
E-BOOK
Muhammad Azamuddin
Hafid Mukhlasin
REACT & EXPRESS
Be Fullstack Javascript Developer
KUNGFU KODING
2020

React & Express - Be Fullstack Javascript Developer

Sample Ebook https://bukureact.id

Daftar Isi

        - Mengeluarkan File-File Agar Tidak Ikut Tersimpan di Git Repository
        - Menghubungkan Express ke MongoDB Via Mongoose
            - Membuat *Database* Baru pada MongoDB
    - Membuat API products
        - Membuat Schema & Model `Product`
            - Membuat *Endpoint Create / Store* `Product`
            - Export fungsi *store* pada *controller product*
        - Membuat `router` *products*
        - Mengetest *Endpoint Store* `Product` dengan Postman
        - Menggunakan *Package* Multer untuk Menerima *Form Data*
        - Menangani Kemungkinan *Error*
        - Menangani *Error* Validasi Mongoose
        - Bentuk *Error Response*
        - Upload *Image* `Product`
            - Menyesuaikan `router` untuk *Upload Image*
        - Membuat *Endpoint* Daftar `Product`
        - Pagination
        - Membuat *Endpoint Update* `Product`
        - Membuat *Endpoint Delete* `Product`
    - Membuat API `categories`
        - Membuat *Schema & Model* `Category`
        - Membuat *Endpoint Store* `Category`
        - Membuat *Endpoint Update* `Category`
        - Membuat *Endpoint Delete* `Category`
    - Membuat API `tags`
        - Membuat *Schema & Model* `Tag`
        - Membuat *Endpoint Create* `Tag`
        - Membuat *Endpoint Update* `Tag`
        - Membuat *Endpoint Delete* `Tag`
    - Konsep *Relation One-to-one* di Mongoose
    - Membuat *Relation* antara `Product` dengan `Category`
        - Memodifikasi *Endpoint Store* `Product`
        - Memodifikasi *Endpoint Update* `Product`
    - Konsep *Relation One-to-many* di Mongoose
        - Memodifikasi *Endpoint Store* `Product`
        - Memodifikasi *Endpoint Update* `Tag`
    - Menjadikan *Endpoint* Daftar `Product` Mengembalikan `tags` dan `category`
    - Membuat *Endpoint* Daftar `Product` Bisa Difilter
        - Filter berdasarkan *keyword*
        - Filter Berdasarkan `category`
        - Filter Berdasarkan `tags`
    - Otentikasi dengan `passport` dan JSON Web Token
        - Membuat *Schema & Model* `User`
        - Validasi Email dengan *Mongoose Custom Validation*
        - Validasi Apakah Email Sudah Terdaftar
        - *Hashing* Password pada Mongoose Hooks

        - Fitur
        - Mockup
            - Mockup Halaman Register
            - Mockup Halaman Login
            - Mockup Halaman Home
            - Mockup Halaman Akun
            - Mockup Halaman Daftar Alamat
            - Mockup Halaman Riwayat Pemesanan
            - Mockup Halaman Checkout - Items
            - Mockup Halaman Checkout - Pilih Alamat
            - Mockup Halaman Checkout - Konfirmasi
            - Mockup Halaman Invoice
    - Persiapan
        - Mengimport Data dari MongoDB untuk Server Web API
        - Mengimport Data Images ke Aplikasi Express
        - Menjalankan Server Web API
        - Membuat Proyek React Baru
        - Konfigurasi Tailwind
            - Install `tailwindcss` & `autoprefixer`
            - Install `postcss`
            - Membuat `scripts` di `package.json`
            - *Import* `src/styles/tailwind.css` pada `src/index.js`
        - Membuat *File* Konfigurasi Global `src/config.js`
    - Membuat Halaman *Home*
        - Membuat Menu Item
        - Membuat Layout Awal Halaman Home
    - Membuat Redux *Store*
        - Instalasi *package* yang diperlukan
        - Membuat *Store*
        - Membungkus Kode Aplikasi dengan `<Provider>`
    - Redux *State* `Auth`
        - Membuat *file-file* yang Diperlukan
        - Logika *State* Login
        - Membuat Actions Creator `login` & `logout`
        - Mendaftarkan *reducer* `Auth` ke `store`
        - Menyimpan *State* `Auth` ke *Local Storage*
        - Memuat Nilai Awal *State* `Auth` dari *Local Storage*
    - Membuat API *helper* `auth`
    - Membuat Halaman Register
        - Membuat Tampilan Form
        - Validasi Input Form dengan `react-hook-form`
        - Memproses Data ke Web API
        - Membaca Status Pemrosesan ke Server dan Jadikan Indikator
            - Membuat Halaman Pendaftaran Berhasil
        - Menambahkan Link ke Halaman Login
        - Menambahkan `Logo` pada Form Register

- Membuat Halaman Login
- Membuat Komponen `TopBar`
  - Menggunakan Komponen `<TopBar>` pada Halaman Home
- Redux *State* `Products`
  - Gambaran Umum *State* `Products`
  - Membuat *file-file* Redux yang Diperlukan
  - Menangani *Action* pada `Products` *reducer*
  - *Action Status Fetching* `Products`
    - *Action Fetching* `Products` dari API
    - *Action* Menentukan Halaman yang Aktif
    - *Action* Menentukan Keyword Filter
    - *Action* Menentukan Kategori Produk yang Aktif
    - *Action* Menentukan Tags yang Aktif
    - *Action* Toggle Tag
    - *Action* Halaman Berikutnya
    - *Action* Halaman Sebelumnya
  - Mendaftarkan *State* `Products` ke Redux *store*
- Menampilkan Daftar Produk
  - Pagination produk
  - Indikator Loading
  - Filter Produk Berdasarkan *Keyword*
  - Filter *keyword* berdasarkan Kategori
  - Filter Produk Berdasarkan Tags
- Fitur Keranjang Belanja / `Cart`
  - Gambaran Umum *State* `cart`
  - Membuat _File-file- yang Diperlukan
  - Menangani *Action*
    - *Action* Tambah Item
    - *Action* Hapus Item
    - *Action* Bersihkan Items
    - *Action* Set Items
  - Mendaftarkan *reducer* ke Redux *store*
  - Sync ke *Local Storage*
  - Menyimpan Data `cart` ke Web API
  - Memuat Data `cart` dari Web API
- Membuat Komponen `<Cart>`
  - Menampilkan Item Cart
  - Menambah Item dari Daftar Produk
  - Menambah dan Mengurangi Item dari Keranjang Belanja
  - Tombol Checkout
  - Merapikan Tampilan Komponen `<Cart>`
  - Hitung Subtotal Harga Item
- Fitur Kelola Alamat Pengiriman
  - *Helper* API `address`
  - Custom Hook `useAddressData`
  - Membuat Komponen `<SelectWilayah>`

Mengenal Web Service

Pendahuluan

Sebelum kita bekerja dengan web service maka alangkah baiknya kita mengenal terlebih dahulu apa itu web service. Pada bab pertama ini kita akan membahas hasl-hal fundamental yang perlu kita fahami sebelum membangun aplikasi web service.

Definisi Web Service

Web service merupakan aplikasi yang didesain untuk mendukung interkoneksi & interaksi antar aplikasi berbasis web yang terhubung melalui jaringan komputer. Web service diperlukan karena masing-masing aplikasi bisa jadi memiliki format data yang berbeda, ditulis dengan bahasa pemrograman yang berbeda, dan berjalan pada platform berbeda. Oleh karenanya diperlukan media yang standard untuk dapat saling berkomunikasi, media inilah yang kemudian disebut sebagai web service.

Banyak orang menyebut web service dengan sebutan API, dan menganggap keduanya adalah barang yang sama.

API singkatan dari *application programming interface* atau antarmuka pemrograman aplikasi merupakan sekumpulan perintah atau fungsi yang dapat digunakan untuk berinteraksi dengan sistem tertentu. Pada mulanya sistem yang dimaksud adalah sistem operasi (sehingga API merupakan fungsi-fungsi yang dapat digunakan untuk berinteraksi dengan sistem operasi komputer), namun kemudian istilah ini menjadi generic untuk sistem apapun.

Berdasarkan uraian tersebut dapat disimpulkan bahwa web service merupakan API untuk sistem aplikasi berbasis web.

RESTful Web Service

Salah satu standard dalam web service yang umum digunakan adalah REST. REST merupakan singkatan dari *REpresentational State Transfer* yaitu standard pertukaran resource dalam arsitektur web melalui protokol tertentu, umumnya HTTP. Resource bisa berupa data, dokumen, image, fungsi dsb.

REST pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000 melalui disertasinya. Berikut beberapa konsep dari REST:

- REST berbasis client-server. REST memisahkan antara bagian server yang bertugas menyediakan jalur untuk mengakses resource, dan bagian client melakukan akses resource dan kemudian menggunakannya. Pemisahan ini akan memberikan fleksibilitas baik dari sisi portabilitas dan skalabilitas sistem.
- REST bersifat stateles. Setiap request dari client ke server bersifat independen, tidak boleh ada penyimpanan konteks atau state pada server atas suatu request. Sebuah state pada konsep REST hanya boleh disimpan pada client, sehingga ketika melakukan request ke server bisa menyertakan informasi state tersebut sehingga request tersebut dikenali oleh server.
- Resource pada REST dapat dicache. Setiap resouce yang berasal dari respon REST server bisa diset sebagai resource yang bisa di-cache maupun tidak. Jika resource tersebut bisa di-cache maka client boleh menggunakan resource sebelumnya.

- REST memiliki interface yang standard. REST menggunakan konsep URI (Uniform Resource Identifier) sebagai interface untuk dapat mengakses suatu resource pada REST server.
- REST server bisa berperan sebagai layer perantara. Resource pada REST server tidak harus berasal dari server yang sama dengan REST server, bisa juga berasal dari server lain atau bahkan sistem lain, sehingga dalam hal ini REST server berperan sebagai jembatan perantara antara REST client dengan sistem lain.

Web service yang dibuat dengan konsep REST inilah yang kemudian dikenal sebagai RESTful Web Services.

Mengenal NodeJS

Pendahuluan

Setelah memahami tentang dasar-dasar web service maka selanjutnya kita akan mengenal salah satu tools yang digunakan untuk membangun aplikasi web service.

Mengenal NodeJS

Sebagaimana yang telah kita ketahui bahwa Javascript merupakan bahasa pemrograman di sisi client yang hanya bisa berjalan pada web browser. Hal itu karena pada browser tersebut telah disematkan sebuah engine yang berfungsi menjalankan kode Javascript.

Berangkat dari konsep tersebut, menginspirasi Ryan Dahl pada tahun 2009 untuk menciptakan sebuat tools yang memungkinkan kita menjalankan Javascript di luar browser. Tools inilah yang kemudian diberinama NodeJS.

Engine yang digunakan oleh NodeJS untuk mengkompilasi dan menjalankan Javscript di luar browser adalah adalah V8 buatan Google. Engine V8 yang dibuat dengan menggunakan bahasa C++ ini sebenarnya merupakan engine yang juga digunakan pada browser Google Chrome. Dengan cara ini maka kemudian Javascript bisa dijalankan pada platform apapun.

Nah, jadi NodeJS bukan tergolong bahasa pemrograman ya!. Lebih tepatnya adalah runtime environment untuk menjalankan kode Javascript.

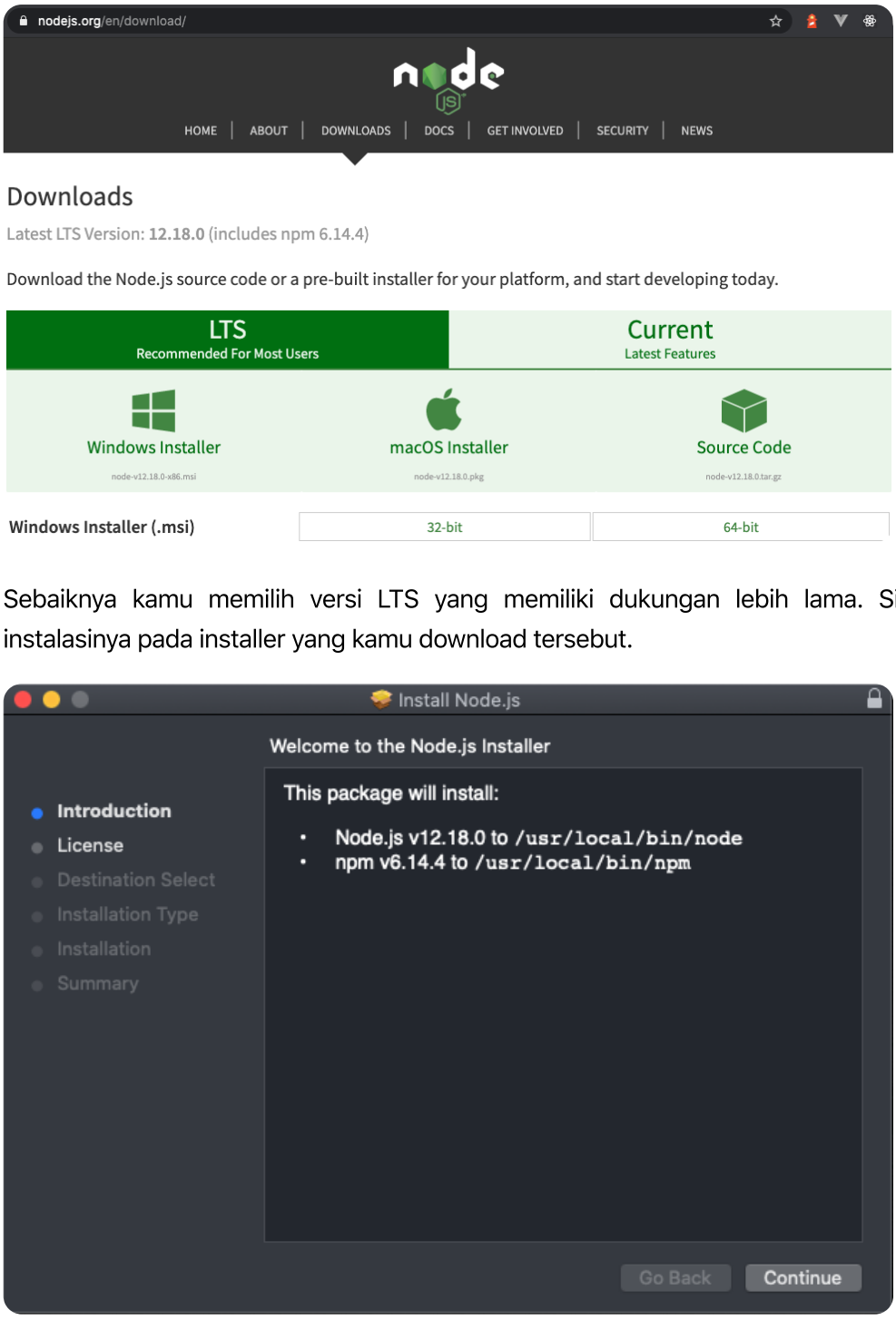
Bagi programmer Javascript yang awalnya hanya bisa bermain di ranah frontend, kehadiran NodeJS ini tentu sangat menggembirakan. Hal ini berarti bahwa untuk membuat kode aplikasi backend bisa juga dengan menggunakan bahasa yang sama yaitu Javascript.

Saat buku ini ditulis versi terbarunya adalah versi 14 namun versi LTS (Long Term Support) terbaru adalah versi 12 dengan codename erbium.

Selengkapnya kamu kunjungi website resminya yaitu https://nodejs.org/ atau https://github.com/nodejs/node/releases.

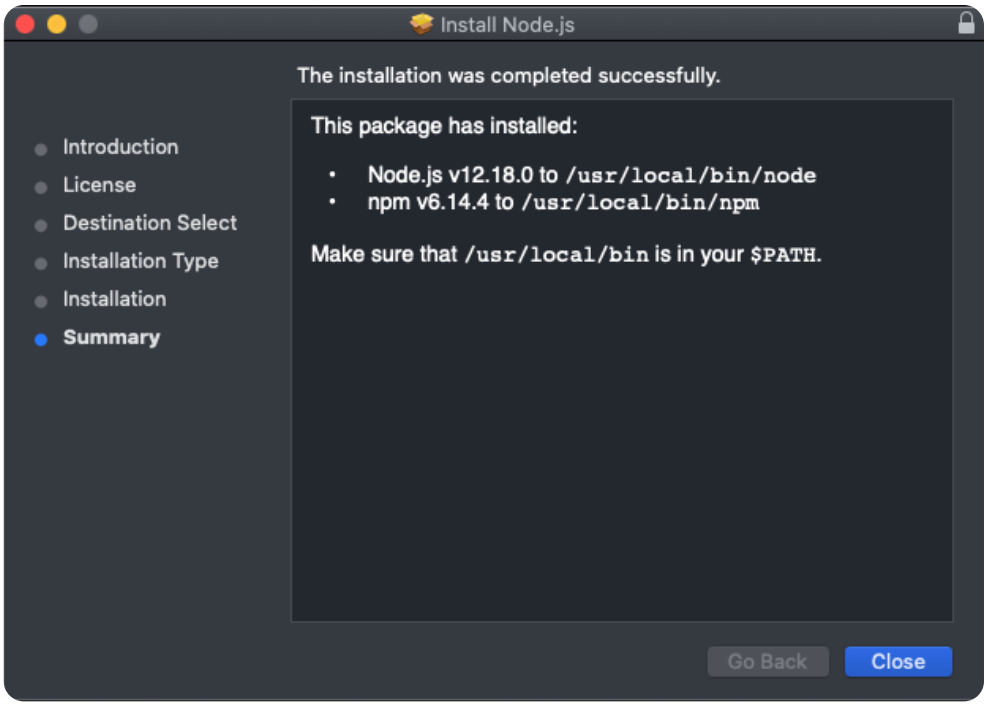
Instalasi NodeJS

Silakan downland installer NodeJS yang disediakan pada web resminya https://nodejs.org/en/download, sesuaikan dengan sistem operasi yang kamu gunakan.



Sebaiknya kamu memilih versi LTS yang memiliki dukungan lebih lama. Silakan ikuti saja panduan instalasinya pada installer yang kamu download tersebut.

tldr;



Jika sudah selesai instalasi, maka kita bisa menguji apakah instalasi yang kita lakukan sukses atau tidak, dengan menjalankan perintah `node -v` pada terminal:

```
node -v
v12.18.0
```

Pada contoh ini, versi yang terinstalasi pada komputer penulis adalah versi 12.18.0 LTS.

Mengenal Framework ExpressJS

Pendahuluan

Pada bagian ini, kita akan mengupas salah satu framework yang cukup populer yaitu ExpressJS.

Mengenal ExpressJS

"Fast, unopinionated, minimalist web framework for Node.js" demikian tagline yang diusung oleh framework ini pada situs resminya https://expressjs.com/.

ExpressJS merupakan framework web berbasis NodeJS yang memiliki berbagai fitur-fitur dasar untuk membangun aplikasi web, baik itu fullstack maupun web service. Fitur-fitur tersebut diantaranya: routing, middleware, dan template engine.

Di samping popularitasnya yang tinggi sehingga banyak resource yang mendukungnya. Kelebihan lain dari framework ini adalah simple, minimalist dan to the point, artinya kita bisa membuat sesuatu dengan kode yang lebih singkat. Hal ini tentu akan memudahkan kita mempelajarinya. Framework ini cukup solid sehingga menjadi dasar dari banyak framework berbasis NodeJS lainnya.

Saat buku ini ditulis, versi stabil terbaru dari framework ini adalah versi 4, adapaun versi 5 masih tahap alpha. Sehingga materi pada buku ini akan merujuk ke versi 4.

Mengenal MongoDB

Pendahuluan

Database yang akan kita gunakan untuk mengembangkan web service pada buku ini adalah MongoDB. Oleh karena itu pada bab ini kita akan membahas dasar-dasar dari database dan mongoDB supaya kita lebih mudah dalam mengembangkan aplikasi web service berbasis database mongoDB ke depannya.

Mengenal Database

Database atau basis data merupakan kumpulan data yang disimpan di dalam komputer secara sistematik sehingga kelak dapat dengan mudah digunakan kembali. Jadi yang disimpan di dalam database itu berupa data.

Apa itu data?

Data merupakan fakta yang mewakili suatu obyek seperti manusia, hewan, peristiwa, konsep, keadaan dan sebagainya, yang dapat dicatat dan mempunyai arti yang implisit.

Sebuah database memiliki setidaknya satu tabel/file, dan pada setiap tabel/file tersebut memiliki beberapa record.

Dalam dunia nyata database ibarat sebuah lemari arsip, lemari tersebut memiliki beberapa bagian atau laci, dimana setiap laci berisi dokumen arsip. Jika kita sandingkan dengan konsep database maka lemari arsip tersebut adalah database, laci merupakan tabel atau file sedangkan dokumen adalah record.

Mungkin tanpa kita sadari, selama ini kita tak bisa lepas dari database, minimal manfaat database telah kita rasakan atau gunakan. Misalnya database sebagai backbone penyimpanan data pada aplikasi yang sering kita gunakan seperti aplikasi social media (di mana data postingan status kita disimpan?), aplikasi transportasi online (gojek, grab), aplikasi toko online (di mana data pesanan kita disimpan?), dll

Tujuan & Manfaat Database

Tujuan utama adanya database tentu adalah untuk menyimpan data. Namun database juga memiliki beberapa manfaat: diantaranya:

- memudahkan pengelolaan data karena berada pada tempat khusus,
- memudahkan akses data karena terdapat bahasa SQL yang universal untuk berinteraksi dengan data pada database, memudahkan pengamanan data karena terdapat mekanisme untuk membatasi hak akses,
- memudahkan sharing data karena database yang berbasis server bisa diakses melalui jaringan komputer, dsb.

Komponen Sistem Database

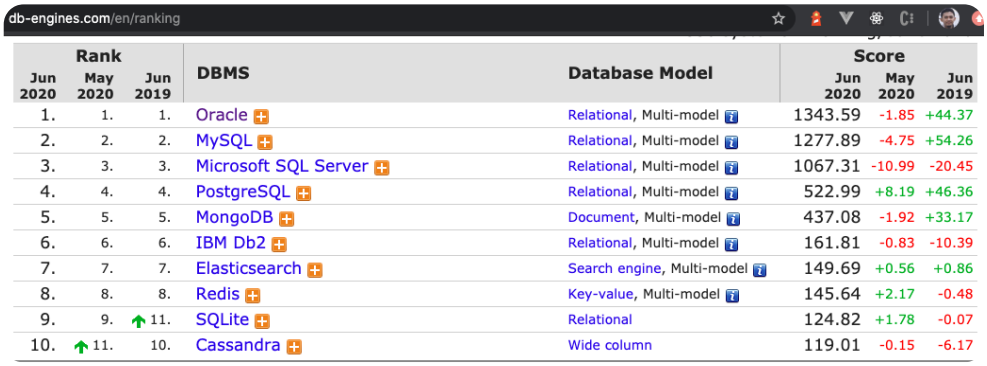
Pada sebuah sistem database terdapat beberapa lima komponen penyusunnya yaitu:

- Hardware (perangkat keras berupa komputer atau server tempat dimana file database tersebut disimpan)
- Software (perangkat lunak atau aplikasi yang akan menjalankan database tersebut, sepert: operating system & DBMS software)
- Data (data yang disimpan di dalam database itu sendiri)
- Procedure (prosedur untuk mengelola database)
- People (pengguna database baik itu database administrator, programmer, maupun pengguna lainnya yang memanfaatkan data pada database)

Database Management System (DBMS)

DBMS adalah software atau perangkat lunak untuk menjalankan dan mengelola database, contoh: MySQL, PosgreSQL, MongoDB, SQL Sever, Oracle, dll.

Berdasarkan web https://db-engines.com/en/ranking, kita bisa melihat DBMS yang diurutkan berdasarkan popularitasnya.



Rank			DBMS	Database Model	Score		
Jun 2020	May 2020	Jun 2019			Jun 2020	May 2020	Jun 2019
1.	1.	1.	Oracle	Relational, Multi-model	1343.59	-1.85	+44.37
2.	2.	2.	MySQL	Relational, Multi-model	1277.89	-4.75	+54.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1067.31	-10.99	-20.45
4.	4.	4.	PostgreSQL	Relational, Multi-model	522.99	+8.19	+46.36
5.	5.	5.	MongoDB	Document, Multi-model	437.08	-1.92	+33.17
6.	6.	6.	IBM Db2	Relational, Multi-model	161.81	-0.83	-10.39
7.	7.	7.	Elasticsearch	Search engine, Multi-model	149.69	+0.56	+0.86
8.	8.	8.	Redis	Key-value, Multi-model	145.64	+2.17	-0.48
9.	9.	↑ 11.	SQLite	Relational	124.82	+1.78	-0.07
10.	↑ 11.	10.	Cassandra	Wide column	119.01	-0.15	-6.17

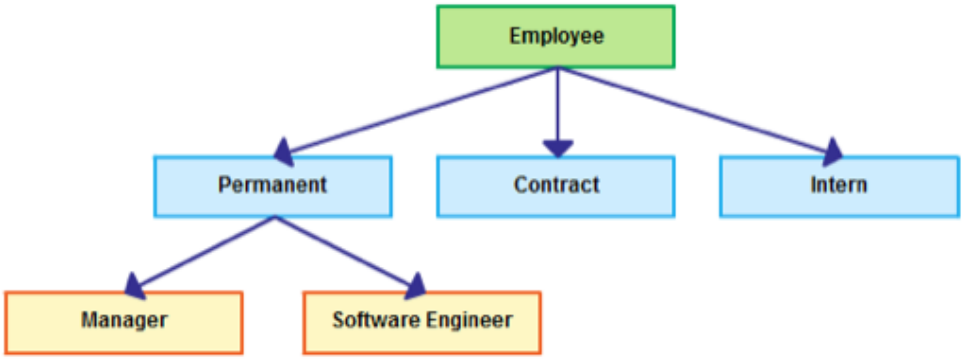
Model Database

Jika kita perhatikan pada tabel rangking popularitas database diatas terdapat kolom `database model`, nah apa itu database model atau model database?

Model database ini terkait dengan mekanisme penyimpanan data pada database. Terdapat beberapa mekanisme penyimpanan data pada database, diantaranya:

- Hierarchical Database

Mekanisme penyimpanan data pada model ini menggunakan konsep hirarki data yaitu kumpulan datanya dihubungkan satu sama lain melalui hubungan berdasarkan pointer yang membentuk struktur pohon.

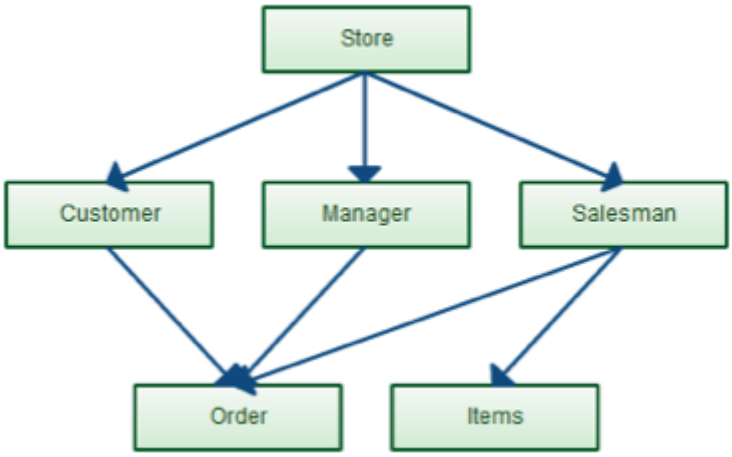


> Catatan: konsep ini berkembang tahun 1960-an dan tidak digunakan lagi.

- Network Database

Mekanisme penyimpanan data pada model ini menggunakan konsep jaringan yaitu kumpulan recordnya dihubungkan melalui pointer yang membentuk relasi antar record dalam bentuk ring.

Model ini sebenarnya bermaksud menyempurnakan konsep database hirarki. Namun model ini memiliki banyak kelemahan, yaitu tidak memungkinkannya relasi many to many. Fleksibiltas dalam menambah atau menyisipkan record sangat rendah dan kompleks.



> Catatan: konsep ini berkembang tahun 1960-an dan tidak digunakan lagi.

- Relational Database (RDB)

Model ini menyempurnakan dua model sebelumnya. Konsep relasional mengharuskan data terstruktur dan disimpan dalam bentuk tabel-tabel, dimana tabel-tabel tersebut dapat saling berhubungan (relasi).

Model database relasional ini mulai muncul sekitar tahun 1970-an.

Model ini fokus pada konsep ACID:

- Atomicity: Suatu proses selesai secara menyeluruh/tidak.
- Consistency: Semua proses (transaction) yang terjadi di database harus memiliki state yang jelas. Setiap data yang disimpan harus memenuhi semua constraint, Cascades, dan triggers.
- Isolation: Sebuah proses tidak boleh mempengaruhi proses lain.

- Durability: Proses yang telah selesai harus bersifat permanen, walau aplikasinya di restart.

Contoh DBMS yang menggunakan model ini adalah: MySQL, Ms SQL Server, Ms Access, Oracle dll.

> Catatan: Saat ini umumnya aplikasi menggunakan model ini.

- NoSQL Database

NoSQL merupakan singkatan dari Not Only SQL. Model ini menyimpan data pada database tidak dalam wujud tabular sebagaimana pada relasional database namun berbentuk selain itu seperti: dokumen, key-value dan graph.

Model database NoSQL ini mulai muncul sekitar tahun 2000-an, bersama dengan berkembangnya internet dan big data.

Contoh DBMS yang menggunakan model ini:

- Document (e.g. MongoDB, CouchDB),
- Key-Value (e.g. Redis),
- Graph (e.g. Neo4J, OrientDb), dsb

Kelebihan dari database NoSQL ini adalah dari sisi kecepatan dan fleksibilitasnya.

Mana model database yang cocok untuk kita? apakah relasional atau noSQL?

Jawabannya adalah tergantung kebutuhan pada bisnis proses aplikasi yang kita bangun serta data yang disimpan.

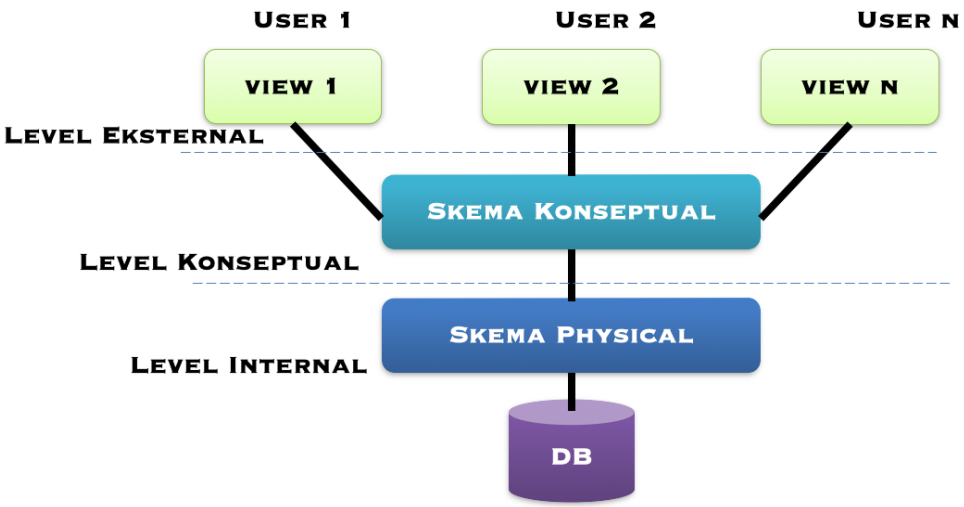
Jika data yang disimpan adalah data terstruktur, yang memiliki aturan data tertentu, memiliki batasan relasi, memiliki skema yang kompleks, maka sebaiknya menggunakan relasional database.

Namun jika sebaliknya, kita membutuhkan fleksibilitas penyimpanan data, data yang disimpan non tabular, tidak ada batasan relasi yang rumit, maka kita bisa menggunakan noSQL.

Kabar baiknya adalah, beberapa DBMS relasional juga mulai mengadopsi fitur-fitur yang dimiliki noSQL database, demikian juga sebaliknya.

Arsitektur Database

Berdasarkan arsitekturnya, database tersusun atas tiga level lapisan, yaitu:



1. Level External

Level ini ditujukan untuk memberikan kemudahan bagi user dlm mengakses database tanpa harus mengetahui mekanisme pengaksesan secara rinci. Pengguna hanya dapat mengakses bagian tertentu dari database untuk alasan perlindungan dan keamanan data

2. Level Konseptual

Level ini menggambarkan database secara keseluruhan yg merupakan gabungan informasi dari berbagai sudut pandang pengguna terhadap database.

Dalam bentuk Entity Relationship Diagram, meliputi:

- Semua entitas (tabel), atribut dan relasi
- batasan-batasan data
- Integritas dan keamanan data
- Informasi semantik tentang data

3. Level Internal

Level ini menggambarkan bagaimana data disimpan dalam media penyimpanan komputer secara fisik, yaitu:

- Alokasi ruang media penyimpanan utk data & indeks
- Deskripsi ukuran record yang disimpan
- Penempatan record
- Kompresi & enkripsi data

Mekanisme pengaksesan data biasanya diatur oleh sistem operasi & DBMS.

Bahasa Database

Bahasa atau perintah yang digunakan untuk berinteraksi dengan database disebut dengan Query atau nama panjangnya adalah Structured Query Language (SQL).

Berdasarkan fungsinya, query dikelompokkan menjadi tiga jenis:

- Data Definition Language (DDL). Digunakan untuk mengedit struktur tabel, membuat database/tabel, menghapus database/tabel
- Data Manipulation Language (DML). Digunakan untuk menampilkan, menambah, mengedit, dan menghapus record (baris) data
- Data Control Language (DCL). Digunakan untuk mengatur hak akses pengguna terhadap database.

Contoh perintah query untuk menampilkan data users yang levelnya `member`.

```
SELECT * FROM users WHERE level = 'member'
```

Perintah diatas akan menampilkan semua data dari tabel `users` dengan kriteria levelnya bernilai `member`.

Perintah query ini secara umum dapat digunakan pada semua DBMS namun secara spesifik tergantung dari masing-masing DBMS terutama terkait fitur khusus dan karakterister modelnya.

Interaksi ExpressJS & MongoDB

Pendahuluan

Pada bab ini kita akan belajar tentang interaksi antara ExpressJS dengan database MongoDB seperti: pustaka apa yang digunakan, konfigurasi koneksinya seperti apa, bagaimana cara menjalankan perintah query pada kode ExpressJS kita, dsb.

Sebelum memulai materi ini, pastikan kamu telah menjalankan server mongodb (mongod) terlebih dahulu. Jika kamu lupa bagaimana cara menjalankannya, maka silakan baca kembali materi tentang ini pada bab sebelumnya.

Untuk mengecek apakah server MongoDB sudah jalan atau belum, maka jalankan perintah:

```
mongo --nodb
```

Kemudian untuk mempraktekkan materi pada bab ini, kita akan menggunakan atau melanjutkan projek `express-app` yang telah dibahas pada bab 3.

Instalasi & Konfigurasi

Untuk berinteraksi dengan MongoDB, kita membutuhkan tools perantara yang perlu kita instalasi pada aplikasi ExpressJS kita. Tools tersebut berupa sebuah pustaka yaitu `node-mongodb-native` yang kebetulan dibuat sendiri secara official oleh MongoDB.

Repo dari pustaka ini bisa kamu jumpai pada tautan ini `https://github.com/mongodb/node-mongodb-native`.

Instalasi Pustaka

Sebagaimana pustaka JS lain, kita bisa menginstalasinya dengan menggunakan perintah npm berikut:

```
npm install mongodb
```

Setelah proses instalasi selesai maka untuk menggunakan pustaka ini pada NodeJS atau ExpressJS cukup dengan me-require-nya melalui kode berikut: `require('mongodb')`.

> Catatan: jika kita melakukan instalasi standard MongoDB maka seharusnya tidak ada konfigurasi khusus.

Menguji Instalasi

Untuk memastikan bahwa instalasi yang kita lakukan berhasil maka kita bisa mencobanya dengan membuat kode koneksi ke server database MongoDB.

```
/* file: mongodb.js */
const MongoClient = require("mongodb").MongoClient
// const connectionString = "mongodb://localhost:27017"; // tanpa
authentication
const connectionString =
"mongodb://user_latihan:123456@localhost:27017?authSource=admin"

MongoClient.connect(connectionString, { useUnifiedTopology: true },
(error, client) => {
  if (error) return console.error(error)
  console.log("Server database connect!")
})
```

> Catatan: silakan sesuaikan username dan password database server MongoDB, pada contoh ini menggunakan `user_latihan:123456`. Parameter `?authSource=admin` digunakan untuk menunjukkan dimana letak authentication disimpan.

Silakan jalankan file ini pada terminal dengan perintah `node mongodb.js`

```
$ node mongodb.js
Server database connect!
```

Jika berhasil maka pada terminal akan muncul teks `Server database connect!`, namun jika tidak, berarti gagal terkoneksi ke server database mongodb, pastikan bahwa server mongodb sudah berjalan.

Interaksi ReactJS & ExpressJS

Pendahuluan

Pada bab ini kita akan belajar tentang interaksi antara ReactJS dan web service yang dibangun menggunakan ExpressJS.

Adapun web service yang akan kita gunakan untuk mempraktikkan tutorial ini adalah web service yang telah kita buat pada bab sebelumnya yaitu aplikasi express-app.

Terdapat 5 endpoint yang akan kita gunakan:

1. GET products ~ list products
2. GET product/:id ~ single product
3. POST product ~ create new product
4. PUT product/:id ~ update product
5. DELETE product/:id ~ delete product

Instalasi & Konfigurasi

Untuk praktik pada bab ini, kita akan menggunakan tiga pustaka yaitu CRA, React Router dan Axios. CRA dan React Router telah kita bahas pada bab fundamental, sedangkan pustaka Axios belum.

Instalasi & Konfigurasi CRA

Silakan install CRA menggunakan perintah `npx create-react-app nama_folder_app`, maka otomatis akan menginstall CRA versi terbaru.

```
npx create-react-app react-express-app
```

Pada contoh ini, penulis menggunakan nama folder `react-express-app`. Jika instalasi selesai maka masuk ke folder aplikasi dan jalankan aplikasi `npm run start` (atau jika menggunakan yarn maka `yarn start`)

```
cd react-express-app
npm run start
```

Maka pada console akan keluar tampilan kurang lebih sebagai berikut:

```
Compiled successfully!

You can now view react-express-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.2:3000
```

```
Note that the development build is not optimized.
To create a production build, use yarn build.
```

Dan otomatis browser akan terbuka dengan alamat tersebut.

Muncul tampilan default aplikasi berbasis CRA.

Nah, supaya tidak bertabrakan dengan web service yang juga menggunakan port 3000, maka kita ganti saja aplikasi `react-express-app` ini ke port lain misalnya 4000. Caranya cukup mudah dengan membuat file `.env` pada root folder projek yang berisi sebagai berikut:

```
PORT = 4000
```

Kemudian restart aplikasi dan pastikan saat ini aplikasi menggunakan port baru.

Instalasi React Router

Pustaka react router digunakan untuk menangani routing aplikasi kita. Misal route untuk menampilkan list product, route untuk membuat product baru, route untuk menampilkan single product, dsb.

```
npm install react-router-dom
```

Instalasi & Konfigurasi Axios

Pustaka Axios merupakan pustaka populer yang digunakan untuk berkomunikasi dengan web service. Axios memiliki banyak feature diantaranya: dapat digunakan pada browser dan NodeJs, mendukung browser lama melalui XMLHttpRequest, mendukung Promise API, otomatis mengubah response ke JSON, dsb.

Alternatif lain yang bisa digunakan untuk komunikasi dengan web service adalah menggunakan fungsi Javascript fetch.

```
npm install axios
```

Untuk menggunakan pada aplikasi ReactJS kita maka kita cukup dengan mengimportnya maka kemudian fungsi Axios otomatis bisa digunakan.

```
const Axios = require('axios')

// atau
```

```
import Axios from 'axios'
```

Karena berbasis promise maka kita bisa menggunakan Axios dengan dua cara yaitu promise dan async/await.

Cara promise:

```
Axios.get('http://localhost:3000/products')
  .then( (response) => {
    // handle success
    console.log(response);
  })
  .catch( (error) => {
    // handle error
    console.log(error);
  })
  .then( () => {
    // always executed
  });
```

Cara async/await:

```
const getProducts = async () => {
  try {
    const response = await Axios.get('http://localhost:3000/products');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
getProducts()
```

Studi kasus Express

Overview Projek

Coba bayangkan, kita mendapatkan tugas untuk membuat aplikasi dari klien atau dari perusahaan tempat kita bekerja.

Dan kita diberikan informasi *user stories*, dari *user stories* tersebut kita akan melakukan analisa untuk menentukan API apa saja yang perlu dibuat.

User Stories

Kita menerima *user stories* yang dibagi berdasarkan tipe pengguna aplikasi yang akan kita buat.

Berikut *user stories* nya:

sebagai ...	**saya ingin ...**	**sehinga ...**
guest	melihat daftar makanan	bisa memilih makanan
guest	mendaftar dan masuk	bisa memesan makanan
customer	melihat daftar makanan	bisa memilih makanan
customer	melihat nama & harga makanan	
customer	memfilter daftar makanan berdasarkan kategori	
customer	memfilter makanan berdasarkan tags	

sebagai ...	saya ingin ...	sehinga ...
customer	mencari produk berdasarkan nama produk	
customer	memasukan item ke keranjang belanja	
customer	mengubah jumlah pesanan per makanan	
customer	melihat total harga di keranjang belanja sebelum checkout	mendapatkan perkiraan yang akan dibayar
customer	melakukan checkout	pesanan saya diproses
customer	memilih atau menambahkan alamat pengiriman saat checkout	pesanan dikirim ke tujuan yang benar
customer	melihat invoice	bisa melakukan pembayaran
customer	melihat riwayat pesanan	bisa melihat detail pesanan dan statusnya
customer	mengelola alamat pengiriman	bisa menambahkan lebih dari satu alamat pengiriman

Mengidentifikasi Entitas & Atributnya

Entitas merupakan sesuatu yang merepresentasikan sebuah objek dalam aplikasi. Dari entitas tersebut kita bisa tahu kira-kira tabel apa saja yang perlu kita buat di *database*.

Dan kemudian kita bisa membuat *Model* apa saja yang diperlukan oleh aplikasi kita.

Jika kamu belum tahu apa itu *Model*, saya jelaskan sedikit tentangnya. *Model* merupakan representasi dari tabel di dalam database jika di RDBMS.

Misalnya adalah setiap aplikasi hampir pasti punya tabel *user*, sehingga di dalam aplikasi kita memerlukan *model User*. Terutama jika kita mengadopsi konsep MVC / *Model View Controller*.

Meskipun kita tidak akan menerapkan MVC secara utuh, karena untuk API kita tidak memerlukan *View*.

Kita tetap akan mengadopsi *Model* dan *Controller* agar sumber kode kita tersusun lebih rapi.

Baik, mari kita lanjutkan untuk mengidentifikasi entitas dari *user stories* yang sudah kita terima.

Untuk mengidentifikasi entitas kita cukup baca lagi *user stories* dan tandai setiap kita menemukan sebuah objek yang penting untuk aplikasi kita.

Misalnya, dari *user story* berikut:

> Sebagai guest saya ingin melihat daftar makanan

Dari cerita di atas kita menemukan dua objek yang penting yaitu **guest** dan **makanan**.

Kemudian kita bertanya lagi, apakah **guest** bisa mewakili sebuah tabel di database? atau bisa kita jadikan *model* tersendiri? Jawabanya tidak, jadi sepertinya objek **guest** ini lebih cocok menjelaskan tipe dari entitas.

Kita bisa tahu bahwa **guest** merupakan tipe pengguna sehingga kita menemukan entitas pertama kita yaitu **user**.

Berikutnya adalah **makanan**, apakah **makanan** bisa mewakili sebuah *model* di aplikasi? Iya, sehingga kita menemukan entitas kedua yaitu **makanan**.

Bisa saja kita menyebut entitas yang kita identifikasi dengan nama lain yang lebih general, misalnya kita akan menyebut **makanan** dengan **product** supaya lebih umum.

Kita sudah menemukan dua entitas dari *user story* pertama, yaitu:

- user
- product

Kita akan melanjutkannya dengan cara seperti di atas sehingga kita bisa menandai setiap calon entitas ini dengan huruf tebal seperti ini:

sebagai ...	**saya ingin ...**	**sehinga ...**
guest	melihat daftar **makanan**	bisa memilih makanan
guest	mendaftar dan masuk	bisa memesan makanan
customer	melihat daftar makanan	bisa memilih makanan
customer	melihat **nama** & **harga** makanan	
customer	memfilter daftar makanan berdasarkan **kategori**	
customer	memfilter makanan berdasarkan **tags**	
customer	mencari produk berdasarkan nama produk	
customer	memasukan item ke **keranjang belanja**	
customer	mengubah jumlah **pesanan** per makanan	
customer	melihat total harga di keranjang belanja sebelum checkout	mendapatkan perkiraan yang akan dibayar
customer	melakukan checkout	pesanan saya diproses
customer	memilih atau menambahkan **alamat pengiriman** saat checkout	pesanan dikirim ke tujuan yang benar
customer	melihat **invoice**	bisa melakukan pembayaran
customer	melihat riwayat pesanan	bisa melihat detail pesanan dan **status**nya

sebagai ...	saya ingin ...	sehinga ...
customer	mengelola alamat pengiriman	bisa menambahkan lebih dari satu alamat pengiriman

Sehingga kita dapatkan daftar calon entitas berikut:

- user
- product
- nama
- harga
- kategori
- tag
- keranjang belanja
- alamat pengiriman
- pesanan
- invoice
- status pesanan

Setelah kita lihat lagi ternyata ada yang tidak cocok menjadi entitas tetapi lebih pas menjadi atribut, yaitu:

- nama (nama produk)
- harga (harga produk)
- status (status pesanan)

Sehingga sekarang kita mendapatkan daftar entitas berikut:

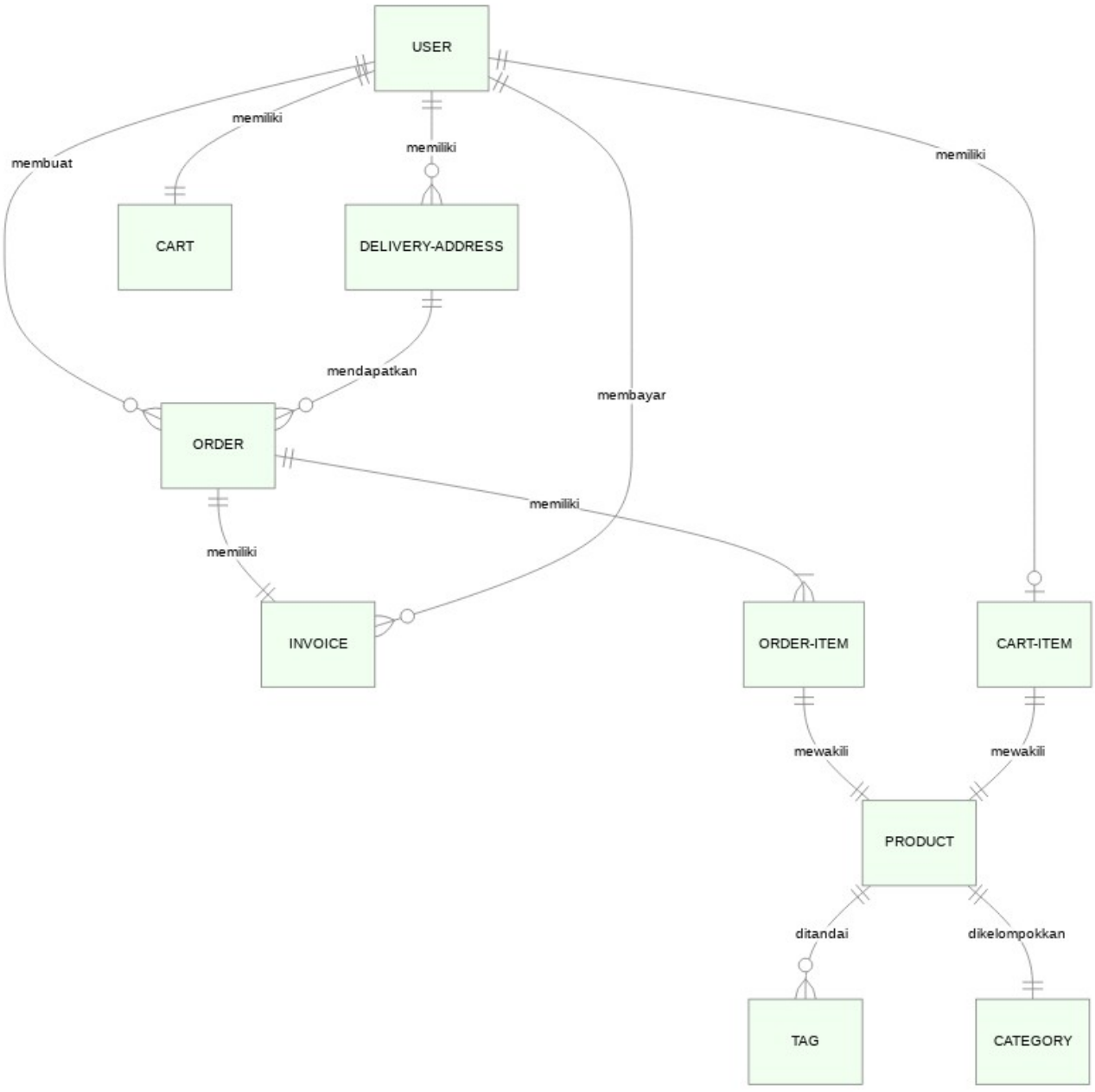
- user
- product
- kategori
- tag
- keranjang belanja
- alamat pengiriman
- pesanan
- invoice

Untuk keseragaman, mari kita ubah semua nama entitas di atas agar menggunakan bahasa inggris seperti ini:

- user
- product
- category
- tag
- cart
- delivery address
- order
- invoice

Membuat *Entity Relationship Diagram*

Selanjutnya dari daftar entitas di atas dan *user stories* kita akan coba membuat *entity relationship diagram* nya. Seperti ini:



Dari ERD di atas, kamu menyadari ada dua entitas baru yang belum kita identifikasi sebelumnya, yaitu `order item` dan `cart item`. Kedua entitas tersebut berfungsi sebagai perantara antara `order` dengan `product` dan antara `user` dengan `product`.

Kenapa kita memerlukannya? karena setiap `order` akan memiliki beberapa item `product` yang di order, di dalamnya terdapat informasi jumlah yang dipesan untuk masing-masing `product`.

Sehinga kita tidak akan mereferensikan langsung antara `order` ke `product` atau antara `user` ke `product`.

Saat `order` dibuat oleh user, sistem akan menyalin informasi `product` ke masing-masing `order item` sehingga ketika `product` diubah atau dihapus dari sistem, data `order` tidak terpengaruh karena telah memiliki salinan informasi `product` pada saat `order` dibuat.

Begitu pula dengan `cart item`, merupakan salinan informasi dari `product`, bedanya `cart item` disimpan di dalam `cart`.

Selain itu, baik `order item` atau `cart item` juga akan menyimpan informasi jumlah / kuantitas masing-masing `product` yang tentu saja informasi ini tidak ada di dalam entitas `product`.

Kamu mungkin bertanya, kenapa saya tidak melihat entitas `cart` di dalam ERD di atas? jawabannya karena ternyata kita tidak memerlukannya. Kita bisa langsung mereferensikan setiap `cart item` langsung ke `user`.

Karena dalam perjalanannya di aplikasi, seorang `user` hanya memiliki satu `cart` dan kita tidak memiliki informasi lainnya yang perlu disimpan tersendiri di dalam `cart`.

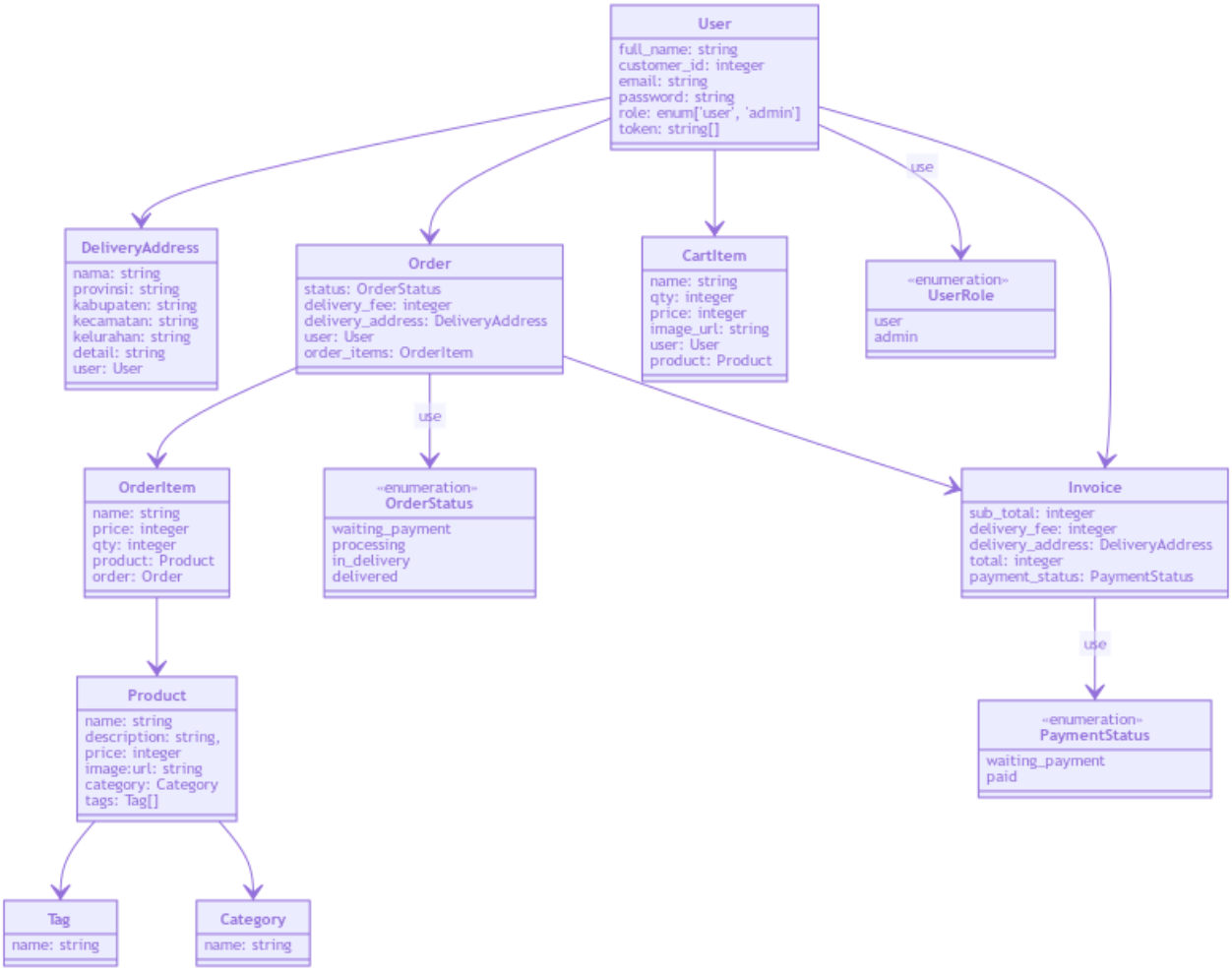
Secara konsep `cart` tersebut tetap ada, tetapi secara entitas di aplikasi, kita tidak memerlukannya. Kita bisa menampilkan `cart` dari `user` hanya dengan mencari `cart item` yang dimiliki oleh `user` tersebut.

Dengan penambahan dua entitas di atas yaitu `order item` dan `cart item` serta penghapusan entitas `cart`, sekarang daftar entitas kita menjadi seperti berikut:

- user
- product
- category
- tag
- delivery address
- order
- invoice
- order item
- cart item

Entitas-entitas di atas yang kemudian akan kita buat *Model* di dalam aplikasi kita.

Selanjutnya kita perlu memikirkan untuk masing-masing entitas di atas, atribut apa yang dimilikinya.



Dari mana kita bisa tiba-tiba mengetahui atribut yang diperlukan untuk tiap-tiap entitas? Yang pertama tentu dari *user stories*, tetapi terkadang *user stories* saja tidak akan cukup, karena *user stories* hanya menangkap alur umum saja.

Untuk itu perlu dilakukan diskusi dengan pemilik proses bisnis atau dengan klien yang meminta dibuatkan aplikasi. Dengan begitu kita akan bisa menangkap apa saja atribut yang diperlukan.

Akan lebih baik lagi jika di tim kita sudah ada *Business Analyst* sehingga kita bisa mendapatkan sistem desain yang diperlukan darinya. Dan kita bisa berfokus untuk menerjemahkannya ke dalam kode.

Untuk lebih detailnya saya akan perjelas dengan tabel untuk masing-masing entitas.

> Pada diagram di atas kita tidak mendefinisikan field `id` sebagai primary untuk masing-masing entitas. Hal itu karena MongoDB memiliki default primary key dengan field `_id` yang menggunakan UUID dan bukan auto increment. Oleh karena itu kita juga memerlukan `customer_id` untuk menyimpan nomor unique autoincrement yang mudah dibaca untuk mengidentifikasi user.

Entitas User

atribut	tipe	deskripsi

atribut	tipe	deskripsi
full_name	string	nama lengkap dari user
customer_id	string	nomor *unique auto increment* untuk mengidentifikasi user
email	string	email user
password	string	password user
role	enum UserRole	peran dari user, nilai yang dibolehkan adalah "user" atau "admin"
token	array string	berisi array token, token ini digunakan untuk otentikasi dan otorisasi user

Entitas DeliveryAddress

atribut	tipe	deskripsi
nama	string	nama atau label dari alamat pengiriman, misalnya alamat rumah, alamat kantor
provinsi	string	cukup jelas
kabupaten	string	cukup jelas
kecamatan	string	cukup jelas
kelurahan	string	cukup jelas
detail	string	keterangan lebih detail tentang alamat, misalnya dekat indomart dll

Entitas Order

atribut	tipe	deskripsi
status	enum OrderStatus	status dari pesanan, berisi *enumeration* OrderStatus yaitu *waiting payment*, *processing*, *in_delivery*, *delivered*

Entitas OrderItem

atribut	tipe	deskripsi
name	string	nama item, diambil dari produk saat order dibuat
price	integer	harga item, diambil dari produk saat order dibuat
qty	integer	jumlah item yang diorder
product	Product	mereferensikan entitas Product terkait

atribut	tipe	deskripsi
order	Order	mereferensikan entitas Order terkait

Entitas CartItem

atribut	tipe	deskripsi
name	string	nama item, diambil dari produk saat user menambahkan produk
price	integer	harga item, diambil dari produk saat user menambahkan produk
qty	integer	jumlah item yang diorder
image_url	string	image url dari produk
user	User	mereferensikan entitas User pemilik cart item
product	Product	mereferensikan entitas Product terkait

Entitas Product

atribut	tipe	deskripsi
name	string	nama produk
description	string	deskripsi produk
price	integer	harga produk
image_url	string	string yang digunakan untuk menyimpan nama image produk
category	Category	mererensikan entitas Category untuk produk
tag	array Tag	mereferensikan entitas Tag untuk produk, bisa lebih dari satu

Entitas Tag

atribut	tipe	deskripsi
name	string	nama tag

Entitas Category

atribut	tipe	deskripsi
name	string	nama category

Entitas Invoice

atribut	tipe	deskripsi
sub_total	integer	nilai total dari item yang dipesan (tidak termasuk delivery fee)
delivery_fee	integer	nilai ongkos kirim
delivery_address	DeliveryAddress	berisi data alamat pengiriman sesuai field pada entitas DeliveryAddress
total	integer	subtotal + delivery_fee
payment_status	enum PaymentStatus	status pembayaran dari invoice ini, nilai yang dibolehkan adalah *waiting payment* dan *paid*

Mengidentifikasi API yang dibutuhkan

Berdasarkan *user stories*, *ERD* dan *Class Diagram* yang sudah dibuat di atas, sekarang kita identifikasi API endpoint yang diperlukan.

entitas	method	route	deskripsi
Product	GET	/products/	dapatkan daftar produk
	POST	/products	buat produk baru
	PUT	/products/:id	update produk berdasarkan parameter `id`
	DELETE	/products/:id	hapus produk berdasarkan parameter `id`
Category	GET	/categories	dapatkan daftar kategori
	POST	/categories	buat kategori baru
	PUT	/categories/:id	update kategori berdasarkan parameter `id`
	DELETE	/categories/:id	hapus kategori berdasarkan parameter `id`
Tag	GET	/tags	dapatkan daftar tag
	POST	/tags	buat tag baru
	PUT	/tags/:id	update tag berdasarkan parameter `id`
	DELETE	/tags/:id	hapus tag berdasarkan parameter `id`
DeliveryAddress	GET	/delivery-addresses	dapatkan daftar alamat pengiriman user
	POST	/delivery-addresses	buat alamat pengiriman baru
	PUT	/delivery-addresses/:id	update alamat pengiriman berdasarkan parameter `id`

entitas	method	route	deskripsi
	DELETE	/delivery-address/:id	hapus alamat pengiriman berdasarkan parameter `id`
Order	GET	/orders	dapatkan daftar order milik user
	POST	/orders	buat order baru
CartItem	GET	/carts	dapatkan daftar items di keranjang belanja dari user
	PUT	/carts	update items di keranjang belanja user

Studi Kasus React

Pendahuluan

Pada bab ini kita akan melanjutkan aplikasi pemesanan makanan, sebelumnya kita sudah membuat Web API / Web *service* sebagai *backend*. Dan sekarang kita akan membuat *frontend* menggunakan ilmu yang sudah kita pelajari pada *ebook* Modern React maupun pada bab-bab sebelumnya di Ebook ini.

Gambaran Umum Proyek

Proyek *frontend* ini tentu saja akan menggunakan React, kita akan menggunakan `create-react-app` atau CRA untuk menginisiasi pembuatan proyek React tersebut.

Karena kita sudah membuat *web API* sebelumnya dengan Express, maka sekarang kita tinggal berfokus bagaimana mengambil data dari web API tersebut dan menampilkannya dengan menarik. Dan juga menghubungkan setiap fitur yang ada di web API supaya bisa digunakan oleh pengguna.

Fitur

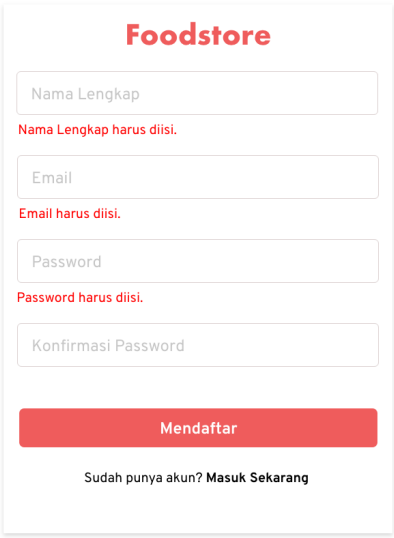
Fitur-fitur utama yang akan kita bangun antara lain:

- Daftar makanan
- Pencarian makanan berdasarkan keyword
- *Filter* makanan berdasarkan kategori
- *Filter* makanan berdasarkan tags
- *Login & register user*
- Keranjang belanja (*cart*)
- Checkout
- Riwayat pemesanan
- Kelola daftar alamat pengiriman

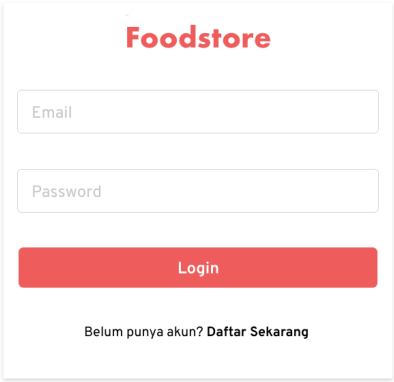
Mockup

Berikut adalah *mockup* untuk aplikasi yang akan kita buat.

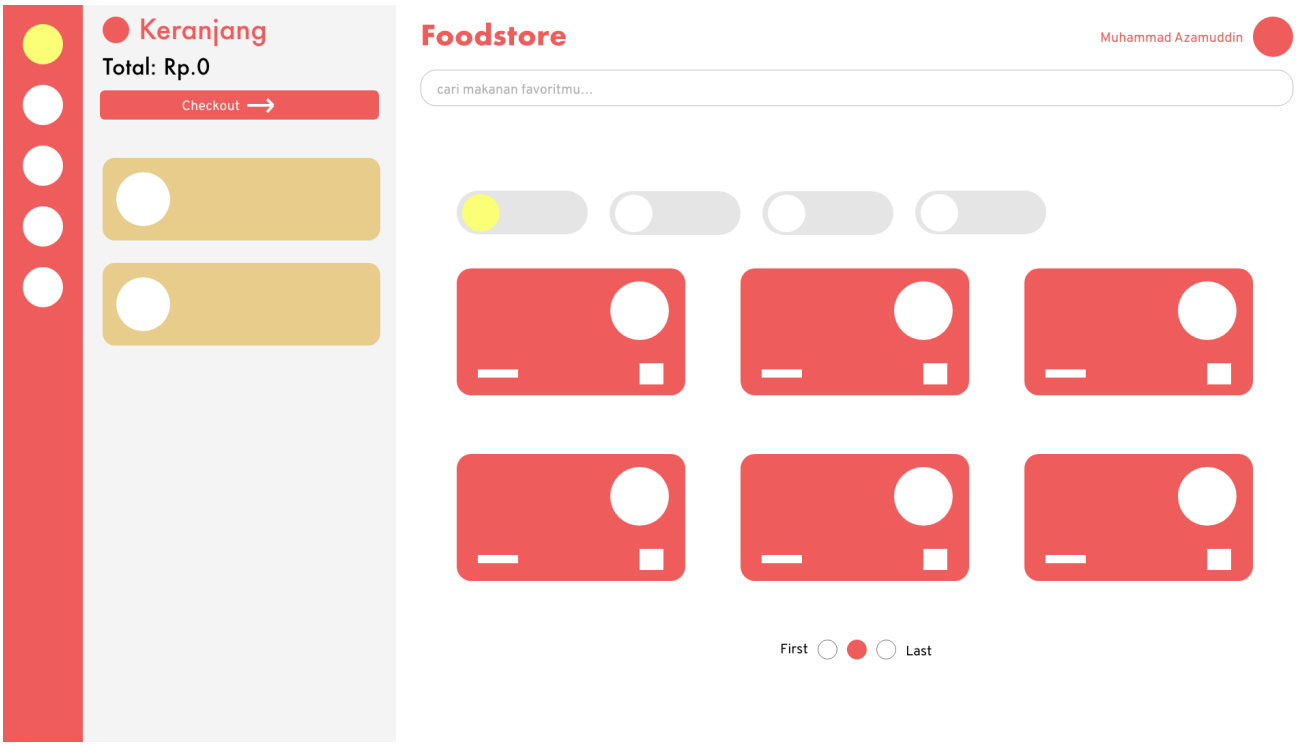
Mockup Halaman Register



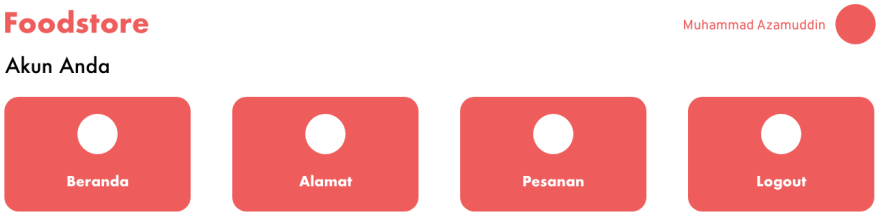
Mockup Halaman Login



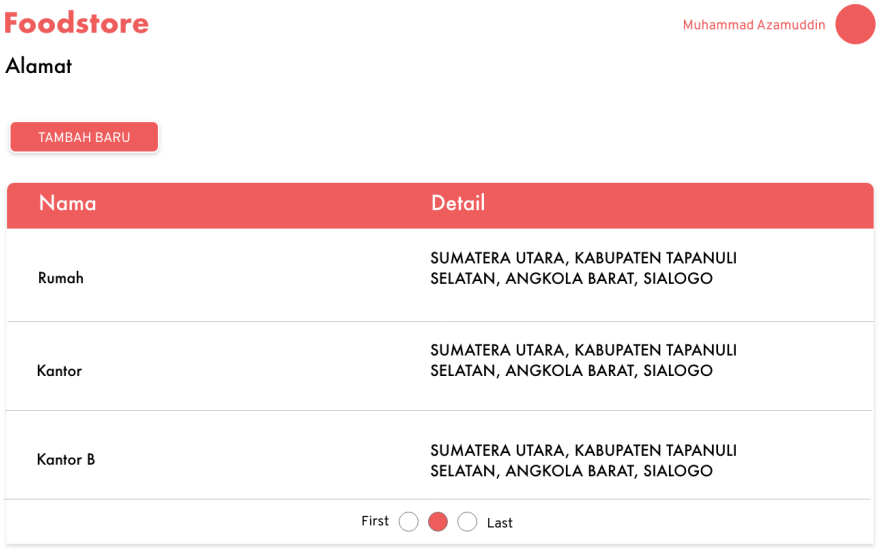
Mockup Halaman Home



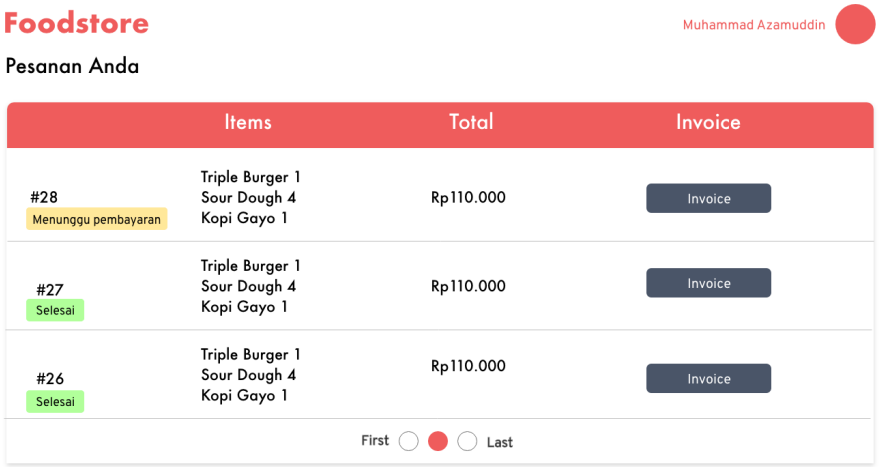
Mockup Halaman Akun



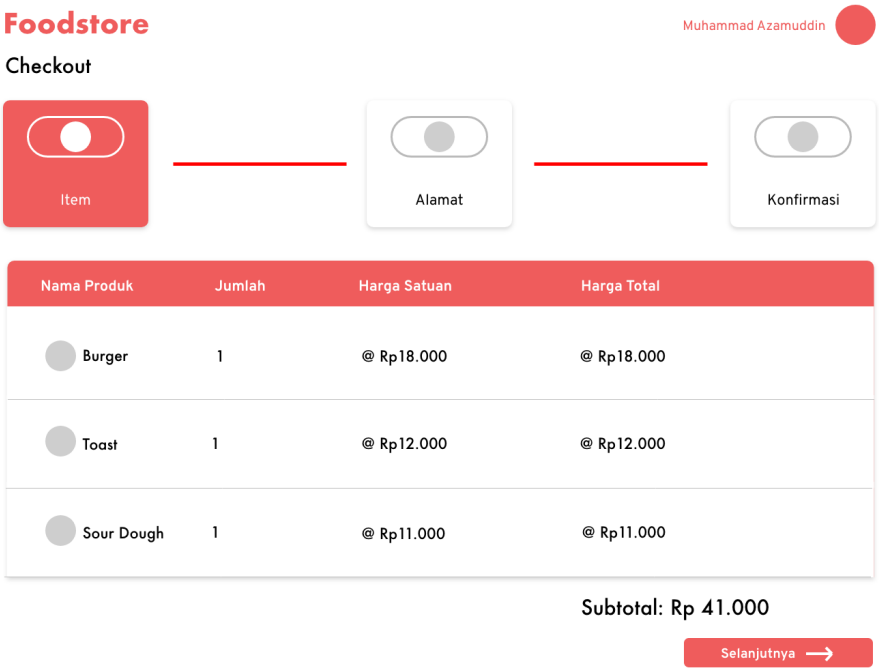
Mockup Halaman Daftar Alamat



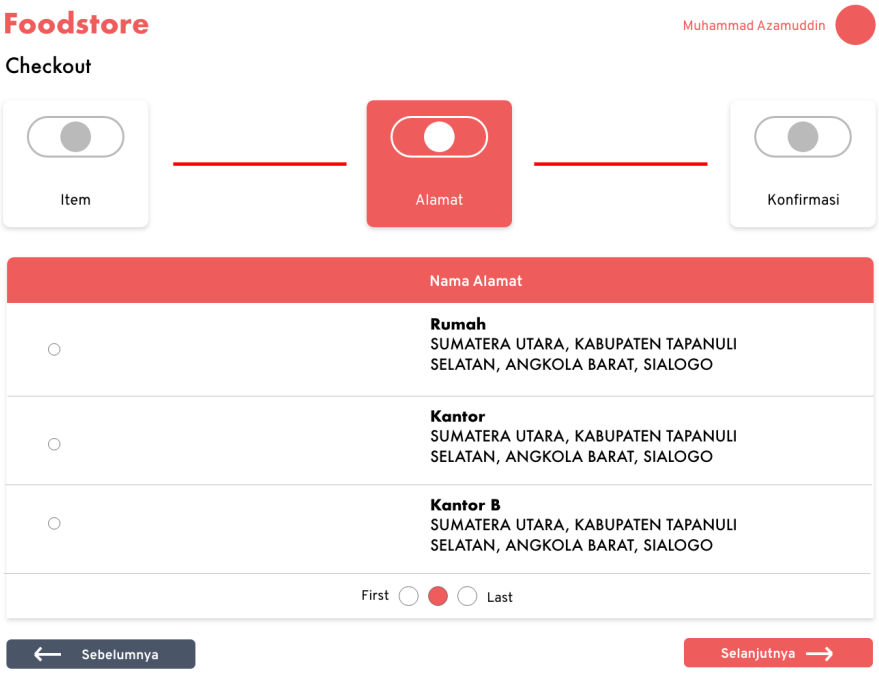
Mockup Halaman Riwayat Pemesanan



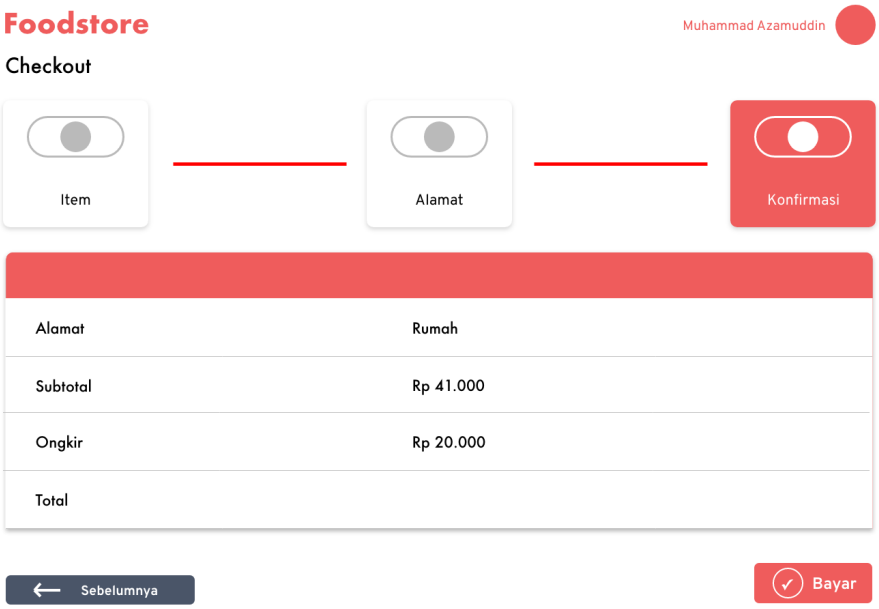
Mockup Halaman Checkout - Items



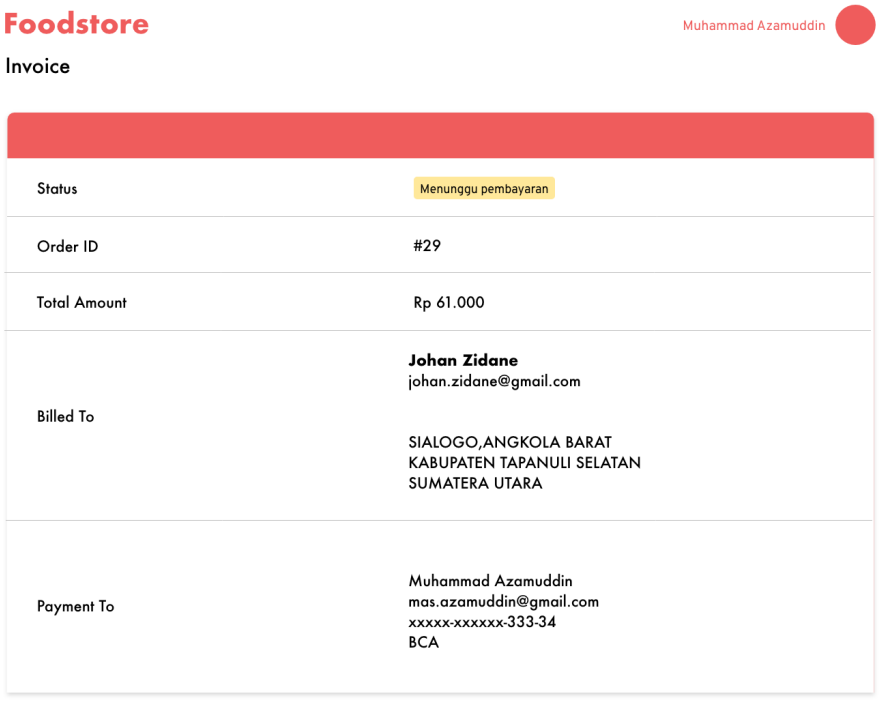
Mockup Halaman Checkout - Pilih Alamat



Mockup Halaman Checkout - Konfirmasi



Mockup Halaman Invoice



Publikasi Aplikasi