.

Lalu di baris 4 variabel `$beasiswas` akan berisi 3 beasiswa yang namanya sesuai dengan yang terdapat dalam perintah tersebut.

Kali ini proses input beasiswa menggunakan perintah `foreach` dan `sync()` seperti di baris 7 - 9. Kembali, penjelasan tentang cara penggunaan method `sync()` sudah kita bahas di bab many to many relationship.

Di baris 12, saya menulis proses input dalam 1 perintah bersambung. Perintah ini akan menghubungkan mahasiswa dengan id-4 ke beasiswa yang ber-id 3.

Berikut isi tabel `beasiswaables` setelah menjalankan kode di atas:

```
MySQL [laravel]> SELECT * FROM beasiswaables;
+----+-----+-----+-----+-----+-----+
| id | beasiswa_id | beasiswaable_type | beasiswaable_id | created_at | updated_at |
+----+-----+-----+-----+-----+-----+
| 1  |      3     | App\Models\Dosen |          3       | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 2  |      4     | App\Models\Dosen |          3       | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 3  |      3     | App\Models\Dosen |          4       | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 4  |      4     | App\Models\Dosen |          4       | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 5  |      3     | App\Models\Dosen |          5       | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 6  |      4     | App\Models\Dosen |          5       | 2020-10-11 03:13:09 | 2020-10-11 03:13:09 |
| 7  |      1     | App\Models\Mahasiswa |          1       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 8  |      4     | App\Models\Mahasiswa |          1       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 9  |      5     | App\Models\Mahasiswa |          1       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 10 |      1     | App\Models\Mahasiswa |          3       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 11 |      4     | App\Models\Mahasiswa |          3       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 12 |      5     | App\Models\Mahasiswa |          3       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 13 |      1     | App\Models\Mahasiswa |          5       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 14 |      4     | App\Models\Mahasiswa |          5       | 2020-10-11 03:15:12 | 2020-10-11 03:15:12 |
| 15 |      5     | App\Models\Mahasiswa |          5       | 2020-10-11 03:15:13 | 2020-10-11 03:15:13 |
| 16 |      3     | App\Models\Mahasiswa |          4       | 2020-10-11 03:15:13 | 2020-10-11 03:15:13 |
+----+-----+-----+-----+-----+-----+
16 rows in set (0.002 sec)
```

Gambar: Isi tabel beasiswaables

Terdapat tambahan 10 data baru (baris 7 - 16). Pada kode input sebelumnya, setiap mahasiswa yang ada di variabel `$mahasiswas`, mengambil semua beasiswa di variabel `$beasiswas` (total $3 \times 3 = 9$), plus tambahan 1 data dari method `attach()`.

Menampilkan Data Relationship

Jika berangkat dari tabel utama, perintah yang dipakai untuk menampilkan data polymorphic many to many pada dasarnya sama seperti *relationship many to many* biasa:

Eloquent Relationship: Many to Many Polymorphic

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa1()
2 {
3     $dosen = Dosen::where('nama', 'Puji Riyanti M.Sc')->first();
4
5     echo "## Daftar beasiswa $dosen->nama ##";
6     echo "<hr>";
7     foreach ($dosen->beasiswas as $beasiswa)
8     {
9         echo "Beasiswa $beasiswa->nama
10            (Rp. ".number_format($beasiswa->jumlah_dana).")<br>";
11     }
12 }
```



Gambar: Menampilkan semua beasiswa dari 1 dosen

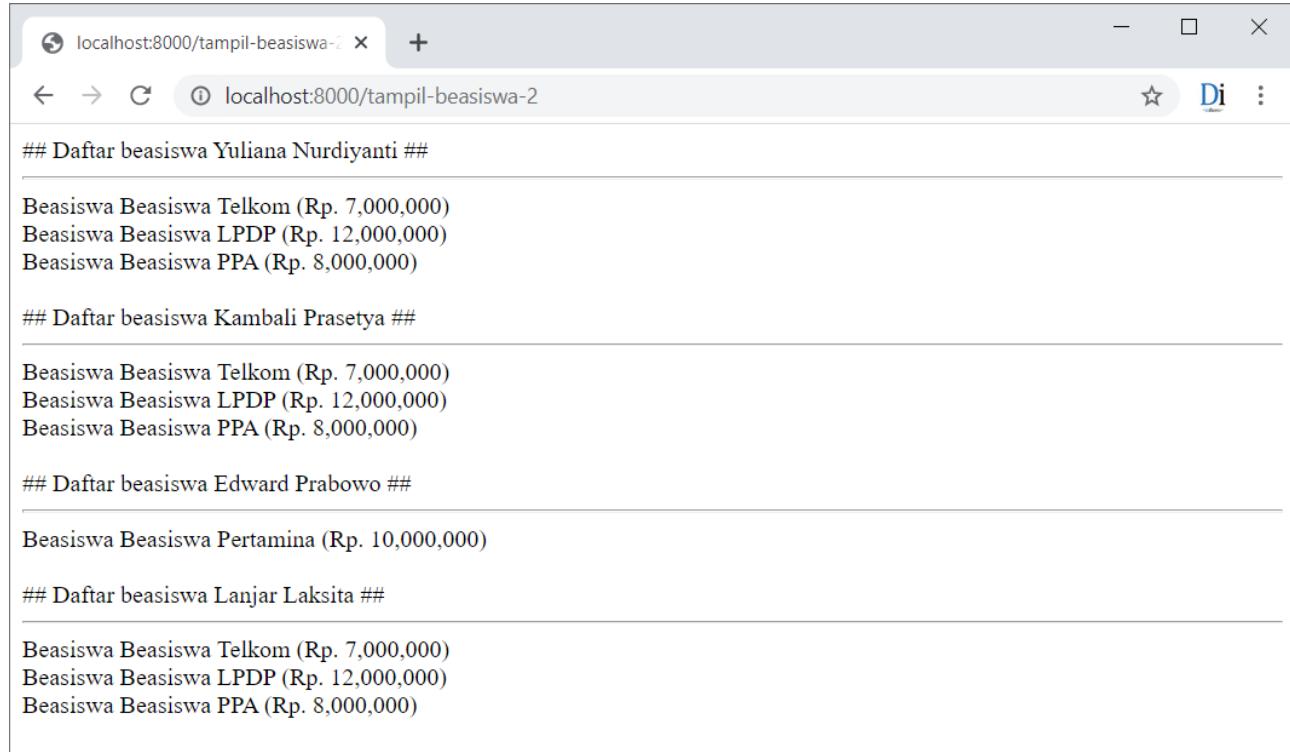
Karena seorang dosen bisa memiliki banyak beasiswa, maka perlu perulangan foreach di baris 7 untuk membuka hasil collection.

Contoh lain adalah menampilkan apa saja beasiswa yang diambil oleh mahasiswa:

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa2()
2 {
3     $mahasiswas = Mahasiswa::has('beasiswas')->get();
4
5     foreach ($mahasiswas as $mahasiswa) {
6         echo "## Daftar beasiswa $mahasiswa->nama ##";
7         echo "<hr>";
8         foreach ($mahasiswa->beasiswas as $beasiswa)
9         {
10             echo "Beasiswa $beasiswa->nama
11                (Rp. ".number_format($beasiswa->jumlah_dana).")<br>";
12         }
13         echo "<br>";
14     }
15 }
```

Eloquent Relationship: Many to Many Polymorphic



```
localhost:8000/tampil-beasiswa-2

## Daftar beasiswa Yuliana Nurdyanti ##

Beasiswa Beasiswa Telkom (Rp. 7,000,000)
Beasiswa Beasiswa LPDP (Rp. 12,000,000)
Beasiswa Beasiswa PPA (Rp. 8,000,000)

## Daftar beasiswa Kambali Prasetya ##

Beasiswa Beasiswa Telkom (Rp. 7,000,000)
Beasiswa Beasiswa LPDP (Rp. 12,000,000)
Beasiswa Beasiswa PPA (Rp. 8,000,000)

## Daftar beasiswa Edward Prabowo ##

Beasiswa Beasiswa Pertamina (Rp. 10,000,000)

## Daftar beasiswa Lanjar Laksita ##

Beasiswa Beasiswa Telkom (Rp. 7,000,000)
Beasiswa Beasiswa LPDP (Rp. 12,000,000)
Beasiswa Beasiswa PPA (Rp. 8,000,000)
```

Gambar: Menampilkan semua mahasiswa yang mengambil beasiswa

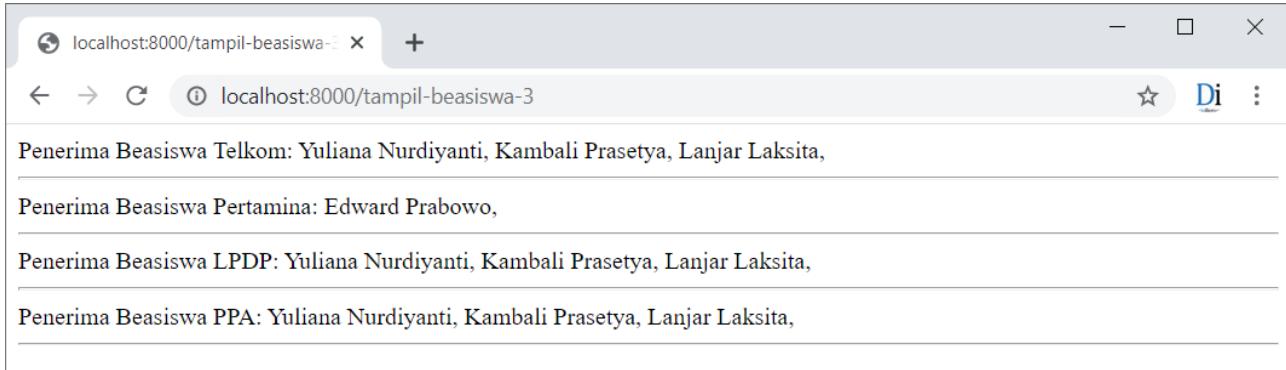
Contoh ini sangat mirip seperti di *one to many polymorphic* sehingga saya rasa tidak perlu di bahas lagi.

Proses menampilkan data juga bisa dilakukan dari model Beasiswa. Dalam kode program berikut saya ingin menampilkan semua beasiswa yang sudah diambil mahasiswa, serta nama mahasiswa tersebut:

```
app\Http\Controllers\AplikasiController.php

1 public function tampilBeasiswa3()
2 {
3     $beasiswas = Beasiswa::has('mahasiswas')->get();
4
5     foreach ($beasiswas as $beasiswa) {
6         echo "Penerima $beasiswa->nama: ";
7         foreach ($beasiswa->mahasiswas as $mahasiswa)
8         {
9             echo "$mahasiswa->nama, ";
10            }
11            echo "<hr>";
12        }
13    }
```

Eloquent Relationship: Many to Many Polymorphic



Gambar: Menampilkan nama yang mengambil beasiswa

Kali ini kita tidak butuh method `whereHasMorph()`, tapi cukup method `has()` biasa. Perintah `Beasiswa::has('mahasiswa')->get()` akan mengembalikan semua object Beasiswa yang memiliki (has) mahasiswa.

Kita juga bisa memakai method `doesntHave()` jika ingin menampilkan data yang tidak memiliki pasangan. Sebagai contoh, kode berikut menampilkan semua beasiswa yang tidak diambil oleh satu pun dosen:

app\Http\Controllers\AplikasiController.php

```
1 public function tampilBeasiswa4()
2 {
3     $beasiswas = Beasiswa::doesntHave('dosens')->get();
4
5     echo "## Daftar beasiswa yang tidak diambil dosen ##";
6     echo "<hr>";
7     foreach ($beasiswas as $beasiswa) {
8         echo "$beasiswa->nama <br>";
9     }
10 }
```



Gambar: Daftar beasiswa yang tidak diambil dosen

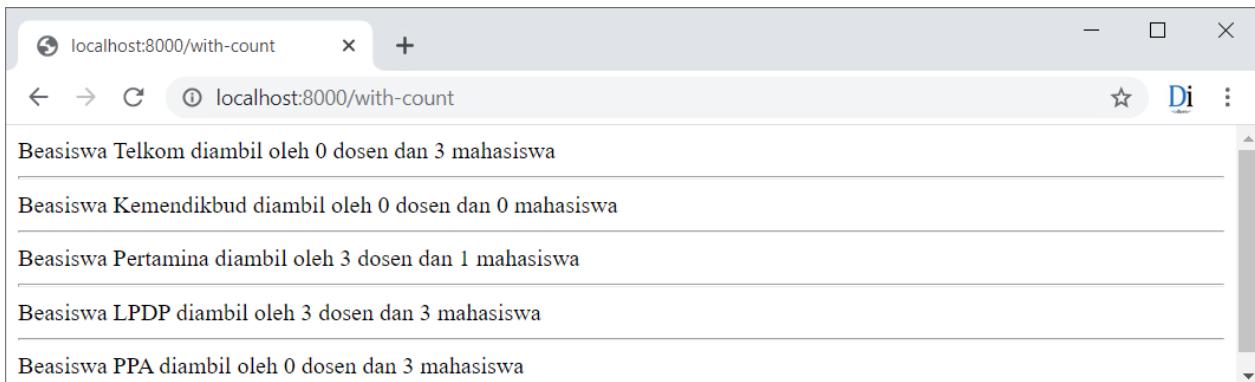
Contoh menampilkan data relationship lain adalah menggunakan method `withCount()` untuk menghitung berapa kali sebuah beasiswa diambil:

app\Http\Controllers\AplikasiController.php

```
1 public function withCount()
2 {
```

Eloquent Relationship: Many to Many Polymorphic

```
3 // Tampilkan total dosen dan mahasiswa yang mengambil beasiswa
4 $beasiswas = Beasiswa::withCount(['dosens', 'mahasiswa'])->get();
5 foreach ($beasiswas as $beasiswa) {
6     echo "$beasiswa->nama diambil oleh $beasiswa->dosens_count dosen dan
7         $beasiswa->mahasiswa_count mahasiswa <hr>";
8 }
9 }
```



Gambar: Jumlah dosen dan mahasiswa yang mengambil mahasiswa

Perintah `Beasiswa::withCount(['dosens', 'mahasiswa'])->get()` akan mengembalikan semua object `Beasiswa` dengan tambahan kolom `dosens_count` serta `mahasiswa_count` yang berisi informasi jumlah dosen dan mahasiswa yang mengambil beasiswa tersebut.

Mengupdate Data Relationship

Proses update data di *many to many polymorphic* bisa dilakukan dengan method yang juga tersedia untuk many to many biasa, diantaranya `attach()`, `detach()`, `syncWithoutDetaching()`, serta `toggle()`.

Sebagai contoh, saat ini dosen Puji Riyanti M.Sc sudah mengambil 2 buah beasiswa:



Gambar: Beasiswa yang diambil oleh Puji Riyanti M.Sc

Untuk "memutus hubungan" antara Puji Riyanti M.Sc dengan Beasiswa Pertamina, bisa menggunakan method `detach()` sebagai berikut:

app\Http\Controllers\AplikasiController.php

```
1 public function detach()
2 {
3     $dosen = Dosen::where('nama', 'Puji Riyanti M.Sc')->first();
4     $beasiswa = Beasiswa::where('nama', 'Beasiswa Pertamina')->first();
```

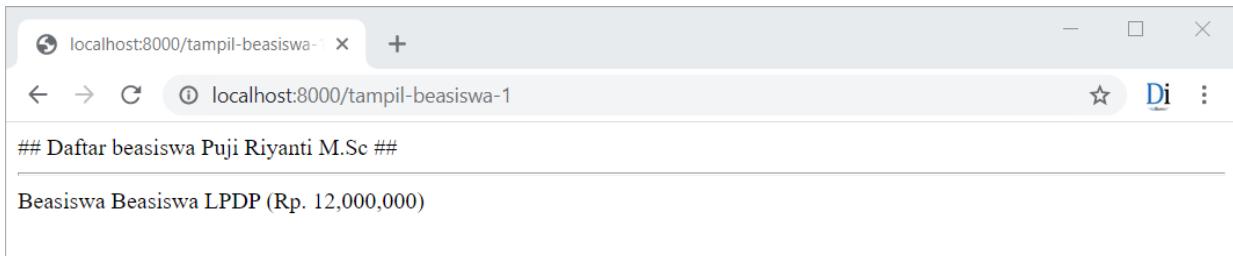
Eloquent Relationship: Many to Many Polymorphic

```
5      $dosen->beasiswa->detach($beasiswa);
6
7      echo " $dosen->nama tidak lagi dapat $beasiswa->nama";
8
9 }
```



Gambar: Proses detach Beasiswa Pertamina dari Puji Riyanti M.Sc

Setelah menjalankan kode di atas, maka beasiswa yang terdaftar untuk Puji Riyanti M.Sc hanya tinggal satu saja:



Gambar: Beasiswa yang diambil oleh Puji Riyanti M.Sc

Untuk contoh penggunaan method `syncWithoutDetaching()` dan `toggle()` tidak akan saya bahas lagi karena penggunaannya sama persis seperti many to many relationship biasa.

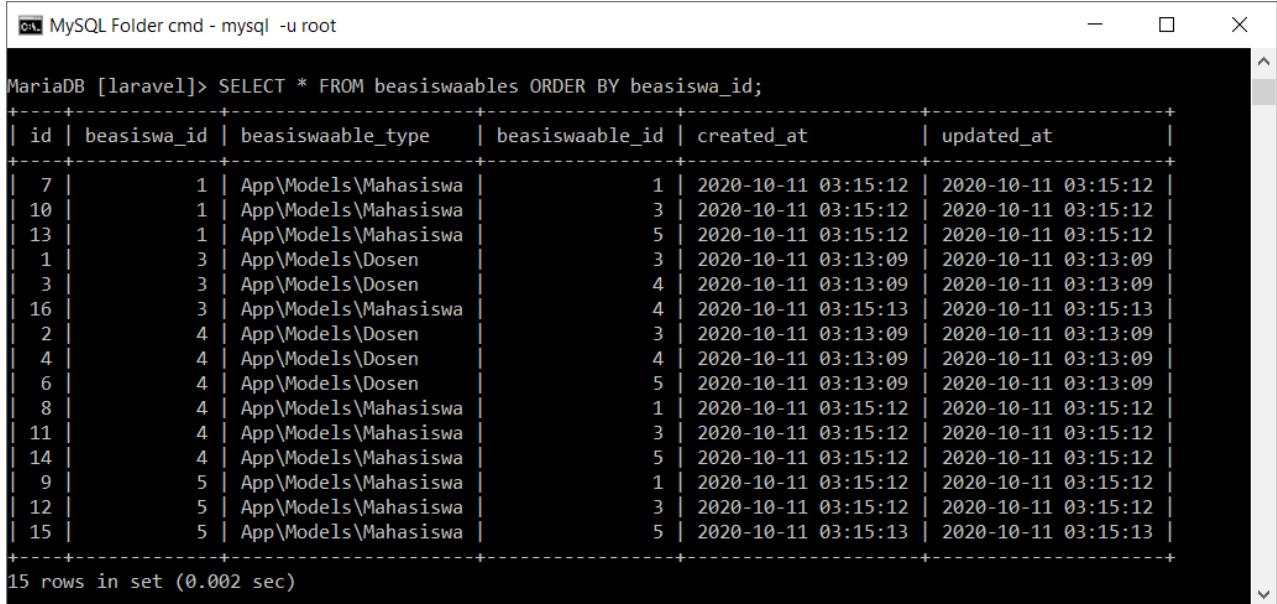
Menghapus Data Relationship

Untuk menghapus data, kita bisa memakai method `delete()` seperti biasa.

Pada saat membuat file migration tabel `beasiswaables`, ikut disertakan perintah `onDelete ('cascade')` pada kolom `beasiswa_id`. Ini berarti jika sebuah beasiswa dihapus dari tabel `beasiswas`, catatan mengenai beasiswa tersebut juga ikut terhapus dari tabel `beasiswaables`.

Sebelum uji coba proses penghapusan, berikut isi tabel `beasiswaables` saat ini:

Eloquent Relationship: Many to Many Polymorphic



The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". The query executed is "SELECT * FROM beasiswaables ORDER BY beasiswa_id;". The output displays 15 rows of data across six columns: id, beasiswa_id, beasiswaable_type, beasiswaable_id, created_at, and updated_at. The data indicates many-to-many relationships between Beasiswa and Mahasiswa/Dosen.

id	beasiswa_id	beasiswaable_type	beasiswaable_id	created_at	updated_at
7	1	App\Models\Mahasiswa	1	2020-10-11 03:15:12	2020-10-11 03:15:12
10	1	App\Models\Mahasiswa	3	2020-10-11 03:15:12	2020-10-11 03:15:12
13	1	App\Models\Mahasiswa	5	2020-10-11 03:15:12	2020-10-11 03:15:12
1	3	App\Models\Dosen	3	2020-10-11 03:13:09	2020-10-11 03:13:09
3	3	App\Models\Dosen	4	2020-10-11 03:13:09	2020-10-11 03:13:09
16	3	App\Models\Mahasiswa	4	2020-10-11 03:15:13	2020-10-11 03:15:13
2	4	App\Models\Dosen	3	2020-10-11 03:13:09	2020-10-11 03:13:09
4	4	App\Models\Dosen	4	2020-10-11 03:13:09	2020-10-11 03:13:09
6	4	App\Models\Dosen	5	2020-10-11 03:13:09	2020-10-11 03:13:09
8	4	App\Models\Mahasiswa	1	2020-10-11 03:15:12	2020-10-11 03:15:12
11	4	App\Models\Mahasiswa	3	2020-10-11 03:15:12	2020-10-11 03:15:12
14	4	App\Models\Mahasiswa	5	2020-10-11 03:15:12	2020-10-11 03:15:12
9	5	App\Models\Mahasiswa	1	2020-10-11 03:15:12	2020-10-11 03:15:12
12	5	App\Models\Mahasiswa	3	2020-10-11 03:15:12	2020-10-11 03:15:12
15	5	App\Models\Mahasiswa	5	2020-10-11 03:15:13	2020-10-11 03:15:13

15 rows in set (0.002 sec)

Gambar: Isi tabel beasiswaables

Kita akan coba hapus beasiswa dengan id 4 dari tabel beasiswas:

```
app\Http\Controllers\AplikasiController.php
```

```
1 public function deleteBeasiswa()
2 {
3     Beasiswa::find(4)->delete();
4     echo "Beasiswa dengan id 4 sudah terhapus";
5 }
```



Gambar: Menghapus beasiswa dengan id = 4

Dan berikut isi tabel beasiswas dan beasiswaables setelah proses penghapusan:

Eloquent Relationship: Many to Many Polymorphic

```
MariaDB [laravel]> SELECT * FROM beasiswa;
+----+-----+-----+-----+
| id | nama | jumlah_dana | created_at | updated_at |
+----+-----+-----+-----+
| 1 | Beasiswa Telkom | 7000000 | 2020-10-11 03:21:55 | 2020-10-11 03:21:55 |
| 2 | Beasiswa Kemendikbud | 6000000 | 2020-10-11 03:21:55 | 2020-10-11 03:21:55 |
| 3 | Beasiswa Pertamina | 10000000 | 2020-10-11 03:21:55 | 2020-10-11 03:21:55 |
| 5 | Beasiswa PPA | 8000000 | 2020-10-11 03:21:55 | 2020-10-11 03:21:55 |
+----+-----+-----+-----+
4 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM beasiswaables ORDER BY beasiswa_id;
+----+-----+-----+-----+
| id | beasiswa_id | beasiswaable_type | beasiswaable_id | created_at | updated_at |
+----+-----+-----+-----+
| 7 | 1 | App\Models\Mahasiswa | 1 | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 10 | 1 | App\Models\Mahasiswa | 3 | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 13 | 1 | App\Models\Mahasiswa | 5 | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 1 | 3 | App\Models\Dosen | 3 | 2020-10-11 03:24:32 | 2020-10-11 03:24:32 |
| 3 | 3 | App\Models\Dosen | 4 | 2020-10-11 03:24:32 | 2020-10-11 03:24:32 |
| 16 | 3 | App\Models\Mahasiswa | 4 | 2020-10-11 03:35:52 | 2020-10-11 03:35:52 |
| 9 | 5 | App\Models\Mahasiswa | 1 | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 12 | 5 | App\Models\Mahasiswa | 3 | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 15 | 5 | App\Models\Mahasiswa | 5 | 2020-10-11 03:35:52 | 2020-10-11 03:35:52 |
+----+-----+-----+-----+
9 rows in set (0.002 sec)
```

Gambar: Isi tabel beasiswa dan beasiswaables

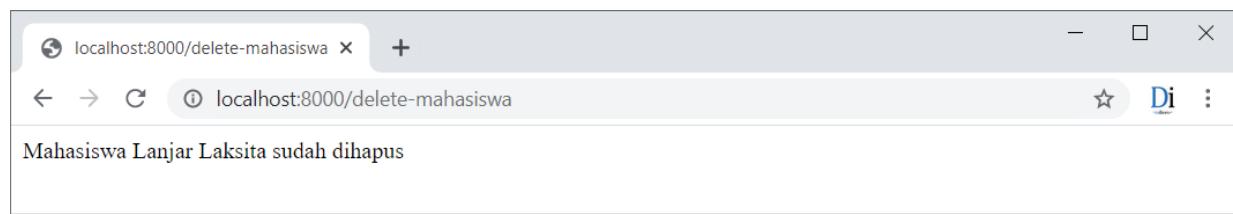
Terlihat data `beasiswa_id = 4` di tabel `beasiswaables` juga ikut terhapus.

Namun jika yang dihapus adalah tabel `mahasiswas` atau tabel `dosesns`, efek ON DELETE CASCADE tidak bisa berjalan, sebab kedua tabel tidak memiliki hubungan langsung dengan tabel `beasiswaables`.

Untuk menghindari data yang tidak valid, kita bisa menghapus manual data di tabel `beasiswaables` menggunakan method `detach()`:

app\Http\Controllers\AplikasiController.php

```
1 public function deleteMahasiswa()
2 {
3     $mahasiswa = Mahasiswa::where('nama', 'Lanjar Laksita')->first();
4     $mahasiswa->delete();
5     $mahasiswa->beasiswa()->detach();
6
7     echo "Mahasiswa $mahasiswa->nama sudah dihapus";
8 }
```



Gambar: Menghapus mahasiswa Lanjar Laksita

Di baris 3, variabel `$mahasiswa` akan berisi object Mahasiswa yang bernama Lanjar Laksita. Kemudian saya menghapusnya dengan perintah `$mahasiswa->delete()`. Agar catatan beasiswa di tabel `beasiswaables` juga ikut dihapus, perlu tambahan perintah `$mahasiswa->beasiswas()->detach()` di baris 5.

Berikut isi tabel `mahasiswas` dan `beasiswaables` setelah proses penghapusan:

```
MySQL Folder cmd - mysql -u root
MariaDB [laravel]> SELECT * FROM mahasiswas;
+----+-----+-----+-----+-----+
| id | nama           | jurusan        | created_at    | updated_at   |
+----+-----+-----+-----+-----+
| 1  | Yuliana Nurdyanti | Teknik Informatika | 2020-10-11 03:21:54 | 2020-10-11 03:21:54 |
| 2  | Pia Nasyidah    | Sistem Informasi | 2020-10-11 03:21:54 | 2020-10-11 03:21:54 |
| 3  | Kambali Prasetya | Teknik Informatika | 2020-10-11 03:21:54 | 2020-10-11 03:21:54 |
| 4  | Edward Prabowo  | Ilmu Komputer   | 2020-10-11 03:21:54 | 2020-10-11 03:21:54 |
+----+-----+-----+-----+-----+
4 rows in set (0.002 sec)

MariaDB [laravel]> SELECT * FROM beasiswaables ORDER BY beasiswaable_type;
+----+-----+-----+-----+-----+
| id | beasiswa_id | beasiswaable_type | beasiswaable_id | created_at    | updated_at   |
+----+-----+-----+-----+-----+
| 1  | 3           | App\Models\Dosen  | 3               | 2020-10-11 03:24:32 | 2020-10-11 03:24:32 |
| 3  | 3           | App\Models\Dosen  | 4               | 2020-10-11 03:24:32 | 2020-10-11 03:24:32 |
| 7  | 1           | App\Models\Mahasiswa | 1               | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 9  | 5           | App\Models\Mahasiswa | 1               | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 10 | 1           | App\Models\Mahasiswa | 3               | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 12 | 5           | App\Models\Mahasiswa | 3               | 2020-10-11 03:35:51 | 2020-10-11 03:35:51 |
| 16 | 3           | App\Models\Mahasiswa | 4               | 2020-10-11 03:35:52 | 2020-10-11 03:35:52 |
+----+-----+-----+-----+-----+
7 rows in set (0.001 sec)
```

Gambar: Isi tabel `mahasiswas` dan `beasiswaables`

Di dalam tabel `mahasiswas` tidak ada lagi mahasiswa yang ber-id 5. Begitu juga di tabel `beasiswaables`, kolom `beasiswaable_id` yang berisi angka 5 untuk `App\Models\Mahasiswa` juga sudah tidak ada.

Dengan selesainya bab *polymorphic relationship many to many* ini maka selesai juga pembahasan tentang eloquent relationship. Total kita sudah mempelajari 8 relationship yang tersedia di Laravel.

Pada prakteknya, relationship yang sering digunakan hanya 3, yakni *one to one*, *one to many*, serta *many to many relationship*. Tapi pengetahuan seputar jenis-jenis relationship akan sangat bermanfaat. Bukan saja untuk penggunaan di Laravel, tapi juga pengetahuan umum seputar teori database.

Berikutnya, kita akan masuk ke studi kasus pembuatan **Sistem Informasi Universitas ILKOOKM**.

20. Sistem Informasi Universitas ILKOOOM: Seed

Akhirnya kita sampai ke studi kasus utama dalam buku ini, yaitu membuat sebuah aplikasi **Sistem Informasi Universitas ILKOOOM**. Universitas ILKOOOM sendiri adalah kampus fiktif yang sudah beberapa kali saya pakai sebagai studi kasus di buku-buku DuniaIlkom.

Sistem Informasi yang akan kita buat memang bukan sebuah aplikasi lengkap, tapi lebih fokus ke cara penerapan konsep *relationship* untuk 5 tabel.

Yup, aplikasi **Sistem Informasi Universitas ILKOOOM** akan memiliki 4 tabel, atau 6 tabel jika memasukkan tabel `users` bawaan Laravel (untuk proses login/register) serta satu tabel pivot. Setiap tabel saling terhubung satu sama lain serta memiliki fitur CRUD lengkap, terdiri dari proses menambah, menampilkan, mengupdate dan menghapus data.

Meskipun belum berbentuk aplikasi Sistem Informasi utuh, namun kode yang akan kita tulis terbilang lumayan banyak. Sekedar gambaran, karena ada 4 tabel dan masing-masing tabel memiliki fitur CRUD, butuh setidaknya 16 file view untuk menampilkan form CRUD, belum termasuk kode untuk controller dan route.

Oleh sebab itu pembahasan materi akan dipecah ke dalam beberapa bab (supaya lebih mudah diikuti). Pada bab ini kita akan bahas tentang desain database serta pembuatan seeder.

Agar seragam dan menghindari error akibat praktik dari bab sebelumnya, kita akan mulai dari installer baru Laravel 8:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel01
```

Dalam bab ini juga perlu menjalankan ulang proses migration. Sehingga jika dalam database `laravel` masih terdapat tabel, silahkan hapus terlebih dahulu.

20.1. Proses Analisa Awal

Sebelum masuk ke pembuatan kode program, langkah awal yang harus dilakukan adalah menganalisa seperti apa kebutuhan client. Ini sangat penting karena jangan sampai kita memakai asumsi pribadi tanpa berkonsultasi dengan client.

Client di sini merujuk ke pihak yang meminta kita membuat aplikasi, bisa jadi itu adalah rekan kerja (biasanya dari divisi bisnis), atasan, atau seseorang yang menghubungi via WA.

Proses analisa sebenarnya lebih kepada "seni" dan setiap programmer memiliki pendekatan

yang berbeda-beda. Namun intinya adalah kita perlu menggali informasi terkait apa saja data yang harus disimpan di aplikasi dan seperti apa hasil akhir yang diinginkan.

Untuk sebuah sistem informasi, ini bisa berangkat dari mewawancara pihak kampus, terutama pimpinan organisasi yang butuh laporan berkala. Catat apa saja bentuk laporan yang diinginkan.

Setelah itu hubungi bagian administrasi atau tata usaha untuk mempelajari bagaimana alur dokumen yang sudah ada saat ini. Jika masih menggunakan kertas atau Excel, kumpulkan semua formulir karena itulah yang nantinya harus kita konversi ke dalam bentuk form HTML.

Wawancara juga karyawan yang bertugas menginput data, karena dialah yang langsung berhubungan dengan aplikasi kita setiap hari. Pahami kendala yang dihadapi selama ini dan apakah ada fitur yang ingin ditambah.

Dalam beberapa kasus, bisa saja pihak kampus sudah memiliki sistem informasi namun butuh perbaikan serta penambahan fitur baru. Di sini kita bisa putuskan apakah akan membuat aplikasi baru atau cukup memodifikasi kode yang sudah ada.

Untuk **Sistem Informasi Universitas ILKOOOM**, berikut hasil wawancara dengan pihak kampus:

- Perlu mencatat informasi terkait data jurusan, dosen, mahasiswa, dan mata kuliah.
- Sebuah jurusan terdiri dari banyak mahasiswa dan beberapa dosen.
- Setiap mata kuliah diajarkan oleh satu dosen, namun satu dosen bisa mengajar beberapa mata kuliah.
- Perlu tampilan mata kuliah yang diambil oleh setiap mahasiswa.
- Perlu tampilan semua mahasiswa yang mengambil satu mata kuliah tertentu.

Berdasarkan form yang dikumpulkan, berikut data yang harus tersedia:

- Nama jurusan, nama kepala jurusan, jumlah mahasiswa yang bisa ditampung.
- Nama dosen, NID (Nomor Induk Dosen), jurusan dosen.
- Nama mahasiswa, NIM (Nomor Induk Mahasiswa), jurusan mahasiswa.
- Nama matakuliah, kode matakuliah, jumlah sks, dosen pengajar.

Dalam proses analisis awal ini bisa saja terdapat data yang terlewat dan baru disadari setelah masuk ke tahap berikutnya. Ini tidak masalah karena selama pengembangan aplikasi kita harus selalu berkonsultasi dengan client.

20.2. Perancangan Design Database

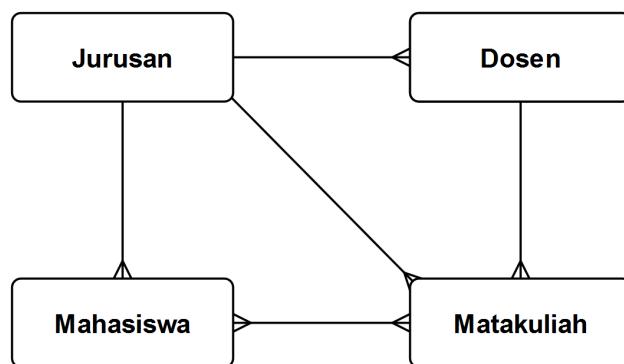
Setelah proses analisis awal, tahap selanjutnya adalah **perancangan design database**. Hasil

wawancara dan pengumpulan dokumen tadi akan kita konversi ke dalam bentuk tabel.

Pemahaman tentang teori dan praktik bahasa SQL sangat penting di sini. Apakah satu form harus dipecah menjadi beberapa tabel, atau apakah beberapa form bisa digabung menjadi satu tabel, itu butuh proses analisa mendalam.

Sebuah aplikasi sistem informasi umumnya butuh beberapa tabel yang saling terhubung satu sama lain. Kita juga harus definisikan bentuk hubungan ini, apakah *one to one*, *one to many*, atau *many to many*. Struktur tabel yang kurang baik akan menyulitkan pembuatan kode program nantinya.

Berdasarkan data yang didapat dari tahap analisis, saya akan membuat 4 buah tabel, yakni tabel **jurus**, tabel **dosen**, tabel **mahasiswa** dan tabel **matakuliah**. Berikut hubungan antar tabel:



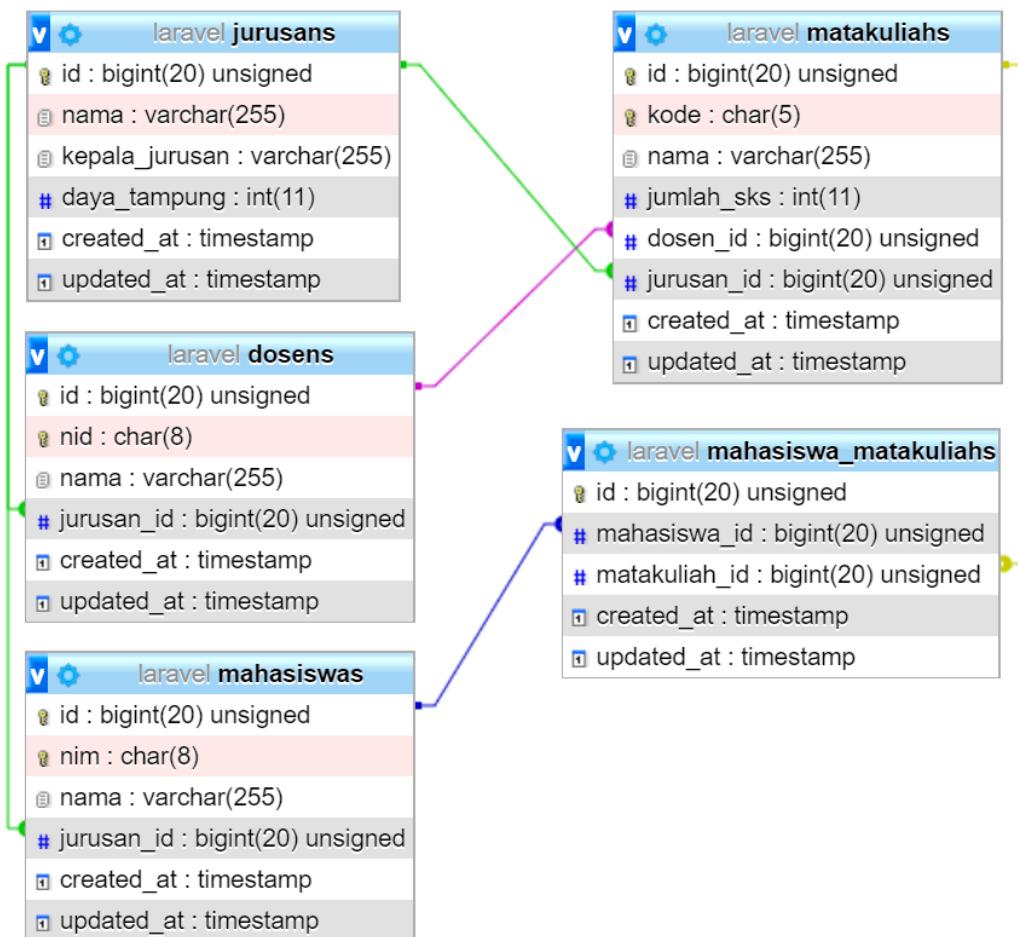
Gambar: Diagram ERD awal untuk Sistem Informasi Universitas Ilkoom

- Satu jurusan bisa memiliki banyak dosen, namun satu dosen hanya bisa terdaftar di satu jurusan saja. Artinya hubungan antara tabel jurusan dengan tabel dosen adalah **one to many**.
- Satu jurusan bisa memiliki banyak mahasiswa, namun satu mahasiswa hanya bisa terdaftar di satu jurusan saja. Artinya hubungan antara tabel jurusan dengan tabel mahasiswa adalah **one to many**.
- Satu jurusan bisa memiliki banyak mata kuliah, namun satu mata kuliah hanya bisa terdaftar di satu jurusan saja. Artinya hubungan antara tabel jurusan dengan tabel matakuliah adalah **one to many**.
- Satu dosen bisa mengajar banyak mata kuliah, namun satu mata kuliah hanya bisa diajar oleh satu dosen. Artinya hubungan antara tabel dosen dengan tabel matakuliah adalah **one to many**.
- Satu mahasiswa bisa mengambil banyak mata kuliah, serta satu mata kuliah juga bisa diambil oleh banyak mahasiswa. Artinya hubungan antara tabel mahasiswa dengan tabel matakuliah adalah **many to many**.

Diagram di atas belum memiliki daftar kolom, namun fokus utama kita ada di hubungan setiap tabel. Ujung garis mewakili relationship yang dibuat. Jika terdapat tanda cabang, itu mewakili kata "banyak". Logika hubungan ini biasanya harus di diskusikan lagi dengan client karena akan berdampak besar ke kode program.

Jika hubungan logika tabel sudah final, kita bisa masuk ke perancangan kolom tabel, termasuk menentukan tipe data kolom serta kolom apa saja yang bertindak sebagai *primary key* serta *foreign key*. Khusus untuk hubungan *many to many*, butuh juga bantuan tabel pivot.

Berikut diagram ERD serta penjelasan dari setiap kolom tabel:



Gambar: Diagram ERD final untuk Sistem Informasi Universitas Ilkoom

Tabel **jurusans**:

- ✓ **id**: sebagai primary key.
- ✓ **nama**: berisi nama jurusan seperti "Ilmu Komputer" atau "Teknik Informatika".
- ✓ **kepala_jurusan**: berisi nama kepala jurusan seperti "Prof. Gunarto M.Kom".
- ✓ **daya_tampung**: berisi angka daya tampung mahasiswa seperti 120 atau 200.

Tabel **dosens**:

- ✓ **id**: sebagai primary key.
- ✓ **nid**: berisi 8 digit nomor NID dosen seperti "99477796".
- ✓ **nama**: berisi nama dosen seperti "Hairyanto M.Kom" atau "Indah Wulandari M.Si".
- ✓ **jurusans_id**: berisi nomor id jurusan, yakni sebagai *foreign key* dari tabel **jurusans**.

Tabel **mahasiswa**:

- ✓ **id**: sebagai primary key.
- ✓ **nim**: berisi 8 digit nomor NIM mahasiswa seperti "10148052".
- ✓ **nama**: berisi nama mahasiswa seperti "Jane Oktaviani" atau "Eman Saragih".
- ✓ **jurusans_id**: berisi nomor id jurusan, yakni sebagai *foreign key* dari tabel **jurusans**.

Tabel **matakuliah**:

- ✓ **id**: sebagai primary key.
- ✓ **kode**: berisi 5 digit kode mata kuliah seperti "CG754".
- ✓ **nama**: berisi nama mata kuliah seperti "Basis Data" atau "Logika Informatika".
- ✓ **jumlah_sks**: berisi angka jumlah sks mata kuliah seperti 2 atau 3.
- ✓ **dosen_id**: berisi nomor id dosen yang mengajar, sebagai *foreign key* dari tabel **dosens**.
- ✓ **jurusans_id**: berisi nomor id jurusan, yakni sebagai *foreign key* dari tabel **jurusans**.

Tabel **mahasiswa_matakuliah**:

- ✓ **id**: sebagai primary key.
- ✓ **matakuliah_id**: berisi nomor id mata kuliah, yakni *foreign key* dari tabel **matakuliah**.
- ✓ **mahasiswa_id**: berisi nomor id mahasiswa, yakni *foreign key* dari tabel **mahasiswa**.

Inilah struktur database yang akan kita rancang. Silahkan pelajari sebentar diagram ERD di atas, terutama hubungan antar setiap tabel (garis-garis penghubung antara kolom *primary key* dengan kolom *foreign key*).

Total kita akan membuat 5 buah tabel: **jurusans**, **dosens**, **mahasiswa**, **matakuliah** dan **mahasiswa_matakuliah**. Tabel terakhir ini diperlukan sebagai tabel pivot dari hubungan *many to many* antara tabel **mahasiswa** dengan tabel **matakuliah**.

Untuk project yang sebenarnya, hasil diagram ERD ini bisa di diskusikan kembali dengan client. Bahas hubungan yang terjadi antara setiap tabel serta nama-nama kolom yang disimpan. Bisa jadi dalam tahap ini masih ada revisi atau client meminta tambahan kolom baru.

Jika sudah tidak ada masalah, kita lanjut ke pekerjaan teknis untuk membuat tabel-tabel ini.

20.3. Membuat File Migration

Jika menggunakan PHP native, proses pembuatan tabel bisa dilakukan secara langsung dari phpMyAdmin atau menulis perintah query dari cmd MySQL client. Namun karena kita menggunakan Laravel, proses ini lebih pas ditangani oleh *migration*.

Selain membuat file migration, nantinya kita juga butuh file pendukung lain seperti *model*, *controller*, *seeder*, dan *factory*. Menggunakan perintah `php artisan`, semua file bisa di generate dengan cepat.

Silahkan buka cmd, masuk ke folder instalasi Laravel, lalu jalankan 4 perintah berikut secara bergantian:

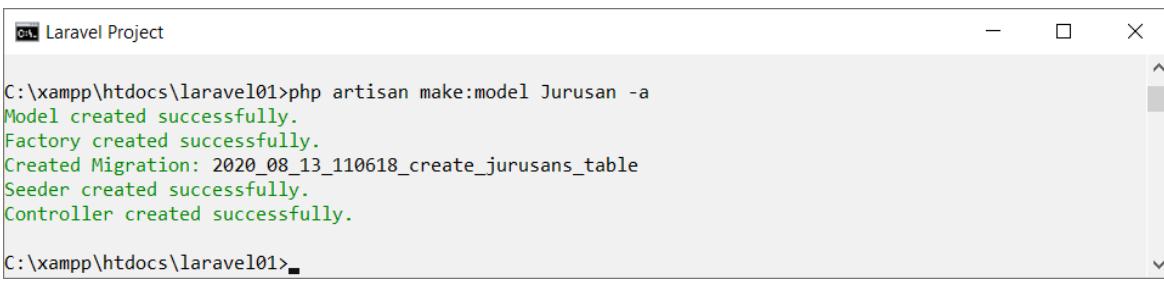
```
php artisan make:model Jurusan -a  
php artisan make:model Mahasiswa -a  
php artisan make:model Dosen -a  
php artisan make:model Matakuliah -a
```

Perintah `php artisan make:model -a` akan men-generate semua file yang terhubung dengan model, yakni: **migration**, **seeder**, **factory**, dan **resource controller**. Option `-a` sendiri merupakan singkatan dari "all".

Sebagai contoh, perintah **php artisan make:model Jurusan a** akan menggenerate 5 file berikut:

- ✓ File model: App\Models\Jurusan.php
- ✓ File migration: database\migrations\<timestamp>_create_jurusans_table.php
- ✓ File seeder: database\seeders\JurusanSeeder.php
- ✓ File factory: database\factories\JurusanFactory.php
- ✓ File controller: app\Http\Controllers\JurusanController.php

Untuk file controller, itu sudah langsung berisi method resource untuk pemrosesan **CRUD**, yakni kerangka method seperti `index()`, `create()`, `store()` dan `destroy()`.

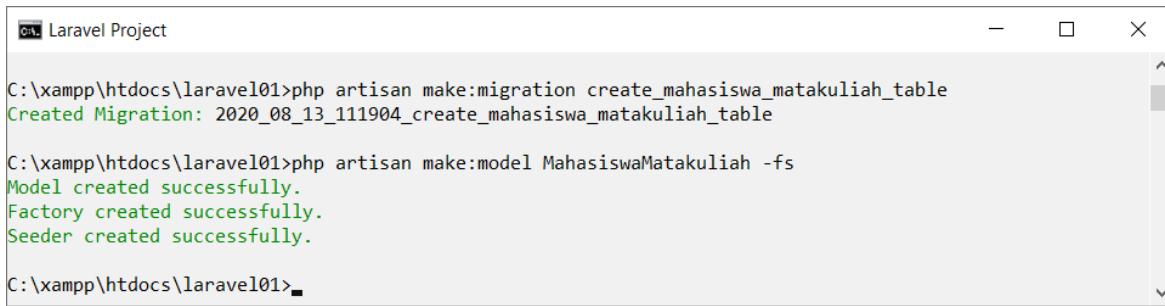


```
C:\xampp\htdocs\laravel01>php artisan make:model Jurusan -a  
Model created successfully.  
Factory created successfully.  
Created Migration: 2020_08_13_110618_create_jurusans_table  
Seeder created successfully.  
Controller created successfully.  
C:\xampp\htdocs\laravel01>
```

Gambar: Proses pembuatan file model Jurusan beserta semua file pendukung

Untuk tabel pivot `mahasiswa_matakuliah`, harus di buat terpisah dengan kode berikut:

```
php artisan make:migration create_mahasiswa_matakuliah_table  
php artisan make:model MahasiswaMatakuliah -fs
```



```
C:\xampp\htdocs\laravel01>php artisan make:migration create_mahasiswa_matakuliah_table
Created Migration: 2020_08_13_111904_create_mahasiswa_matakuliah_table

C:\xampp\htdocs\laravel01>php artisan make:model MahasiswaMatakuliah -fs
Model created successfully.
Factory created successfully.
Seeder created successfully.

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

Gambar: Proses pembuatan file migration, model, factory dan seeder mahasiswa_matakuliah

Perintah pertama dipakai untuk membuat tabel `mahasiswa_matakuliah`, sedangkan perintah kedua berguna untuk men-generate file `model`, `factory` dan `seeder` **MahasiswaMatakuliah**. File ini diperlukan karena akan kita pakai untuk proses seeder nantinya.

20.4. Generate Tabel (migration)

Setelah semua file tersedia, saatnya masuk ke migration untuk mendefinisikan struktur tabel.

Berikut kode migration untuk tabel `jurusans`:

```
database\migrations\<timestamp>_create_jurusans_table.php
```

```
1 public function up()
2 {
3     Schema::create('jurusans', function (Blueprint $table) {
4         $table->id();
5         $table->string('nama');
6         $table->string('kepala_jurusan');
7         $table->integer('daya_tampung');
8         $table->timestamps();
9     });
10 }
```

Berikut kode migration untuk tabel `dosens`:

```
database\migrations\<timestamp>_create_dosens_table.php
```

```
1 public function up()
2 {
3     Schema::create('dosens', function (Blueprint $table) {
4         $table->id();
5         $table->char('nid',8)->unique();
6         $table->string('nama');
7         $table->foreignId('jurusan_id')->constrained()->onDelete('cascade');
8         $table->timestamps();
9     });
10 }
```

Berikut kode migration untuk tabel `mahasiswas`:

database\migrations\<timestamp>_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->foreignId('jurusan_id')->constrained()->onDelete('cascade');
8         $table->timestamps();
9     });
10 }
```

Berikut kode migration untuk tabel matakuliahs:

database\migrations\<timestamp>_create_matakuliahs_table.php

```
1 public function up()
2 {
3     Schema::create('matakuliahs', function (Blueprint $table) {
4         $table->id();
5         $table->char('kode',5)->unique();
6         $table->string('nama');
7         $table->integer('jumlah_sks');
8         $table->foreignId('dosen_id')->constrained()->onDelete('cascade');
9         $table->foreignId('jurusan_id')->constrained()->onDelete('cascade');
10        $table->timestamps();
11    });
12 }
```

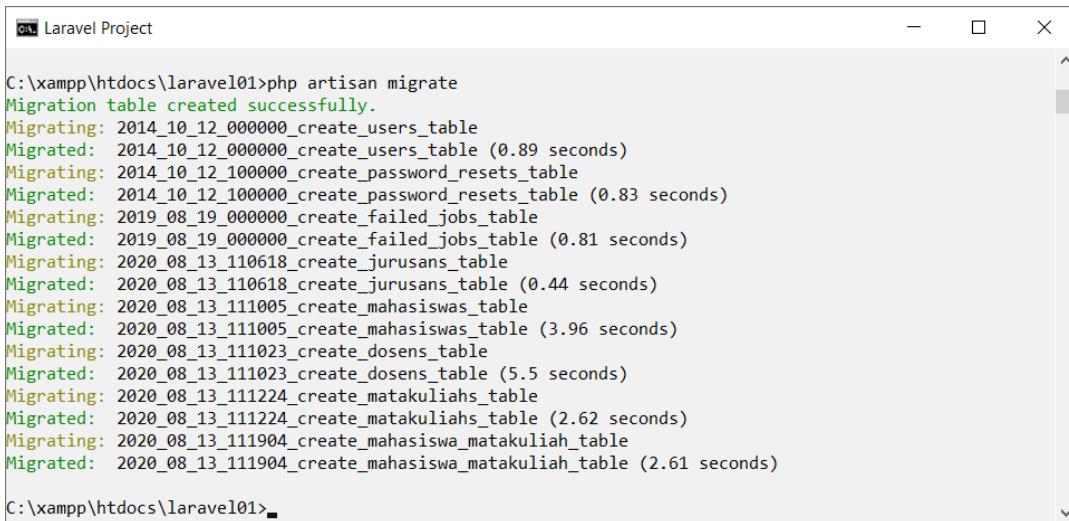
Dan berikut kode migration untuk tabel mahasiswa_matakuliah:

database\migrations\<timestamp>_create_mahasiswa_matakuliah_table.php

```
1 public function up()
2 {
3     Schema::create('mahasiswa_matakuliah', function (Blueprint $table) {
4         $table->id();
5         $table->foreignId('mahasiswa_id')->constrained()->onDelete('cascade');
6         $table->foreignId('matakuliah_id')->constrained()->onDelete('cascade');
7         $table->timestamps();
8     });
9 }
```

Dari kelima file migration ini, semuanya sudah kita pelajari. Untuk seluruh *foreign key*, langsung saya set dengan tambahan method `onDelete('cascade')` supaya jika tabel utama dihapus, data yang berhubungan di tabel kedua juga akan ikut terhapus.

Simpan semua file dan langsung test dengan menjalankan migration dengan perintah `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 (0.89 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.83 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.81 seconds)
Migrating: 2020_08_13_110618_create_jurusans_table
Migrated: 2020_08_13_110618_create_jurusans_table (0.44 seconds)
Migrating: 2020_08_13_111005_create_mahasiswa_table
Migrated: 2020_08_13_111005_create_mahasiswa_table (3.96 seconds)
Migrating: 2020_08_13_111023_create_dosens_table
Migrated: 2020_08_13_111023_create_dosens_table (5.5 seconds)
Migrating: 2020_08_13_111224_create_matakuliahs_table
Migrated: 2020_08_13_111224_create_matakuliahs_table (2.62 seconds)
Migrating: 2020_08_13_111904_create_mahasiswa_matakuliah_table
Migrated: 2020_08_13_111904_create_mahasiswa_matakuliah_table (2.61 seconds)

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

Gambar: Menjalankan migration

Pastikan tidak ada error. Apabila diperlukan, bisa juga cek langsung ke database dari cmd MySQL client atau lewat phpMyAdmin.

20.5. Generate Isi Tabel (factory dan seeder)

Tahapan berikutnya adalah proses generate sample data tabel menggunakan **seeder**. Langkah ini bersifat opsional dan sebenarnya tidak berpengaruh langsung ke aplikasi kita. Namun memiliki data awal akan memudahkan pembuatan kode program serta bisa langsung di demo-kan ke client.

Agar lebih lengkap dan sebagai materi latihan, saya akan pakai "paket lengkap" **factory + seeder**. Dengan demikian alur proses generate data adalah dari file factory, kemudian ke file seeder, dan terakhir dijalankan dari file master seeder (`DatabaseSeeder.php`).

Kita berangkat dari factory untuk model **Jurusan** terlebih dahulu:

database\factories\JurusanFactory.php

```
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Jurusan;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class JurusanFactory extends Factory
9 {
10     protected $model = Jurusan::class;
11
12     public function definition()
13     {
14         $daftar_jurusan = ["Ilmu Komputer",
15                            "Sistem Informasi",
```

```

16             "Teknik Informatika"];
17     return [
18         'nama' => $this->faker->unique()->randomElement($daftar_jurusan),
19         'kepala_jurusan' => "Dr. ".$this->faker->firstName." ".
20                               $this->faker->lastName,
21         'daya_tampung'   => $this->faker->numberBetween(5,8)*10,
22     ];
23 }
24 }
```

Di baris 14 saya membuat array `$daftar_jurusan`. Array ini berisi 3 string: "Ilmu Komputer", "Sistem Informasi", dan "Teknik Informatika". Array `$daftar_jurusan` akan di acak menggunakan method `$this->faker->unique()->randomElement($daftar_jurusan)` dan menjadi nilai awal untuk kolom `nama` di baris 18.

Di baris 19, kolom `kepala_jurusan` di generate dari gabungan "Dr. ".\$this->faker->`firstName`." ".\$this->faker->`lastName`. Sehingga nama setiap kepala jurusan akan memiliki gelar doktor.

Pada baris 21, kolom `daya_tampung` di buat dengan perintah `$this->faker->numberBetween(5,8)*10`. Artinya, daya tampung jurusan akan random antara 50, 60, 70 dan 80.

Supaya lebih mudah di dipahami, berikut hasil proses generate tabel `jurusans` nantinya:

	id	nama	kepala_jurusan	daya_tampung	created_at	updated_at
1	1	Ilmu Komputer	Dr. Mustofa Simanjuntak	60	2020-08-15 11:24:34	2020-08-15 11:24:34
2	2	Sistem Informasi	Dr. Marsito Purnawati	80	2020-08-15 11:24:34	2020-08-15 11:24:34
3	3	Teknik Informatika	Dr. Ika Puspasari	50	2020-08-15 11:24:34	2020-08-15 11:24:34

Gambar: Hasil factory untuk tabel `jurusans`

Hasil di atas sebenarnya masih belum ada karena kode factory harus di panggil dari file seeder (akan kita bahas sesaat lagi).

Lanjut, berikut kode program untuk file factory model **Dosen**:

database\factories\DosenFactory.php

```

1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Dosen;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class DosenFactory extends Factory
9 {
10     protected $model = Dosen::class;
11 }
```

```

12  public function definition()
13  {
14      $daftar_titel = ["M.Kom", "M.Sc", "M.T", "M.Si"];
15
16      return [
17          'nid' => $this->faker->unique()->numerify('99#####'),
18          'nama' => $this->faker->firstName()." ".$this->faker->lastName()." ".
19                      $this->faker->randomElement($daftar_titel),
20          'jurusan_id' => $this->faker->numberBetween(1,
21                                              \App\Models\Jurusan::count()),
22      ];
23  }
24 }
```

Di baris 14 saya membuat array `$daftar_titel`. Array ini berisi 4 string: "M.Kom", "M.Sc", "M.T", dan "M.Si". Array `$daftar_titel` akan di pakai untuk membuat titel nama dosen di baris 18 – 19.

Kolom `nid` dosen di generate menggunakan perintah Faker `$this->faker->unique()->numerify('99#####')` pada baris 17. Tanda # nantinya akan diganti Faker dengan angka acak.

Sedikit teknik baru ada di baris 20 – 21. Di sini saya mengisi nilai kolom `jurusany_id` dengan perintah `$this->faker->numberBetween(1, \App\Models\Jurusan::count())`.

Sebagaimana yang pernah kita bahas di bab Faker, method `numberBetween()` mengembalikan satu angka acak dengan *range* yang tertulis sebagai argument. Dalam kode di atas, range tersebut mulai dari angka 1 sampai `\App\Models\Jurusan::count()`.

Perintah `\App\Models\Jurusan::count()` adalah kode eloquent. Betul, kode eloquent bisa diakses tidak hanya dari controller atau route saja, tapi bisa dari file Laravel lain seperti factory dan seeder.

Nantinya, kode factory untuk proses generate data tabel jurusans akan saya jalankan terlebih dahulu, setelah itu baru file factory untuk tabel dosens. Dengan demikian, tabel jurusans sudah berisi beberapa data. Perintah `\App\Models\Jurusan::count()` dipakai untuk mencari total jumlah data yang terdapat di tabel jurusans tadi.

Sebagai contoh, jika tabel jurusans sudah di generate dan berisi 3 data, maka perintah `$this->faker->numberBetween(1, \App\Models\Jurusan::count())` akan di proses sebagai `$this->faker->numberBetween(1, 3)`. Dengan teknik ini, kode factory tabel dosens menjadi lebih fleksibel karena kita tidak harus menginput manual angka untuk `jurusany_id`.

Berikut tampilan hasil proses generate tabel dosens nantinya:

The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM dosen;". The output is a table with the following data:

id nid	nama	jurusan_id	created_at	updated_at
1 99605319	Queen Suryatmi M.T	2	2020-08-15 11:24:35	2020-08-15 11:24:35
2 99750967	Rahmi Prasasta M.T	2	2020-08-15 11:24:35	2020-08-15 11:24:35
3 99812991	Putu Prasetya M.Sc	2	2020-08-15 11:24:35	2020-08-15 11:24:35
4 99574701	Edward Prabowo M.Sc	1	2020-08-15 11:24:35	2020-08-15 11:24:35
5 99611023	Halima Mansur M.Sc	2	2020-08-15 11:24:35	2020-08-15 11:24:35
6 99629495	Darmanto Agustina M.T	3	2020-08-15 11:24:36	2020-08-15 11:24:36

Gambar: Hasil factory untuk tabel dosen

Kemudian, kita masuk ke file factory model **Mahasiswa**:

database\factories\MahasiswaFactory.php

```

1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Mahasiswa;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class MahasiswaFactory extends Factory
9 {
10     protected $model = Mahasiswa::class;
11
12     public function definition()
13     {
14         return [
15             'nim'      => $this->faker->unique()->numerify('10#####'),
16             'nama'    => $this->faker->firstName." ".$this->faker->lastName,
17             'jurusan_id' => $this->faker->numberBetween(1,
18                                         \App\Models\Jurusan::count()),
19         ];
20     }
21 }
```

Kode factory ini mirip seperti kode untuk model Dosen, dimana kolom nim berasal dari hasil perintah `$this->faker->unique()->numerify('10#####')`, kolom nama dari `$this->faker->firstName." ".$this->faker->lastName`, serta kolom jurusan_id dari perintah `$this->faker->numberBetween(1, \App\Models\Jurusan::count())`.

Berikut tampilan hasil proses generate tabel mahasiswas nantinya:

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	jurusan_id	created_at	updated_at
1 10137962	Dimas Puspasari	2	2020-08-15 11:24:36	2020-08-15 11:24:36
2 10985559	Jail Manullang	3	2020-08-15 11:24:37	2020-08-15 11:24:37
3 10120628	Eva Permata	3	2020-08-15 11:24:37	2020-08-15 11:24:37
4 10545068	Makuta Siregar	2	2020-08-15 11:24:37	2020-08-15 11:24:37
5 10579439	Ika Hassanah	1	2020-08-15 11:24:38	2020-08-15 11:24:38
6 10510422	Icha Natsir	1	2020-08-15 11:24:38	2020-08-15 11:24:38

Gambar: Hasil factory untuk tabel mahasiswa

Lanjut ke file factory model **Matakuliah**:

database\factories\MatakuliahFactory.php

```

1  <?php
2
3  namespace Database\Factories;
4
5  use App\Models\Matakuliah;
6  use Illuminate\Database\Eloquent\Factories\Factory;
7
8  class MatakuliahFactory extends Factory
9  {
10     protected $model = Matakuliah::class;
11
12     public function definition()
13     {
14         $daftar_matakuliah= [
15             "Matematika", "Fisika Dasar", "Kimia Dasar", "Pengantar Rekayasa & Desain",
16             "Pengenalan Teknologi Informasi", "Bahasa Inggris", "Olah Raga",
17             "Pengantar Rekayasa & Desain", "Tata Tulis Karya Ilmiah",
18             "Pengantar Analisis Rangkaian", "Dasar Pemrograman",
19             "Algoritma & Struktur Data", "Matematika Diskrit", "Logika Informatika",
20             "Probabilitas & Statistika", "Aljabar Geometri",
21             "Organisasi & Arsitektur Komputer",
22             "Pemrograman Berorientasi Objek", "Strategi Algoritma",
23             "Teori Bahasa Formal dan Otomata", "Sistem Operasi", "Basis Data",
24             "Dasar Rekayasa Perangkat Lunak", "Pengembangan Aplikasi Berbasis Web",
25             "Pengembangan Aplikasi pada Platform Khusus", "Jaringan Komputer",
26             "Manajemen Proyek Perangkat Lunak", "Manajemen Basis Data",
27             "Interaksi Manusia & Komputer", "Inteligensi Buatan", "Agama dan Etika",
28             "Sistem Paralel dan Terdistribusi", "Sistem Informasi",
29             "Proyek Perangkat Lunak", "Grafika Komputer",
30             "Socio-Informatika dan Profesionalisme",
31             "Wawasan Teknologi & Komunikasi Ilmiah", "Algoritma & Struktur Data",
32             "Bahasa Inggris", "Pengantar Sistem Operasi", "Arsitektur SI/TI Perusahaan",
33             "Kepemimpinan & Ketrampilan Interpersonal", "Statistika",
34             "Desain & Manajemen Proses Bisnis", "Desain & Manajemen Jaringan Komputer",
35             "Dasar-Dasar Pengembangan Perangkat Lunak", "Pengantar Basis Data",
36             "Pemrograman Berorientasi Objek", "Analisis & Desain Perangkat Lunak",
37             "Interaksi Manusia & Komputer", "Keamanan Aset Informasi",
38             "Desain Basis Data", "Pemrograman Berbasis Web", "Sistem Cerdas",
39             "Konstruksi & Pengujian Perangkat Lunak", "Tata Tulis Ilmiah",
40             "Manajemen Proyek TI", "Riset Operasi", "Simulasi Sistem",
41             "Manajemen Resiko TI", "Perencanaan Sumber Daya Perusahaan",
42             "Manajemen Layanan TI", "Tata Kelola TI"];
43
44         $jurusan_id = $this->faker->numberBetween(1, \App\Models\Jurusan::count());
45         $array_dosen = \App\Models\Dosen::where('jurusan_id', $jurusan_id)->get('id');
46
47         return [
48             'kode' => strtoupper($this->faker->unique()->bothify('??###')),
49             'nama' => $this->faker->randomElement($daftar_matakuliah),
50             'jumlah_sks' => $this->faker->numberBetween(1,4),

```

```
51     'jurusan_id' => $jurusan_id,
52     'dosen_id' => $this->faker->randomElement($array_dosen),
53 ];
54 }
55 }
```

Kode factory cukup panjang karena terdapat array `$daftar_matakuliah` antara baris 14 – 42. Array ini berisi berbagai nama mata kuliah dan akan diacak untuk nilai kolom `nama` di baris 49.

Kolom kode di generate melalui perintah `strtoupper($this->faker->unique()->bothify('??###'))`. Method `unique()` dipakai agar tidak ada kode yang berulang. Method `bothify()` akan mengganti karakter "?" menjadi huruf, serta karakter "#" menjadi angka. Hasil perintah faker ini kemudian dilewatkan ke function `strtoupper()` bawaan PHP agar menjadi huruf besar.

Kolom `jumlah_sks` di generate menggunakan perintah `$this->faker->numberBetween(1,4)`, sehingga akan menghasilkan angka acak antara 1, 2, 3 atau 4.

Nilai untuk kolom `jurusان_id` di buat terlebih dahulu di baris 44 dari perintah `$this->faker->numberBetween(1, \App\Models\Jurusan::count())`. Hasilnya disimpan ke variabel `$jurusan_id` yang juga dipakai di baris 51. Nilai ini menunjukkan ke jurusan apa sebuah mata kuliah terdaftar.

Kolom `dosen_id` dipakai untuk menampung nilai id dari dosen yang mengajar mata kuliah. Di sini kita perlu membahas logika terkait ketergantungan mata kuliah ke sebuah jurusan. Saya ingin agar suatu mata kuliah terdaftar di satu jurusan tertentu, dan dosen yang mengajar mata kuliah tersebut juga harus berasal dari jurusan yang sama.

Dengan demikian, kita tidak bisa langsung mengisi nilai `dosen_id` dengan perintah `$this->faker->numberBetween(1, App\Models\Dosen::count())`, karena itu akan mengacak semua id dosen. Saya ingin agar mata kuliah yang terdaftar di jurusan "Ilmu Komputer", juga di ajar oleh dosen yang berasal dari jurusan "Ilmu Komputer".

Solusinya adalah membuat variabel `$array_dosen` di baris 45. Array ini berisi hasil dari perintah `App\Models\Dosen::where('jurusan_id', $jurusan_id)->get('id')`. Ini merupakan perintah eloquent yang mengembalikan kumpulan nomor `id` dari semua dosen di jurusan `$jurusan_id`.

Nilai `id` yang tersimpan di dalam `$array_dosen` kemudian diacak menggunakan perintah `$this->faker->randomElement($array_dosen)`, dan inilah yang menjadi nilai dari kolom `dosen_id`.

Penjelasan di atas memang agak rumit dan akan saya bahas ulang dengan sebuah contoh.

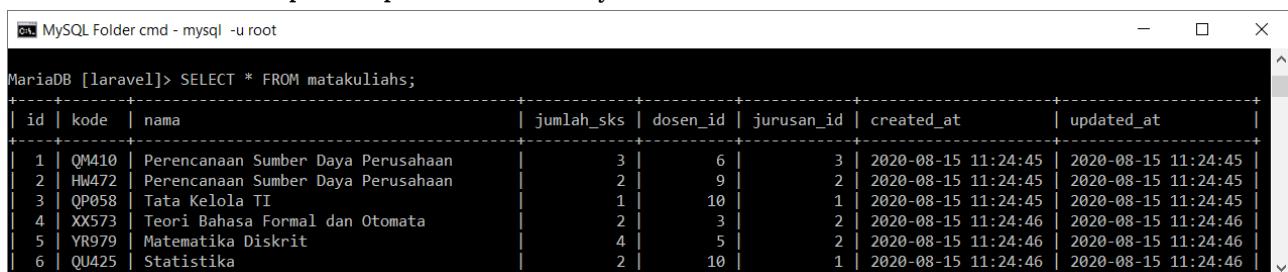
Misalkan pada saat proses generate, variabel `$jurusan_id` di baris 44 berisi angka 3 (id untuk jurusan "Teknik Informatika"). Variabel `$jurusan_id` ini kemudian dipakai sebagai nilai kolom `jurusان_id` di baris 51. Ini berarti mata kuliah yang sedang di generate akan terdaftar di jurusan "Teknik Informatika" (jurusan dengan id = 3).

Ketika masuk ke baris 45, dijalankan perintah eloquent App\Models\Dosen::where('jurusan_id',3)->get('id'). Perintah ini akan mencari semua id dosen yang juga berasal dari jurusan id = 3. Misalnya hasil kode ini terdapat 4 dosen, maka variabel \$array_dosen akan berisi [2,3,5,8], yakni kumpulan id dosen yang juga terdaftar di jurusan id = 3.

Terakhir di baris 52, kolom dosen_id akan berisi salah satu angka yang berasal dari perintah \$this->faker->randomElement(2,3,5,8). Dengan demikian, mata kuliah yang di generate akan diajar oleh dosen dari jurusan yang sama.

Proses generate data faker untuk tabel yang saling terhubung seperti ini memang cukup kompleks, namun bisa menjadi ajang latihan untuk melatih logika programming.

Berikut contoh hasil penerapan kode factory di atas ke dalam tabel matakuliah:



The screenshot shows a MySQL command-line interface window titled "MySQL Folder cmd - mysql -u root". The command entered is "SELECT * FROM matakuliah;". The output is a table with the following data:

id	kode	nama	jumlah_sks	dosen_id	jurusan_id	created_at	updated_at
1	QM410	Perencanaan Sumber Daya Perusahaan	3	6	3	2020-08-15 11:24:45	2020-08-15 11:24:45
2	Hw472	Perencanaan Sumber Daya Perusahaan	2	9	2	2020-08-15 11:24:45	2020-08-15 11:24:45
3	QP058	Tata Kelola TI	1	10	1	2020-08-15 11:24:45	2020-08-15 11:24:45
4	XX573	Teori Bahasa Formal dan Otomata	2	3	2	2020-08-15 11:24:46	2020-08-15 11:24:46
5	YR979	Matematika Diskrit	4	5	2	2020-08-15 11:24:46	2020-08-15 11:24:46
6	QU425	Statistik	2	10	1	2020-08-15 11:24:46	2020-08-15 11:24:46

Gambar: Hasil factory untuk tabel matakuliah

Terakhir, kita masuk ke factory dari model **MahasiswaMatakuliah**:

database\factories\MahasiswaMatakuliahFactory.php

```

1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\MahasiswaMatakuliah;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class MahasiswaMatakuliahFactory extends Factory
9 {
10     protected $model = MahasiswaMatakuliah::class;
11
12     public function definition()
13     {
14         $mahasiswa_id = $this->faker->numberBetween(1,
15                                         \App\Models\Mahasiswa::count());
16
17         $jurusan_mahasiswa_id = \App\Models\Mahasiswa::find($mahasiswa_id)
18                         ->jurusan_id;
19
20         $array_matakuliah = \App\Models\Matakuliah::where('jurusan_id',
21                                         $jurusan_mahasiswa_id)->get('id');
22
23         return [
24             'mahasiswa_id' => $mahasiswa_id,

```

```

25     'matakuliah_id' => $this->faker->randomElement($array_matakuliah),
26 ];
27 }
28 }
```

Tabel `mahasiswa_matakuliah` berfungsi sebagai tabel pivot untuk menyimpan mata kuliah yang diambil oleh setiap mahasiswa. Prinsipnya, mahasiswa hanya bisa mengambil mata kuliah yang terdaftar di jurusan yang sama dengan jurusannya saat ini. Oleh sebab itu kita harus buat batasan logika ini.

Di baris 14 – 15 saya mencari id mahasiswa yang akan didaftarkan. Id ini didapat dari perintah `$this->faker->numberBetween(1, App\Models\Mahasiswa::count())` yang mengembalikan sebuah angka acak antara 1 hingga jumlah total mahasiswa. Angka ini juga menjadi nilai untuk kolom `mahasiswa_id` di baris 24.

Di baris 17, perintah eloquent `App\Models\Mahasiswa::find($mahasiswa_id)->jurusan_id` dipakai untuk mencari id jurusan dari mahasiswa yang di generate tadi.

Setelah id jurusan mahasiswa di dapat, nilai tersebut dipakai untuk mencari daftar mata kuliah dari jurusan yang sama. Inilah yang dilakukan oleh kode program di baris 20 – 21. Hasilnya, variabel `$array_matakuliah` akan berisi kumpulan id mata kuliah dari jurusan yang sama dengan jurusan mahasiswa saat ini.

Terakhir, kolom `matakuliah_id` akan diisi dengan salah satu nilai acak dari `$array_matakuliah`.

Berikut contoh hasil penerapan kode factory ke dalam tabel `mahasiswa_matakuliah`:

id	mahasiswa_id	matakuliah_id	created_at	updated_at
1	48	10	2020-08-15 11:24:53	2020-08-15 11:24:53
2	34	49	2020-08-15 11:24:53	2020-08-15 11:24:53
3	17	17	2020-08-15 11:24:53	2020-08-15 11:24:53
4	26	36	2020-08-15 11:24:54	2020-08-15 11:24:54
5	39	43	2020-08-15 11:24:54	2020-08-15 11:24:54
6	12	26	2020-08-15 11:24:54	2020-08-15 11:24:54

Gambar: Hasil factory untuk tabel `mahasiswa_matakuliah`

Kelima file factory sudah selesai dibuat, sekarang saatnya masuk ke file **seeder**. Kode untuk seeder relatif singkat karena proses generate data sudah ada di file factory. Dalam file seeder kita tinggal menginput jumlah data yang akan di generate. Berikut kode program untuk semua file seeder:

`database\seeders\JurusanSeeder.php`

```

1 public function run()
2 {
3     \App\Models\Jurusan::factory()->count(3)->create();
4 }
```

database\seeders\DosenSeeder.php

```
1 public function run()
2 {
3     \App\Models\Dosen::factory()->count(10)->create();
4 }
```

database\seeders\MahasiswaSeeder.php

```
1 public function run()
2 {
3     \App\Models\Mahasiswa::factory()->count(50)->create();
4 }
```

database\seeders\MatakuliahSeeder.php

```
1 public function run()
2 {
3     \App\Models\Matakuliah::factory()->count(50)->create();
4 }
```

database\seeders\MahasiswaMatakuliahSeeder.php

```
1 public function run()
2 {
3     \App\Models\MahasiswaMatakuliah::factory()->count(200)->create();
4 }
```

Untuk mempersingkat kode program, saya langsung menampilkan isi method run() saja. Dari kelima kode ini akan di generate 3 jurusan, 10 dosen, 50 mahasiswa, 50 matakuliah serta 200 data hubungan mahasiswa dengan matakuliah.

Namun pekerjaan belum selesai, semua file seeder harus di panggil lagi dari file master seeder, yakni file DatabaseSeeder.php:

database\seeders\DatabaseSeeder.php

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class DatabaseSeeder extends Seeder
8 {
9     public function run()
10    {
11        $faker = \Faker\Factory::create('id_ID');
12        $faker->seed(123);
13
14        $this->call(JurusanSeeder::class);
15        $this->call(DosenSeeder::class);
16        $this->call(MahasiswaSeeder::class);
17    }
18 }
```

```
17     $this->call(MatakuliahSeeder::class);
18     $this->call(MahasiswaMatakuliahSeeder::class);
19 }
20 }
```

Agar kode yang digenerate menggunakan bahasa Indonesia, kita juga harus mengubah pengaturan 'faker_locale' di file config\app.php. Silahkan buka file ini, lalu tukar baris 'faker_locale' => 'en_US' menjadi 'faker_locale' => 'id_ID'.

Kemudian masih ada satu lagi pengaturan yang harus di tambah, yakni menyangkut nama tabel `mahasiswa_matakuliah`.

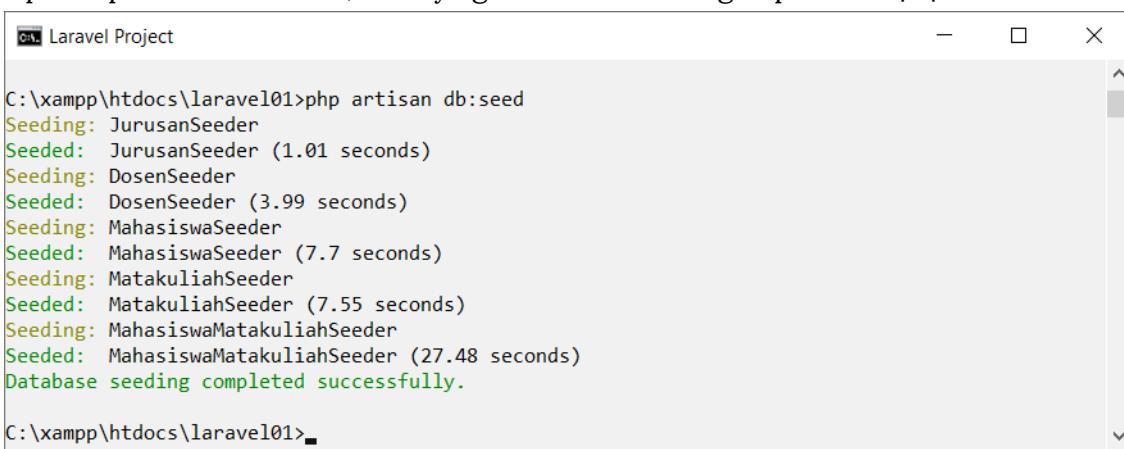
Secara default, factory dan seeder akan mengisi data ke tabel `mahasiswa_matakuliah`s (nama dalam bentuk plural), padahal nama tabel menggunakan versi singular tanpa tambahan 's'. Untuk memberitahu Laravel mengenai nama tabel ini, tambah property `protected $table = 'mahasiswa_matakuliah'` ke dalam file app\Models\MahasiswaMatakuliah.php:

app\Models\MahasiswaMatakuliah.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 MahasiswaMatakuliah extends Model
9 {
10     use HasFactory;
11     protected $table = 'mahasiswa_matakuliah';
12 }
```

Dengan tambahan property di atas, seeder akan mengisi data ke tabel `mahasiswa_matakuliah`, bukan ke tabel `mahasiswa_matakuliah`s.

Semua persiapan sudah selesai, saatnya generate data dengan perintah `php artisan db:seed`:



```
C:\xampp\htdocs\laravel01>php artisan db:seed
Seeding: JurusanSeeder
Seeded: JurusanSeeder (1.01 seconds)
Seeding: DosenSeeder
Seeded: DosenSeeder (3.99 seconds)
Seeding: MahasiswaSeeder
Seeded: MahasiswaSeeder (7.7 seconds)
Seeding: MatakuliahSeeder
Seeded: MatakuliahSeeder (7.55 seconds)
Seeding: MahasiswaMatakuliahSeeder
Seeded: MahasiswaMatakuliahSeeder (27.48 seconds)
Database seeding completed successfully.

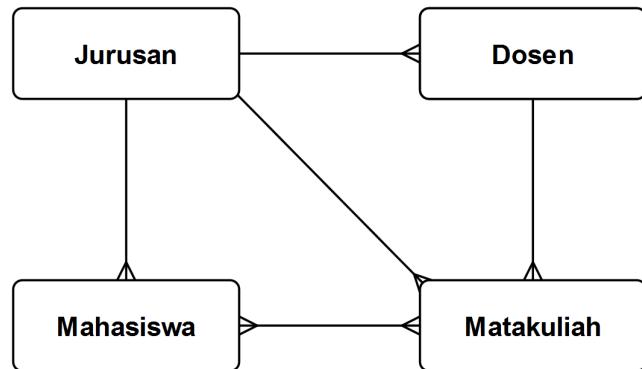
C:\xampp\htdocs\laravel01>
```

Gambar: Proses seeder ke 5 tabel

Proses seeder akan berlangsung beberapa saat. Ini butuh waktu sedikit lama karena banyak kode yang harus dijalankan. Setelah selesai, periksa database dari cmd MySQL client atau phpMyAdmin untuk memastikan semua tabel sudah berisi data.

20.6. Pendefinisian Relationship

Sampai di sini kita sudah selesai membuat tabel beserta file model. Sekarang saatnya mendefinisikan *relationship* untuk setiap tabel. Sebagai patokan, berikut saya tampilkan kembali diagram ERD database:



Gambar: Diagram ERD Sistem Informasi Universitas Ilkoom

Dari diagram ini bisa terlihat hubungan yang terjadi, yakni **one to many relationship** untuk Jurusan->Dosen, Jurusan->Mahasiswa, Jurusan->Matakuliah dan Dosen->Matakuliah, serta **many to many relationship** untuk Mahasiswa->Dosen.

Berikut relationship untuk semua file model:

app\Models\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     public function dosens()
12     {
13         return $this->hasMany('App\Models\Dosen');
14     }
15
16     public function mahasiswas()
17     {
18         return $this->hasMany('App\Models\Mahasiswa');
19     }
}
```

```
20     public function matakuliah()
21     {
22         return $this->hasMany('App\Models\Mahasiswa');
23     }
24 }
```

app\Models\Dosen.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 Dosen extends Model
9 {
10    use HasFactory;
11    public function matakuliah()
12    {
13        return $this->hasMany('App\Models\Matakuliah');
14    }
15
16    public function jurusan()
17    {
18        return $this->belongsTo('App\Models\Jurusan');
19    }
20 }
```

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    public function jurusan()
12    {
13        return $this->belongsTo('App\Models\Jurusan');
14    }
15
16    public function matakuliah()
17    {
18        return $this->belongsToMany('App\Models\Matakuliah')->withTimestamps();
19    }
20 }
```

app\Models\Matakuliah.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 Matakuliah extends Model
9 {
10     use HasFactory;
11     public function jurusan()
12     {
13         return $this->belongsTo('App\Models\Jurusan');
14     }
15
16     public function dosen()
17     {
18         return $this->belongsTo('App\Models\Dosen');
19     }
20
21     public function mahasiswa()
22     {
23         return $this->hasMany('App\Models\Mahasiswa')->withTimestamps();
24     }
25 }
```

Beberapa nama method menggunakan kata plural (dengan akhiran 's') untuk hubungan *one to many* seperti `dosens()`, `mahasiswa()` dan `matakuliah()` di model **Jurusan**. Sebagian lagi menggunakan kata singular (tanpa akhiran 's') untuk hubungan *many to one* seperti `jurusan()` dan `dosen()` di model **Matakuliah**.

Tipsnya, jika hubungan tersebut bisa mengembalikan banyak data, pakai versi plural. Namun jika hanya bisa mengembalikan satu data, pakai nama singular. Penamaan ini sebenarnya tidak wajib, lebih ke saran penulisan saja.

Khusus file model MahasiswaMatakuliah tidak perlu diisi karena hanya dipakai untuk proses seeder.

Dalam bab ini kita telah selesai membuat 5 tabel untuk aplikasi Sistem Informasi Universitas ILKOOOM. Kemudian mengenerate data menggunakan factory + seeder, serta mendefinisikan hubungan antar tabel.

Dalam bab selanjutnya, akan dibahas tentang pembuatan file view untuk menampilkan data tabel tabel di web browser.

21. Sistem Informasi Universitas ILKOOOM: Read

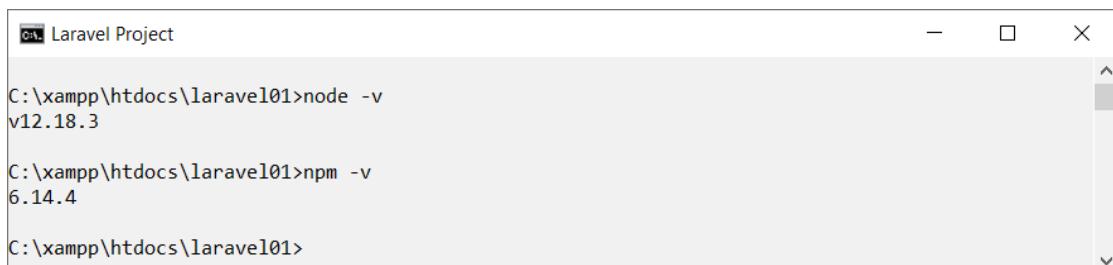
Dalam bab sebelumnya kita sudah selesai membuat tabel serta men-generate data awal untuk Sistem Informasi Universitas Ilkom. Kali ini akan dilanjutkan ke proses perancangan bagian **Read** dari CRUD, yakni kode program untuk menampilkan data.

21.1. Instalasi Laravel UI dan Laravel Mix

Jika sudah berhubungan dengan front-end, kita perlu menginstall modul tambahan seperti Laravel UI dan Laravel Mix. Materi tentang keduanya sudah saya bahas di buku **Laravel Uncover** sehingga kita bisa langsung masuk ke praktek.

Sebelum itu pastikan juga telah menginstall **Node.js**. Jika belum, bisa download dari <https://nodejs.org>.

Untuk memeriksa apakah nodejs dan npm sudah terinstall, silahkan buka cmd lalu jalankan perintah `node -v` dan `npm -v`. Jika tampil angka seperti gambar di bawah ini, artinya nodejs dan npm sudah tersedia.



```
C:\xampp\htdocs\laravel01>node -v
v12.18.3

C:\xampp\htdocs\laravel01>npm -v
6.14.4

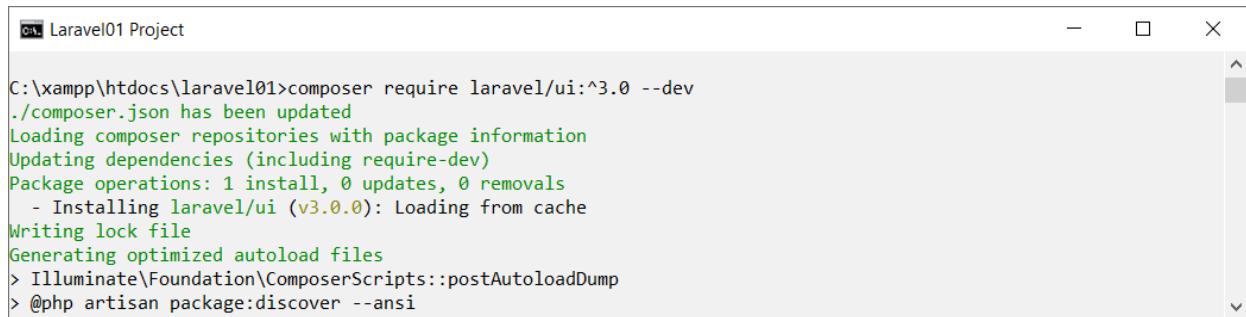
C:\xampp\htdocs\laravel01>
```

Gambar: Periksa versi nodejs dan npm

Dalam beberapa kasus, bisa saja proses instalasi Laravel UI dan Laravel Mix gagal berjalan. Penyebab utama adalah koneksi internet yang kurang lancar, atau bisa juga karena versi Nodejs atau composer yang belum update.

Lanjut masuk ke folder instalasi laravel dan install **Laravel UI** dengan perintah berikut:

```
composer require laravel/ui:^3.0 --dev
```

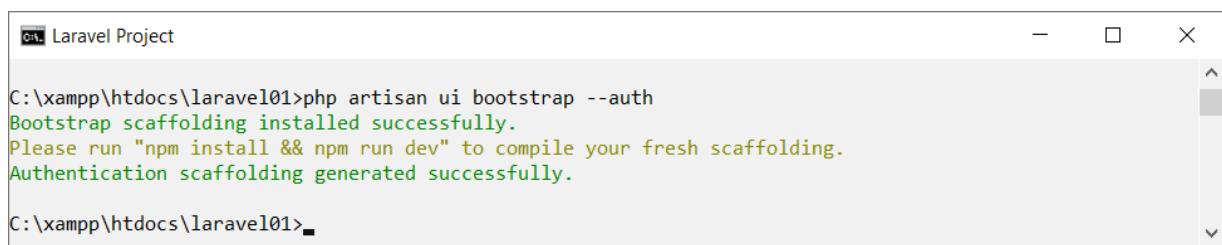


```
C:\xampp\htdocs\laravel01>composer require laravel/ui:^3.0 --dev
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing laravel/ui (v3.0.0): Loading from cache
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
```

Gambar: Proses instalasi Laravel UI

Setelah itu install **Laravel Authentication**:

```
php artisan ui bootstrap --auth
```



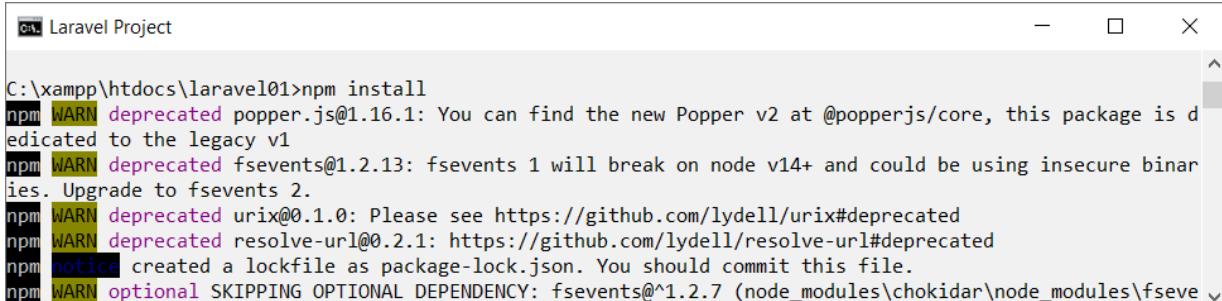
```
C:\xampp\htdocs\laravel01>php artisan ui bootstrap --auth
Bootstrap scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.

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

Gambar: Proses instalasi Laravel Authentication

Lanjutkan dengan proses **install npm**:

```
npm install
```



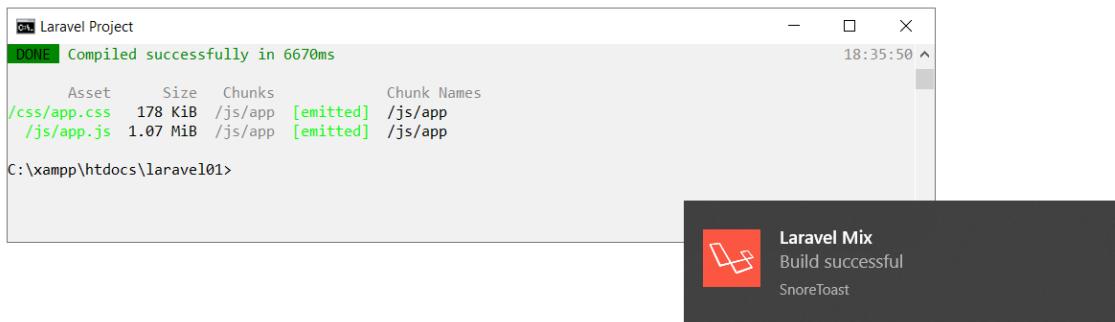
```
C:\xampp\htdocs\laravel01>npm install
npm WARN deprecated popper.js@1.16.1: You can find the new Popper v2 at @popperjs/core, this package is dedicated to the legacy v1
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fsevents 2.
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.2.7 (node_modules\chokidar\node_modules\fsevents)
```

Gambar: Menjalankan npm install

Seperti biasa, proses instalasi ini butuh waktu yang cukup lama mengingat ada sekitar 100MB file yang di download ke folder node_modules. Kemungkinan akan terdapat beberapa pesan warning (npm WARN), namun itu tidak masalah selama bukan pesan error.

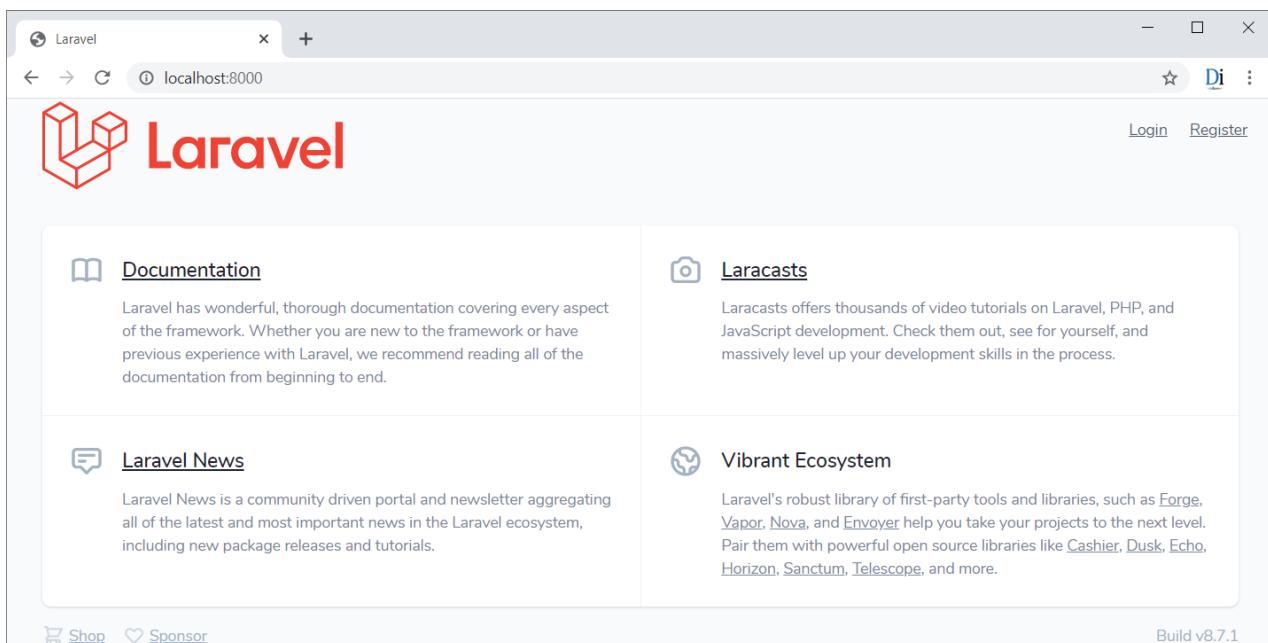
Setelah instalasi selesai, test **compile file assets** dengan perintah:

```
npm run dev
```



Gambar: Proses compile file assets berhasil

Terakhir akses halaman <http://localhost:8000> di web browser:



Gambar: Tampilan halaman <http://localhost:8000> Laravel

Dengan tampilnya menu login dan register di sudut kanan atas, berarti proses instalasi npm, Laravel UI dan Laravel Mix sudah berhasil.

21.2. Persiapan File Asset

Untuk proses design tampilan, saya akan memakai framework CSS Bootstrap serta beberapa icon dari Font Awesome. File Bootstrap sudah tersedia saat proses instalasi Laravel UI sebelumnya sehingga tidak perlu langkah lanjutan.

Untuk file Font Awesome, akan kita install menggunakan **npm**. Silahkan buka cmd, masuk ke folder instalasi Laravel lalu jalankan perintah berikut:

```
npm install @fortawesome/fontawesome-free
```

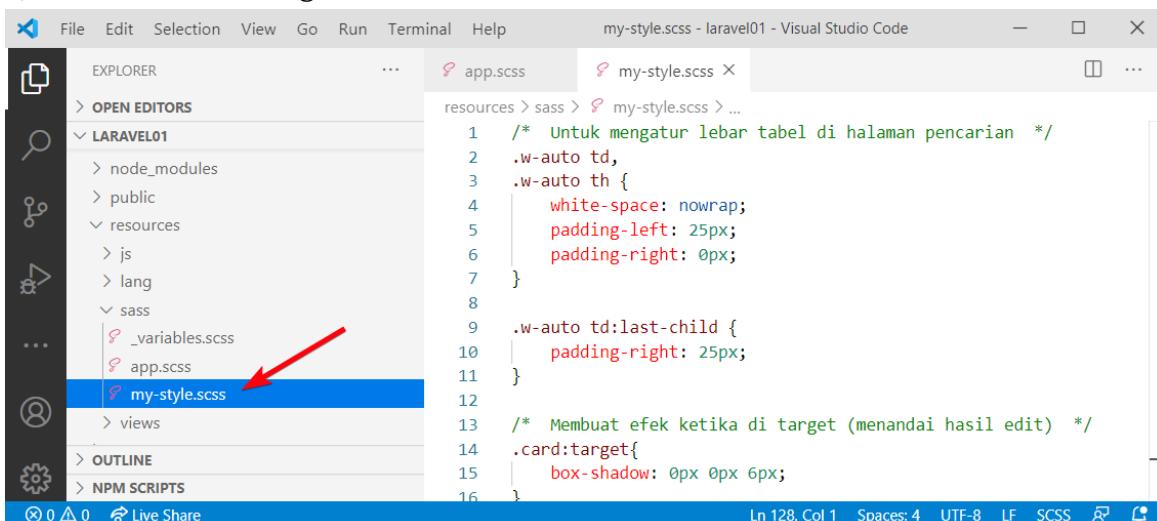


```
C:\xampp\htdocs\laravel01>npm install @fortawesome/fontawesome-free
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules\watchpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
```

Gambar: Instalasi font awesome menggunakan npm

Proses instalasi akan berjalan beberapa saat. Perintah ini akan menambah satu folder baru bernama `@fortawesome/fontawesome-free` ke dalam folder `node_modules`.

Setelah selesai, lanjut dengan membuka folder `resources\sass\`, lalu buat satu file bernama `my-style.scss` dan isi dengan kode berikut:



Gambar: Buat file my-style.scss

`resources\sass\my-style.scss`

```
1 /* Untuk mengatur lebar tabel di halaman pencarian */
2 .w-auto td,
3 .w-auto th {
4     white-space: nowrap;
5     padding-left: 25px;
6     padding-right: 0px;
7 }
8
9 .w-auto td:last-child {
10     padding-right: 25px;
11 }
12
13 /* Membuat efek ketika di target (menandai hasil edit) */
14 .card:target{
15     box-shadow: 0px 0px 6px;
16 }
17
18 tr:target{
```

```
19     background-color: lightyellow !important;
20 }
21
22 /* Style untuk navbar */
23 #main-navbar .nav-link{
24   color: #212529;
25   text-transform:uppercase;
26 }
27 #main-navbar .nav-link:hover {
28   color: #9e7b1c;
29 }
30 #main-navbar .container {
31   border-bottom: 2px solid black;
32 }
33
34 .main-logo {
35   height: 70px;
36 }
37 .small-logo {
38   height: 60px; /* Untuk logo kecil di footer */
39 }
40
41 /* Untuk membuat button hover di halaman jurusan (card) */
42 .card .btn-action {
43   position: absolute;
44   top: 10px;
45   right: 10px;
46 }
47
48 .card .btn-action {
49   opacity: 0;
50 }
51
52 .card:hover .btn-action {
53   opacity: 0.7;
54 }
55
56 .card .btn-action:hover {
57   opacity: 1.0 !important;
58 }
59
60 .card .btn-hapus {
61   border-top-left-radius: 0;
62   border-bottom-left-radius: 0;
63   margin-left: -1px;
64 }
65
66 /* Media query agar tombol selalu tampil di layar kecil */
67 @media (max-width: 700px) {
68   .btn-action {
69     opacity: 0.7;
70   }
71 }
72
73 @media (hover: none) {
```

```
74     .btn-action {
75         opacity: 0.7;
76     }
77 }
78
79 /* Mengubah warna button bawaan Bootstrap */
80 .btn-info, .btn-primary, .btn-secondary,
81 .btn-info:focus, .btn-primary:focus, .btn-secondary:focus {
82     background-color: #1f3f7a;
83     border-color: #1f3f7a;
84 }
85 .btn-info:hover, .btn-primary:hover, .btn-secondary:hover {
86     background-color: #0a2351;
87     border-color: #0a2351;
88 }
89 .btn-info:focus, .btn-primary:focus, .btn-secondary:focus {
90     box-shadow: 0 0 0 0.2rem rgba(31, 63, 122, 0.5) !important;
91 }
92 .btn-info:active,.btn-primary:active, .btn-info:active {
93     background-color: #0a2351 !important;
94 }
95 .btn-success {
96     background-color: #b99737;
97     border-color: #b99737;
98 }
99 .btn-success:hover {
100    background-color: #9e7b1c;
101    border-color: #9e7b1c;
102 }
103 .btn-success:focus {
104    box-shadow: 0 0 0 0.2rem rgba(185, 151, 55, 0.5) !important;
105 }
106 .btn-success:active {
107    background-color: #b99737 !important;
108 }
109
110 /* Menukar warna ketika di select */
111 ::selection {
112     background: #0a2351;
113     color: white;
114 }
115
116 /* Menukar warna ketika di select */
117 table td a{
118     text-decoration: none;
119 }
120
121 /* Style untuk footer */
122 #main-footer {
123     background-color: #172c53 !important;
124 }
```

File `my-style.scss` ini berisi kode CSS yang akan dipakai untuk design tampilan. Meskipun ber-ekstensi `.scss`, kode yang ada hanya perintah CSS biasa.

Agar file font awesome dan file `my-style.scss` ikut di compile oleh Laravel Mix, modifikasi file `resources\sass\app.scss` menjadi berikut:

```
resources\sass\app.scss
```

```
1 // Bootstrap
2 @import '~bootstrap/scss/bootstrap';
3
4 // Font Awesome
5 @import '~@fortawesome/fontawesome-free/scss/fontawesome';
6 @import '~@fortawesome/fontawesome-free/scss/regular';
7 @import '~@fortawesome/fontawesome-free/scss/solid';
8 @import '~@fortawesome/fontawesome-free/scss/brands';
9
10 // Custom Style
11 @import 'my-style';
```

Pada bagian atas file `app.scss` sebenarnya ada beberapa baris kode Sass bawaan Laravel, yakni perintah `@import font` dan `@import 'variables'`. Namun karena tidak kita butuhkan, kode tersebut boleh dihapus.

Di baris 2 terdapat perintah import file bootstrap, yang diikuti import file font awesome di baris 5 – 8, serta import file `my-style.scss` yang baru saja kita buat di baris 11.

Save file di atas dan compile file assets menggunakan perintah: `npm run dev`

Asset	Size	Chunks	Chunk Names
/css/app.css	254 KiB	/js/app	[emitted]
/js/app.js	1.07 MiB	/js/app	[emitted]
fonts/vendor/@fortawesome/fontawesome-free/webfa-brands-400.eot?0fabbb6606be4c45acfeedd115d0caca4	131 KiB		[emitted]
fonts/vendor/@fortawesome/fontawesome-free/webfa-brands-400.svg?ccfdb9dc442be0c629d331e94497428b	713 KiB		[emitted]
fonts/vendor/@fortawesome/fontawesome-free/webfa-brands-400.ttf?085b1dd8427dbeff110bd55410915a3f6	131 KiB		[emitted]

Gambar: Hasil compile assets dengan perintah `npm run dev`

Jika keluar tampilan di atas, artinya file asset sudah sukses di compile.

Tahapan selanjutnya adalah menginput file gambar. Saya sudah menyediakan 3 buah file gambar yang terdiri dari 2 logo dan 1 favicon.

Gambar ini bisa diakses dari file `belajar_laravel_in_depth_1.zip` yang ada di Google Drive. Extract file tersebut, lalu masuk ke folder "Bab 16 - Sistem Informasi Universitas ILKOOOM (Read)", kemudian copy folder **img** ke folder **public** di instalasi Laravel.

Setelah selesai, periksa folder `public/img/` dan pastikan terdapat 3 buah gambar berikut:



Gambar: File favicon dan logo

Sampai di sini file assets kita sudah lengkap, saatnya masuk ke perancangan view.

21.3. Membuat View Template (app.blade.php)

Setelah menginstall Laravel UI, di dalam folder resources\views akan muncul 2 folder baru, yakni folder **auth** yang berisi berbagai file view form authentication, serta folder **layout** yang berisi file app.blade.php.

Secara bawaan, file app.blade.php berperan sebagai "view induk" yang akan di-extends atau diturunkan ke view – view lain (terutama ke file auth). Bahasan mendalam tentang cara menurunkan view sudah kita bahas di buku **Laravel Uncover**, yakni menggunakan pasangan perintah @yield dan @extends.

Dalam project ini saya tetap memakai file app.blade.php yang nantinya berisi kode untuk bagian header dan footer. Silahkan modifikasi file app.blade.php dengan kode berikut:

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      <!-- CSRF Token -->
7      <meta name="csrf-token" content="{{ csrf_token() }}>
8      <title>Sistem Informasi Universitas ILKOOOM</title>
9      <link rel="icon" href="{{ asset('img/favicon.png') }}" type="image/png">
10     <!-- Scripts -->
11     <script src="{{ asset('js/app.js') }}" defer></script>
12     <!-- Styles -->
13     <link href="{{ asset('css/app.css') }}" rel="stylesheet">
14 </head>
15 <body>
16
17     <!-- NAVBAR -->
18     <nav id="main-navbar" class="navbar navbar-expand-lg navbar-light
19         bg-white py-3">
20         <div class="container pb-3">
```

```

21 <a class="navbar-brand" href="{{ url('/') }}>
22   <span class="d-none" >ILKOOM</span>
23   
25   
27 </a>
28 <button class="navbar-toggler" type="button" data-toggle="collapse"
29   data-target="#navbarNav">
30   <span class="navbar-toggler-icon"></span>
31 </button>
32 <div class="collapse navbar-collapse" id="navbarNav">
33   <ul class="navbar-nav m-auto">
34     <li class="nav-item ">
35       <a class="nav-link px-4" href="{{ route('jurusans.index') }}>
36         Jurusan</a>
37     </li>
38     <li class="nav-item">
39       <a class="nav-link px-4" href="{{ route('dosens.index') }}>
40         Dosen</a>
41     </li>
42     <li class="nav-item">
43       <a class="nav-link px-4" href="{{ route('mahasiswa.index') }}>
44         Mahasiswa</a>
45     </li>
46     <li class="nav-item">
47       <a class="nav-link px-4" href="{{ route('matakuliahs.index') }}>
48         MataKuliah</a>
49     </li>
50     <li class="nav-item">
51       <a class="nav-link px-4" href="{{ url('/pencarian') }}>Search</a>
52     </li>
53   </ul>
54   <!-- Right Side Of Navbar -->
55   <ul class="navbar-nav ml-auto">
56     <!-- Authentication Links -->
57     @guest
58       <li class="nav-item">
59         <a class="nav-link px-4" href="{{ route('login') }}>
60           {{ __('Login') }}</a>
61       </li>
62     @else
63       <li class="nav-item dropdown">
64         <a id="navbarDropdown" class="nav-link dropdown-toggle px-4"
65           href="#" role="button" data-toggle="dropdown" v-pre>
66           {{ Auth::user()->name }} <span class="caret"></span>
67         </a>
68         <div class="dropdown-menu dropdown-menu-right">
69           <a class="dropdown-item px-4" href="{{ route('logout') }}"
70             onclick="event.preventDefault();
71               document.getElementById('logout-form').submit();">
72             {{ __('Logout') }}</a>
73           <form id="logout-form" action="{{ route('logout') }}"
74             method="POST" style="display: none;">
```

```

76          @csrf
77      </form>
78      </div>
79      </li>
80      @endguest
81  </ul>
82  </div>
83</div>
84</nav>
85
86<div class="container mt-3" style="min-height:550px">
87  <div class="row">
88    <div class="col-12">
89
90      @yield('content')
91
92    </div>
93  </div>
94</div>
95
96<!-- FOOTER -->
97<footer id="main-footer" class="text-white bg-dark py-4 mt-5">
98  <div class="container">
99    <div class="row">
100      <div class="col-md-3 text-center text-md-left">
101        <a href="{{ url('/') }}>
102          
104        </a>
105        <p>Lorem ipsum dolor sit amet veniam consectetur adipisicing elit.
106           Aperiam cumque, esse modi maxime.
107         </p>
108      </div>
109      <div class="col-md-3 text-center d-none d-md-inline">
110        <h5>Information</h5>
111        <ul class="list-unstyled">
112          <li><a href="{{ route('jurusans.index') }}" class="text-white">
113            Jurusan</a></li>
114          <li><a href="{{ route('dosens.index') }}" class="text-white">
115            Dosen</a></li>
116          <li><a href="{{ route('mahasiswa.index') }}" class="text-white">
117            Mahasiswa</a></li>
118          <li><a href="{{ route('matakuliahs.index') }}" class="text-white">
119            Mata Kuliah</a></li>
120        </ul>
121      </div>
122      <div class="col-md-3 text-center">
123        <h5>Our Services</h5>
124        <ul class="list-unstyled">
125          @guest
126            @if (Route::has('register'))
127              <li>
128                <a class="text-white" href="{{ route('register') }}>
129                  {{ __('Register') }}
130                </a>

```

```

131      </li>
132      @endif
133      @endguest
134      <li><a href="#" class="text-white">Help/Contact Us</a></li>
135      <li><a href="#" class="text-white">Privacy Policy</a></li>
136      <li><a href="#" class="text-white">Terms & Conditions</a></li>
137      </ul>
138  </div>
139  <div class="col-md-3 text-center text-md-left">
140      <h5>Hubungi Kami</h5>
141      <div class="text-nnowrap"><i class="fas fa-envelope fa-fw mr-3"></i>
142          info@ilkoom.ac.id</div>
143      <div class="text-nnowrap"><i class="fas fa-phone fa-fw mr-3"></i>
144          (021) 123456</div>
145      <div class="text-nnowrap"><i class="fas fa-globe fa-fw mr-3"></i>
146          www.ilkoom.ac.id</div>
147      </div>
148  </div>
149  <div class="row mt-3 mt-md-0">
150      <div class="col-md-3 mr-md-auto text-center text-md-left">
151          <small>&copy; ILKOOM {{date("Y")}}</small>
152      </div>
153      <div id="footer-icon" class="col-md-3 text-center text-md-left">
154          <div>
155              <a href="#" class="text-white mr-1">
156                  <i class="fab fa-facebook fa-lg"></i>
157              </a>
158              <a href="#" class="text-white mr-1">
159                  <i class="fab fa-twitter fa-lg"></i>
160              </a>
161              <a href="#" class="text-white mr-1">
162                  <i class="fab fa-instagram fa-lg"></i>
163              </a>
164              <a href="#" class="text-white mr-1">
165                  <i class="fab fa-google-plus fa-lg"></i>
166              </a>
167              <a href="#" class="text-white mr-1">
168                  <i class="fab fa-github fa-lg"></i>
169              </a>
170          </div>
171      </div>
172  </div>
173 </div>
174 </footer>
175 </body>
176 </html>

```

Struktur halaman ini bisa dibagi ke dalam 4 kelompok:

- ◆ Tag <head> di baris 1 – 15
- ◆ Menu navigasi di baris 18 – 84
- ◆ Konten di baris 86 – 94
- ◆ Footer di baris 97 – 174

Kode untuk tag <head> sebenarnya masih sama seperti yang ada di file app.blade.php bawaan Laravel, plus sedikit modifikasi. Diantaranya saya mengubah isi tag <title> menjadi "Sistem Informasi Universitas ILKOOM", serta menambah tag <link> untuk menampilkan favicon.

Kemudian di baris 11 dan 13 terdapat kode untuk mengakses file assets hasil Laravel Mix, yakni js/app.js dan css/app.css.

Selanjutnya bagian menu navigasi di buat menggunakan komponen **Navbar** bawaan Bootstrap. Gambar logo ditampilkan dengan kode di baris 21 – 27, yang disambung dengan 6 menu menuju halaman **Jurusan**, **Dosen**, **Mahasiswa**, **Matakuliah**, **Search**, dan **Login**.

Perhatikan isi atribut href dari setiap menu. Beberapa menggunakan helper function route() seperti di baris 35, 39, 43, dan 47. Ini merupakan cara akses named route yang nantinya akan kita siapkan.

Saya juga memakai struktur kode bawaan Laravel untuk membuat menu **Login**. Bagian ini sudah berisi perintah @guest di baris 57 – 80 untuk percabangan menu. Ketika user belum login, menu yang tampil adalah **Login**, namun begitu user sudah login, menu akan berganti menjadi **Logout**.

Konten utama ada di baris 86 – 94, yang diawali dengan class .container, .row dan .col-12 untuk membuat Bootstrap grid. Setelah itu terdapat perintah blade @yield('content') di baris 90. Di posisi inilah nantinya file view lain akan mengisi bagian konten.

Terakhir, terdapat kode untuk footer antara baris 97 – 176. Bagian footer lebih ke dekorasi saja, yakni terdapat gambar logo, sedikit teks, beberapa menu serta icon social media.

Di dalam footer juga terdapat menu link **Register** di baris 125 – 133. Menu ini berada dalam blok kondisi @guest, sehingga hanya akan tampil bagi user yang belum login saja (user yang sudah login tentu tidak perlu register lagi).

Pada dasarnya file app.blade.php di rancang bukan untuk diakses langsung, melainkan sebagai file "master" atau template untuk view turunan nanti. Akan tetapi saya tetap ingin memeriksa apakah tampilannya sudah sesuai atau belum. Supaya bisa diakses langsung, kita perlu tambah route baru.

Silahkan lihat isi file routes\web.php yang tersedia 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();
```

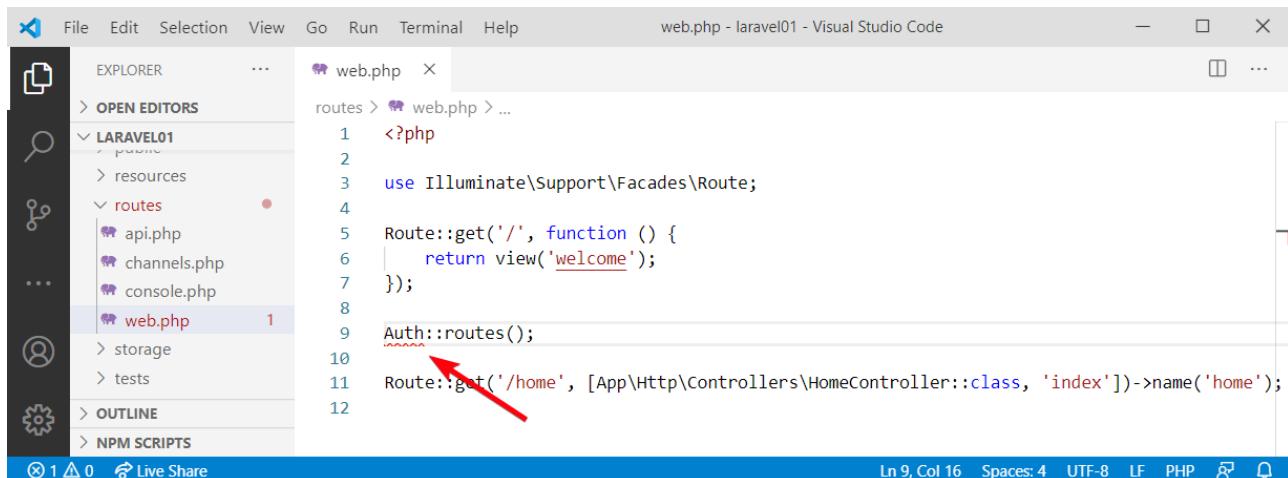
```

10
11 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
12 ->name('home');

```

Kode antara baris 1 – 7 merupakan route bawaan awal Laravel. Kemudian terdapat tambahan Auth::routes() di baris 9 serta Route::get('/home') di baris 11. Kedua route ini berasal dari proses instalasi modul authentication dari Laravel UI.

Jika anda menggunakan aplikasi VS Code dengan extension PHP Intelephense dalam keadaan aktif, baris Auth::routes() kemungkinan akan diberi tanda merah:



Gambar: Garis merah di bagian Auth

Pada baris tersebut, PHP Intelephense menebak ada sesuatu yang salah. Jika ini adalah file PHP "normal", memang akan error karena tidak ada kode yang mendefinisikan class **Auth**. Akan tetapi Laravel memiliki mekanisme sendiri untuk men-inject class bawaan tanpa harus di deklarasikan atau di import dari namespace.

Error ini bisa saja diabaikan (Laravel tetap berjalan normal), atau agar PHP Intelephense 'senang', tambah perintah use Illuminate\Support\Facades\Auth di bagian atas file route:

routes\web.php

```

1 <?php
2 use Illuminate\Support\Facades\Route;
3 use Illuminate\Support\Facades\Auth;
4 ...

```

Baik, kembali ke percobaan file app.blade.php, silahkan ubah baris `return view('welcome')` menjadi `return view('layouts.app')`. Berikut kode lengkap file route dengan perubahan ini:

routes\web.php

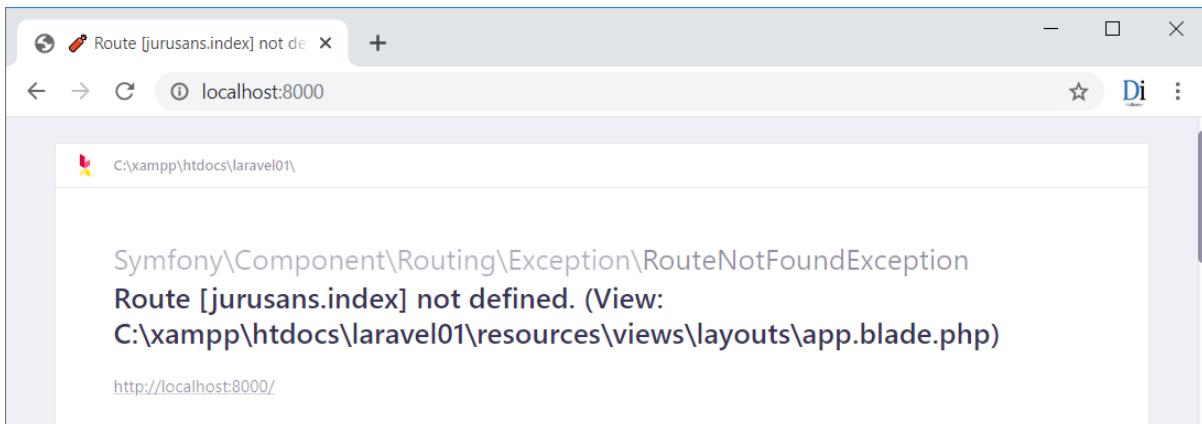
```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use Illuminate\Support\Facades\Auth;

```

```
5
6 Route::get('/', function () {
7     return view('layouts.app');
8 });
9
10 Auth::routes();
11
12 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
13 ->name('home');
```

Save file di atas, lalu akses halaman root <http://localhost:8000> :



Gambar: Error saat mengakses http://localhost:8000

Ternyata tampil error. Bisakah anda tebak kira-kira apa yang salah?

Pesan error "Route [jurusans.index] not defined" berarti Laravel menemukan ada satu route menuju jurusans.index namun route tersebut tidak terdefinisi. Route yang dimaksud ada pada bagian menu navigasi file app.blade.php, dimana saya menggunakan function route() untuk membuat link seperti ini:

```
<a class="nav-link px-4" href="{{ route('jurusans.index') }}>Jurusan</a>
```

Kode {{ route('jurusans.index') }} dipakai untuk membuat *named route*, atau "route alias". Di buku **Laravel Uncover** kita sudah bahas tentang ini, termasuk kelebihan serta kekurangan *named route*. Solusi dari error di atas adalah dengan mendefinisikan *named route* tersebut.

Kembali buka file routes\web.php, lalu modifikasi dengan tambahan route berikut:

```
routes\web.php

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use Illuminate\Support\Facades\Auth;
5
6 use App\Http\Controllers\JurusanController;
7 use App\Http\Controllers\DosenController;
8 use App\Http\Controllers\MatakuliahController;
9 use App\Http\Controllers\MahasiswaController;
```

```

10 Route::get('/', function () {
11     return view('layouts.app');
12 });
13
14 Route::resource('jurusans', JurusanController::class);
15 Route::resource('dosens', DosenController::class);
16 Route::resource('mahasiswa', MahasiswaController::class);
17 Route::resource('matakuliahs', MatakuliahController::class);
18
19 Auth::routes();
20
21 Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])
22 ->name('home');

```

Jika anda masih ingat, di awal bab sebelumnya kita membuat file controller yang sudah langsung berisi method resources. Penggunaan `Route::resource()` di baris 15 – 18 masih berhubungan dengan file tersebut dan membuat pekerjaan menjadi lebih mudah.

Satu perintah `Route::resource()` sebenarnya berisi 7 route RESTfull untuk membuat fitur CRUD. Inilah alasan kenapa saya menggunakan *named route* untuk membuat link halaman, karena sudah langsung tersedia di `Route::resource()`.

Jika anda sedikit bingung dengan penjelasan ini, boleh buka kembali buku **Laravel Uncover** dan masuk ke akhir bab **CRUD**, yakni sub-bab **Resource Controllers**. Di sana terdapat penjelasan bahwa satu perintah `Route::resource()` berisi 7 route RESTfull yang sudah dilengkapi *named route*.

Sebagai contoh, perintah `Route::resource('mahasiswa', MahasiswaController::class)` akan berisi semua route berikut:

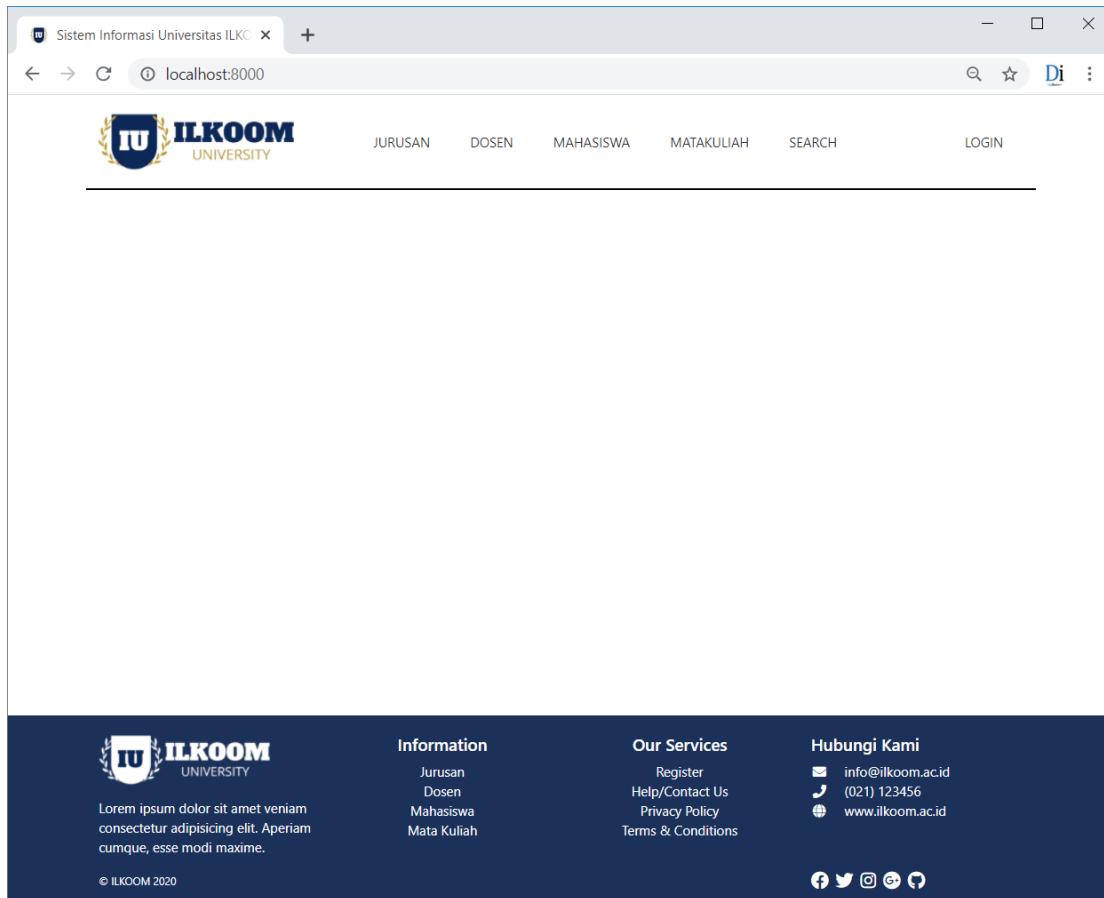
```

1 Route::get('/mahasiswa', [MahasiswaController::class, 'index'])
2 ->name('mahasiswa.index');
3
4 Route::get('/mahasiswa/create', [MahasiswaController::class, 'create'])
5 ->name('mahasiswa.create');
6
7 Route::post('/mahasiswa', [MahasiswaController::class, 'store'])
8 ->name('mahasiswa.store');
9
10 Route::get('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'show'])
11 ->name('mahasiswa.show');
12
13 Route::get('/mahasiswa/{mahasiswa}/edit', [MahasiswaController::class, 'edit'])
14 ->name('mahasiswa.edit');
15
16 Route::patch('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'update'])
17 ->name('mahasiswa.update');
18
19 Route::delete('/mahasiswa/{mahasiswa}', [MahasiswaController::class, 'destroy'])
20 ->name('mahasiswa.destroy');

```

Dalam bab ini kita akan fokus ke 2 route saja, yakni `mahasiswa.index` untuk menampilkan semua data, serta `mahasiswa.show` untuk menampilkan satu data. Route lain akan dibahas dalam bab-bab berikutnya.

Save file route tersebut, lalu akses kembali halaman `localhost:8000`:



Gambar: Tampilan view layouts\app.blade.php

Sip, tampilan `app.blade.php` sudah sempurna, termasuk logo favicon. File assets (kode CSS) juga sukses diakses karena jika tidak, halaman akan berantakan.

21.4. Menampilkan Semua Data Jurusan

Sekarang saatnya masuk ke cara menampilkan semua data tabel `jurusans`. Saat ini kita sudah memiliki 4 file controller untuk masing-masing tabel, pemrosesan untuk tabel `jurusans` akan ditulis ke file `JurusanController.php`.

Silahkan buka file ini dan seharusnya sudah ada beberapa `method resources`. Kita akan mengikuti aturan pembuatan CRUD dengan metode RESTfull, dimana kode program untuk menampilkan semua data tabel ditulis dalam method `index()`:

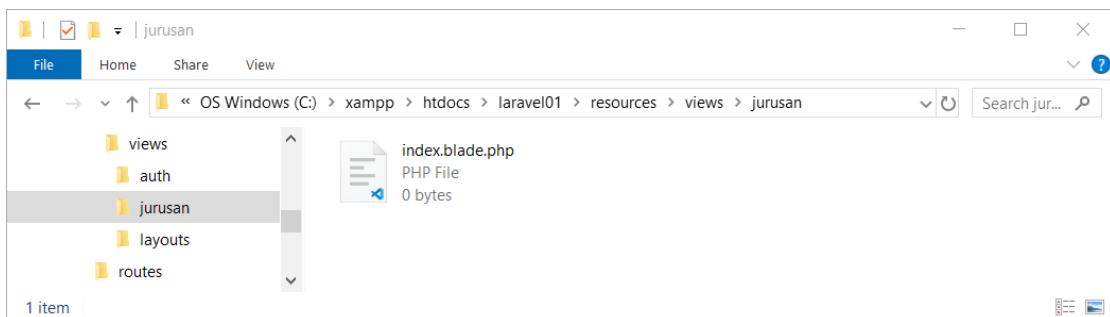
app\Http\Controllers\JurusanController.php

```
1 public function index()
2 {
3     $jurusans = Jurusan::withCount('mahasiswa')->orderBy('nama')->get();
4     return view('jurusan.index',[ 'jurusans' => $jurusans]);
5 }
```

Di baris 3 saya mengambil semua isi tabel jurusans yang di urutkan berdasarkan kolom nama. Method `withCount('mahasiswa')` dipakai karena dalam file view nanti saya ingin menampilkan total jumlah mahasiswa untuk setiap jurusan. Hasil perintah ini disimpan ke variabel `$jurusans` untuk kemudian dikirim ke view `jurusans.index` di baris 4.

Jika terdapat perintah `return view('jurusan.index')`, view yang dimaksud merujuk ke file `index.blade.php` yang berada di folder `jurusans`. Saat ini file dan folder tersebut belum tersedia dan mari kita buat.

Silahkan buka folder `resources\views`, lalu buat sebuah folder baru bernama **jurusans**. Nantinya, semua view RESTfull untuk tabel `jurusans` akan berada di dalam folder ini. Masuk ke folder `jurusans` tersebut dan buat satu file baru bernama `index.blade.php`.



Gambar: Buat file `index.blade.php` di dalam folder `resources\views\jurusans`

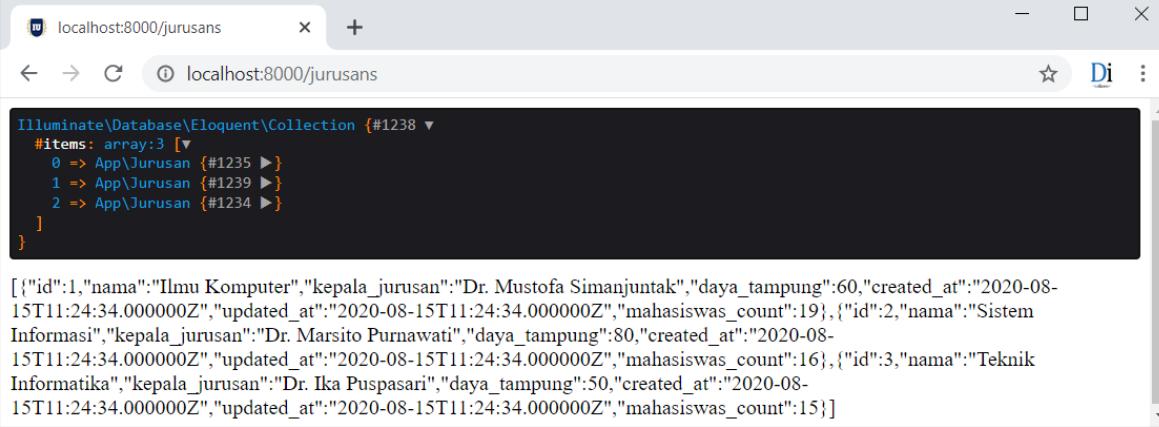
Buka file `index.blade.php` dan ketik kode berikut:

`resources\views\jurusans\index.blade.php`

```
1 {{ dump($jurusans) }}
```

Perintah ini sekedar memastikan bahwa variabel `$jurusans` yang berasal dari controller sudah sampai ke file view. Langkah ini sangat amat penting karena akan lebih mudah mencari sumber error sekarang daripada menunda uji coba setelah membuat semua kode view.

View `jurusans\index.blade.php` bisa diakses dari alamat `localhost:8000/jurusans`. Alamat URL ini sudah tersedia dari penulisan `Route::resource('jurusans',...)`:



```

Illuminate\Database\Eloquent\Collection {#1238 ▾
  #items: array:3 [▼
    0 => App\Jurusan {#1235 ▶}
    1 => App\Jurusan {#1239 ▶}
    2 => App\Jurusan {#1234 ▶}
  ]
}

[{"id":1,"nama":"Ilmu Komputer","kepala_jurusan":"Dr. Mustofa Simanjuntak","daya_tampung":60,"created_at":"2020-08-15T11:24:34.000000Z","updated_at":"2020-08-15T11:24:34.000000Z","mahasiswa_count":19}, {"id":2,"nama":"Sistem Informasi","kepala_jurusan":"Dr. Marsito Purnawati","daya_tampung":80,"created_at":"2020-08-15T11:24:34.000000Z","updated_at":"2020-08-15T11:24:34.000000Z","mahasiswa_count":16}, {"id":3,"nama":"Teknik Informatika","kepala_jurusan":"Dr. Ika Puspasari","daya_tampung":50,"created_at":"2020-08-15T11:24:34.000000Z","updated_at":"2020-08-15T11:24:34.000000Z","mahasiswa_count":15}]

```

Gambar: Hasil dari perintah {{ dump(\$jurusans) }}

Jika tampil hasil di atas, artinya variabel \$jurusans sudah bisa diakses dan berisi 3 object model Jurusan dalam bentuk collection. Namun jika tampil pesan error atau array kosong, maka ada sesuatu yang salah di method index() sebelumnya.

Selain tampilan dump() yang biasa terlihat, di bagian bawah terdapat teks yang cukup panjang. Teks ini merupakan string JSON yang digenerate Laravel karena kita memakai tanda {{ }} untuk menampilkan hasil dump().

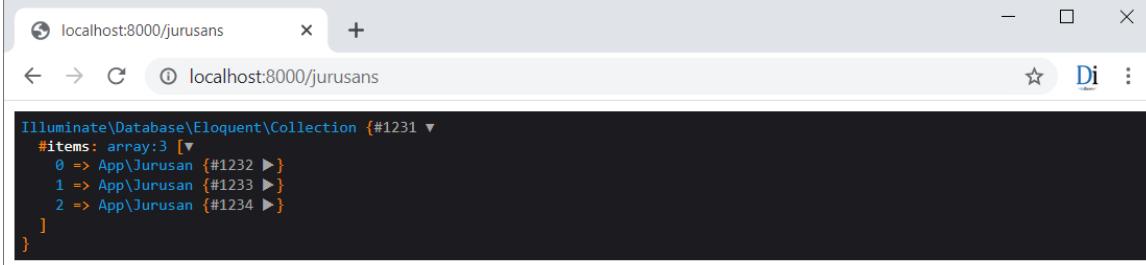
Tambahan string JSON ini boleh diabaikan, atau jalankan perintah dump(\$jurusans) dengan tag PHP biasa agar teks JSON tidak ikut tampil:

resources\views\jurusan\index.blade.php

```

1 <?php
2   dump($jurusans);

```



```

Illuminate\Database\Eloquent\Collection {#1231 ▾
  #items: array:3 [▼
    0 => App\Jurusan {#1232 ▶}
    1 => App\Jurusan {#1233 ▶}
    2 => App\Jurusan {#1234 ▶}
  ]
}

[{"id":1,"nama":"Ilmu Komputer","kepala_jurusan":"Dr. Mustofa Simanjuntak","daya_tampung":60,"created_at":"2020-08-15T11:24:34.000000Z","updated_at":"2020-08-15T11:24:34.000000Z","mahasiswa_count":19}, {"id":2,"nama":"Sistem Informasi","kepala_jurusan":"Dr. Marsito Purnawati","daya_tampung":80,"created_at":"2020-08-15T11:24:34.000000Z","updated_at":"2020-08-15T11:24:34.000000Z","mahasiswa_count":16}, {"id":3,"nama":"Teknik Informatika","kepala_jurusan":"Dr. Ika Puspasari","daya_tampung":50,"created_at":"2020-08-15T11:24:34.000000Z","updated_at":"2020-08-15T11:24:34.000000Z","mahasiswa_count":15}]

```

Gambar: Hasil dari perintah {{ dump(\$jurusans) }}

Baik, karena data \$jurusans sudah bisa diakses, saatnya rancang kode HTML dan CSS view:

resources\views\jurusan\index.blade.php

```

1 @extends('layouts.app')
2
3 @section('content')
4 <h1 class="display-4 text-center my-5">Sistem Informasi Universitas ILKOOOM</h1>
5
6 <div class="card-columns mt-3">

```

```

7  @foreach($jurusans as $jurusan)
8
9   <div class="card mt-1">
10    <div class="card-body text-center">
11      <h3 class="card-title py-1">{{ $jurusan->nama }}</h3>
12      <hr>
13
14      <div class="card-text py-1">Kepala Jurusan:
15        <b>{{$jurusan->kepala_jurusan}}</b></div>
16      <div class="card-text pb-4">Total Mahasiswa: {{$jurusan->mahasiswas_count}}
17        (daya tampung {{$jurusan->daya_tampung}})</div>
18      <a href="{{ route('jurusan-dosen', ['jurusan_id' => $jurusan->id]) }}"
19        class="btn btn-info btn-block">Dosen</a>
20      <a href="{{ route('jurusan-mahasiswa', ['jurusan_id' => $jurusan->id]) }}"
21        class="btn btn-success btn-block">Mahasiswa</a>
22    </div>
23  </div>
24
25  @endforeach
26 </div>
27 @endsection

```

Di baris 1 terdapat perintah `@extends('layouts.app')`, yang berarti view ini diturunkan dari `layouts\app.blade.php`, yakni file view "master" yang sudah kita buat sebelumnya.

Bagian konten dibuka dengan tag `<h1>` sebagai judul halaman (baris 4).

Untuk menampilkan data tabel jurusan, saya akan memakai komponen **Card** bawaan Bootstrap. Ini ditandai dengan penggunaan class `.card-columns` di baris 6.

Semua data tabel `jurusans` sudah ada di dalam variabel `$jurusans`. Karena variabel ini berbentuk collection, kita perlu perintah `@foreach($jurusans as $jurusan)` untuk membuka satu per satu object yang tersimpan.

Nama jurusan diakses dengan perintah `{{ $jurusan->nama }}` di baris 11, yang menjadi judul dari setiap komponen Card.

Di dalam Card, saya menampilkan nama kepala jurusan di baris 15, jumlah mahasiswa yang sudah terdaftar di baris 16, serta daya tampung di baris 17. Semua data-data ini sudah tersedia di object `$jurusan`.

Khusus untuk jumlah mahasiswa yang sudah terdaftar, bisa diakses dari `$jurusan->mahasiswas_count`. Property ini berasal dari tambahan method `withCount('mahasiswas')` di dalam controller sebelumnya.

Di akhir Card, saya membuat tag `<a>Dosen` dan `<a>Mahasiswa`. Kedua link ini terhubung ke URL *named route* 'jurusan-dosen' serta 'jurusan-mahasiswa'.

Idenya adalah, saya ingin menampilkan daftar semua dosen dan semua mahasiswa untuk satu jurusan tertentu. Agar fleksibel, id jurusan juga dikirim sebagai argument ke dalam *named route* tadi.

Misalnya kode berikut:

```
<a href="{{ route('jurusan-dosen',[ 'jurusan_id' => $jurusan->id]) }}>Dosen</a>
```

Ketika link di klik, web browser akan menuju URL yang di definisikan oleh *named route* 'jurusan-dosen'. Bersamaan dengan itu, data jurusan_id juga ikut dikirim.

View index.blade.php di atas masih belum bisa di akses (terjadi error) sebab terdapat penggunaan *named route* yang harus di definisikan terlebih dahulu. Silahkan buka file routes\web.php, lalu tambah 2 route berikut di bagian bawah:

routes\web.php

```
1 ...
2 Route::get('/jurusan-dosen/{jurusan_id}', [JurusanController::class,
3     'jurusanDosen'])->name('jurusan-dosen');
4 Route::get('/jurusan-mahasiswa/{jurusan_id}', [JurusanController::class,
5     'jurusanMahasiswa'])->name('jurusan-mahasiswa');
```

Dengan ini, maka ketika tombol **Dosen** di klik, akan terbuka halaman baru di alamat jurusan-dosen/{jurusan_id}.

Sebagai contoh, jurusan Ilmu Komputer memiliki id = 1. Maka ketika tombol **Dosen** di klik, akan menuju ke alamat localhost:8000/jurusan-dosen/1. Nantinya, angka "1" di akhir URL berguna untuk membatasi tampilan dosen dari jurusan Ilmu Komputer saja.

Kode view index.blade.php sudah bisa diakses. Berikut hasilnya:

The screenshot shows a web browser window with the title 'Sistem Informasi Universitas ILKOOOM'. The address bar displays 'localhost:8000/jurusans'. The page content includes a header with the ILKOOOM logo and navigation links for JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and LOGIN. Below the header, there are three main sections: 'Ilmu Komputer', 'Sistem Informasi', and 'Teknik Informatika'. Each section contains the name of the department, the head's name, the total number of students, and two buttons: 'Dosen' (blue) and 'Mahasiswa' (gold). The 'Ilmu Komputer' section has Dr. Mustofa Simanjuntak as the head and 19 students. The 'Sistem Informasi' section has Dr. Marsito Purnawati as the head and 16 students. The 'Teknik Informatika' section has Dr. Ika Puspasari as the head and 15 students.

Gambar: Tampilan halaman localhost:8000/jurusans

Tombol Dosen dan Mahasiswa masih belum berfungsi dan akan kita tambah di akhir bab nanti.

Karena memakai komponen Card milik Bootstrap, setiap jurusan akan tampil dalam kotak tersendiri. Jika nanti terdapat jurusan lain, akan muncul kotak ke-4, kotak ke-5, dst.

21.5. Menampilkan Semua Data Dosen

Untuk menampilkan semua isi tabel dosens, caranya mirip seperti data jurusan sebelumnya. Kita tetap menggunakan method `index()`, tapi kali ini di file `DosenController.php`:

app\Http\Controllers\DosenController.php

```
1 public function index()
2 {
3     $dosens = Dosen::orderBy('nama')->paginate(5);
4     return view('dosen.index',[ 'dosens' => $dosens]);
5 }
```

Di baris 3 terdapat perintah untuk mengambil semua isi tabel dosens yang di urutkan berdasarkan nama. Method `paginate(5)` saya pakai agar data dosen tampil dalam bentuk pagination, yakni 5 data pada setiap halaman. Semua data dosen kemudian disimpan ke dalam variabel `$dosens` untuk selanjutnya di kirim ke view `dosen.index`.

Karena kita memakai pagination, maka perlu mengubah sedikit isi file `app\Providers\AppServiceProvider.php`. Sebab di Laravel 8 pagination secara default ditampilkan dengan framework TailwindCSS. Agar mendukung Bootstrap, silahkan modifikasi sebagai berikut:

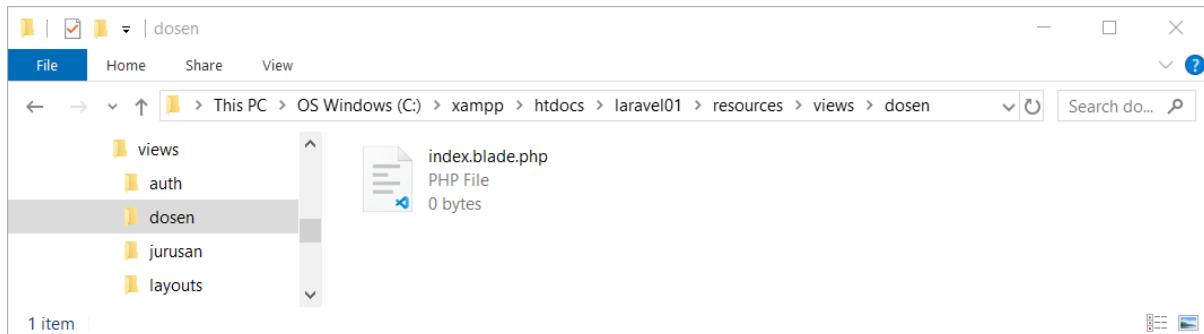
app\Providers\AppServiceProvider.php

```
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Support\ServiceProvider;
6 use Illuminate\Pagination\Paginator;
7
8 class AppServiceProvider extends ServiceProvider
9 {
10     /**
11      * Register any application services.
12      *
13      * @return void
14      */
15     public function register()
16     {
17         //
18     }
19
20     /**
21      * Bootstrap any application services.
22      *
23      * @return void
24 
```

```
24     */
25     public function boot()
26     {
27         Paginator::useBootstrap();
28     }
29 }
```

Tambahannya hanya 2 baris, yakni `use Illuminate\Pagination\Paginator` di baris 6, serta `Paginator::useBootstrap()` di baris 27.

Kembali ke menampilkan data dosen, silahkan buat folder **dosen** di dalam `resources\views`, serta file `index.blade.php` di dalam folder tersebut:

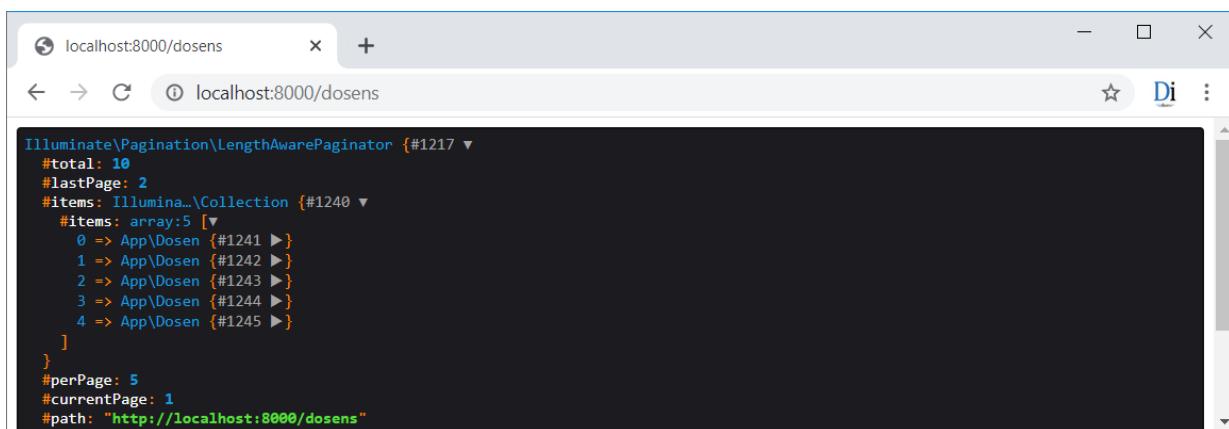


Gambar: Buat file index.blade.php di dalam folder resources\views\dosen

Sama seperti pada view jurusan, kita akan tes terlebih dahulu apakah variabel `$dosens` sudah bisa diakses atau tidak:

`resources\views\dosen\index.blade.php`

```
1 <?php
2     dump($dosens);
```



Gambar: Hasil dari perintah {{ dump(\$dosens) }}

Isi variabel `$jurusans` berasal dari pemanggilan pagination, sehingga variabel yang terkirim berbentuk instance dari object **LengthAwarePaginator** (baris paling atas). Ini tidak masalah karena seperti itulah pagination bawaan Laravel bekerja. Jika ingin melihat isi data dosen, tersedia di tab **items**.

Karena variabel \$jurusans sudah bisa diakses, kita masuk ke kode HTML, CSS dan blade untuk view index.blade.php:

resources\views\dosen\index.blade.php

```
1  @extends('layouts.app')
2  @section('content')
3
4  <h1 class="display-4 text-center my-5" id="judul">
5      Data Dosen {{ $nama_jurusan ?? 'Universitas ILKOOM' }}
6  </h1>
7
8  <table class="table table-striped">
9      <thead>
10         <tr>
11             <th>#</th>
12             <th>NID</th>
13             <th>Nama Dosen</th>
14             <th>Jurusan Dosen</th>
15         </tr>
16     </thead>
17     <tbody>
18         @foreach ($dosens as $dosen)
19         <tr>
20             <th>{{ $dosens->firstItem() + $loop->iteration - 1 }}</th>
21             <td>{{ $dosen->nid }}</td>
22             <td>
23                 <a href="{{ route('dosens.show', ['dosen' => $dosen->id]) }}>
24                     {{ $dosen->nama }}</a>
25             </td>
26             <td>{{ $dosen->jurusan->nama }}</td>
27         </tr>
28     @endforeach
29     </tbody>
30 </table>
31 <div class="row">
32     <div class="mx-auto mt-3">
33         {{ $dosens->fragment('judul')->links() }}
34     </div>
35 </div>
36
37 @endsection
```

#	NID	Nama Dosen	Jurusan Dosen
1	99154082	Betania Yolanda M.Sc	Teknik Informatika
2	99629495	Darmanto Agustina M.T	Teknik Informatika
3	99574701	Edward Prabowo M.Sc	Ilmu Komputer
4	99568430	Erik Pudjiastuti M.Sc	Ilmu Komputer
5	99611023	Halima Mansur M.Sc	Sistem Informasi

Gambar: Tampilan halaman localhost:8000/dosens

Di baris 1 terdapat perintah `@extends('layouts.app')`, yang artinya view ini juga diturunkan dari `layouts\app.blade.php`.

Di baris 5 saya menampilkan judul halaman memakai operator "??" Operator tanda tanya 2 kali ini disebut sebagai *null coalescing operator*. Perintah tersebut bisa dibaca: "Jika variabel `$nama_jurusan` berisi sesuatu, tampilkan isi variabel tersebut. Namun jika `$nama_jurusan` tidak berisi apapun (null), maka tampilkan teks 'Universitas ILKOOOM'".

Kode ini saya rancang agar judul halaman bisa diganti berdasarkan data yang dikirim dari controller. Di method `index()` sebelumnya, tidak ada pengiriman variabel `$nama_jurusan`, sehingga judul halaman menjadi "Data Dosen Universitas ILKOOOM".

Untuk menampilkan semua data dosen, saya menggunakan tag `<table>` HTML. Bagian judul kolom dibuat pada baris 9 – 16. Sedangkan isi tabel berada di perulangan `foreach` baris 18 – 28.

Di baris 18, perintah `@foreach ($dosens as $dosen)` akan "membuka" collection `$dosens` menjadi 1 object `$dosen` dalam setiap perulangan.

Kolom pertama dipakai untuk menampilkan nomor urut. Nomor urut ini dihitung dengan perintah `{{$dosens->firstItem() + $loop->iteration - 1}}`. Perhitungannya memang agak kompleks karena tabel terpecah ke dalam beberapa halaman (efek pagination).

Method `$dosens->firstItem()` akan mengembalikan angka urutan pertama dari sebuah halaman pagination. Sebagai contoh, karena pagination untuk tabel `dosens` saya set menjadi 5, maka untuk setiap halaman tampil 5 baris data. Di halaman pertama, method `$dosens->firstItem()` akan berisi angka 1. Untuk halaman kedua akan berisi angka 6, untuk halaman

ketiga akan berisi angka 11, dst.

Sedangkan `$loop->iteration` dipakai untuk mencari nomor urut perulangan. Ingat, kita menampilkan data `dosens` dalam sebuah perulangan `foreach`, maka `$loop->iteration` akan berisi angka menaik dari setiap perulangan, yakni angka 1 untuk perulangan pertama, angka 2 untuk perulangan kedua, dst sampai perulangan `foreach` selesai di lakukan.

Hasil penambahan `$dosens->firstItem() + $loop->iteration` harus di kurangi lagi dengan angka 1 karena `$dosens->firstItem()` dan `$loop->iteration` sama-sama berangkat dari angka 1. Jika tidak dikurangi, baris pertama akan mulai dari angka 2.

Perhitungan ini memang agak panjang karena Laravel tidak menyediakan angka urut untuk data pagination.

Alternatif lain, nomor urut bisa diganti dengan kolom `id` yang ada di tabel. Akan tetapi karena data dosen saya urutkan berdasarkan nama, maka `id` ini menjadi acak.

Setelah nomor urut, kolom kedua dipakai untuk menampilkan nomor **NID** dosen dengan perintah `{{$dosen->nid}}` di baris 21.

Kolom ketiga dipakai untuk menampilkan nama dosen dengan perintah `{{$dosen->nama}}`. Untuk kolom ini saya ingin agar nama dosen menjadi sebuah link, yang ketika di klik akan menuju halaman baru berisi data rinci tentang dosen tersebut.

Dalam konsep RESTfull, data rincian dari satu dosen akan diproses oleh *named route* `dosens.show` (halamannya akan kita buat sesaat lagi). Nomor id dosen juga ikut dikirim sebagai argument.

Lanjut ke kolom keempat, perintah `{{$dosen->jurusan->nama}}` dipakai untuk menampilkan jurusan tempat dosen terdaftar. Ini merupakan perintah eloquent relationship, karena nama jurusan sebenarnya tidak ada di tabel `dosens` (yang ada hanya kolom `jurusan_id`), jadi nama jurusan harus diambil dari tabel `jurusans`.

Setelah menampilkan tabel, di baris 33 terdapat perintah `{{$dosens->fragment('judul')->links() }}` untuk menampilkan tombol pagination. Di sini saya memakai tambahan `fragment('judul')` agar ketika link pagination di klik, halaman tertahan di bagian judul, yakni atribut `id="judul"` di baris 4. Mengenai teknik ini pernah kita bahas di bab Pagination bagian Menambah Hash Fragment.

View `jurusans\dosen.blade.php` ini bisa diakses dari alamat `localhost:8000/dosens`.

Optimisasi dengan Eager Loading

View dosen sudah tampil sempurna, tapi saya ingin membahas sedikit tentang lazy loading vs eager loading. Yup, halaman view dosen di atas sebenarnya masih bisa dioptimalkan lagi karena terdapat 7 kali akses ke database.

Silahkan install **laravel debugbar** dengan perintah `composer require barryvdh/laravel-debugbar --dev`, lalu refresh halaman dan lihat tab Queries:

#	NID	Nama Dosen	Jurusan Dosen
1	99154082	Betania Yolanda M.Sc	Teknik Informatika

Messages Timeline Exceptions Views 3 Route **Queries 7** Models 19 Mails Gate Session Request

7 statements were executed, 4 of which were duplicated, 3 unique

```

select count(*) as aggregate from `dosens`
select * from `dosens` order by `nama` asc limit 5 offset 0
select * from `jurusans` where `jurusans`.`id` = 3 limit 1
select * from `jurusans` where `jurusans`.`id` = 3 limit 1
select * from `jurusans` where `jurusans`.`id` = 1 limit 1
select * from `jurusans` where `jurusans`.`id` = 1 limit 1
select * from `jurusans` where `jurusans`.`id` = 2 limit 1

```

GET dosens 18MB 3.24s 7.4.2 110ms 93.62ms \app\Http\Controllers\DosenController.php:17 laravel 3.54ms \app\Http\Controllers\DosenController.php:17 laravel 2.47ms view::dosen.index:26 laravel 2.75ms view::dosen.index:26 laravel 3.12ms view::dosen.index:26 laravel 1.79ms view::dosen.index:26 laravel 2.28ms view::dosen.index:26 laravel

Gambar: Efek lazy loading

Total terdapat 7 query yang diproses Laravel. Ini terjadi karena penggunaan perintah `$dosen->jurusan->nama` yang mengakses tabel jurusans. Karena perintah tersebut berada di dalam perulangan `foreach`, maka akan di ulang sebanyak 5 kali.

Ini bisa di optimalkan dengan cara menambah perintah `with('jurusans')` di method `index()`:

`app\Http\Controllers\DosenController.php`

```

1 public function index()
2 {
3     $dosens = Dosen::with('jurusan')->orderBy('nama')->paginate(5);
4     return view('dosen.index',[ 'dosens' => $dosens]);
5 }

```

Penambahan kode ada di baris 3, dimana kita menggunakan teknik *eager loading* untuk mengambil data tabel dosen beserta tabel jurusan. Sekarang halaman yang sama hanya butuh 3 buah query saja:

#	NID	Nama Dosen	Jurusan Dosen
1	99154082	Betania Yolanda M.Sc	Teknik Informatika

Messages Timeline Exceptions Views 3 Route **Queries 3** Models 8 Mails Gate Session Request

3 statements were executed

```

select count(*) as aggregate from `dosens`
select * from `dosens` order by `nama` asc limit 5 offset 0
select * from `jurusans` where `jurusans`.`id` in (1, 2, 3)

```

GET dosens 18MB 402ms 7.4.2 33.65ms 29.23ms \app\Http\Controllers\DosenController.php:20 laravel 2.16ms \app\Http\Controllers\DosenController.php:20 laravel 2.26ms \app\Http\Controllers\DosenController.php:20 laravel

Gambar: Hasil eager loading

Secara berkala, tab Queries milik laravel debugbar sebaiknya terus dipantau, karena menjadi salah satu aspek yang bisa dioptimalkan.

21.6. Menampilkan Semua Data Mahasiswa

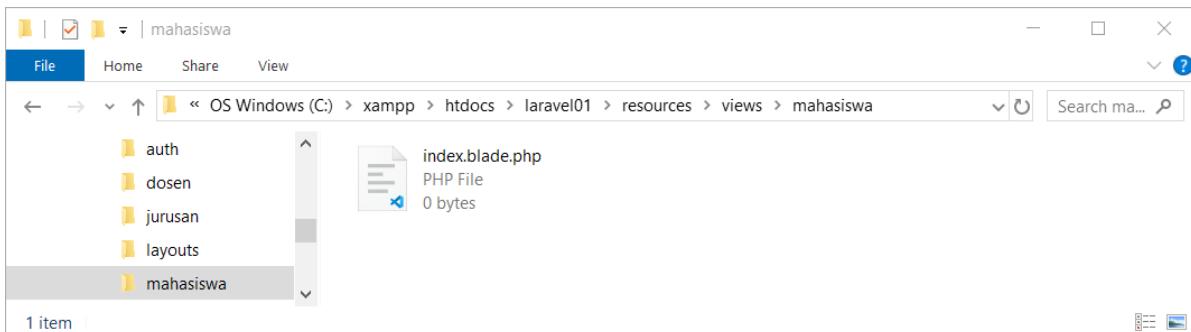
Sekarang kita lanjut membuat view untuk menampilkan semua data mahasiswa. Caranya juga mirip seperti view dosen. Pertama, ketik kode berikut ke method `index()` di file `MahasiswaController.php`:

`app\Http\Controllers\MahasiswaController.php`

```
1 public function index()
2 {
3     $mahasiswas = Mahasiswa::with('jurusan')->orderBy('nama')->paginate(10);
4     return view('mahasiswa.index',['mahasiswas' => $mahasiswas]);
5 }
```

Kode ini nyaris sama seperti method `index()` milik dosen, hanya saja sekarang saya akses dari class `Mahasiswa` serta menggunakan pagination 10. Selain itu tambahan method `with('jurusan')` akan mengambil data secara *eager loading*.

Kemudian buat folder **mahasiswa** di dalam `resources\views` beserta file `index.blade.php`:



Gambar: Buat file `index.blade.php` di dalam folder `resources\views\mahasiswa`

Untuk mempersingkat pembahasan, saya tidak lagi menguji apakah variabel `$mahasiswas` sudah bisa diakses dari dalam view atau belum. Namun langkah ini sebaiknya tidak dilewati. Dan berikut kode HTML, CSS dan blade untuk view `index.blade.php`:

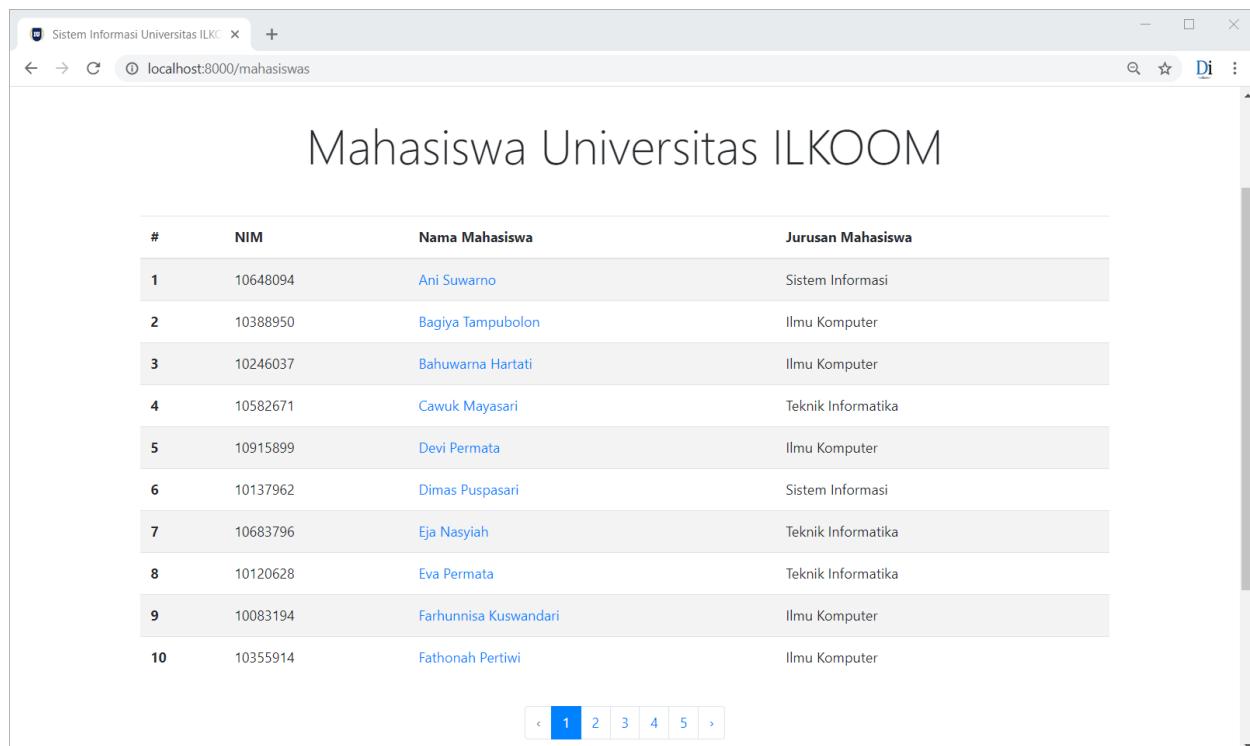
`resources\views\mahasiswa\index.blade.php`

```
1 @extends('layouts.app')
2 @section('content')
3
4 <h1 class="display-4 text-center my-5" id="judul">
5     Mahasiswa {{ $nama_jurusan ?? 'Universitas ILKOOOM' }}
6 </h1>
7
8 <table class="table table-striped">
9     <thead>
```

```

10   <tr>
11     <th>#</th>
12     <th>NIM</th>
13     <th>Nama Mahasiswa</th>
14     <th>Jurusan Mahasiswa</th>
15   </tr>
16 </thead>
17 <tbody>
18 @foreach ($mahasiswas as $mahasiswa)
19   <tr>
20     <th>{{$mahasiswas->firstItem() + $loop->iteration - 1}}</th>
21     <td>{{$mahasiswa->nim}}</td>
22     <td>
23       <a href="{{ route('mahasiswas.show',[ 'mahasiswa'=>$mahasiswa->id]) }}">
24         {{$mahasiswa->nama}}</a>
25     </td>
26     <td>{{$mahasiswa->jurusan->nama}}</td>
27   </tr>
28 @endforeach
29 </tbody>
30 </table>
31 <div class="row">
32   <div class="mx-auto mt-3">
33     {{ $mahasiswas->fragment('judul')->links() }}
34   </div>
35 </div>
36
37 @endsection

```



The screenshot shows a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/mahasiswa". The page displays a table titled "Mahasiswa Universitas ILKOOOM". The table has four columns: "#", "NIM", "Nama Mahasiswa", and "Jurusan Mahasiswa". The data is paginated with a navigation bar at the bottom.

#	NIM	Nama Mahasiswa	Jurusan Mahasiswa
1	10648094	Ani Suwarno	Sistem Informasi
2	10388950	Bagiya Tampubolon	Ilmu Komputer
3	10246037	Bahuwarna Hartati	Ilmu Komputer
4	10582671	Cawuk Mayasari	Teknik Informatika
5	10915899	Devi Permata	Ilmu Komputer
6	10137962	Dimas Puspasari	Sistem Informasi
7	10683796	Eja Nasyiah	Teknik Informatika
8	10120628	Eva Permata	Teknik Informatika
9	10083194	Farhunnisa Kuswandari	Ilmu Komputer
10	10355914	Fathonah Pertwi	Ilmu Komputer

Gambar: Tampilan halaman localhost:8000/mahasiswa

Kode program di atas nyaris sama seperti yang kita pakai untuk menampilkan data dosen.

Sedikit perbedaan ada di perulangan foreach yang sekarang mengakses variabel `$mahasiswa` as `$mahasiswa` di baris 18. Kemudian sepanjang perulangan ditampilkan nomor urut, NIM mahasiswa, nama mahasiswa serta jurusan tempat mahasiswa terdaftar.

Kolom nama mahasiswa juga saya buat menjadi link, yang ketika di klik akan menuju halaman data detail untuk mahasiswa tersebut. Halaman ini di proses oleh *named route* '`mahasiswa.show`'. Tidak lupa di bagian bawah terdapat kode untuk menampilkan pagination.

21.7. Menampilkan Semua Data Mata Kuliah

Data matakuliah juga dibuat dengan cara yang sama seperti data dosen dan mahasiswa.

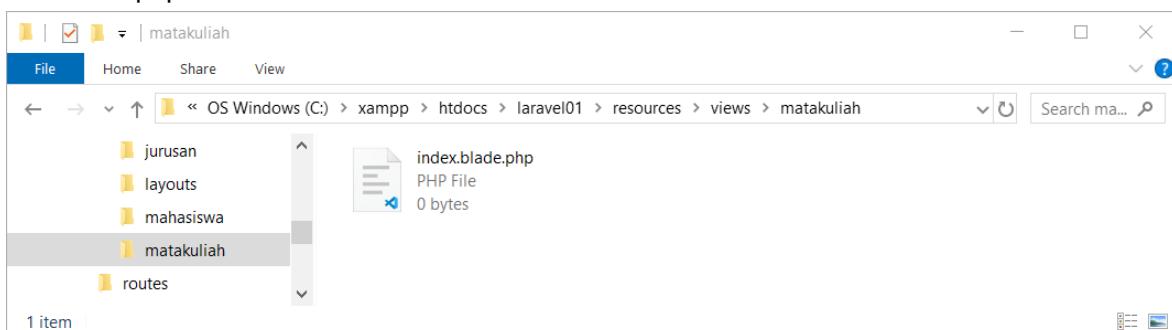
Silahkan buka file `MatakuliahController.php` lalu tambah kode berikut ke method `index()`:

`app\Http\Controllers\MatakuliahController.php`

```
1     public function index()
2     {
3         $matakuliahs = Matakuliah::with('dosen','jurusan')
4             ->orderBy('nama')->paginate(10);
5         return view('matakuliah.index',[ 'matakuliahs' => $matakuliahs]);
6     }
```

Sedikit perbedaan ada di penulisan argument method `with()`. Di sini saya menulis 2 buah relationship, yakni `with('dosen', 'jurusan')` untuk membuat *eager loading*. Ini diperlukan karena dalam view nanti terdapat kode untuk mengakses kedua tabel. Variabel `$matakuliahs` yang dikirim ke view dipecah menjadi 10 data per halaman, yakni hasil dari method `paginate(10)`.

Lanjut ke membuat folder **matakuliah** di dalam `resources\views`, serta satu file `index.blade.php` di dalam folder tersebut:



Gambar: Buat file `index.blade.php` di dalam folder `resources\views\matakuliah`

Berikut kode HTML, CSS dan blade untuk view `index.blade.php`:

`resources\views\matakuliah\index.blade.php`

```
1 @extends('layouts.app')
2 @section('content')
```

```

3
4 <h1 class="display-4 text-center my-5" id="judul">
5   Mata Kuliah Universitas ILKOOOM
6 </h1>
7
8 <table class="table table-striped">
9   <thead>
10    <tr>
11      <th>#</th>
12      <th>Kode</th>
13      <th>Nama Mata Kuliah</th>
14      <th>Dosen Pengajar</th>
15      <th>Jumlah SKS</th>
16      <th>Jurusan</th>
17    </tr>
18  </thead>
19  <tbody>
20    @foreach ($matakuliahs as $matakuliah)
21    <tr>
22      <th>{{$matakuliahs->firstItem() + $loop->iteration - 1}}</th>
23      <td>{{$matakuliah->kode}}</td>
24      <td>
25        <a href="{{ route('matakuliahs.show',
26          ['matakuliah' => $matakuliah->id]) }}">{{$matakuliah->nama}}</a>
27      </td>
28      <td>
29        <a href="{{route('dosens.show',[ 'dosen' => $matakuliah->dosen->id])}}">
30          {{$matakuliah->dosen->nama}}
31        </a>
32      </td>
33      <td>{{$matakuliah->jumlah_sks}}</td>
34      <td>{{$matakuliah->jurusan->nama}}</td>
35    </tr>
36  @endforeach
37  </tbody>
38 </table>
39 <div class="row">
40   <div class="mx-auto mt-3">
41     {{ $matakuliahs->fragment('judul')->links() }}
42   </div>
43 </div>
44
45 @endsection

```

Kode yang dipakai tetap sama seperti sebelumnya, namun kali ini memakai perulangan @foreach (\$matakuliahs as \$matakuliah) untuk proses looping data tabel.

Kolom yang ditampilkan adalah nomor urut, kode mata kuliah, nama mata kuliah, dosen yang mengajar mata kuliah tersebut, jumlah sks serta nama jurusan dari mata kuliah.

Kolom nama mata kuliah dan nama dosen saya buat menjadi link, yang ketika di klik akan menampilkan halaman detail dari mata kuliah dan dosen tersebut. Ini terdapat di baris 24 – 32.

Terakhir di baris 41 terdapat kode {{ \$matakuliahs->fragment('judul')->links() }} untuk

menampilkan tombol pagination.

Sampai di sini kita sudah memiliki 4 buah file view untuk menampilkan semua data jurusan, dosen, mahasiswa dan mata kuliah. Dalam konsep RESTfull, ini merupakan bagian dari **index**. Berikutnya kita akan lanjut ke **show**, yakni menampilkan data secara individu.

21.8. Menampilkan Daftar Dosen dan Mahasiswa Jurusan

Dalam konsep RESTfull, route **show** dipakai untuk menampilkan informasi detail dari sebuah data. Pada tabel jurusan, saya memutuskan tidak menggunakan route show ini, tapi memecahnya ke dalam 2 buah view. Masing-masing view akan menampilkan data dosen dan data mahasiswa secara terpisah.

Ketika membuat view index untuk tabel jurusan, saya merancang 2 buah tombol di setiap Card, yakni tombol '**Dosen**' dan tombol '**Mahasiswa**'. Idenya adalah, ketika tombol ini di klik, akan tampil daftar semua dosen dan semua mahasiswa untuk jurusan tersebut.

Berikut tombol yang dimaksud:



Gambar: Tombol "Dosen" dan "Mahasiswa" di setiap Card Jurusan

Tombol ini sebelumnya dibuat dengan kode berikut:

resources\views\matakuliah\index.blade.php

```
1 ...
2 <a href="{{ route('jurusan-dosen',[ 'jurusan_id' => $jurusan->id]) }}" 
3     class="btn btn-info btn-block">Dosen</a>
4 <a href="{{ route('jurusan-mahasiswa',[ 'jurusan_id' => $jurusan->id]) }}" 
5     class="btn btn-success btn-block">Mahasiswa</a>
6 ...
```

Terlihat bahwa link tombol menuju ke *named route* 'jurusan-dosen' dan *named route* 'jurusan-mahasiswa'. Keduanya diproses oleh route di bawah ini:

routes\web.php

```
1 ...
2 Route::get('/jurusan-dosen/{jurusan_id}', [JurusanController::class,
```

```
3     'jurusanDosen'])->name('jurusan-dosen');
4 Route::get('/jurusan-mahasiswa/{jurusan_id}', [JurusanController::class,
5         'jurusanMahasiswa'])->name('jurusan-mahasiswa');
```

Dengan membaca kedua route, maka kode yang memproses tombol tersebut nantinya ada di method jurusanDosen() milik JurusanController.php serta method jurusanMahasiswa() milik JurusanController@.php.

Memproses Tombol Dosen

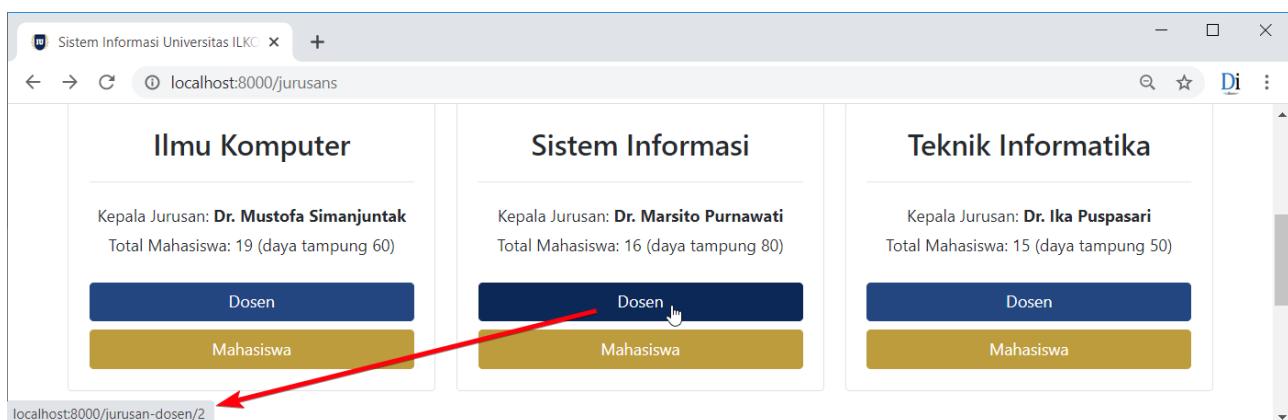
Ketika tombol 'Dosen' di klik, itu akan di proses oleh method jurusanDosen() di file JurusanController.php. Berikut kode yang diperlukan:

app\Http\Controllers\JurusanController.php

```
1 ...
2 use App\Models\Dosen;
3 ...
4
5 public function jurusanDosen($jurusan_id)
6 {
7     $dosens = Dosen::where('jurusan_id', $jurusan_id)->orderBy('nama')
8             ->paginate(5);
9     $nama_jurusan = Jurusan::find($jurusan_id)->nama;
10
11    return view('dosen.index', [
12        'dosens' => $dosens,
13        'nama_jurusan' => $nama_jurusan,
14    ]);
15 }
```

Agar lebih mudah diikuti, saya akan bahas dengan memakai contoh jurusan 'Sistem Informasi'.

Di dalam tabel jurusans, Sistem Informasi memiliki **id = 2**. Maka ketika view index jurusan di proses, tombol 'Dosen' akan menuju ke alamat `localhost:8000/jurusan-dosen/2`:



Gambar: URL yang dituju oleh tombol Dosen untuk jurusan Sistem Informasi

Pada saat tombol 'Dosen' di klik, route akan memanggil method jurusanDosen() yang kita tulis

di atas. Sesampainya di method ini, angka 2 ditampung oleh parameter \$jurusan_id di baris 5.

Pada baris 7, variabel \$dosens akan di berisi hasil perintah Dosen::where('jurusan_id', 2)->orderBy('nama')->paginate(5). Perintah ini mengambil semua data dosen dengan syarat kolom jurusan_id berisi angka 2. Hasil tersebut di urutkan berdasarkan kolom nama serta di pecah dengan pagination 5.

Perintah di baris 9 berfungsi untuk mencari nama jurusan yang saat ini sedang diakses. Harap diperhatikan bahwa method jurusanDosen() tidak memiliki informasi apa nama jurusan yang sedang diakses saat ini, karena informasi yang diterima hanya id jurusan saja.

Dalam contoh kita, perintah Jurusan::find(\$jurusan_id)->nama akan di proses sebagai Jurusan::find(2)->nama, sehingga variabel \$nama_jurusan berisi string 'Sistem Informasi'.

Terakhir di baris 11 - 14, variabel \$dosens dan \$nama_jurusan di kirim ke view dosen.index. Betul, file view ini sama persis dengan view yang di rancang untuk menampilkan semua data tabel dosens. Kita tidak perlu buat file view baru karena sudah pas untuk keperluan ini.

Untuk uji coba, silahkan buka alamat localhost:8000/jurusans, lalu klik salah satu tombol 'Dosen':

#	NID	Nama Dosen	Jurusan Dosen
1	99611023	Halima Mansur M.Sc	Sistem Informasi
2	99092104	Kuncara Purnawati M.Kom	Sistem Informasi
3	99812991	Putu Prasetya M.Sc	Sistem Informasi
4	99605319	Queen Suryatmi M.T	Sistem Informasi
5	99750967	Rahmi Prasasta M.T	Sistem Informasi

Gambar: Hasil tampilan halaman localhost:8000/jurusan-dosen/2

Hasilnya, akan tampil daftar dosen khusus untuk jurusan itu saja. Judul halaman juga akan berganti sesuai dengan nama jurusan. Ini berasal dari penggunaan operator ?? yang sudah ada di bagian atas view dosen.index:

resources\views\dosen\index.blade.php

```

1 ...
2 <h1 class="display-4 text-center my-5" id="judul">
3     Data Dosen {{ $nama_jurusan ?? 'Universitas ILKOOOM' }}
```

```
4 </h1>
5 ...
```

Karena sekarang variabel \$nama_jurusan berisi string 'Sistem Informasi', maka teks yang tampil adalah 'Data Dosen Sistem Informasi', bukan lagi 'Data Dosen Universitas ILKOOOM'.

Memproses Tombol Mahasiswa

Sekarang kita masuk ke kode untuk menampilkan daftar mahasiswa. Pada saat tombol 'Mahasiswa' di klik, itu akan di proses oleh method jurusanMahasiswa() di file JurusanController.php:

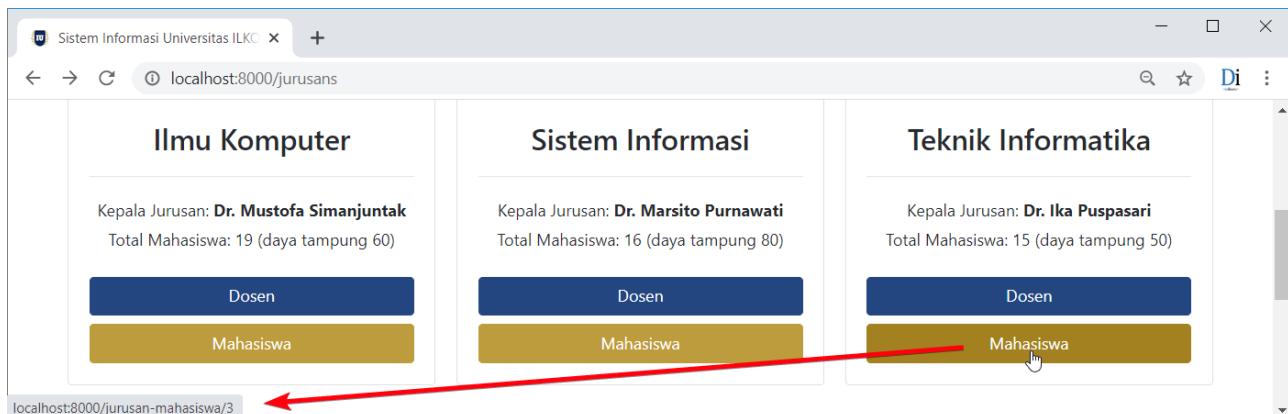
app\Http\Controllers\JurusanController.php

```
1 ...
2 use App\Models\Mahasiswa;
3 ...
4 public function jurusanMahasiswa($jurusan_id)
5 {
6     $mahasiswas = Mahasiswa::where('jurusan_id',$jurusan_id)->orderBy('nama')
7         ->paginate(5);
8     $nama_jurusan = Jurusan::find($jurusan_id)->nama;
9
10    return view('mahasiswa.index',[ 
11        'mahasiswas' => $mahasiswas,
12        'nama_jurusan' => $nama_jurusan,
13    ]);
14 }
```

Secara konsep, kode ini sama seperti sebelumnya, hanya saja sekarang diakses dari class Mahasiswa. Variabel \$mahasiswas akan berisi collection dari object Mahasiswa, serta variabel \$nama_jurusan akan berisi string dari nama jurusan yang sekarang sedang diakses.

Kedua variabel ini kemudian dikirim ke view mahasiswa.index, yang sebelumnya juga di pakai untuk menampilkan semua data mahasiswa.

Silahkan klik tombol 'Mahasiswa' untuk jurusan Teknik Informatika:



Gambar: URL yang dituju oleh tombol Mahasiswa untuk jurusan Teknik Informatika

Dalam contoh ini, jurusan Teknik Informatika memiliki id = 3, maka ketika tombol **Mahasiswa** di klik, akan menuju ke alamat `localhost:8000/jurusan-mahasiswa/3`. Angka '3' dipakai oleh method `jurusanMahasiswa()` untuk mencari semua mahasiswa yang memilih jurusan tersebut:



#	NIM	Nama Mahasiswa	Jurusan Mahasiswa
1	10582671	Cawuk Mayasari	Teknik Informatika
2	10683796	Eja Nasyiah	Teknik Informatika
3	10120628	Eva Permata	Teknik Informatika
4	10110155	Garang Sudiati	Teknik Informatika

Gambar: Hasil tampilan halaman `localhost:8000/jurusan-mahasiswa/3`

Teknik yang sama sebenarnya juga bisa dipakai untuk menampilkan daftar mata kuliah untuk satu jurusan tertentu. Namun saya memutuskan untuk tidak membuatnya. Jika tertarik, anda bisa coba rancang sendiri.

21.9. Menampilkan Data Satu Dosen

Sekarang kita masuk ke view untuk menampilkan data satu dosen. Dalam konsep RESTfull, bagian ini di proses oleh method `show()`. Silahkan buka file `DosenController.php` lalu tambah kode berikut ke dalam method `show()`:

app\Http\Controllers\DosenController.php

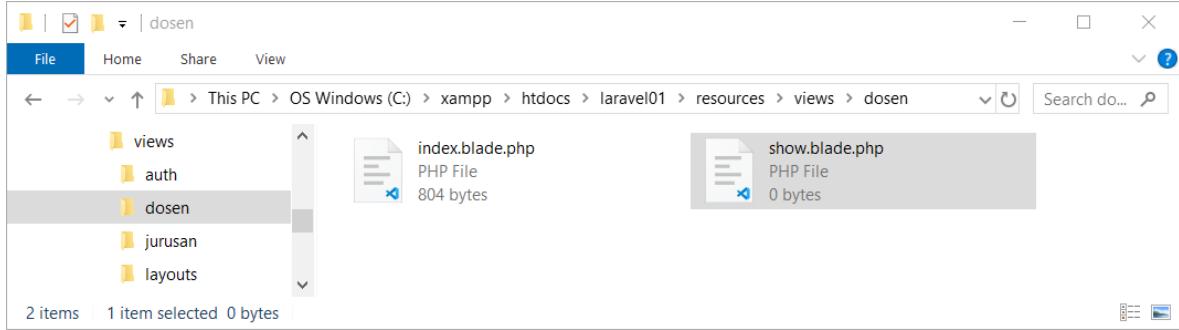
```

1 public function show(Dosen $dosen)
2 {
3     return view('dosen.show', compact('dosen'));
4 }
```

Cukup satu baris saja. Di sini kita memanfaatkan fitur *route model binding*, dimana parameter `$dosen` di baris 1 akan langsung terisi dengan object `Dosen` yang sedang di akses.

Sebagai contoh, jika URL yang di akses adalah `localhost:8000/dosens/4`, maka variabel `$dosen` sudah langsung berisi object dari `Dosen` yang memiliki id = 4. Variabel `$dosens` selanjutnya di kirim ke view `dosen.show` di baris 3.

Silahkan buka folder `resources\views\matakuliah`, lalu buat file `show.blade.php`:



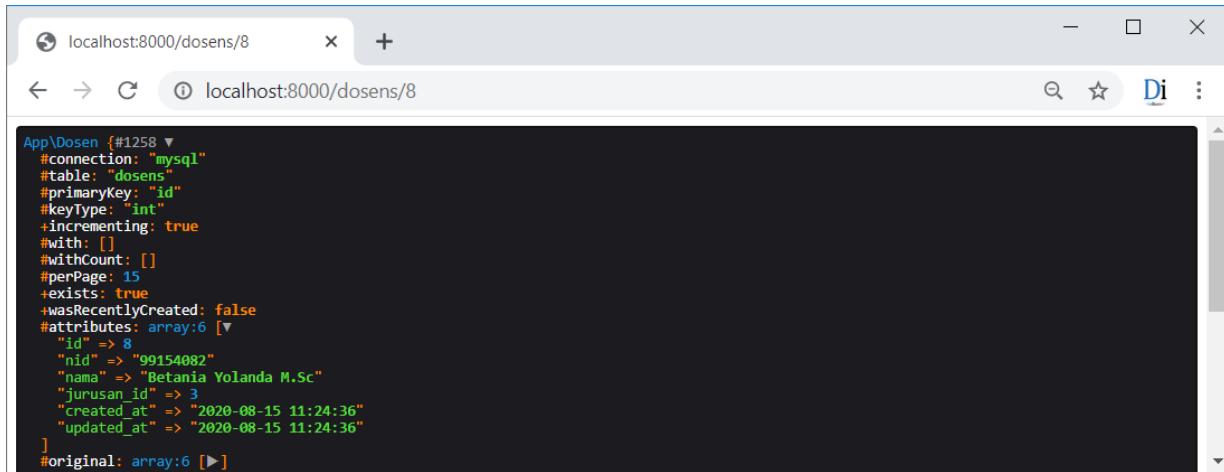
Gambar: Buat file show.blade.php di dalam folder resources\views\dosen

Seperti biasa, kita akan test terlebih dahulu apakah variabel \$dosen sudah bisa diakses atau tidak:

resources\views\dosen\show.blade.php

```
1 <?php
2     dump($dosen);
```

Untuk menguji kode ini, bisa dengan membuka URL `localhost:8000/dosens`, lalu klik salah satu nama dosen (yang sudah menjadi link). Atau bisa juga ketik alamat URL manual seperti `localhost:8000/dosens/8`:



Gambar: Hasil dump(\$dosen)

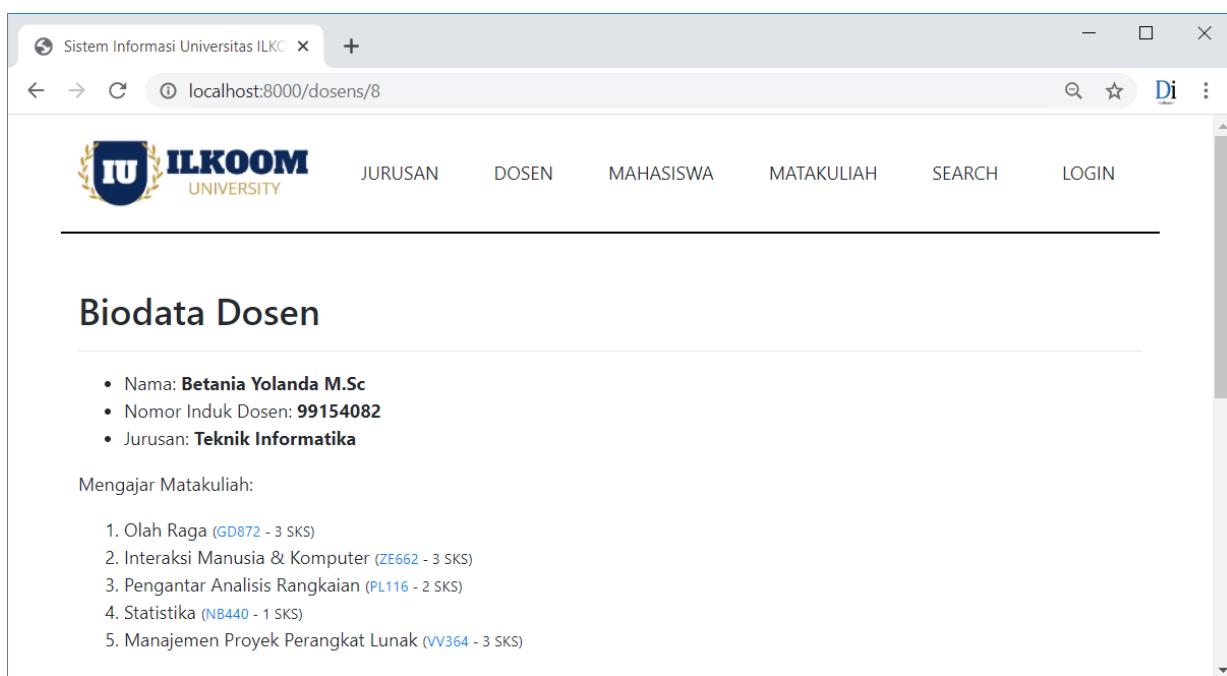
Pada saat alamat `localhost:8000/dosens/8` di akses, angka '8' akan dikirim sebagai argument \$jurusan ke method `show()` di controller. Dengan teknik route model binding, Laravel akan mencari data dosen yang memiliki id = 8 kemudian mengisinya ke variabel \$jurusan.

Sejauh ini tidak ada masalah, kita bisa masuk ke kode lengkap dari view `show.blade.php`:

resources\views\dosen\show.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
```

```
5   <h1 class="h2 mr-4">Biodata Dosen</h1>
6 </div>
7 <hr>
8 <ul>
9   <li>Nama: <strong>{{$dosen->nama}} </strong></li>
10  <li>Nomor Induk Dosen: <strong>{{$dosen->nid}} </strong></li>
11  <li>Jurusan: <strong>{{$dosen->jurusan->nama}} </strong></li>
12 </ul>
13 <p>Mengajar Matakuliah:</p>
14 <ol>
15  @foreach ($dosen->matakuliah as $matakuliah)
16  <li>
17    {{$matakuliah->nama}}
18    <small>
19      (<a href="{{ route('matakuliah.show', [
20        'matakuliah' => $matakuliah->id]) }}>{{$matakuliah->kode}}</a>
21        - {{$matakuliah->jumlah_sks}} SKS)
22      </small>
23    </li>
24  @endforeach
25 </ol>
26
27 @endsection
```



Gambar: Hasil tampilan halaman `http://localhost:8000/dosens/8`

Di baris 1 terdapat perintah `@extends('layouts.app')`, yang artinya view ini juga diturunkan dari `layouts.app`.

Di baris 5 saya membuat judul halaman sebagai "Biodata Dosen", yang kemudian diikuti dengan perintah untuk menampilkan nama dosen, NIP dosen, serta jurusan dosen antara baris 9 – 11. Data ini pada dasarnya sama seperti yang terdapat pada halaman `index.blade.php`, hanya saja

sekarang tampil dalam bentuk list menggunakan tag dan .

Agar lebih lengkap, saya juga menampilkan semua mata kuliah yang diajar oleh dosen. Untuk keperluan ini, kita butuh mengakses tabel `matakuliah`s yang ber-relasi dengan tabel `dosen`s. Data setiap mata kuliah bisa diakses dengan perulangan `@foreach ($dosen->matakuliah as $matakuliah)` di baris 15 – 24.

Sepanjang perulangan `foreach`, variabel `$matakuliah` berisi object dari setiap `MataKuliah`. Di sini saya menampilkan nama, kode dan jumlah sks dari setiap mata kuliah.

Untuk kode mata kuliah, saya tulis di dalam tag <a> yang menuju ke URL *named route* '`matakuliah.show`'. Sehingga jika kode mata kuliah di klik, akan menuju ke halaman detail dari mata kuliah tersebut (akan kita buat sesaat lagi).

21.10. Menampilkan Data Satu Mahasiswa

Kode untuk menampilkan data satu mahasiswa pada dasarnya sama seperti menampilkan data satu dosen. Kita mulai dengan mengisi method `show()` di file `Mahasiswa Controller.php`:

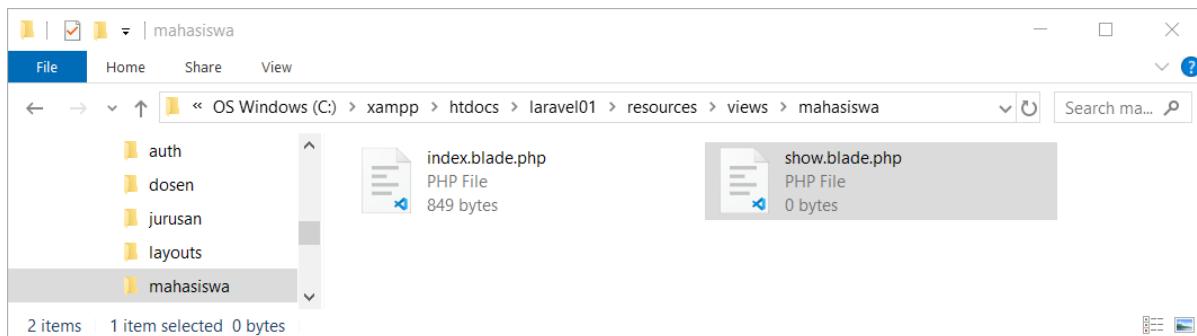
`app\Http\Controllers\HomeController.php`

```
1 public function show(Mahasiswa $mahasiswa)
2 {
3     $matakuliah = $mahasiswa->matakuliah->sortBy('nama');
4     return view('mahasiswa.show',[
5         'mahasiswa' => $mahasiswa,
6         'matakuliah' => $matakuliah,
7     ]);
8 }
```

Kode di atas masih menggunakan *route model binding*, hanya saja sekarang ditambah dengan perintah untuk mengambil data semua mata kuliah yang terhubung ke si mahasiswa (baris 3). Ini saya lakukan agar data mata kuliah bisa langsung terurut berdasarkan nama.

Variabel `$mahasiswa` yang berisi object dari 1 mahasiswa serta variabel `$matakuliah`s yang berisi daftar mata kuliah akan di kirim ke view '`mahasiswa.show`' di baris 4 – 7.

Selanjutnya silahkan buat file `show.blade.php` pada folder `resources\views\mahasiswa`:



Gambar: Buat file `show.blade.php` di dalam folder `resources\views\mahasiswa`

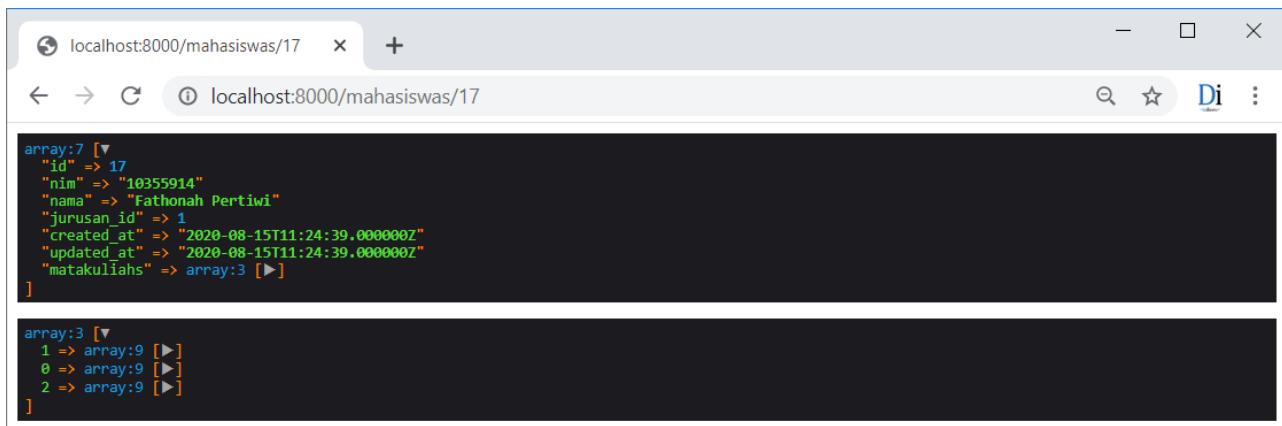
Lalu isi view tersebut dengan kode berikut:

```
resources\views\mahasiswa\show.blade.php
```

```
1 <?php
2     dump($mahasiswa->toArray());
3     dump($matakuliahs->toArray());
```

Karena ada 2 variabel yang dikirim, kita harus periksa apakah keduanya sudah sampai di view atau belum. Di sini saya memakai tambahan method toArray() agar hasil akhir berbentuk array.

Untuk uji coba, bisa dengan membuka URL `localhost:8000/mahasiswa`, lalu klik salah satu nama mahasiswa (yang sudah menjadi link). Atau bisa juga ketik alamat manual seperti `localhost:8000/mahasiswa/17`:



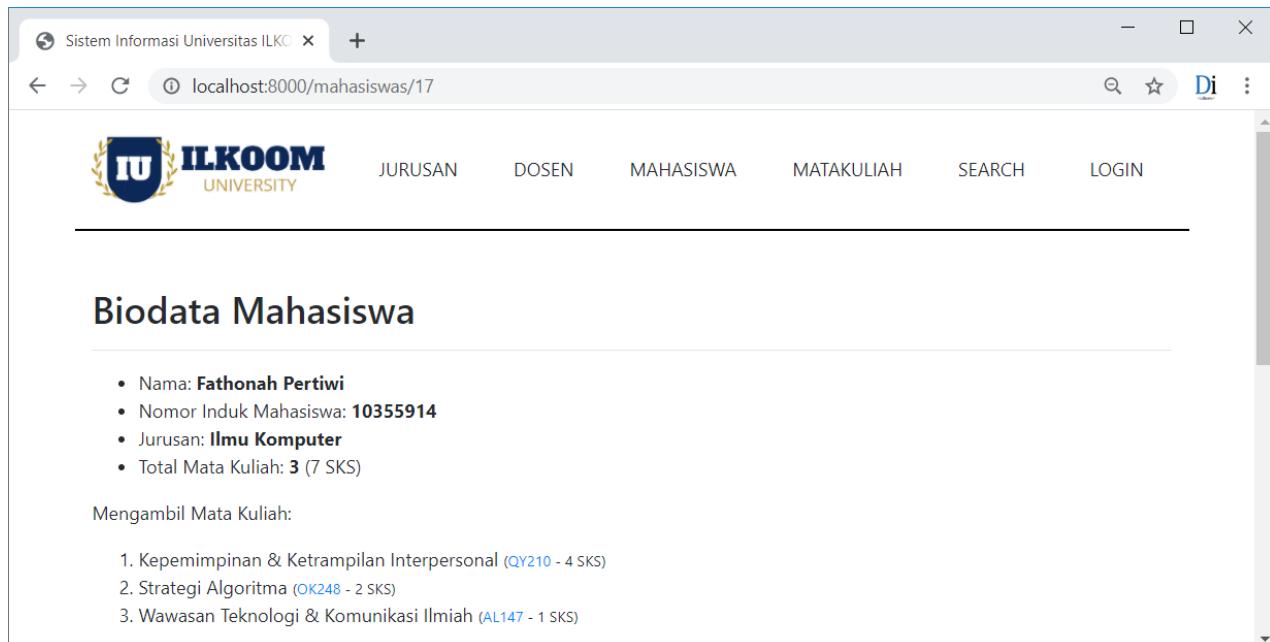
Gambar: Hasil `dump($mahasiswa->toArray())` dan `dump($matakuliahs->toArray())`

Sip, kedua variabel sudah bisa diakses, kita masuk ke kode view yang sebenarnya:

```
resources\views\mahasiswa\show.blade.php
```

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5     <h1 class="h2 mr-4">Biodata Mahasiswa</h1>
6 </div>
7 <hr>
8 <ul>
9     <li>Nama: <strong>{{ $mahasiswa->nama }} </strong></li>
10    <li>Nomor Induk Mahasiswa: <strong>{{ $mahasiswa->nim }} </strong></li>
11    <li>Jurusan: <strong>{{ $mahasiswa->jurusan->nama }} </strong></li>
12    <li>Total Mata Kuliah:
13        <strong> {{ count($matakuliahs) }} </strong>
14        ({{ $matakuliahs->sum('jumlah_sks') }} SKS)
15    </li>
16 </ul>
17 <p>Mengambil Mata Kuliah:</p>
18 <ol>
19     @foreach ($matakuliahs as $matakuliah)
```

```
20 <li>
21     {{$matakuliah->nama}}
22     <small>
23         (<a href="{{ route('matakuliah.show', [
24             ['matakuliah' => $matakuliah->id] ]) }}>{{$matakuliah->kode}}</a>
25             - {{$matakuliah->jumlah_sks}} SKS)
26     </small>
27 </li>
28 @endforeach
29 </ol>
30
31 @endsection
```



Gambar: Hasil tampilan halaman <http://localhost:8000/mahasiswa/17>

Kode ini juga mirip seperti menampilkan data satu dosen. Halaman saya beri judul 'Biodata Mahasiswa' di baris 5. Kemudian diikuti dengan list informasi mahasiswa seperti nama, nim, jurusan serta total mata kuliah antara baris 9 -15.

Khusus untuk menampilkan total mata kuliah, didapat dari perintah `count($matakuliah)`, yakni total jumlah element collection yang ada di variabel `$matakuliah`. Selain itu dalam tanda kurung saya juga menampilkan total jumlah sks dari semua mata kuliah yang diambil mahasiswa. Ini didapat dari perintah `$matakuliah->sum('jumlah_sks')` pada baris 14.

Sama seperti data dosen, saya juga menampilkan data seluruh mata kuliah yang diambil oleh mahasiswa. Data ini sudah tersedia dalam variabel `$matakuliah` yang diproses dengan perulangan foreach antara baris 19 – 28. Sepanjang perulangan ditampilkan data nama, kode, serta jumlah sks dari setiap mata kuliah.

Kode mata kuliah juga berfungsi sebagai link yang jika diklik akan menampilkan data detail dari satu mata kuliah tersebut.

21.11. Menampilkan Data Satu Matakuliah

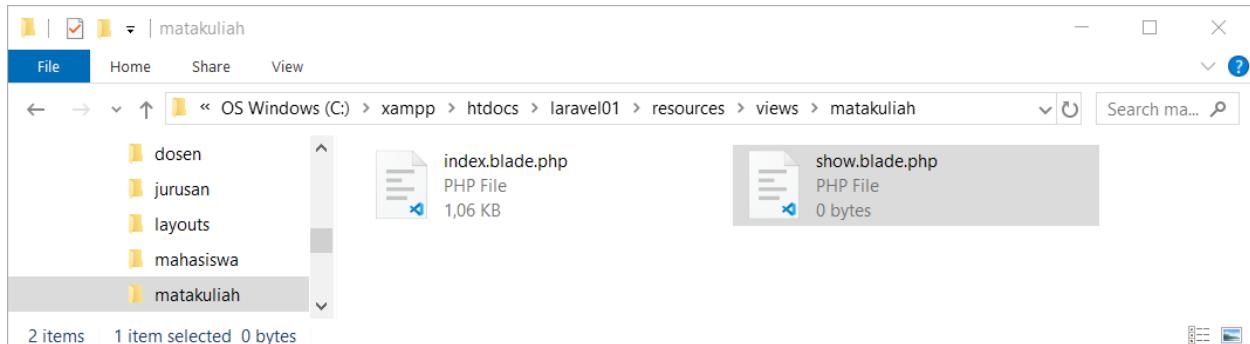
View **show** terakhir yang akan kita buat adalah untuk menampilkan data satu mata kuliah. Sama seperti sebelumnya, ini akan diproses oleh method **show()** di file **MatakuliahController.php**:

app\Http\Controllers\MatakuliahController.php

```
1 public function show(Matakuliah $matakuliah)
2 {
3     $mahasiswa = $matakuliah->mahasiswa->sortBy('nama');
4     return view('matakuliah.show',[
5         'matakuliah' => $matakuliah,
6         'mahasiswa' => $mahasiswa,
7     ]);
8 }
9 }
```

Kode ini nyaris sama seperti yang dipakai untuk method **show()** di **MahasiswaController**. Selain variabel **\$matakuliah** yang sudah langsung berisi satu object Matakuliah, saya juga mengambil data semua mahasiswa yang terhubung ke mata kuliah saat ini (baris 3). Data mahasiswa disimpan ke variabel **\$mahasiswa** untuk selanjutnya dikirim ke view '**mahasiswa.show**' di baris 4 – 7.

Kemudian silahkan buat file **show.blade.php** pada folder **resources\views\matakuliah**:



Gambar: Buat file **show.blade.php** di dalam folder **resources\views\matakuliah**

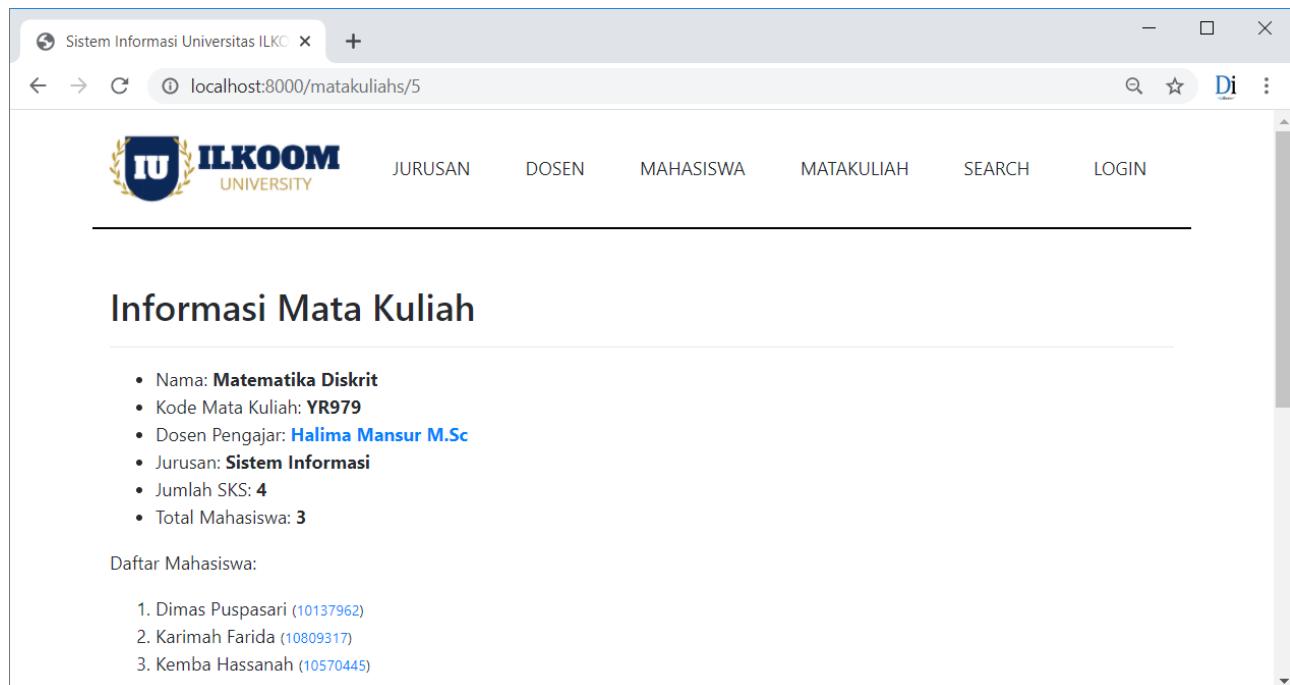
Untuk menyingkat pembahasan, langsung saja kita masuk ke kode view lengkap:

resources\views\matakuliah\show.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5     <h1 class="h2 mr-4">Informasi Mata Kuliah</h1>
6 </div>
7 <hr>
8 <ul>
9     <li>Nama: <strong>{{$matakuliah->nama}}</strong></li>
```

```
10 <li>Kode Mata Kuliah: <strong>{{$matakuliah->kode}} </strong></li>
11 <li>Dosen Pengajar:
12   <strong>
13     <a href="{{ route('dosens.show',[ 'dosen' => $matakuliah->dosen->id]) }}">
14       {{$matakuliah->dosen->nama}}
15     </a>
16   </strong></li>
17 <li>Jurusan: <strong>{{$matakuliah->jurusan->nama}} </strong></li>
18 <li>Jumlah SKS: <strong>{{ $matakuliah->jumlah_sks }} </strong></li>
19 <li>Total Mahasiswa: <strong>{{ count($mahasiswas) }} </strong></li>
20 </ul>
21 <p>Daftar Mahasiswa:</p>
22 <ol>
23   @foreach ($mahasiswas as $mahasiswa)
24     <li>
25       {{$mahasiswa->nama}}
26       <small>
27         (<a href="{{ route('mahasiswas.show',
28           ['mahasiswa'=>$mahasiswa->id]) }}>{{$mahasiswa->nim}}</a>)
29       </small>
30     </li>
31   @endforeach
32 </ol>
33
34 @endsection
```

Untuk mengakses view, bisa dengan membuka URL `localhost:8000/matakuliah`s, lalu klik salah satu nama mata kuliah (yang sudah menjadi link). Atau bisa juga ketik alamat URL manual seperti `localhost:8000/matakuliah/5`:



Gambar: Hasil tampilan halaman `localhost:8000/matakuliah/5`

Kode program yang ada juga mirip seperti file show lain, dimana data detail mata kuliah disajikan antara baris 9 - 19. Semua data ini bisa diakses dari variabel `$matakuliah`.

Di baris 13 - 15, saya menampilkan nama dosen yang mengajar mata kuliah dengan perintah relationship `$matakuliah->dosen->nama`. Nama dosen di set sebagai link yang jika di klik akan tersambung ke detail dosen, yakni route '`dosens.show`'.

Di bagian bawah terdapat rincian semua data mahasiswa yang mengambil mata kuliah ini. Data didapat dari variabel `$mahasiswas` yang di proses dengan perulangan foreach antara baris 23 - 31. Kali ini nim mahasiswa yang saya jadikan sebagai link yang ketika di klik akan menuju route '`mahasiswas.show`'.

Dengan selesainya pembuatan view show untuk tabel `matakuliah`s, maka bagian '**Read**' dari sistem CRUD Sistem Informasi Universitas ILKOOM sudah selesai.

Silahkan anda coba semua link yang ada, misalnya akses halaman `localhost:8000/dosens` untuk melihat semua daftar dosen, lalu klik nama salah satu nama dosen untuk melihat biodata dosen. Pada halaman biodata ini, klik lagi salah satu mata kuliah yang diajarkan dosen tersebut, dst.

Inilah contoh penerapan dari konsep *relationship*, dimana setiap tabel yang terpisah disatukan untuk membuat aplikasi utuh.

Sedikit tambahan, silahkan ganti route ' / ' agar bisa langsung me-load halaman data jurusan:

`routes\web.php`

```
1 ...  
2 Route::get('/', [JurusanController::class, 'index']);
```

Dengan perubahan ini maka ketika URL `http://localhost:8000` di akses, langsung terbuka halaman yang menampilkan semua data jurusan.

Setelah bagian **Read**, kita akan masuk ke bagian **Create**, yakni merancang proses input ke aplikasi Sistem Informasi Universitas ILKOOM. Namun sebelum itu saya ingin rehat sejenak dengan membahas satu library pendukung: **SweetAlert**.

22. SweetAlert

Dalam bab ini saya ingin membahas library JavaScript **SweetAlert**. Sesuai namanya, SweetAlert dipakai untuk membuat jendela alert atau jendela popup.

Sebenarnya cukup banyak library serupa. Bootstrap juga sudah menyediakan fitur bawaan melalui komponen Alert dan Modal. Namun saya lihat cukup banyak project yang menggunakan SweetAlert dan ingin menerapkannya pada project kita.

Praktek bab kali ini **tidak berhubungan** dengan bab sebelumnya. Oleh karena itu silahkan install Laravel 8 ke folder berbeda, misalnya ke laravel02 dengan perintah berikut:

```
composer create-project --prefer-dist laravel/laravel="^8.0" laravel02
```

22.1. Pengertian SweetAlert

SweetAlert adalah sebuah library JavaScript yang dipakai untuk menampilkan *jendela alert*. Jendela alert merupakan jendela popup kecil yang berisi info pemberitahuan seperti data berhasil diinput ke database, atau konfirmasi apakah yakin menghapus suatu data atau tidak.

Yang agak membingungkan, saat ini terdapat 2 library yang sama-sama menggunakan nama 'SweetAlert'. Yakni **SweetAlert** versi awal yang beralamat di sweetalert.js.org, dan **SweetAlert2** yang beralamat di sweetalert2.github.io. Keduanya dikembangkan oleh tim yang berbeda.

SweetAlert2 merupakan *fork* atau cabang yang dibuat oleh Limon Monte (github.com/limonte) dengan alasan SweetAlert pertama tidak lagi di maintenance oleh tim awal. Namun belakangan ini tim SweetAlert kembali aktif dan sudah melakukan beberapa update.

Dalam kesempatan kali ini, kita akan bahas library SweetAlert2 yang dikelola Limon Monte.

Agar materinya tidak terlalu panjang, saya tidak sempat membahas semua fitur bawaan SweetAlert2. Jika tertarik, anda bisa kunjungi dokumentasinya di sweetalert2.github.io.

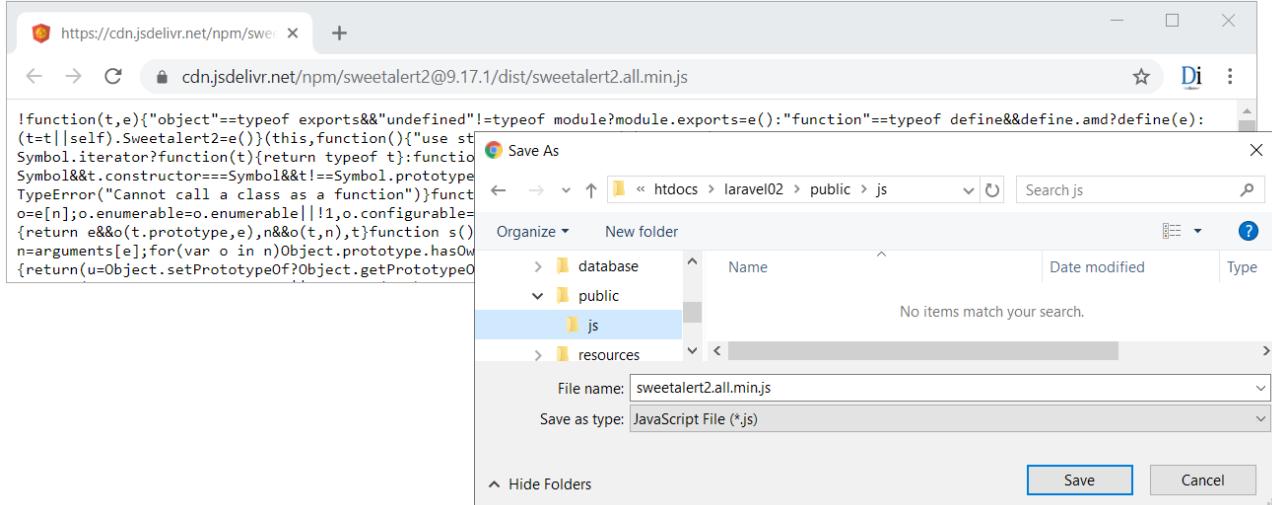
22.2. Download File SweetAlert2

File **SweetAlert2** tidak terlalu besar, hanya terdiri dari 1 file berukuran 66kb (versi minified). Terdapat beberapa cara untuk mendownload file ini, mulai dari CDN, install menggunakan npm,

atau download manual file source code.

Kali ini saya memilih untuk mendownload manual file SweetAlert2. Caranya, buka alamat <https://cdn.jsdelivr.net/npm/sweetalert2@9.17.1/dist/sweetalert2.all.min.js>, lalu save dengan menekan tombol CRTL + S.

Simpan halaman tersebut sebagai `sweetalert2.all.min.js` di folder `public\js`. Bawaan Laravel, folder `js` ini belum tersedia dan bisa dibuat terlebih dahulu.



Gambar: Save file sweetalert2.all.min.js di folder public\js

Jika mengalami kendala dengan cara di atas, file `sweetalert2.all.min.js` juga tersedia pada file `belajar_laravel_in_depth_1.zip` yang ada di Google Drive.

22.3. Cara Penggunaan SweetAlert2

Kita akan bahas cara penggunaan SweetAlert2 dari dalam view Laravel. Oleh karena itu silahkan tambah 3 route berikut ke file `routes\web.php`:

```
routes\web.php
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::view('/swal', 'swal');
6 Route::view('/swal-icon', 'swal-icon');
7 Route::view('/swal-display', 'swal-display');
```

Ketiga route akan mengakses file `swal.blade.php`, `swal-icon.blade.php` dan `swal-display.blade.php`. Kata 'swal' sendiri merupakan singkatan dari `sweetalert`.

Kita mulai dari route '/swal' terlebih dahulu. Silahkan buat file `swal.blade.php` di folder

resources\views\ dan isi dengan kode berikut:

resources\views\swal.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>Belajar SweetAlert</title>
8  </head>
9  <body>
10     <h1>Belajar SweetAlert</h1>
11     <button id="tombol">Click Me</button>
12
13     <script src="/js/sweetalert2.all.min.js"></script>
14
15     <script>
16         let tombol = document.getElementById('tombol');
17         tombol.addEventListener('click', jalankanSweetAlert);
18
19         function jalankanSweetAlert(){
20             Swal.fire('Selamat!', 'Anda berhasil menjalankan Sweet Alert', 'success')
21         }
22     </script>
23 </body>
24 </html>
```

Perintah di baris 1 – 9 merupakan kode pembuka HTML biasa. Konten halaman dimulai pada baris 10 dengan tag `<h1>` yang berisi teks "Belajar SweetAlert". Kemudian di baris 11 terdapat tag `<button id="tombol">`. Inilah tombol yang akan kita pakai untuk uji coba tampilan SweetAlert.

Tag `<script>` di baris 13 berfungsi untuk proses import file library SweetAlert2. Pastikan file `sweetalert2.all.min.js` sudah tersedia di folder `public\js` Laravel.

Proses pemanggilan dan konfigurasi SweetAlert berada dalam tag `<script>` antara baris 15 – 22. Di baris 16 saya membuat variabel `tombol` yang akan berisi `node object` dari element HTML dengan `id='tombol'`. Element tersebut tidak lain tag `<tombol>` yang ada di baris 11.

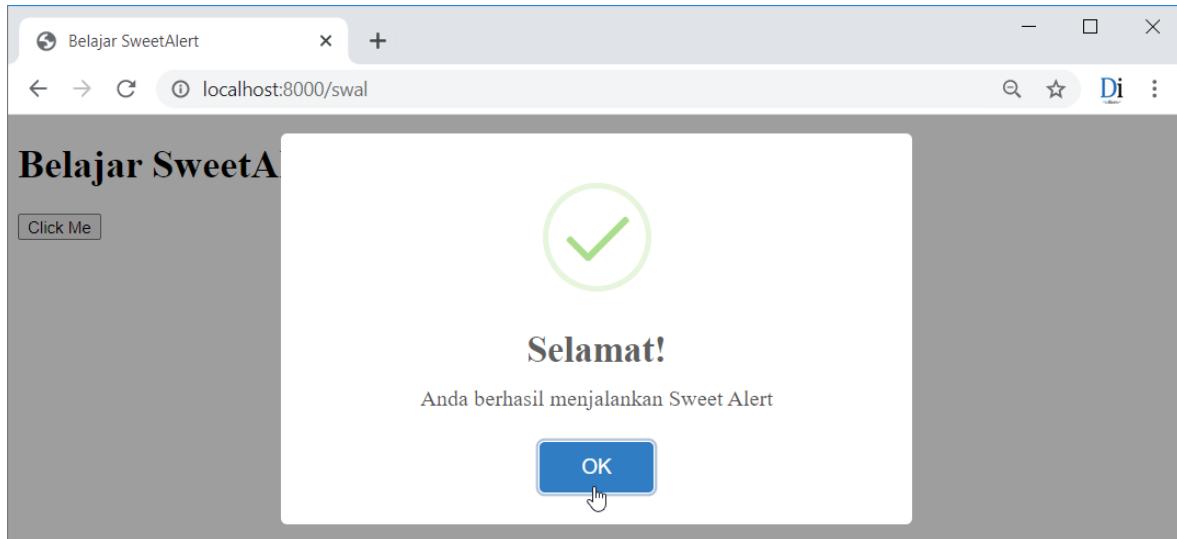
Untuk tag `<button>` ini, tambah sebuah event 'click' yang ketika di eksekusi akan menjalankan function `jalankanSweetAlert`. Dengan kata lain, setiap kali tombol di klik, function `jalankanSweetAlert()` akan di panggil.

Di dalam function `jalankanSweetAlert()` inilah saya menulis 1 baris perintah SweetAlert, yakni pemanggilan method `Swal.fire()` dengan 3 buah argument bertipe string.

Argument pertama dari method `Swal.fire()` dipakai untuk membuat judul alert, yang dalam contoh ini berisi teks 'Selamat!'. Kemudian argument kedua untuk pesan alert, yang berisi teks 'Anda berhasil menjalankan Sweet Alert'. Serta argument ketiga berisi pilihan icon alert,

dimana saya memilih 'success'.

Silahkan akses view di atas dari alamat `localhost:8000/swal` dan klik tombol 'Click Me':



Gambar: SweetAlert 'success' berhasil di jalankan

Inilah tampilan dari SweetAlert. Jika tombol OK di klik, tampilan popup akan hilang, atau bisa juga dengan men-klik latar belakang yang berwarna abu-abu. Jika tombol 'Click Me' di klik untuk kedua kali, jendela alert juga akan muncul lagi.

22.4. Pilihan Icon SweetAlert2

Dalam praktek sebelumnya, jendela SweetAlert2 tampil dengan icon tanda ceklist hijau. Total, SweetAlert2 menyediakan 6 pilihan icon, yakni:

- ◆ `none`: tanpa icon.
- ◆ `success`: icon ceklist berwarna hijau.
- ◆ `info`: icon huruf 'i' berwarna biru.
- ◆ `error`: icon tanda silang 'x' berwarna merah.
- ◆ `warning`: icon tanda seru '!' berwarna oranye.
- ◆ `question`: icon tanda tanya '?' berwarna kebiruan.

Pilihan icon ini diatur dari argument ketiga method `Swal.fire()`. Kita akan coba satu per satu pada route kedua, yakni '`swal-icon`'. Silahkan buat view `swal-icon.blade.php` dan isi dengan kode berikut:

`resources\views\swal-icon.blade.php`

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
```

SweetAlert

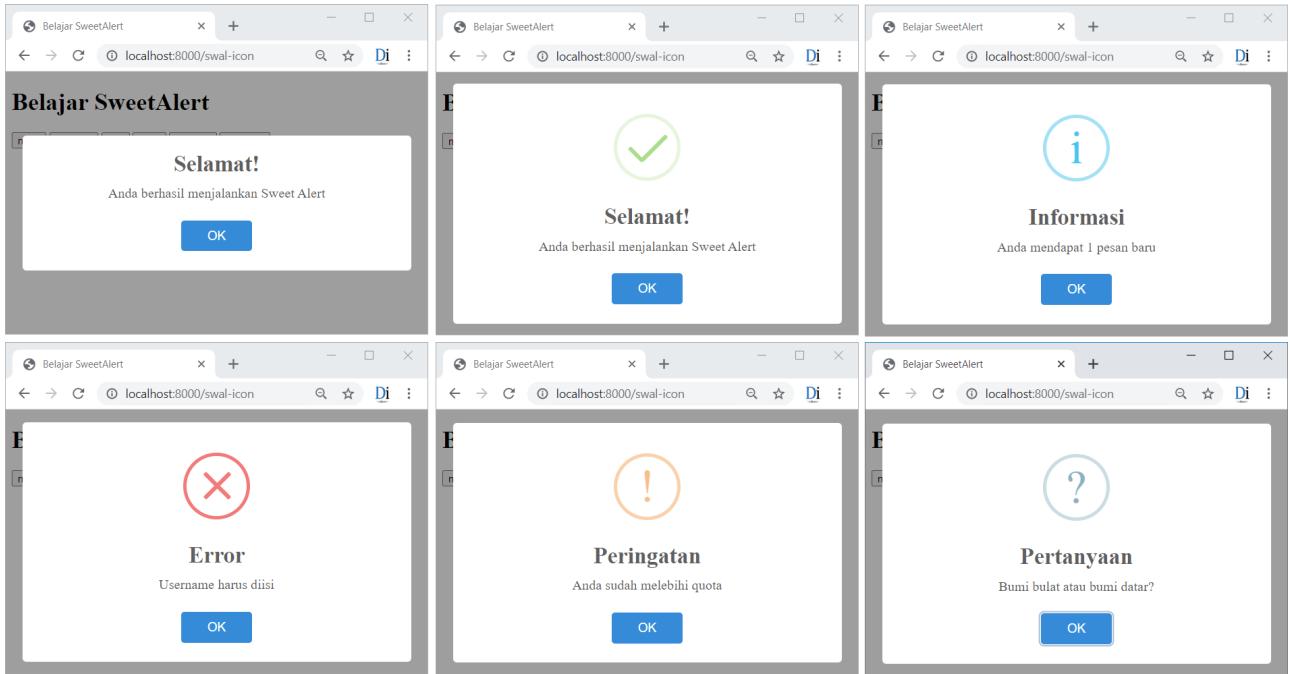
```
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Belajar SweetAlert</title>
8 </head>
9 <body>
10  <h1>Belajar SweetAlert</h1>
11
12  <button onclick="alertNone()">none</button>
13  <button onclick="alertSuccess()">success</button>
14  <button onclick="alertInfo()">info</button>
15  <button onclick="alertError()">error</button>
16  <button onclick="alertWarning()">warning</button>
17  <button onclick="alertQuestion()">question</button>
18
19  <script src="/js/sweetalert2.all.min.js"></script>
20
21  <script>
22  function alertNone(){
23      Swal.fire('Selamat!', 'Anda berhasil menjalankan Sweet Alert',)
24  }
25
26  function alertSuccess(){
27      Swal.fire('Selamat!', 'Anda berhasil menjalankan Sweet Alert', 'success')
28  }
29
30  function alertInfo(){
31      Swal.fire('Informasi', 'Anda mendapat 1 pesan baru', 'info')
32  }
33
34  function alertError(){
35      Swal.fire('Error', 'Username harus diisi', 'error')
36  }
37
38  function alertWarning(){
39      Swal.fire('Peringatan', 'Anda sudah melebihi quota', 'warning')
40  }
41
42  function alertQuestion(){
43      Swal.fire('Pertanyaan', 'Bumi bulat atau bumi datar?', 'question')
44  }
45  </script>
46 </body>
47 </html>
```

Untuk mempersingkat penulisan, kali ini saya langsung menulis event JavaScript menggunakan atribut onclick pada semua tag <button> antara baris 12 – 17. Ketika tombol-tombol ini diklik, function yang sesuai akan dijalankan.

Pada setiap function, terdapat penulisan method `Swal.fire()` dengan argument yang berbeda-beda. Fokus utama kita ada di argument ketiga, karena itulah yang menentukan jenis icon dari setiap jendela alert.

View ini bisa diakses dari alamat `localhost:8000/swal-icon`:

SweetAlert



Gambar: Berbagai tampilan icon SweetAlert2

22.5. Fitur SweetAlert2

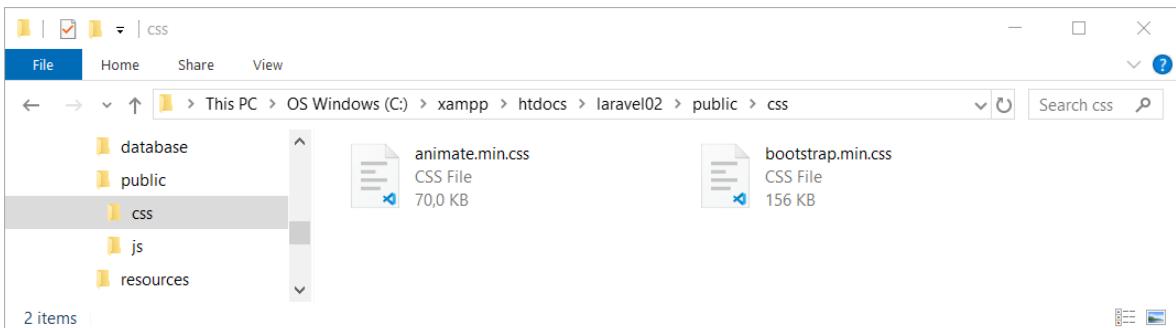
Selain jenis icon, SweetAlert2 juga menyediakan banyak pengaturan lain seperti menambah teks footer, menginput tag HTML, menambah gambar, membuat timer, membuat jendela konfirmasi, hingga efek animasi.

Agar tampilan jendela alert lebih menarik, untuk praktik ini saya perlu file CSS Bootstrap dan juga Animate.css, yakni sebuah library CSS untuk membuat efek animasi.

File CSS Bootstrap bisa diambil dari project sebelumnya, atau download ke <https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css>.

Sedangkan untuk file Animate.css bisa download ke alamat <https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.0.0/animate.min.css>.

Simpan kedua file sebagai bootstrap.min.css dan animate.min.css di folder public\css:



Gambar: Isi folder public\css

File `bootstrap.min.css` dan `animate.min.css` juga tersedia di file `belajar_laravel_in_depth_1.zip` yang ada di Google Drive.

Setelah itu silahkan buat file `view swal-display.blade.php` dan isi dengan kode berikut:

`resources\views\swal-display.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>Belajar SweetAlert</title>
8      <link rel="stylesheet" href="/css/bootstrap.min.css">
9      <link rel="stylesheet" href="/css/animate.min.css">
10 </head>
11 <body>
12     <div class="container mt-5">
13         <h1>Belajar SweetAlert</h1>
14         <hr>
15         <button class="btn btn-primary" onclick="alertFooter()">footer</button>
16         <button class="btn btn-primary" onclick="alertHtml()">html</button>
17         <button class="btn btn-primary" onclick="alertImage()">image</button>
18         <button class="btn btn-primary" onclick="alertPosition()">position</button>
19         <button class="btn btn-primary" onclick="alertTimer()">timer</button>
20         <button class="btn btn-primary" onclick="alertConfirm()">confirm</button>
21         <button class="btn btn-primary" onclick="alertToast()">toast</button>
22         <button class="btn btn-primary" onclick="alertToast2()">toasttimer</button>
23         <button class="btn btn-primary" onclick="alertAnimate()">animate</button>
24     </div>
25
26     <script src="/js/sweetalert2.all.min.js"></script>
27     <script>
28         function alertFooter(){
29             Swal.fire({
30                 icon: 'success',
31                 title: 'Selamat!',
32                 text: 'Anda berhasil menjalankan Sweet Alert',
33                 footer: '<span>Dipersembahkan oleh '+
34                     '<a href="https://sweetalert2.github.io">'+
35                     'sweetalert2</a> </span>',
36             })
37         }
38
39         function alertHtml(){
40             Swal.fire({
41                 icon: 'info',
42                 title: '<strong>Pesan sponsor</strong>',
43                 html: '<i>Belajar programming?</i> di '+
44                     '<a href="https://www.duniailkom.com"> DuniaIlkom</a> aja!',
45                 footer: '<span>Di sponsor oleh <a href="https://www.duniailkom.com"> '+
46                     'DuniaIlkom</a></span>',

```

SweetAlert

```
47     })
48 }
49
50 function alertImage(){
51     Swal.fire({
52         title: 'Selamat!',
53         text: 'Anda berhasil menjalankan Sweet Alert',
54         footer: 'Dipersembahkan oleh <a href="https://www.duniailkom.com"> '+
55             'DuniaIlkom</a>',
56         imageUrl: 'https://picsum.photos/450/200',
57         imageWidth: 450,
58         imageHeight: 200,
59         imageAlt: 'Sebuah gambar random'
60     })
61 }
62
63 function alertPosition(){
64     Swal.fire({
65         title: 'Selamat!',
66         text: 'Anda berhasil menjalankan Sweet Alert',
67         position: 'top-end',
68     })
69 }
70
71 function alertTimer(){
72     Swal.fire({
73         position: 'top-end',
74         icon: 'success',
75         title: 'Email sudah berhasil dikirim',
76         showConfirmButton: false,
77         timer: 1500
78     })
79 }
80
81 function alertConfirm(){
82     Swal.fire({
83         title: 'Konfirmasi',
84         text: "Apakah anda yakin ingin menghapus data ini?",
85         icon: 'warning',
86         showCancelButton: true,
87         confirmButtonColor: '#3085d6',
88         cancelButtonColor: '#d33',
89         confirmButtonText: 'Ya, hapus!'
90     }).then((result) => {
91         if (result.value) {
92             Swal.fire(
93                 'Di hapus',
94                 'Data berhasil di hapus',
95                 'success'
96             )
97         }
98     })
99 }
100
101 function alertToast(){
```

SweetAlert

```
102     Swal.fire({
103         toast: true,
104         title: "Anda mendapat 1 pesan baru",
105         icon: 'info',
106         position: 'top-end',
107         showConfirmButton: false,
108         showCloseButton: true,
109     })
110 }
111
112 function alertToast2(){
113     Swal.fire({
114         toast: true,
115         title: "Anda mendapat 1 pesan baru",
116         icon: 'info',
117         position: 'top-end',
118         showConfirmButton: false,
119         timer: 3000,
120         timerProgressBar: true,
121     })
122 }
123
124 // Perlu tambahan animate.css
125 function alertAnimate(){
126     Swal.fire({
127         title: 'Selamat!',
128         icon: 'success',
129         text: 'Anda berhasil menjalankan Sweet Alert',
130         showClass: {
131             popup: 'animate__animated animate__jackInTheBox animate__fast'
132         },
133         hideClass: {
134             popup: 'animate__animated animate__hinge animate__slow'
135         }
136     })
137 }
138
139 </script>
140 </div>
141 </body>
142 </html>
```

Di baris 8 dan 9 terdapat tambahan tag `<link>` untuk mengakses file CSS `bootstrap.min.css` dan `animate.css`. Kemudian di bagian konten saya membuat 9 tag `<button>` yang ketika diklik akan memproses berbagai function.

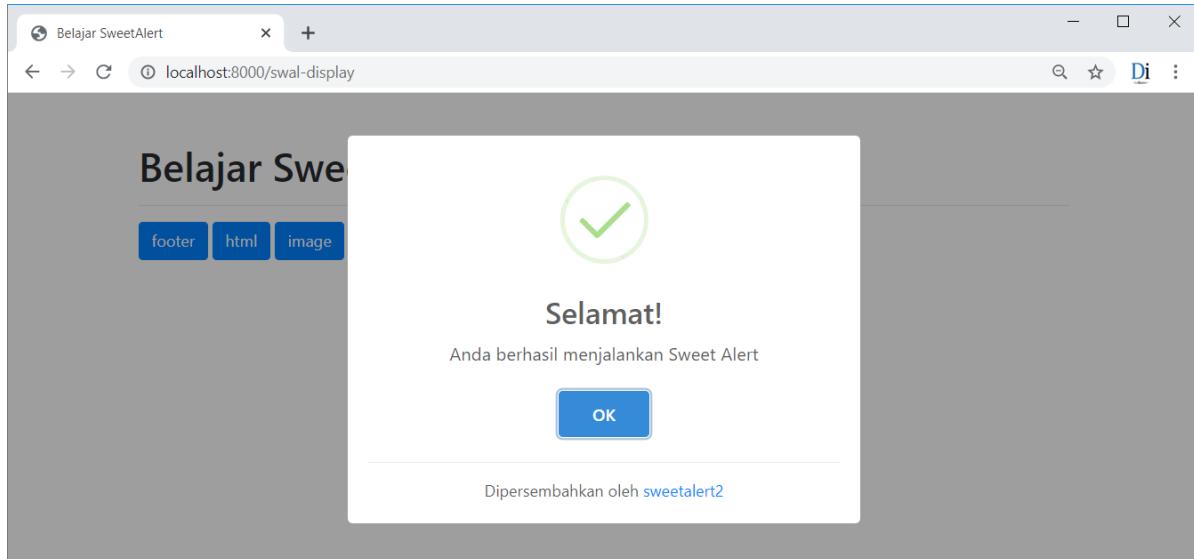
Pada saat button **footer** diklik (baris 15), JavaScript akan menjalankan function `alertFooter()`. Isi function ini tetap memanggil method `Swal.fire()`, tapi sekarang dengan argument berbentuk JavaScript object yang terdiri dari pasangan property dan value.

Property `icon` di baris 30 berfungsi untuk menentukan jenis icon, yakni salah satu string yang kita bahas sebelumnya. Dalam contoh ini saya menggunakan icon 'success'. Property `title`

SweetAlert

dipakai untuk membuat judul jendela alert, serta property `teks` untuk membuat teks jendela alert. Ketiga property ini sudah kita gunakan sebelumnya.

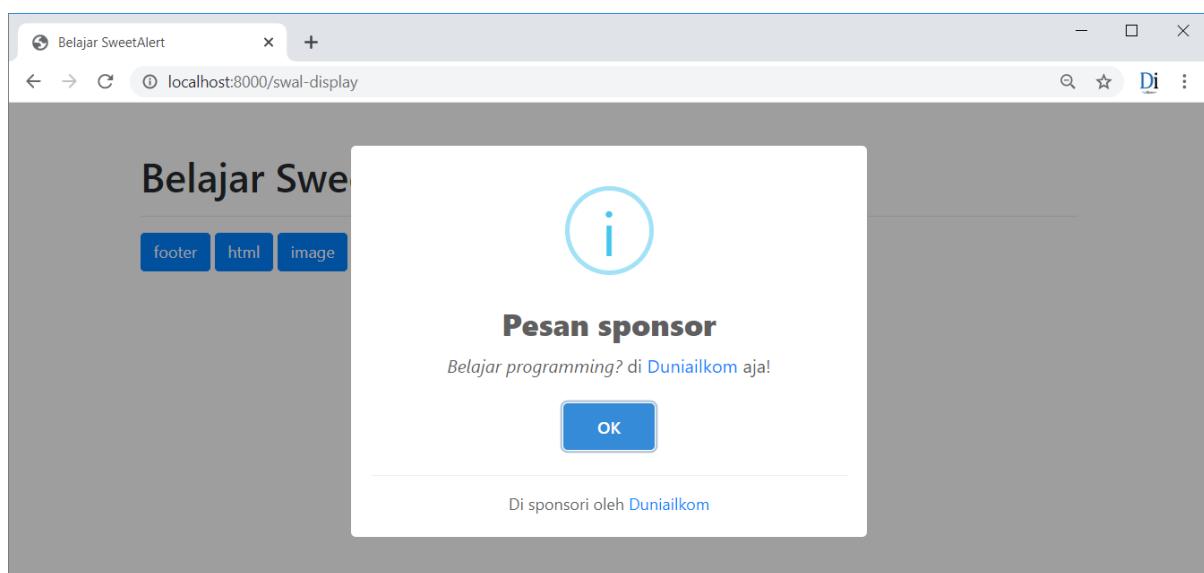
Tambahan pengaturan baru ada di property `footer` (baris 33) yang dipakai untuk menambah teks pada bagian footer jendela alert. Di footer ini kita bisa menulis tag `<html>` seperti `` dan `<a>`. Berikut tampilan jendela alert pada saat tombol di klik:



Gambar: Tampilan pada saat tombol 'footer' di klik

Selain teks 'Dipersembahkan oleh sweetalert2' pada bagian bawah, font yang dipakai juga sedikit berubah karena tambahan kode CSS Bootstrap.

Jika di dalam teks alert kita ingin menulis kode HTML, bisa menulisnya di property `html` seperti baris 43. Di situ saya menambah tag `<i>` dan tag `<a>` ke dalam teks. Berikut tampilan saat tombol **html** di klik:



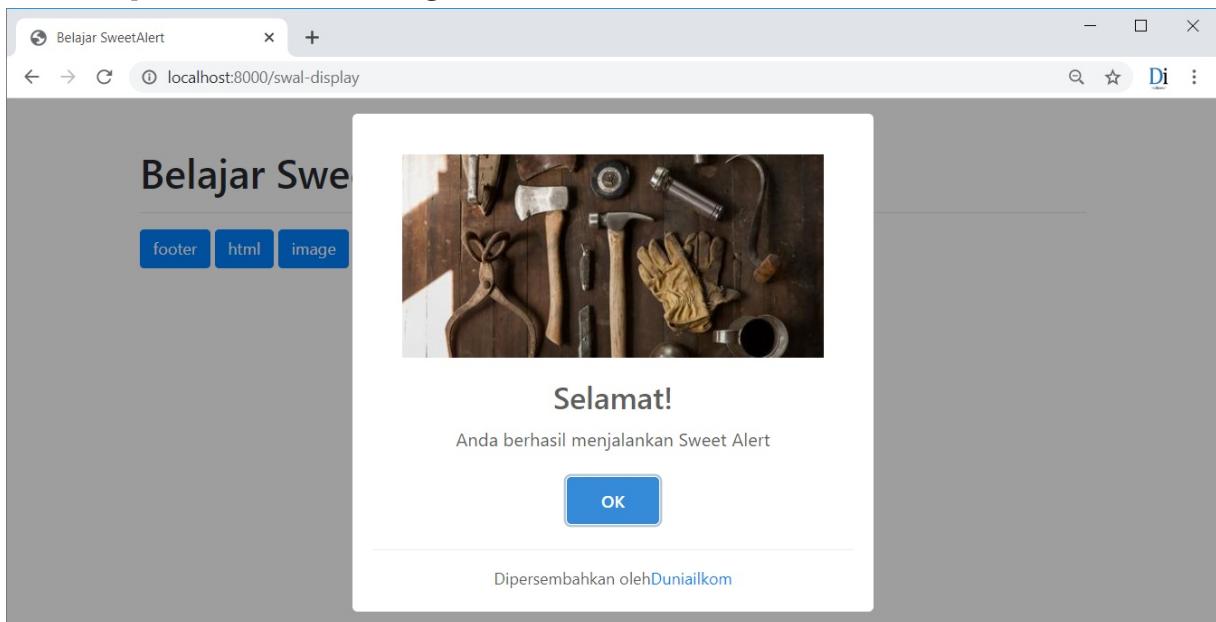
Gambar: Tampilan pada saat tombol 'html' di klik

Untuk tombol **image**, saya menambah 4 property sekaligus antara baris 56-59. Property **imageUrl** dipakai untuk menulis alamat gambar yang akan menggantikan icon. Kali ini saya mengambil gambar acak dari <https://picsum.photos/450/200>.

Web <https://picsum.photos> adalah website *image generator*, dimana angka 450 dan 200 merujuk ke lebar dan tinggi gambar. Jika anda butuh gambar dummy saat pembuatan project, bisa menggunakan layanan ini agar tidak repot harus crop gambar. Namun kekurangannya selama pembuatan project harus selalu terhubung ke internet.

Lanjut, property **imageWidth** dan **imageHeight** dipakai untuk mengatur lebar dan tinggi gambar. Serta property **imageAlt** dipakai untuk mengisi atribut alt, yakni teks yang tampil jika gambar gagal di akses.

Berikut tampilan saat tombol **image** di klik:



Gambar: Tampilan pada saat tombol 'image' di klik

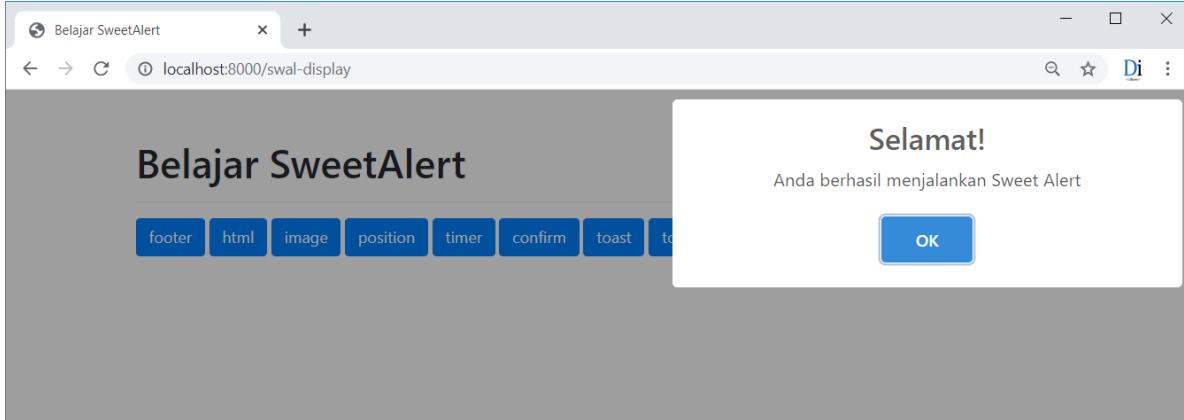
Gambar yang anda dapati kemungkinan besar akan berbeda karena di acak oleh **picsum.photos**. Atau kita juga bisa memakai gambar yang di upload sendiri.

SweetAlert2 juga menyediakan pengaturan mengenai posisi di mana jendela alert akan tampil. Ini diatur dari property **position** seperti contoh di baris 67. Secara default, jendela tampil di tengah halaman, yakni sama dengan nilai 'center'.

Nilai yang bisa dipakai untuk property **position** ada 9, yakni 'top', 'top-start', 'top-end', 'center', 'center-start', 'center-end', 'bottom', 'bottom-start', dan 'bottom-end'.

Berikut tampilan saat tombol **position** di klik:

SweetAlert

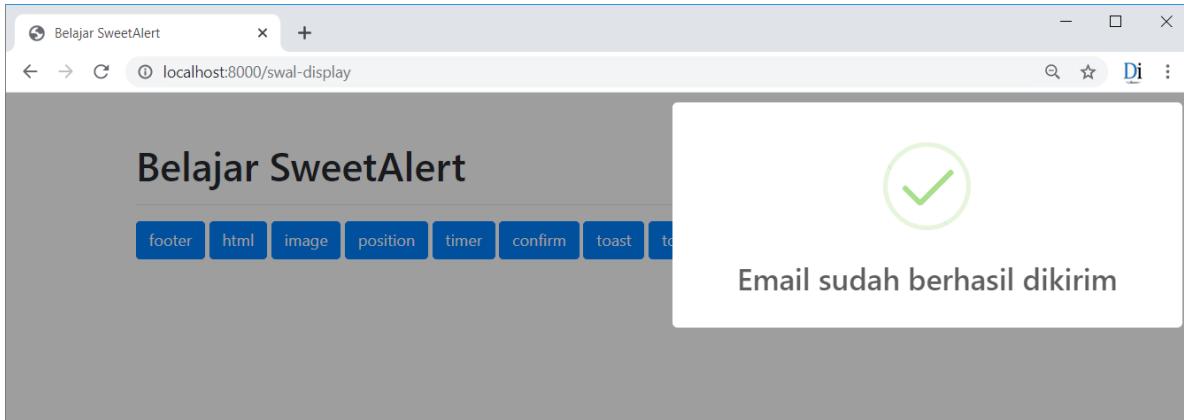


Gambar: Tampilan pada saat tombol 'position' di klik

Jendela alert juga bisa dibuat agar tertutup secara otomatis (tanpa harus men-klik tombol OK). Caranya, tambah property `timer` seperti di baris 77. Nilai untuk property `timer` berupa angka dalam satuan millisecond. Dalam contoh ini saya mengisi angka 1500, sehingga jendela alert akan tertutup otomatis setelah 1,5 detik.

Tambahan property `showConfirmButton: false` berguna untuk menyembunyikan tombol OK. Karena jika jendela alert bisa tertutup dengan cepat, kita tidak perlu menampilkan tombol OK.

Berikut tampilan jendela alert saat tombol **timer** di klik:



Gambar: Tampilan pada saat tombol 'timer' di klik

Selain jendela alert yang berisi pesan, untuk beberapa kasus kita ingin menampilkan jendela konfirmasi, yakni jendela popup yang berisi 2 buah tombol.

Jendela konfirmasi dipakai agar user benar-benar setuju dengan tindakan yang akan diambil, misalnya saat menghapus sebuah data. Untuk membuat jendela konfirmasi, penulisannya sedikit lebih panjang seperti di baris 81 - 99.

Tiga property pertama sudah sering kita pakai, yakni `title`, `text` dan `icon`. Kemudian property `showCancelButton:true` berguna untuk menampilkan tombol cancel.

Property `confirmButtonColor` dan `cancelButtonColor` dipakai untuk menentukan warna dari tombol `confirm` dan tombol `cancel`. Kedua property ini butuh nilai string berbentuk kode

SweetAlert

warna CSS seperti '#3085d6' atau '#d33'.

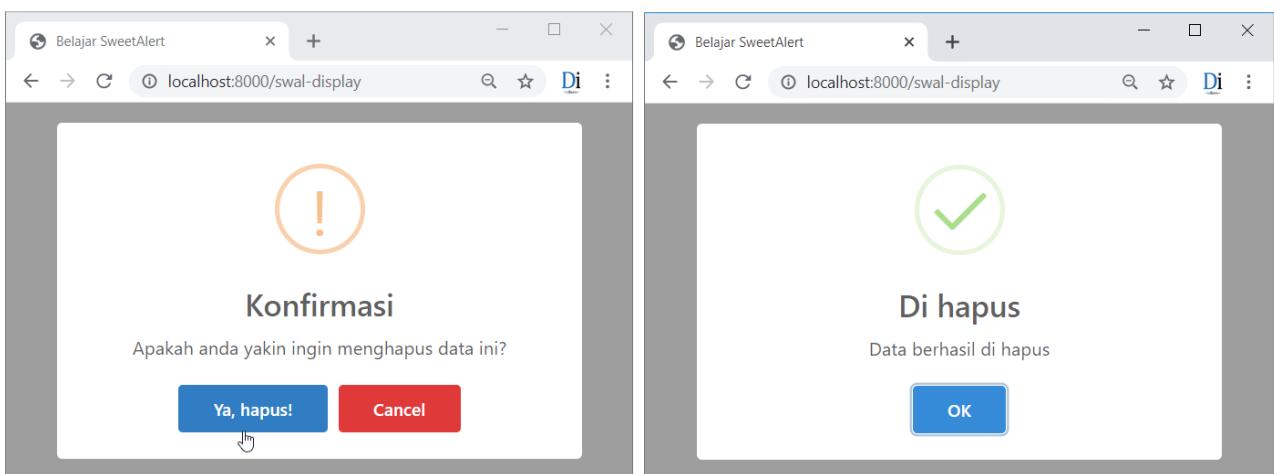
Di baris 89, property `confirmButtonText` berguna untuk mengganti teks tombol confirm dari 'OK' menjadi teks lain yang dalam contoh ini diisi teks 'Ya, hapus!'.

Penulisan method `Swal.fire()` sebenarnya sudah selesai di baris 90, namun karena ini adalah jendela konfirmasi, kita perlu menentukan apa yang terjadi saat tombol confirm di klik.

Caranya, sambung penulisan method `Swal.fire()` dengan method `then((result) => {})`.

Dalam method `then()` inilah kode lanjutan ditulis, misalnya menjalankan jalankan lagi method `Swal.fire()` untuk menampilkan jendela sukses (baris 92 – 95). Kode di bagian ini umumnya akan lebih kompleks, seperti mengeksekusi data AJAX atau redirect ke halaman lain.

Berikut tampilan saat tombol **confirm** di klik:



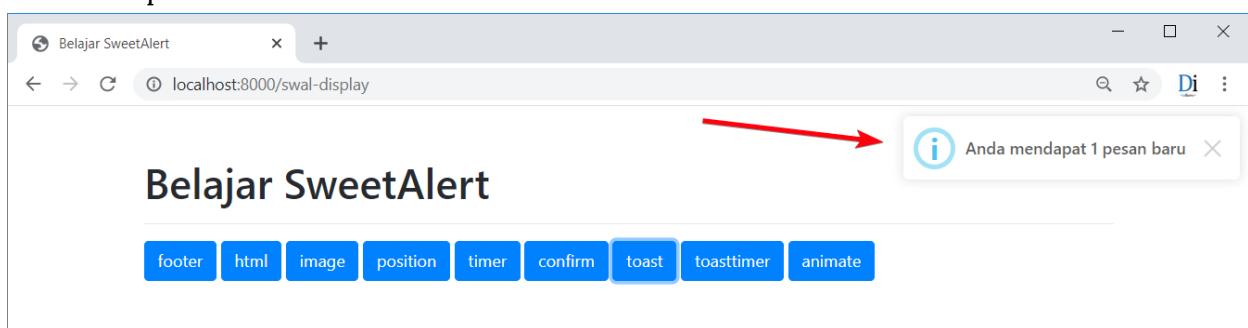
Gambar: Tampilan pada saat tombol 'confirm' di klik

Selain untuk jendela alert dan konfirmasi, SweetAlert2 juga menyediakan jendela **toast**.

Jendela toast ini lebih kecil dan lebih sederhana dibandingkan jendela alert biasa.

Cara membuat jendela toast adalah dengan menambah property `toast:true` seperti di baris 103. Selain itu saya juga menambah property `showConfirmButton:false` untuk menyembunyikan tombol confirm, serta property `showCloseButton:true` untuk menampilkan tombol close (tanda silang di akhir toast).

Berikut tampilan saat tombol **toast** di klik:



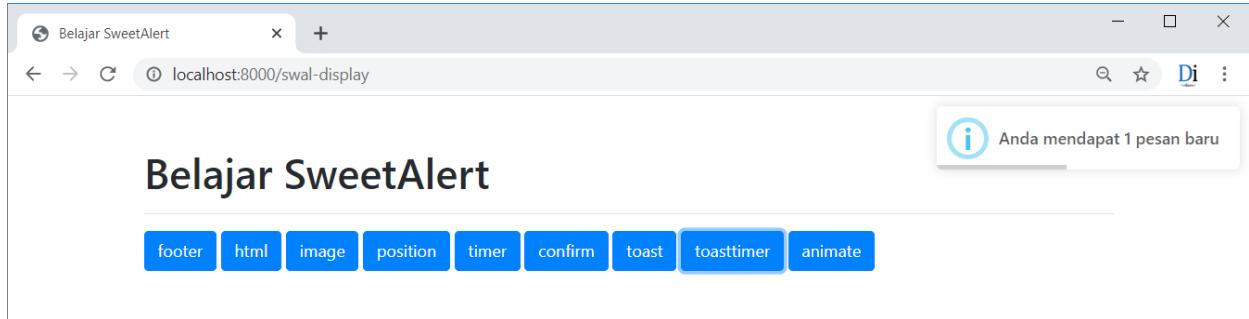
Gambar: Tampilan pada saat tombol 'toast' di klik

SweetAlert

Jendela toast ini bisa di pakai untuk menampilkan notifikasi, misalnya saat user mendapat pesan baru atau ada update dari database.

Toast juga sangat cocok dipasangkan dengan timer yang langsung tertutup setelah beberapa saat. Caranya, tambah property `timer` seperti di baris 119. Untuk kali ini saya juga menambah property `timerProgressBar:true` agar muncul garis progress di bagian bawah toast.

Berikut tampilan saat tombol **toasttimer** di klik:

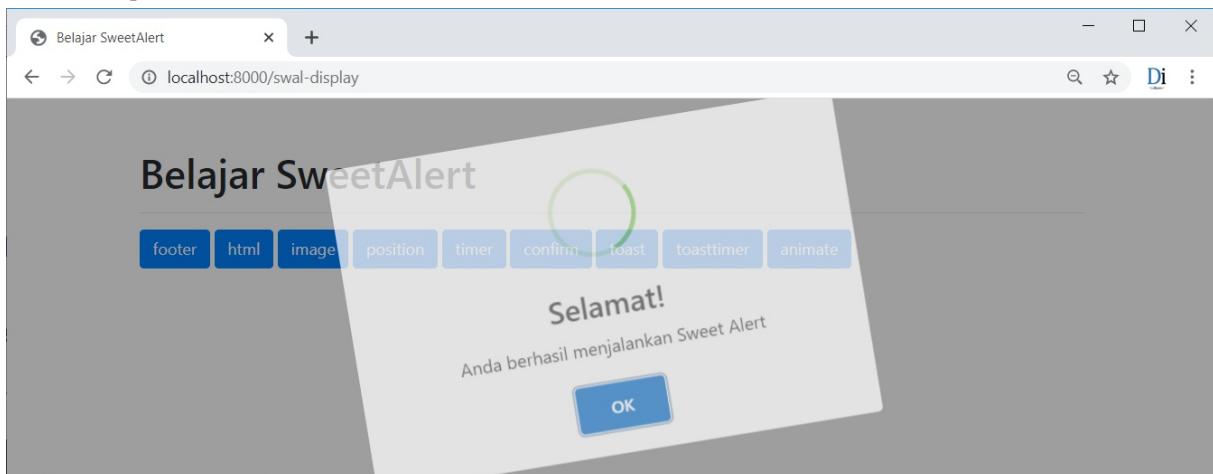


Gambar: Tampilan pada saat tombol 'toasttimer' di klik

Untuk alert terakhir, saya menambah property `showClass` di baris 131 dan `hideClass` di baris 133. Kedua property diisi dengan string nama class milik Animate.css. Property `showClass` berisi efek animasi pada saat jendela alert tampil, dan property `hideClass` diisi efek yang dijalankan pada saat jendela ditutup.

Penjelasan tentang nama class ini tidak akan saya bahas karena bisa cukup panjang, untuk lengkapnya bisa mengunjungi <https://animate.style>.

Berikut tampilan saat tombol **animate** di klik:



Gambar: Tampilan pada saat tombol 'animate' di klik

Itulah beberapa penggunaan dasar dari **SweetAlert2**. Pengaturan serta efek-efek lain bisa di pelajari ke dokumentasi resminya di <https://sweetalert2.github.io>.

22.6. Instalasi realrashid/sweetalert

Kita sudah bisa menjalankan SweetAlert2 dari dalam Laravel, akan tetapi masih cukup ribet saat diterapkan untuk aplikasi yang sebenarnya. Sebagai contoh, jika saya ingin menampilkan jendela alert saat data berhasil di input ke database. dimanakah kode ini harus ditulis?

Secara umum, proses input ke database dilakukan dari controller (menggunakan eloquent). Setelah itu user akan di redirect ke halaman lain, misalnya `index.blade.php` yang menampilkan semua isi tabel. Di halaman `index.blade.php` inilah kita harus tulis kode JavaScript untuk mendeteksi user baru saja selesai input data dan tampilkan pesan alert.

Pendeteksian ini bisa dilakukan dengan membuat sebuah *flash session*. Jika session ditemukan, tampilkan jendela alert, jika tidak ditemukan (berarti halaman `index.blade.php` di buka manual), maka jangan tampilkan alert.

Untuk membuat proses ini lebih mudah (terutama bagi yang belum mendalami JavaScript), tersedia library khusus untuk integrasi SweetAlert2 dengan Laravel. Salah satunya adalah **realrashid/sweetalert** yang beralamat di realrashid.github.io/sweetalert.

Dengan memakai libary ini, kita bisa menampilkan jendela SweetAlert2 tanpa menulis perintah JavaScript sama sekali.

Untuk menginstall `realrashid/sweetalert`, bisa dilakukan dengan perintah composer. Silahkan masuk ke folder instalasi Laravel, lalu ketik kode berikut:

```
composer require realrashid/sweetalert
```

```
C:\xampp\htdocs\laravel02>composer require realrashid/sweetalert
Using version ^3.1 for realrashid/sweetalert
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing realrashid/sweetalert (v3.1.6): Downloading (100%)
```

Gambar: Proses instalasi `realrashid/sweetalert`

Setelah selesai, ketik lagi perintah berikut:

```
php artisan sweetalert:publish
```

```
C:\xampp\htdocs\laravel02>php artisan sweetalert:publish
Copied File [\vendor\realrashid\sweetalert\src\config\sweetalert.php] To [\config\sweetalert.php]
Publishing complete.
Copied Directory [\vendor\realrashid\sweetalert\resources\views] To [\resources\views\vendor\sweetalert]
Publishing complete.
Copied Directory [\vendor\realrashid\sweetalert\resources\js] To [\public\vendor\sweetalert]
Publishing complete.
```

Gambar: Menjalankan perintah `php artisan sweetalert:publish`

SweetAlert

Perintah ini dipakai untuk men-copy file SweetAlert2 dari folder vendor ke beberapa folder lain agar bisa di akses.

Sebagai bahan percobaan, saya akan buat satu file controller dengan perintah berikut:

```
php artisan make:controller SwalController
```

Di dalam file `SwalController.php` inilah kita akan bahas cara penggunaan `realrashid/sweetalert`.

22.7. Realrashid/sweetalert Facade

Terdapat beberapa cara untuk menampilkan jendela alert dari `realrashid/sweetalert`. Cara pertama adalah dengan menjalankan static method `alert()` dari facade class **Alert**.

Untuk bahan praktik, silahkan tambah route berikut ke dalam file `routes\web.php`:

```
routes\web.php
```

```
1 ...  
2 use App\Http\Controllers\SwalController;  
3  
4 Route::get('/swal-alert-success', [SwalController::class, 'alertSuccess']);  
5 Route::get('/swal-alert-info', [SwalController::class, 'alertInfo']);  
6 Route::get('/swal-success', [SwalController::class, 'success']);  
7 Route::get('/swal-info', [SwalController::class, 'info']);  
8 Route::get('/swal-html', [SwalController::class, 'html']);  
9 Route::get('/swal-toast', [SwalController::class, 'toast']);
```

Dari nama route ini mungkin anda sudah bisa menebak bahwa beberapa dipakai untuk menampilkan pesan alert untuk icon success, info, dengan kode html serta toast.

Lanjut, silahkan buka file `SwalController.php` lalu modifikasi sebagai berikut:

```
app\Http\Controllers\SwalController.php
```

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4 use RealRashid\SweetAlert\Facades\Alert;  
5  
6 class SwalController extends Controller  
7 {  
8     public function alertSuccess()  
9     {  
10         Alert::alert('Selamat!', 'Anda berhasil menjalankan Sweet Alert', 'success');  
11         return view('swal-laravel');  
12     }  
13  
14     public function alertInfo()  
15     {  
16         Alert::alert('Selamat!', 'Anda berhasil menjalankan Sweet Alert', 'info');
```

SweetAlert

```
17     return view('swal-laravel');
18 }
19
20 public function success()
21 {
22     Alert::success('Selamat!', 'Anda berhasil menjalankan Sweet Alert');
23     return view('swal-laravel');
24 }
25
26 public function info()
27 {
28     Alert::info('Selamat!', 'Anda berhasil menjalankan Sweet Alert');
29     return view('swal-laravel');
30 }
31
32 public function html()
33 {
34     Alert::html('<strong>Pesan sponsor</strong>',
35                 '<i>Belajar programming?</i> di <a href="https://www.duniaikom.com">
36                 Duniaikom </a> aja!', 'success');
37     return view('swal-laravel');
38 }
39
40 public function toast()
41 {
42     Alert::toast('Anda mendapat 1 pesan baru', 'info');
43     return view('swal-laravel');
44 }
45 }
```

Di baris 4 terdapat perintah `use RealRashid\SweetAlert\Facades\Alert` untuk proses import facade class **Alert**. Ini harus ditulis agar kita bisa mengakses berbagai method bawaan class **Alert**.

Method controller pertama, `alertSuccess()` di baris 8 – 12 berisi pemanggilan `Alert::alert()`. Method inilah yang dipakai untuk menampilkan jendela sweetalert.

Method `Alert::alert()` butuh 3 buah argument dengan urutan yang sama seperti method `Swal.fire()` milik SweetAlert2, yakni argument pertama untuk judul alert, argument kedua untuk bagian penjelasan, dan argument ketiga untuk jenis icon.

Sebagai contoh, perintah `Alert::alert('Selamat!', 'Anda berhasil menjalankan Sweet Alert', 'success')` di baris 10 akan menampilkan jendela alert dengan judul 'Selamat!', isi teks 'Anda berhasil menjalankan Sweet Alert', serta icon ceklist hijau.

Jendela alert akan tampil saat view 'swal-laravel' di akses. View tersebut saat ini belum tersedia dan akan kita buat sesaat lagi.

Lanjut ke method `Alert::alert()` kedua di baris 16. Perbedaan dengan method di baris 10 hanya di argument ketiga, dimana sekarang saya menggunakan icon 'info'.

Class `Alert()` juga menyediakan cara pemanggilan lain, yakni langsung menulis jenis icon

SweetAlert

seperti Alert::success() di baris 22 serta Alert::info() di baris 28. Untuk kedua method ini, kita hanya perlu menulis 2 buah argument untuk judul dan teks alert.

Seperti yang bisa ditebak, tersedia juga method Alert::warning(), Alert::error(), dan Alert::question() dengan susunan argument yang sama. Perbedaan hanya di bagian icon saja.

Lanjut di baris 34, terdapat method Alert::html() yang bisa dipakai jika kita ingin menulis tag HTML ke dalam judul dan pesan alert. Method ini butuh 3 argument yang urutannya sama seperti method Alert::alert().

Terakhir di baris 42 saya menjalankan method Alert::toast() yang bisa dipakai untuk membuat jendela toast. Method ini butuh 2 argument berupa pesan teks dan jenis icon.

Semua method di SwalController.php akan mengakses view swal-laravel, oleh sebab itu silahkan buat file swal-laravel.blade.php di folder resources\views\ dan isi dengan kode berikut:

resources\views\swal-laravel.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      <title>Belajar SweetAlert</title>
7      <link rel="stylesheet" href="/css/bootstrap.min.css">
8  </head>
9  <body>
10     <div class="container mt-5">
11         <h1>Belajar SweetAlert</h1>
12         <hr>
13     </div>
14     @include('sweetalert::alert')
15 </body>
16 </html>
```

Syarat agar view ini bisa menampilkan jendela sweetalert adalah tambahan perintah @include('sweetalert::alert') seperti di baris 14. Inilah cara library RealRashid\SweetAlert untuk men-inject kode JavaScript milik sweetalert.

Jika anda perhatikan, di dalam folder resources\views\ sekarang terdapat folder vendor\sweetalert yang diberisi file alert.blade.php. File inilah yang diakses oleh perintah @include('sweetalert::alert').

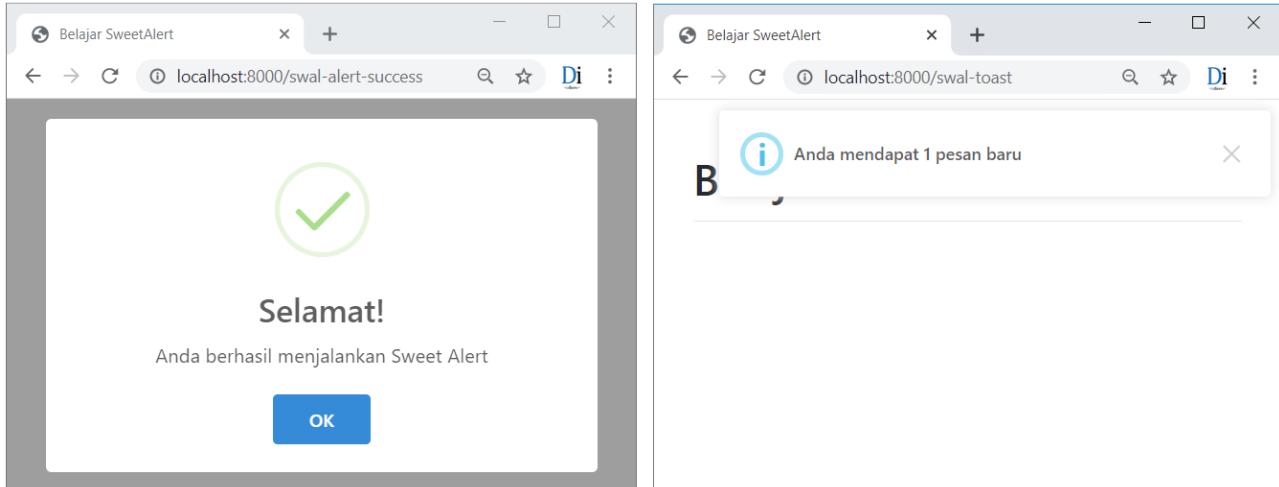
Untuk uji coba, silahkan akses URL berikut:

- ◆ <http://localhost:8000/swal-alert-success>
- ◆ <http://localhost:8000/swal-alert-info>
- ◆ <http://localhost:8000/swal-success>

SweetAlert

- ◆ <http://localhost:8000/swal-info>
- ◆ <http://localhost:8000/swal-html>
- ◆ <http://localhost:8000/swal-toast>

Maka akan tampil jendela alert yang kita tulis di controller sebelumnya.



Gambar: Tampilan jendela alert dari RealRashid\SweetAlert

Bawaan dari RealRashid\SweetAlert, jendela alert akan langsung tertutup jika tidak di klik dalam waktu 5 detik. Ini bisa diatur dengan menambah method `autoClose()` yang akan kita bahas sesaat lagi, atau bisa juga set secara global dari file konfigurasi di `config\sweetalert.php`

File `config\sweetalert.php` berisi pengaturan lanjutan dari RealRashid\SweetAlert, misalnya apakah ingin menggunakan link CDN, mengatur tinggi jendela alert, posisi jendela, dll. Khusus untuk pengaturan timer, bisa di set dari baris berikut:

```
'timer' => env('SWEET_ALERT_TIMER', 5000),
```

Angka 5000 bisa diganti dengan nilai lain atau **false** untuk menonaktifkan timer:

```
'timer' => env('SWEET_ALERT_TIMER', false),
```

22.8. Realrashid/sweetalert Helper Function

Cara lain untuk menampilkan jendela alert adalah dengan **helper function**. Untuk contoh praktek, silahkan tambah route berikut:

```
routes\web.php
```

```
1 ...  
2 Route::get('/swal-helper-alert', [SwalController::class, 'helperAlert']);  
3 Route::get('/swal-helper-success', [SwalController::class, 'helperSuccess']);  
4 Route::get('/swal-helper-info', [SwalController::class, 'helperInfo']);  
5 Route::get('/swal-helper-toast', [SwalController::class, 'helperToast']);
```

Masuk ke file `SwalController.php` dan tambah method berikut:

`app\Http\Controllers\SwalController.php`

```

1 ...
2
3     public function helperAlert()
4     {
5         alert('Selamat!', 'Anda berhasil menjalankan Sweet Alert', 'success');
6         return view('swal-laravel');
7     }
8
9     public function helperSuccess()
10    {
11        alert()->success('Selamat!', 'Anda berhasil menjalankan Sweet Alert');
12        return view('swal-laravel');
13    }
14
15    public function helperInfo()
16    {
17        alert()->info('Selamat!', 'Anda berhasil menjalankan Sweet Alert');
18        return view('swal-laravel');
19    }
20
21    public function helperToast()
22    {
23        toast('Menu yang dipilih tidak tersedia', 'warning');
24        return view('swal-laravel');
25    }

```

Cara penggunaan function `alert()` di baris 5 sama seperti method `Alert::alert()`, yakni butuh 3 argument untuk judul, teks dan jenis icon.

Alternatif penulisan bisa lewat function `alert()->success()` dan `alert()->info()` seperti di baris 11 dan 17. Untuk pemanggilan ini, butuh 2 argument untuk judul dan teks alert. Terakhir, function `toast()` dipakai untuk membuat jendela toast seperti di baris 23.

Untuk uji coba, silahkan akses URL berikut:

- ◆ <http://localhost:8000/swal-helper-alert>
- ◆ <http://localhost:8000/swal-helper-success>
- ◆ <http://localhost:8000/swal-helper-info>
- ◆ <http://localhost:8000/swal-helper-toast>

22.9. Realrashid/sweetalert Helper Method

Untuk membuat jendela alert yang lebih fleksibel, Realrashid/sweetalert juga menyediakan beberapa helper method. Sama seperti sebelumnya, untuk contoh praktek silahkan tambah route berikut:

SweetAlert

routes\web.php

```
1 ...  
2 Route::get('/swal-autoclose', [SwalController::class, 'autoClose']);  
3 Route::get('/swal-position', [SwalController::class, 'position']);  
4 Route::get('/swal-confirm', [SwalController::class, 'confirm']);  
5 Route::get('/swal-cancel', [SwalController::class, 'cancel']);  
6 Route::get('/swal-addimage', [SwalController::class, 'addImage']);  
7 Route::get('/swal-animation', [SwalController::class, 'animation']);  
8 Route::get('/swal-progressbar', [SwalController::class, 'progressBar']);
```

Kemudian berikut kode untuk file SwalController.php:

app\Http\Controllers\SwalController.php

```
1 ...  
2  
3 public function autoClose()  
4 {  
5     Alert::success('Selamat!', 'Anda berhasil menjalankan Sweet Alert')  
6         ->autoClose(1000);  
7     return view('swal-laravel');  
8 }  
9  
10 public function position()  
11 {  
12     Alert::success('Selamat!', 'Anda berhasil menjalankan Sweet Alert')  
13         ->position('bottom-end');  
14     return view('swal-laravel');  
15 }  
16  
17 public function confirm()  
18 {  
19     Alert::warning('Konfirmasi', 'Apakah anda yakin akan menghapus data ini?')  
20         ->showConfirmButton('Ya, saya yakin', 'red');  
21     return view('swal-laravel');  
22 }  
23  
24 public function cancel()  
25 {  
26     Alert::warning('Konfirmasi', 'Apakah anda yakin akan menghapus data ini?')  
27         ->showConfirmButton('Ya, saya yakin', 'red')  
28         ->showCancelButton('Tidak jadi', '#aaa');  
29     return view('swal-laravel');  
30 }  
31  
32 public function addImage()  
33 {  
34     Alert::success('Selamat!', 'Anda berhasil menjalankan Sweet Alert')  
35         ->addImage('https://picsum.photos/450/200');  
36     return view('swal-laravel');  
37 }  
38  
39 public function animation()  
40 {
```

SweetAlert

```
41     Alert::success('Selamat!', 'Anda berhasil menjalankan Sweet Alert')
42         ->animation('animate_animated animate_tada animate_faster',
43                         'animate_animated animate_rollOut animate_slower');
44     return view('swal-laravel');
45 }
46
47 public function progressBar()
48 {
49     Alert::success('Selamat!', 'Anda berhasil menjalankan Sweet Alert')
50         ->timerProgressBar()->autoClose(3000);
51     return view('swal-laravel');
52 }
```

Apa yang dilakukan oleh method-method ini sebenarnya sama seperti fitur bawaan SweetAlert2, hanya saja sekang ditulis menggunakan method. Penulisannya juga bisa di *chaining* satu sama lain untuk mendapatkan efek gabungan.

Di baris 6 saya menambah method `autoClose(1000)` ke dalam `Alert::success()`. Ini membuat jendela alert akan tertutup otomatis setelah 1 detik atau 1000 millisecond.

Di baris 13, terdapat tambahan method `position('bottom-end')` yang berfungsi untuk menentukan posisi jendela alert. Nilai 'bottom-end' berarti jendela alert akan tampil di sudut kanan bawah.

Pada baris 20, method `showConfirmButton()` dipakai untuk mengubah teks dan warna tombol. Dengan perintah ini, tombol alert akan tampil dalam warna merah dan memiliki tulisan 'Ya, saya yakin'. Sedangkan untuk menambah tombol cancel, bisa memakai method `showCancelButton()` seperti di baris 28.

Jika ingin menambah gambar, bisa menggunakan method `addImage()` seperti di baris 35. Hasilnya, icon success akan tertimpa dengan gambar dari <https://picsum.photos>.

Untuk membuat animasi tersedia method `animation()` yang ada baris 42. Method ini butuh 2 buah argument berupa nama class Animate.css dalam bentuk string. Khusus untuk efek animasi ini, kita harus ubah pengaturan 'animation' menjadi **true** di file konfigurasi config\sweetalert.php:

```
1     'animation' => [
2         'enable' => env('SWEET_ALERT_ANIMATION_ENABLE', true),
3     ],
```

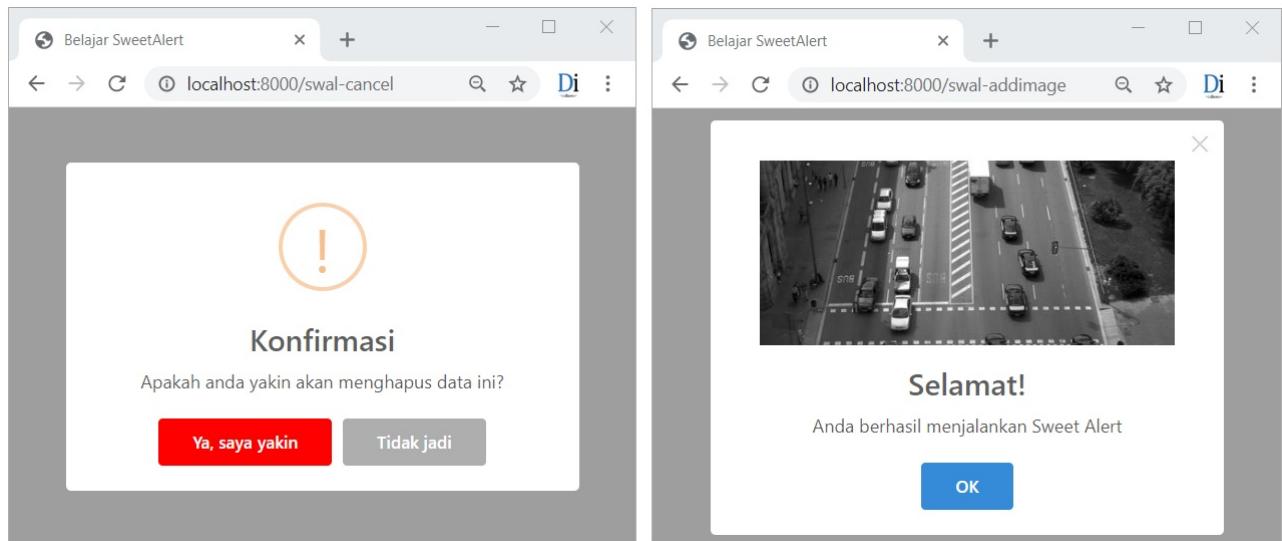
Terakhir di baris 49 saya memakai method `timerProgressBar()` untuk membuat garis progress. Ini harus dipadukan dengan method `autoClose()` supaya jendela alert tertutup otomatis.

Untuk uji coba, bisa dari URL berikut:

- ◆ <http://localhost:8000/swal-autoclose>
- ◆ <http://localhost:8000/swal-position>

SweetAlert

- ◆ <http://localhost:8000/swal-confirm>
- ◆ <http://localhost:8000/swal-cancel>
- ◆ <http://localhost:8000/swal-addimage>
- ◆ <http://localhost:8000/swal-animation>
- ◆ <http://localhost:8000/swal-progressbar>



Gambar: Tampilan jendela alert dengan penggunaan helper method

Selain apa yang kita bahas di sini, masih ada beberapa helper method lain dari Realrashid/sweetalert. Lebih lengkapnya bisa ke [realrashid.github.io/sweetalert.](https://realrashid.github.io/sweetalert/)

22.10. Realrashid/sweetalert Flash Session

Dari beberapa praktek sebelum ini, kita sudah berhasil menjalankan sweetalert dari dalam controller untuk ditampilkan ke view. Akan tetapi setiap kali URL tersebut di akses, jendela alert akan selalu muncul.

Umumnya, kita hanya ingin jendela alert tampil 1 kali saja, misal setelah data diinput ke database dan user di redirect ke halaman index. Namun jika halaman index langsung di akses tanpa input data, jendela alert mestinya tidak tampil.

Realrashid/sweetalert menyediakan fitur khusus untuk masalah ini, yaitu dengan memanfaatkan *flash session* atau *flash message* (pesan yang langsung terhapus setelah di akses).

Untuk bisa menggunakannya, Realrashid/sweetalert harus di daftarkan ke **group middleware** terlebih dahulu. Silahkan buka file `app\Http\Kernel.php`, cari array `$middlewareGroups`, lalu tambah baris berikut ke element 'web':

```
\RealRashid\SweetAlert\ToSweetAlert::class,
```

Berikut posisi penulisannya:

```

26  /**
27  * The application's route middleware groups.
28  *
29  * @var array
30  */
31 protected $middlewareGroups = [
32     'web' => [
33         \App\Http\Middleware\EncryptCookies::class,
34         \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
35         \Illuminate\Session\Middleware\StartSession::class,
36         // \Illuminate\Session\Middleware\AuthenticateSession::class,
37         \Illuminate\View\Middleware\ShareErrorsFromSession::class,
38         \App\Http\Middleware\VerifyCsrfToken::class,
39         \Illuminate\Routing\Middleware\SubstituteBindings::class,
40         \RealRashid\SweetAlert\ToSweetAlert::class,
41     ],

```

Gambar: Tambah \RealRashid\SweetAlert\ToSweetAlert::class ke file Kernel.php

Sebagai bahan praktek, tambah 3 route berikut ke file routes\web.php:

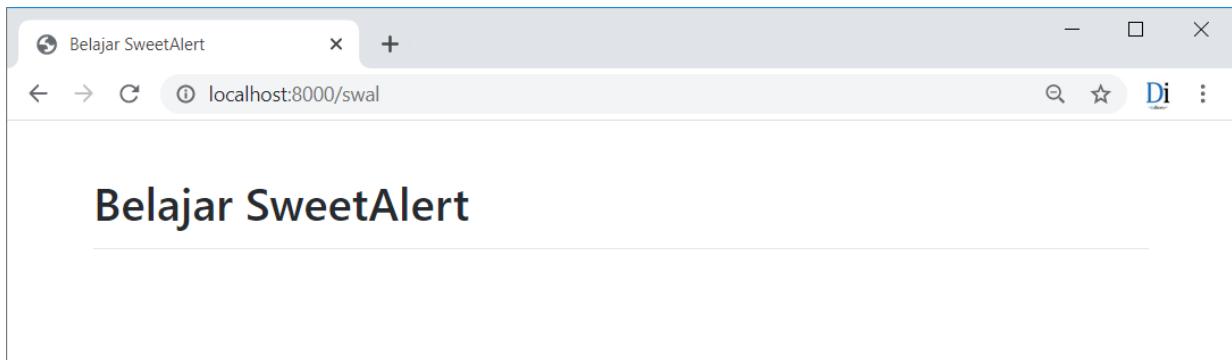
routes\web.php

```

1 ...
2 Route::view('/swal', 'swal-laravel');
3 Route::get('/swal-with', [SwalController::class, 'with']);
4 Route::get('/swal-with-success', [SwalController::class, 'withSuccess']);

```

Route 'swal' di baris 2 dipakai untuk mengakses langsung view swal-laravel.blade.php. Ini adalah view yang sudah kita buat dan berisi 1 perintah @include('sweetalert::alert'). Pada saat URL ini diakses, tidak ada jendela alert karena memang tidak ada perintah untuk itu:



Gambar: Tampilan halaman http://localhost:8000/swal

Halaman 'swal' ini akan menjadi target redirect dari route '/swal-with' dan juga '/swal-with-success'.

Pada saat route '/swal-with', itu akan menjalankan method with() di file SwalController.php. Berikut kode yang diperlukan:

app\Http\Controllers\SwalController.php

```

1 public function with()
2 {

```

SweetAlert

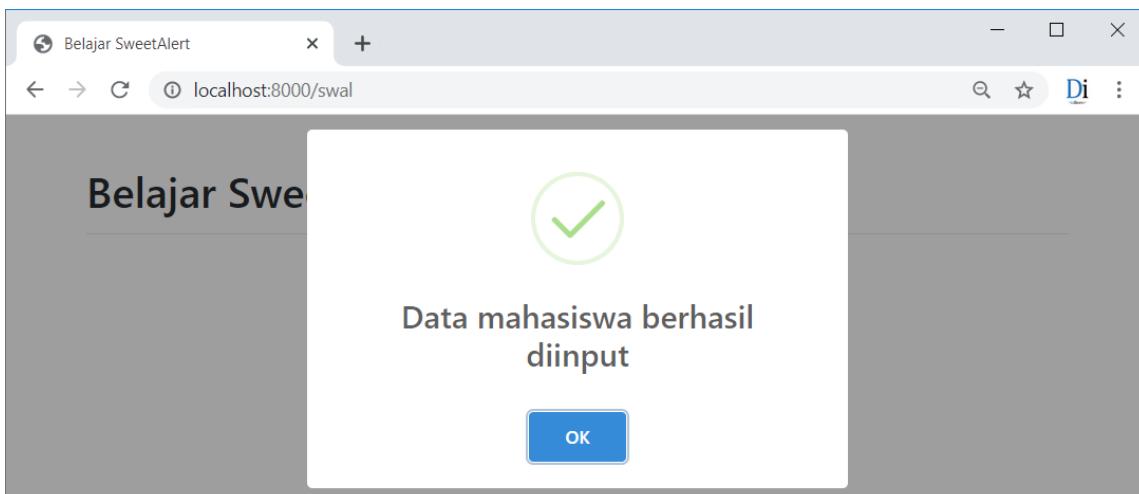
```
3     return redirect('swal')->with('success', 'Data mahasiswa berhasil diinput');
4 }
```

Di buku Laravel Uncover kita pernah membahas bahwa jika fungsi `redirect()` langsung di chaining dengan method `with()`, maka isi dari method `with()` tersebut akan menjadi sebuah flash session.

Dalam contoh di atas, begitu URL '`/swal-with`' diakses, halaman akan langsung redirect ke '`swal`', beserta flash session '`success`' yang berisi teks 'Data mahasiswa berhasil diinput'. Di dalam route '`swal`', kita bisa mengakses flash session ini, misalnya dengan perintah `session()->get('success')`.

Akan tetapi karena kita sudah mendaftarkan `Realrashid/sweetalert` ke dalam middleware, flash session tersebut akan langsung menjadi jendela alert.

Save file `SwalController.php` lalu akses URL `localhost:8000/swal-with` di web browser:



Gambar: Jendela sweetalert tampil saat URL `localhost:8000/swal-with` diakses

Perhatikan bahwa jendela alert ini tampil di halaman `localhost:8000/swal`, bukan di halaman `localhost:8000/swal-with`. Ini terjadi karena begitu `localhost:8000/swal-with` di akses, web browser langsung di redirect ke `localhost:8000/swal`, plus sebuah flash session '`success`'.

Silahkan klik tombol OK di jendela alert untuk menutupnya, lalu refresh halaman. Tidak ada lagi alert yang muncul karena flash session langsung dihapus setelah ditampilkan sekali.

String '`success`' di dalam method `with()` akan menentukan jenis icon jendela alert. Misalnya kita ingin memakai icon info, maka bisa menulisnya sebagai berikut:

```
return redirect('swal')->with('info', 'Data mahasiswa berhasil diinput');
```

Cara penulisan lain untuk membuat flash session dari method `withSuccess()`. Inilah yang menjadi method untuk route '`swal-with-success`'. Silahkan tambah method berikut ke file `SwalController.php`:

SweetAlert

app\Http\Controllers\SwalController.php

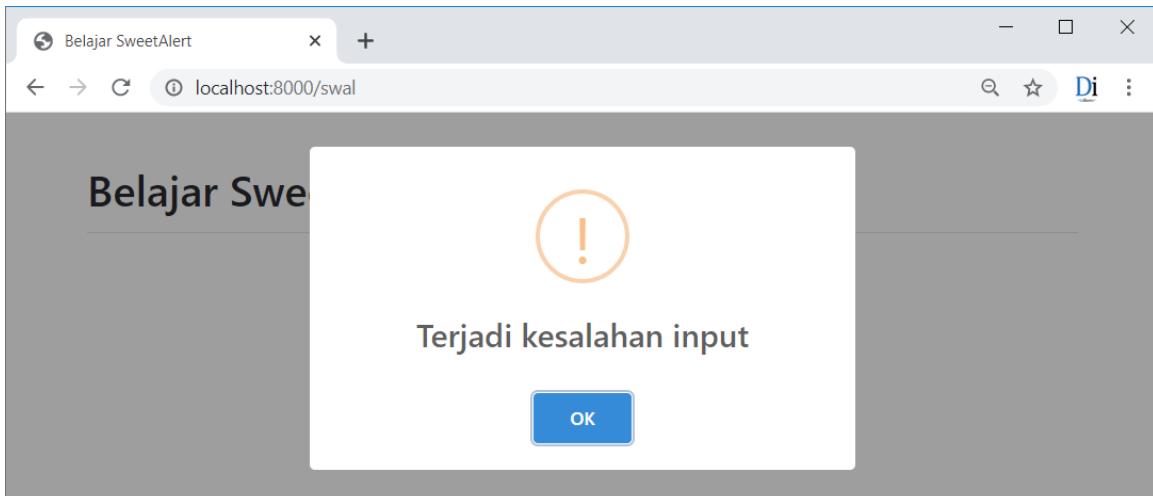
```
1 public function withSuccess()
2 {
3     return redirect('swal')->withSuccess('Data mahasiswa berhasil diinput');
4 }
```

Kemudian akses localhost:8000/swal-with-success, maka halaman akan langsung redirect ke localhost:8000/swal dan tampil pesan alert dengan icon success.

Seperti yang bisa ditebak, Realrashid/sweetalert juga menyediakan method withInfo(), withWarning(), dst. Sebagai contoh jika baris ke-3 diganti dengan kode berikut:

```
return redirect('swal')->withWarning('Terjadi kesalahan input');
```

Maka sekarang akan tampil jendela alert dengan icon tanda seru di localhost:8000/swal:



Gambar: Jendela sweetalert tampil saat URL localhost:8000/swal-with-success diakses

22.11. Realrashid/sweetalert Validation Alert

Library Realrashid/sweetalert juga menyediakan fitur untuk langsung menampilkan pesan error validasi dalam bentuk jendela alert. Secara bawaan, fitur ini tidak aktif. Untuk mengaktifkannya, silahkan tambah baris berikut ke dalam file .env:

```
SWEET_ALERT_AUTO_DISPLAY_ERROR_MESSAGES = true
```

Berikut posisi penambahan baris tersebut:

A screenshot of a file explorer showing a list of files: storage, tests, vendor, .editorconfig, and .env. The .env file is highlighted with a blue selection bar. To the right of the file list, there is a code editor window displaying the contents of the .env file. The code is as follows:

```
43 PUSHER_APP_CLUSTER=mt1
44
45 MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
46 MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
47
48 SWEET_ALERT_AUTO_DISPLAY_ERROR_MESSAGES = true
```

A red arrow points to the line "SWEET_ALERT_AUTO_DISPLAY_ERROR_MESSAGES = true" in the code editor.

Gambar: Mengaktifkan fitur auto display error sweetalert

SweetAlert

Setelah itu silahkan restart cmd `php artisan serve` agar perubahan ini bisa efektif.

Sekarang setiap pesan error validasi otomatis tampil dalam jendela alert. Untuk uji coba, kita akan praktik dengan beberapa route berikut:

`routes\web.php`

```
1 ...  
2 Route::view('/swal-form', 'swal-form');  
3 Route::post('/swal-validate-satu', [SwalController::class, 'swalValidateSatu']);  
4 Route::post('/swal-validate-banyak', [SwalController::class, 'swalValidateBanyak']);
```

Route di baris 2 dipakai untuk menampilkan form, sedangkan route di baris 3 dan 4 berisi kode untuk validasi form. Untuk route validasi ini saya menggunakan `Route::post()`.

Pada saat route '/swal-form' diakses, itu akan mencari view `swal-form.blade.php`. Silahkan buat file blade ini dan isi dengan kode program berikut:

`resources\views\swal-form.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   <title>Belajar SweetAlert</title>  
7   <link rel="stylesheet" href="/css/bootstrap.min.css">  
8 </head>  
9 <body>  
10  <div class="container mt-5">  
11    <h1>Belajar SweetAlert</h1>  
12    <hr>  
13  
14    <form method="POST" action="/swal-validate-satu">  
15      @csrf  
16      <div class="form-group row">  
17        <label class="col-1 col-form-label">Nama</label>  
18        <input type="text" class="form-control col-3" name="nama">  
19        <button type="submit" class="btn btn-primary ml-3">Submit</button>  
20      </div>  
21    </form>  
22  
23  </div>  
24  @include('sweetalert::alert')  
25 </body>  
26 </html>
```

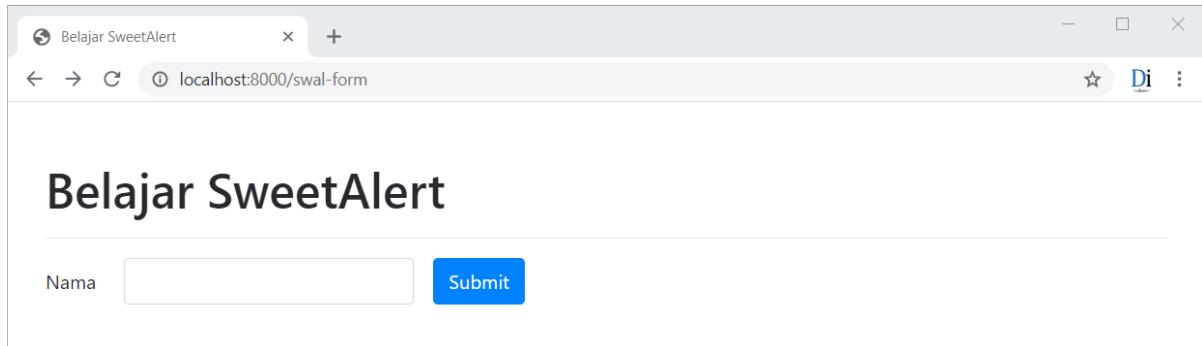
File blade ini berisi form antara baris 14 – 21. Di dalamnya saya membuat sebuah tag `<input>` dengan atribut `name="nama"` serta sebuah tombol Submit. Form ini akan dikirim ke route `/swal-validate-satu`, seperti nilai atribut `action` di baris 14.

Tidak lupa di akhir tag `<body>` terdapat perintah `@include('sweetalert::alert')` untuk

SweetAlert

menginput kode Realrashid/sweetalert.

View di atas bisa diakses dari URL `localhost:8000/swal-form`:



Gambar: Tampilan halaman `localhost:8000/swal-form`

Masuk ke controller, berikut method yang akan men-validasi inputan form tersebut:

`app\Http\Controllers\SwalController.php`

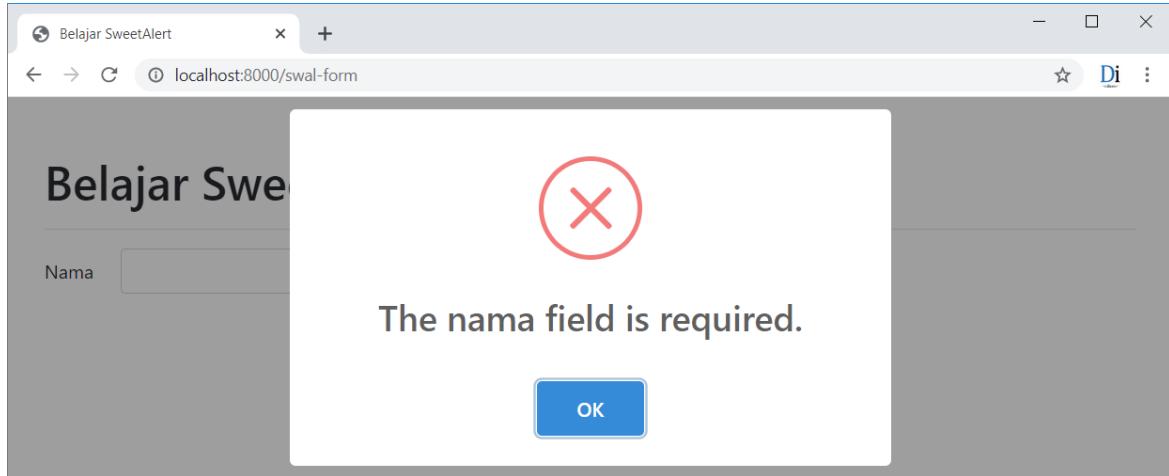
```
1  use Illuminate\Http\Request;
2  ...
3
4  public function swalValidateSatu(Request $request)
5  {
6      $request->validate([
7          'nama' => 'required|min:3|max:10',
8      ]);
9
10     echo "Lolos Validasi";
11 }
```

Dalam method `swalValidateSatu()` saya membuat 3 syarat validasi untuk inputan nama, yakni tidak boleh kosong, minimal berisi 3 karakter dan maksimal 10 karakter. Jika salah satu syarat tidak terpenuhi, Laravel akan men-redirect ke halaman sebelumnya (tempat form berada) plus disertai pesan error.

Sesampainya kembali di view, pesan error tersebut bisa ditampilkan dengan perintah blade seperti `@error('nama')`, Namun karena kita sudah men-set Realrashid/sweetalert, pesan error akan tampil dalam bentuk jendela sweetalert.

Untuk uji coba, silahkan buka `localhost:8000/swal-form`, dan langsung klik tombol Submit tanpa menginput form nama:

SweetAlert



Gambar: Pesan error tampil dalam bentuk jendela alert

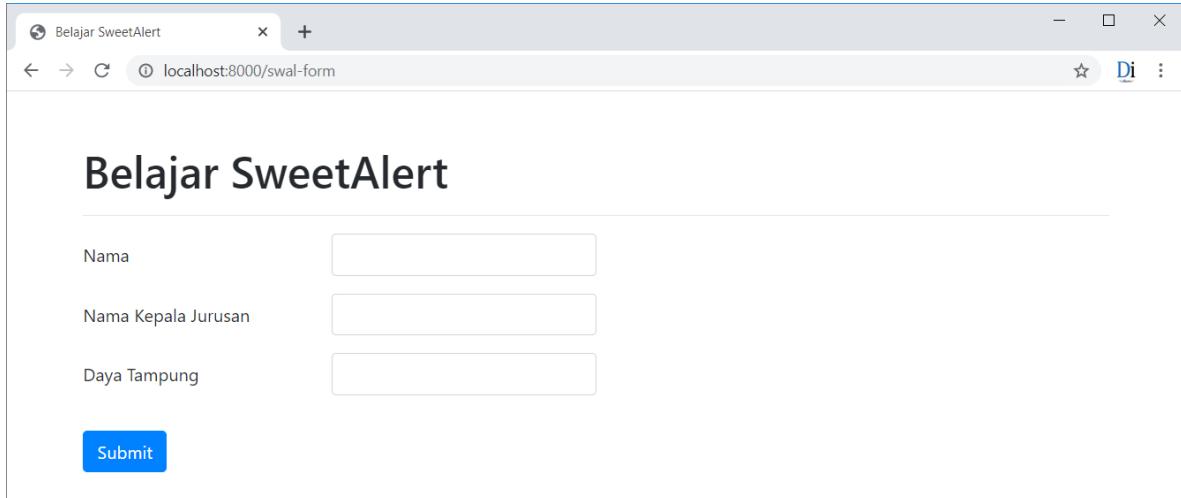
Hasilnya, pesan error tampil dalam jendela alert. Bisa juga lanjut test dengan mengisi kolom nama 1 huruf saja dan submit, maka akan tampil pesan error '*The nama must be at least 3 characters*'.

Pesan error ini juga berlaku untuk beberapa inputan. Sebagai contoh, silahkan modifikasi isi tag <form> di file swal-form.blade.php menjadi sebagai berikut:

resources\views\swal-form.blade.php

```
1 ...  
2 <form method="POST" action="/swal-validate-banyak">  
3     @csrf  
4     <div class="form-group row">  
5         <label class="col-3 col-form-label">Nama</label>  
6         <input type="text" class="form-control col-3" name="nama">  
7     </div>  
8  
9     <div class="form-group row">  
10        <label class="col-3 col-form-label">Nama Kepala Jurusan</label>  
11        <input type="text" class="form-control col-3" name="kepala_jurusan">  
12    </div>  
13  
14    <div class="form-group row">  
15        <label class="col-3 col-form-label">Daya Tampung</label>  
16        <input type="text" class="form-control col-3" name="daya_tampung">  
17    </div>  
18  
19    <button type="submit" class="btn btn-primary mt-3">Submit</button>  
20 </form>  
21 ...
```

SweetAlert



The screenshot shows a browser window titled "Belajar SweetAlert" at the URL "localhost:8000/swal-form". The page contains a form with three input fields and a submit button. The first field is labeled "Nama" and has a placeholder "Nama". The second field is labeled "Nama Kepala Jurusan" and has a placeholder "Nama Kepala Jurusan". The third field is labeled "Daya Tampung" and has a placeholder "Daya Tampung". Below the fields is a blue "Submit" button.

Gambar: Tampilan halaman localhost:8000/swal-form

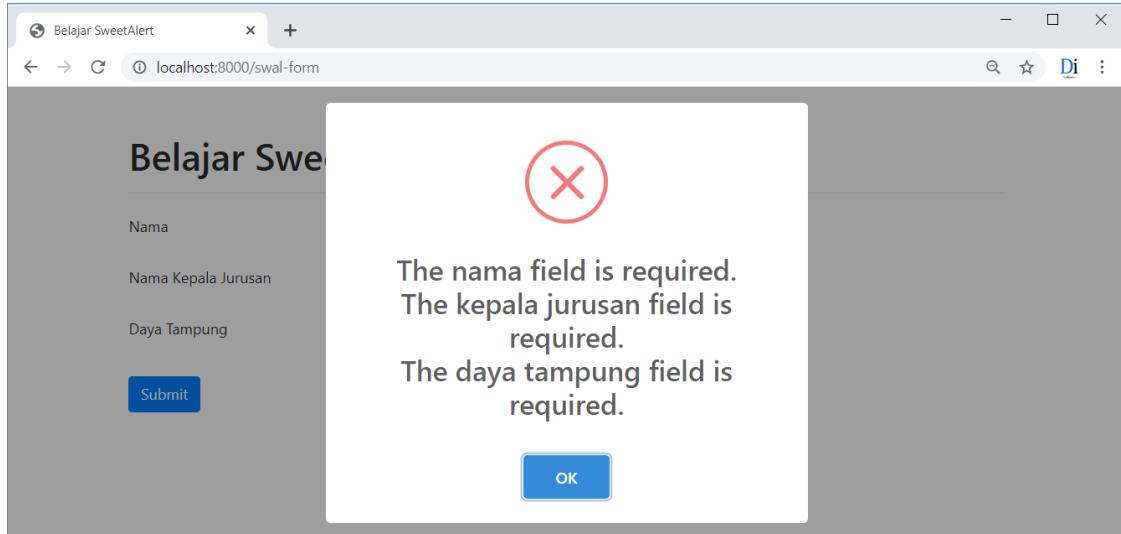
Sekarang saya menambah 2 inputan baru, yakni `kepala_jurusan` dan `daya_tampung`. Alamat pengiriman form juga berubah menjadi `/swal-validate-banyak`. Proses validasi untuk form ini di kerjakan oleh method `swalValidateBanyak()`:

app\Http\Controllers\SwalController.php

```
1 public function swalValidateBanyak(Request $request)
2 {
3     $request->validate([
4         'nama' => 'required|min:3|max:10',
5         'kepala_jurusan' => 'required',
6         'daya_tampung' => 'required|min:10|integer',
7     ]);
8
9     echo "Lolos Validasi";
10 }
```

Pada baris 4 – 6 saya membuat 3 buah proses validasi untuk setiap inputan form. Langsung saja kita uji, silahkan submit form tanpa mengisi inputan apapun:

SweetAlert



Gambar: Banyak pesan error tampil dalam jendela alert

Hasilnya, semua pesan error tampil di jendela alert.

Secara fungsi ini sudah berjalan, akan tetapi saya merasa tampilannya kurang bagus karena teks pesan error menjadi terlalu panjang, belum lagi jika kita punya 5 inputan form atau lebih.

Salah satu solusi adalah menampilkan 1 pesan error saja, yakni pesan yang muncul paling awal (tidak langsung semua sekaligus). Ini bisa dilakukan dengan mengubah method `swalValidateBanyak()` sebagai berikut:

app\Http\Controllers\SwalController.php

```
1 ...  
2 use Illuminate\Support\Facades\Validator;  
3 ...  
4  
5 public function swalValidateBanyak(Request $request)  
6 {  
7     $validator = Validator::make($request->all(), [  
8         'nama' => 'required|min:3|max:10',  
9         'kepala_jurusan' => 'required',  
10        'daya_tampung' => 'required|min:10|integer',  
11    ]);  
12  
13    if ($validator->fails()) {  
14        // Kirim hanya validasi pertama saja  
15        return back()->withErrors($validator->errors()->first())->withInput();  
16    }  
17    else {  
18        // Kode untuk proses ke database disini  
19        return redirect('swal')->withSuccess('Data jurusan berhasil diinput');  
20    }  
21 }
```

Untuk keperluan ini kita harus modifikasi proses validasi bawaan Laravel. Karena itulah perlu

proses import Validator class seperti di baris 2 agar bisa mengakses **Validator** facade.

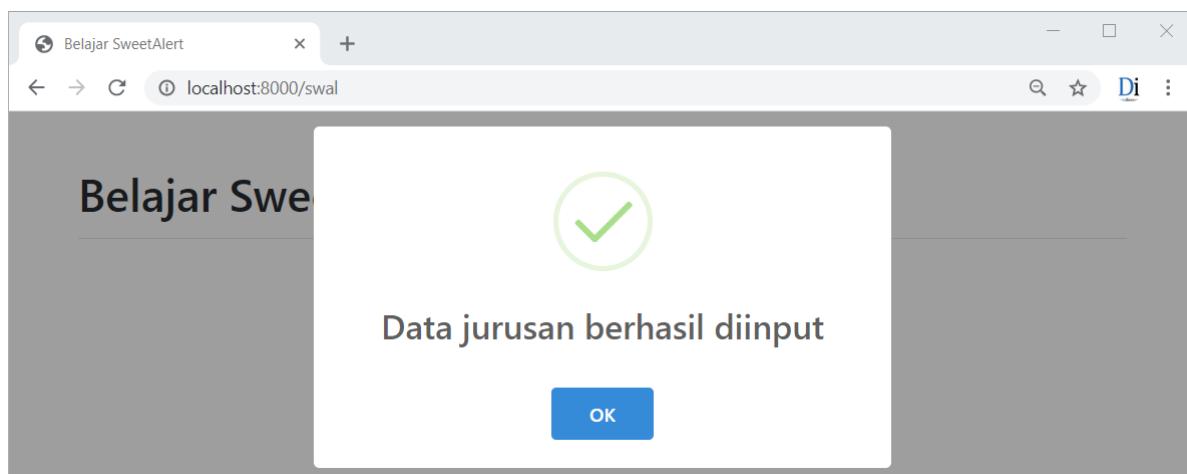
Syarat validasi masih sama seperti sebelumnya, akan tetapi sekarang dijalankan dari method **Validator::make()** yang kemudian ditampung ke variabel `$validator` di baris 7.

Kemudian terdapat kondisi `if ($validator->fails())` di baris 13. Kondisi ini akan bernilai **true** jika terdapat inputan yang tidak lolos validasi. Jika ini terjadi, *redirect* user ke halaman sebelumnya menggunakan perintah `return back()`.

Selain itu kirim juga pesan error dengan men-chaining method `withErrors()`. Argument `$validator->errors()->first()` dipakai untuk mengambil satu pesan error pertama saja. Terakhir, di-chaining lagi dengan method `withInput()` untuk proses repopulate inputan form.

Blok `else` di baris 17 – 20 akan dijalankan jika semua inputan form lolos validasi. Jika ini yang terjadi, *redirect* user ke route 'swal' beserta pesan alert 'Data jurusan berhasil diinput'. Pada prakteknya nanti, di dalam blok inilah kita menulis kode program untuk proses input data jurusan ke dalam database.

Dengan perubahan ini, hanya ada 1 pesan error yang tampil di jendela alert, tidak semuanya sekaligus. Selain itu test juga dengan mengisi semua inputan, yang jika lolos validasi akan di *redirect* ke halaman `localhost:8000/swal` beserta pesan sukses.



Gambar: Pesan sukses jika syarat validasi terpenuhi

22.12. Realrashid/sweet-alert Delete Confirmation

Materi terakhir yang akan kita bahas adalah cara menampilkan jendela konfirmasi menggunakan Realrashid/sweet-alert. Jendela konfirmasi ini sering dipakai pada saat menghapus data, yang harus selalu dibuat untuk menghindari tombol delete yang ter-tekan secara tidak sengaja.

Namun sayangnya hingga saat buku ini ditulis, fitur tersebut belum tersedia secara bawaan di library Realrashid/sweet-alert. Pada kolom diskusi github sudah ada permintaan tapi

sepertinya perlu perubahan besar ke library (github.com/realrashid/sweetalert/issues/57).

Akan tetapi, bukan berarti kita tidak bisa membuat jendela konfirmasi. Hanya saja perlu balik ke asal menggunakan kode JavaScript.

Untuk membuat jendela konfirmasi, kita harus set agar file source code SweetAlert2 selalu aktif untuk setiap halaman. Bawaan dari Realrashid/sweetalert, file source code ini hanya di load saat ada pemanggilan dari route sebelumnya.

Sebagai contoh, silahkan akses langsung URL `localhost:8000/swal`, lalu lihat source code halaman tersebut di web browser:



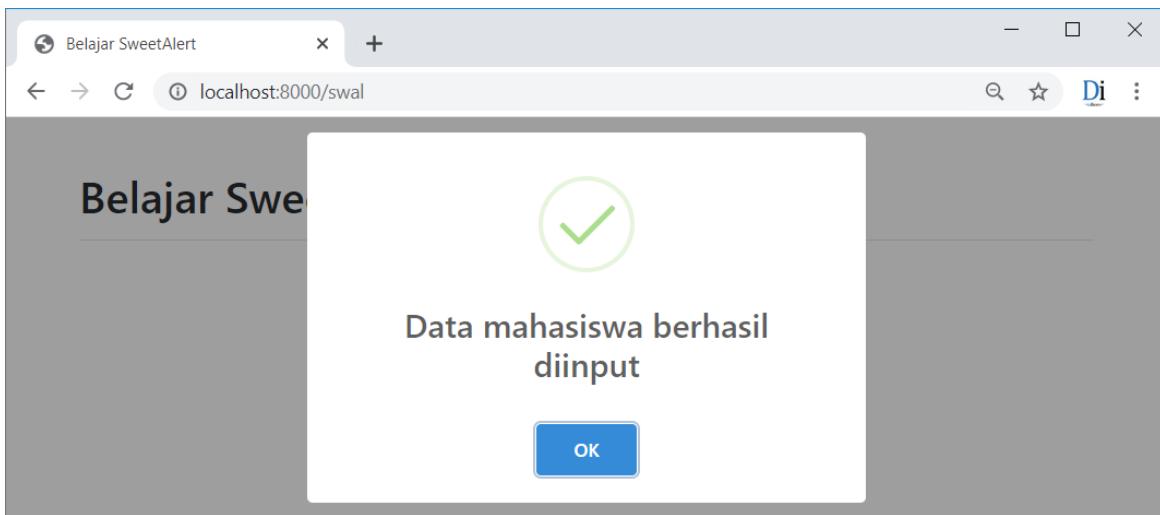
The screenshot shows a browser window with two tabs. The active tab is titled "view-source:localhost:8000/swal" and displays the HTML source code of a SweetAlert page. The code includes a title, a message, and an "OK" button. The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with icons for refresh, stop, and others.

```
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   <title>Belajar SweetAlert</title>
7   <link rel="stylesheet" href="/css/bootstrap.min.css">
8 </head>
9 <body>
10  <div class="container mt-5">
11    <h1>Belajar SweetAlert</h1>
12    <hr>
13  </div>
14 </body>
15 </html>
16
```

Gambar: Source code halaman `localhost:8000/swal`

Perhatikan bahwa tidak ada tag `<script>` yang dipakai untuk mengakses file SweetAlert2. Padahal di halaman ini ada perintah `@include('sweetalert::alert')` yang ditempatkan sebelum tag `</body>`.

Sekarang kita coba akses URL `localhost:8000/swal-with` di web browser. Halaman akan loading sesaat dan langsung redirect ke `localhost:8000/swal`. Ketika selesai di redirect, akan tampil jendela alert success yang merupakan praktek dari materi kita sebelumnya:



Gambar: Tampilan jendela alert success

SweetAlert

Tutup jendela alert dan cek kembali source code halaman `localhost:8000/swal` :



```
6 <title>Belajar SweetAlert</title>
7 <link rel="stylesheet" href="/css/bootstrap.min.css">
8 </head>
9 <body>
10 <div class="container mt-5">
11   <h1>Belajar SweetAlert</h1>
12   <hr>
13 </div>
14 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/animate.css">
15   <script src="http://localhost:8000/vendor/sweetalert/sweetalert.all.js"></script>
16   <script>
17     Swal.fire({
18       title: "Data mahasiswa berhasil
diinput",
19       text: "",
20       timer: false,
21       width: "32rem",
22       heightAuto: true,
23       padding: "1.25rem",
24       showConfirmButton: true,
25       showCloseButton: false,
26       customClass: {
27         container: null,
28         popup: null,
29         header: null,
30         title: null,
31         closeButton: null,
32         icon: null,
33         image: null,
34         content: null,
35         input: null,
36         actions: null,
37         confirmButton: null,
38         cancelButton: null,
39         footer: null,
40         icon: "success"
41       }
42     })
43   </script>
44 </body>
45 </html>
```

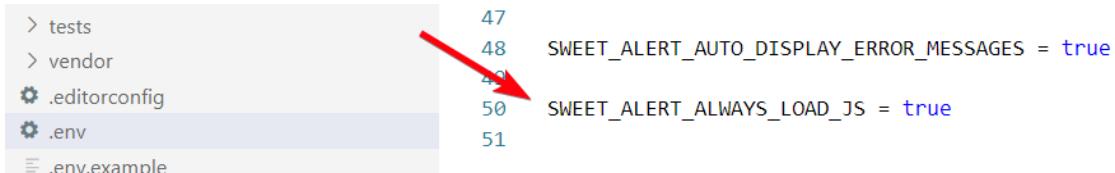
Gambar: Source code halaman `localhost:8000/swal` dengan sweetalert

Sekarang terdapat tambahan kode di bagian bawah, yakni tag `<link>` untuk mengakses file `animate.css` di baris 14 (via CDN), serta tag `<script>` untuk mengakses file `sweetalert.all.js` di baris 15. Selain itu di bagian bawah juga muncul beberapa baris kode JavaScript yang ditambahkan oleh Realrashid/sweetalert.

Inilah cara kerja dari perintah `@include('sweetalert::alert')`, yakni hanya mengakses file source code sweetalert pada saat dibutuhkan saja.

Namun karena kita akan membuat jendela konfirmasi menggunakan JavaScript, file `sweetalert.all.js` ini harus selalu tersedia. Caranya, tambah baris berikut ke file `.env`:

```
SWEET_ALERT_ALWAYS_LOAD_JS = true
```



Gambar: Menambah pengaturan di file `.env`

Pengaturan ini akan membuat perintah `@include('sweetalert::alert')` untuk selalu menampilkan link ke file `sweetalert.all.js`.

Agar efektif, silahkan restart server dengan cara menutup dan menjalankan ulang perintah `php artisan serve`. Setelah itu akses kembali URL `localhost:8000/swal` dan cek apakah link ke file sweetalert sudah langsung tampil:

SweetAlert



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Belajar SweetAlert</title>
  <link rel="stylesheet" href="/css/bootstrap.min.css">
</head>
<body>
  <div class="container mt-5">
    <h1>Belajar SweetAlert</h1>
    <hr>
  </div>
<script src="http://localhost:8000/vendor/sweetalert/sweetalert.all.js"></script>
</body>
</html>
```

Gambar: Link ke file sweetalert.all.js sudah aktif

Jika sudah tampil, maka kita bisa masuk ke praktik membuat jendela konfirmasi.

Dalam Laravel, proses delete setidaknya butuh 2 buah route. Route pertama untuk menampilkan tombol delete, dan route kedua untuk memproses operasi penghapusan:

routes\web.php

```
1 Route::get('/swal-delete', [SwalController::class, 'delete']);
2 Route::delete('/swal-delete/{id}', [SwalController::class, 'destroy']);
```

Kita berangkat dari route pertama. Pada saat route '/swal-delete' diakses, itu akan menjalankan method `delete()` milik `SwalController.php`. Berikut kode program untuk method tersebut:

app\Http\Controllers\SwalController.php

```
1 public function delete()
2 {
3     return view('swal-delete')->with(['id' => 7, 'nama' => 'Novi Lestari']);
4 }
```

Dalam kode ini, saya langsung mengakses view 'swal-delete' beserta 2 buah data: `id = 7` dan `nama = 'Novi Lestari'`. Sesampainya di file view nanti, data ini bisa diakses dari variabel `$id` dan `$nama`. Ini sebagai simulasi dari data yang akan dihapus.

Lanjut ke view, silahkan buat file `swal-delete.blade.php` dan isi dengan kode berikut:

resources\views\swal-delete.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   <title>Belajar SweetAlert</title>
7   <link rel="stylesheet" href="/css/bootstrap.min.css">
```

SweetAlert

```
8  </head>
9  <body>
10 <div class="container mt-5">
11   <h1>Belajar SweetAlert</h1>
12   <hr>
13   <form action="{{url('/swal-delete/'.$id)}}" method="POST">
14     @csrf @method('DELETE')
15     <button type="submit" class="btn btn-danger ml-3 btn-hapus"
16       data-name="{{$nama}}>Hapus Data</button>
17   </form>
18 </div>
19 @include('sweetalert::alert')
20 <script>
21 let tombol = document.getElementsByClassName('btn-hapus')[0];
22 tombol.addEventListener('click',konfirmasi);
23
24 function konfirmasi(event){
25   event.preventDefault();
26   Swal.fire({
27     title: 'Apakah anda yakin?',
28     text: 'Hapus data mahasiswa '+ event.target.getAttribute('data-name'),
29     icon: 'warning',
30     showCancelButton: true,
31     cancelButtonColor: '#6c757d',
32     confirmButtonColor: '#dc3545',
33     confirmButtonText: 'Ya, hapus!',
34     reverseButtons: true,
35   }).then((result) => {
36     if (result.value) {
37       event.target.parentElement.submit();
38     }
39   })
40 }
41 </script>
42 </body>
43 </html>
```

Kita langsung masuk ke bagian konten yang sebenarnya hanya berisi 1 tombol saja, yakni tombol "Hapus Data" di baris 15 – 16. Tombol ini berada di dalam tag `<form>` yang ketika di submit akan menuju alamat `'/swal-delete/'.$id`, sesuai dengan nilai atribut `action`.

Alamat penghapusan ini di definisikan pada route kedua yang sudah kita tulis sebelumnya:

```
Route::delete('/swal-delete/{id}', [SwalController::class, 'destroy']);
```

Teknik menghapus data seperti ini merupakan penerapan dari konsep **RESTfull**, yang penjelasan lengkapnya tersedia di bab CRUD buku **Laravel Uncover**.

Normalnya, begitu tombol "Hapus Data" di klik, form akan langsung ter-submit. Akan tetapi alur ini akan saya interupsi untuk menampilkan jendela konfirmasi. Inilah fungsi dari kode JavaScript antara baris 21 – 40.

SweetAlert

Di baris 21, saya mengisi variabel `tombol` dengan `node element` dari tombol "Hapus Data" yang ada di dalam form. Ini dilakukan dengan perintah `document.getElementsByClassName('btn-hapus')[0]`.

Jika diperhatikan dengan seksama, tag `<button>` di dalam form memiliki tambahan class "btn-hapus". Inilah yang bisa diakses dari method `getElementsByClassName()`. Tambahan `[0]` di akhir method berguna untuk mengakses element pertama, karena hasil method `getElementsByClassName()` akan berbentuk array.

Perintah `tombol.addEventListener('click', konfirmasi)` di baris 22 berfungsi untuk menambah event `click` ke tombol "Hapus Data". Dengan perintah ini, sewaktu tombol "Hapus Data" di klik, JavaScript akan menjalankan isi function `konfirmasi()`. Kode program untuk function `konfirmasi()` berada antara baris 24 – 39.

Di baris 25, terdapat perintah `event.preventDefault()` yang dipakai untuk menonaktifkan perilaku default dari tombol submit. Perintah ini sangat penting karena inilah yang menahan agar form tidak terkirim begitu tombol "Hapus data" di klik.

Selanjutnya terdapat perintah `Swal.fire()` yang akan menampilkan jendela konfirmasi milik SweetAlert2. Kode ini sudah kita bahas di awal bab, tapi yang menarik ada di property `text`:

```
text: 'Hapus data mahasiswa ' + event.target.getAttribute('data-name')
```

Di sini saya ingin agar teks jendela konfirmasi menampilkan nama mahasiswa yang ingin dihapus. Dari mana nama mahasiswa ini didapat? Itu bisa didapat dari atribut 'data-name' yang ada di dalam tombol "Hapus Data".

Bisa diperhatikan kembali bahwa di dalam tag `<tombol>`, saya menempatkan atribut `data-name="{$nama}"` pada baris 16. Variabel `$nama` berasal dari controller, yang dalam contoh kita akan di proses menjadi `data-name="Novi Lestari"`.

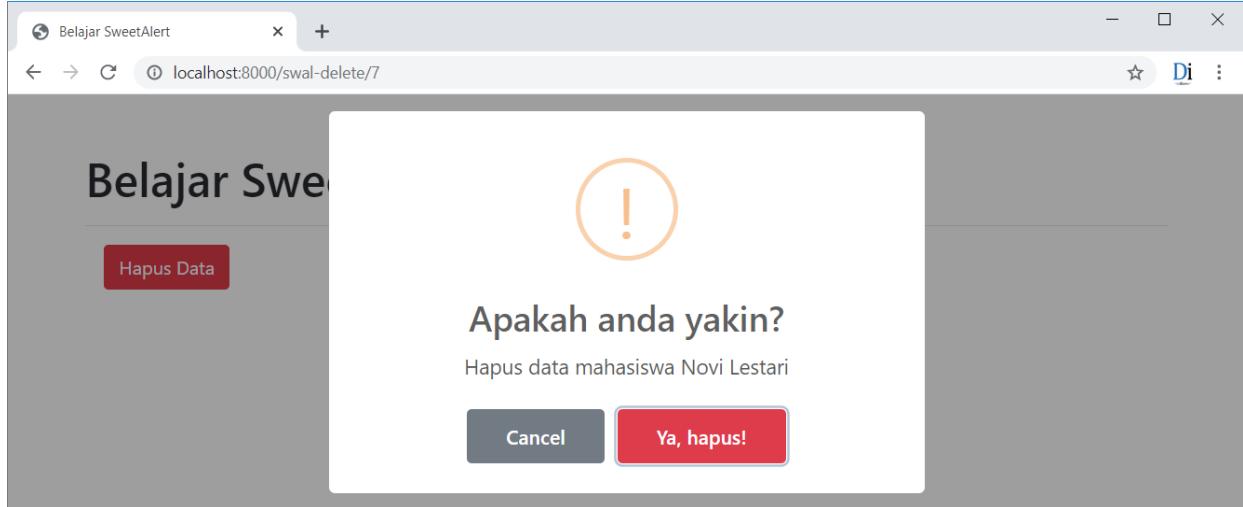
Selain property `text`, saya juga menambah beberapa property sweetalert lain, diantaranya untuk menampilkan tombol cancel melalui property `showCancelButton`, serta mengubah warna tombol dengan property `cancelButtonColor` dan `confirmButtonColor`.

Property `reverseButtons: true` berfungsi untuk mengubah posisi tombol cancel agar pindah ke sisi kiri (secara bawaan tombol ini berada di kanan).

Jika user men-klik tombol cancel, jendela alert akan tertutup dan tidak terjadi proses penghapusan data. Namun jika tombol "**Ya, hapus!**" di klik, perintah di baris 37 akan dijalankan, yakni kode JavaScript untuk men-submit form.

Berikut tampilan jendela konfirmasi ini:

SweetAlert



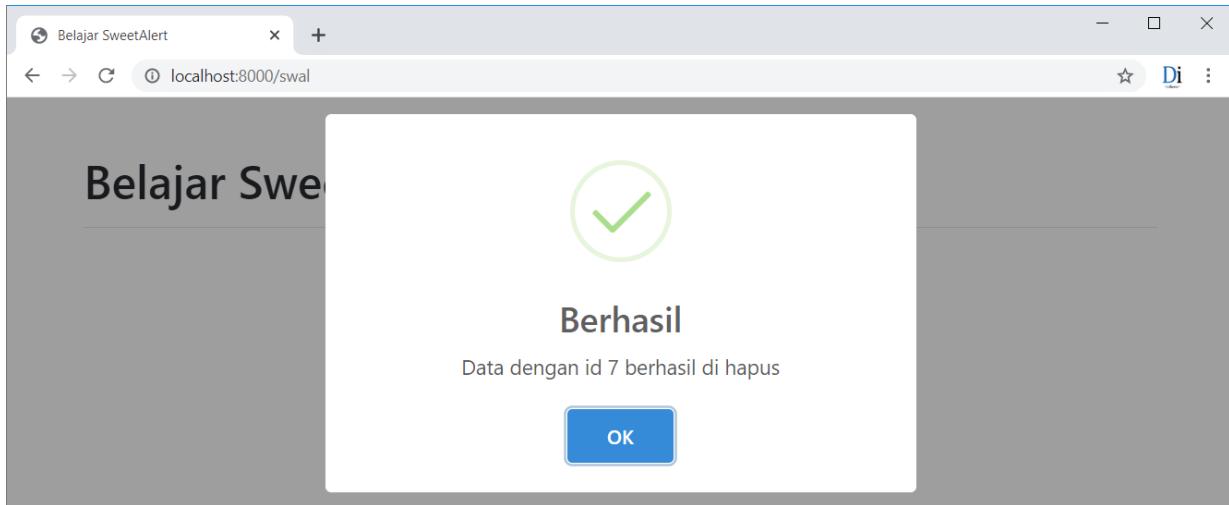
Gambar: Tampilan jendela konfirmasi

Namun tugas kita masih belum selesai. Ketika tombol "Ya, hapus!" di klik, form akan ter-submit ke route '/swal-delete'. Route ini akan mengakses method `destroy()` di `SwalController.php`. Berikut kode program untuk method tersebut:

app\Http\Controllers\SwalController.php

```
1  public function destroy($id)
2  {
3      Alert::alert('Berhasil', "Data dengan id $id berhasil di hapus", 'success');
4      return redirect('swal');
5 }
```

Saat ini kode yang ada hanya membuat jendela alert success, lalu me-redirect user ke route 'swal'. Dalam praktek sebenarnya, kita perlu menjalankan perintah eloquent untuk menghapus data dari dalam database.



Gambar: Proses delete berhasil

Pembuatan jendela konfirmasi ini memang sedikit rumit, apalagi jika kurang memahami kode

SweetAlert

JavaScript. Bagi yang ingin fokus ke back-end, saya tetap sarankan untuk belajar JavaScript dasar, karena dalam satu atau dua kasus, kita tetap butuh menulis kode JavaScript seperti pada contoh ini. Nantinya jendela konfirmasi yang sama akan saya terapkan ke project Sistem Informasi Universitas ILKOOM.

Dalam bab ini kita sudah membahas cukup detail tentang cara penggunaan SweetAlert2. Materi ini sebenarnya bukan bagian dari Laravel, akan tetapi menjadi nilai tambah tersendiri. Apa yang kita pelajari di sini juga akan saya terapkan saat kita masuk ke bagian kedua dari proses CRUD "Sistem Informasi Universitas ILKOOM", yakni bagian **Create** yang akan dibahas pada bab selanjutnya.

23. Sistem Informasi Universitas ILKOOM: Create

Kita kembali ke project Universitas ILKOOM. Bab ini akan berfokus ke proses **Create**, yakni membuat form untuk penambahan data, memproses form tersebut (termasuk validasi), serta menampilkan pesan error jika inputan tidak lolos validasi.

23.1. Menginstall Realrashid/sweetalert

Library sweetalert yang kita bahas dalam bab sebelumnya akan langsung saya terapkan ke project Universitas ILKOOM. Silahkan install dengan kode berikut secara berurutan:

```
composer require realrashid/sweetalert
```

```
php artisan sweetalert:publish
```

Kemudian buka file view resources\views\layouts\app.blade.php, lalu tambah perintah @include('sweetalert::alert') pada posisi setelah tag </footer> dan sebelum tag </body>:



```

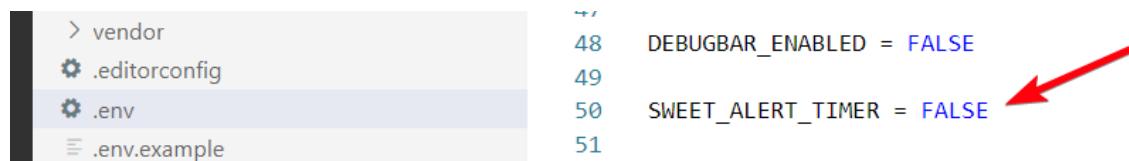
    index.blade.php
    layouts
      app.blade.php
        ↴
        | 174   </div>
        | 175   </footer>
        | 176   @include('sweetalert::alert') ← Red arrow here
        | 177   </body>
        | 178   </html>
        | 179
    mahasiswa
    index.blade.php
  
```

Gambar: Tambah @include('sweetalert::alert') ke file layouts\app.blade.php

Dengan perintah ini, semua halaman view sudah bisa mengakses file sweetalert2.

Selain itu saya juga ingin mengubah pengaturan fitur auto close. Bawaan dari Realrashid/sweetalert, jendela alert otomatis tertutup setelah 5 detik. Untuk menonaktifkannya, tambah satu baris berikut ke file .env:

```
SWEET_ALERT_TIMER = FALSE
```



```

    vendor
    .editorconfig
    .env
    .env.example
  
```

<pre>48 DEBUGBAR_ENABLED = FALSE 49 50 SWEET_ALERT_TIMER = FALSE ← Red arrow here 51</pre>
--

Gambar: Set timer sweetalert menjadi False

Konfigurasi Realrashid/sweetalert bisa diatur dari 2 tempat, yakni dari file .env atau dari file config\sweetalert.php. Penjelasan lengkap dari setiap konfigurasi ada di file config\sweetalert.php, namun lebih praktis jika aturan tersebut di set dari file .env.

23.2. Menambah Data Jurusan

Kita masuk ke proses penambahan data Jurusan. Ini dimulai dengan membuat tombol "Tambah Jurusan" yang ketika diklik akan menampilkan halaman form. Tombol ini saya tempatkan ke halaman index.

Silahkan buka file jurusan\index.blade.php, lalu tambah kode berikut di bagian atas:

resources\views\jurusan\index.blade.php

```
1 @extends('layouts.app')
2
3 @section('content')
4 <h1 class="display-4 text-center my-5">Sistem Informasi Universitas ILKOOOM</h1>
5
6 <div class="text-right pt-5">
7   @auth
8     <a href="{{ route('jurusans.create') }}" class="btn btn-info">
9       Tambah Jurusan</a>
10    @endauth
11 </div>
12
13 <div class="card-columns mt-3">
14   @foreach($jurusans as $jurusan)
15     ...
```

Kode yang perlu ditambah ada di baris 6 – 11, dimana saya membuat sebuah tag <div> yang di dalamnya terdapat link menuju route 'jurusans.create'. Link ini akan tampil sebagai tombol karena menggunakan class Bootstrap .btn dan .btn-info.

Tombol ini juga berada dalam blok @auth dan @endauth, yang artinya hanya tampil untuk user yang sudah login saja.

Silahkan buat user baru dengan cara men-klik link register di menu footer, atau bisa juga dengan mengakses alamat localhost:8000/register.

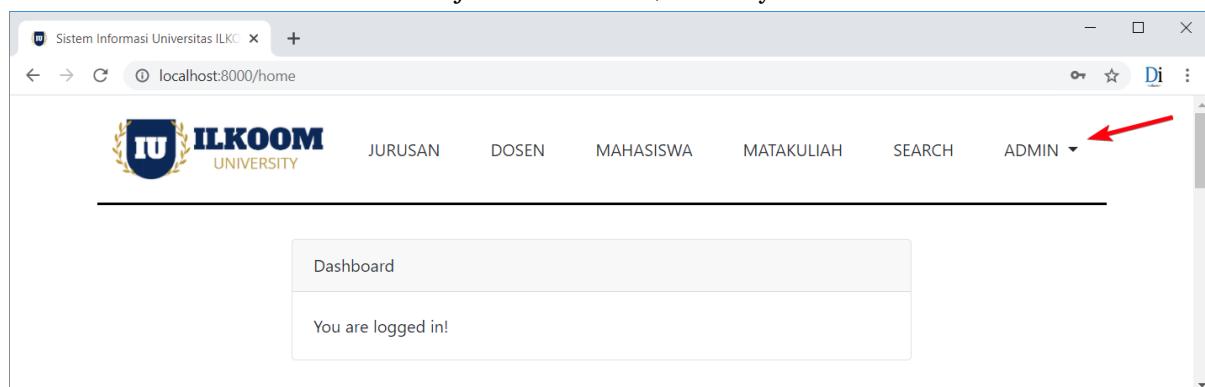
Dalam contoh ini saya membuat user bernama "Admin" (sebenarnya bisa menggunakan nama apa saja).

Sistem Informasi Universitas ILKOOOM: Create

The screenshot shows a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/register". The page content is a registration form titled "Register". It includes fields for "Name" (containing "Admin"), "E-Mail Address" (containing "admin@gmail.com"), "Password", and "Confirm Password". A "Register" button is at the bottom of the form.

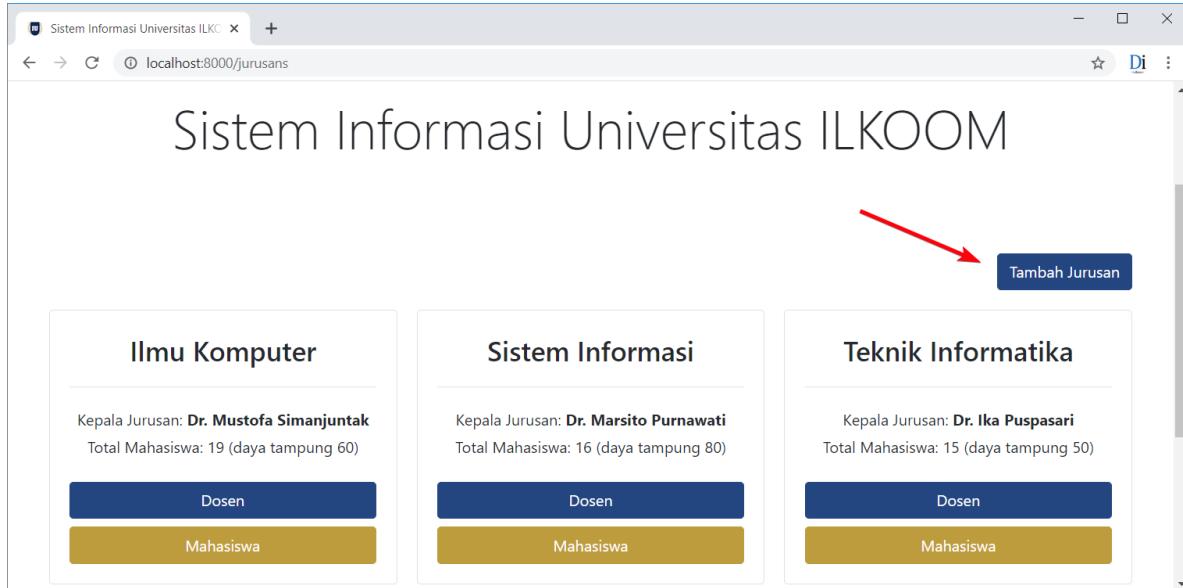
Gambar: Membuat user baru di alamat <http://localhost:8000/register>

Setelah register, halaman akan redirect ke `localhost:8000/home`. Kemudian menu login di sudut kanan atas akan berubah menjadi nama user, misalnya user "Admin" :



Gambar: Menu login sudah berubah menjadi "Admin"

Lanjut, silahkan klik menu Jurusan atau buka alamat `localhost:8000/jurusans`. Di sisi tengah halaman akan tampil tombol "**Tambah Jurusan**":



Gambar: Tombol "Tambah Jurusan"

Sekarang kita masuk kebagian utama, yakni membuat form input data Jurusan.

Dari file `index.blade.php` sebelumnya, tombol "Tambah Jurusan" terhubung ke *named route* '`jurusans.create`'. Ini merupakan sebuah *route resources* yang akan di proses oleh method `create()` di file `JurusanController.php`. Berikut kode yang dibutuhkan untuk method ini:

`app\Http\Controllers\JurusanController.php`

```
1 public function create()
2 {
3     return view('jurusan.create');
4 }
```

Isi dari method `create()` tidak lain hanya memanggil view '`jurusans.create`'. Silahkan buat file view `create.blade.php` di folder `resources\views\jurusans`, lalu isi dengan kode berikut:

`resources\views\jurusans\create.blade.php`

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5     <h1 class="h2">Tambah Jurusan</h1>
6 </div>
7 <hr>
8
9 <form method="POST" action="{{ route('jurusans.store') }}">
10    @include('jurusans.form',['tombol' => 'Tambah'])
11 </form>
12
13 @endsection
```

Sama seperti view lain, di baris 1 saya meng-extends file master `layout\app.blade.php`,

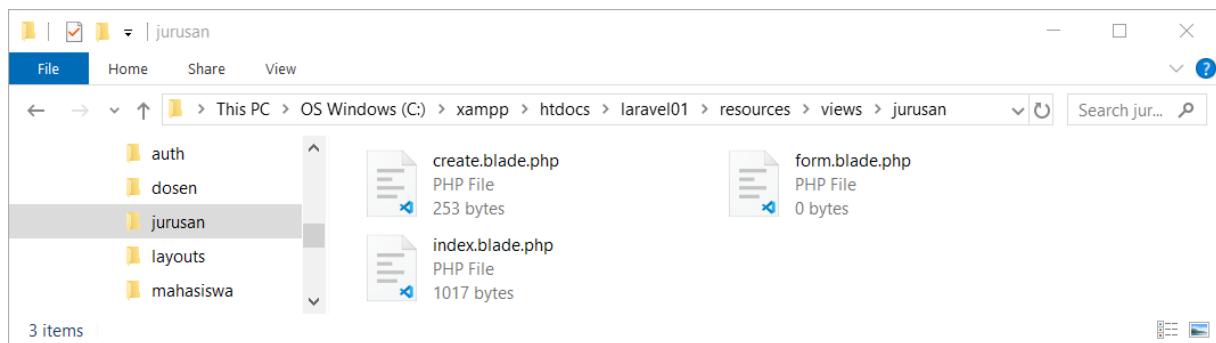
kemudian di ikuti dengan judul "Tambah Jurusan" di baris 4 – 7.

Form input ada di baris 9, yakni sebuah tag <form> yang ketika diklik akan mengirim data ke named route 'jurusans.store'. Di route inilah nantinya data form akan di proses.

Isi form hanya terdiri dari 1 perintah @include('jurusan.form', ...) di baris 10. Artinya, saya menempatkan struktur form ke view terpisah, yakni file form.blade.php.

Ini dilakukan karena nantinya kita juga perlu membuat form update. Daripada menulis ulang struktur form yang sama, lebih baik dipisah untuk kemudian di-include pada kedua form.

Silakan buat file form.blade.php ini di folder resources\views\jurusan, sehingga di folder jurusan sudah terdapat 3 buah file view:



Gambar: Isi folder resources\views\jurusan

Berikut kode program untuk file form.blade.php:

resources\views\jurusan\form.blade.php

```

1 @csrf
2 <div class="form-group row">
3   <label for="nama" class="col-md-3 col-form-label text-md-right">
4     Nama Jurusan </label>
5   <div class="col-md-6">
6     <input id="nama" type="text"
7     class="form-control col-md-8 @error('nama') is-invalid @enderror"
8     name="nama" value="{{ old('nama') ?? $jurusan->nama ?? '' }}"
9     autofocus>
10    @error('nama')
11      <span class="invalid-feedback" role="alert">
12        <strong>{{ $message }}</strong>
13      </span>
14    @enderror
15  </div>
16 </div>
17
18 <div class="form-group row">
19   <label for="kepala_jurusan" class="col-md-3 col-form-label text-md-right">
20     Nama Kepala Jurusan </label>
21   <div class="col-md-6">
22     <input id="kepala_jurusan" type="text"
23     class="form-control col-md-8 @error('kepala_jurusan') is-invalid @enderror"

```

```

24    name="kepala_jurusan" autofocus
25    value="{{ old('kepala_jurusan') ?? $jurusan->kepala_jurusan ?? '' }}">>
26    @error('kepala_jurusan')
27        <span class="invalid-feedback" role="alert">
28            <strong>{{ $message }}</strong>
29        </span>
30    @enderror
31  </div>
32 </div>
33
34 <div class="form-group row">
35     <label for="daya_tampung" class="col-md-3 col-form-label text-md-right">
36         Daya Tampung Jurusan </label>
37     <div class="col-md-6">
38         <input id="daya_tampung" type="text"
39             class="form-control col-md-2 @error('daya_tampung') is-invalid @enderror"
40             name="daya_tampung" autofocus
41             value="{{ old('daya_tampung') ?? $jurusan->daya_tampung ?? '' }}">>
42         @error('daya_tampung')
43             <span class="invalid-feedback" role="alert">
44                 <strong>{{ $message }}</strong>
45             </span>
46         @enderror
47     </div>
48 </div>
49
50 <div class="form-group row mt-5">
51     <div class="col-md-6 offset-md-3">
52         <button type="submit" class="btn btn-primary">{{$tombol}}</button>
53     </div>
54 </div>

```

Gambar: Tampilan form tambah jurusan

Dalam view ini saya membuat 3 buah inputan text dan 1 tombol submit. Jika anda sudah mempelajari buku **Laravel Uncover**, sebenarnya struktur form ini sangat mirip seperti studi kasus ILKOOM Profile Manager yang ada di bab terakhir.

Misalnya untuk inputan nama jurusan, terdapat kode blade `@error('nama') is-invalid @enderror` dalam atribut `class` di baris 7. Kode ini dipakai untuk menambah class Bootstrap `.is-invalid` jika terjadi error. Class `.is-invalid` akan membuat efek border merah di inputan form.

Kemudian nilai atribut `value` saya isi dengan "`{{ old('nama') ?? $jurusan->nama ?? '' }}`". Kode ini akan mengisi nilai awal inputan form sesuai urutan prioritas berikut:

1. Nilai `old('nama')` akan tampil jika form tidak lolos validasi. Ini dipakai untuk proses *repopulate* inputan form.
2. Nilai `$jurusan->nama` akan tampil jika form dipakai untuk proses update. Pada saat update, variabel `$jurusan` akan dikirim dari controller.
3. Nilai kosong '' akan tampil jika form diakses untuk proses create, yakni saat tombol "Tambah Jurusan" di klik.

Pesan error untuk setiap inputan form akan tampil dalam blok `@error('nama')` seperti di baris 10 – 14.

Struktur kode yang sama saya pakai untuk inputan kepala jurusan dan daya tampung, sehingga tidak perlu kita bahas lagi.

Di baris 52, teks tombol submit saya buat dinamis dengan perintah `{{ $tombol }}`. Nilai variabel `$tombol` berasal dari view induk, yang dalam contoh ini adalah file `create.blade.php`.

Jika anda lihat kembali cara pemanggilan file `form.blade.php` dari view `create.blade.php`, saya tulisnya dengan perintah berikut:

```
@include('jurusan.form', ['tombol' => 'Tambah'])
```

Selain proses include view `jurusany.form`, argument `['tombol' => 'Tambah']` akan mengisi nilai variabel `$tombol`. Dengan teknik ini, teks untuk tombol submit bisa diubah-ubah.

Misalnya menjadi 'Update' saat view `form.blade.php` di include dari view update nantinya.

Untuk uji coba, silahkan klik tombol "Tambah Jurusan". Jika tidak ada masalah, akan tampil form dengan 3 inputan + 1 tombol "Tambah".

Sekarang kita lanjut ke pemrosesan form. Pada saat form di submit, data inputan akan dikirim ke *named route* `'jurusans.store'`, yakni sesuai dengan nilai atribut `action` dari tag `<form>`. Route ini terhubung ke method `store()` di dalam `JurusanController.php`.

Berikut kode yang dibutuhkan:

app\Http\Controllers\JurusanController.php

```

1 <?php
2 ...
3 use RealRashid\SweetAlert\Facades\Alert;
4 ...
5
6     public function store(Request $request)
7     {
8         $validateData = $request->validate([
9             'nama' => 'required',
10            'kepala_jurusan' => 'required',
11            'daya_tampung' => 'required|min:10|integer',
12        ]);
13        $jurusan = Jurusan::create($validateData);
14        Alert::success('Berhasil', "Jurusan $request->nama berhasil dibuat");
15        return redirect("/jurusans#card-{$jurusan->id}");
16    }

```

Di bagian paling atas (baris 3), saya mengimport class Alert milik RealRashid\SweetAlert. Ini diperlukan karena kita akan mengirim pesan sukses menggunakan sweetalert.

Syarat validasi ada di baris 8 – 12, dimana ketiga inputan tidak boleh kosong. Untuk inputan daya tampung, juga harus bertipe integer dan nilainya tidak boleh kurang dari 10. Apabila salah satu syarat validasi tidak terpenuhi, otomatis akan di-redirect ke halaman form sebelumnya.

Jika lolos validasi, kode di baris 13 – 15 akan dijalankan. Ini dimulai dengan perintah eloquent Jurusan::create(\$validateData) untuk menginput data hasil validasi ke tabel jurusans.

Setelah itu saya membuat pesan sweetalert di baris 14. Sebagai isi teks, diinput juga nama jurusan dengan mengakses nilai \$request->nama.

Terakhir, user di redirect ke halaman "/jurusans#card-{\$jurusan->id}" di baris 15. Halaman '/jurusans' adalah halaman index yang menampilkan semua data jurusan. Di sini terdapat tambahan *hash fragment* dalam bentuk card-{\$jurusan->id}. Fungsinya adalah untuk membuat efek visual dari jurusan yang baru saja ditambah.

Sebagai contoh, jika jurusan yang saat ini diproses memiliki id = 4, maka pada saat redirect akan menuju halaman '/jurusans#card-4'. Dalam CSS, terdapat *pseudo selector* :target untuk mencari element yang sedang diakses oleh *hash fragment*.

Agar bisa bekerja, silahkan buka view index.jurusan.blade, lalu tambah sedikit kode di bagian awal card (yakni tag <div> pertama setelah perintah foreach):

resources\views\jurusan\index.blade.php

```

1 ...
2 <div class="card-columns mt-3">
3     @foreach($jurusans as $jurusan)

```

```
4
5     <div class="card mt-1" id=card-{{ $jurusan->id }}>
6         <div class="card-body text-center">
7             ...
```

Dengan tambahan `id=card-{{ $jurusan->id }}` seperti di baris 5, setiap "kotak" atau card jurusan memiliki atribut `id` sesuai dengan `$jurusan->id`. Jika jurusan saat ini memiliki `id = 4`, maka kode di baris 5 akan di proses sebagai:

```
<div class="card mt-1" id=card-4>
```

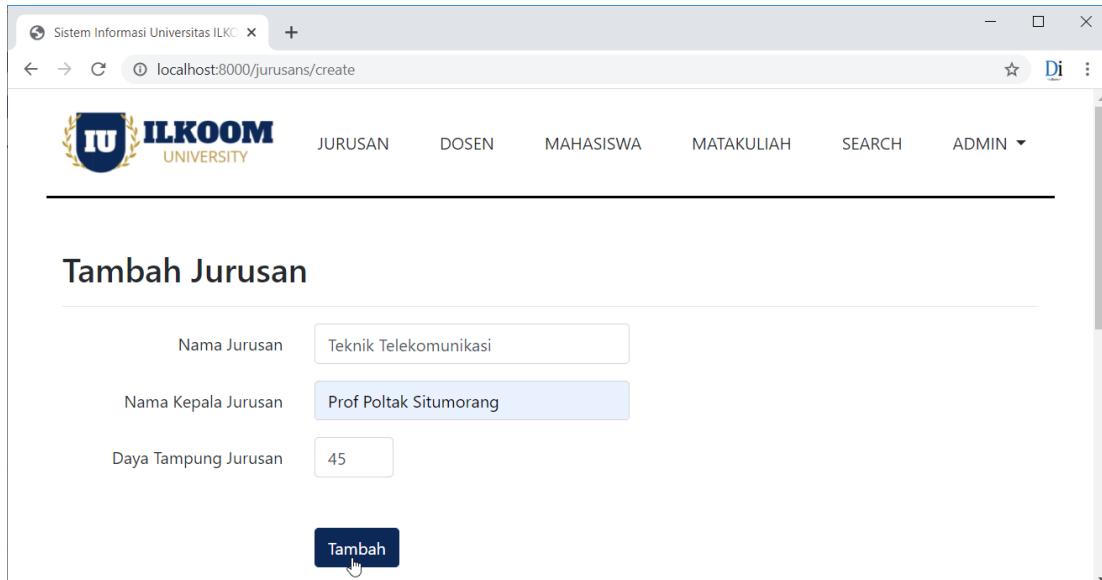
Sebelum uji coba menambah Jurusan, ada satu lagi yang kurang, yakni menambah property `$fillable` ke model `Jurusan.php`. Silahkan buka file model ini lalu tambah 1 baris berikut:

app\Models\Jurusan.php

```
1 ...
2 class Jurusan extends Model
3 {
4     protected $fillable = [ 'nama', 'kepala_jurusan', 'daya_tampung' ];
5 }
```

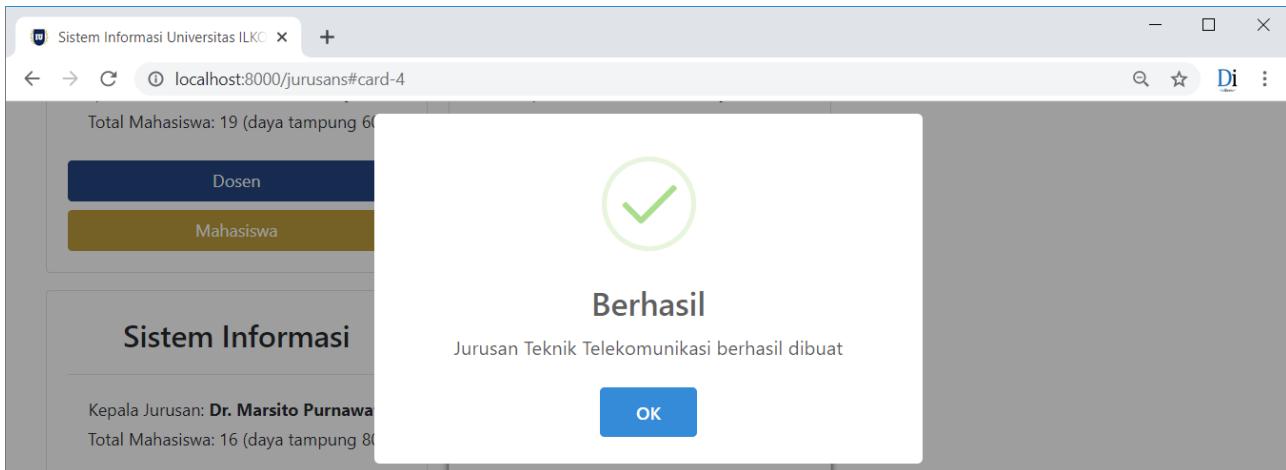
Ini diperlukan karena kita memakai perintah `mass assignment` untuk menginput data jurusan di controller, yakni perintah `Jurusan::create($validateData)`.

Semua persiapan sudah selesai, saatnya uji coba menginput data jurusan:

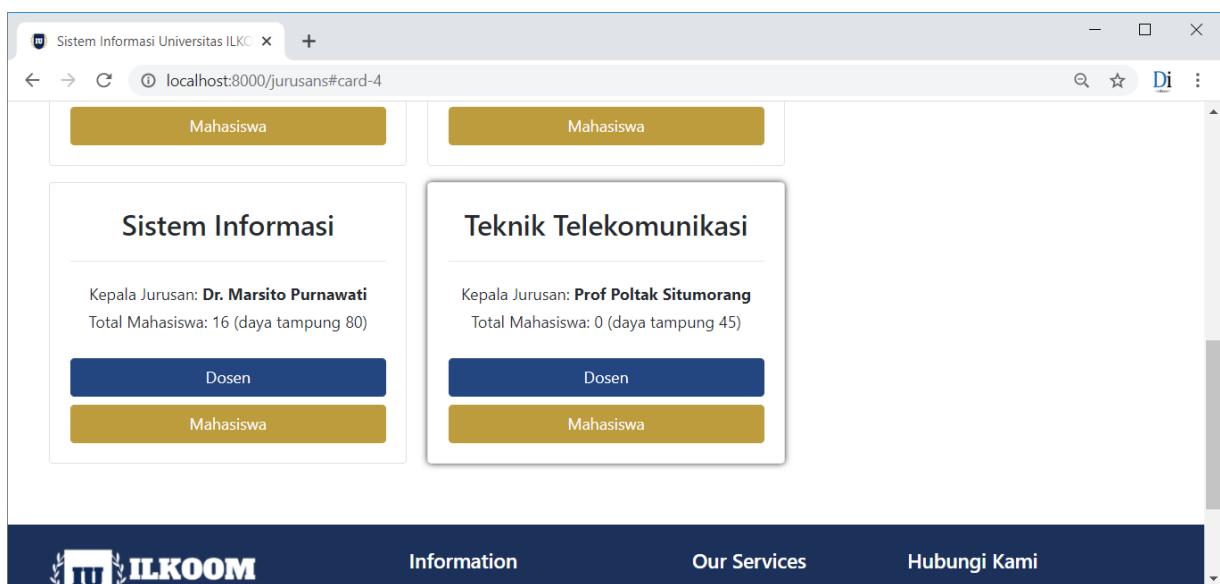


The screenshot shows a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/jurusans/create". The page header includes the university logo and navigation links for JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and ADMIN. The main content area is titled "Tambah Jurusan". It contains three input fields: "Nama Jurusan" with value "Teknik Telekomunikasi", "Nama Kepala Jurusan" with value "Prof Poltak Situmorang", and "Daya Tampung Jurusan" with value "45". A large blue "Tambah" button is at the bottom of the form.

Gambar: Tambah Jurusan "Teknik Telekomunikasi"



Gambar: Jendela alert penambahan jurusan



Gambar: Jurusan "Teknik Telekomunikasi" berhasil di tambah

Dalam contoh ini saya menambah jurusan "Teknik Telekomunikasi". Silahkan tes menambah beberapa jurusan lain.

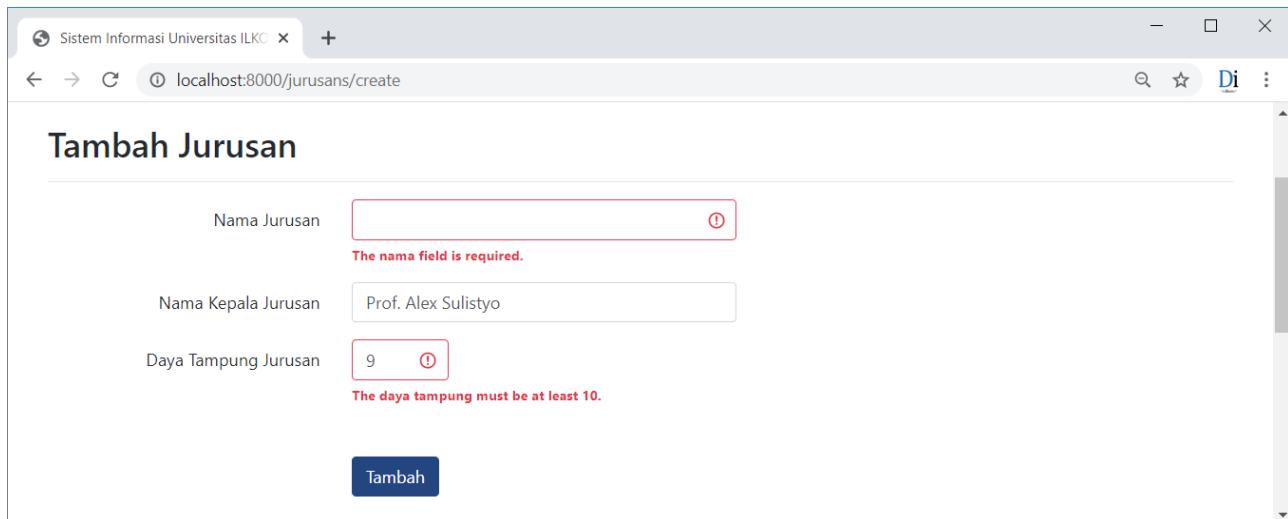
Kotak jurusan yang baru saja ditambah akan memiliki efek border abu-abu. Inilah fungsi dari *hash fragment* sebelumnya yang dikombinasikan dengan kode CSS berikut:

resources\sass\my-style.scss

```
1 .card:target{  
2     box-shadow: 0px 0px 6px;  
3 }
```

Kode ini sudah tersedia di file `my-style.scss` pada saat proses compile menggunakan Laravel Mix. Jika anda iseng mengedit alamat URL menjadi '`localhost:8000/jurusans#card-2`', maka efek border abu-abu akan pindah ke jurusan yang memiliki id = 2.

Mari test validasi form dengan mengisi data kosong atau data yang tidak sesuai:



The screenshot shows a web page titled "Tambah Jurusan". It contains three input fields: "Nama Jurusan" (empty), "Nama Kepala Jurusan" (containing "Prof. Alex Sulistyо"), and "Daya Tampung Jurusan" (containing "9"). Below each input field is a red error message: "The nama field is required." and "The daya tampung must be at least 10.". A blue "Tambah" button is at the bottom.

Gambar: Tampilan pesan error form Tambah Jurusan

Sip, pesan error tampil sempurna dan proses re-populate form juga sudah berhasil.

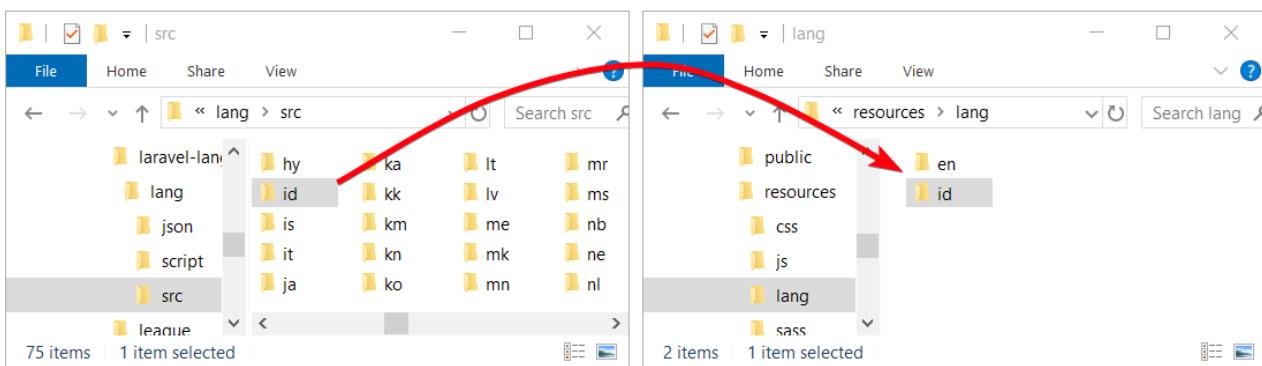
Instalasi Localization

Pesan error yang tampil masih dalam bahasa inggris. Agar bisa di translate ke Bahasa Indonesia, perlu install file localization. Cara ini juga pernah kita bahas di buku Laravel Uncover.

Silahkan buka cmd, masuk ke folder instalasi Laravel dan ketik kode berikut:

```
composer require laravel-lang/lang:~7.0
```

Setelah selesai, copy isi folder vendor\laravel-lang\lang\src\ ke dalam folder resources\lang\.



Gambar: Copy folder lang id ke resources\lang

Lalu ubah pengaturan locale di file config\app.php dari 'locale'=>'en', menjadi 'locale'=>'id'. Dengan penambahan ini, pesan error form akan tampil dalam Bahasa Indonesia:

The screenshot shows a web page titled "Tambah Jurusan". It contains three input fields: "Nama Jurusan" (empty, highlighted in red) with the error message "nama wajib diisi.", "Nama Kepala Jurusan" (filled with "Prof. Alex Sulistyo"), and "Daya Tampung Jurusan" (empty, highlighted in red) with the error message "daya tampung wajib diisi.". Below the form is a blue "Tambah" button.

Gambar: Tampilan pesan error form Tambah Jurusan dalam Bahasa Indonesia

23.3. Menambah Data Dosen

Sekarang kita masuk ke pembuatan form tambah dosen. Cara yang dipakai tetap sama seperti pada form jurusan. Pertama, tempatkan tombol "**Tambah Dosen**" di bagian atas file `dosen\index.blade.php`:

`resources\views\dosen\index.blade.php`

```
1 ...  
2 <h1 class="display-4 text-center my-5" id="judul">  
3   Data Dosen {{ $nama_jurusan ?? 'Universitas ILKOOOM' }}  
4 </h1>  
5  
6 <div class="text-right py-4">  
7   @auth  
8     <a href="{{ route('dosens.create') }}" class="btn btn-info">Tambah Dosen</a>  
9   @endauth  
10 </div>  
11  
12 <table class="table table-striped">  
13   <thead>  
14   ...
```

The screenshot shows a table of student data with columns for #, NID, Name, and Major. A red arrow points to a blue 'Tambah Dosen' button in the top right corner.

#	NID	Nama Dosen	Jurusan Dosen
1	99154082	Betania Yolanda M.Sc	Teknik Informatika
2	99629495	Darmanto Agustina M.T	Teknik Informatika
3	99574701	Edward Prabowo M.Sc	Ilmu Komputer

Gambar: Tombol "Tambah Dosen"

Tombol ini juga berada pada blok @auth, sehingga hanya bisa diakses bagi user yang sudah login.

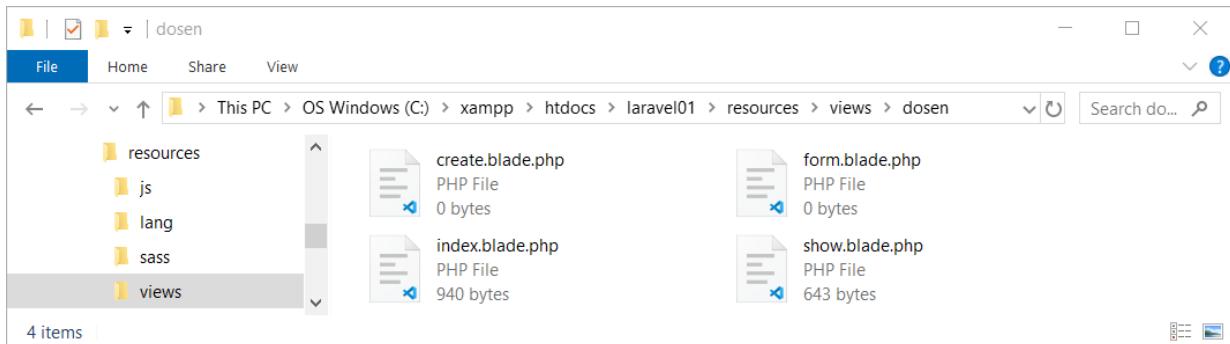
Pada saat tombol di klik, akan menuju ke named route 'dosens.create' yang diproses oleh method create() di DosenController.php. Berikut kode yang diperlukan:

app\Http\Controllers\DosenController.php

```
1 <?php
2 ...
3 use App\Models\Jurusan;
4 ...
5     public function create()
6     {
7         $jurusans = Jurusan::orderBy('nama')->get();
8         return view('dosen.create',compact('jurusans'));
9     }
```

Di baris 7 saya mengambil semua data jurusan, yang akan dipakai untuk membuat menu pilihan jurusan sesampainya di form nanti. Variabel \$jurusans kemudian di kirim ke view 'dosen.create' pada baris 8.

Saatnya masuk ke view, silahkan buat 2 buah file di dalam folder resources\views\dosen, yakni file create.blade.php dan file form.blade.php:



Gambar: Isi folder resources\views\dosen

File `create.blade.php` berfungsi sebagai view induk, sedangkan struktur form akan berada di file `form.blade.php`.

Berikut kode untuk view `create.blade.php`:

`resources\views\dosen\create.blade.php`

```

1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Tambah Dosen</h1>
6 </div>
7 <hr>
8
9 <form method="POST" action="{{ route('dosens.store') }}">
10  @include('dosen.form',['tombol' => 'Tambah'])
11 </form>
12
13 @endsection

```

Kode yang ada sangat mirip seperti view jurusan, hanya saja sekarang data form akan dikirim ke *named route* '`dosens.store`'.

Lanjut, berikut kode untuk view `form.blade.php`:

`resources\views\dosen\form.blade.php`

```

1 @csrf
2 <div class="form-group row">
3   <label for="nid" class="col-md-3 col-form-label text-md-right"
4     title="8 digit angka NID">
5     Nomor Induk Dosen </label>
6   <div class="col-md-6">
7     <input id="nid" type="text"
8       class="form-control col-md-8 @error('nid') is-invalid @enderror"
9       name="nid" value="{{ old('nid') ?? $dosen->nid ?? '' }}" autofocus
10      placeholder="8 digit angka NID, contoh: 99754972">
11      @error('nid')
12        <span class="invalid-feedback" role="alert">
13          <strong>{{ $message }}</strong>

```

```

14      </span>
15      @enderror
16  </div>
17 </div>
18
19 <div class="form-group row">
20   <label for="nama" class="col-md-3 col-form-label text-md-right">
21     Nama Dosen </label>
22   <div class="col-md-6">
23     <input id="nama" type="text"
24     class="form-control col-md-8 @error('nama') is-invalid @enderror"
25     name="nama" value="{{ old('nama') ?? $dosen->nama ?? '' }}>
26     @error('nama')
27       <span class="invalid-feedback" role="alert">
28         <strong>{{ $message }}</strong>
29       </span>
30     @enderror
31   </div>
32 </div>
33
34 <div class="form-group row">
35   <label for="jurusan_id" class="col-md-3 col-form-label text-md-right">
36     Jurusan </label>
37   <div class="col-md-6">
38     <select name="jurusan_id" id="jurusan_id"
39     class="custom-select col-md-5 @error('jurusan_id') is-invalid @enderror"
40     @foreach ($jurusans as $jurusan)
41       @if ($jurusan->id == (old('jurusan_id') ?? $dosen->jurusan_id ?? ''))
42         <option value="{{ $jurusan->id }}" selected>{{ $jurusan->nama }}</option>
43       @else
44         <option value="{{ $jurusan->id }}>{{ $jurusan->nama }}</option>
45       @endif
46     @endforeach
47     </select>
48     @error('jurusan_id')
49       <span class="invalid-feedback" role="alert">
50         <strong>{{ $message }}</strong>
51       </span>
52     @enderror
53   </div>
54 </div>
55
56 {{-- Trik agar bisa kembali ke halaman yang klik --}}
57 @isset($dosen)
58   <input type="hidden" name="url_asal"
59   value="{{ old('url_asal') ?? url()->previous(). '#row-' . $dosen->id }}>
60 @else
61   <input type="hidden" name="url_asal"
62   value="{{ old('url_asal') ?? url()->previous()}}>
63 @endisset
64
65 <div class="form-group row mt-5">
66   <div class="col-md-6 offset-md-3">
67     <button type="submit" class="btn btn-primary">{{$tombol}}</button>
68 </div>

```

69 </div>

The screenshot shows a browser window titled "Sistem Informasi Universitas ILKOOOM". The address bar says "localhost:8000/dosens/create". The main content is a form titled "Tambah Dosen". It contains three fields: "Nomor Induk Dosen" with placeholder "8 digit angka NID, contoh: 99754972", "Nama Dosen" (empty text input), and "Jurusan" with a dropdown menu currently showing "Ilmu Komputer". At the bottom is a blue "Tambah" button.

Gambar: Form Tambah Dosen

Form tambah dosen ini terdiri dari 2 inputan text, 1 menu dropdown serta 1 tombol submit.

Inputan teks dipakai untuk nomor induk dosen (nip) serta nama dosen. Keduanya tidak saya bahas lagi karena sangat mirip seperti di jurusan\form.blade.php.

Menu dropdown berguna untuk pilihan jurusan yang berasal dari tag <select> dan tag <option>. Daftar jurusan terdiri dari dari tag <option> yang di ulang oleh perintah @foreach (\$jurusans as \$jurusan) antara baris 40 – 46.

Di dalam perulangan foreach terdapat kondisi if yang cukup panjang:

```
@if ($jurusan->id == (old('jurusan_id') ?? $dosen->jurusan_id ?? ''))
```

Kode ini dipakai untuk proses re-populate, yakni supaya pilihan jurusan tidak berganti ke pilihan pertama saat halaman reload.

Berikut urutan prioritas dari kondisi if tersebut:

1. Apabila form tampil karena error validasi, variabel `old('jurusan_id')` akan berisi sebuah nilai id.
2. Apabila form tampil dari proses update, variabel `$dosen->jurusan_id` akan berisi sebuah nilai id.
3. Apabila form tampil dari proses create, maka akan dipakai string kosong ''.

Misalnya saat ini saya memilih jurusan "Ilmu Komputer" dan men-klik tombol submit. Jika ada data form yang tidak lolos validasi, Laravel akan me-redirect balik ke halaman form beserta nilai `old('jurusan_id')`.

Nilai `old('jurusan_id')` akan dibandingkan dengan `$jurusan->id`. Jika hasilnya **true**, tag <option> yang sesuai akan ditambah atribut `selected` supaya tetap terpilih.

Di baris 57 – 63 terdapat kode untuk membuat tag <input> bertipe hidden. Karena bertipe

hidden, inputan dengan atribut name="url_asal" ini tidak akan terlihat di web browser tapi tetap ikut dikirim sebagai bagian dari form.

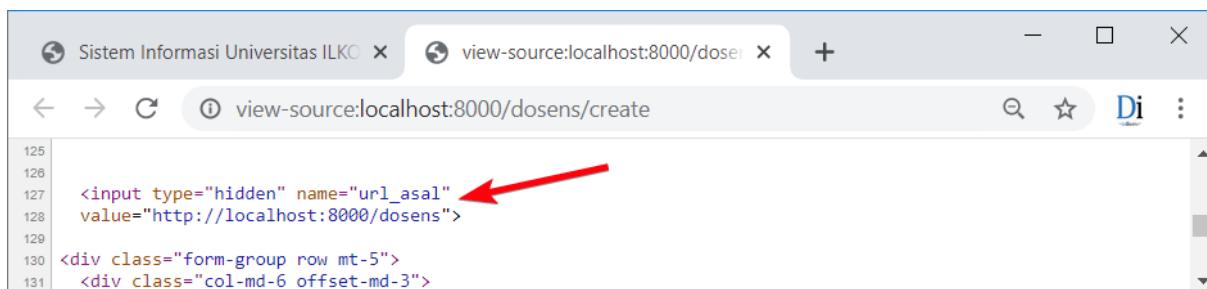
Nantinya, form di atas bisa diakses dari berbagai tempat dan untuk berbagai keperluan. Di dalam controller, saya perlu informasi halaman mana yang mengakses form agar bisa me-redirect user kembali ke halaman tersebut.

Blok kondisi @isset(\$dosen) berfungsi untuk menentukan apakah form di load untuk proses create atau proses update. Perbedaan antara keduanya hanya di penambahan hash fragment setelah function url()->previous().

Dalam Laravel, function url()->previous() bisa dipakai untuk mencari tahu apa alamat URL yang diakses user sebelum halaman saat ini.

Sebagai contoh, jika form diakses dari tombol "Tambah Dosen" yang ada di halaman '<http://localhost:8000/dosens>', maka function url()->previous() akan berisi string alamat <http://localhost:8000/dosens>.

Ini bisa langsung kita coba. Silahkan buka halaman <http://localhost:8000/dosens>, klik tombol "Tambah Dosen" dan lihat source code halaman form tersebut:



```
125  
126  
127 <input type="hidden" name="url_asal" value="http://localhost:8000/dosens">  
128  
129  
130 <div class="form-group row mt-5">  
131 <div class="col-md-6 offset-md-3">
```

Gambar: Nilai url()->previous()

Hasilnya, atribut value dari inputan url_asal akan berisi string '<http://localhost:8000/dosens>'. Fungsi dari inputan ini akan lebih jelas saat kita masuk ke controller sesaat lagi.

Terakhir, terdapat tombol submit di bagian paling bawah. Teks untuk tombol ini ditulis sebagai variabel {{\$tombol}} agar bisa di modifikasi pada saat proses include.

Form kita sudah selesai, total terdapat 4 inputan form untuk nid, nama, jurusan_id, dan url_asal. Nilai form akan di proses oleh named route dosens.store yang terhubung ke method store() di file DosenController.php.

Berikut kode program untuk method tersebut:

app\Http\Controllers\DosenController.php

```
1 <?php  
2 ...  
3 use RealRashid\SweetAlert\Facades\Alert;  
4 ...  
5 public function store(Request $request)
```

```

6      {
7          $validateData = $request->validate([
8              'nid' => 'required|alpha_num|size:8|unique:dosens,nid',
9              'nama' => 'required',
10             'jurusan_id' => 'required|exists:App\Models\Jurusan,id',
11         ]);
12         Dosen::create($validateData);
13         Alert::success('Berhasil', "Dosen $request->nama berhasil dibuat");
14         return redirect($request->url_asal);
15     }

```

Syarat validasi ada di baris 8 – 10, dimana saya ingin ketiga inputan tidak boleh kosong (**required**).

Untuk **nid**, hanya bisa menerima karakter huruf dan angka saja (**alpha_num**), harus terdiri dari 8 digit (**size:8**), dan tidak boleh sama dengan data **nid** yang sudah ada di tabel **dosens** (**unique:dosens,nid**).

Sedangkan untuk **jurusans_id**, nilainya juga harus sesuai dengan kolom **id** di tabel **jurusans** (berasal dari syarat **exists:App\Models\Jurusan,id**).

Jika semua syarat validasi terpenuhi, perintah eloquent di baris 12 akan dijalankan. Kemudian buat pesan alert dan user di **redirect** ke halaman **\$request->url_asal**. Nilai **\$request->url_asal** ini di dapat dari inputan hidden pada akhir form.

Sebelum uji coba, jangan lupa menambah property **\$fillable** ke model **Dosen.php**:

app\Models\Dosen.php

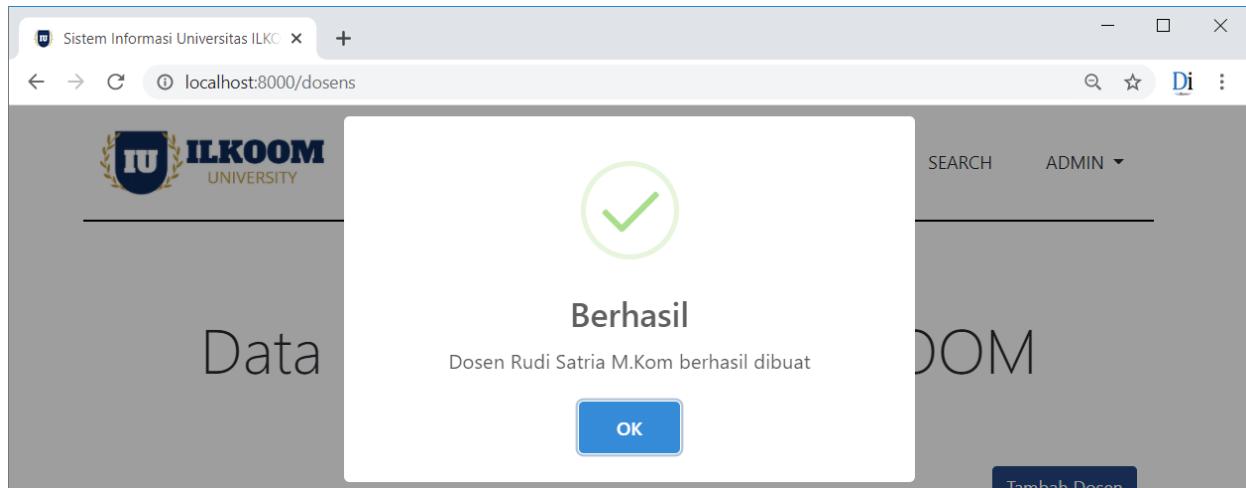
```

1 ...
2 class Dosen extends Model
3 {
4     protected $fillable = ['nid', 'nama', 'jurusan_id'];
5 ...

```

Dan... mari kita test form input data dosen ini:

Gambar: Input data dosen



Gambar: Jendela alert berhasil input data dosen

 A screenshot of a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL is "localhost:8000/dosens?page=2#judul". The main content area displays a table with columns: #, NID, Nama Dosen, and Jurusan Dosen. The table contains five rows of data. A red arrow points to the last row, which shows "Rudi Satria M.Kom" under the "Nama Dosen" column. The table has a navigation bar at the bottom with buttons for <, 1, 2, 3, >.

#	NID	Nama Dosen	Jurusan Dosen
6	99092104	Kuncara Purnawati M.Kom	Sistem Informasi
7	99812991	Putu Prasetya M.Sc	Sistem Informasi
8	99605319	Queen Suryatmi M.T	Sistem Informasi
9	99750967	Rahmi Prasasta M.T	Sistem Informasi
10	99754992	Rudi Satria M.Kom	Teknik Informatika

Gambar: Dosen 'Rudi Satria M.Kom' berhasil masuk ke database

Data dosen berhasil diinput ke database!

Satu kelemahan yang terasa adalah, begitu proses input selesai, halaman akan di redirect ke `localhost:8000/dosens`. Halaman ini berisi tabel yang menampilkan semua dosen, tapi kita tidak tau pada posisi mana dosen yang baru saja ditambah.

Tabel dosen di halaman `localhost:8000/dosens` saya set agar terurut berdasarkan nama, sehingga bisa saja dosen itu berada di halaman ke 2 atau halaman ke 3 karena dipecah oleh pagination.

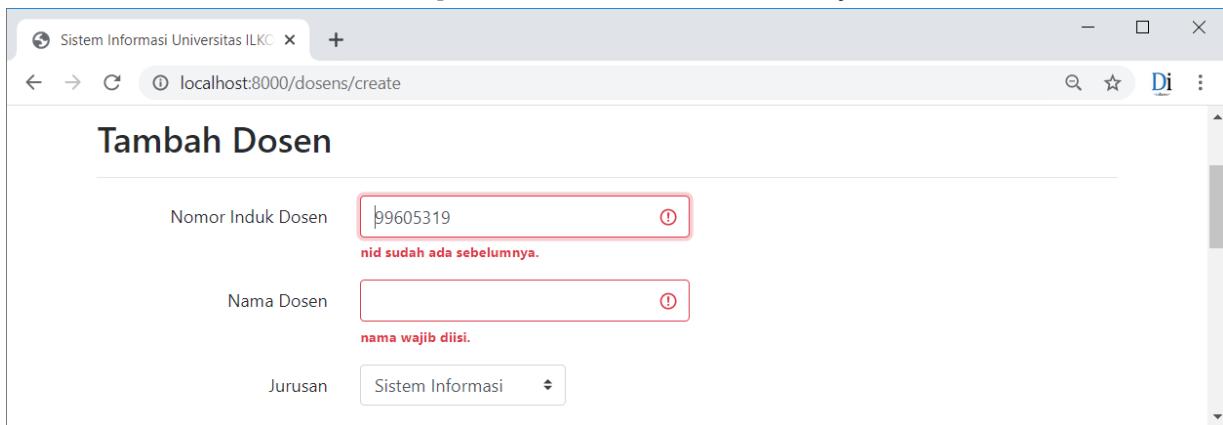
Di sini kita tidak bisa menggunakan fitur `hash fragment` karena tidak mendapat info pada halaman pagination mana dosen tersebut berada. Di form jurusan, ini bisa dilakukan karena halaman `localhost:8000/jurusans` tidak memakai pagination.

Salah satu solusi adalah dengan mengurutkan tabel dosen berdasarkan kolom `created_at`. Dengan demikian, dosen terbaru akan selalu berada di posisi paling atas. Akan tetapi ini jadi menyulitkan proses mencari karena urutan dosen menjadi acak.

Keputusan apakah ingin mengurutkan tabel berdasarkan nama atau tanggal perlu dipertimbangkan sisi plus dan minusnya. Untuk kali ini, saya akan tetap biarkan data tabel dosen tersusun berdasarkan kolom nama.

Solusi yang lebih baik adalah membuat tampilan tabel dinamis dengan urutan yang bisa diubah-ubah. Misal dengan men-klik judul kolom nama, data akan otomatis terurut berdasarkan nama, atau jika di klik judul kolom jurusan, data akan terkelompok berdasarkan jurusan. Fitur ini cukup rumit untuk dibuat dan umumnya memakai library khusus seperti [datatables.net](#).

Pemeriksaan terakhir, mari coba apakah validasi form sudah berjalan atau tidak:



Gambar: Pesan error validasi data dosen

Sip, pesan error sudah tampil. Saatnya lanjut ke form input data mahasiswa.

23.4. Menambah Data Mahasiswa

Form penambahan data mahasiswa juga sama seperti form dosen. Kita akan buat tombol "**Tambah Mahasiswa**" di bagian atas file `mahasiswa\index.blade.php`:

`resources\views\mahasiswa\index.blade.php`

```
1 ...  
2 <h1 class="display-4 text-center my-5" id="judul">  
3   Mahasiswa {{ $nama_jurusan ?? 'Universitas ILKOOM' }}  
4 </h1>  
5  
6 <div class="text-right py-4">  
7   @auth  
8     <a href="{{ route('mahasiswa.create') }}"  
9       class="btn btn-info">Tambah Mahasiswa</a>  
10  @endauth  
11 </div>  
12  
13 <table class="table table-striped">
```

```
14 <thead>
15 ...
```

The screenshot shows a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/mahasiswa". The page displays a table of student information:

#	NIM	Nama Mahasiswa	Jurusan Mahasiswa
1	10648094	Ani Suwarno	Sistem Informasi

Below the table is a blue button labeled "Tambah Mahasiswa" with a hand cursor icon over it.

Gambar: Tombol "Tambah Mahasiswa"

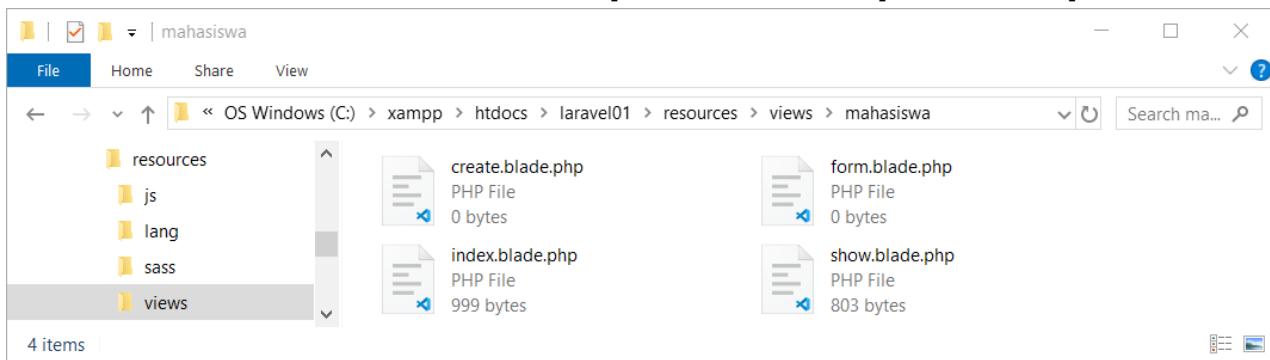
Berikutnya masuk ke method `create()` di `MahasiswaController.php`. Method inilah yang akan dijalankan saat tombol "Tambah Mahasiswa" di klik:

`app\Http\Controllers\MahasiswaController.php`

```
1 <?php
2 ...
3 use App\Models\Jurusan;
4 ...
5     public function create()
6     {
7         $jurusans = Jurusan::orderBy('nama')->get();
8         return view('mahasiswa.create',compact('jurusans'));
9     }
```

Kode ini sangat mirip seperti method `create()` di `DosenController.php`. Di baris 7 saya kembali mengambil data jurusan yang nantinya dipakai untuk membuat menu dropdown.

Lanjut, silahkan buat file `create.blade.php` dan file `form.blade.php` ke dalam folder `resources\views\mahasiswa`. Kedua file ini dipakai untuk menampilkan form input:



Gambar: Isi folder `resources\views\mahasiswa`

File create.blade.php berfungsi sebagai view induk. Berikut kode yang diperlukan:

resources\views\mahasiswa\create.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Tambah Mahasiswa</h1>
6 </div>
7 <hr>
8
9 <form method="POST" action="{{ route('mahasiswa.store') }}">
10   @include('mahasiswa.form',['tombol' => 'Tambah'])
11 </form>
12
13 @endsection
```

Sepertinya tidak ada yang perlu kita bahas karena sangat mirip seperti kode yang sudah ada.

Mari lanjut ke form.blade.php:

resources\views\mahasiswa\form.blade.php

```
1 @csrf
2 <div class="form-group row">
3   <label for="nid" class="col-md-3 col-form-label text-md-right">
4     title="8 digit angka NID">
5     Nomor Induk Dosen </label>
6   <div class="col-md-6">
7     <input id="nid" type="text"
8     class="form-control col-md-8 @error('nid') is-invalid @enderror"
9     name="nid" value="{{ old('nid') ?? $dosen->nid ?? '' }}" autofocus
10    placeholder="8 digit angka NID, contoh: 99754972">
11    @error('nid')
12      <span class="invalid-feedback" role="alert">
13        <strong>{{ $message }}</strong>
14      </span>
15    @enderror
16  </div>
17 </div>
18
19 <div class="form-group row">
20   <label for="nama" class="col-md-3 col-form-label text-md-right">
21     Nama Dosen </label>
22   <div class="col-md-6">
23     <input id="nama" type="text"
24     class="form-control col-md-8 @error('nama') is-invalid @enderror"
25     name="nama" value="{{ old('nama') ?? $dosen->nama ?? '' }}">
26     @error('nama')
27       <span class="invalid-feedback" role="alert">
28         <strong>{{ $message }}</strong>
29       </span>
30     @enderror
31 </div>
```

```

32 </div>
33
34 <div class="form-group row">
35   <label for="jurusan_id" class="col-md-3 col-form-label text-md-right">
36     Jurusan </label>
37   <div class="col-md-6">
38     <select name="jurusan_id" id="jurusan_id"
39       class="custom-select col-md-5 @error('jurusan_id') is-invalid @enderror">
40       @foreach ($jurusans as $jurusan)
41         @if ($jurusan->id == (old('jurusan_id') ?? $dosen->jurusan_id ?? ''))
42           <option value="{{ $jurusan->id }}" selected>{{ $jurusan->nama }}</option>
43         @else
44           <option value="{{ $jurusan->id }}>{{ $jurusan->nama }}</option>
45         @endif
46       @endforeach
47     </select>
48     @error('jurusan_id')
49       <span class="invalid-feedback" role="alert">
50         <strong>{{ $message }}</strong>
51       </span>
52     @enderror
53   </div>
54 </div>
55
56 {{-- Trik agar bisa kembali ke halaman yang klik --}}
57 @isset($dosen)
58   <input type="hidden" name="url_asal"
59   value="{{ old('url_asal') ?? url()->previous().'#row-' . $dosen->id}}">
60 @else
61   <input type="hidden" name="url_asal"
62   value="{{ old('url_asal') ?? url()->previous()}}">
63 @endisset
64
65 <div class="form-group row mt-5">
66   <div class="col-md-6 offset-md-3">
67     <button type="submit" class="btn btn-primary">{{ $tombol }}</button>
68   </div>
69 </div>

```

The screenshot shows a browser window with the title 'Sistem Informasi Universitas ILKOOOM'. The URL in the address bar is 'localhost:8000/mahasiswas/create'. The page content is titled 'Tambah Mahasiswa'. It contains three form fields: 'Nomor Induk Mahasiswa' with a placeholder '8 digit angka NIM, contoh: 10311492', 'Nama Mahasiswa' (empty input field), and 'Jurusan' (dropdown menu showing 'Ilmu Komputer'). Below the form is a blue 'Tambah' button.

Gambar: Form Tambah Mahasiswa

Form tambah mahasiswa terdiri dari 2 inputan text untuk nomor induk mahasiswa (nim) dan nama mahasiswa, kemudian 1 menu dropdown untuk pilihan jurusan, serta 1 tombol submit.

Kode yang digunakan sangat mirip seperti form penambahan data dosen, termasuk inputan hidden `url_asal` di bagian akhir form.

Pada saat di submit, form akan mengirim 4 data: `nim`, `nama`, `jurusan_id` serta `url_asal`. Data ini akan diproses oleh method `store()` di `MahasiswaController.php`:

app\Http\Controllers\MahasiswaController.php

```

1  <?php
2  ...
3  use RealRashid\SweetAlert\Facades\Alert;
4  ...
5      public function store(Request $request)
6      {
7          $validateData = $request->validate([
8              'nim' => 'required|alpha_num|size:8|unique:mahasiswas,nim',
9              'nama' => 'required',
10             'jurusan_id' => 'required|exists:App\Models\Jurusan,id',
11         ]);
12
13         // Cek apakah daya tampung jurusan masih belum penuh
14         $daya_tampung = Jurusan::find($request->jurusan_id)->daya_tampung;
15         $total_mahasiswa = Mahasiswa::where('jurusan_id',$request->jurusan_id)
16             ->count();
17         if ($total_mahasiswa >= $daya_tampung){
18             Alert::error('Pendaftaran Gagal',
19                         'Sudah melebihi daya tampung jurusan');
20             return back()->withInput();
21         }
22
23         Mahasiswa::create($validateData);
24         Alert::success('Berhasil', "Mahasiswa $request->nama berhasil dibuat");
25         return redirect($request->url_asal);
26     }

```

Di baris 8 – 10 terdapat syarat validasi untuk inputan form, dimana ketiga inputan harus diisi (`required`).

Nilai `nim` hanya bisa menerima karakter huruf dan angka saja (`alpha_num`), terdiri dari 8 digit (`size:8`), dan tidak boleh sama dengan data nim yang sudah ada di tabel `mahasiswas` (`unique: mahasiswas,nim`).

Untuk nilai `jurusan_id`, harus sesuai dengan kolom `id` yang ada di tabel `jurusans` (berasal dari syarat `exists:App\Models\Jurusan,id`).

Tambahan pemeriksaan ada di baris 14 – 21. Ini di pakai untuk memastikan daya tampung jurusan belum penuh. Jika anda masih ingat, di tabel `jurusans` terdapat kolom `daya_tampung`. Daya tampung ini merujuk ke jumlah total mahasiswa yang bisa diterima oleh sebuah jurusan.

Logikanya, jika daya tampung jurusan sudah penuh, mahasiswa baru tidak bisa lagi kita tambah. Cara pemeriksaan syarat ini dilakukan sebagai berikut:

1. Ambil total mahasiswa dari jurusan yang dipilih. Ini dilakukan oleh kode program di baris 15 – 16, yang hasilnya disimpan ke dalam variabel `$total_mahasiswa`.
2. Ambil jumlah daya tampung dari jurusan yang dipilih, ini dilakukan oleh kode program di baris 14. Hasilnya disimpan ke dalam variabel `$daya_tampung`.
3. Bandingkan isi variabel `$total_mahasiswa` dengan `$daya_tampung`. Jika total mahasiswa sudah sama atau lebih dari daya tampung, maka tolak penambahan data (artinya jurusan sudah penuh). Kemudian buat pesan alert dan *redirect* user kembali ke halaman form dengan perintah `return back()->withInput()`.

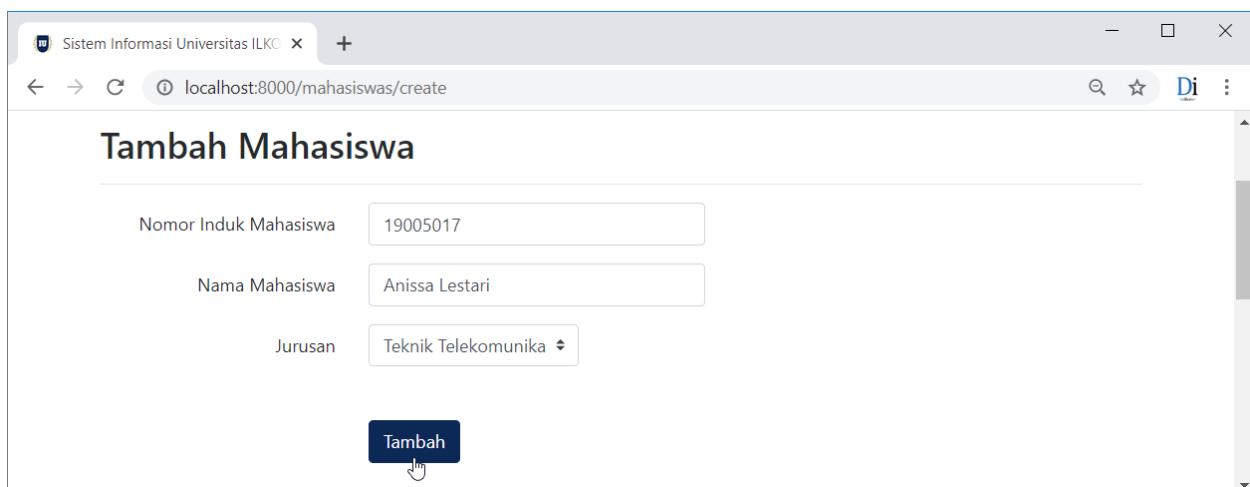
Jika pemeriksaan kondisi di baris 14 – 21 terlewati (yang berarti daya tampung masih tersedia), maka input data mahasiswa ke dalam tabel dengan perintah eloquent di baris 23, kemudian buat pesan alert dan *redirect* user ke halaman `$request->url_asal`,

Sebelum kita uji coba, tambah property `$fillable` ke model `Mahasiswa.php`:

`app\Models\Mahasiswa.php`

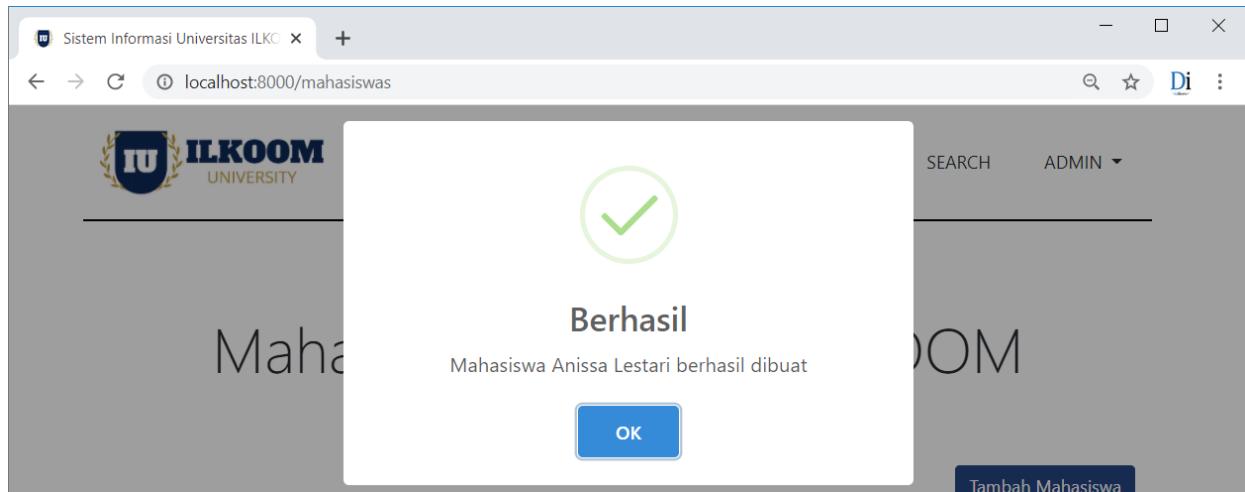
```
1 ...  
2 class Mahasiswa extends Model  
3 {  
4     protected $fillable = ['nim', 'nama', 'jurusan_id'];  
5 }
```

Done! mari kita test input data mahasiswa:



The screenshot shows a browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/mahasiswas/create". The page displays a form titled "Tambah Mahasiswa" (Add Student). The form contains three input fields: "Nomor Induk Mahasiswa" with the value "19005017", "Nama Mahasiswa" with the value "Anissa Lestari", and "Jurusan" with the value "Teknik Telekomunika". Below the form is a blue "Tambah" (Add) button with a hand cursor icon pointing at it.

Gambar: Input data mahasiswa



Gambar: Jendela alert berhasil input data mahasiswa

#	NIM	Nama Mahasiswa	Jurusan Mahasiswa
1	10648094	Ani Suworno	Sistem Informasi
2	19005017	Anissa Lestari	Teknik Telekomunikasi
3	10388950	Bagiya Tampubolon	Ilmu Komputer
4	1246037	Bahuwarna Hartati	Ilmu Komputer

Gambar: Mahasiswa 'Anissa Lestari' berhasil masuk ke database

Tidak ada masalah, data mahasiswa berhasil masuk ke tabel `mahasiswa`. Sekarang tes apakah validasi form sudah berjalan atau tidak:

A screenshot of a web browser window showing a validation error for a new student entry. The title bar says 'Sistem Informasi Universitas ILKOOOM'. The URL is 'localhost:8000/mahasiswa/create'. The main content area has a heading 'Tambah Mahasiswa'. It contains three form fields: 'Nomor Induk Mahasiswa' with value '10545068' (which is highlighted with a red border and has a red error message below it), 'Nama Mahasiswa' (which is also highlighted with a red border and has a red error message below it), and 'Jurusan' with a dropdown menu set to 'Teknik Informatika'. At the bottom is a blue 'Tambah' button.

Gambar: Pesan error validasi data mahasiswa

Pesan error sukses tampil yang berarti validasi sudah aktif.

Terakhir kita akan uji batasan daya tampung. Saat ini, total jumlah mahasiswa untuk jurusan

Sistem Informasi ada 16 orang (atau lebih jika anda sudah menambah beberapa mahasiswa). Total mahasiswa ini bisa dilihat dari halaman index jurusan:

Gambar: Total mahasiswa jurusan Sistem Informasi

Dalam tampilan di atas, terlihat juga daya tampung jurusan Sistem Informasi adalah 80 orang. Agar kita tidak perlu menambah 64 mahasiswa lagi, saya akan langsung edit isi tabel jurusans.

Silahkan buka cmd MySQL Client atau phpMyAdmin dan ubah manual kolom daya tampung untuk jurusan Sistem Informasi menjadi 16:

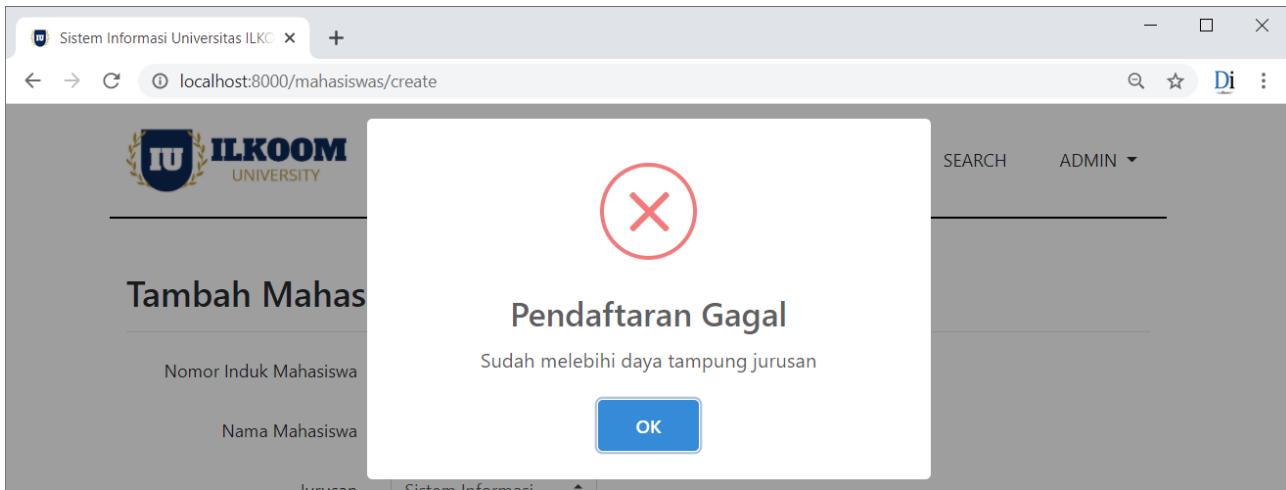
ows:	25	Filter rows:		
Rows per page:				
	id	nama	kepala_jurusan	daya_tampung
e	1	Ilmu Komputer	Dr. Mustofa Simanjuntak	60
e	2	Sistem Informasi	Dr. Marsito Purnawati	16
e	3	Teknik Informatika	Dr. Ika Puspasari	50
e	4	Teknik Telekomunikasi	Prof Poltak Situmorang	45

Gambar: Mengubah daya tampung Jurusan Informasi dari phpMyAdmin

Jika menggunakan cmd MySQL client, ini bisa dilakukan dengan perintah query berikut:

```
UPDATE jurusans SET daya_tampung = 16 WHERE nama = 'Sistem Informasi';
```

Sekarang kita tes tambah satu mahasiswa ke jurusan Sistem Informasi:



Gambar: Penambahan mahasiswa gagal

Hasilnya, tampil error "Pendaftaran Gagal", yang berarti validasi batasan daya tampung juga sudah berjalan. Teknik seperti ini bisa dipakai untuk membuat syarat validasi lain yang tidak disediakan Laravel.

23.5. Menambah Data Matakuliah

Pembuatan form tambah data matakuliah juga sama seperti sebelumnya. Silahkan buka file `matakuliah\index.blade.php` dan ketik kode berikut untuk menambah tombol "**Tambah Mata Kuliah**" :

`resources\views\matakuliah\index.blade.php`

```
1 ...  
2 <h1 class="display-4 text-center my-5" id="judul">  
3   Mata Kuliah Universitas ILKOOOM  
4 </h1>  
5  
6 <div class="text-right py-4">  
7   @auth  
8     <a href="{{ route('matakuliah.create') }}"  
9       class="btn btn-info">Tambah Mata Kuliah</a>  
10  @endauth  
11 </div>  
12  
13 <table class="table table-striped">  
14   <thead>  
15   ...
```

#	Kode	Nama Mata Kuliah	Dosen Pengajar	Jumlah SKS	Jurusan
1	RM463	Agama dan Etika	Kuncara Purnawati M.Kom	2	Sistem Informasi
		Prisma & Struktur Data	Halima Mansur M.Sc	4	Sistem Informasi

Gambar: Tombol "Tambah Mata Kuliah"

Lanjut ke file `MatakuliahController.php`, tambah kode berikut ke method `create()`:

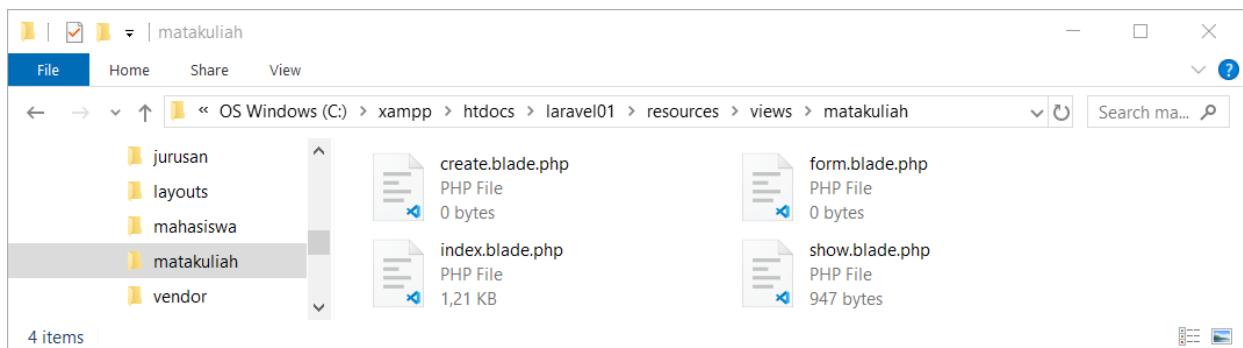
`app\Http\Controllers\MatakuliahController.php`

```

1  <?php
2  ...
3  use App\Models\Jurusan;
4  use App\Models\Dosen;
5
6  public function create()
7  {
8      $jurusans = Jurusan::orderBy('nama')->get();
9      $dosens = Dosen::orderBy('nama')->get();
10     return view('matakuliah.create',[
11         'jurusans' => $jurusans,
12         'dosens' => $dosens,
13     ]);
14 }
```

Selain mengirim data `$jurusans`, saya juga mengirim data `$dosens` karena nantinya kita perlu membuat 2 buah menu dropdown, satu untuk jurusan dan satu lagi untuk pilihan dosen.

Untuk form, buat file `create.blade.php` dan `form.blade.php` di folder `resources\views\matakuliah`:



Gambar: Isi folder `resources\views\matakuliah`

Berikut kode yang diperlukan untuk `create.blade.php`:

`resources\views\matakuliah\create.blade.php`

```

1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Tambah Mata Kuliah</h1>
6 </div>
7 <hr>
8
9 <form method="POST" action="{{ route('matakuliah.store') }}">
10   @include('matakuliah.form',[ 'tombol' => 'Tambah'])
11 </form>
12
13 @endsection

```

Juga sama seperti sebelumnya. Kita langsung masuk ke `form.blade.php`:

`resources\views\mahasiswa\form.blade.php`

```

1 @csrf
2 <div class="form-group row">
3   <label for="kode" class="col-md-3 col-form-label text-md-right"
4     title="5 digit kode mata kuliah">
5     Kode Mata Kuliah </label>
6   <div class="col-md-6">
7     <input name="kode" id="kode" type="text" autofocus
8     class="form-control col-md-8 @error('kode') is-invalid @enderror"
9     value="{{ old('kode') ?? $matakuliah->kode ?? '' }}"
10    placeholder="5 digit kode mata kuliah, contoh: IX264">
11    @error('kode')
12      <span class="invalid-feedback" role="alert">
13        <strong>{{ $message }}</strong>
14      </span>
15    @enderror
16  </div>
17 </div>
18
19 <div class="form-group row">
20   <label for="nama" class="col-md-3 col-form-label text-md-right">
21     Nama Mata Kuliah </label>
22   <div class="col-md-6">
23     <input name="nama" id="nama" type="text"
24     class="form-control col-md-8 @error('nama') is-invalid @enderror"
25     value="{{ old('nama') ?? $matakuliah->nama ?? '' }}"
26     placeholder="Nama Mata Kuliah, contoh: Matematika">
27     @error('nama')
28       <span class="invalid-feedback" role="alert">
29         <strong>{{ $message }}</strong>
30       </span>
31     @enderror
32   </div>

```

```

33 <div class="form-group row">
34   <label for="jurusan_id" class="col-md-3 col-form-label text-md-right">
35     Jurusan </label>
36   <div class="col-md-6">
37     <select name="jurusan_id" id="jurusan_id"
38       class="custom-select col-md-5 @error('jurusan_id') is-invalid @enderror">
39       @foreach ($jurusans as $jurusan)
40         @if ($jurusan->id == (old('jurusan_id') ?? $dosen->jurusan_id ?? ''))
41           <option value="{{ $jurusan->id }}" selected>{{ $jurusan->nama }}</option>
42         @else
43           <option value="{{ $jurusan->id }}">{{ $jurusan->nama }}</option>
44         @endif
45       @endforeach
46     </select>
47     @error('jurusan_id')
48       <span class="invalid-feedback" role="alert">
49         <strong>{{ $message }}</strong>
50       </span>
51     @enderror
52   </div>
53 </div>
54
55 <div class="form-group row">
56   <label for="dosen_id" class="col-md-3 col-form-label text-md-right">
57     Dosen Pengajar </label>
58   <div class="col-md-6">
59     <select name="dosen_id" id="dosen_id"
60       class="custom-select col-md-5 @error('dosen_id') is-invalid @enderror">
61       @foreach ($dosens as $dosen)
62         @if($dosen->id == (old('dosen_id') ?? $matakuliah->dosen->id ?? ''))
63           <option value="{{ $dosen->id }}" selected>{{ $dosen->nama }}</option>
64         @else
65           <option value="{{ $dosen->id }}">{{ $dosen->nama }}</option>
66         @endif
67       @endforeach
68     </select>
69     @error('dosen_id')
70       <span class="invalid-feedback" role="alert">
71         <strong>{{ $message }}</strong>
72       </span>
73     @enderror
74   </div>
75 </div>
76
77 <div class="form-group row">
78   <label for="jumlah_sks" class="col-md-3 col-form-label text-md-right">
79     Jumlah SKS </label>
80   <div class="col-md-6">
81     <select name="jumlah_sks" id="jumlah_sks"
82       class="custom-select col-md-2 @error('jumlah_sks') is-invalid @enderror">
83       @for ($i = 1; $i <= 6; $i++)
84         @if($i == (old('jumlah_sks') ?? $matakuliah->jumlah_sks ?? ''))
85           <option value="{{ $i }}" selected>{{ $i }}</option>
86         @else
87

```

```

88         <option value="{{ $i }}>{{ $i }}</option>
89     @endif
90   @endfor
91 </select>
92 @error('jumlah_sks')
93   <span class="invalid-feedback" role="alert">
94     <strong>{{ $message }}</strong>
95   </span>
96 @enderror
97 </div>
98 </div>
99
100 {{-- Trik agar bisa kembali ke halaman yang klik --}}
101 @isset($matakuliah)
102   <input type="hidden" name="url_asal"
103   value="{{ old('url_asal') ?? url()->previous().'#row-' . $matakuliah->id}}">
104 @else
105   <input type="hidden" name="url_asal"
106   value="{{ old('url_asal') ?? url()->previous()}}">
107 @endisset
108
109 <div class="form-group row mt-5">
110   <div class="col-md-6 offset-md-3">
111     <button type="submit" class="btn btn-primary">{{$tombol}}</button>
112   </div>
113 </div>

```

The screenshot shows a web browser window with the title 'Sistem Informasi Universitas ILKOOOM'. The URL in the address bar is 'localhost:8000/matakuliahhs/create'. The page displays a form titled 'Tambah Mata Kuliah'. The form consists of five input fields: 'Kode Mata Kuliah' (text input with placeholder '5 kode mata kuliah, contoh: IX264'), 'Nama Mata Kuliah' (text input), 'Jurusan' (dropdown menu with 'Ilmu Komputer' selected), 'Dosen Pengajar' (dropdown menu with 'Betania Yolanda M.Sc' selected), and 'Jumlah SKS' (number input with '1'). Below the form is a blue 'Tambah' button.

Gambar: Form Tambah Mata Kuliah

Kode untuk form ini cukup panjang karena memiliki 5 buah inputan.

Di bagian atas terdapat tag `<input>` teks untuk **Kode Mata Kuliah** dan **Nama Mata Kuliah**. Kedua inputan memiliki atribut `name="kode"` dan `name="nama"`. Cara pengisian atribut `value` (untuk nilai awal) serta menampilkan pesan error sama seperti yang kita pakai selama ini.

Kemudian antara baris 34 – 76 terdapat dua menu dropdown untuk inputan **Jurusan** dan

Dosen Pengajar. Kode yang dipakai juga sama seperti pada form dosen dan form mahasiswa, yakni dengan perulangan foreach untuk menampilkan isi variabel \$jurusans dan \$dosens ke dalam tag <option>. Kedua variabel ini dikirim dari method `create()` di controller.

Sedikit penjelasan, jurusan di sini merujuk ke jurusan mata kuliah yang akan dibuat dan tidak berhubungan dengan dosen pengajar. Maksudnya walaupun seorang dosen terdaftar di jurusan Ilmu Komputer, dia bisa saja mengajar mata kuliah untuk jurusan Teknik Informatika.

Ini sekaligus membatasi agar kode program kita tidak terlalu rumit. Jika ingin dosen yang mengajar harus sesuai dengan jurusannya, maka perlu kode AJAX untuk mengubah pilihan nama dosen pada saat menu jurusan di tukar.

Inputan selanjutnya dipakai untuk memilih jumlah sks. Struktur kode yang dipakai sebenarnya sama seperti menu dropdown jurusan dan nama dosen, hanya saja perulangan yang dipakai adalah perulangan for dari 1 sampai 6.

Terakhir terdapat inputan hidden `url_asal` di baris 101 – 107 serta ditutup oleh tombol submit.

Total, pada saat form di kirim ke named route 'matakuliahs.store', ada 6 nilai yang dikirim, yakni `kode`, `nama`, `jurusan_id`, `dosen_id`, `jumlah_sks`, dan `url_asal`. Berikut kode program untuk memproses nilai-nilai ini:

```

1  <?php
2  ...
3  use RealRashid\SweetAlert\Facades\Alert;
4  ...
5  public function store(Request $request)
6  {
7      $validateData = $request->validate([
8          'kode' => 'required|alpha_num|size:5|unique:matakuliahs,kode',
9          'nama' => 'required',
10         'dosen_id' => 'required|exists:App\Models\Dosen,id',
11         'jurusan_id' => 'required|exists:App\Models\Jurusan,id',
12         'jumlah_sks' => 'required|digits_between:1,6',
13     ]);
14     Matakuliah::create($validateData);
15     Alert::success('Berhasil', "Mata Kuliah $request->nama berhasil dibuat");
16     return redirect($request->url_asal);
17 }
```

Berdasarkan syarat validasi, semua inputan harus diisi (`required`).

Nilai `kode` hanya bisa menerima karakter huruf dan angka saja (`alpha_num`), terdiri dari 5 digit (`size:5`), dan tidak boleh sama dengan data nim yang sudah ada di tabel `matakuliahs` (`unique: matakuliahs,kode`).

Untuk nilai `dosen_id`, harus sesuai dengan kolom `id` yang ada di tabel `dosens` (berasal dari syarat `exists:App\Models\Dosen,id`). Begitu juga dengan nilai `jurusans_id` yang harus sesuai dengan kolom `id` di tabel `jurusans` (berasal dari syarat `exists:App\Models\Jurusan,id`).

Serta nilai `jumlah_sks` harus berada dalam jangkauan 1 sampai 6.

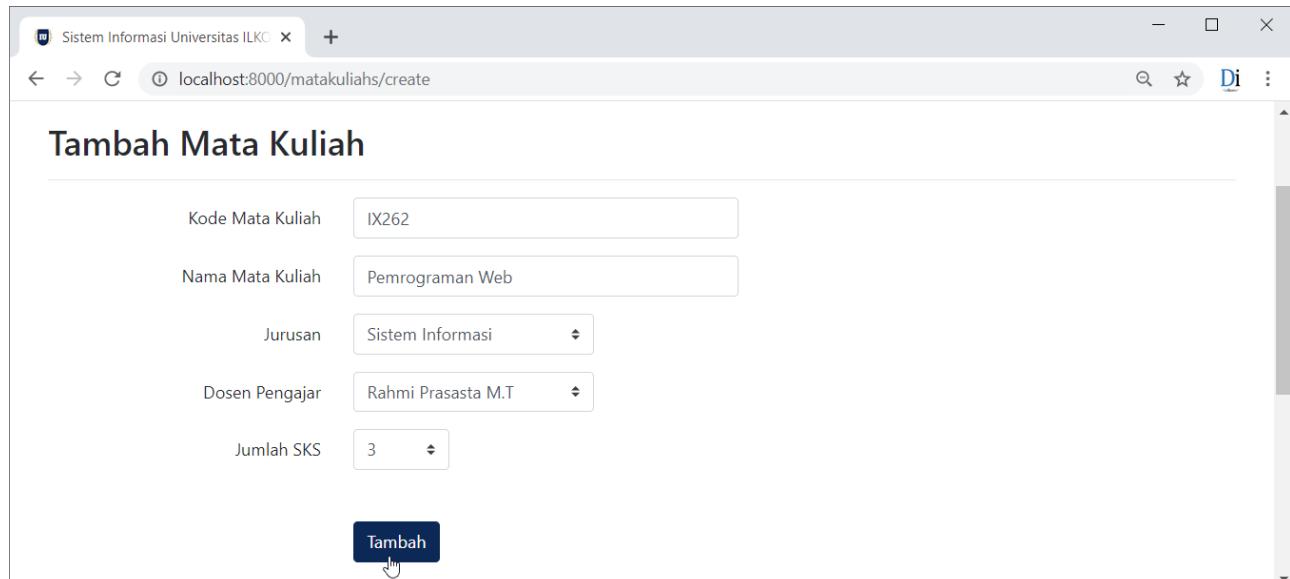
Jika semua syarat validasi terpenuhi, input data mahasiswa ke dalam tabel dengan perintah eloquent di baris 14, kemudian buat pesan alert dan *redirect* user ke halaman `$request->url_asal`.

Seperi biasa, sebelum uji coba silahkan isi property `$fillable` ke model `Matakuliah.php`:

app\Models\Matakuliah.php

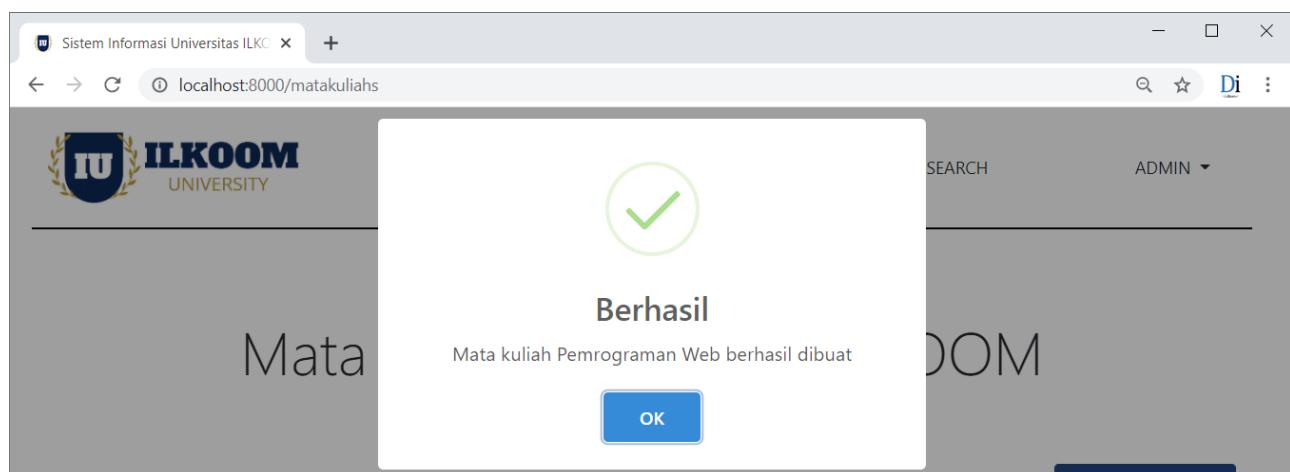
```
1 ...  
2 class Matakuliah extends Model  
3 {  
4     protected $fillable = ['kode', 'nama', 'jurusan_id', 'dosen_id', 'jumlah_sks'];  
5 }
```

Berikut percobaan input data mata kuliah:



The screenshot shows a browser window titled "Sistem Informasi Universitas ILKOOOM". The URL is "localhost:8000/matakuliahs/create". The page title is "Tambah Mata Kuliah". There are five input fields: "Kode Mata Kuliah" (IX262), "Nama Mata Kuliah" (Pemrograman Web), "Jurusan" (Sistem Informasi), "Dosen Pengajar" (Rahmi Prasasta M.T.), and "Jumlah SKS" (3). Below the form is a blue "Tambah" button with a cursor pointing to it.

Gambar: Input data matakuliah



Gambar: Jendela alert berhasil input data matakuliah

25	AX049	Organisasi & Arsitektur Komputer	Darmanto Agustina M.T	4	Teknik Informatika
26	IX262	Pemrograman Web	Rahmi Prasasta M.T	3	Sistem Informasi
27	BY543	Pengantar Analisis Rangkaian	Viktor Pranowo M.Sc	1	Sistem Informasi
localhost:8000/matakuliahs/51					

Gambar: Matakuliah 'Pemrograman Web' berhasil masuk ke database

Tambah Mata Kuliah

Kode Mata Kuliah	IX262	kode sudah ada sebelumnya.
Nama Mata Kuliah		nama wajib diisi.
Jurusan	Sistem Informasi	
Dosen Pengajar	Rahmi Prasasta M.T.	
Jumlah SKS	3	

Tambah

Gambar: Pesan error validasi data matakuliah

Data mata kuliah sudah berhasil di input ke database, serta pesan error juga sukses tampil ketika ada syarat validasi yang tidak terpenuhi.

23.6. Menambah Matakuliah untuk Dosen

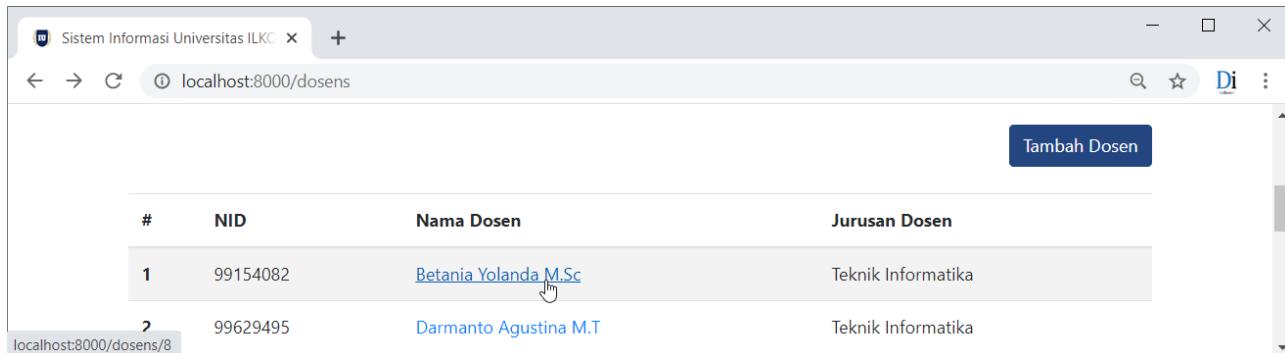
Sampai di sini kita telah memiliki 4 form untuk menambah data jurusan, data dosen, data mahasiswa dan data matakuliah. Keempat form ini sebenarnya sudah melengkapi fitur **create** dari Sistem Informasi Universitas ILKOOOM.

Namun saya masih ingin membuat 3 form tambahan lain, yaitu:

1. Menambah Matakuliah untuk Dosen
2. Mengambil Matakuliah untuk Mahasiswa
3. Mendaftarkan Mahasiswa untuk Matakuliah

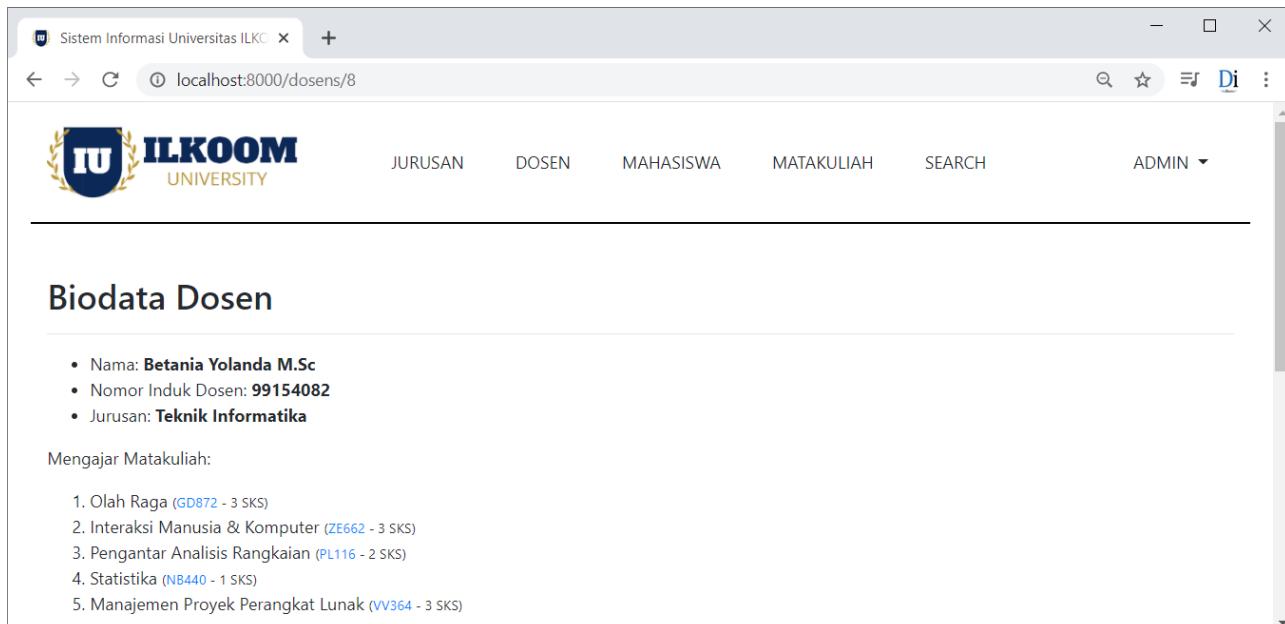
Kita akan bahas yang pertama terlebih dahulu, yakni **Menambah Matakuliah untuk Dosen**. Idenya adalah, saya ingin menambah mata kuliah secara langsung untuk satu dosen.

Saat ini kita sudah memiliki halaman biodata dosen yang bisa diakses dari alamat `localhost:8000/dosens/<id_dosen>`. Halaman ini juga akan tampil saat men-klik nama dosen dari berbagai link yang ada, misalnya dari halaman index dosen:



#	NID	Nama Dosen	Jurusan Dosen
1	99154082	Betania Yolanda M.Sc	Teknik Informatika
2	99629495	Darmanto Agustina M.T	Teknik Informatika

Gambar: Klik nama dosen "Betania Yolanda M.Sc"



Biodata Dosen

- Nama: **Betania Yolanda M.Sc**
- Nomor Induk Dosen: **99154082**
- Jurusan: **Teknik Informatika**

Mengajar Mata Kuliah:

- Olah Raga ([GD872](#) - 3 SKS)
- Interaksi Manusia & Komputer ([ZE662](#) - 3 SKS)
- Pengantar Analisis Rangkaian ([PL116](#) - 2 SKS)
- Statistika ([NB440](#) - 1 SKS)
- Manajemen Proyek Perangkat Lunak ([VV364](#) - 3 SKS)

Gambar: Tampilan halaman show dosen "Betania Yolanda M.Sc"

Pada halaman ini bisa terlihat informasi lengkap dari seorang dosen, mulai nama, nid, jurusan, hingga daftar mata kuliah yang diajar oleh dosen tersebut.

Saya ingin menambah satu tombol "**Buat Mata Kuliah**" di bagian bawah. Ketika tombol ini diklik, akan tampil form tambah mata kuliah, tapi pilihan dosen pengajar sudah langsung berisi dan tidak bisa ditukar lagi.

Berikut tampilan form yang akan kita rancang:

Gambar: Pilihan "Dosen Pengajar" sudah langsung terisi

Bisakah anda bayangkan alur kode program untuk membuat tampilan ini?

Kita mulai dari menambah tombol "**Buat Mata Kuliah**" ke halaman biodata dosen, yang ada di view dosen\show.blade.php. Silahkan buka file ini dan tambah kode berikut di bagian bawah:

resources\views\dosen\show.blade.php

```

6     ...
7     </li>
8     @endforeach
9 </ol>
10
11 @auth
12   <a href="{{ route('buat-matakuliah', [ 'dosen' => $dosen->id])}}"
13     class="btn btn-info" title="Buat Mata Kuliah">Buat Mata Kuliah</a>
14 @endauth
15
16 @endsection

```

Tombol "Buat Mata Kuliah" akan menuju sebuah named route 'buat-matakuliah'. Bersama dengan route tersebut, ikut dikirim variabel 'dosen' yang berisi nilai id dari dosen yang sedang diakses.

Silahkan tambah route berikut ke file routes\web.php:

routes\web.php

```

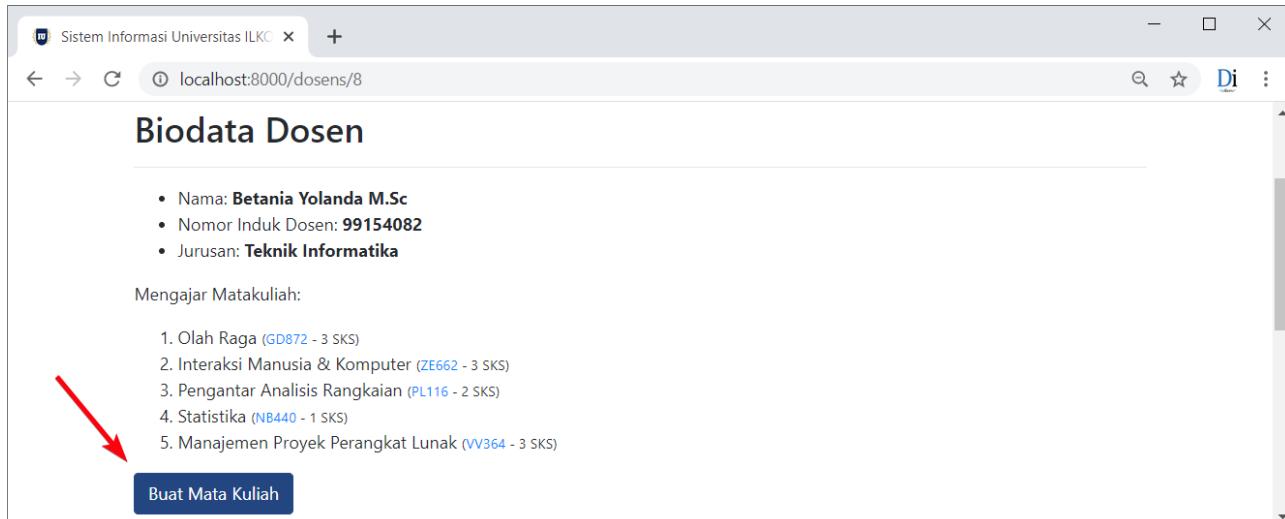
1 ...
2 Route::get('/buat-matakuliah/{dosen}', [MatakuliahController::class,
3           'buatMatakuliah'])->name('buat-matakuliah');

```

Route ini akan di proses oleh method `buatMatakuliah()` di file `MatakuliahController.php`. Perhatikan bahwa variabel 'dosen' dari tombol "Buat Mata Kuliah" diteruskan sebagai route

parameter `{dosen}`. Di dalam controller, parameter ini bisa kita akses untuk diolah lebih lanjut.

Karena named route sudah di definisikan, silahkan akses kembali halaman biodata dosen. Tombol "Buat Matakuliah" akan terlihat di bagian bawah:



Gambar: Tombol "Buat Mata Kuliah" di halaman biodata dosen

Masuk ke method `buatMatakuliah()` di file `MatakuliahController.php`, silahkan tambah method berikut di bagian bawah file controller:

`app\Http\Controllers\MatakuliahController.php`

```

1 public function buatMatakuliah(Dosen $dosen)
2 {
3     $jurusans = Jurusan::orderBy('nama')->get();
4     return view('matakuliah.create', [
5         'dosen' => $dosen,
6         'jurusans' => $jurusans,
7     ]);
8 }
```

Di baris 1, method `buatMatakuliah()` menerima satu argument `$dosen` yang berasal dari *route parameter*. Argument ini berisi nomor id dosen seperti 1, 3 atau 8. Namun karena ditulis dalam bentuk `(Dosen $dosen)`, itu akan mengaktifkan *route model binding*, sehingga variabel `$dosen` akan langsung berisi satu model Dosen pemilik id tersebut.

Kemudian di baris 3 saya mengambil semua data jurusan yang hasilnya di simpan ke variabel `$jurusans`. Data ini akan dipakai untuk membuat daftar pilihan jurusan pada form nantinya.

Variabel `$dosen` dan `$jurusans` lalu di kirim ke view '`matakuliah.create`' melalui perintah di baris 4 – 6.

Tapi tunggu dulu, apakah view '`matakuliah.create`' ini merupakan file yang sama dengan form tambah mata kuliah?

Betul, saya memutuskan untuk memakai ulang form yang sama, yakni file `matakuliah\`

`create.blade.php`. Ini agar kode program kita lebih efisien karena tidak perlu membuat dua form terpisah.

Namun konsekuensinya, file form menjadi lebih kompleks karena harus ada "trik" agar nama dosen menjadi statis (tidak lagi berbentuk menu dropdown). Berikut perubahan yang dimaksud:

Tambah Mata Kuliah

Kode Mata Kuliah	5 kode mata kuliah, contoh: IX264
Nama Mata Kuliah	<input type="text"/>
Jurusan	Ilmu Komputer
Dosen Pengajar	Betania Yolanda M.Sc
Jumlah SKS	1

Tambah Mata Kuliah

Kode Mata Kuliah	5 kode mata kuliah, contoh: IX264
Nama Mata Kuliah	<input type="text"/>
Jurusan	Ilmu Komputer
Dosen Pengajar	Betania Yolanda M.Sc
Jumlah SKS	1

Gambar: Mengubah tag `<select>` menjadi teks biasa

Yang perlu di pikirkan adalah, bagaimana cara mengubah pilihan **Dosen Pengajar** dari sebelumnya berbentuk menu dropdown saat form diakses dari halaman index dosen, menjadi bentuk teks saat form diakses dari halaman show dosen (halaman biodata).

Ini bisa dilakukan dengan memeriksa ada atau tidaknya variabel `$dosen`.

Ketika form diakses dari halaman index, method `create()` di `DosenController` hanya mengirim variabel `$jurusans` saja. Sedangkan jika form diakses dari halaman show, method `buatMatakuliah()` akan mengirim variabel `$jurusans` dan juga variabel `$dosen`.

Dengan logika ini, silahkan modifikasi bagian tag `<select>` pilihan "Dosen Pengajar" menjadi sebagai berikut:

`resources\views\dosen\form.blade.php`

```

1 ...
2 <div class="form-group row">
3   <label for="dosen_id" class="col-md-3 col-form-label text-md-right">
4     Dosen Pengajar </label>
5
6   {{-- Pemeriksaan kondisi agar pilihan dosen tidak bisa diubah,
7     Ini akan aktif jika form diakses dari dosens.show / halaman biodata --}}
8   @if (isset($dosen))
9     <div class="col-md-6 d-flex align-items-center">
10       <div>{{ $dosen->nama }}</div>
11     </div>
12   {{-- Kirim id dosen ke form awal agar tidak bermasalah dengan validasi --}}
13   <input type="hidden" name="dosen_id" id="dosen_id" value="{{ $dosen->id }}>
14 @else
15   {{-- Aktif jika form diakses dari dosens.index / halaman index --}}
16   <div class="col-md-6">
17     <select name="dosen_id" id="dosen_id">

```

```

18     class="custom-select col-md-5 @error('dosen_id') is-invalid @enderror">
19     @foreach ($dosens as $dosen)
20         @if($dosen->id == (old('dosen_id') ?? $matakuliah->dosen->id ?? ''))
21             <option value="{{ $dosen->id }}" selected>{{ $dosen->nama }}</option>
22         @else
23             <option value="{{ $dosen->id }}>{{ $dosen->nama }}</option>
24         @endif
25     @endforeach
26     </select>
27     @error('dosen_id')
28         <span class="invalid-feedback" role="alert">
29             <strong>{{ $message }}</strong>
30         </span>
31     @enderror
32     </div>
33     @endif
34 </div>
35 ...

```

File yang perlu kita modifikasi adalah `matakuliah\form.blade.php`, yakni view yang berisi struktur form. File `matakuliah\create.blade.php` sendiri tidak perlu di utak-atik.

Pada kode di atas, saya memindahkan isi tag `<select>` ke dalam percabangan `if-else` blade. Perintah `@if (isset($dosen))` di baris 8 akan memeriksa kehadiran variabel `$dosen`. Jika ada, yang artinya form diakses dari halaman show, maka tampilkan nama dosen sebagai teks dengan perintah `{{ $dosen->nama}}` di baris 10.

Juga agar nilai nama dosen ini sampai ke pemrosesan form, saya membuat tag `<input>` tipe `hidden` dengan atribut `name="dosen_id"` dan `value="{{ $dosen->id}}"` di baris 13. Dengan demikian, proses validasi tetap akan berjalan sebagaimana biasa.

Dan jika perintah `@if (isset($dosen))` menghasilkan nilai **false**, yang berarti form diakses dari halaman index, blok `@else` di baris 16 – 32 akan dijalankan, dimana inputan "Dosen Pengajar" tampil dalam bentuk menu dropdown.

Dengan tambahan ini, tombol "Buat Matakuliah" di halaman show dosen sudah bisa dipakai. Saat tombol di klik, akan tampil form tambah matakuliah dengan nama dosen sudah langsung berisi.

23.7. Mengambil Matakuliah untuk Mahasiswa

Kita sudah memiliki form untuk menambah data mahasiswa dan data matakuliah. Akan tetapi belum ada form untuk menghubungkan keduanya. Inilah yang akan kita buat kali ini, yaitu membuat form agar mahasiswa bisa mengambil matakuliah.

Untuk mengakses form ini, saya ingin menambah satu tombol "**Ambil Mata Kuliah**" ke halaman show mahasiswa. Silahkan buka file `mahasiswa\show.blade.php`, lalu ketik kode berikut di bagian bawah:

resources\views\mahasiswa\show.blade.php

```
1 ...  
2 @endforeach  
3 </ol>  
4  
5 @auth  
6   <a href="{{ route('ambil-matakuliah', ['mahasiswa' => $mahasiswa->id])}}"  
7     class="btn btn-info" title="Daftarkan Mata Kuliah">Ambil Mata Kuliah</a>  
8 @endauth  
9  
10 @endsection
```

Ketika di klik, tombol "Ambil Mata Kuliah" akan mengakses *named route* 'ambil-matakuliah' serta mengirim route parameter 'mahasiswa'. Parameter 'mahasiswa' ini berisi nomor id dari mahasiswa yang sedang diakses.

Lanjut, silahkan buka file routes\web.php dan tambah 2 route berikut:

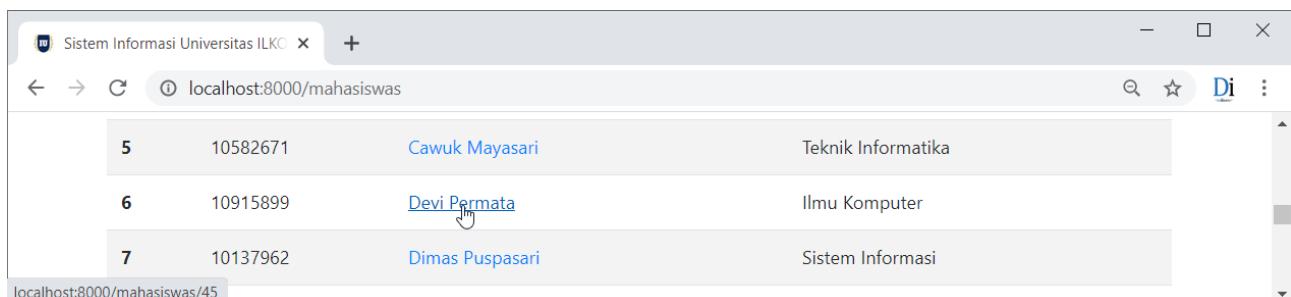
routes\web.php

```
1 ...  
2 Route::get('/mahasiswas/ambil-matakuliah/{mahasiswa}',  
3             [MahasiswaController::class, 'ambilMatakuliah'])  
4             ->name('ambil-matakuliah');  
5  
6 Route::post('/mahasiswas/ambil-matakuliah/{mahasiswa}',  
7                 [MahasiswaController::class, 'prosesAmbilMatakuliah'])  
8                 ->name('proses-ambil-matakuliah');
```

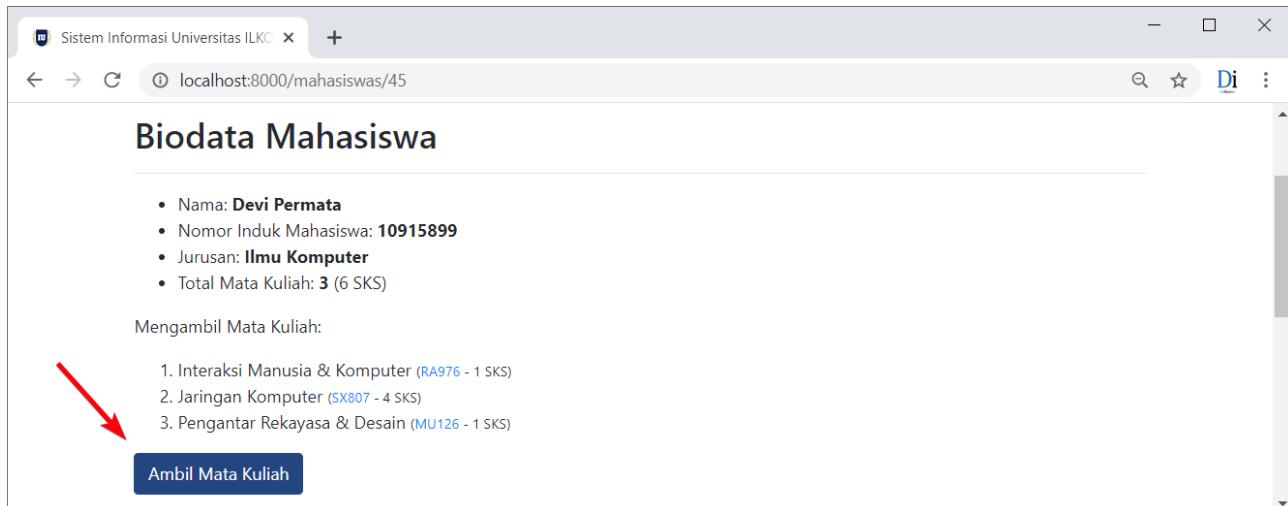
Route pertama ditujukan saat tombol "Ambil Mata Kuliah" di klik. Kode program untuk route akan berada di method `ambilMatakuliah()` pada file `MahasiswaController.php`. Tidak lupa, route ini juga meneruskan parameter `{mahasiswa}`.

Route kedua saya siapkan untuk memproses form (proses input data ke database). Meskipun memiliki alamat URL yang sama dengan route pertama, route kedua ini menerima data melalui jalur POST.

Save file routes\web.php, dan akses halaman show mahasiswa di web browser. Halaman show mahasiswa ini bisa diakses dari URL `localhost:8000/mahasiswas/<nomor_id>`. Atau melalui link dari salah satu nama mahasiswa:



Gambar: Mengakses halaman show dari link nama mahasiswa



Gambar: Tombol "Ambil Mata Kuliah"

Hasilnya, terlihat tombol "Ambil Mata Kuliah" di bagian bawah halaman. Pada saat tombol ini diklik, akan di proses oleh method `ambilMatakuliah()` di file `MahasiswaController.php`, yakni sesuai dengan *named route* '`ambil-matakuliah`' :

`app\Http\Controllers\MahasiswaController.php`

```

1  <?php
2  ...
3  use App\Models\Matakuliah;
4  ...
5  public function ambilMatakuliah(Mahasiswa $mahasiswa)
6  {
7      // Ambil semua daftar mata kuliah dari jurusan yang sama dengan mahasiswa
8      $matakuliah = Matakuliah::where('jurusan_id', $mahasiswa->jurusan_id)
9          ->orderBy('nama')->get();
10
11     // Buat array dari daftar jurusan yang sudah di ambil mahasiswa
12     // Ini akan dipakai untuk proses repopulate form
13     $matakuliah_sudah_diambil = $mahasiswa->matakuliah->pluck('id')->all();
14
15     return view('mahasiswa.ambil-matakuliah',
16     [
17         'mahasiswa' => $mahasiswa,
18         'matakuliah' => $matakuliah,
19         'matakuliah_sudah_diambil' => $matakuliah_sudah_diambil,
20     ]);
21 }
```

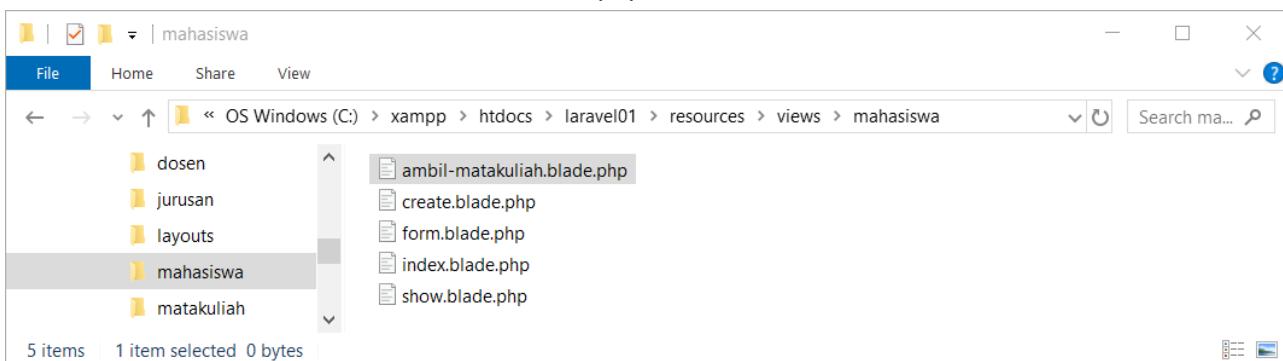
Di baris 5, method `ambilMatakuliah()` menerima argument `$mahasiswa` yang berisi angka id seperti 1, 9 atau 45. Dengan *route model binding*, isi variabel `$mahasiswa` akan diganti Laravel dengan satu object model `Mahasiswa` dari id yang sama.

Lanjut di baris 8, saya mengambil semua matakuliah dari jurusan yang sama dengan mahasiswa saat ini. Jika si mahasiswa berasal dari jurusan Ilmu Komputer, maka variabel

\$matakuliah akan berisi collection dari semua mata kuliah di jurusan Ilmu Komputer. Data ini nantinya akan kita pakai untuk membuat checkbox pilihan mata kuliah.

Perintah di baris 13 akan mencari mata kuliah yang sudah diambil oleh si mahasiswa. Data yang dimaksud sebenarnya ada di dalam tabel `mahasiswa_matakuliah`, oleh karena itu kita butuh perintah eloquent relationship `$mahasiswa->matakuliah`. Method `pluck('id')` saya pakai untuk mengambil kolom `id` saja. Hasilnya, variabel `$matakuliah_sudah_diambil` akan berisi array dari nomor id matakuliah.

Terakhir, variabel `$mahasiswa`, `$matakuliah` dan `$matakuliah_sudah_diambil` di kirim ke view 'mahasiswa.ambil-matakuliah'. File view ini masih belum tersedia, oleh karena itu silahkan buat file `ambil-matakuliah.blade.php` di dalam folder `resources\views\mahasiswa`:



Gambar: Buat file ambil-matakuliah.blade.php

Sesampainya di file `ambil-matakuliah.blade.php`, sebaiknya langsung dump file `$mahasiswa`, `$matakuliah` dan `$matakuliah_sudah_diambil` untuk melihat apakah ketiganya sudah berisi nilai yang diinginkan. Namun untuk menyingkat bahasan, langsung saja kita masuk ke kode lengkap view `ambil-matakuliah.blade.php`:

```
resources\views\mahasiswa\ambil-matakuliah.blade.php

1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Ambil Mata Kuliah untuk Mahasiswa</h1>
6 </div>
7 <hr>
8 <ul>
9   <li>Nama: <strong>{{$mahasiswa->nama}} </strong></li>
10  <li>Nomor Induk Mahasiswa: <strong>{{$mahasiswa->nim}} </strong></li>
11  <li>Jurusan: <strong>{{$mahasiswa->jurusan->nama}} </strong></li>
12  <li>Total Mata Kuliah:
13    <strong> {{ $mahasiswa->matakuliah->count() }} {{ $mahasiswa->matakuliah->sum('jumlah_sks') }} sks </strong>
14  </li>
15 </ul>
16 <hr>
17 <h5 class="mt-5 mb-4">Daftar Mata Kuliah {{ $mahasiswa->jurusan->nama }}</h5>
```

```
19 yang diambil oleh {{ $mahasiswa->nama }}:</h5>
20
21 <form method="POST" action=
22 "{{ route('proses-ambil-matakuliah',['mahasiswa' => $mahasiswa->id]) }}">
23 @csrf
24
25 <div class="form-group row" >
26 @error('matakuliah.*')
27 <div class="invalid-feedback my-3 d-block ml-3">
28   <strong>Error: Pilihan mata kuliah ada yang berulang /
29   bukan dari jurusan {{ $mahasiswa->jurusan->nama }}!</strong>
30 </div>
31 @enderror
32 <div class="col-md-12" style="column-count: 3;">
33 @foreach ($matakuliah as $matakuliah)
34   <div class="custom-control custom-checkbox mb-2">
35     <input type="checkbox" class="custom-control-input" name="matakuliah[]"
36     value="{{ $matakuliah->id }}" id="matakuliah-{{ $matakuliah->id }}"
37     @if( in_array($matakuliah->id,(old('matakuliah') ??
38       $matakuliah->sudah_diambil?? [] )) )
39       checked
40     @endif
41   >
42   <label class="custom-control-label" for="matakuliah-{{ $matakuliah->id }}">
43   {{ $matakuliah->nama }}
44   <small>
45     (<a href="{{ route('matakuliah.show',
46       ['matakuliah'=>$matakuliah->id]) }}>{{ $matakuliah->kode }}</a>)
47   </small>
48   </label>
49   </div>
50 @endforeach
51 </div>
52 </div>
53
54 <div class="form-group row mt-4">
55   <div class="col-md-6">
56     <button type="submit" class="btn btn-primary">Daftarkan</button>
57   </div>
58 </div>
59
60 </form>
61
62 @endsection
```

Ambil Mata Kuliah untuk Mahasiswa

- Nama: **Devi Permata**
- Nomor Induk Mahasiswa: **10915899**
- Jurusan: **Ilmu Komputer**
- Total Mata Kuliah: **3 (6 sks)**

Daftar Mata Kuliah Ilmu Komputer yang diambil oleh Devi Permata:

<input type="checkbox"/> Bahasa Inggris (X1822)	<input type="checkbox"/> Konstruksi & Pengujian Perangkat Lunak (AT265)	<input type="checkbox"/> Tata Kelola TI (QP058)
<input checked="" type="checkbox"/> Interaksi Manusia & Komputer (RA976)	<input type="checkbox"/> Pengantar Analisis Rangkaian (DG622)	<input type="checkbox"/> Teori Bahasa Formal dan Otomata (GS219)
<input checked="" type="checkbox"/> Jaringan Komputer (SX807)	<input checked="" type="checkbox"/> Pengantar Rekayasa & Desain (MU126)	<input type="checkbox"/> Wawasan Teknologi & Komunikasi Ilmiah (AL147)
<input type="checkbox"/> Kepemimpinan & Ketrampilan Interpersonal (QY210)	<input type="checkbox"/> Statistika (QU425)	<input type="checkbox"/> Strategi Algoritma (OK248)

Daftarkan

Gambar: Form Ambil Mata Kuliah untuk mahasiswa dengan id 45

Agar lebih mudah dipahami, silahkan pelajari dengan melihat tampilan akhir form.

Di bagian awal terdapat biodata mahasiswa. Data ini sudah tersedia dalam variabel \$mahasiswa dan tinggal di echo seperti {{\$mahasiswa->nama}} atau menggunakan perintah relationship seperti {{\$mahasiswa->jurusan->nama}}. Tampilan ini pada dasarnya sama dengan isi halaman show mahasiswa.

Kemudian terdapat list mata kuliah yang terdiri dari banyak checkbox. Semua checkbox ini berada dalam tag <form> antara baris 21 – 60.

Beberapa checkbox sudah langsung terpilih sebagai tanda bahwa mata kuliah tersebut sudah diambil oleh si mahasiswa. Inilah bagian yang cukup sulit dari kode program kita.

Data semua mata kuliah sudah tersedia dalam variabel \$matakuliahs, sehingga bisa di proses dengan perulangan foreach antara baris 33 – 50. Dalam setiap iterasi, tag <input type="checkbox"> akan terus di tampilkan sejumlah mata kuliah yang ada. Jika total terdapat 30 mata kuliah, maka akan tampil 30 tombol checkbox.

Untuk atribut name, saya tulis dengan perintah name="matakuliah[]". Ini akan mengelompokkan seluruh inputan checkbox menjadi array. Sedangkan untuk atribut value, diambil dari nilai id mata kuliah, yakni dengan perintah value="{{\$matakuliah->id}}" seperti di baris 36.

Kondisi if pada baris 37 dipakai untuk menentukan apakah checkbox terpilih atau tidak. Jika

hasilnya **true**, maka tambah atribut checked ke dalam tag <input>:

```
@if(in_array($matakuliah->id,(old('matakuliah')?? $matakuliah_sudah_diambil??[])))
    checked
@endif
```

Kondisi if ini cukup panjang karena sekaligus saya pakai untuk penanganan error. Agar lebih mudah dipahami, untuk sementara bisa di sederhanakan menjadi berikut:

```
@if(in_array($matakuliah->id,($matakuliah_sudah_diambil)))
    checked
@endif
```

Pada dasarnya, kondisi if berpatokan pada hasil function `in_array()` bawaan PHP. Function `in_array()` dipakai untuk memeriksa apakah sebuah element ada di dalam array atau tidak. Hasilnya **true** jika ditemukan, atau **false** jika element tidak ditemukan.

Function `in_array()` butuh 2 buah argument. Argument pertama berupa element yang akan dicari, serta argument kedua berupa array tempat pencarian.

Dari method `ambilMatakuliah()` di controller sebelumnya, saya mengirim variabel `$matakuliah_sudah_diambil` ke dalam view. Sesuai namanya, variabel ini berisi array dari id mata kuliah yang sudah diambil oleh si mahasiswa.

Sebagai contoh, misalkan variabel `$matakuliah_sudah_diambil` berisi array [13, 23, 32, 8]. Maka function `in_array($matakuliah->id,($matakuliah_sudah_diambil))` akan bernilai **true** jika variabel `$matakuliah->id` sampai di angka 13, 23, 32 dan 8. Jika ini terjadi, tampilkan atribut checked ke dalam tag <input>. Inilah yang membuat beberapa checkbox langsung terpilih saat halaman di load.

Perintah `old('matakuliah')` dipakai untuk proses re-populate checkbox jika terjadi error validasi. Sedangkan tambahan array kosong [] pada akhir baris dipakai jika ternyata si mahasiswa belum memilih satu pun mata kuliah. Untuk kondisi seperti ini, variabel `$matakuliah_sudah_diambil` akan berisi nilai NULL dan bisa menyebabkan error.

Kondisi `@if` di atas tidak saya tulis sekali jalan, tapi sudah melalui berbagai percobaan dan utak-atik kode program.

Konsep logika yang kuat dan pemahaman dasar algoritma akan sangat membantu mencari solusi dari masalah programming seperti ini.

Banyak latihan studi kasus juga akan membantu. Jika merasa lemah di logika, sering-seringlah buat studi kasus. Biasanya kasus programming ini akan berulang dan jika kita punya pengalaman yang mirip, akan lebih mudah mencari solusinya.

Pesan error form saya tempatkan di bagian atas, yakni di baris 26 – 31. Di sini terdapat teknik baru penanganan error, yakni dengan perintah `@error('matakuliah.*')`.

Tanda bintang '*' merujuk ke semua inputan mata kuliah. Seperti yang terlihat, di halaman ini kita punya banyak tag <input> dengan atribut name berisi array matakuliah[]. Jadi semua tag input pada dasarnya sudah terkelompok dengan nama yang sama.

Daripada menulis pesan error untuk setiap inputan, lebih ringkas jika tampilkan satu pesan error saja. Dalam kode ini, jika terjadi error di satu checkbox, teks di baris 28 – 29 akan ditampilkan.

Terakhir, terdapat tombol submit "Daftarkan" di baris 56. Pada saat tombol ini diklik, akan menuju ke *named route* 'proses-ambil-matakuliah'.

Route ini sudah kita tulis sebelumnya, dimana kode program pemrosesan form berada di method prosesAmbilMatakuliah() di MahasiswaController. Berikut kode yang diperlukan:

app\Http\Controllers\MahasiswaController.php

```

1  public function prosesAmbilMatakuliah(Request $request, Mahasiswa $mahasiswa)
2  {
3      // Ambil semua id matakuliah untuk jurusan yang sama dengan mahasiswa.
4      $matakuliah_jurusan=Matakuliah::where('jurusan_id',$mahasiswa->jurusan_id)
5          ->pluck('id')->toArray();
6
7      $validateData = $request->validate([
8          'matakuliah.*' => 'distinct|in:' . implode(', ', $matakuliah_jurusan),
9      ]);
10
11     // Input ke database
12     $mahasiswa->matakuliahs()->sync($validateData['matakuliah'] ?? []);
13
14     Alert::success('Berhasil', "Terdapat ".
15                     count($validateData['matakuliah']) ?? []).
16                     " mata kuliah yang diambil $mahasiswa->nama");
17
18     return redirect(route('mahasiswas.show',
19                     ['mahasiswa' => $mahasiswa->id]));
20 }
```

Di baris 1, method prosesAmbilMatakuliah() menerima dua buah argument, yakni \$request yang berisi data request dari form (termasuk nilai form), serta \$mahasiswa yang sudah berisi satu object mahasiswa (dari *route model binding*).

Pada awal kode program, variabel \$matakuliah_jurusan saya isi dengan semua id mata kuliah untuk jurusan yang sama dengan si mahasiswa. Hasil akhir akan berbentuk array nomor id matakuliah seperti [4, 23, 43, 12]. Ini akan dipakai untuk proses validasi.

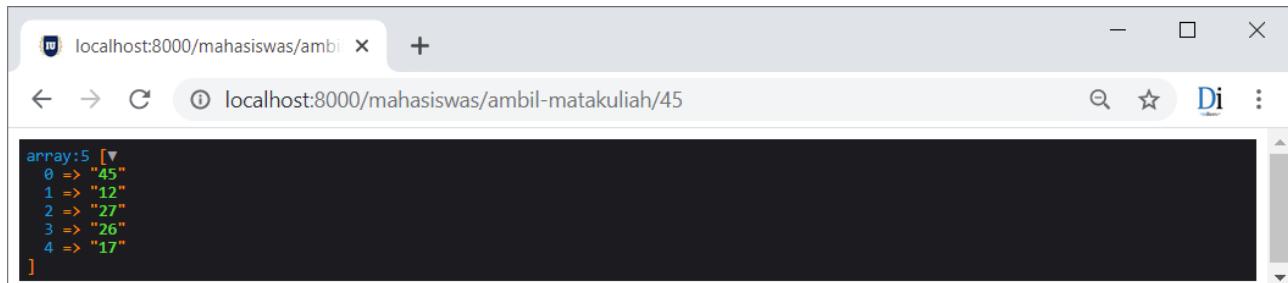
Proses validasi sendiri ada di baris 7 – 9. Syarat validasi ini belum pernah kita pakai sebelumnya.

Dari dalam form, semua inputan checkbox berisi atribut name=matakuliah[]. Sesampainya di method ini, data setiap inputan form akan tersimpan sebagai array.

Untuk uji coba, silahkan tambah perintah `dd($request->matakuliah)` di awal method `prosesAmbilMatakuliah()`, kira-kira seperti ini:

```
public function prosesAmbilMatakuliah(Request $request, Mahasiswa $mahasiswa)
{
    dd($request->matakuliah);
    // ...
}
```

Silahkan buka form 'ambil-matakuliah', pilih beberapa checkbox, lalu submit. Hasilnya adalah sebagai berikut:



Gambar: Hasil dari perintah `dd($request->matakuliah)`

Karena berbentuk array, syarat validasi saya tulis sebagai berikut:

```
'matakuliah.*' => 'distinct|in:'.$matakuliah_jurusan),
```

Tanda bintang '*' artinya saya ingin melakukan proses validasi untuk semua isi array 'matakuliah'. Untuk setiap element array tersebut, pastikan tidak ada yang sama (`distinct`), serta ada di dalam daftar array `$matakuliah_jurusan` (berasal dari syarat `in`).

Penggunaan syarat 'in' sebenarnya mirip seperti function `in_array()`, tapi harus ditulis dalam bentuk string. Oleh karena itulah perlu dikonversi menggunakan function `implode()` bawaan PHP.

Sebagai contoh, jika menggunakan cara manual, syarat `in` bisa ditulis seperti ini :

```
'matakuliah.*' => 'distinct|in:4,23,43,12',
```

Syarat validasi di atas akan lolos jika semua element di dalam array `matakuliah.*` ada di antara angka 4, 23, 43 dan 12.

Ilustrasi yang lebih nyata adalah sebagai berikut: Misalkan kita sedang mengambil mata kuliah untuk mahasiswa 'Devi' yang berasal dari jurusan Teknik Informatika. Saat di form 'ambil-matakuliah', Devi memilih 3 mata kuliah yang memiliki id 5, 8 dan 11.

Ketika di submit, variabel `$request->matakuliah` akan berisi 3 element array, yakni [5, 8, 11]. Bagaimana kita memastikan bahwa id matakuliah ini memang ada di jurusan Teknik Informatika?

Caranya, ambil semua id mata kuliah untuk jurusan Teknik Informatika. Inilah yang dilakukan oleh perintah di baris 4 – 5 , dimana variabel \$matakuliah_jurusan akan berisi daftar id seluruh mata kuliah di jurusan Teknik Informatika, yakni [3, 5, 8, 9, 11, 23, 24, 31].

Sesampainya di syarat validasi, yang akan diperiksa adalah sebagai berikut:

```
'matakuliah.*' => 'distinct|in:3,5,8,9,11,23,24,31',
```

Karena id mata kuliah yang dipilih semuanya ada di dalam syarat 'in', maka akan lolos validasi. Namun jika data variabel \$request->matakuliah ternyata berisi [5, 8, 17], itu tidak akan lolos karena id '17' tidak ada di daftar id mata kuliah jurusan Teknik Informatika.

Huff... validasi yang cukup 'njelimet'.

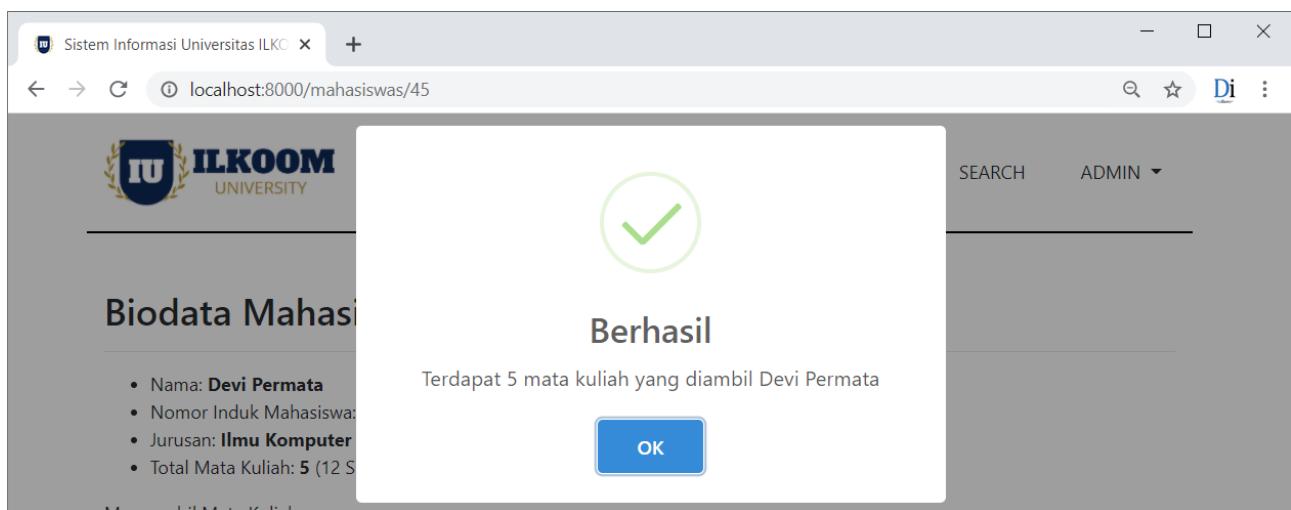
Jika akhirnya validasi lolos, data tersebut sudah bisa kita input ke database. Caranya dengan method sync() seperti di baris 12:

```
$mahasiswa->matakuliah->sync($validateData[ 'matakuliah' ] ?? []);
```

Method sync() dipakai karena hubungan tabel mahasiswas dan matakuliah adalah *many to many*. Tambahan perintah '?? []' di akhir saya pakai untuk mencegah error seandainya user tidak mengisi satu pun mata kuliah. Jika ini yang terjadi, variabel \$validateData ['matakuliah'] akan berisi null dan bisa menyebabkan error.

Setelah proses input ke database, buat pesan alert di baris 14, lalu me-redirect user ke halaman show mahasiswa di baris 15.

Saatnya uji coba. Silahkan akses salah satu mahasiswa, klik tombol 'Ambil Mata Kuliah', ceklist beberapa mata kuliah, lalu submit:



Gambar: Pesan alert berhasil mengambil mata kuliah

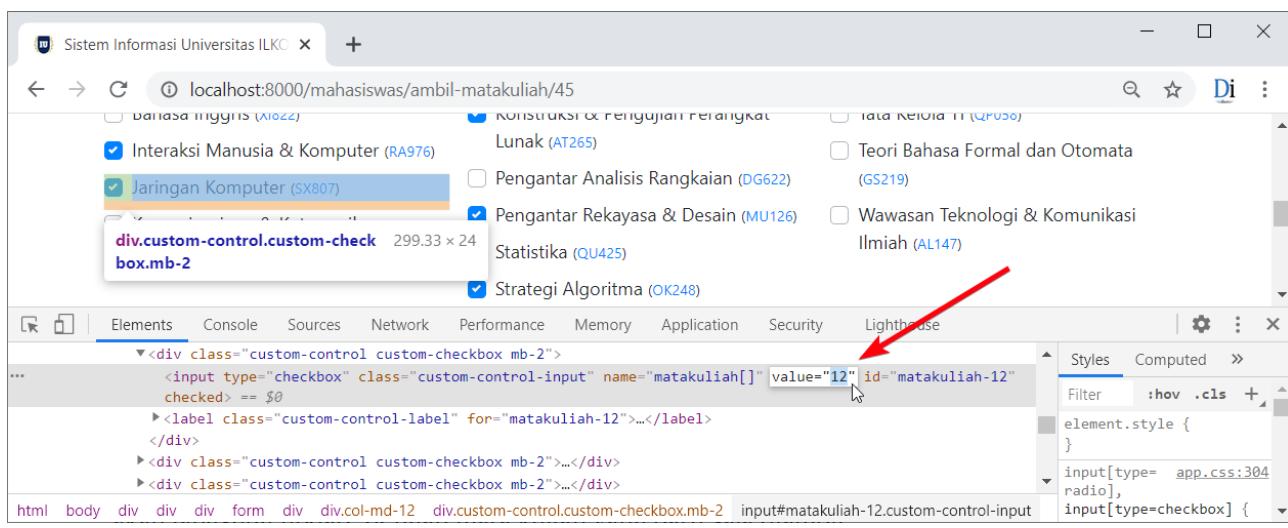
Done! Proses pengambilan mata kuliah sukses di lakukan. Isi halaman biodata mahasiswa juga akan langsung update dengan mata kuliah yang baru saja diambil.

Sedikit pertanyaan, bagaimana cara menguji proses validasi?

Semua checkbox di halaman 'ambil mahasiswa' di generate langsung dari kode program. Di sini user tidak bisa membuat mata kuliah baru, maka seharusnya id mata kuliah yang dipilih sudah pasti hanya untuk jurusan itu saja. Jadi bagaimana bisa terjadi error?

Betul, untuk kebanyakan kasus tidak akan terjadi error karena pilihan daftar matakuliah sudah fixed dan tidak bisa diubah. Kecuali, si user "iseng" mengedit langsung kode HTML dari menu web developer tools / inspect element.

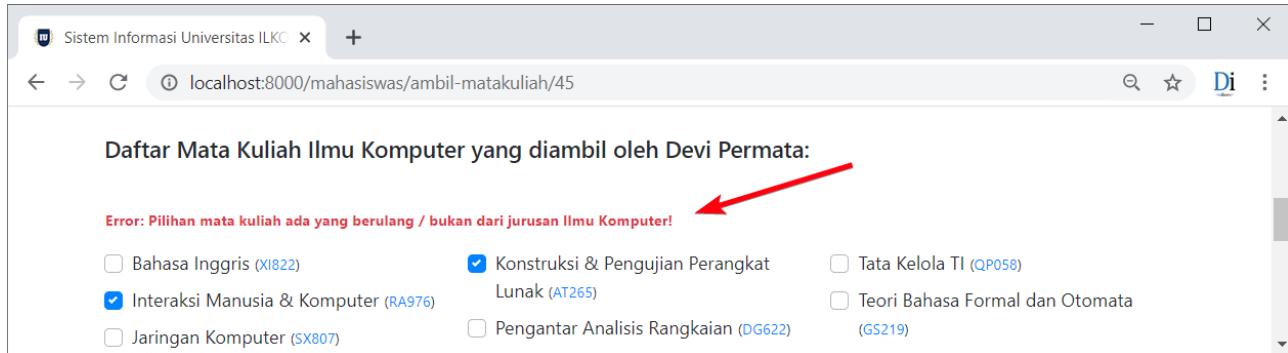
Sebagai contoh, dalam tampilan gambar berikut saya bisa mengubah id mata kuliah "Jaringan Komputer" dari sebelumnya 12 menjadi nilai lain, misalkan 13:



Gambar: Modifikasi form dengan web developer tool

Setelah diubah dan form di submit, maka data yang sampai ke method `prosesAmbilMatakuliah()` adalah id 13 yang sudah diubah tadi. Ini bisa menjadi masalah karena id 13 bisa jadi tidak terdapat di jurusan tersebut.

Namun karena kita sudah membuat syarat validasi untuk memeriksa kemungkinan ini, akan tampil pesan error di halaman form:



Gambar: Error validasi form

Untuk aplikasi sederhana seperti untuk tugas / skripsi, penanganan error sampai ke tahap ini

mungkin tidak perlu. Tapi untuk aplikasi "real world", kita harus selalu waspada terhadap segala kemungkinan. Semua kode front-end (HTML, CSS dan JavaScript) bisa diubah oleh user, sehingga validasi ulang di back-end (PHP) tetap menjadi kunci dari keamanan form.

23.8. Mendaftarkan Mahasiswa untuk Mata Kuliah

Form yang akan kita buat kali ini merupakan kebalikan dari praktek sebelumnya. Sekarang giliran mahasiswa yang di daftarkan dari mata kuliah. Konsep yang dipakai juga sama, hanya saja akan berangkat dari halaman show mata kuliah.

Pertama, saya akan tambah tombol "**Daftarkan Mahasiswa**" ke file `matakuliah\show.blade.php`. Silahkan ketik kode berikut di bagian bawah:

`resources\views\matakuliah\show.blade.php`

```

1   ...
2   @endforeach
3 </ol>
4
5 @auth
6   <a href="{{ route('daftarkan-mahasiswa',[ 'matakuliah' => $matakuliah->id])}}"
7     class="btn btn-info" title="Daftarkan Mahasiswa">Daftarkan Mahasiswa</a>
8 @endauth
9
10 @endsection

```

Ketika di klik, tombol "Daftarkan Mahasiswa" akan mengakses named route 'daftarkan-mahasiswa' serta mengirim route parameter 'matakuliah'. Parameter 'matakuliah' ini berisi nomor id dari mata kuliah yang sedang diakses.

Lanjut, silahkan buka file `routes\web.php` dan tambah 2 route berikut:

`routes\web.php`

```

1 ...
2 Route::get('/matakuliah/daftarkan-mahasiswa/{matakuliah}', [
3   MatakuliahController::class, 'daftarkanMahasiswa'])
4   ->name('daftarkan-mahasiswa');
5
6 Route::post('/matakuliah/daftarkan-mahasiswa/{matakuliah}', [
7   MatakuliahController::class, 'prosesDaftarkanMahasiswa'])
8   ->name('proses-daftarkan-mahasiswa');

```

Kedua route berfungsi sama seperti di praktek sebelumnya. Route pertama di pakai untuk menampilkan form pada saat tombol "Daftarkan Mahasiswa" di klik. Sedangkan route kedua di pakai untuk pemrosesan form.

Save file `routes\web.php`, dan akses halaman show matakuliah di web browser. Halaman show matakuliah ini bisa diakses dari URL `localhost:8000/matakuliah/<nomor_id>`. Atau melalui

link dari salah satu nama matakuliah:

3	XI822	Bahasa Inggris	Edward Prabowo M.Sc	2	Ilmu Komputer
4	ZW150	Basis Data	Darmanto Agustina M.T.	2	Teknik Informatika
5	QB773	Dasar Pemrograman	Viktor Pranowo M.Sc	1	Sistem Informasi

Gambar: Mengakses halaman show dari link nama mata kuliah

Informasi Mata Kuliah					
• Nama: Basis Data					
• Kode Mata Kuliah: ZW150					
• Dosen Pengajar: Darmanto Agustina M.T.					
• Jurusan: Teknik Informatika					
• Jumlah SKS: 2					
• Total Mahasiswa: 3					
Daftar Mahasiswa:					
1. Eva Permata (10120628)					
2. Kawaca Pradipta (10696311)					
3. Makuta Tampubolon (10321980)					
Daftarkan Mahasiswa					

Gambar: Tombol "Daftarkan Mahasiswa"

Tombol "Daftarkan Mahasiswa" terlihat di bagian bawah halaman. Pada saat tombol ini di klik, akan di proses oleh method `daftarkanMahasiswa()` di file `MatakuliahController.php`, yakni sesuai dengan *named route* '`daftarkan-mahasiswa`' :

app\Http\Controllers\MatakuliahController.php

```

1 <?php
2 ...
3 use App\Models\Mahasiswa;
4 ...
5 public function daftarkanMahasiswa(Matakuliah $matakuliah)
6 {
7     // Ambil semua daftar mahasiswa dari jurusan yang sama dengan matakuliah
8     $mahasiswas = Mahasiswa::where('jurusan_id', $matakuliah->jurusan_id)
9         ->orderBy('nama')->get();
10
11    // Buat array dari daftar mahasiswa yang sudah terdaftar sebelumnya
12    // Ini akan dipakai untuk proses repopulate form
13    $mahasiswas_sudah_terdaftar = $matakuliah->mahasiswas->pluck('id')->all();
14    return view('matakuliah.daftarkan-mahasiswa',
15    [
16        'matakuliah' => $matakuliah,

```

```
17     'mahasiswa' => $mahasiswa,
18     'mahasiswa_sudah_terdaftar' => $mahasiswa_sudah_terdaftar,
19 ];
20 }
```

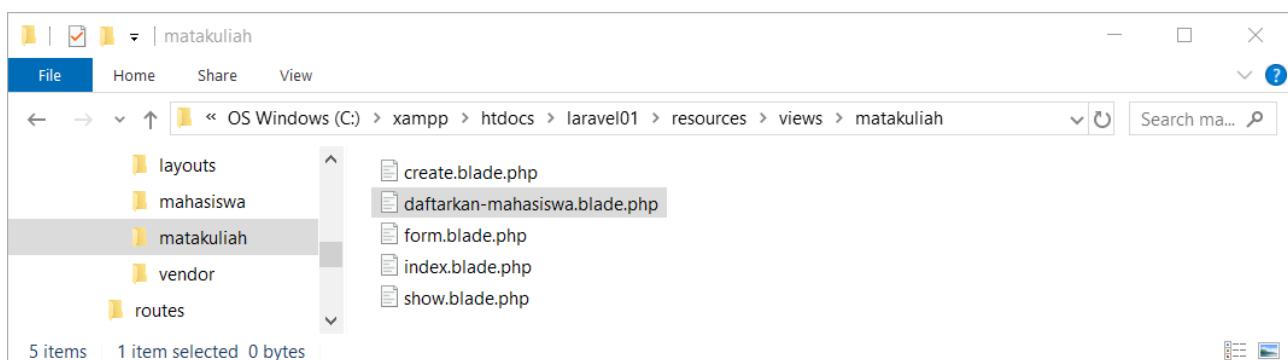
Kode ini juga mirip seperti di versi mahasiswa.

Di baris 5, method `daftarkanMahasiswa()` menerima argument `$matakuliah` yang berisi angka id seperti 1, 9 atau 45. Dengan *route model binding*, isi variabel `$matakuliah` akan diganti Laravel dengan satu object model Mahasiswa dari id yang sama.

Lanjut di baris 8, saya mengambil semua daftar mahasiswa dari jurusan yang sama dengan mata kuliah saat ini. Jika si mata kuliah berasal dari jurusan Ilmu Komputer, maka variabel `$mahasiswa` akan berisi collection dari semua mahasiswa di jurusan Ilmu Komputer. Data ini nantinya akan kita pakai untuk membuat checkbox pilihan mahasiswa.

Perintah di baris 13 akan mencari mahasiswa yang sudah terdaftar di matakuliah saat ini. Data yang dimaksud sebenarnya ada di dalam tabel `mahasiswa_matakuliah`, oleh karena itu kita butuh perintah eloquent relationship `$matakuliah->mahasiswa`. Method `pluck('id')` saya pakai untuk mengambil kolom `id` saja. Hasilnya, variabel `$mahasiswa_sudah_terdaftar` akan berisi array dari nomor id mahasiswa.

Terakhir, variabel `$matakuliah`, `$mahasiswa` dan `$mahasiswa_sudah_terdaftar` di kirim ke view '`matakuliah.daftarkan-mahasiswa`'. File view ini masih belum tersedia, oleh karena itu silahkan buat file `daftarkan-mahasiswa.blade.php` di dalam folder `resources\views\matakuliah`:



Gambar: Buat file `daftarkan-mahasiswa.blade.php`

Berikut isi dari file `daftarkan-mahasiswa.blade.php`:

`resources\views\mahasiswa\daftarkan-mahasiswa.blade.php`

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Daftarkan Mahasiswa ke Mata Kuliah </h1>
6 </div>
7 <hr>
```

```

8 <ul>
9   <li>Nama: <strong>{{$matakuliah->nama}} </strong></li>
10  <li>Kode Mata Kuliah: <strong>{{$matakuliah->kode}} </strong></li>
11  <li>Dosen Pengajar:
12    <strong>
13      <a href="{{ route('dosens.show',[ 'dosen' => $matakuliah->dosen->id]) }}">
14        {{$matakuliah->dosen->nama}}
15      </a>
16    </strong></li>
17  <li>Jurusan: <strong>{{$matakuliah->jurusan->nama}} </strong></li>
18  <li>Jumlah SKS: <strong>{{ $matakuliah->jumlah_sks }} </strong></li>
19  <li>Total Mahasiswa:
20    <strong> {{count($mahasiswas_sudah_terdaftar)}} </strong>
21  </li>
22 </ul>
23 <hr>
24 <h5 class="mt-5 mb-4">Daftar mahasiswa {{ $matakuliah->jurusan->nama }} yang mengambil mata kuliah {{ $matakuliah->nama }}:</h5>
25
26 <form method="POST" action=
27 "{{ route('proses-daftarkan-mahasiswa',[ 'matakuliah' => $matakuliah->id]) }}"
28 @csrf
29
30 <div class="form-group row" >
31   @error('mahasiswa.*')
32     <div class="invalid-feedback my-3 d-block ml-3">
33       <strong>Error: Pilihan mahasiswa ada yang berulang / bukan dari jurusan {{ $matakuliah->jurusan->nama }}!</strong>
34     </div>
35   @enderror
36 <div class="col-md-12" style="column-count: 3;">
37   @foreach ($mahasiswas as $mahasiswa)
38     <div class="custom-control custom-checkbox mb-2">
39       <input type="checkbox" class="custom-control-input" name="mahasiswa[]"
40         value="{{ $mahasiswa->id }}" id="mahasiswa-{{ $mahasiswa->id }}"
41       @if( in_array($mahasiswa->id,(old('mahasiswa') ??
42           $mahasiswas_sudah_terdaftar ?? [] )) )
43         checked
44       @endif
45     >
46     <label class="custom-control-label" for="mahasiswa-{{ $mahasiswa->id }}">
47       {{ $mahasiswa->nama }}
48       <small>
49         (<a href="{{route('mahasiswas.show',
50           [ 'mahasiswa'=>$mahasiswa->id])}}>{{ $mahasiswa->nim }}</a>
51         </small>
52       </label>
53     </div>
54   @endforeach
55 </div>
56 </div>
57 </div>
58 </div>
59
60 <div class="form-group row mt-4">
61   <div class="col-md-6">
62     <button type="submit" class="btn btn-primary">Daftarkan</button>

```

```
63    </div>
64 </div>
65
66 </form>
67
68 @endsection
```

The screenshot shows a web application interface for managing student enrollment. At the top, there's a header with the university logo 'ILKOOM UNIVERSITY' and navigation links for JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and ADMIN. Below the header, the main title is 'Daftarkan Mahasiswa ke Mata Kuliah'. A list of course details is provided:

- Nama: **Basis Data**
- Kode Mata Kuliah: **ZW150**
- Dosen Pengajar: **Darmanto Agustina M.T**
- Jurusan: **Teknik Informatika**
- Jumlah SKS: **2**
- Total Mahasiswa: **3**

Below this, a section titled 'Daftar mahasiswa Teknik Informatika yang mengambil mata kuliah Basis Data:' lists student names with checkboxes. Some checkboxes are checked, indicating they are already enrolled. The list includes:

- Cawuk Mayasari (10582671)
- Eja Nasyihah (10683796)
- Eva Permata (10120628)
- Garang Sudiati (10110155)
- Hafshah Zulkarnain (10607308)
- Halim Utami (10062575)
- Jail Manullang (10985559)
- Kawaca Pradipta (10696311)
- Lili Farida (10704948)
- Mahdi Widiasuti (10093326)
- Makuta Tampubolon (10321980)
- Purwa Nababan (10801162)
- Queen Putra (10240855)
- Saka Yulianti (10328857)
- Vinsen Lailasari (10263630)

A blue 'Daftarkan' button is located at the bottom left of the form.

Gambar: Form Daftarkan Mahasiswa untuk matakuliah dengan id 40

Teknik pada halaman ini juga mirip seperti di halaman Ambil Mata Kuliah.

Di bagian awal terdapat keterangan mata kuliah. Data ini sudah tersedia dalam variabel \$matakuliah dan tinggal di echo seperti {{\$matakuliah->nama}} atau menggunakan perintah relationship seperti {{\$matakuliah->jurusan->nama}}. Tampilan ini pada dasarnya sama dengan isi halaman show matakuliah.

Kemudian terdapat list mahasiswa yang terdiri dari banyak checkbox. Semua checkbox ini berada dalam tag <form> antara baris 27 – 66.

Beberapa checkbox sudah langsung terpilih sebagai tanda bahwa mahasiswa tersebut sudah terdaftar ke mata kuliah saat ini.

Data semua mahasiswa sudah tersedia dalam variabel \$mahasiswa, sehingga bisa di proses dengan perulangan foreach antara baris 39 – 56. Dalam setiap iterasi, tag <input type="checkbox"> akan terus di tampilkan sejumlah mahasiswa yang ada. Jika total terdapat 30

mahasiswa, maka akan tampil 30 tombol checkbox.

Untuk atribut name, saya tulis dengan perintah `name="mahasiswa[]"`. Ini akan mengelompokkan seluruh inputan checkbox menjadi array. Sedangkan untuk atribut value, diambil dari nilai id mahasiswa, yakni dengan perintah `value="{{\$mahasiswa->id}}"` seperti di baris 42.

Kondisi if pada baris 43 dipakai untuk menentukan apakah checkbox terpilih atau tidak. Jika hasilnya **true**, maka tambah atribut checked ke dalam tag `<input>`:

```
@if( in_array($mahasiswa->id,(old('mahasiswa') ??
    $mahasiswa_sudah_terdaftar ?? [] )) )
    checked
@endif
```

Penjelasan dari kondisi if ini tidak akan saya bahas lagi karena sangat mirip seperti di form Ambil Matakuliah. Pesan error form juga ada di bagian atas, yakni di baris 32 – 37.

Terakhir, terdapat tombol submit "Daftarkan" di baris 62. Pada saat tombol ini di klik, akan menuju ke *named route* 'proses-daftarkan-mahasiswa'.

Route ini sudah kita tulis sebelumnya, dimana kode program pemrosesan form berada di method `prosesDaftarkanMahasiswa()` di `MatakuliahController`. Berikut kode yang diperlukan:

app\Http\Controllers\MatakuliahController.php

```
1  public function prosesDaftarkanMahasiswa(Request $request,
2                                              Matakuliah $matakuliah)
3  {
4      // Ambil semua id mahasiswa untuk jurusan yang sama dengan mata kuliah.
5      $mahasiswa_jurusan=Mahasiswa::where('jurusan_id',$matakuliah->jurusan_id)
6          ->pluck('id')->toArray();
7
8      $validateData = $request->validate([
9          'mahasiswa.*' => 'distinct|in:' . implode(',', $mahasiswa_jurusan),
10     ]);
11
12     // Proses mahasiswa yang didaftarkan
13     $matakuliah->mahasiswa() -> sync($validateData['mahasiswa'] ?? []);
14
15     Alert::success('Berhasil', "Terdapat ".
16                     count($validateData['mahasiswa'] ?? []) .
17                     " mahasiswa yang mengambil $matakuliah->nama");
18
19     return redirect(route('matakuliah.show',
20                         ['matakuliah' => $matakuliah->id]));
21 }
```

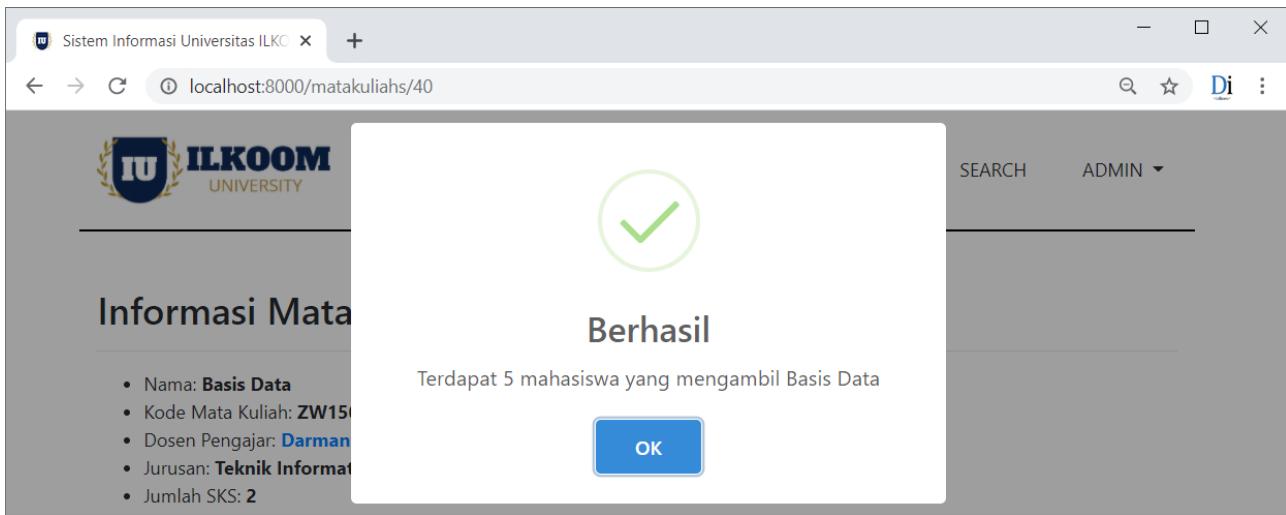
Kembali, cara yang saya pakai juga sama seperti di pemrosesan form Ambil Matakuliah.

Di baris 1, method `prosesDaftarkanMahasiswa()` menerima dua buah argument, yakni `$request` yang berisi data request dari form (termasuk nilai form), serta `$matakuliah` yang sudah berisi satu object matakuliah (dari *route model binding*).

Pada awal kode program, variabel `$mahasiswa_jurusan` di isi dengan semua id mahasiswa untuk jurusan yang sama dengan matakuliah. Ini menjadi bahan untuk proses validasi di baris 8 – 10. Teknik validasi ini juga sama seperti sebelumnya, hanya beda di nama variabel saja.

Jika data mahasiswa lolos lolos, akan diinput ke database menggunakan method `sync()` seperti di baris 13. Kemudian buat pesan alert di baris 15, lalu me-redirect user ke halaman show mahasiswa di baris 16.

Saatnya uji coba. Silahkan akses salah satu mata kuliah, klik tombol 'Daftarkan Mahasiswa', ceklist beberapa mahasiswa, lalu submit:



Gambar: Pesan alert berhasil mendaftarkan mahasiswa

Proses pendaftaran mahasiswas sukses di lakukan. Isi halaman informasi mata kuliah ini juga akan langsung update dengan mahasiswa yang baru saja diambil.

Pesan error seharusnya juga sudah tampil. Silahkan anda coba dengan teknik inspect element seperti di halaman ambil mata kuliah.

Sampai di sini kita sudah menyelesaikan salah satu bagian tersulit dari pembuatan CRUD, yakni proses pembuatan form untuk menambah data (**Create**) termasuk proses validasinya. Lanjuta kita akan masuk ke bagian Update dari aplikasi Sistem Informasi Universitas ILKOOOM.

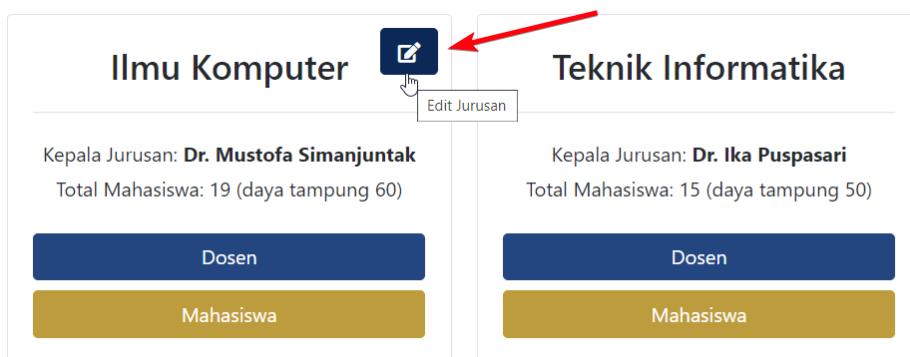
24. Sistem Informasi Universitas ILKOOM: Update

Melanjutkan project Universitas ILKOOM, kita akan rancang kode untuk proses **Update**, yakni membuat form untuk memodifikasi data.

Struktur form untuk proses update sebenarnya sangat mirip seperti create. Hanya saja ketika di update, form tersebut sudah langsung berisi data yang diambil dari database. Dalam bab sebelumnya kita sudah merancang form yang bisa digunakan ulang, sehingga akan memudahkan pembuatan form update.

24.1. Mengedit Data Jurusan

Langkah awal untuk membuat form update adalah menempatkan tombol "**Edit**" yang ketika diklik akan menuju halaman form. Untuk data jurusan, saya ingin memakai teknik yang mirip seperti studi kasus ILKOOM Profile Manager di akhir buku Laravel Uncover. Berikut tampilan yang dimaksud:



Gambar: Tombol "Edit Jurusan"

Tombol "Edit" ini berada di sudut kanan atas card jurusan. Secara default tombol akan tersembunyi dan baru muncul saat cursor mouse berada di dalam kotak card. Efek ini dibuat dari kode CSS dengan mengatur property `opacity`.

Silahkan buka file index jurusan di `jurusan\index.blade.php`, lalu tambah kode berikut ke dalam struktur Card:

```
resources\views\jurusan\index.blade.php
```

```
1 ...  
2 <div class="card-columns mt-3">
```

```

3  @foreach($jurusans as $jurusan)
4
5      <div class="card mt-1" id=card-{{ $jurusan->id }}>
6
7          @auth
8              <div class="btn-group btn-action">
9                  <a href="{{ route('jurusans.edit',['jurusan' => $jurusan->id])}">
10                     class="btn btn-primary d-inline-block" title="Edit Jurusan">
11                         <i class="fa fa-edit fa-fw"></i>
12                     </a>
13                 </div>
14             @endauth
15
16         <div class="card-body text-center">
17             <h3 class="card-title py-1">{{ $jurusan->nama }}</h3>
18             <hr>
19             ...

```

Tombol "Edit" saya buat menggunakan tag `<i>` yang berisi icon edit milik font-awesome. Ini didapat dari penggunaan class `.fa .fa-edit` dan `.fa-fw` seperti di baris 11.

Icon edit berada di dalam tag `<a>` yang ketika di klik akan menuju ke *named route* 'jurusans.edit'. Selain itu ikut disertakan route parameter 'jurusan' yang berisi id jurusan saat ini.

Tag `<a>` juga berada di dalam tag `<div>` yang berfungsi sebagai container dari tombol, plus terdapat tambahan class `.btn-group` dan `.btn-action` yang menjadi bagian komponen **button group** bawaan Bootstrap (baris 8).

Efek *hover* yang dipakai untuk menyembunyikan tombol berasal dari kode CSS di file `my-style.scss`. Kode tersebut sudah kita tambah saat menjalankan Laravel Mix pertama kali:

`resources\sass\my-style.scss`

```

1  ...
2  /* Untuk membuat button hover di halaman jurusan (card) */
3
4  .card .btn-action {
5      position: absolute;
6      top: 10px;
7      right: 10px;
8  }
9  .card .btn-action {
10     opacity: 0;
11 }
12 .card:hover .btn-action {
13     opacity: 0.7;
14 }
15 .card .btn-action:hover {
16     opacity: 1.0 !important;
17 }

```

Lanjut, tombol "Edit" ini terhubung ke *named route* 'jurusans.edit', yang pada gilirannya akan

di proses oleh method `edit()` di `JurusanController`. Berikut kode untuk method tersebut:

`app\Http\Controllers\JurusanController.php`

```
1 public function edit(Jurusan $jurusan)
2 {
3     return view('jurusan.edit',compact('jurusan'));
4 }
```

Method `edit()` hanya perlu satu perintah untuk menampilkan view `'jurusan.edit'`. Ke dalam view ini ikut dikirim variabel `$jurusan` yang berisi satu object `Jurusan` (menggunakan teknik *route model binding*).

View `'jurusan.edit'` yang dimaksud belum tersedia saat ini. Oleh karena itu silahkan buka folder `resources\views\jurusan` lalu buat satu file bernama `edit.blade.php`:



Gambar: Isi folder `resources\views\jurusan`

Kemudian isi kode berikut ke dalam file `edit.blade.php`:

`resources\views\jurusan\edit.blade.php`

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5     <h1 class="h2">Edit Jurusan</h1>
6 </div>
7 <hr>
8
9 <form method="POST"
10    action="{{ route('jurusans.update',[ 'jurusan' => $jurusan->id]) }}"
11    @method('PATCH')
12    @include('jurusan.form',['tombol' => 'Update'])
13 </form>
14
15 @endsection
```

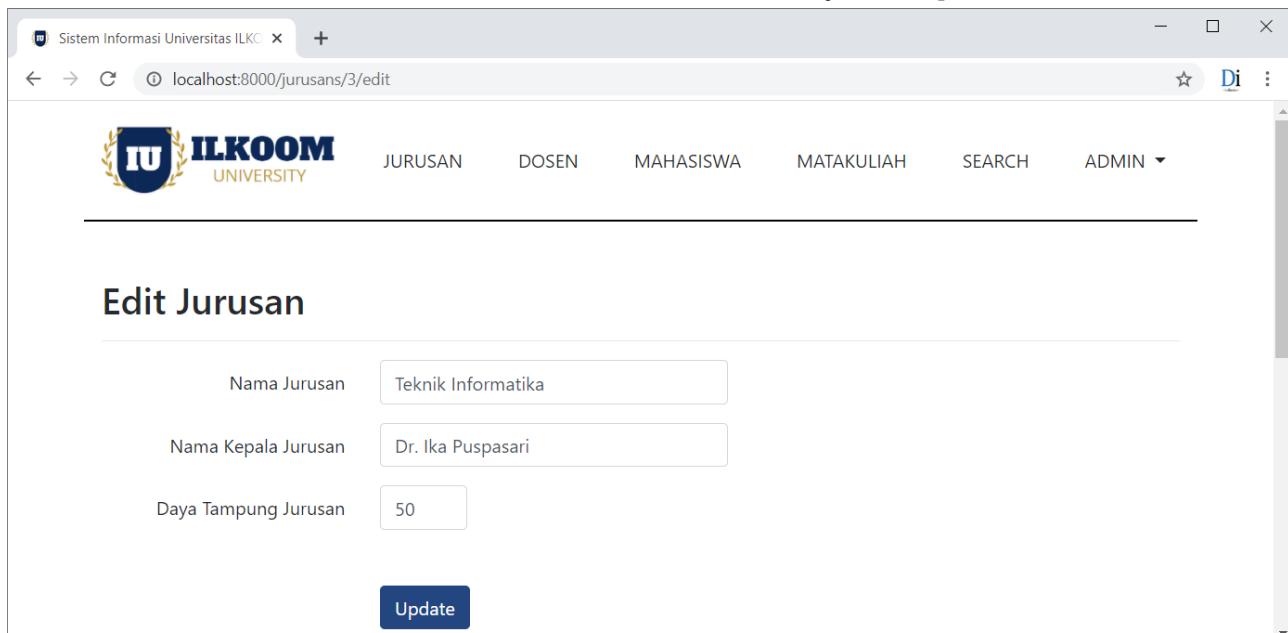
Kode ini sangat mirip seperti yang kita pakai untuk menampilkan form `create` (file `create.blade.php`). Bedanya ada di bagian judul yang sekarang menjadi "Edit Jurusan" (baris 5), lalu alamat pengiriman data form menuju named route `'jurusans.update'`, serta mengisi

string 'Update' ke variabel 'tombol' pada perintah @include di baris 12.

Tidak lupa karena ini adalah proses update, maka butuh tambahan perintah @method('PATCH') di baris 11 agar sesuai dengan route resources Laravel.

Kode HTML untuk pembuatan form ada di view 'jurusan.form'. View ini tidak perlu kita utak-atik karena sudah di siapkan untuk form update. Jika ingin penjelasan lebih lanjut, boleh buka kembali bab sebelumnya tentang pembuatan form menambah data jurusan.

Save file edit.blade.php dan klik tombol "Edit" di salah satu jurusan pada halaman index:



The screenshot shows a web browser window titled 'Sistem Informasi Universitas ILKOOOM'. The URL in the address bar is 'localhost:8000/jurusans/3/edit'. The page displays a navigation menu with links for 'JURUSAN', 'DOSEN', 'MAHASISWA', 'MATAKULIAH', 'SEARCH', and 'ADMIN'. Below the menu, the title 'Edit Jurusan' is centered. There are three input fields: 'Nama Jurusan' containing 'Teknik Informatika', 'Nama Kepala Jurusan' containing 'Dr. Ika Puspasari', and 'Daya Tampung Jurusan' containing '50'. At the bottom is a blue 'Update' button.

Gambar: Tampilan form edit jurusan

Karena ini merupakan form edit, maka setiap inputan harus berisi data yang diambil dari database. Proses menampilkan data ini berasal dari atribut value di setiap tag <input>:

```
value="{{ old('nama') ?? $jurusan->nama ?? '' }}"
```

Kode di atas adalah potongan tag <input> untuk nama jurusan. Operator '??' inilah yang memutuskan nilai apa yang harus diisi ke dalam atribut value, yakni:

1. `old('nama')`, jika form tampil karena error validasi.
2. `$jurusan->nama`, jika form tampil untuk proses edit.
3. `''`, jika form tampil untuk proses create.

Pada saat tombol Update di klik, data form akan di kirim ke named route 'jurusans.update', yang selanjutnya di proses oleh method `update()` di `JurusanController.php`. Berikut kode untuk method tersebut:

app\Http\Controllers\JurusanController.php

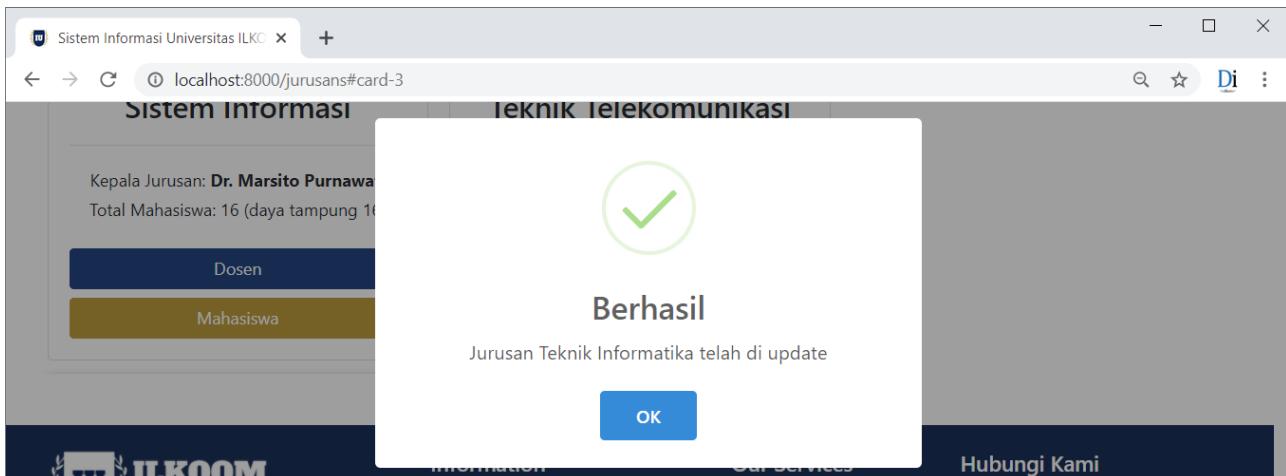
```
1 public function update(Request $request, Jurusan $jurusan)
```

```
2  {
3      $validateData = $request->validate([
4          'nama' => 'required',
5          'kepala_jurusan' => 'required',
6          'daya_tampung' => 'required|min:10|integer',
7      ]);
8
9      $jurusan->update($validateData);
10     Alert::success('Berhasil', "Jurusan $request->nama telah di update");
11     return redirect("/jurusans#card-{$jurusan->id}");
12 }
```

Di awal method terdapat syarat validasi yang sama persis seperti form create. Setiap inputan tidak boleh kosong (`required`) dan khusus untuk inputan daya tampung harus bertipe angka (`integer`) serta tidak boleh kurang dari 10 (`min:10`).

Jika syarat validasi terpenuhi, update data ke database dengan perintah `$jurusan->update ($validateData)` di baris 9. Setelah itu buat pesan alert dan `redirect` user kembali ke halaman index beserta *hash fragment* untuk menandai jurusan yang baru saja di update.

Form edit jurusan sudah selesai, silahkan test dengan mengubah isi salah satu jurusan:



Gambar: Proses update jurusan berhasil

Karena menggunakan struktur form dan syarat validasi yang sama seperti form create, percobaan error validasi tidak akan saya test lagi (seharusnya sudah langsung aktif).

24.2. Mengedit Data Dosen

Kita masuk ke pembuatan form untuk data dosen. Tombol "Edit" juga akan saya tambah di halaman index, yakni di sisi kanan tabel dosen. Silahkan buka file `dosen\index.blade.php` dan modifikasi sebagai berikut:

```
resources\views\dosen\index.blade.php
```

```
1 ...
```

Sistem Informasi Universitas ILKOOOM: Update

```

2 <table class="table table-striped">
3   <thead>
4     <tr>
5       <th>#</th>
6       <th>NID</th>
7       <th>Nama Dosen</th>
8       <th>Jurusan Dosen</th>
9       @auth
10      <th>Action</th>
11      @endauth
12    </tr>
13  </thead>
14  <tbody>
15    @foreach ($dosens as $dosen)
16    <tr id="row-{{$dosen->id}}">
17      <th>{{$dosens->firstItem() + $loop->iteration - 1}}</th>
18      ...
19      <td>{{$dosen->jurusan->nama}}</td>
20      @auth
21      <td>
22        <a href="{{ route('dosens.edit', ['dosen' => $dosen->id])}}" 
23          class="btn btn-secondary" title="Edit Dosen">Edit</a>
24      </td>
25      @endauth
26    </tr>
27    @endforeach
28  </tbody>
29 </table>
30 ...

```

The screenshot shows a web application titled "Data Dosen Universitas ILKOOOM". At the top right is a "Tambah Dosen" button. Below the title is a table with columns: #, NID, Nama Dosen, Jurusan Dosen, and Action. The Action column contains three blue "Edit" buttons, each with a red border, corresponding to the rows. The table has three data rows:

#	NID	Nama Dosen	Jurusan Dosen	Action
1	99154082	Betania Yolanda M.Sc	Teknik Informatika	<button>Edit</button>
2	99629495	Darmanto Agustina M.T	Teknik Informatika	<button>Edit</button>
3	99574701	Edward Prabowo M.Sc	Ilmu Komputer	<button>Edit</button>

Gambar: Tampilan tombol "Edit" dosen

Modifikasi file `index.blade.php` ada di 3 tempat. Yakni pada bagian header tabel untuk membuat teks "**Action**" di baris 9 -11, menambah atribut `id="row-{{$dosen->id}}"` di baris 16, serta menambah tag `<a>` untuk membuat tombol "**Edit**" di baris 20 – 25.

Hasilnya, tombol "Edit" tampil di kolom paling kanan tabel dosen. Tombol ini juga berada

dalam blok @auth, sehingga hanya terlihat bagi user yang sudah login saja.

Atribut **id** yang di tambah ke tag **<tr>** berfungsi sebagai patokan *hash fragment*. Dimana nantinya akan saya pakai untuk membuat efek penanda baris yang baru saja di update. Dengan tambahan ini, setiap tag **<tr>** akan memiliki atribut **id** sesuai dengan angka id dosen tersebut:

The screenshot shows a browser window titled "Sistem Informasi Universitas ILKOO" with the URL "localhost:8000/dosens". Below the header, there is a table with columns: "Nama Dosen", "Jurusan Dosen", and "Action". The first row has an ID of "row-8" and the second row has an ID of "row-5". In the developer tools, the "Elements" tab is selected, showing the HTML code for the table. Two red arrows point to the <tr> elements with IDs "row-8" and "row-5". The right panel of the developer tools shows the CSS styles applied to the table rows.

Gambar: Tambahan atribut id="row-x" ke dalam tag <tr>

Lanjut, kita masuk ke method **edit()** di **DosenController.php**. Method inilah yang akan di proses saat tombol "Edit" di klik:

app\Http\Controllers\DosenController.php

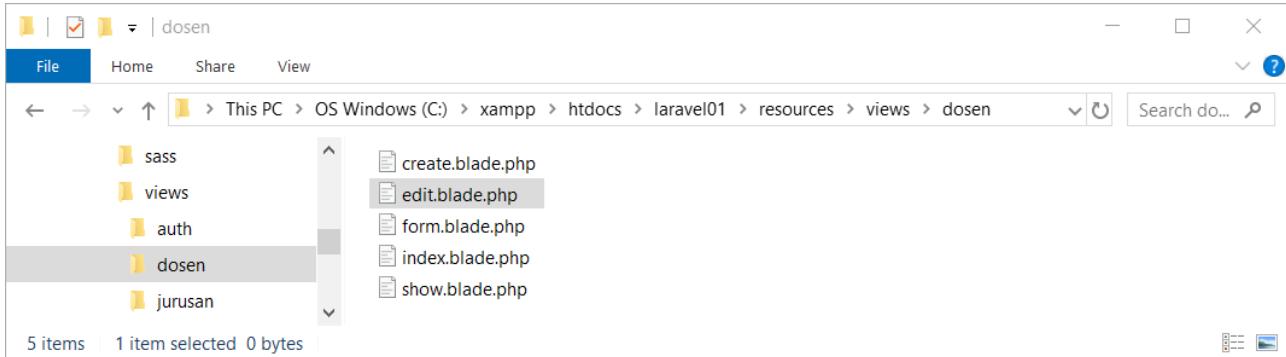
```

1 public function edit(Dosen $dosen)
2 {
3     $jurusans = Jurusan::orderBy('nama')->get();
4     return view('dosen.edit',[ 'dosen' => $dosen, 'jurusans' => $jurusans]);
5 }
```

Kode program di baris 3 akan mengambil semua data jurusan. Sama seperti di form create, variabel **\$jurusans** ini nantinya dipakai untuk membuat menu dropdown pilihan jurusan. Data **\$jurusans** dan juga data **\$dosen** selanjutnya dikirim ke dalam view '**dosen.edit**'.

View '**dosen.edit**' saat ini belum tersedia, oleh karena itu silahkan buat file **edit.blade.php** di dalam folder **resources\views\dosen**:

Sistem Informasi Universitas ILKOM: Update



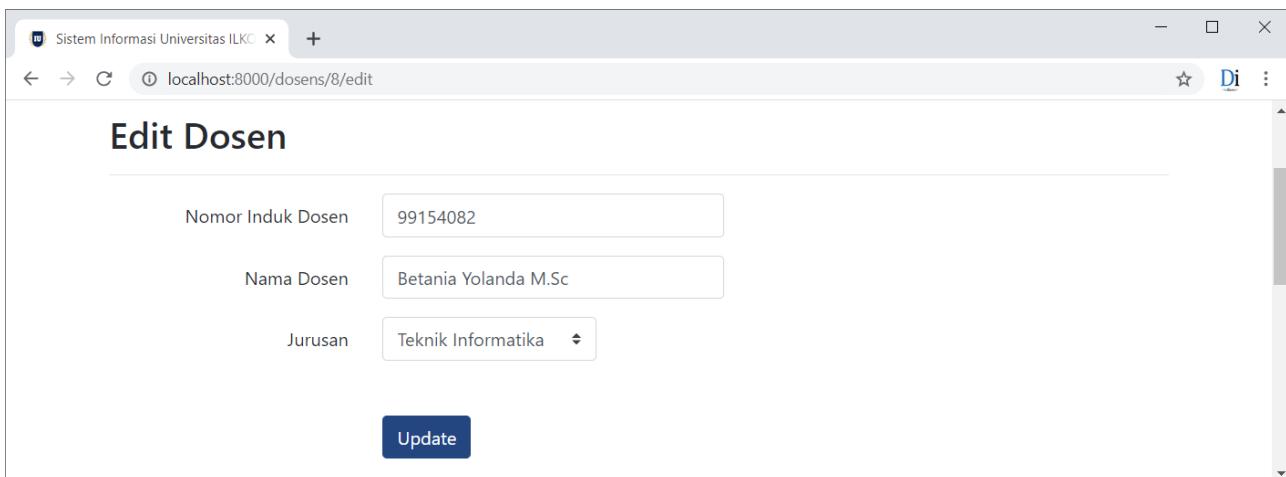
Gambar: Isi folder resources\views\dosen

Kemudian isi kode berikut ke dalam file tersebut:

resources\views\dosen\edit.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5     <h1 class="h2">Edit Dosen</h1>
6 </div>
7 <hr>
8
9 <form method="POST"
10    action="{{ route('dosens.update', ['dosen' => $dosen->id]) }}"
11    @method('PATCH')
12    @include('dosen.form',['tombol' => 'Update'])
13 </form>
14
15 @endsection
```

Sepertinya tidak perlu pembahasan lebih lanjut karena sangat mirip seperti kode yang kita pakai selama ini. Untuk uji coba, buka halaman index dosen lalu klik tombol "Edit" untuk salah satu mahasiswa:



Gambar: Tampilan form edit dosen

Terlihat form edit sudah tampil dengan data yang berasal dari database. Pada saat tombol "**Update**" di klik, data form akan dikirim ke *named route* 'dosens.update', yang selanjutnya di proses oleh method `update()` di `DosenController.php` :

```
app\Http\Controllers\DosenController.php

1 public function update(Request $request, Dosen $dosen)
2 {
3     $validateData = $request->validate([
4         'nid' => 'required|alpha_num|size:8|unique:dosens,nid,' . $dosen->id,
5         'nama' => 'required',
6         'jurusan_id' => 'required|exists:App\Models\Jurusan,id',
7     ]);
8     $dosen->update($validateData);
9     Alert::success('Berhasil', "Dosen $request->nama telah di update");
10    // Trik agar halaman kembali ke awal
11    return redirect($request->url_asal);
12 }
```

Syarat validasi ini juga sama seperti di form create. Hanya saja terdapat tambahan syarat untuk inputan `nid`, yakni `unique:dosens,nid,'.$dosen->id`. Tambahan `$dosen->id` dipakai agar tidak error saat proses update karena nilai `nid` yang sama sudah ada di database.

Apabila syarat validasi terpenuhi, data akan di update ke tabel dengan perintah `$dosen->update($validateData)`. Setelah itu buat pesan alert dan *redirect* user ke `$request->url_asal`.

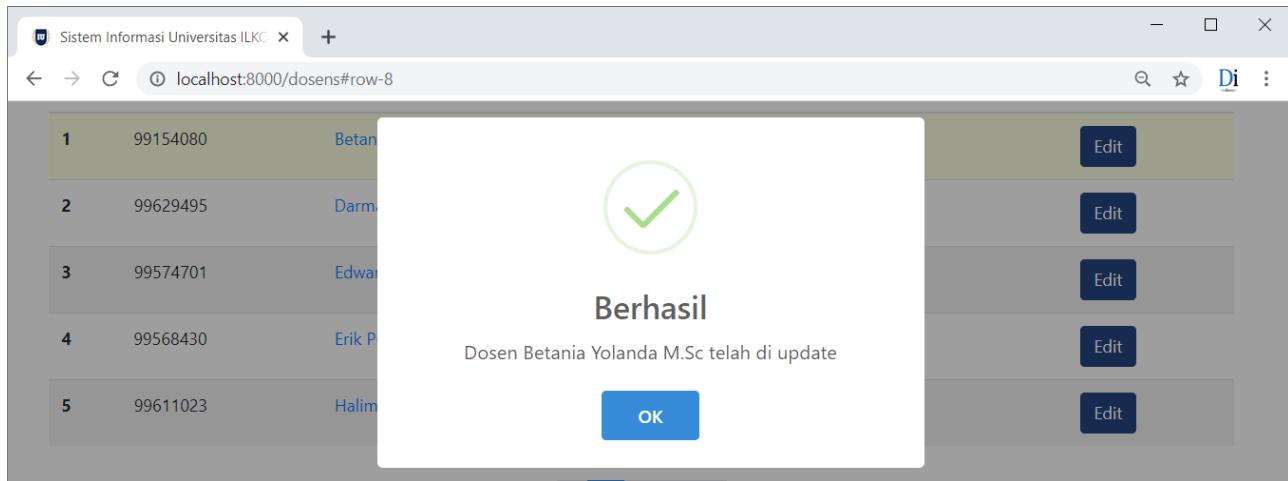
Jika anda masih ingat, nilai `$request->url_asal` ini memiliki 2 kondisi. Berikut potongan kode program yang ada di file `form.blade.php`:

```
1 {{-- Trik agar bisa kembali ke halaman yang klik --}}
2 @isset($dosen)
3     <input type="hidden" name="url_asal"
4     value="{{ old('url_asal') ?? url()->previous(). '#row-' . $dosen->id}}">
5 @else
6     <input type="hidden" name="url_asal"
7     value="{{ old('url_asal') ?? url()->previous()}}">
8 @endisset
```

Ketika menampilkan form edit, kondisi `@isset($dosen)` akan menjadi **true** karena variabel `$dosen` sudah dikirim dari controller. Jika ini yang terjadi, blok di baris 3 – 4 akan di proses.

Di dalam blok ini, nilai atribut `value` akan diisi dengan string `url()->previous(). '#row-' . $dosen->id`. Tambahan `#row-' . $dosen->id` inilah yang dipakai untuk membuat *hash fragment*.

Agar penjelasannya lebih jelas, silahkan edit salah satu dosen lalu klik tombol "**Update**":



Gambar: Proses update dosen berhasil

Pesan alert sukses tampil dan data sudah di update ke database. Sekarang perhatikan alamat URL di web browser, yakni `localhost:8000/dosens#row-8`. Alamat inilah yang tersimpan di nilai `$request->url_asal`.

Pada saat memodifikasi file `index.blade.php`, setiap baris tabel sudah memiliki tambahan atribut `id="row-x"` ke dalam tag `<tr>`. Atribut inilah yang bisa kita pakai untuk menandai baris dosen:

A screenshot of a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/dosens#row-8". The main content area displays a table of student data. The first row, which contains the data for "Betania Yolanda M.Sc", has a yellow background color. A red arrow points from the text "Baris tabel yang baru saja di update" (The row that was just updated) to the first row of the table. At the top right of the table, there is a blue "Tambah Dosen" button. The table has columns for #, NID, Nama Dosen, Jurusan Dosen, and Action.

#	NID	Nama Dosen	Jurusan Dosen	Action
1	99154080	Betania Yolanda M.Sc	Teknik Informatika	Edit
2	99629495	Darmanto Agustina M.T	Teknik Informatika	Edit
3	99574701	Edward Prabowo M.Sc	Ilmu Komputer	Edit

Gambar: Baris tabel yang baru saja di update

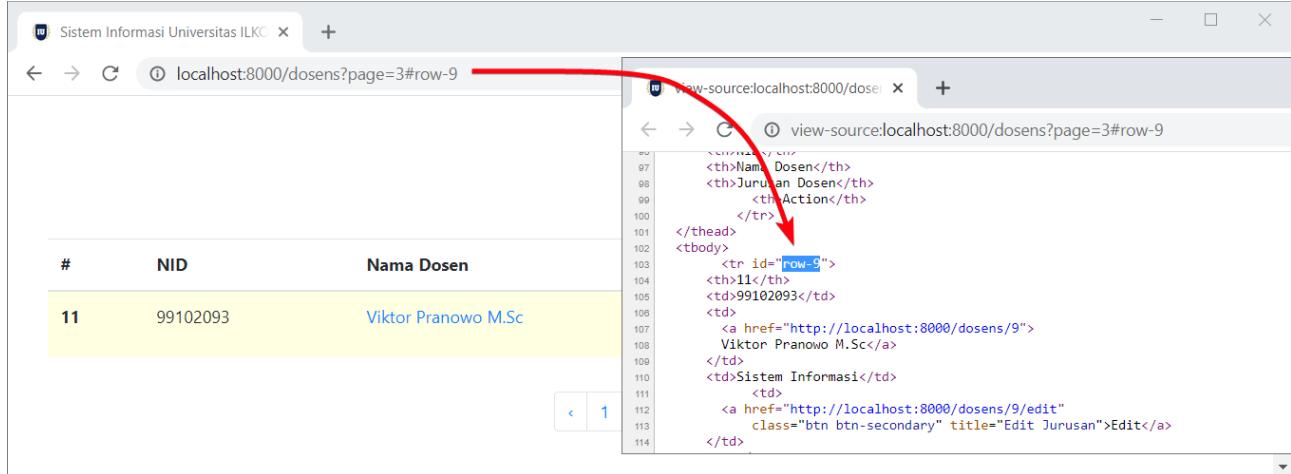
Dengan selector `tr:target`, saya mengubah warna background dari baris yang ditandai oleh `hash fragment`. Berikut kode CSS yang dipakai (sudah ada di file `my-style.scss`):

`resources\sass\my-style.scss`

```

1  /* Membuat efek ketika di target (menandai hasil edit) */
2
3  tr:target{
4      background-color: lightyellow !important;
5 }
```

Fitur *hash fragment* ini juga akan berjalan untuk pagination karena posisi halaman sudah tersimpan di dalam variabel `url_asal`.



Gambar: Hash fragment sesuai dengan nilai atribut id

Dalam gambar di atas, saya mengedit satu dosen yang berada di halaman pagination ke 3. Saat menampilkan form edit, nilai `$request->url_asal` akan berisi alamat lengkap dari halaman asal plus hash fragment, yakni `localhost:8000/dosens?page=3#row-9`.

24.3. Mengedit Data Mahasiswa

Form untuk edit data mahasiswa (dan juga nantinya edit data mata kuliah), akan sangat mirip seperti form update data dosen. Langkah awal adalah menambah tombol "Edit" ke dalam tabel di halaman index. Silahkan buka file `mahasiswa\index.blade.php` lalu modifikasi sebagai berikut:

`resources\views\mahasiswa\index.blade.php`

```

1 ...
2 <table class="table table-striped">
3   <thead>
4     <tr>
5       <th>#</th>
6       <th>NIM</th>
7       <th>Nama Mahasiswa</th>
8       <th>Jurusan Mahasiswa</th>
9       @auth
10      <th>Action</th>
11      @endauth
12    </tr>
13  </thead>
14  <tbody>
15    @foreach ($mahasiswas as $mahasiswa)
16    <tr id="row-{{$mahasiswa->id}}">
17      <th>{{$mahasiswa->firstItem() + $loop->iteration - 1}}</th>
18      ...

```

```
19     <td>{{$mahasiswa->jurusan->nama}}</td>
20     @auth
21     <td>
22         <a href="{{ route('mahasiswas.edit',[ 'mahasiswa' => $mahasiswa->id])}}"
23             class="btn btn-secondary" title="Edit Mahasiswa">Edit</a>
24     </td>
25     @endauth
26     </tr>
27     @endforeach
28   </tbody>
29 </table>
30 ...
```

Bagian yang ditambah juga sama seperti di view index dosen, yakni judul kolom "Action" di baris 9 – 11, atribut `id` untuk membuat efek *hash fragment* di baris 16, serta tombol "Edit" di baris 20 – 25.

Berikut tampilan halaman index mahasiswa setelah penambahan kode di atas:

The screenshot shows a table of student information with columns: #, NIM, Nama Mahasiswa, and Jurusan Mahasiswa. The third row contains data for student ID 10388950, name Bagiya Tampubolon, and major Ilmu Komputer. To the right of the table is a vertical sidebar with a header 'Tambah Mahasiswa'. Below it is a section titled 'Action' containing three blue 'Edit' buttons, each corresponding to one of the three rows in the table. A red arrow points from the text 'Tampilan tombol "Edit" mahasiswa' to the 'Edit' button in the 'Action' column of the third row.

#	NIM	Nama Mahasiswa	Jurusan Mahasiswa
1	10648094	Ani Suwarno	Sistem Informasi
2	19005017	Anissa Lestari	Teknik Telekomunikasi
3	10388950	Bagiya Tampubolon	Ilmu Komputer

Gambar: Tampilan tombol "Edit" mahasiswa

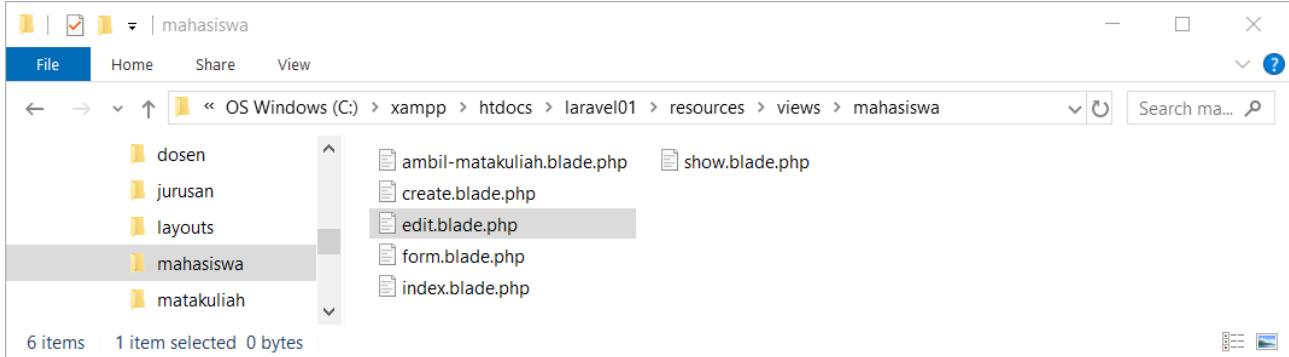
Lanjut, kita masuk ke method `edit()` di `MahasiswaController.php`. Method inilah yang akan di proses saat tombol "Edit" di klik:

app\Http\Controllers\MahasiswaController.php

```
1 public function edit(Mahasiswa $mahasiswa)
2 {
3     $jurusans = Jurusan::orderBy('nama')->get();
4     return view('mahasiswa.edit', [
5         'mahasiswa' => $mahasiswa,
6         'jurusans' => $jurusans,
7     ]);
8 }
```

Dalam kode program ini saya mengambil semua data jurusan, lalu nilainya dikirim ke view 'mahasiswa.edit' beserta object dari mahasiswa yang sedang di edit.

View 'mahasiswa.edit' saat ini belum tersedia, oleh karena itu silahkan buat file **edit.blade.php** di dalam folder **resources\views\mahasiswa**:



Kemudian isi kode berikut ke dalam file tersebut:

resources\views\mahasiswa\edit.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Edit Mahasiswa</h1>
6 </div>
7 <hr>
8
9 <form method="POST"
10 action="{{ route('mahasiswas.update', ['mahasiswa' => $mahasiswa->id]) }}"
11   @method('PATCH')
12   @include('mahasiswa.form', ['tombol' => 'Update'])
13 </form>
14
15 @endsection
```

Kurang lebih sama seperti kode yang selama ini kita pakai. Untuk uji coba, buka halaman index mahasiswa lalu klik tombol "**Edit**" untuk salah satu mahasiswa:

The screenshot shows a web page titled "Edit Mahasiswa". It contains three input fields: "Nomor Induk Mahasiswa" with value "10915899", "Nama Mahasiswa" with value "Devi Permata", and "Jurusan" with value "Ilmu Komputer". Below the fields is a blue "Update" button.

Gambar: Tampilan form edit dosen

Form edit sudah sukses tampil beserta nilai awal yang berasal dari tabel `mahasiswa`.

Namun saya ingin menambah sedikit logika program. Dengan form edit ini, jurusan mahasiswa bisa diubah sesuka hati. Yang jadi masalah adalah, bagaimana dengan hubungannya ke mata kuliah yang sudah diambil oleh mahasiswa tersebut?

Jika saat ini si mahasiswa sudah mengambil beberapa mata kuliah dari jurusan Ilmu Komputer, maka akan terjadi masalah jika dia pindah ke jurusan Sistem Informasi.

Salah satu logika adalah jika si mahasiswa mengubah jurusan, maka semua mata kuliah yang sudah diambil kembali menjadi kosong (dihapus otomatis). Atau bisa juga kita melarang mahasiswa untuk menukar jurusan selama ia masih memiliki jurusan yang sudah diambil.

Saya akan pakai solusi kedua, yakni jika si mahasiswa sudah mengambil minimal 1 mata kuliah, maka jurusan **tidak bisa ditukar**.

Untuk membuatnya, saya akan modifikasi sedikit file `form.blade.php`, tepatnya untuk menu dropdown inputan jurusan:

`resources\views\mahasiswa\form.blade.php`

```
1 ...  
2 <div class="form-group row">  
3   <label for="jurusan_id" class="col-md-3 col-form-label text-md-right">  
4     Jurusan </label>  
5   {{-- jurusan tidak bisa diubah jika ada mata kuliah yang sudah diambil --}}  
6   @if ( ($tombol=='Update') AND ($mahasiswa->matakuliahs->count() > 0) )  
7     <div class="col-md-6 d-flex align-items-center">  
8       <div>{{ $mahasiswa->jurusan->nama }}  
9         <small><i>(tidak bisa di ubah karena sudah mengambil  
10           {{ $mahasiswa->matakuliahs->count() }} mata kuliah)</i></small>  
11       </div>  
12     </div>  
13   {{-- Kirim nilai jurusan awal agar tidak bermasalah dengan validasi --}}  
14   <input type="hidden" name="jurusan_id" id="jurusan_id"
```

```

15      value="{{ $mahasiswa->jurusan_id}}">
16  @else
17      {{-- Untuk form create atau mahasiswa belum mengambil matakuliah --}}
18  <div class="col-md-6">
19      <select name="jurusan_id" id="jurusan_id"
20          class="custom-select col-md-5 @error('jurusan_id') is-invalid @enderror">
21          @foreach ($jurusans as $jurusan)
22              @if ($jurusan->id == (old('jurusan_id') ?? $dosen->jurusan_id ?? ''))
23                  <option value="{{ $jurusan->id }}" selected>{{ $jurusan->nama }}</option>
24              @else
25                  <option value="{{ $jurusan->id }}>{{ $jurusan->nama }}</option>
26              @endif
27          @endforeach
28      </select>
29      @error('jurusan_id')
30          <span class="invalid-feedback" role="alert">
31              <strong>{{ $message }}</strong>
32          </span>
33      @enderror
34  </div>
35  @endif
36 </div>
37 ...

```

Tag `<select>` yang sebelumnya dipakai untuk menampilkan pilihan jurusan, sekarang berada dalam sebuah kondisi if. Kondisi tersebut di definisikan pada baris 6, yakni:

```
@if ( ($tombol=='Update') AND ($mahasiswa->matakuliah->count() > 0) )
```

Nilai variabel `$tombol` berasal dari halaman `create.blade.php` atau `update.blade.php`. Kondisi `($tombol=='Update')` hanya akan bernilai **true** jika halaman form diakses untuk proses edit/update.

Kondisi kedua, yaitu `$mahasiswa->matakuliah->count() > 0`, akan bernilai **true** jika mahasiswa sudah terdaftar setidaknya di satu mata kuliah.

Jika kedua kondisi ini terpenuhi, maka blok kode program di baris 7 – 15 akan dijalankan, dimana pilihan jurusan akan tampil sebagai teks dan tidak bisa ditukar. Supaya nilai jurusan tetap sampai ke proses validasi, diganti dengan sebuah tag `<input>` bertipe hidden di baris 14.

Teknik yang sama sudah pernah kita pakai saat pembuatan form "Menambah Matakuliah untuk Dosen".

Namun jika salah satu atau kedua kondisi tidak terpenuhi, yakni jika form tampil untuk proses create atau mahasiswa yang di edit belum memilih satu pun mata kuliah, maka blok kode program di baris 17 – 34 akan di proses. Dimana pilihan jurusan akan tampil sebagai menu dropdown.

Berikut contoh tampilan form untuk mahasiswa yang sudah memiliki mata kuliah:

Gambar: Tampilan form edit dosen dengan pilihan jurusan tidak aktif

Pada saat tombol "**Update**" di klik, data form akan dikirim ke named route 'mahasiswa.update', yang selanjutnya di proses oleh method update() di MahasiswaController.php :

app\Http\Controllers\MahasiswaController.php

```

1  public function update(Request $request, Mahasiswa $mahasiswa)
2  {
3      $validateData = $request->validate([
4          'nim' => 'required|alpha_num|size:8|unique:mahasiswa,nim,
5              '.$mahasiswa->id,
6          'nama' => 'required',
7          'jurusan_id' => 'required|exists:App\Models\Jurusan,id',
8      ]);
9
10     // Antisipasi jika ada yang edit inputan jurusan_id yang sudah di hidden
11     if (($mahasiswa->matakuliahs()->count() > 0) AND
12         ($mahasiswa->jurusan_id != $request->jurusan_id) ) {
13         Alert::error('Update gagal', "Jurusan tidak bisa diubah!");
14         return back()->withInput();
15     }
16
17     $mahasiswa->update($validateData);
18     Alert::success('Berhasil', "Mahasiswa $request->nama telah di update");
19     // Trik agar halaman kembali ke halaman asal
20     return redirect($request->url_asal);
21 }
```

Syarat validasi ini juga sama seperti di form create. Hanya saja sekarang terdapat tambahan syarat untuk inputan nim, yakni unique:mahasiswa,nim, '.\$mahasiswa->id. Tambahan \$mahasiswa->id dipakai agar tidak error saat proses update karena nilai nim yang sama sudah ada di database.

Apabila syarat validasi terpenuhi, akan ada pemeriksaan lanjutan di baris 11 – 15. Kode ini dipakai untuk memastikan tidak ada perubahan jurusan bagi mahasiswa yang sudah memilih matakuliah.

Kondisi `$mahasiswa->matakuliah->count() > 0` akan bernilai **true** jika si mahasiswa sudah memiliki mata kuliah. Serta kondisi `$mahasiswa->jurusan_id != $request->jurusan_id` juga akan bernilai **true** jika id jurusan mahasiswa yang ada di database tidak cocok dengan jurusan yang berasal dari form.

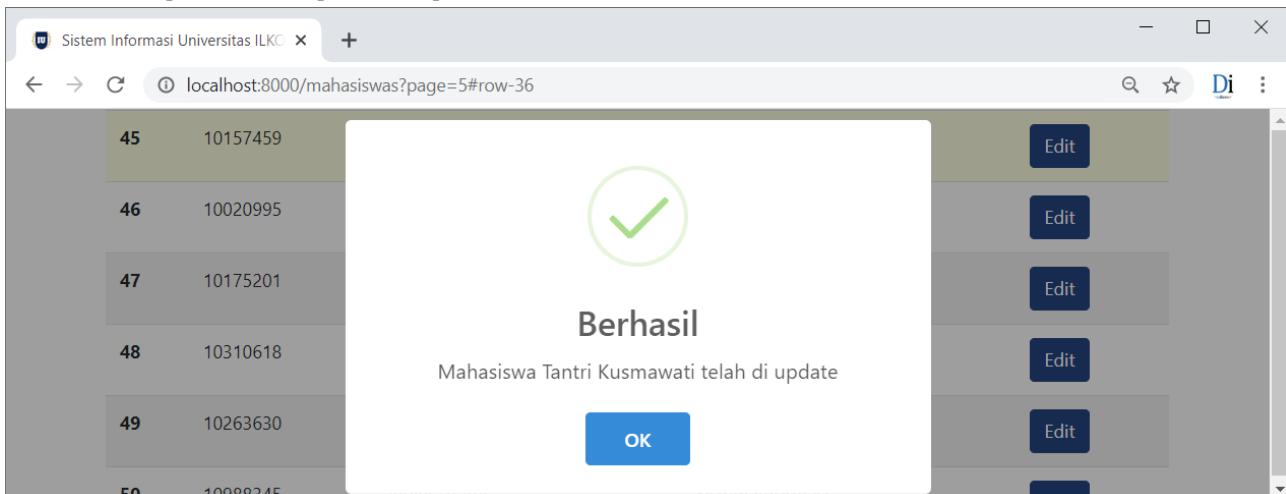
Jika kedua kondisi ini terpenuhi, buat pesan alert dan *redirect* user kembali ke halaman form.

Kondisi di atas hanya bisa terjadi jika form di halaman edit di modifikasi manual menggunakan web developer tools / inspect element.

Selanjutnya kode program di baris 17 – 20 akan dijalankan jika data form lolos validasi, yakni update tabel mahasiswas dengan perintah `$mahasiswa->update($validateData)`, buat pesan alert, dan *redirect* user ke `$request->url_asal`.

URL di web browser, yakni `localhost:8000/dosens#row-8`. Alamat inilah yang tersimpan di nilai `$request->url_asal`.

Berikut tampilan akhir proses update mahasiswa:



Gambar: Pesan alert berhasil update data mahasiswa

44	10328857	Saka Yulianti	Teknik Informatika	Edit
45	10157459	Tantri Kusmawati	Ilmu Komputer	Edit
46	10020995	Timbul Nugroho	Ilmu Komputer	Edit

Gambar: Baris mahasiswa yang baru saja di update

Sama seperti halaman index tabel dosen, tampilan tabel mahasiswa ini juga menggunakan efek *hash fragment* untuk menandai baris yang baru saja di update.

24.4. Mengedit Data Matakuliah

Kembali, apa yang akan kita lakukan sangat mirip seperti di proses edit data mahasiswa. Langkah awal adalah menambah tombol "Edit" ke dalam tabel di halaman index. Silahkan buka file `matakuliah\index.blade.php` lalu modifikasi sebagai berikut:

`resources\views\matakuliah\index.blade.php`

```

1 ... 
2 <table class="table table-striped">
3   <thead>
4     <tr>
5       <th>#</th>
6       <th>Kode</th>
7       ...
8       <th>Jurusan</th>
9       @auth
10      <th>Action</th>
11      @endauth
12    </tr>
13  </thead>
14  <tbody>
15  @foreach ($matakuliah as $matakuliah)
16  <tr id="row-{{$matakuliah->id}}">
17    <th>{{$matakuliah->firstItem() + $loop->iteration - 1}}</th>
18    ...
19    <td>{{$matakuliah->jurusan->nama}}</td>
20    @auth
21    <td>
22      <a href="{{route('matakuliah.edit',[ 'matakuliah'=>$matakuliah->id])}}" 
23        class="btn btn-secondary" title="Edit Mata Kuliah">Edit</a>
24    </td>
25    @endauth
26  </tr>
27  @endforeach
28 </tbody>
29 </table>
```

Bagian yang ditambah tetap sama seperti sebelumnya, yakni judul kolom "**Action**" di baris 9 – 11, atribut `id` untuk membuat efek *hash fragment* di baris 16, serta tombol "**Edit**" di baris 20 – 25.

Berikut tampilan halaman index mata kuliah setelah penambahan kode di atas:

#	Kode	Nama Mata Kuliah	Dosen Pengajar	Jumlah SKS	Jurusan	Action
1	RM463	Agama dan Etika	Kuncara Purnawati M.Kom	2	Sistem Informasi	<button>Edit</button>
2	CR714	Algoritma & Struktur Data	Halima Mansur M.Sc	4	Sistem Informasi	<button>Edit</button>
3	XI822	Bahasa Inggris	Edward Prabowo M.Sc	2	Ilmu Komputer	<button>Edit</button>

Gambar: Tampilan tombol "Edit" matakuliah

Lanjut, kita masuk ke method `edit()` di `MatakuliahController.php`. Method inilah yang akan di proses saat tombol "Edit" di klik:

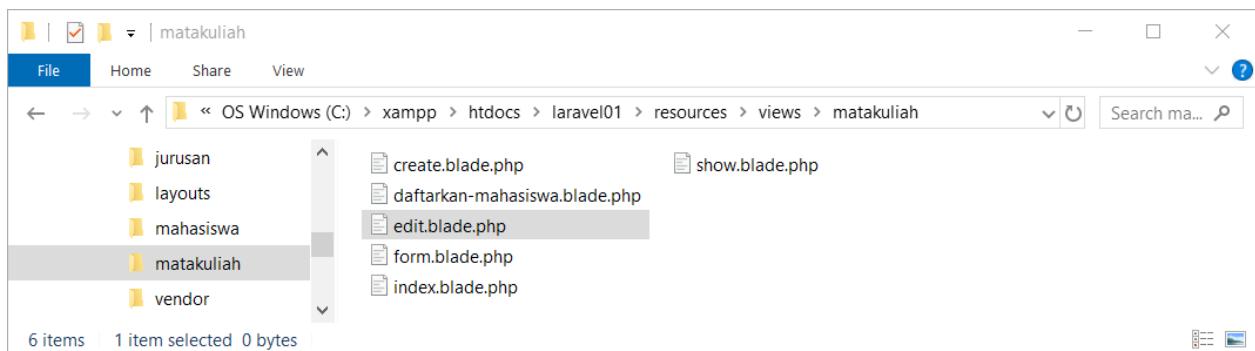
`app\Http\Controllers\MatakuliahController.php`

```

1  public function edit(Matakuliah $matakuliah)
2  {
3      $jurusans = Jurusan::orderBy('nama')->get();
4      $dosens = Dosen::orderBy('nama')->get();
5
6      return view('matakuliah.edit',
7      [
8          'matakuliah' => $matakuliah,
9          'jurusans' => $jurusans,
10         'dosens' => $dosens,
11     ]);
12 }
```

Dalam kode program ini saya mengambil semua data jurusan, semua data dosen yang kemudian dikirim ke view '`mahasiswa.edit`' beserta object dari matakuliah yang sedang di edit.

View '`mahasiswa.edit`' saat ini belum tersedia, oleh karena itu silahkan buat file `edit.blade.php` di dalam folder `resources\views\matakuliah`:



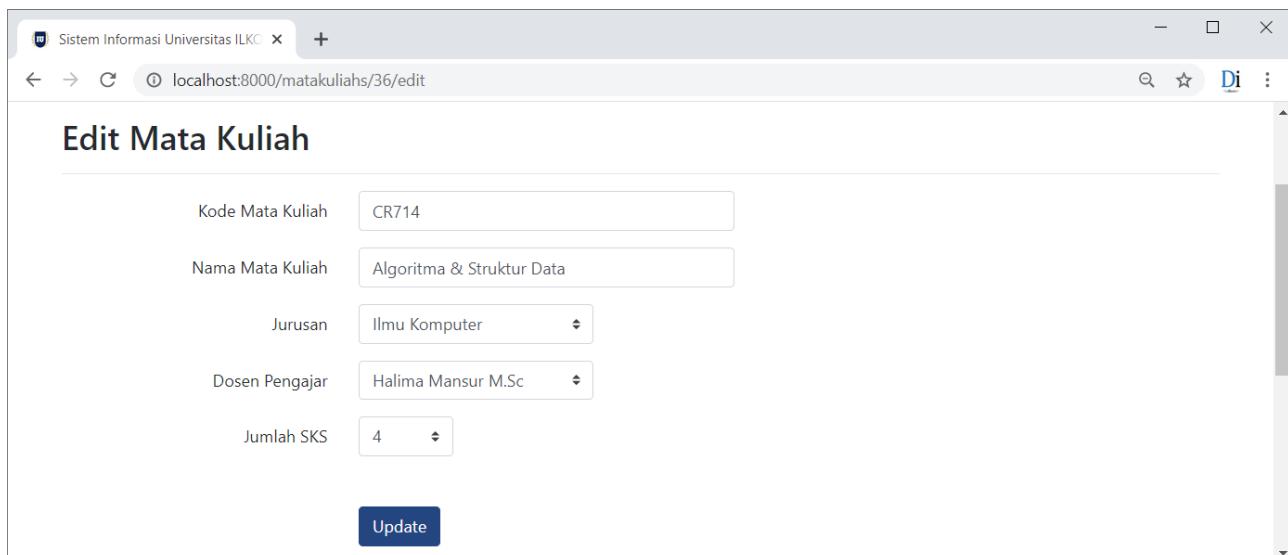
Gambar: Isi folder `resources\views\matakuliah`

Kemudian isi kode berikut ke dalam file tersebut:

resources\views\matakuliah\edit.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3">
5   <h1 class="h2">Edit Mata Kuliah</h1>
6 </div>
7 <hr>
8
9 <form method="POST"
10 action="{{ route('matakuliah.update',[ 'matakuliah' => $matakuliah->id]) }}"
11   @method('PATCH')
12   @include('matakuliah.form',[ 'tombol' => 'Update' ])
13 </form>
14
15 @endsection
```

Masih tetap sama seperti kode yang kita pakai selama ini. Untuk uji coba, buka halaman index matakuliah lalu klik tombol "Edit" untuk salah satu matakuliah:



The screenshot shows a browser window titled "Sistem Informasi Universitas ILKOOOM". The URL in the address bar is "localhost:8000/matakuliah/36/edit". The page content is titled "Edit Mata Kuliah". It contains five input fields: "Kode Mata Kuliah" with value "CR714", "Nama Mata Kuliah" with value "Algoritma & Struktur Data", "Jurusan" with value "Ilmu Komputer" (selected from a dropdown), "Dosen Pengajar" with value "Halima Mansur M.Sc." (selected from a dropdown), and "Jumlah SKS" with value "4" (selected from a dropdown). At the bottom is a blue "Update" button.

Gambar: Tampilan form edit matakuliah

Form edit sudah sukses tampil beserta nilai awal yang berasal dari tabel `matakuliah`.

Namun masalah konsistensi data juga bisa terjadi, yakni bagaimana jika jurusan mata kuliah ditukar? Padahal sudah ada mahasiswa yang mengambil mata kuliah tersebut. Aturan bahwa mahasiswa hanya bisa memilih mata kuliah yang sesuai dengan jurusannya harus kita jaga.

Solusinya, saya kembali akan "membekukan" pilihan jurusan untuk mata kuliah yang sudah diambil oleh minimal 1 mahasiswa. Tapi jika mata kuliah tersebut belum memiliki mahasiswa, maka tidak masalah jika jurusannya ingin ditukar.

Untuk membuatnya, silahkan modifikasi file `form.blade.php` dibagian menu dropdown

inputan jurusan:

resources\views\matakuliah\form.blade.php

```

1 ...
2 <div class="form-group row">
3   <label for="jurusan_id" class="col-md-3 col-form-label text-md-right">
4     Jurusan </label>
5   {{-- jurusan tidak bisa diubah jika ada mata kuliah yang sudah diambil --}}
6   @if ( ($tombol=='Update') AND ($matakuliah->mahasiswa->count() > 0) )
7     <div class="col-md-6 d-flex align-items-center">
8       <div>{{ $matakuliah->jurusan->nama }}</div>
9       <small><i>(tidak bisa di ubah karena sudah diambil
10         {{ $matakuliah->mahasiswa->count() }} mahasiswa)</i></small>
11     </div>
12   </div>
13   {{-- Kirim nilai jurusan awal agar tidak bermasalah dengan validasi --}}
14   <input type="hidden" name="jurusan_id" id="jurusan_id"
15     value="{{ $matakuliah->jurusan_id}}">
16 @else
17   {{-- Untuk form create atau mahasiswa belum mengambil matakuliah --}}
18   <div class="col-md-6">
19     <select name="jurusan_id" id="jurusan_id"
20       class="custom-select col-md-5 @error('jurusan_id') is-invalid @enderror">
21       @foreach ($jurusans as $jurusan)
22         @if ($jurusan->id == (old('jurusan_id') ?? $dosen->jurusan_id ?? ''))
23           <option value="{{ $jurusan->id }}" selected>{{ $jurusan->nama }}</option>
24         @else
25           <option value="{{ $jurusan->id }}>{{ $jurusan->nama }}</option>
26         @endif
27       @endforeach
28     </select>
29     @error('jurusan_id')
30       <span class="invalid-feedback" role="alert">
31         <strong>{{ $message }}</strong>
32       </span>
33     @enderror
34   </div>
35   @endif
36 </div>
37 ...

```

Prinsip yang dipakai sama persis seperti di form edit mahasiswa, yakni dengan memeriksa dua kondisi berikut:

```
@if ( ($tombol=='Update') AND ($matakuliah->mahasiswa->count() > 0) )
```

Kondisi ini akan terpenuhi jika form dibuka untuk proses update, dan matakuliah yang ingin di edit sudah diambil oleh minimal 1 mahasiswa.

Jika keduanya terpenuhi maka blok kode program di baris 7 – 15 akan dijalankan, dimana pilihan jurusan akan tampil sebagai teks dan tidak bisa ditukar. Supaya nilai jurusan tetap

sampai ke proses validasi, diganti dengan sebuah tag <input> bertipe hidden di baris 14.

Namun jika salah satu atau kedua kondisi tidak terpenuhi, yakni jika form tampil untuk proses create atau mata kuliah yang di edit belum dipilih oleh satu pun mahasiswa, maka pilihan jurusan baru akan tampil sebagai menu dropdown.

Berikut contoh tampilan form untuk mata kuliah yang sudah diambil oleh mahasiswa:

The screenshot shows a browser window titled "Sistem Informasi Universitas ILKOOOM". The URL is "localhost:8000/matakuliahs/36/edit". The page title is "Edit Mata Kuliah". The form has five fields: "Kode Mata Kuliah" (CR714), "Nama Mata Kuliah" (Algoritma & Struktur Data), "Jurusan" (dropdown menu showing "Sistem Informasi (tidak bisa di ubah karena sudah diambil 6 mahasiswa)"), "Dosen Pengajar" (dropdown menu showing "Halima Mansur M.Sc"), and "Jumlah SKS" (dropdown menu showing "4"). A red arrow points to the "Jurusan" field, highlighting the validation message "(tidak bisa di ubah karena sudah diambil 6 mahasiswa)".

Gambar: Tampilan form edit dosen dengan pilihan jurusan tidak aktif

Pada saat tombol "**Update**" di klik, data form akan dikirim ke named route 'matakuliahs.update', yang selanjutnya di proses oleh method update() di `MatakuliahController.php`:

app\Http\Controllers\MatakuliahController.php

```
1  public function update(Request $request, Matakuliah $matakuliah)
2  {
3      $validateData = $request->validate([
4          'kode' => 'required|alpha_num|size:5|unique:matakuliahs,kode,'.
5              '$matakuliah->id,
6          'nama' => 'required',
7          'dosen_id' => 'required|exists:App\Models\Dosen,id',
8          'jurusan_id' => 'required|exists:App\Models\Jurusan,id',
9          'jumlah_sks' => 'required|digits_between:1,6',
10     ]);
11
12     // Antisipasi jika ada yang edit inputan jurusan_id yang sudah di hidden
13     if (($matakuliah->mahasiswa->count() > 0) AND
14         ($matakuliah->jurusan_id != $request->jurusan_id) ) {
15         Alert::error('Update gagal','Jurusan tidak bisa diubah!');
16         return back();
17     }
18
19     $matakuliah->update($validateData);
20     Alert::success('Berhasil','Mata kuliah $request->nama telah di update');
21     // Trik agar halaman kembali ke awal
```

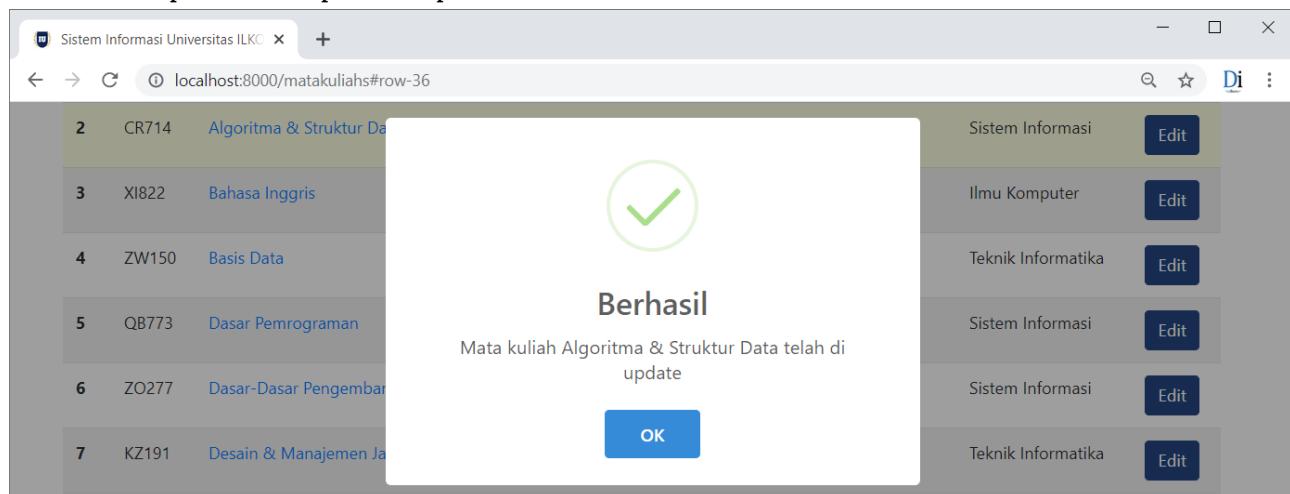
```
22     return redirect($request->url_asal);  
23 }
```

Syarat validasi ini tetap sama seperti di form create. Hanya saja sekarang terdapat tambahan syarat untuk inputan kode, yakni unique:`matakuliah`,`kode`,`'.'``$matakuliah->id`. Tambahan `$matakuliah->id` dipakai agar tidak error saat proses update karena nilai kode yang sama sudah ada di database.

Juga sama seperti di method `update()` untuk form mahasiswa, di baris 13 – 17 terdapat pemeriksaan lanjutan. Kode ini dipakai untuk memastikan tidak ada perubahan jurusan bagi mata kuliah yang sudah memiliki mahasiswa.

Jika sudah lolos validasi, data form akan di update ke tabel `matakuliah` dengan perintah `$matakuliah->update($validateData)`, lalu buat pesan alert, dan *redirect* user ke `$request->url_asal`.

Berikut tampilan akhir proses update mata kuliah:



Gambar: Pesan alert berhasil update data matakuliah

The screenshot shows a web browser window titled "Sistem Informasi Universitas ILKOM". The URL is "localhost:8000/matakuliah#row-36". At the top right, there is a blue button labeled "Tambah Mata Kuliah". Below it is a table with the following data:

#	Kode	Nama Mata Kuliah	Dosen Pengajar	Jumlah SKS	Jurusan	Action
1	RM463	Agama dan Etika	Kuncara Purnawati M.Kom	2	Sistem Informasi	<button>Edit</button>
2	CR714	Algoritma & Struktur Data	Halima Mansur M.Sc	4	Sistem Informasi	<button>Edit</button>
3	XI822	Bahasa Inggris	Edward Prabowo M.Sc	2	Ilmu Komputer	<button>Edit</button>

Gambar: Baris matakuliah yang baru saja di update

Efek *hash fragment* untuk menandai baris yang di update juga aktif di tabel mata kuliah.

Dalam bab ini kita telah menyelesaikan bagian **Update** dari CRUD Sistem Informasi Universitas ILKOOOM. Pembahasannya relatif singkat karena memakai struktur form yang sama dengan form create. Proses validasi pun juga mirip sehingga tidak perlu kita bahas dengan detail.

Berikutnya kita akan masuk ke bagian **Delete**.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

25. Sistem Informasi Universitas ILKOOM: Delete

Sekarang kita masuk ke bagian 'D' dari CRUD Sistem Informasi Universitas ILKOOM, yakni proses **Delete**.

Dalam beberapa aspek, apa yang akan kita buat masih mirip seperti proses **Edit**, yakni menempatkan satu tombol "**Hapus**" ke halaman index, lalu memprosesnya ke database. Kode yang dibutuhkan juga lebih singkat karena tidak perlu membuat halaman form terpisah. Hanya saja kita akan minta sedikit bantuan kode JavaScript untuk menampilkan jendela konfirmasi.

25.1. Menghapus Data Jurusan

Proses penghapusan data diawali dengan membuat tombol "**Hapus**". Untuk data jurusan, tombol tersebut saya tempatkan di sebelah tombol edit. Tampilan tombol ini akan memakai icon dari font awesome.

Silahkan buka halaman index jurusan lalu tambah kode berikut ke dalam struktur Card:

resources\views\jurusan\index.blade.php

```

1 ...
2 <div class="card-columns mt-3">
3   @foreach($jurusans as $jurusan)
4
5     <div class="card mt-1" id=card-{{ $jurusan->id }}>
6       @auth
7         <div class="btn-group btn-action">
8           <a href="{{ route('jurusans.edit',['jurusan' => $jurusan->id]) }}">
9             class="btn btn-primary d-inline-block" title="Edit Jurusan">
10            <i class="fa fa-edit fa-fw"></i>
11          </a>
12          <form action="{{route('jurusans.destroy',[ 'jurusan' => $jurusan->id]) }}"
13            method="POST">
14            @csrf @method('DELETE')
15            <button type="submit" class="btn btn-danger shadow-none btn-hapus"
16              title="Hapus Jurusan" data-name="{{ $jurusan->nama }}" data-table="jurusan">
17              <i class="fa fa-trash fa-fw"></i>
18            </button>
19          </form>
20        </div>
21      @endauth
22    ...

```



Gambar: Tombol hapus jurusan

Dalam teknik RESTfull Laravel, permintaan penghapusan data akan di proses oleh `Route::delete()`. Untuk bisa mengirim data ke route ini, kita butuh sebuah tombol submit di dalam tag `<form method="POST">`, lalu menambah perintah `@method('DELETE')` ke dalam form tersebut.

Inilah alasan penggunaan tag `<form>` antara baris 12 – 19. Form ini hanya memiliki 1 inputan `<input type="submit">` yang ketika diklik akan mengirim permintaan penghapusan data ke *named route 'jurusans.destroy'*. Sebagai info data yang dihapus, ikut dikirim `id` jurusan sebagai route parameter.

Tombol submit sendiri berisi tag `<i>` dengan class `.fa`, `.fa-trash` dan `.fa-fw` (baris 17). Hasilnya, tombol tampil dalam bentuk gambar tong sampah dari font-awesome.

Pada saat diklik, proses penghapusan data akan ditangani oleh method `destroy()` di `JurusanController`. Berikut kode yang diperlukan:

`app\Http\Controllers\JurusanController.php`

```

1 public function destroy(Jurusan $jurusan)
2 {
3     $jurusan->delete();
4     Alert::success('Berhasil', "Jurusan $jurusan->nama telah di hapus");
5     return redirect("/jurusans");
6 }
```

Untuk menghapus data jurusan, hanya perlu 1 perintah sederhana, yakni `$jurusan->delete()` di baris 3. Variabel `$jurusan` sendiri sudah berisi object `Jurusan` yang berasal dari *route model binding* di bagian argument method `destroy()`.

Namun harap diingat bahwa di dalam aplikasi kita, `jurusans` berfungsi sebagai tabel utama. Ketiga tabel lain (`dosens`, `mahasiswas` dan `matakuliahs`) memiliki kolom *foreign key* yang berasal dari tabel ini.

Pada saat pendefinisian migration, saya mengaktifkan fitur `ON DELETE CASCADE`, sehingga jika satu jurusan dihapus, itu ikut **menghapus semua data** yang berhubungan di ketiga tabel lain. Jadi perintah `$jurusan->delete()` di sini sangat *powerful*.

Kembali ke kode program di method `destroy()`, setelah data jurusan dihapus, perintah di baris 4 akan membuat pesan alert yang kemudian me-redirect user kembali ke halaman '/jurusans'.

Untuk uji coba, silahkan hapus salah satu jurusan:

The screenshot shows a web application interface for managing academic departments. At the top right is a 'Tambah Jurusan' button. Below it are three cards representing different departments:

- Ilmu Komputer**: Kepala Jurusan: Dr. Mustofa Simanjuntak, Total Mahasiswa: 19 (daya tampung 60). Buttons: Dosen (blue), Mahasiswa (yellow).
- Teknik Industri**: Kepala Jurusan: Dr. Mahmud Mulyadi, Total Mahasiswa: 0 (daya tampung 120). Buttons: Dosen (blue), Mahasiswa (yellow). A trash icon with a cursor over it is positioned above the card.
- Teknik Telekomunikasi**: Kepala Jurusan: Prof Poltak Situmorang, Total Mahasiswa: 1 (daya tampung 45). Buttons: Dosen (blue), Mahasiswa (yellow).

Gambar: Menghapus jurusan "Teknik Industri"

The screenshot shows a confirmation message after deleting the 'Teknik Industri' department. The message is displayed in a central box:

Berhasil
Jurusan Teknik Industri telah dihapus
OK

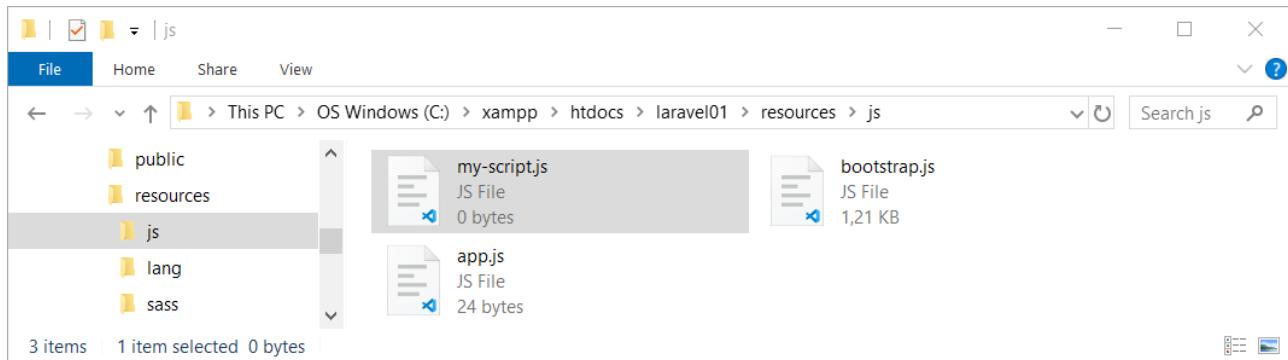
Gambar: Jurusan "Teknik Industri" berhasil dihapus

Dalam contoh ini saya membuat jurusan "Teknik Industri" dan langsung menghapusnya. Proses penghapusan berjalan sukses dan pesan alert juga tampil di halaman index.

Akan tetapi tombol hapus sangat berbahaya jika dibiarkan tanpa jendela konfirmasi. Sangat mungkin user men-klik tombol hapus secara tidak sengaja. Seperti yang kita bahas di atas, menghapus satu jurusan akan ikut menghapus semua data di tabel lain.

Untungnya, di akhir bab SweetAlert kita sudah mempelajari cara membuat jendela konfirmasi. Jendela konfirmasi itu akan saya implementasikan ke dalam project ini.

Silahkan buat sebuah file JavaScript bernama `my-script.js` ke dalam folder `resources\js`:



Gambar: Buat file my-script.js ke dalam folder resources\js

Di dalam file inilah kita akan tulis kode JavaScript untuk menampilkan jendela konfirmasi:

resources\js\my-script.js

```
1 let semuaTombol = document.querySelectorAll('.btn-hapus');
2 semuaTombol.forEach(function(item) {
3     item.addEventListener('click',konfirmasi);
4 })
5
6 function konfirmasi(event){
7     // Buat pesan error untuk setiap tipe tabel
8     let tombol = event.currentTarget;
9     let judulAlert;
10    let teksAlert;
11    switch(tombol.getAttribute('data-table')) {
12        case 'jurusan':
13            judulAlert = 'Hapus Jurusan '+tombol.getAttribute('data-name')+ '?';
14            teksAlert = 'Semua data <b>dosen</b>, <b>mahasiswa</b> dan '+
15                        '<b>mata kuliah</b> untuk jurusan ini juga akan '+
16                        'terhapus!';
17            break;
18        case 'dosen':
19            judulAlert = 'Hapus Dosen '+tombol.getAttribute('data-name')+ '?';
20            teksAlert = 'Semua data <b>mata kuliah</b> untuk dosen '+
21                        'ini juga akan terhapus!';
22            break;
23        default:
24            judulAlert = 'Apakah anda yakin?';
25            teksAlert = 'Hapus data <b>'+tombol.getAttribute('data-name')+ '</b>';
26            break;
27    }
28
29    event.preventDefault();
30    Swal.fire({
31        title: judulAlert,
32        html: teksAlert,
33        icon: 'warning',
34        showCancelButton: true,
35        cancelButtonColor: '#6c757d',
36        confirmButtonColor: '#dc3545',
37        cancelButtonText: 'Tidak jadi',
```

```
38     confirmButtonText: 'Ya, hapus!',  
39     reverseButtons: true,  
40   })  
41   .then((result) => {  
42     if (result.value) {  
43       tombol.parentElement.submit();  
44     }  
45   })  
46  
47 }
```

Pada baris 1, perintah `document.querySelectorAll('.btn-hapus')` dipakai untuk mencari semua tombol hapus yang ada di halaman saat ini. Hasilnya kemudian di simpan ke dalam variabel `semuaTombol`. Sampai di sini, variabel `semuaTombol` akan berisi array, atau tepatnya **NodeList** (kumpulan node element).

Kita ingin menambah event **click** ke semua NodeList yang tersimpan di variabel `semuaTombol`. Caranya adalah dengan memanggil method `forEach()` seperti di baris 2-4. Dengan perintah ini, ketika tombol hapus di klik, function `konfirmasi()` akan dijalankan yang pendefinisianya ada di baris 6 – 47.

Method `forEach()` sebenarnya bisa saja diganti dengan perulangan lain seperti `for`. Fungsinya hanya untuk menempelkan event `click` ke semua node element yang tersimpan di variabel `semuaTombol`.

Function `konfirmasi()` menerima satu argument `event` yang berisi berbagai informasi mengenai event yang sedang terjadi.

Di baris 8, perintah `event.currentTarget` berfungsi untuk mencari node element dari tombol hapus / tombol submit yang saat ini sedang di klik. Kita tidak bisa menggunakan `event.target` karena itu akan mengembalikan node element milik tag `<i>`, sedangkan yang di perlukan adalah node element milik tag `<input>`.

Kode JavaScript ini saya siapkan bukan hanya untuk halaman jurusan saja, tapi juga untuk halaman lain. Oleh karena itu terdapat pendefinisian variabel `judulAlert` dan `pesanAlert` di baris 9 dan 10. Kedua variabel akan dipakai untuk menampung judul dan pesan teks jendela konfirmasi agar berisi pesan yang sesuai.

Untuk menentukan pesan yang harus tampil, saya menempatkan sebuah atribut `data-table` ke dalam tag `<input type="submit">`. Untuk halaman jurusan, atribut tersebut berisi nilai `data-table="jurusan"`.

Untuk mengambil nilai atribut `data-table`, bisa lewat perintah `tombol.getAttribute('data-table')` seperti di baris 11. Perintah ini langsung berada dalam kondisi `switch()` untuk diperiksa nilainya.

Jika nilai atribut `data-table` berisi teks 'jurusan', maka blok case di baris 13- 16 akan

dijalankan. Jika berisi teks 'dosen', giliran kode di baris 19 – 21 yang akan di proses. Selain dua nilai ini, blok case di baris 24 – 25 yang akan dipakai.

Dalam setiap blok case, variabel `judulAlert` dan `pesanAlert` diisi string berbeda. Saat mengisi teks `judulAlert`, terdapat perintah `tombol.getAttribute('data-name')` yang dipakai untuk mengambil nilai atribut `data-name` dari tag `<submit>`. Jika dilihat kembali ke dalam view, atribut `data-name` berisi perintah `{{$jurusan->nama}}`, yakni nama dari jurusan yang akan dihapus.

Selanjutnya perintah `event.preventDefault()` di baris 29 akan menonaktifkan event bawaan tombol submit, lalu diikuti kode program untuk memanggil jendela konfirmasi.

Kode sweetalert ini tidak lagi saya bahas karena sudah kita pelajari di bab SweetAlert. Nilai dari property `title` dan `html` akan diisi oleh variabel `judulAlert` dan `pesanAlert` yang kita siapkan sebelumnya.

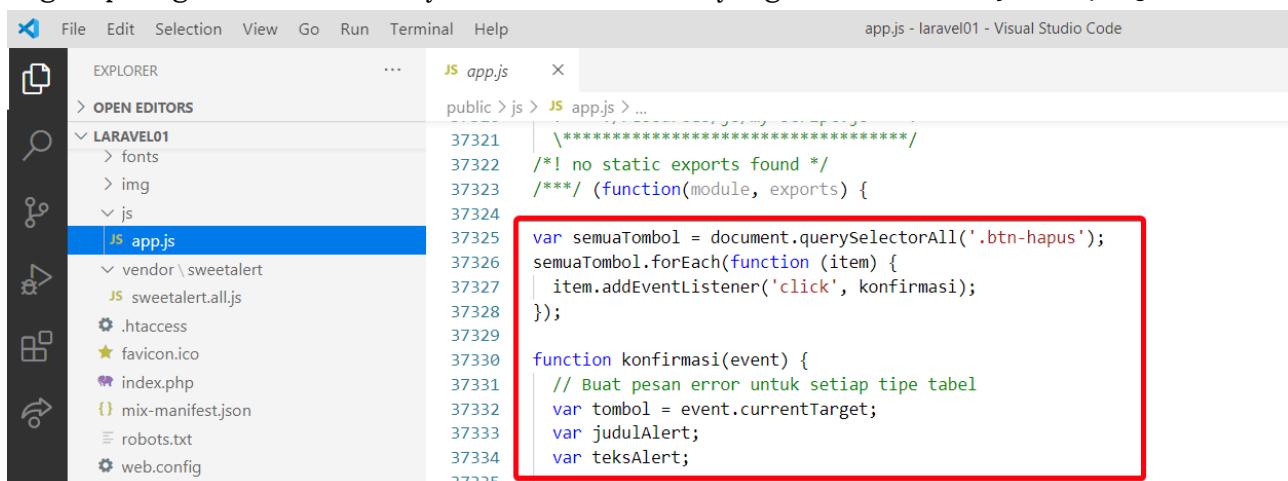
Agar bisa dipakai, file `my-script.js` ini harus di compile menggunakan Laravel Mix. Caranya, buka file `resources\js\app.js` lalu tambah perintah `require('./my-script')` di baris kedua:

`resources\js\my-script.js`

```
1 require('./bootstrap');
2 require('./my-script');
```

Kemudian compile ulang Laravel Mix dengan perintah: `npm run dev`.

Setelah itu untuk sekedar memastikan, silahkan buka file `public\js\app.js` lalu scroll ke bagian paling bawah. Seharusnya akan terlihat kode yang kita tulis di `my-script.js`:



```
File Edit Selection View Go Run Terminal Help app.js - laravel01 - Visual Studio Code

EXPLORER ... JS app.js x
OPEN EDITORS
LARAVEL01
  fonts
  img
  js
    JS app.js
  vendor\sweetalert
    JS sweetalert.all.js
  .htaccess
  favicon.ico
  index.php
  mix-manifest.json
  robots.txt
  web.config

public > js > JS app.js > ...
37321 | *****
37322 /*! no static exports found */
37323 /**/ (function(module, exports) {
37324
37325   var semuaTombol = document.querySelectorAll('.btn-hapus');
37326   semuaTombol.forEach(function (item) {
37327     item.addEventListener('click', konfirmasi);
37328   });
37329
37330   function konfirmasi(event) {
37331     // Buat pesan error untuk setiap tipe tabel
37332     var tombol = event.currentTarget;
37333     var judulAlert;
37334     var teksAlert;
37335
```

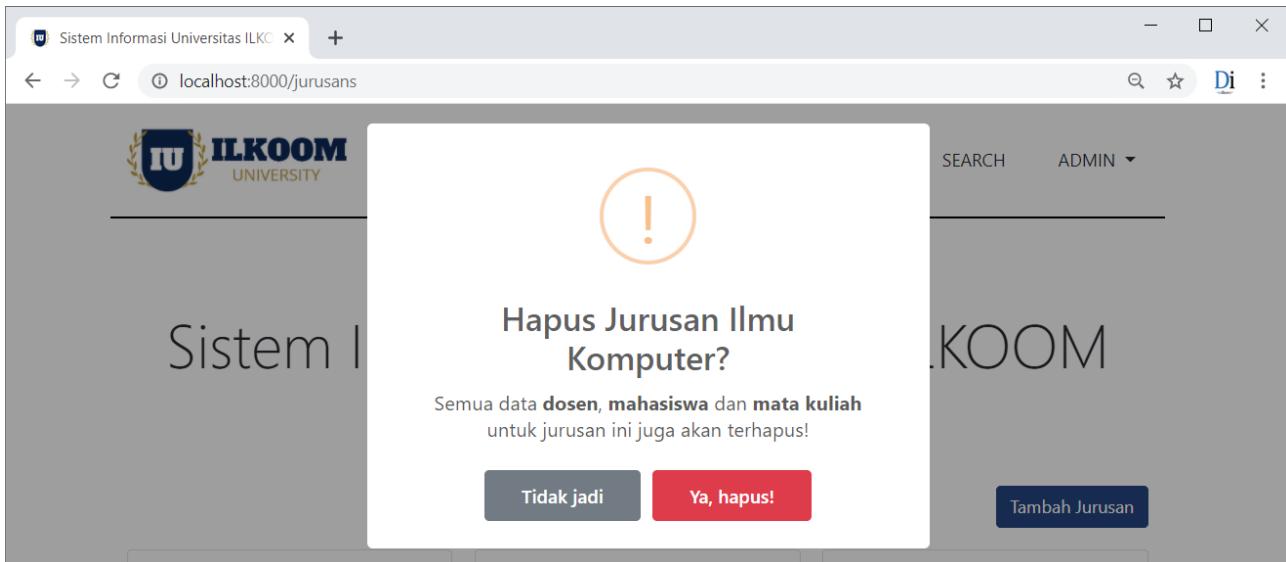
Gambar: Kode my-script.js sudah di compile ke public\js\app.js

Terakhir, buka file konfigurasi `.env`, lalu tambah satu perintah berikut di bagian bawah:

```
SWEET_ALERT_ALWAYS_LOAD_JS = TRUE
```

Perintah ini dipakai agar kode JavaScript milik Realrashid/sweetalert selalu tampil di setiap halaman. Penjelasan lebih lengkap terkait hal ini juga sudah kita bahas di bab SweetAlert.

Saatnya uji coba. Restart `php artisan serve`, lalu buka halaman index jurusan dan klik icon hapus untuk salah satu jurusan:



Gambar: Jendela konfirmasi hapus jurusan

Hasilnya, akan tampil jendela konfirmasi yang berisi pesan peringatan bahwa jika jurusan ini dihapus, maka data dosen, mahasiswa dan mata kuliah yang terkait juga ikut dihapus. Jika yakin dengan hal ini, klik tombol "Ya, hapus!" untuk memulai proses delete.

Silahkan coba hapus jurusan Ilmu Komputer, lalu cek ke halaman index dosen atau index mahasiswa. Hasilnya, dosen dan mahasiswa untuk jurusan Ilmu Komputer sudah tidak lagi ditemukan.

25.2. Menghapus Data Dosen

Untuk menghapus data dosen, juga dimulai dari menempatkan tombol "**Hapus**" ke dalam halaman index dosen. Posisi tombol ini akan bersebelahan dengan tombol edit. Silahkan buka file index dosen lalu modifikasi sebagai berikut:

`resources\views\dosen\index.blade.php`

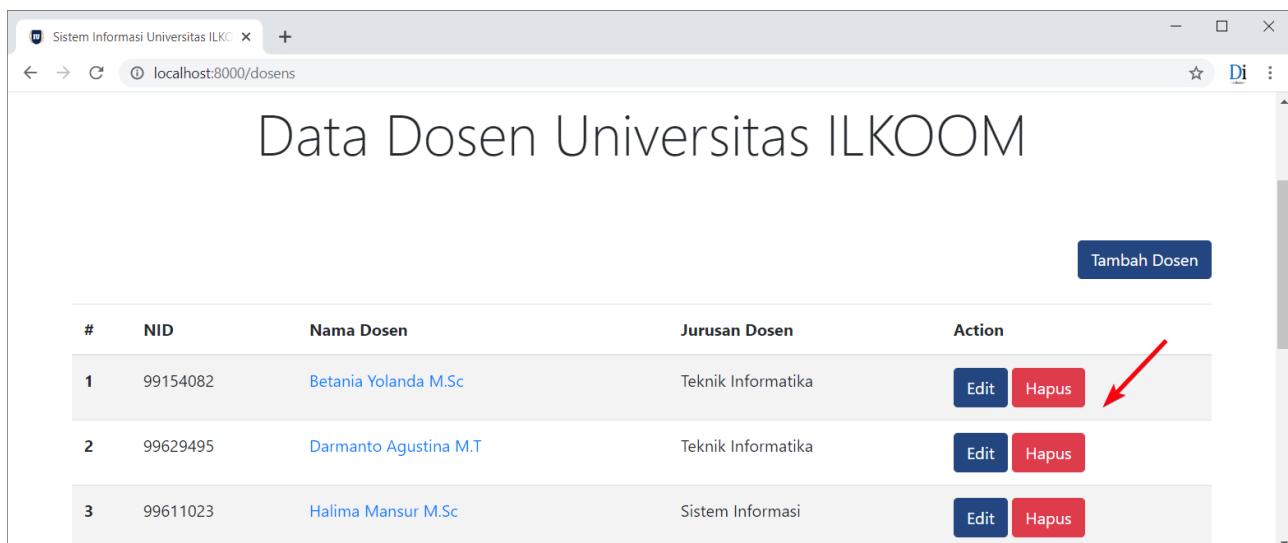
```
1 ...  
2   <tbody>  
3     @foreach ($dosens as $dosen)  
4       <tr id="row-{{$dosen->id}}">  
5         <th>{{$dosens->firstItem() + $loop->iteration - 1}}</th>  
6         <td>{{$dosen->nid}}</td>  
7         <td>  
8           <a href="{{ route('dosens.show', ['dosen' => $dosen->id]) }}>  
9             {{$dosen->nama}}</a>  
10          </td>  
11          <td>{{$dosen->jurusan->nama}}</td>  
12        @auth
```

Sistem Informasi Universitas ILKOOOM: Delete

```

13   <td>
14     <a href="{{ route('dosens.edit',['dosen' => $dosen->id])}}" 
15       class="btn btn-secondary" title="Edit Dosen">Edit</a>
16     <form action="{{route('dosens.destroy',['dosen' => $dosen->id])}}"
17       method="POST" class="d-inline">
18       @csrf @method('DELETE')
19       <button type="submit" class="btn btn-danger shadow-none btn-hapus"
20         title="Hapus Dosen" data-name="{{ $dosen->nama }}"
21         data-table="dosen">Hapus</button>
22     </form>
23   </td>
24   @endauth
25 </tr>
26 @endforeach
27 </tbody>
28 ...

```

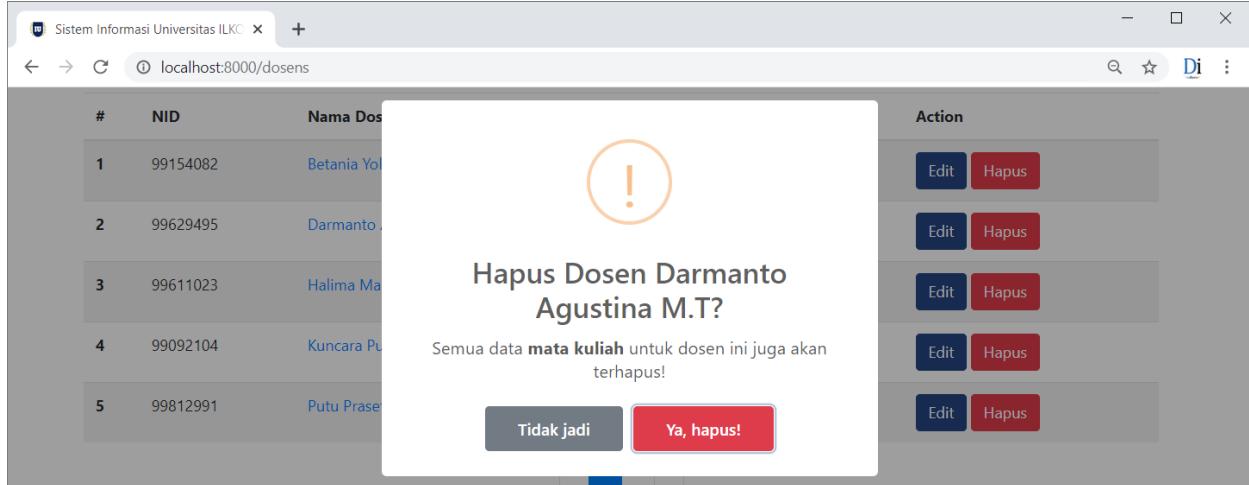


#	NID	Nama Dosen	Jurusan Dosen	Action
1	99154082	Betania Yolanda M.Sc	Teknik Informatika	Edit Hapus
2	99629495	Darmanto Agustina M.T	Teknik Informatika	Edit Hapus
3	99611023	Halima Mansur M.Sc	Sistem Informasi	Edit Hapus

Gambar: Tombol hapus dosen

Kode yang saya pakai untuk membuat tombol hapus dosen mirip seperti di index jurusan. Hanya saja sekarang tidak lagi menggunakan icon font awesome, tapi langsung teks "**Hapus**".

Tag `<submit>` ini juga memiliki atribut `data-name="{{ $dosen->nama }}`" dan `data-table="dosen"` untuk membuat pesan konfirmasi sweetalert. Karena file JavaScript untuk sweetalert sudah tersedia, maka jendela konfirmasi ini juga sudah bisa langsung dicoba:



Gambar: Jendela konfirmasi menghapus Dosen

Pesan teks yang tampil sedikit berubah dari halaman jurusan. Kali ini kita meminta konfirmasi bahwa jika satu data dosen dihapus, maka semua data mata kuliah yang diajar oleh dosen itu juga akan ikut terhapus.

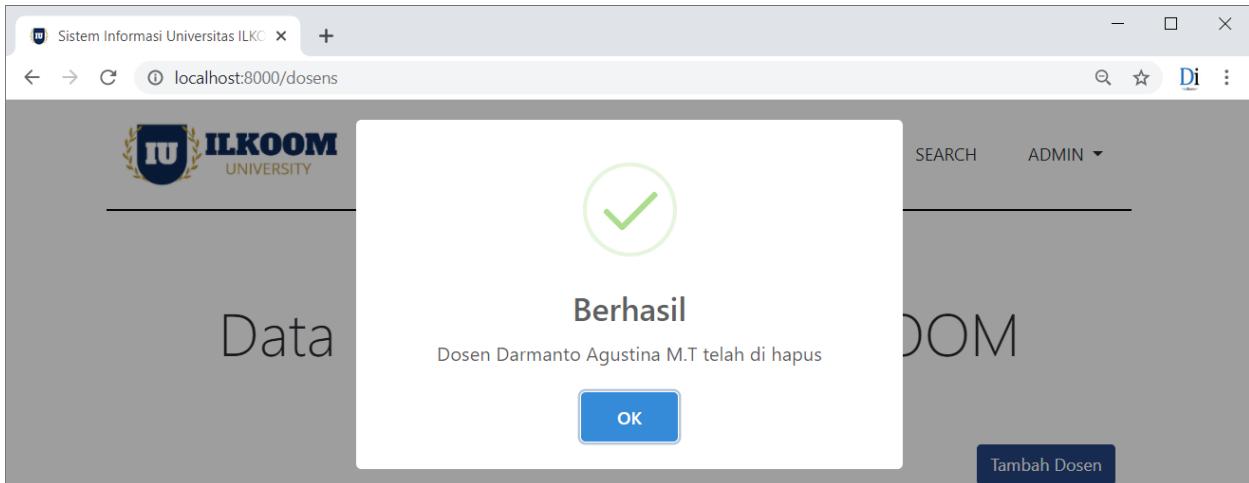
Pada saat tombol "Ya, hapus!" di klik, tidak terjadi apa-apa. Ini karena kode untuk proses delete belum kita tulis. Silahkan ketik perintah berikut ke method `destroy()` di `DosenController.php`:

app\Http\Controllers\DosenController.php

```
1 public function destroy(Dosen $dosen)
2 {
3     $dosen->delete();
4     Alert::success('Berhasil', "Dosen $dosen->nama telah di hapus");
5     return redirect("/dosen");
6 }
```

Sama seperti sebelumnya, di baris 3 saya menghapus dosen dengan perintah `$dosen->delete()`, lalu membuat pesan alert dan me-redirect user kembali ke halaman index dosen.

Berikut tampilan pesan alert saat proses delete berhasil di jalankan:



Gambar: Dosen "Darmanto Agustina M.T" berhasil di hapus

25.3. Menghapus Data Mahasiswa

Langkah yang diperlukan untuk menghapus data mahasiswa sangat mirip seperti data dosen. Silahkan modifikasi halaman index mahasiswa untuk menempatkan tombol "Hapus":

resources\views\mahasiswa\index.blade.php

```

1   ...
2   <tbody>
3     @foreach ($mahasiswas as $mahasiswa)
4       <tr id="row-{{$mahasiswa->id}}">
5         <th>{{$mahasiswa->firstItem() + $loop->iteration - 1}}</th>
6         <td>{{$mahasiswa->nim}}</td>
7         <td>
8           <a href="{{ route('mahasiswa.show', ['mahasiswa'=>$mahasiswa->id]) }}>
9             {{$mahasiswa->nama}}</a>
10        </td>
11        <td>{{$mahasiswa->jurusan->nama}}</td>
12        @auth
13          <td>
14            <a href="{{ route('mahasiswa.edit', ['mahasiswa' => $mahasiswa->id]) }}>
15              class="btn btn-secondary" title="Edit Mahasiswa">Edit</a>
16            <form action="{{route('mahasiswa.destroy', ['mahasiswa' => $mahasiswa->id])}}" method="POST" class="d-inline">
17              @csrf @method('DELETE')
18              <button type="submit" class="btn btn-danger shadow-none btn-hapus" title="Hapus Mahasiswa" data-name="{{$mahasiswa->nama}}>
19                Hapus</button>
20            </form>
21          </td>
22        @endauth
23      </tr>
24    @endforeach
25  </tbody>
26 ...
27 ...
28 ...

```

#	NIM	Nama Mahasiswa	Jurusan Mahasiswa	Action
1	10648094	Ani Suwarno	Sistem Informasi	Edit Hapus
2	10582671	Cawuk Mayasari	Teknik Informatika	Edit Hapus
3	10137962	Dimas Puspasari	Sistem Informasi	Edit Hapus

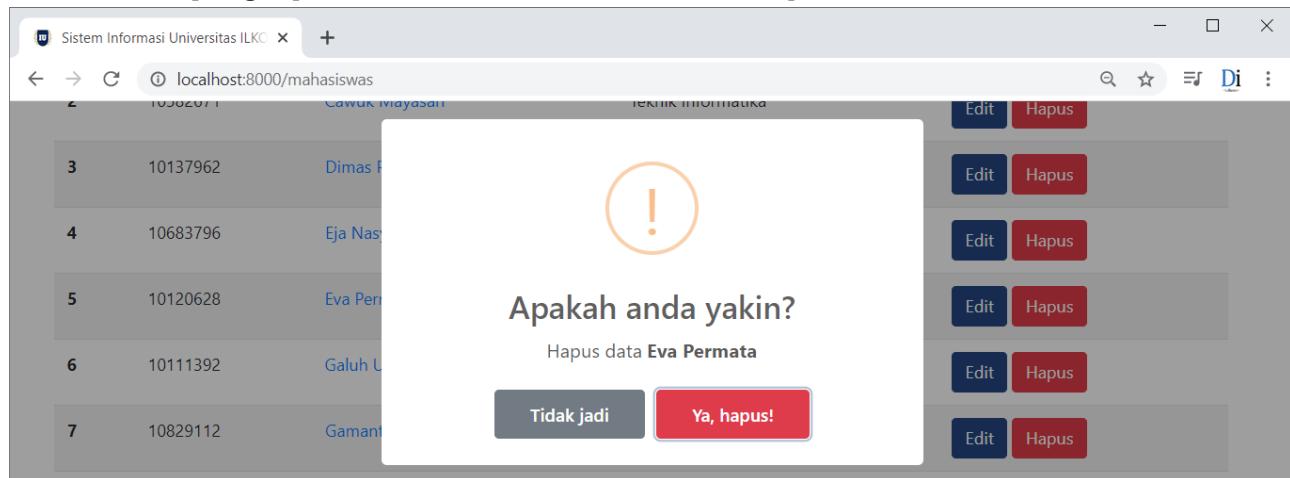
Gambar: Tombol hapus mahasiswa

Langsung saja, silahkan tambah kode berikut ke dalam method `destroy()` di file `MahasiswaController.php`:

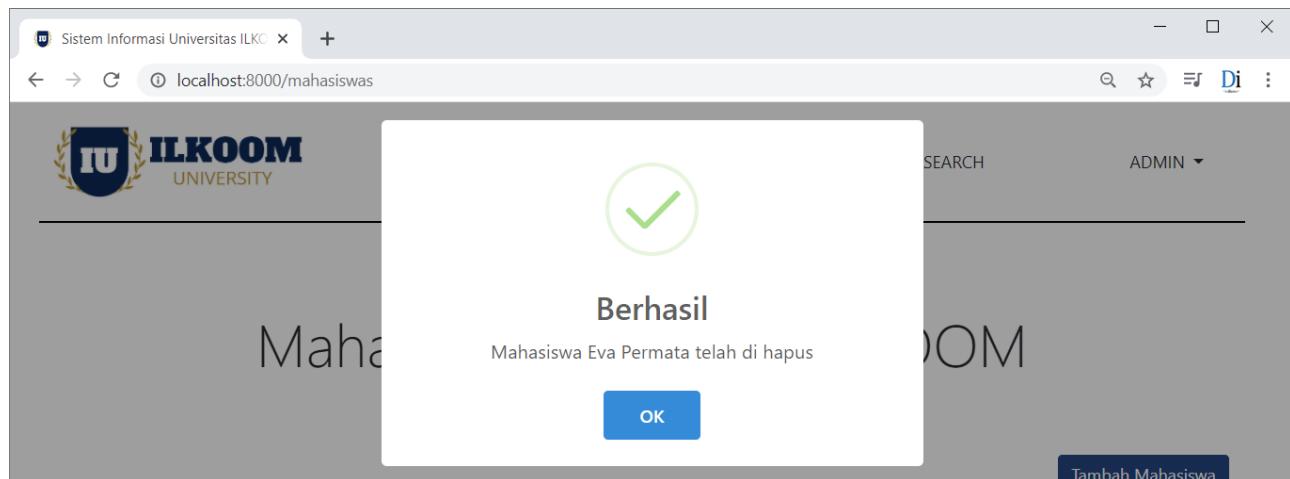
app\Http\Controllers\MahasiswaController.php

```
1 public function destroy(Mahasiswa $mahasiswa)
2 {
3     $mahasiswa->delete();
4     Alert::success('Berhasil', "Mahasiswa $mahasiswa->nama telah di hapus");
5     return redirect("/mahasiswas");
6 }
```

Dan.. selesai, penghapusan data mahasiswa sudah bisa di proses:



Gambar: Jendela konfirmasi menghapus Mahasiswa



Gambar: Mahasiswa "Eva Permata" berhasil di hapus

25.4. Menghapus Data Matakuliah

Proses penghapusan data matakuliah juga sama seperti mahasiswa. Untuk membuat tombol "**Hapus**", silahkan buka file index matakuliah dan modifikasi di bagian berikut:

Sistem Informasi Universitas ILKOOOM: Delete

resources\views\matakuliah\index.blade.php

```
1 ...  
2     <tbody>  
3         @foreach ($matakuliah as $matakuliah)  
4             <tr id="row-{{$matakuliah->id}}">  
5                 ...  
6                     <td>{{$matakuliah->jurusan->nama}}</td>  
7                     @auth  
8                         <td>  
9                             <a href="{{route('matakuliahs.edit',[ 'matakuliah'=>$matakuliah->id])}}"  
10                                class="btn btn-secondary" title="Edit Mata Kuliah">Edit</a>  
11                         <form action="{{route('matakuliahs.destroy',  
12                                [ 'matakuliah' => $matakuliah->id])}}" method="POST" class="d-inline">  
13                             @csrf @method('DELETE')  
14                             <button type="submit" class="btn btn-danger shadow-none btn-hapus"  
15                                title="Hapus Mata Kuliah" data-name="{{$matakuliah->nama}}">  
16                                 Hapus</button>  
17                         </form>  
18                     </td>  
19                     @endauth  
20             </tr>  
21         @endforeach  
22     </tbody>  
23 ...
```

Mata Kuliah Universitas ILKOOOM						
#	Kode	Nama Mata Kuliah	Dosen Pengajar	Jumlah SKS	Jurusan	Action
1	RM463	Agama dan Etika	Kuncara Purnawati M.Kom	2	Sistem Informasi	<button>Edit</button> <button>Hapus</button>
2	CR714	Algoritma & Struktur Data	Halima Mansur M.Sc	4	Sistem Informasi	<button>Edit</button> <button>Hapus</button>
3	QB773	Dasar Pemrograman	Viktor Pranowo M.Sc	1	Sistem Informasi	<button>Edit</button> <button>Hapus</button>

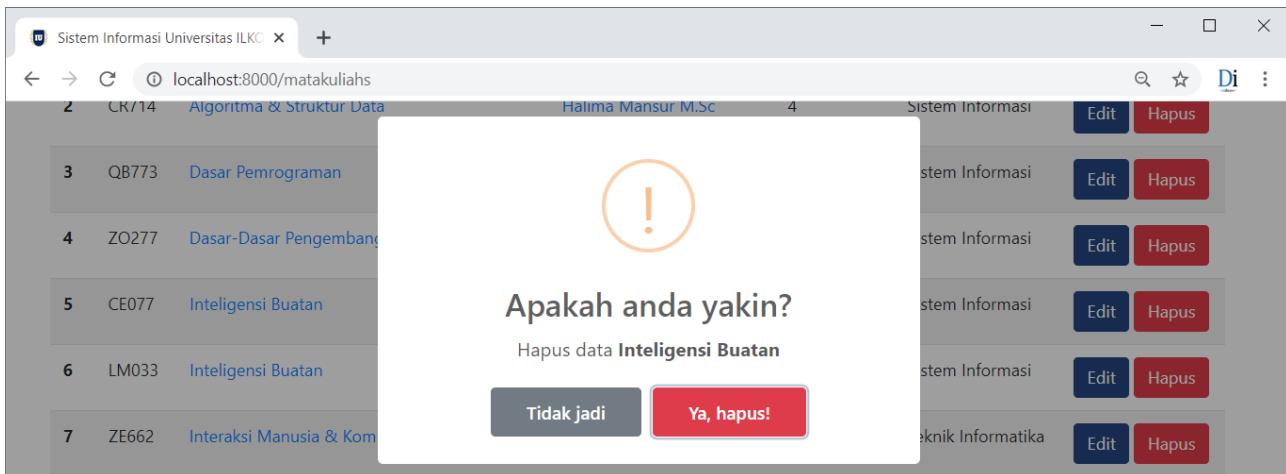
Gambar: Tombol hapus matakuliah

Lanjut, berikut kode untuk method `destroy()` di file `MahasiswaController.php`:

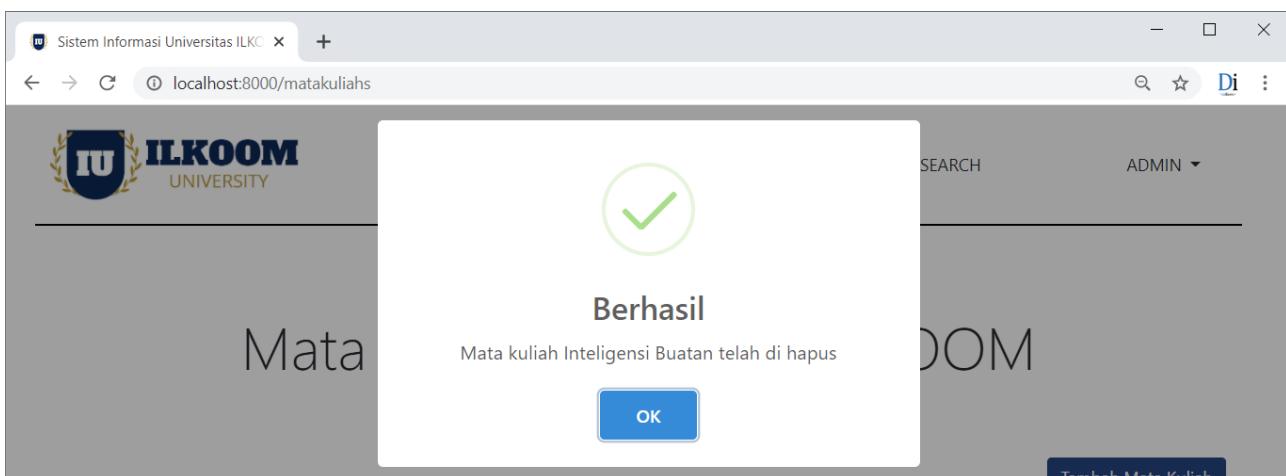
app\Http\Controllers\MatakuliahController.php

```
1 public function destroy(Matakuliah $matakuliah)  
2 {  
3     $matakuliah->delete();  
4     Alert::success('Berhasil', "Mata kuliah $matakuliah->nama telah di hapus");  
5     return redirect('/matakuliah');  
6 }
```

Berikut uji coba menghapus satu mata kuliah:



Gambar: Jendela konfirmasi menghapus Mahasiswa



Gambar: Mata kuliah "Inteligensi Buatan" berhasil dihapus

25.5. Menambah Tombol Edit dan Hapus ke Halaman Show

Dengan selesainya tombol hapus untuk data mata kuliah, maka semua proses delete sebenarnya sudah lengkap. Akan tetapi saya ingin mengedit sedikit halaman show dosen, mahasiswa dan mata kuliah untuk menambah tombol "Edit" dan tombol "Hapus".

Sebagaimana yang kita ketahui, halaman show dipakai untuk menampilkan satu data secara lengkap. Sebagai contoh, berikut tampilan halaman show dosen:

The screenshot shows a web browser window titled "Sistem Informasi Universitas ILKOOOM". The URL is "localhost:8000/dosens/7". The page header includes the ILKOOOM logo, navigation links for JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and ADMIN, and a search bar. The main content area is titled "Biodata Dosen" and displays the following information:

- Nama: **Kuncara Purnawati M.Kom**
- Nomor Induk Dosen: **99092104**
- Jurusan: **Sistem Informasi**

Mengajar Mata Kuliah:

- Agama dan Etika ([RM463](#) - 2 SKS)
- Manajemen Proyek TI ([FZ671](#) - 3 SKS)

A blue button labeled "Buat Mata Kuliah" is visible at the bottom left.

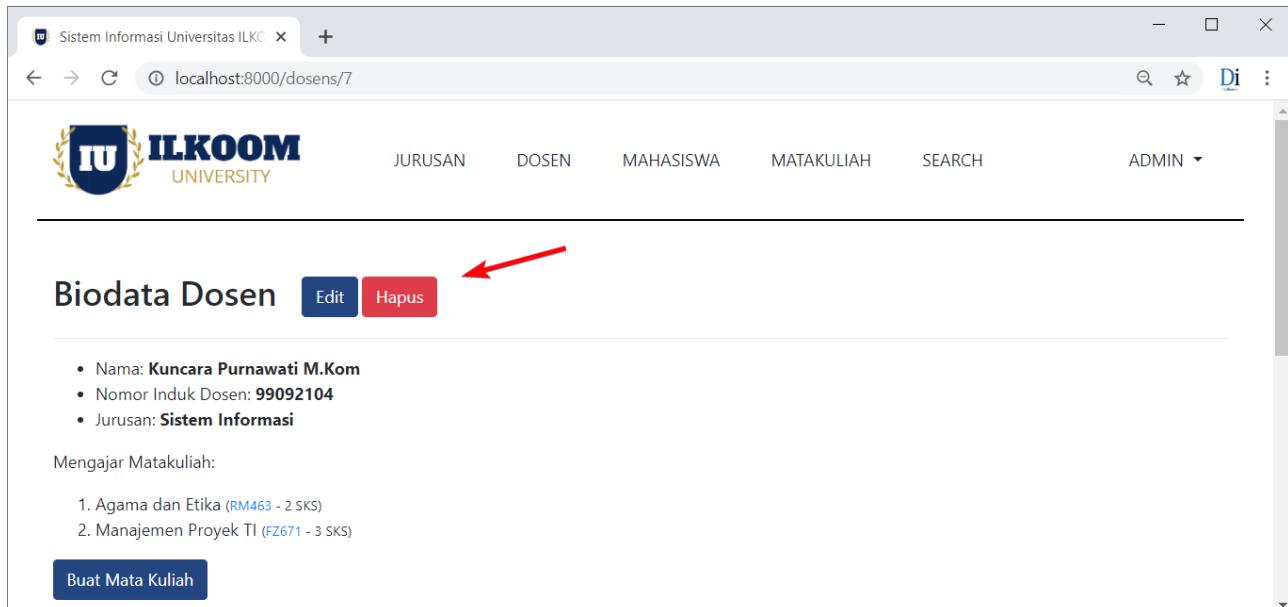
Gambar: Halaman show untuk data dosen

Saya ingin menambah tombol edit dan hapus di sebelah judul teks "Biodata Dosen". Ini akan memudahkan user jika ingin mengedit dan menghapus data dosen tanpa perlu mencarinya ke halaman index.

Kode untuk menambah tombol ini juga cukup singkat, karena kita tinggal menyamakannya dengan kode yang ada di dalam halaman index. Kita mulai dari data dosen terlebih dahulu, silahkan buka file show dosen yang berada di alamat resources\views\dosen\show.blade.php, lalu modifikasi di bagian judul halaman:

```
resources\views\dosen\show.blade.php

1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3 d-flex align-items-center">
5   <h1 class="h2 mr-4">Biodata Dosen</h1>
6   @auth
7     <a href="{{ route('dosens.edit',[ 'dosen' => $dosen->id]) }}">
8       class="btn btn-secondary mr-1" title="Edit Dosen">Edit</a>
9     <form action="{{ route('dosens.destroy', [
10       'dosen' => $dosen->id
11     ]) }}" method="POST" class="d-inline">
12       @csrf @method('DELETE')
13       <button type="submit" class="btn btn-danger shadow-none btn-hapus"
14         title="Hapus Dosen" data-name="{{ $dosen->nama }}" data-table="dosen">
15         Hapus</button>
16     </form>
17   @endauth
18 </div>
19 ...
```



Gambar: Tombol Edit dan Hapus di halaman show dosen

Di baris 4, terdapat tambahan class `.d-flex` dan `.align-items-center` ke dalam tag `<h1>`. Ini dipakai untuk mengatur tampilan tombol agar berada di samping judul "Biodata Dosen".

Kode untuk tombol edit dan hapus sama persis seperti yang ada di halaman index, termasuk penambahan blok `@auth` agar tombol ini hanya bisa diakses bagi user yang sudah login. Sedikit penambahan hanya di class `.mr-1` tombol edit untuk mengatur margin dengan tombol hapus.

Kedua tombol sudah langsung berfungsi karena semua route sudah tersedia. Khusus untuk tombol edit, setelah proses update ke database akan kembali di redirect ke halaman show, bukan ke halaman index. Inilah fungsi lain dari tag input hidden `url_asal` yang ada di dalam halaman `form.blade.php`.

Tombol yang sama akan kita tambah untuk halaman show mahasiswa dan matakuliah:

`resources\views\mahasiswa\show.blade.php`

```

1  @extends('layouts.app')
2  @section('content')
3
4  <div class="pt-3 d-flex align-items-center">
5      <h1 class="h2 mr-4">Biodata Mahasiswa</h1>
6      @auth
7          <a href="{{ route('mahasiswas.edit',[ 'mahasiswa' => $mahasiswa->id])}}"
8              class="btn btn-secondary mr-1" title="Edit Mahasiswa">Edit</a>
9          <form action="{{ route('mahasiswas.destroy',
10             [ 'mahasiswa' => $mahasiswa->id]) }}" method="POST" class="d-inline">
11              @csrf @method('DELETE')
12              <button type="submit" class="btn btn-danger shadow-none btn-hapus"
13                  title="Hapus Mahasiswa" data-name="{{ $mahasiswa->nama }}>Hapus</button>
14          </form>
15      @endauth
16  </div>

```

17 ...

The screenshot shows a web application interface for managing student data. At the top, there's a navigation bar with links for JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and ADMIN. The main content area is titled 'Biodata Mahasiswa' and shows the details for a student named Galuh Uwais with ID 10111392. Below this, there's a list of courses ('Mengambil Mata Kuliah') with three items: Dasar-Dasar Pengembangan Perangkat Lunak, Pengantar Analisis Rangkaian, and Pengembangan Aplikasi Berbasis Web. At the bottom of the list is a blue button labeled 'Ambil Mata Kuliah'. Two buttons are highlighted with red arrows: a blue 'Edit' button and a red 'Hapus' button.

Gambar: Tombol Edit dan Hapus di halaman show mahasiswa

resources\views\matakuliah\show.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="pt-3 d-flex align-items-center">
5   <h1 class="h2 mr-4">Informasi Mata Kuliah</h1>
6   @auth
7     <a href="{{ route('matakuliahs.edit',['matakuliah' => $matakuliah->id])}}"
8       class="btn btn-secondary mr-1" title="Edit Mata Kuliah">Edit</a>
9     <form action="{{ route('matakuliahs.destroy',
10        ['matakuliah' => $matakuliah->id]) }}" method="POST" class="d-inline">
11       @csrf @method('DELETE')
12       <button type="submit" class="btn btn-danger shadow-none btn-hapus"
13         title="Hapus Mata Kuliah" data-name="{{ $matakuliah->nama }}>Hapus</button>
14     </form>
15   @endauth
16 </div>
17 ...
```

The screenshot shows a web browser window for the 'Sistem Informasi Universitas ILKOOOM'. The URL is 'localhost:8000/matakuliahs/14'. The page title is 'Informasi Mata Kuliah'. At the top right, there are navigation links: JURUSAN, DOSEN, MAHASISWA, MATAKULIAH, SEARCH, and ADMIN. Below the title, there is a list of course details: Nama: Logika Informatika, Kode Mata Kuliah: DM999, Dosen Pengajar: Putu Prasetya M.Sc, Jurusan: Sistem Informasi, Jumlah SKS: 1, Total Mahasiswa: 2. There are two buttons at the bottom left: 'Edit' (blue) and 'Hapus' (red). A red arrow points to the 'Hapus' button. Below the course details, it says 'Daftar Mahasiswa:' followed by a list: 1. Labuh Hakim (10842471), 2. Najam Nugroho (10020886). At the bottom left is a blue button labeled 'Daftarkan Mahasiswa'.

Gambar: Tombol Edit dan Hapus di halaman show matakuliah

Tambahan tombol Edit dan Hapus di halaman show ini tidak wajib tapi bisa jadi alternatif pilihan seandainya tidak ingin menempatkan kedua tombol di halaman index.

Dalam bab ini kita telah menyelesaikan materi Delete untuk aplikasi Sistem Informasi Universitas ILKOOOM. Kode yang diperlukan relatif sederhana namun bisa berdampak besar karena akan menghapus data secara permanen. Oleh karena itulah jendela konfirmasi menjadi fitur wajib untuk setiap proses penghapusan data.

Berikutnya, kita akan masuk ke tahap final dari aplikasi Sistem Informasi Universitas ILKOOOM.

26. Sistem Informasi Universitas ILKOOOM: Final

Dalam bab terakhir ini kita akan masuk ke proses *finishing* untuk melengkapi hal-hal yang dirasa masih kurang untuk project Sistem Informasi Universitas ILKOOOM. Diantaranya adalah membuat menu pencarian serta pembatasan route dengan middleware.

26.1. Membuat Halaman Search

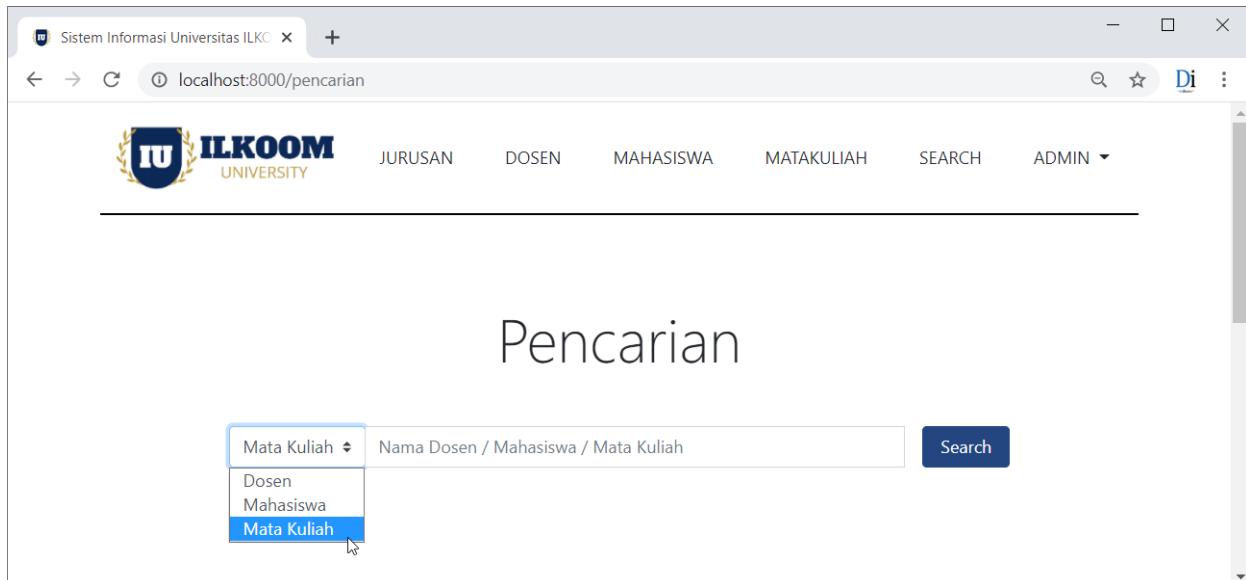
Jika diperhatikan, dalam template saat ini terdapat satu menu yang masih belum ada kode programnya, yakni menu "**Search**". Seperti yang bisa di tebak, menu ini dipakai untuk proses pencarian.

Project kita punya 4 tabel, sehingga membuat halaman pencarian menjadi tantangan tersendiri. Apakah harus menyiapkan 4 form (satu untuk masing-masing tabel)? Lalu kolom apa saja yang bisa dicari?

Agar tidak terlalu kompleks, saya akan buat beberapa batasan. Pertama, pencarian bisa dilakukan untuk 3 tabel saja, yakni tabel `dosen`, `mahasiswa` dan `matakuliah`. Untuk tabel `jurusans` tidak dirasa penting karena asumsinya project ini tidak akan memiliki banyak jurusan.

Batasan kedua, kolom yang bisa dicari hanya `nama` saja, yakni apakah itu nama dosen, nama mahasiswa atau nama matakuliah. Kita tidak bisa mencari dosen berdasarkan kolom `nid` atau mencari mata kuliah berdasarkan jumlah `skls`. Kembali, batasan ini supaya kode program kita tidak terlalu kompleks.

Berikut tampilan halaman pencarian yang akan dibuat:



Gambar: Halaman pencarian

Hasil pencarian nantinya juga akan tampil pada halaman yang sama, yakni di bagian bawah dalam bentuk tabel.

Baik, mari mulai rancang kode programnya. Silahkan tambah 2 route berikut di bagian bawah file `routes\web.php`:

`routes\web.php`

```
1 ...  
2 use App\Http\Controllers\PencarianController;  
3  
4 Route::get('/pencarian', [PencarianController::class, 'index']);  
5 Route::get('/pencarian/proses', [PencarianController::class, 'proses']);
```

Route pertama dipakai untuk menampilkan halaman form pencarian, sedangkan route kedua untuk pemrosesan form.

Dari pendefinisian kedua route, kode program akan kita tulis ke dalam method `index()` dan method `proses()` di **PencarianController**. Saat ini file `PencarianController.php` masih belum tersedia. Untuk membuatnya, buka cmd dan ketik perintah berikut:

```
php artisan make:controller PencarianController
```

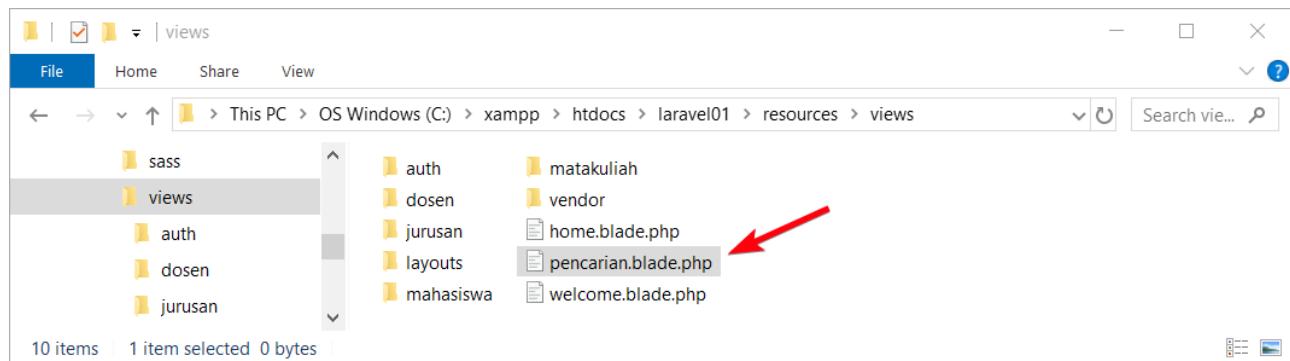
Kemudian buka file `PencarianController.php` dan tambah kode berikut ke dalam method `index()`:

`app\Http\Controllers\PencarianController.php`

```
1 <?php  
2  
3 namespace App\Http\Controllers;  
4 use Illuminate\Http\Request;
```

```
5
6 class PencarianController extends Controller
7 {
8     public function index()
9     {
10         return view('pencarian');
11     }
12 }
```

Isi method `index()` hanya sekedar perintah untuk menampilkan view 'pencarian'. File view ini juga masih belum tersedia. Oleh karena itu buat file view baru bernama `pencarian.blade.php` ke dalam folder `resources\views\`:



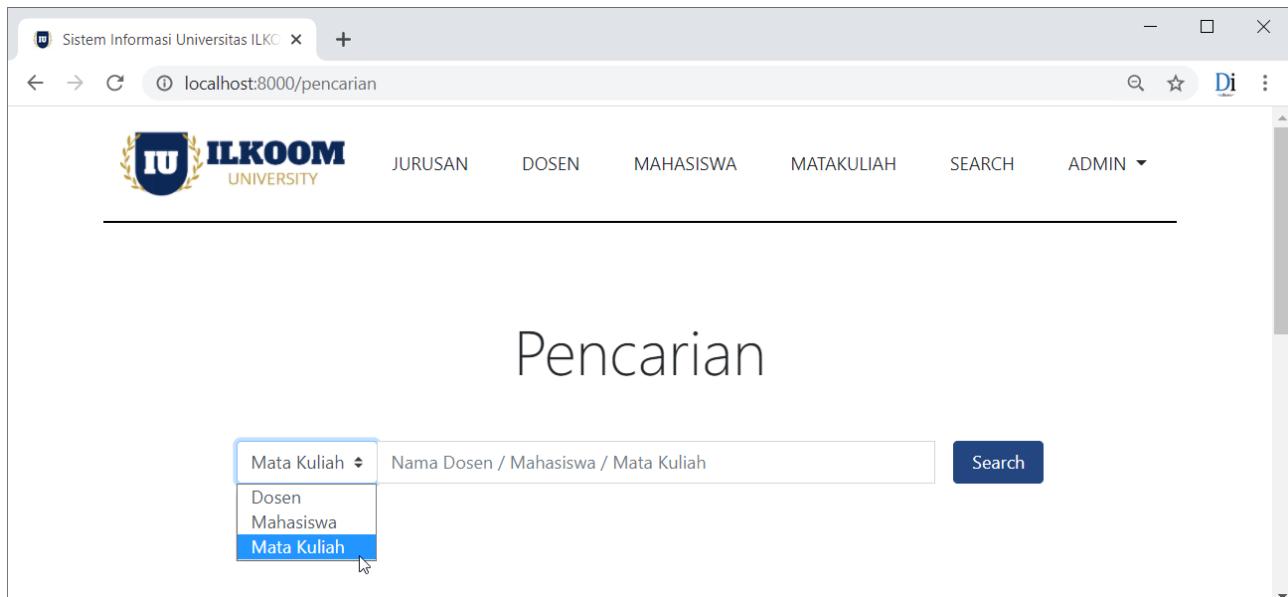
Gambar: Buat file view pencarian.blade.php di resources\views\

Buka file view tersebut dan isi dengan kode berikut:

resources\views\pencarian.blade.php

```
1 @extends('layouts.app')
2 @section('content')
3
4 <div class="container">
5     <div class="row">
6         <div class="col-12">
7
8             <h1 class="display-4 text-center my-5">Pencarian</h1>
9
10            <form class="form-inline mx-auto" action="{{ url('/pencarian/proses')}}" method="GET">
11                @csrf
12                <div class="input-group mb-3 mx-auto">
13                    <div class="input-group-prepend">
14                        <select class="custom-select" id="kategori" name="kategori">
15                            <option value="dosen"
16                                {{ ($kategori ?? '') == 'dosen' ? 'selected': ''}}>
17                                Dosen</option>
18                            <option value="mahasiswa"
19                                {{ ($kategori ?? '') == 'mahasiswa' ? 'selected': ''}}>
20                                Mahasiswa</option>
21                            <option value="matakuliah"
22                                {{ ($kategori ?? '') == 'matakuliah' ? 'selected': ''}}>
```

```
24         Mata Kuliah</option>
25     </select>
26 </div>
27 <input type="text" class="form-control" style="width:500px"
28 placeholder="Nama Dosen / Mahasiswa / Mata Kuliah"
29 name="s" id="s" value="{{ $s ?? '' }}">
30 <button type="submit" class="btn btn-primary mb-2 ml-3 px-3">
31     Search</button>
32 </div>
33
34 </form>
35
36 </div>
37 </div>
38 </div>
39
40 @endsection
```



Gambar: Tampilan halaman <http://localhost:8000/pencarian>

Untuk membuat form, saya memakai komponen **Input Group** dari Bootstrap. Komponen ini pada dasarnya hanya menyatukan beberapa inputan form agar tampil lebih rapi.

Terdapat 3 buah inputan form pada halaman ini:

1. Tag `<select name="kategori">` untuk membuat menu dropdown dengan 3 pilihan: **Dosen**, **Mahasiswa** dan **Mata Kuliah**. Setiap menu mewakili tabel yang akan dicari. Misalnya jika dipilih menu Mata Kuliah, maka proses pencarian akan dilakukan ke tabel `matakuliah`. Proses pencarian ini nantinya akan kita buat di dalam controller.
2. Tag `<input type="text" name="s">` sebagai tempat menulis teks yang akan di cari.
3. Tag `<input type="submit">` yang berfungsi sebagai tombol "**Search**" untuk memulai proses pencarian.

Setiap tag <option> pilihan tabel memiliki nilai atribut value sesuai dengan pilihan tersebut, yakni value="dosen", value="mahasiswa", dan value="matakuliah". Nilai ini nantinya akan diambil saat pemrosesan form.

Di dalam tag <option> juga terdapat sebuah pemeriksaan kondisi dengan kode berikut:

```
{{ ($kategori ?? '') == 'dosen' ? 'selected': ''}}
```

Ini dipakai untuk proses re-populate form agar nama tabel tidak kembali ke pilihan pertama ketika halaman di load. Nantinya hasil search akan ditampilkan pada halaman ini juga, maka agar "menahan" pilihan tabel tidak berubah, akan diperiksa apa isi dari variabel \$kategori. Jika berisi string 'dosen', maka tambahkan atribut 'selected'.

Hal yang sama juga dibuat untuk tag <input type="text" name="s">, dimana atribut valuenya ditulis sebagai berikut:

```
value="{{ $s ?? '' }}
```

Nantinya, nilai \$s ini dan juga variabel \$kategori akan kita kirim balik dari dalam controller.

Pada saat tombol "Search" di klik, nilai form akan dikirim ke method proses() di PencarianController. Berikut kode yang diperlukan:

app\Http\Controllers\PencarianController.php

```
1  <?php
2  ...
3  use App\Models\Dosen;
4  use App\Models\Mahasiswa;
5  use App\Models\Matakuliah;
6  ...
7  public function proses(Request $request)
8  {
9      if ($request->kategori == 'dosen') {
10          $result = Dosen::where('nama', 'LIKE', '%' . $request->s . '%')
11              ->orderBy('nama')->paginate(10);
12      }
13
14      if ($request->kategori == 'mahasiswa') {
15          $result = Mahasiswa::where('nama', 'LIKE', '%' . $request->s . '%')
16              ->orderBy('nama')->paginate(10);
17      }
18
19      if ($request->kategori == 'matakuliah') {
20          $result = Matakuliah::where('nama', 'LIKE', '%' . $request->s . '%')
21              ->orderBy('nama')->paginate(10);
22      }
23
24      return view('pencarian',[
25          'result' => $result,
26          'kategori' => $request->kategori,
27          's' => $request->s,
```

```
28     ]);
29 }
```

Method proses() berisi 3 kondisi if untuk memeriksa hasil \$request->kategori. Property \$request->kategori ini berisi nilai pilihan tag <select name="kategori"> yang dipakai untuk menentukan ke tabel mana pencarian akan dilakukan.

Sebagai contoh, jika di form pencarian user memilih kategori "**Mahasiswa**", maka proses pencarian akan dilakukan ke dalam tabel `mahasiswas` yang perintahnya ada di baris 15 – 16.

Teks yang dicari bisa diakses dari property \$request->s. Inilah yang menjadi argument dari perintah `where('nama', 'LIKE', '%'. $request->s . '%')`. Misalnya jika user sedang mencari nama 'andi', maka perintah eloquent akan menjadi `where('nama', 'LIKE', '%andi%')`.

Dalam query LIKE MySQL, tanda '%' berfungsi sebagai penanda 'wildcard' yang artinya 'nol atau banyak karakter'.

Hasil pencarian ini selanjutnya disimpan ke variabel \$result dan juga dipecah menggunakan `pagination(10)` untuk antisipasi jumlah baris yang cukup banyak.

Perintah return di baris 24 akan menampilkan ulang halaman pencarian.blade.php tapi kali ini dengan mengirim 3 nilai: 'result', 'kategori' dan 's'.

Tugas kita sekarang adalah membuat kode program di `pencarian.blade.php` untuk menampilkan hasil pencarian. Silahkan tambah kode berikut di bagian bawah file, yakni di posisi setelah tag penutup </form>:

`resources\views\pencarian.blade.php`

```
1 ...
2     </div>
3
4 </form>
5
6 @isset($result)
7     @if (count($result) == 0)
8         {{-- Artinya pencarian tidak ditemukan --}}
9         <h3 class="text-center my-5">Maaf, hasil tidak ditemukan...</h3>
10    @else
11        {{-- Tampilkan tabel hasil pencarian --}}
12        <table class="table table-striped w-auto mx-auto mt-4" id="hasil">
13            <thead>
14                <tr>
15                    <th colspan="2" class="text-center">Hasil Pencarian</th>
16                </tr>
17            </thead>
18            <tbody>
19                <tr>
20                    @if ($kategori == 'dosen')
21                        @foreach ($result as $dosen)
22                            <tr>
23                                <th>{{$loop->iteration}}</th>
```

```

24         <td><a href="{{ route('dosens.show',
25             ['dosen' => $dosen->id]) }}">
26             {{$dosen->nama}} ({{$dosen->nid }})</a>
27         </td>
28     </tr>
29     @endforeach
30     @elseif ($kategori == 'mahasiswa')
31     @foreach ($result as $mahasiswa)
32         <tr>
33             <th>{{$loop->iteration}}</th>
34             <td><a href="{{ route('mahasiswas.show',
35                 ['mahasiswa' => $mahasiswa->id]) }}">
36                 {{$mahasiswa->nama}} ({{$mahasiswa->nim }})</a>
37             </td>
38         </tr>
39     @endforeach
40     @elseif ($kategori == 'matakuliah')
41     @foreach ($result as $matakuliah)
42         <tr>
43             <th>{{$loop->iteration}}</th>
44             <td><a href="{{ route('matakuliahs.show',
45                 ['matakuliah' => $matakuliah->id]) }}">
46                 {{$matakuliah->nama}} ({{$matakuliah->kode }})</a>
47             </td>
48         </tr>
49     @endforeach
50     @endif
51     </tbody>
52   </table>
53
54   <div class="row">
55     <div class="mx-auto mt-3">
56       {{ $result->appends(['kategori' => $kategori, 's' => $s])
57           ->fragment('hasil')->links() }}
58     </div>
59   </div>
60   @endif
61   @endisset
62   </div>
63 </div>
64 </div>
65
66 @endsection

```

Seluruh tambahan kode program ada di dalam blok `@isset($result)`, dengan demikian bagian ini hanya akan diproses setelah form di submit.

Perintah `@if (count($result) == 0)` pada baris 7 dipakai untuk memeriksa apakah proses pencarian menghasilkan sesuatu atau tidak. Jika jumlah object yang ada di dalam variabel `$result` sama dengan 0, artinya data yang dicari tidak ada. Maka tampilkan teks "Maaf, hasil tidak ditemukan..." seperti berikut:



Gambar: Hasil pencarian tidak ditemukan

Namun jika kondisi `@if (count($result) == 0)` menghasilkan nilai **false**, yang berarti ada sesuatu di dalam variabel `$result`, kita masuk ke blok `@else` antara baris 11 – 59.

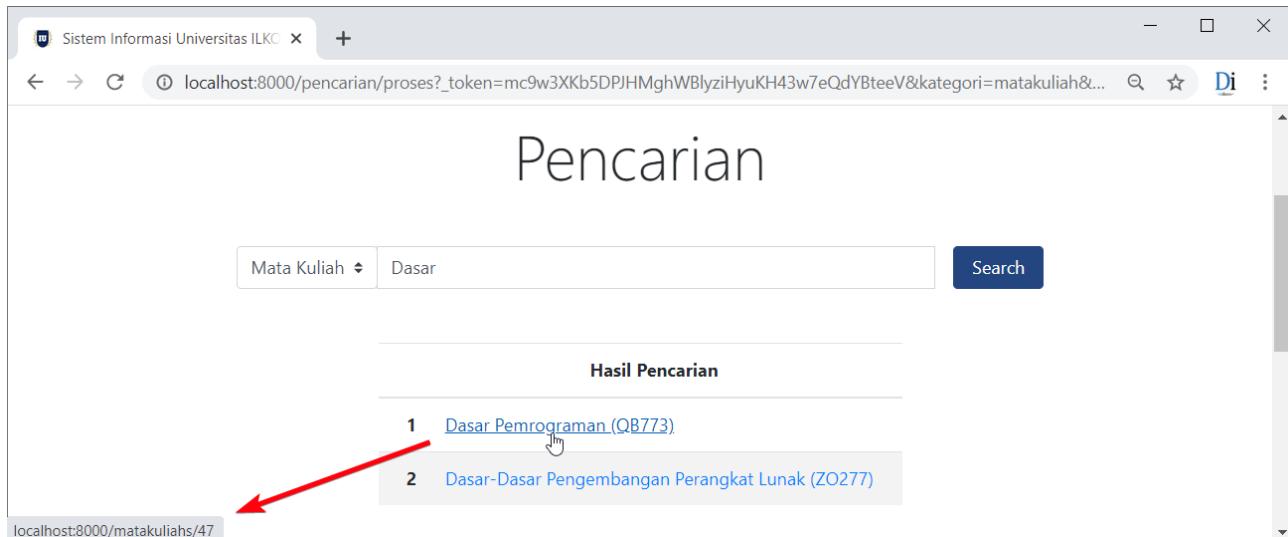
Hasil pencarian akan tampil dalam bentuk tabel. Kode pada baris 12 – 19 dipakai untuk membuat struktur header dari tag `<table>`, sedangkan isi tabel itu sendiri akan berada di dalam struktur if.

Struktur if ini diperlukan karena saya ingin menampilkan hasil pencarian dalam bentuk link. Misalnya jika user sedang mencari data mahasiswa, maka hasil mahasiswa ini bisa di klik yang akan menuju halaman show mahasiswa. Begitu jika jika user sedang mencari nama dosen, maka hasilnya bisa di klik untuk menuju halaman show dosen.

Proses menampilkan hasil pencarian dilakukan dengan perulangan `foreach` untuk membuka collection yang tersimpan dalam variabel `$result`. Isi variabel `$result` ini adalah object Model dari setiap tabel, sehingga kita bisa mengakses kolom yang sesuai. Misalnya jika berisi data dosen, maka kita bisa mengakses property `$dosen->nid` untuk menampilkan nomor NID dosen.

Di bagian bawah (baris 54 – 59), terdapat perintah untuk menampilkan link pagination. Tambahan kode `$result->appends()` dipakai agar ketika link pagination di klik, seolah-olah sama dengan menginput data form yang baru.

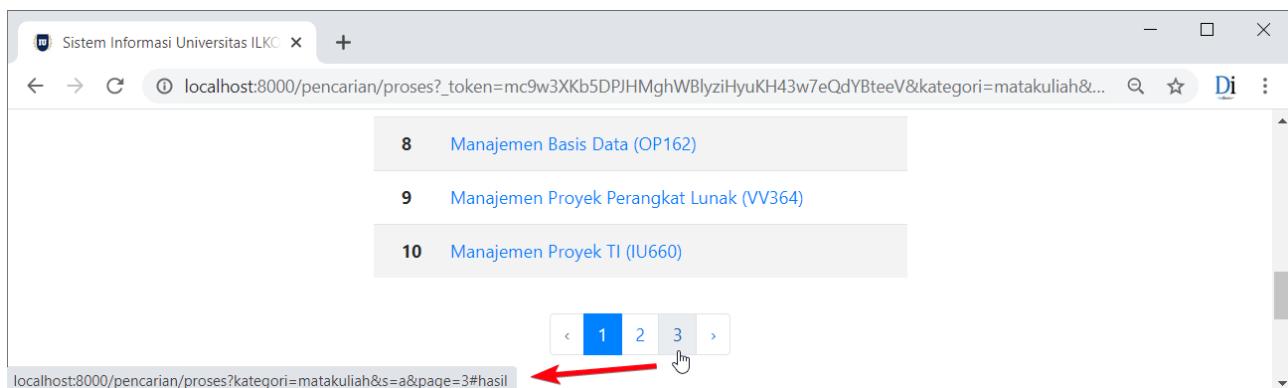
Berikut contoh hasil proses pencarian kata "Dasar" ke dalam kategori Mata Kuliah:



Gambar: Contoh Hasil Pencarian

Hasil pencarian tampil di bawah form. Ketika di klik, itu akan terhubung ke halaman show mata kuliah.

Contoh lain, berikut hasil pencarian teks 'a' untuk tabel mata kuliah:



Gambar: Pagination untuk halaman pencarian

Hasil pencarian ini akan cocok nyaris untuk semua mata kuliah. Karena jumlahnya cukup banyak, akan tampil link pagination di bagian bawah.

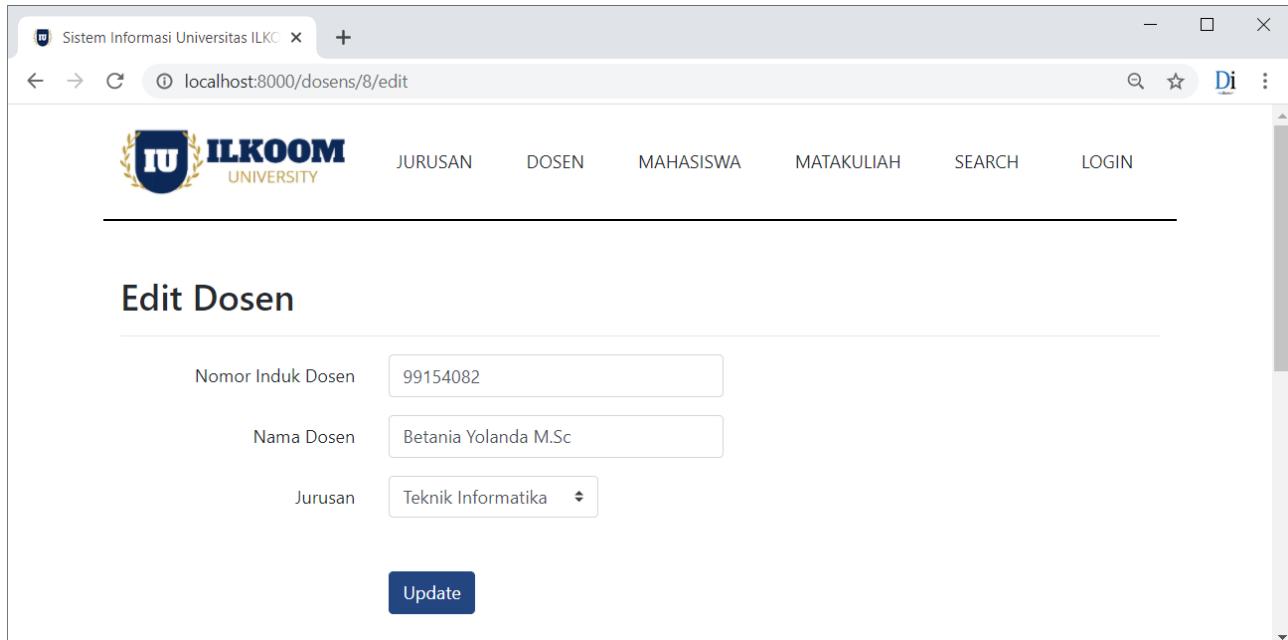
Perhatikan link untuk pagination ini. Ketika di klik, ikut dikirim nilai `kategori` dan juga nilai `s` untuk mempertahankan hasil pencarian. Pada dasarnya, ketika kita men-klik link pagination, itu akan membuka ulang halaman pencarian. Jika kedua data ini tidak disertakan, tabel hasil pencarian akan kembali kosong.

26.2. Pembatasan Route dengan Middleware

Di hampir setiap halaman, kita sudah membatasi hak akses ke tombol CRUD dengan blok `@auth`. Dengan demikian seorang user harus login terlebih dahulu untuk bisa mengakses tombol "**Tambah Dosen**" atau tombol "**Edit Mahasiswa**". Akan tetapi tidak ada yang

membatasinya jika langsung menulis alamat route tersebut.

Sebagai contoh, silahkan logout dari user saat ini dan buka halaman index dosen. Karena batasan perintah @auth, tombol Tambah, Edit dan Delete tidak akan terlihat. Namun tes ketik manual alamat <http://localhost:8000/dosens/8/edit> di web browser:



Gambar: Halaman edit masih bisa diakses

Ternyata halaman edit masih bisa diakses. Ini terjadi karena di aplikasi kita tidak ada proteksi untuk route, perintah @auth hanya membatasi tampilan tombol di level view saja.

Hal yang sama juga berlaku untuk halaman CRUD lain yang semuanya masih bisa diakses. Misalnya untuk mengedit data jurusan dengan id 2, bisa diakses dari alamat <http://localhost:8000/jurusans/2/edit>.

Untungnya, solusi dari masalah ini cukup mudah. Kita bisa batasi hak akses route dengan memakai `middleware('auth')`. Middleware ini bisa ditulis di dalam file `routes\web.php`, atau di file controller. Kali ini saya akan pakai penulisan di file controller saja.

Pembahasan lebih detail tentang middleware tersedia di buku **Laravel Uncover**.

Kita mulai dari JurusanController terlebih dahulu. Silahkan tambah kode berikut ke dalam file `JurusanController.php`:

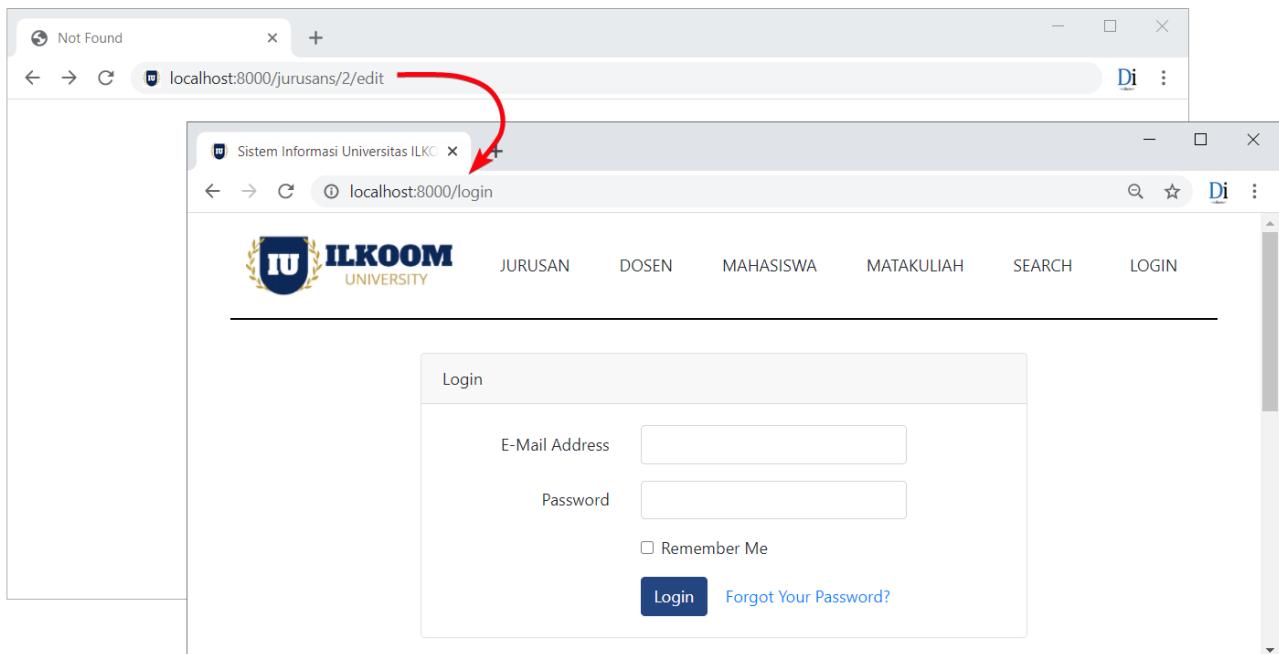
`app\Http\Controllers\JurusanController.php`

```
1 <?php
2 ...
3
4 class JurusanController extends Controller
5 {
```

```
6 // Untuk membatasi hak akses
7 public function __construct()
8 {
9     $this->middleware('auth')->except([
10         'index', 'jurusanDosen', 'jurusanMahasiswa'
11     ]);
12 }
13 ...
14 }
```

Saya menambah sebuah constructor ke dalam class **JurusanController**. Isi constructor ini sebenarnya hanya 1 perintah, yakni `$this->middleware('auth')->except([...])`. Karena menggunakan `except`, maka **selain** method `index()`, `jurusanDosen()` dan `jurusanMahasiswa()`, tidak bisa diakses secara bebas (user harus login terlebih dahulu).

Dengan kode ini, maka jika user mengetik manual alamat seperti `http://localhost:8000/jurusans/2/edit`, secara otomatis akan di redirect ke halaman login:



Gambar: User langsung di redirect ke halaman login

Untuk controller lain caranya juga sama, yakni tambah sebuah constructor yang berisi perintah `$this->middleware('auth')->except(['index', 'show'])`:

app\Http\Controllers\DosenController.php

```
1 <?php
2 ...
3
4 class DosenController extends Controller
5 {
6     // Untuk membatasi hak akses
7     public function __construct()
```

```
8      {
9          $this->middleware('auth')->except(['index', 'show']);
10     }
11     ...
12 }
```

app\Http\Controllers\MahasiswaController.php

```
1 <?php
2 ...
3
4 class MahasiswaController extends Controller
5 {
6     // Untuk membatasi hak akses
7     public function __construct()
8     {
9         $this->middleware('auth')->except(['index', 'show']);
10    }
11    ...
12 }
```

app\Http\Controllers\MatakuliahController.php

```
1 <?php
2 ...
3
4 class MatakuliahController extends Controller
5 {
6     // Untuk membatasi hak akses
7     public function __construct()
8     {
9         $this->middleware('auth')->except(['index', 'show']);
10    }
11    ...
12 }
```

Dengan batasan seperti ini, maka hanya halaman index dan show saja yang bisa diakses oleh user secara bebas.

26.3. Redirect Setelah Login

Tambahan terakhir yang akan kita buat adalah me-redirect user ketika sudah login. Saat ini setelah user login, akan di redirect ke halaman '/home' yang tidak berisi konten apapun:



Gambar: Tampilan halaman home

Saya ingin me-redirect user langsung ke halaman index jurusan, plus sedikit notifikasi login menggunakan jendela toast sweetalert.

Silahkan buka file `HomeController.php` lalu modifikasi isi method `index()` sebagai berikut:

`app\Http\Controllers\HomeController.php`

```
1 ...  
2     public function index()  
3     {  
4         toast('Selamat Datang '.auth()->user()->name, 'success')  
5             ->timerProgressBar()->autoClose(3000);  
6         return redirect("/jurusans");  
7     }
```

Dengan kode ini, begitu user login maka akan di redirect ke '/jurusans' serta akan tampil jendela toast yang langsung tertutup setelah 3 detik:



Gambar: Toast "Selamat Datang" ketika proses login

Jendela toast ini menjadi sedikit "pemanis" pada saat user masuk ke aplikasi.

Done! Aplikasi Sistem Informasi Universitas ILKOOOM sudah selesai. Semoga dari mini project ini anda bisa lebih memahami cara pembuatan aplikasi dengan banyak tabel. Fitur-fitur Laravel seperti seeder, authentication, dan eloquent relationship sangat membantu dalam menyederhanakan alur kerja program.

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. Mari dukung karya penulis negeri sendiri.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

Penutup

Akhirnya.., setelah lebih dari 600 halaman (mencakup 26 bab), kita sampai di penghujung buku **Laravel In Depth #1**. Semoga semua materi ini bisa menjadi pondasi dasar yang kuat untuk memahami apa itu eloquent relationship dan bagaimana cara penerapannya.

Seperti yang disinggung pada pengantar buku, ke depan saya berencana menulis buku Laravel In Depth #2 untuk pendalaman materi lain. Namun sembari menunggu buku tersebut, silahkan explore dokumentasi Laravel untuk mencari hal yang dirasa perlu.

Database dan proses CRUD merupakan aspek paling penting pada kebanyakan aplikasi, dan ini sudah kita bahas dalam buku ini. Topik lanjutan yang menurut saya tidak kalah penting adalah proses **authentication**, yakni pembatasan hak akses.

Di buku Laravel Uncover kita sudah membahas authentication menggunakan Laravel UI, middleware, dan juga policy. Tapi itu baru untuk satu user saja (admin dan guest). Biasanya sebuah aplikasi butuh sistem otentikasi berjenjang seperti moderator, editor, publisher, dll. Pembuatan hak akses multi level ini menjadi materi yang bisa di explorasi lebih lanjut.

Laravel LiveWire juga bisa dipertimbangkan, yang dipakai untuk membuat efek AJAX tanpa harus menggunakan JavaScript. Materi **RESTfull API**, jika ingin meng-integrasikan web berbasis Laravel dengan aplikasi lain. Tidak lupa **Laravel Jetstream** yang baru saja di perkenalkan pada Laravel 8. Semua materi ini yang sangat menarik untuk dipelajari.

Akhir kata, semoga materi yang ada di buku **Laravel In Depth #1** ini bisa bermanfaat. Mohon maaf jika ada kata-kata atau penjelasan yang salah.

Terima kasih juga atas dukungannya dengan membeli versi asli buku DuniaIlkom (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 In Depth**, saya mengumpulkan bahan dari berbagai sumber. Anda bisa mengunjungi daftar pustaka ini untuk menambah pengetahuan seputar Laravel:

- Laravel Manual: <https://laravel.com/docs/7.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>