BAB 2
LANDASAN TEORI

2.1 HTTP

Sebuah aplikasi web berkomunikasi dengan client melalui HTTP (Hypertext Transfer Protocol)[5]. HTTP, sebagai protokol yang berbicara menggunakan request dan response menjadikan aplikasi web bergantung kepada siklus ini untuk menghasilkan dokumen yang ingin diakses oleh pengguna. Secara umum, aplikasi web yang akan kita kembangkan harus memiliki satu cara untuk membaca HTTP Request dan mengembalikan HTTP Response ke pengguna.

HTTP Request yaitu dimana server membaca apa yang dikirimkan oleh client melalui aplikasi web server. Sedangkan HTTP Response yaitu dimana server akan merespon permintaan yang telah dikirimkan oleh client.

2.2 API

API adalah singkatan dari *Application Programming Interface* yang merupakan perantara antar perangkat lunak. API berisi sekumpulan definisi dan protokol untuk membuat dan mengintegrasikan perangkat lunak. Dalam perangkat lunak dengan arsitektur *client-server*, API dapat kita analogikan sebagai pelayan, dengan *client* sebagai pengunjung dan *server* sebagai dapur. Tugas dari API adalah menjembatani antara pengunjung dan dapur, dengan menerima pesanan dari pengunjung lalu menyampaikannya ke dapur (*request*). Setelah pesanan berhasil dibuat, pelayan kemudian mengantarkan makanan dari dapur kepada pengunjung yang memesan (*response*).

API juga seringkali disebut sebagai sebuah kontrak, yang berisi dokumentasi yang berisi kesepakatan antara pihak-pihak yang terlibat[6]. Pihak 1 harus mengirim *request* terstruktur dengan cara yang telah ditentukan, dan Pihak 2 akan mengirim *response* dengan cara yang telah disepakati.

2.3 REST

REST adalah singkatan dari *Representational State Transfer*. REST merupakan gaya arsitektur perangkat lunak yang berisi serangkaian batasan yang didefinisikan untuk membantu membuat dan mengatur sistem terdistribusi [7]. Serangkaian batasan yang dimaksud adalah sebagai berikut:

1. *Client-server architecture*

   Arsitektur *client-server* berarti sisi tampilan pengguna harus dipisahkan dari sisi penyimpanan data untuk meningkatkan *portability* dari sisi tampilan pengguna di berbagai platform.
2. *Stateless*

   Server yang *stateless* tidak menyimpan informasi tentang pengguna yang menggunakan API-nya. Ini artinya *server* tidak mengingat apakah pengguna mengirimkan *request* pertamanya atau bukan.
3. *Cacheability*

   Sebuah *response* dari REST API harus jelas apakah boleh di-cache atau tidak untuk menghindari pengguna menggunakan data yang tidak seharusnya pada *request* yang akan datang.
4. *Layered system*

   Sistem berlapis berarti jika terdapat *proxy* atau *load balancer* antara *client* dan *server*, hubungan di antara keduanya tidak boleh terpengaruh, dan *client* tidak tahu apakah ia terhubung dengan ujung dari *server*-nya atau tidak.
5. *Uniform interface*

   Interface yang seragam berarti perlu ada keseragaman dalam cara berinteraksi dengan *server* terlepas dari jenis perangkat atau aplikasi (website, aplikasi mobile) di sisi *client*.

REST ini memiliki standarisasi dalam pemakaian yaitu mengenai URL dan HTTP verbs. Dengan menggunakan URL yang sama kita dapat melakukan 5 request method, yaitu:

1. GET

   Method yang biasa digunakan untuk mendapatkan data dari server. Ketika kita melakukan request dengan method GET maka server akan mencarikan data yang sesuai dengan kebutuhan kita dan kemudian mengembalikan data tersebut melalui response. Inilah yang biasa disebut dengan aksi READ.
2. POST

   Method yang biasa digunakan untuk membuat data baru pada database yang ada di server. Kita dapat mengirimkan data yang akan dibuat melalui body dari request yang kita kirim. Inilah yang biasa disebut dengan aksi CREATE.
3. PUT

   Method yang biasa digunakan untuk mengupdate data yang sudah ada pada database yang ada di server. Mirip seperti POST namun method ini biasanya digunakan untuk mengubah data bukan membuat. Inilah yang biasa disebut dengan aksi UPDATE.
4. PATCH

   Method yang biasa digunakan untuk mengupdate data yang sudah ada pada database yang ada di server. Mirip seperti PUT namun method ini biasanya digunakan untuk mengupdate beberapa field dalam sebuah record saja, bukan mengupdate semua field dalam sebuah record seperti PUT.
5. DELETE

   Method yang biasa digunakan untuk menghapus data yang sudah ada pada databse yang ada di server. Inilah yang biasa disebut dengan aksi DELETE .

2.3.1 *Underfetching*

Underfetching terjadi ketika *server* menerima satu GET request dari *client* namun tidak dapat sepenuhnya memberikan data yang diperlukan sehingga dibutuhkan GET *request* lain sebagai pelengkap. Hal ini dapat menyebabkan meningkatnya latensi sehingga pengguna perlu menunggu lebih lama sampai halaman berhasil dimuat, dan tingginya kompleksitas program karena dari sisi *client* perlu melakukan lebih dari satu *request* hanya untuk satu kebutuhan data.

Masalah *underfetching* pada REST ini dapat diatasi dengan menambahkan *endpoint* yang sesuai dengan kebutuhan *client*, namun hal ini justru akan

menimbulkan masalah baru. Jika terdapat kode yang memiliki fungsi yang mirip ditulis lebih dari satu kali, seiring dengan berkembangnya aplikasi, kompleksitas kode akan semakin meningkat. Selain itu, proses *development* di sisi *client* juga akan melambat karena tertunda sampai API selesai diperbarui.

2.3.2 *Overfetching*

Overfetching merupakan keadaan di mana *server* memberikan *response* melebihi dari yang *client* butuhkan saat client melakukan GET *request*. Hal ini tentu tidak efisien karena pengguna mengunduh data yang tidak dipakai, Ukuran *payload* yang besar akan menjadi masalah terutama pada platform mobile yang mengharapkan sistem yang responsif dan memiliki latensi rendah.

2.4 GraphQL

Pada tahun 2012, Facebook mengalami masalah pada aplikasi *mobile* mereka yang seringkali mengalami *crash* dan akhirnya membangun GraphQL sebagai solusi lalu mempublikasikannya pada tahun 2015 [3]. GraphQL adalah bahasa *query* yang digunakan untuk pengambilan data deklaratif agar *client* dapat menentukan secara spesifik data yang dibutuhkan dari API. GraphQL membuat pengembangan dan perubahan pada API menjadi lebih mudah.

GraphQL telah digunakan oleh banyak perusahaan. Beberapa perusahaan besar yang menggunakan GraphQL adalah Facebook, Twitter, dan GitHub[8]. Berikut merupakan contoh *query* GraphQL untuk mengambil data *user* beserta nama dan umurnya:

```
query {
  user {
    name
    age
  }
}
```

Gambar 2-1. Contoh Query GraphQL

Dari query di atas, response berupa JSON dari API akan terlihat seperti ini:

```
{
  "user": {
    "name": "Dimas Miftahul Huda",
    "age": 21
   }
}
```

Gambar 2-2. Contoh Response dari API

Jika hanya ingin menampilkan “name” saja, *client* hanya perlu menghapus “age” pada *query* tanpa perlu ada perubahan atau penambahan pada *server*, begitu juga jika ingin menambah atribut lain. Flexibilitas inilah yang diharapkan dapat menyelesaikan masalah *underfetching* dan *overfetching*.

2.5 JSON

JSON adalah singkatan dari *JavaScript Object Notation.* JSON merupakan format ringan yang digunakan untuk menyimpan dan mengirimkan data [9]. JSON dinilai relatif mudah ditulis dan dibaca oleh manusia, serta mudah juga dibuat dan diterjemahkan oleh komputer. Terlepas dari namanya, JSON merupakan format data yang tidak terbatas pada satu Bahasa pemrograman tertentu dan dapat digunakan untuk pertukaran data di berbagai platform baik website, desktop, maupun perangkat seluler. Karena sifat-sifat inilah JSON dinilai ideal sebagai bahasa pertukaran data.

2.6 NodeJS

Node.js adalah runtime environment untuk JavaScript yang bersifat open-source dan cross-platform. Dengan Node.js kita dapat menjalankan kode JavaScript di mana pun, tidak hanya terbatas pada lingkungan browser[10]. Node.js menjalankan V8 JavaScript engine (yang juga merupakan inti dari Google Chrome) di luar browser. Ini memungkinkan Node.js memiliki performa yang tinggi.

Node.js juga menyediakan banyak library/module JavaScript yang membantu menyederhanakan pengembangan aplikasi web. Berikut ini adalah beberapa fitur penting dari Node.js yang menjadikannya pilihan utama dalam pengembangan aplikasi:

- *Asynchronous & Event-driven*
  Semua API dari Node.js bersifat asynchronous, artinya tidak memblokir proses lain sembari menunggu satu proses selesai. Server Node.js akan melanjutkan ke ke pemanggilan API berikutnya lalu memanfaatkan mekanisme event notification untuk mendapatkan respon dari panggilan API sebelumnya.
- *Very Fast*
  Eksekusi kode dengan Node.js sangat cepat karena berjalan pada V8 JavaScript Engine dari Google Chrome.
- *Single Threaded but Highly Scalable*
  Node.js menggunakan model single thread dengan event looping. Mekanisme ini membantu server untuk merespon secara asynchronous dan menjadikan server lebih scalable dibandingkan server tradisional yang menggunakan banyak thread untuk menangani permintaan.

2.7 Apollo Server

Apollo Server adalah server GraphQL open-source, sesuai spesifikasi yang kompatibel dengan klien GraphQL apa pun, termasuk Apollo Client. Apollo Server merupakan salah satu pilihan terbaik untuk membangun GraphQL API yang siap untuk produksi dan memiliki fitur self documentaion dan dapat menggunakan data dari berbagai macam sumber. Berikut merupakan fitur-fitur yang mencolok dari Apollo Server:

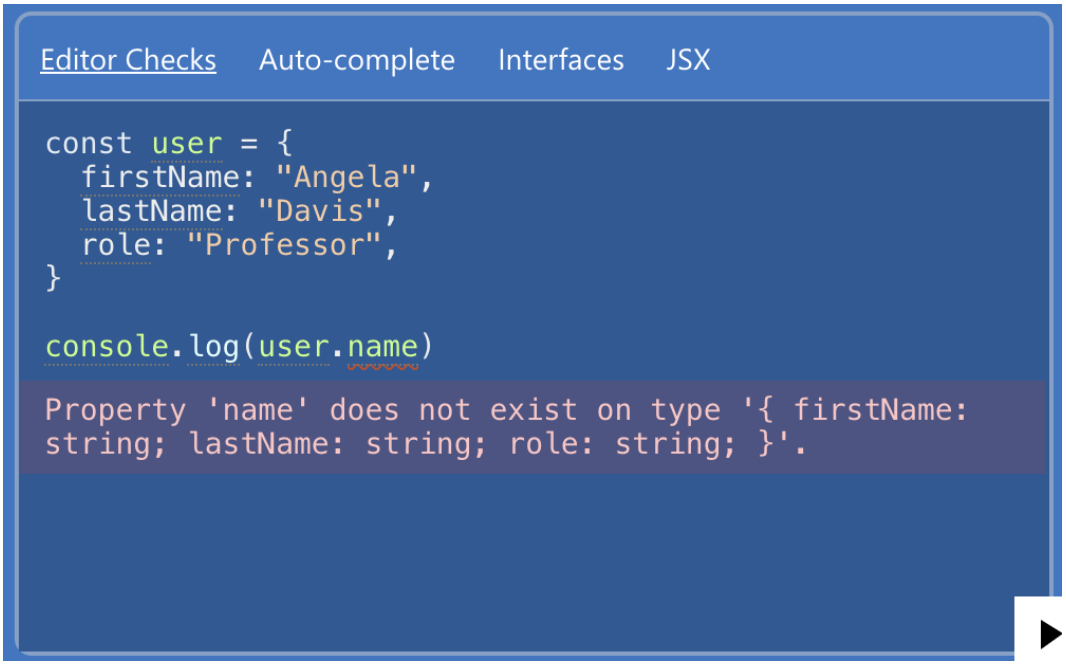
- Setup yang mudah, sehingga pengembang klien Anda dapat mulai mengambil data dengan cepat.
- Adopsi yang bertahap, memungkinkan penggunanya untuk menambahkan fitur sesuai kebutuhan.

- Kompatibilitas universal dengan sumber data apa pun, *build tool* apa pun, dan klien GraphQL apa pun.
- Kesiapan untuk produksi, memungkinkan developer merilis fitur lebih cepat

2.8 TypeScript

TypeScript merupakan bahasa pemrograman *strongly typed* yang dibangun di atas bahasa pemrograman JavaScript. Penggunaan TypeScript dapat membantu developer untuk menghindari *type error* yang merupakan masalah yang sering terjadi pada bahasa pemrograman dynamic type seperti JavaScript.

Berikut merupakan contoh type checking yang dilakukan TypeScript.



Gambar 2-3. Contoh Type Checking Pada TypeScript

2.9 Prisma

Prisma adalah library *Object Relational Mapping* (ORM) untuk Node.js dan TypeScript. ORM adalah sebuah Teknik yang memungkinkan developer melakukan *data definition* dan *data manipulation* menggunakan paradigma object-oriented. Dengan menggunakan library ORM, developer dapat berinteraksi dengan database tanpa menggunakan Bahasa SQL, melainkan menggunakan bahasa pemrograman yang sama dengan yang digunakan untuk membangun sistem.

Berikut merupakan contoh penggunaan prisma dalam melakukan pendefinisian table dan melakukan query.

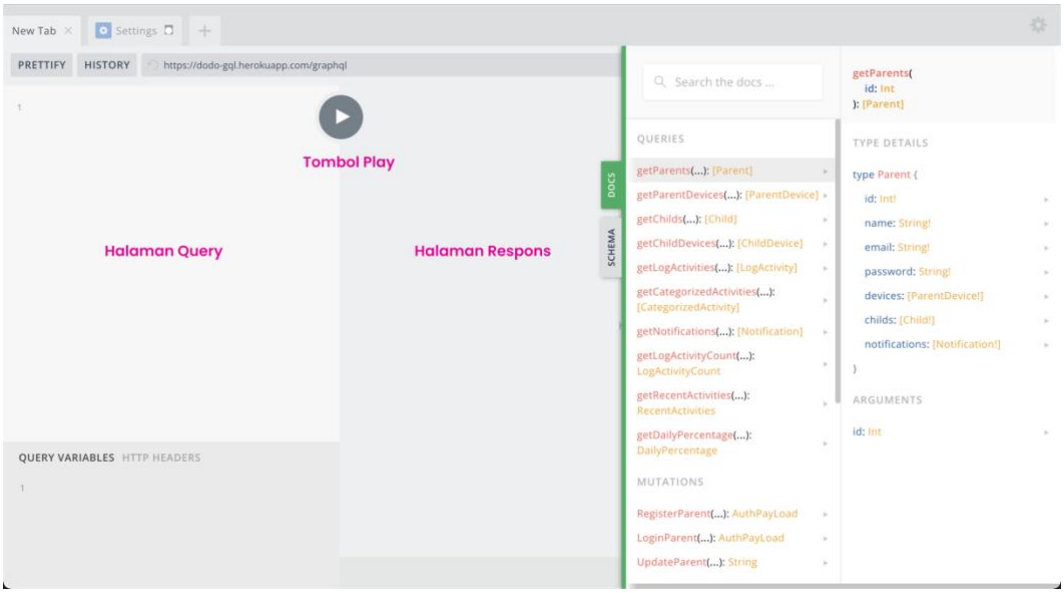
Gambar 2-4. Contoh Penggunaan Prisma Dalam Pendefinisan Table

Gambar 2-5. Contoh Penggunaan Prisma Dalam Melakukan Query

2.10 GraphQL Playground

GraphQL Playground merupakan sebuah IDE untuk GraphQL. GraphQL Playground memungkinkan developer melihat dokumentasi dari GraphQL API yang telah dibangun dan melakukan query di dalamanya.

Beriku merupakan contoh tampilan dari GraphQL Playground.



Gambar 2-6. Tampilan GraphQL Playground