Implementasi NestJS dan Prisma pada pengembangan Backend Monolitik pada Aplikasi Web Antria

Tugas Akhir

diajukan untuk memenuhi salah satu syarat

memperoleh gelar sarjana

dari Program Studi Rekayasa Perangkat Lunak

Fakultas Informatika

Universitas Telkom

1302204044

Muhammad Rovino Sanjaya

Program Studi Sarjana Rekayasa Perangkat Lunak

Fakultas Informatika

Universitas Telkom

Bandung

2024

LEMBAR PENGESAHAN

Implementasi NestJS dan Prisma pada pengembangan Backend Monolitik pada Aplikasi Web Antria

Implementation of Monolithic Backend Development with NestJS and Prisma for Antria's Web Application

NIM: 1302204044

Muhammad Rovino Sanjaya

Tugas akhir ini telah diterima dan disahkan untuk memenuhi sebagian syarat memperoleh gelar pada Program Studi Sarjana Rekayasa Perangkat Lunak
Fakultas Informatika
Universitas Telkom

Bandung, 23 Desember 2024
Menyetujui

Pembimbing I	Pembimbing II
(Dr. Mira Kania Sabariah, S.T., M.T.)	(Monterico Adrian, S.T., M.T.)
NIP: 14770011	NIP: 20870024

Ketua Program Studi
Sarjana Rekayasa Perangkat Lunak,

Dr. Mira Kania Sabariah, S.T., M.T.
NIP: 14770011

LEMBAR PERNYATAAN

Dengan ini saya, Muhammad Rovino Sanjaya, menyatakan sesungguhnya bahwa Tugas Akhir saya dengan judul "**Implementasi NestJS dan Prisma pada pengembangan Backend Monolitik pada Aplikasi Web Antria**" beserta dengan seluruh isinya adalah merupakan hasil karya sendiri, dan saya tidak melakukan penjiplakan yang tidak sesuai dengan etika keilmuan yang belaku dalam masyarakat keilmuan. Saya siap menanggung resiko/sanksi yang diberikan jika dikemudian hari ditemukan pelanggaran terhadap etika keilmuan dalam buku TA atau jika ada klaim dari pihak lain terhadap keaslian karya.

Bandung, 23 Desember 2024

Yang Menyatakan,

Muhammad Rovino Sanjaya

Implementasi NestJS dan Prisma pada pengembangan Backend Monolitik pada Aplikasi Web Antria

Muhammad Rovino Sanjaya[1], Mira Kania Sabariah[2], Monterico Adrian[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[4]Divisi Digital Service PT Telekomunikasi Indonesia
[1]rovino@students.telkomuniversity.ac.id, [2]mirakarnia@telkomuniversity.ac.id,
[3]monterico@telkomuniversity.ac.id

Abstrak

Populasi penduduk yang tinggi di Indonesia mengakibatkan antrian panjang dalam berbagai pelayanan publik atau pelayanan konsumen. Pelanggan harus datang ke tempat untuk mengambil antrian dan menunggu gilirannya, semakin panjang antrian, semakin lama waktu tunggu yang dibutuhkan. Hal ini tidak efisien dan menyebabkan banyak waktu terbuang hanya untuk menunggu antrian. Jika nomor antrian bisa didapatkan secara online dan dapat dipantau secara online, maka dapat mengurangi waktu yang terbuang. Dengan aplikasi antrian virtual dapat memperpendek antrian secara fisik, dengan cara antri secara virtual, melihat antrian yang sedang berjalan, dan booking tempat. Pada Pengembangan aplikasi ini, framework yang digunakan adalah NestJS dan PrismaJS dengan menerapkan RESTful API, Object Relational Mapping, dan menghindari Anti-Pattern. Framework NestJS mendukung pembuatan aplikasi ber-arsitektur monolitik dan *microservice*. Setelah aplikasi dibangun di arsitektur monolitik, aplikasi dapat dengan mudah dimigrasikan ke *microservice* saat penggunaan aplikasi sudah hampir mendekati batas muat pengguna.

Kata kunci : NestJS, PrismaJS, Anti-Pattern, Antrian, Backend, REST

Abstract

The abstract should state briefly the general aspects of the subject and the main concolusions. The length of abstract should bo no more than 200 word and should be typed be with 10 pts.

Keywords: keyword should be chosen that they best describe the contents of the paper and should be typed in lower-case, except abbreviation. Keyword should be no more than 6 word

1. Pendahuluan

Latar Belakang

Meningkatnya populasi di indonesia mengakibatkan banyak pelanggan ya-ng mengantri untuk mendapatkan layanan di bank, restoran, rumah sakit, dan tempat penyedia jasa lainnya. Mengantri merupakan kegiatan yang membosankan dan menguras waktu. Panjangnya antrian juga mampu berdampak pada mutu pelayanan di suatu tempat. Pelanggan yang harus menunggu lama berpotensi beralih ke pesaing, atau jika ada urusan lain yang lebih penting, maka pelanggan akan keluar dari tempat antrian, meninggalkan antriannya [8][4][17]. Solusi yang ada pada bank, kantor pos, dan rumah sakit saat ini menggunakan *ticketting* nomor antrian secara manual, dimana antrian yang sedang dilayani ditampilkan di layar pada ruang tunggu. Hal ini kurang efektif karena pelanggan harus berada di ruang tunggu[4].

Perkembangan teknologi yang cepat mengakibatkan penggunaan perangkat pintar atau *smartphone* merupakan hal lumrah, banyak bermunculan aplikasi antrian virtual seperti Antrique, Qiwee, ExaQue dimana pengguna dapat mengantri dari jarak jauh melalui aplikasi maupun *website*. Para pengguna aplikasi tersebut dapat melakukan hal lain saat mengantri sebelum gilirannya. Namun, aplikasi-aplikasi tersebut memiliki banyak kelemahan seperti *UI/UX* yang tidak bagus, sering *crash* dan *freeze*, tidak ada estimasi waktu antrian, dan masih belum ada yang berfokus ke sektor *food and beverage*.

Oleh karena itu, perlunya dikembangkan sebuah aplikasi yang memiliki fitur yang sama atau lebih dengan menutup kekurangan pada aplikasi tersebut. Pengembangan aplikasi yang direncanakan menggunakan arsitektur monolitik karena mudahnya untuk dibuat dan di-*deploy* secara cepat untuk di iterasikan. Namun, arsitektur monolitik memiliki kelemahan seperti sulitnya untuk di-*maintenance*, *scale*, dan reliabilitas nya. Oleh karena itu, perlu diperhatikan bagaimana cakupan aplikasi kedepannya dan perlunya migrasi ke arsitektur *microservice* [6] [7].

Dalam pengembangan aplikasi web, pemilihan bahasa pemrograman untuk digunakan di *backend* sangatlah penting karena dapat mempengaruhi performa aplikasi yang dibangun. Dalam pemilihan bahasa pemrograman *backend*, banyak pilihan yang tersedia seperti PHP, Python, Ruby, PERL, dan banyak lagi. NodeJs merupakan *tools* yang memungkinkan bahasa JavaScript dapat dijalankan pada sisi *backend*. Dalam sisi performa, NodeJs lebih unggul dibanding PHP dan Python dalam sisi kecepatan melayani *request* dari client [18] [11].

NestJs merupakan *framework backend* dari Nodejs yang menggunakan bahasa typescript, dan bisa digunakan untuk pengembangan arsitektur berbasis *microservice* dan monolitik, jadi jika aplikasi dikembangkan pada arsitektur monolitik dapat dengan mudah dimigrasikan ke *microservice*. NestJs juga bisa digunakan bersamaan dengan *framework* PrismaJs untuk mengelola database [10]. PrismaJs merupakan *framework* Object Relational Mapping (ORM) [13], digunakan untuk mempercepat, dan mempermudah pengembangan aplikasi yang database-nya memiliki relasi yang kompleks dan sulit di-*maintenance* jika menggunakan Structured Query Language (SQL) [19].

Implementasi Application Programming Interface (API) yang digunakan adalah Representational State Transfer (RESTful) API, RESTful API adalah arsitektur untuk mempermudah komunikasi client-server agar efektif untuk transaksi data. Namun, pada implementasi RESTful API, ada beberapa hal yang perlu diperhatikan seperti keamanan saat transaksi atau komunikasi [3]. Keamanan yang lemah dapat mengakibatkan hacker dapat dengan mudah melakukan *request tampering*, mengambil data pengguna, dan dapat membocorkan data keuangan mitra. *Design pattern* juga perlu diperhatikan dalam penggunaan bahasa untuk API *endpoint* nya agar tidak terjadi *anti pattern*. *Anti pattern* terjadi saat penamaan API tidak sesuai dengan fungsi, atau ada fungsi sejenis tapi penamaannya berbeda jauh. Dengan menghindari *anti pattern*, dapat berakibat ke aplikasi yang lebih mudah di-*sustain* dan di-*maintain* [1] [2].

Berdasarkan uraian di atas, penelitian ini akan membuat sebuah *backend* aplikasi antrian dengan menggunakan arsitektur monolitik dengan *framework* NestJs dan PrismaJs sebagai *framework* nya. Setelah fitur-fitur aplikasi dibuat, perlu dilakukan unit testing untuk memvalidasi kodingan yang telah ditulis. Hal ini bertujuan untuk meminimalisir bug dan mencegah terjadinya regresi saat fitur baru ditambahkan [15].

Topik dan Batasannya

Aplikasi antria memerlukan backend developer untuk mengimplementasi-kan fungsi fungsi API dan manajemen database nya. Maka dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana meningkatkan *maintainability* penggunaan database pada projek.
2. Bagaimana merancang API yang bebas dari *anti pattern*.
3. Bagaimana merancang sistem keamanan pada API untuk melayani *request*.

dan batasan masalah sebagai berikut:

1. Hanya berfokus kepada implementasi database menggunakan Prisma ORM.
2. Berfokus ke bagaimana membuat *endpoint* API yang tidak menimbulkan *anti pattern*.
3. Implementasi keamanan pada saat penanganan *request* menggunakan JSON Web Token (JWT).

Tujuan

Tujuan dari pengerjaan Tugas Akhir ini yaitu:

1. Mengimplementasikan Prisma ORM untuk meningkatkan *maintainability* pada penggunaan database.
2. Membuat dokumentasi API yang dapat dengan mudah dimengerti dan di *maintain*.
3. Mengamankan data pengguna dengan memerlukan *Authorization* pada setiap header request.

Organisasi Tulisan

Pada sub-bagian ini dituliskan bagian-bagian selanjutnya (setelah Pendahuluan) pada jurnal TA ini, disertai penjelasan sangat singkat.

2. Kajian Pustaka

2.1 NodeJs

NodeJs adalah *runtime* javascript yang basisnya dibangun dari V8 Java-Script Engine. NodeJs berjalan dalam bentuk *event-driven*, dan menggunakan model *non blocking I/O*. meskipun menggunakan *event-driven* untuk melayani *request*, NodeJs dapat melayani jutaan koneksi dalam waktu bersamaan secara *asynchronous* [16].

2.2 NestJs

NestJs merupakan *framework* untuk Nodejs yang dikembangkan oleh Kamil Myśliwiec yang bertujuan untuk membuat aplikasi NodeJs yang efektif dan *scalable*. NestJs mendukung penggunaan bahasa typescript dan javascript. NestJs juga menggabungkan komponen-komponen dari Functional Programming, Object Oriented Programming, dan Functional Reactive Programming [12] [10].

2.3 Object Relational Mapping

Object Relational Mapping (ORM) adalah sebuah teknologi yang memeta-kan tabel database ke dalam objek, biasanya dipakai dalam bahasa yang berbasis Object Oriented Programming. Dengan menggunakan ORM, developer dapat berfokus ke *business logic* tanpa mengkhawatirkan penggunaan akses database yang rumit [9].

2.4 PrismaJs

PrismaJs adalah ORM Open Source, biasanya digunakan sebagai alternatif dari menggunakan Structured Query Language (SQL) secara langsung. PrismaJs mendukung penggunaan database MySQL, PostgreSQL, SQLite, SQL Server, CockroachDB, dan MongoDB. PrismaJs digunakan untuk mempermudah pengembangan database yang memiliki relasi yang kompleks dan besar, dengan cara memberikan API yang *type-safe* untuk *query* database nya dan mengembalikan hasil *query* dalam bentuk JavaScript Object Notation (JSON) [13].

2.5 Arsitektur Monolitik

Arsitektur Monolitik adalah arsitektur sebuah *software* dimana beberapa fungsi komponen yang berbeda seperti fungsi otorisasi, *business logic*, notifikasi, dan pembayaran. Semua fungsi tersebut berada dalam satu program dan *platform* yang sama. Arsitekru monolitik mudah untuk dikembangkan dan di-*deploy*. Namun, sulit untuk di-*maintenance* dan di-*scale* [6].

2.6 JSON Web Token

JSON Web Token (JWT) adalah sebuah token berbentuk *string* json yang dapat digunakan untuk melakukan otorisasi. Ukuran JWT tergolong kecil jadi dapat dengan cepat ditransfer antar client dan server. JWT menggunakan algoritma HMAC atau RSA untuk mengenkripsi *digital signature* yang digunakan. JWT memiliki 3 bagian pada *string* nya yang dipisahkan menggunakan ".", bagian ini berupa *header*, *payload*, dan *signature* [14].

2.7 Anti Pattern

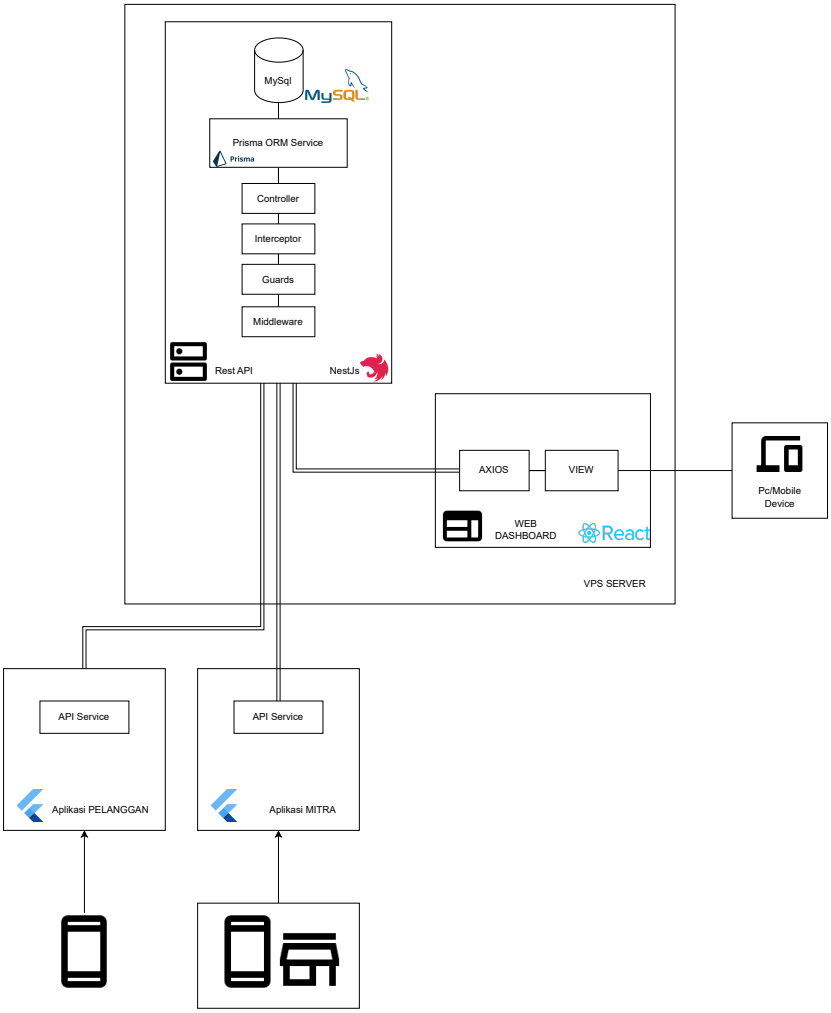
Anti Pattern terjadi jika pembuatan nama sebuah objek tidak konsisten dengan yang lain. Objek disini dapat berupa *endpoint* API, nama *variable*, nama fungsi, dan nama lain yang penggunaannya bersifat publik. Terjadinya *anti pattern* dapat mengakibatkan sulitnya untuk memahami suatu dokumentasi dan kodingan aplikasi [1] [2].

2.8 RESTful API

Representational State Transfer (RESTful) Application Programming Interface (API) adalah arsitektur untuk mempermudah komunikasi client-server agar efektif untuk transaksi data. Tipe data yang paling sering digunakan untuk transaksi client server adalah JSON. Karakteristik RESTful meliputi : Client-Server, Stateless, Layered Architecture, Caching, Code on Demand, dan Uniform Interface [5].

3. Sistem yang Dibangun

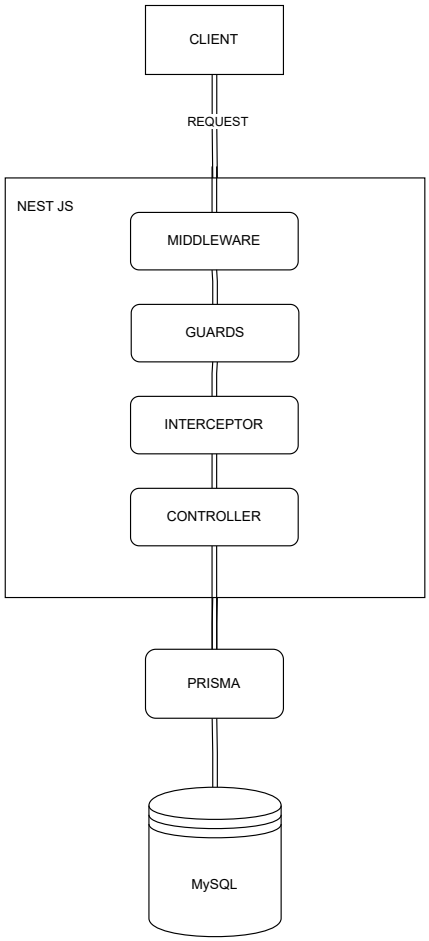
Implementasi dan perancangan RESTful API, Database, dan fungsi bisnis dikembangkan menggunakan *framework* NestJs. Pada NestJs terdapat beberapa komponen seperti: Middleware, Guards, Interceptor, Controller, dan Service. Service yang dipakai adalah PrismaJs untuk menghubungkan NestJs ke *Database System*. Desain arsitektur sistem secara keseluruhan pada gambar 1. Terdapat 3 Aplikasi frontend yang saling terhubung via backend, Aplikasi Pelanggan berupa flutter, Aplikasi Mitra berupa flutter, dan Website Dashboard Mitra berupa ReactJs.



Gambar 1. Arsitektur Desain Sistem

3.1 Request Life Cycle

Gambar 2 merupakan *Request Lifecycle* yang menjelaskan management API atau bagaimana alur *request* ditangani dari awal sampai akhir.



Gambar 2. Desain Sistem

3.1.1 Middleware

Pada Middleware, fungsi akan dipanggil sebelum masuk ke routing. fungsi middleware dapat mengakses data *request* dan *response*. beberapa fungsi Middleware seperti *logger*, dan cek notifikasi. Contoh penggunaan nya bisa dilihat pada listing 1, request dapat di hentikan maupun di alihkan ke middleware selanjutnya.

Listing 1: Middleware

```
export class LoggerMiddleware implements NestMiddleware {
  use(req: Request, res: Response, next: NextFunction) {
    console.log('Request...');
    next();
  }
}
```

3.1.2 Guard

Pada Guard, *request* akan dicek *authenticity*, untuk mengetahui validitas dari *request* tersebut. Tahap ini juga akan dicek keamanan session menggunakan JWT dan CSRF. Contoh penggunaan guard seperti pada listing 2, request yang menuju suatu controller akan di cek oleh guards, jika memiliki otorisasi maka akan dilanjutkan ke controller, jika tidak akan di reject dengan response forbidden access.

Listing 2: Penggunaan Guards pada Controller

```
@UseGuards(AuthGuard)
@Get('profile')
getProfile(@Request() req) {
    return req.user;
}
```

3.1.3 Interceptor

Setelah melalui Guard, Request akan masuk ke Interceptor. Dimana jika suatu *request* mempunyai suatu karakteristik yang ditentukan, maka akan menjalankan fungsi tambahan. Interceptor terjadi ketika *request* datang *(pre)*, dan response *(post)*. Contoh implementasi Interceptor terdapat pada listing 3.

Listing 3: Interceptor

```
@UseInterceptors(FileInterceptor('profile_picture',fileInterceptor()))
async update(@Param('id') id: string, @Body() data: Karyawan, @UploadedFile() file:
    Express.Multer.File): Promise<Karyawan> {}
```

3.1.4 Controller

Setelah melewati Interceptor, fungsi di Controller akan dijalankan. Jika pada Controller tersebut perlu data dari database maka akan turun ke service PrismaJs. Contoh karyawan controller pada listing 4 dimana request akan diteruskan ke service setelah password di hash.

Listing 4: Controller

```
@Controller('karyawan')
@Post()
async create(@Body() data: any): Promise<Karyawan> {
  const hashedPassword = await bcrypt.hash(data.password, 10);
  const karyawanData = { ...data, password: hashedPassword };
  return this.karyawanService.createKaryawan(karyawanData);
}
```

3.1.5 Service

Pada Service, fungsi akan melakukan database call menggunakan ORM ke database MySQL yang akan di kembalikan (return) ker Controller[10]. Contoh implementasi service pada listing 5 dimana berfungsi untuk mengambil banyak karyawan.

Listing 5: Service

```
async karyawans(params: {
  skip?: number;
  take?: number;
  cursor?: Prisma.KaryawanWhereUniqueInput;
  where?: Prisma.KaryawanWhereInput;
  orderBy?: Prisma.KaryawanOrderByWithRelationInput;
}): Promise<Karyawan[]> {
  const { skip, take, cursor, where, orderBy } = params;
  return this.prisma.karyawan.findMany({
    skip,take,cursor,where,orderBy,
  });
}
```

3.2 Persiapan

Untuk memulai projek, ada beberapa tahap yang perlu dilakukan, pertama membuat projek nestjs menggunakan node, lalu dilanjutkan menginstall dependensi yang diperlukan

3.2.1 Instalasi NestJS dan Prisma

Hal pertama yang dilakukan untuk menginstall NestJS adalah membuka terminal lalu menjalankan command npm untuk menginstall NestJS, dilanjut dengan menginstall prisma client.

Listing 6: terminal: npm

```
$ npm i -g @nestjs/cli
$ nest new antria
$ npm install prisma --save-dev
```

Command pada listing 6 untuk melakukan generasi folder projek pada NestJS, dan menginstall prisma client sebagai dependensi. Prisma sendiri merupakan aplikasi CLI untuk membantu dalam manajemen database pada projek NestJS menggunakan ORM.

3.2.2 Konfigurasi Prisma

Berdasarkan SRS, didapatkan beberapa entity pada database meliputi: Pelanggan, Mitra, Produk, OrderList, Pesanan, Antrian, Karyawan, Review, Chat, dan Analytic. Implementasi juga harus sesuai dengan ERD (Entity Relationship Diagram) yang telah dibuat pada gambar 3.

Listing 7: terminal: npx

```
$ npx prisma init
$ npx prisma migrate dev --name init
```

Command pada listing 7 untuk inisialisasi prisma pada projek, berfungsi untuk generasi template config yang nanti harus diubah, seperti database connection dan database name nya. Pada tahap ini juga penulis perlu mendefinisikan model dan relasi pada database ke bentuk notasi modelling prisma.

Listing 8: scheme.prisma

```
model Pesanan {
  invoice    String       @id
  payment    Payment
  pemesanan  OrderType?   @default(ONLINE)
  takeaway   Boolean      @default(false)
  status     PaymentStatus @default(PENDING)
  oderlist   OrderList[]
  pelanggan  Pelanggan    @relation(fields: [pelangganId], references: [id])
  pelangganId Int
  mitra      Mitra        @relation(fields: [mitraId], references: [id])
  mitraId    Int
  antrian    Antrian?
  antrianId  Int?
  created_at DateTime     @default(now())
  updated_at DateTime     @updatedAt
}
model OrderList {
  id       Int    @default(autoincrement()) @id
  quantity Int    @default(1)
  note     String @default("")
  pesanan  Pesanan @relation(fields: [pesananId], references: [invoice])
  produk   Produk @relation(fields: [produkId], references: [id])
  produkId Int
  pesananId String
}
```

Pada listing 8 penulis mendefinisikan skema model pesanan dan model orderlist beserta relasi nya pada prisma. syntax @relation berguna untuk mendefinisikan relasi nya pada model.

3.3 Implementasi

3.3.1 Guards

implementasi code guards dibuat 2 fungsi, guards untuk pengguna, dan guards untuk mitra. Contoh implementasi pada listing 9, disini penulis mendefinisikan guard untuk memeriksa JWT Token dari request yang diterima apakah valid.

Listing 9: Authentication Guards

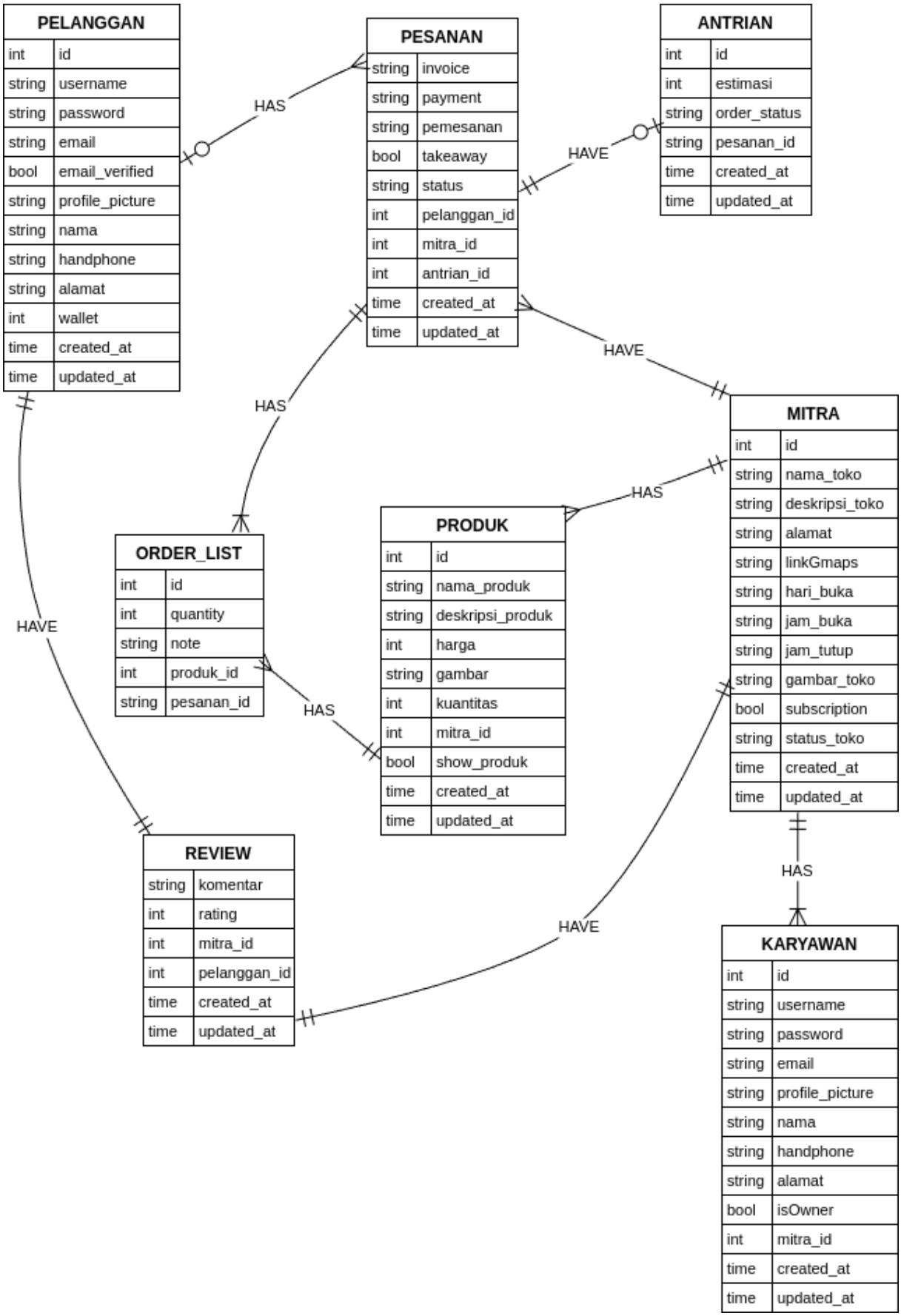
```
export class AuthGuard implements CanActivate {
  constructor(private jwtService: JwtService) {}
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const token = this.extractTokenFromHeader(request);
    if (!token) {
      throw new UnauthorizedException();
    }
    try {
      const payload = await this.jwtService.verifyAsync(
        token,
        {
          secret: jwtConstants.secret
        }
      );
      request['user'] = payload;
    } catch {
      throw new UnauthorizedException();
    }
    return true;
  }

  private extractTokenFromHeader(request: Request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}
```

3.3.2 Interceptor

Interceptor digunakan untuk mengambil file gambar dari request client, lalu disimpan ke dalam file storage server.

Listing 10: File Interceptor

```
@UseInterceptors(FileInterceptor('profile_picture',{
  storage: diskStorage({
    destination: "./MediaUpload/",
    filename: (req, file, callback) => {
      const uniqueSuffix = uuidv4();
      const fileExtName = path.extname(file.originalname);
      const newFileName = `${uniqueSuffix}${fileExtName}`;
      callback(null, newFileName);
    }
  })
}))
```



Gambar 3. Entity Relationship Diagram

3.3.3 Controller

Controller minimal mengikuti banyak entity pada model. masing masing controller memiliki 4 fungsi yang merepresentasikan method http yaitu GET, POST, PUT, DELETE.

1. Pelanggan controller
2. Mitra controller
3. Produk controller
4. OrderList controller
5. Pesanan controller
6. Antrian controller
7. Karyawan controller
8. Review controller
9. Analytic controller

3.3.4 Service

Sama seperti controller, banyak service minimal mengikuti banyak entity pada model. bedanya service berinteraksi langsung dengan ORM Prisma.

1. Pelanggan Service
2. Mitra Service
3. Produk Service
4. OrderList Service
5. Pesanan Service
6. Antrian Service
7. Karyawan Service
8. Review Service
9. Analytic Service

3.3.5 API Endpoint

Untuk menghindari Anti Pattern, API endpoint sesuai dengan nama entity pada model. setiap endpoint entity mendukung 4 method http.

1. Auth Endpoint, merupakan endpoint yang berhubungan dengan autentikasi client.

Tabel 1. Auth Endpoint Table

Path	**Method**	**Description**
/auth/login/pelanggan	POST	Endpoint untuk aplikasi mobile pelanggan, pengguna memasukkan credential login lalu akan mendapatkan JWT Token untuk mengakses enpoint API lain nya
/auth/login/mitra	POST	Endpoint untuk aplikasi mobile mitra, dan Web Dashboard. pengguna memasukkan credential login lalu akan mendapatkan JWT Token untuk mengakses enpoint API lain nya
/auth/profile	GET	Endpoint untuk decode JWT payload

2. Pelanggan Endpoint Endpoint ini berhubungan dengan entity pelanggan pada model prisma.

Tabel 2. Pelanggan Endpoint Table

Path	Method	Description
/pelanggan	GET	Endpoint untuk mengambil seluruh pelanggan terdaftar pada aplikasi
/pelanggan	POST	Endpoint untuk membuat akun pelanggan baru
/pelanggan/{id}	GET	Endpoint untuk mengambil akun pelanggan dengan id tertentu
/pelanggan/{id}	PUT	Endpoint untuk memperbarui akun pelanggan dengan id tertentu
/pelanggan/{id}	DELETE	Endpoint untuk menghapus akun pelanggan dengan id tertentu

3. Karyawan Endpoint Endpoint ini berhubungan dengan entity karyawan pada model prisma.

Tabel 3. Karyawan Endpoint Table

Path	Method	Description
/karyawan	GET	Endpoint untuk mengambil seluruh karyawan terdaftar pada aplikasi
/karyawan	POST	Endpoint untuk membuat akun karyawan baru
/karyawan/{id}	GET	Endpoint untuk mengambil akun karyawan dengan id tertentu
/karyawan/{id}	PUT	Endpoint untuk memperbarui akun karyawan dengan id tertentu
/karyawan/{id}	DELETE	Endpoint untuk menghapus akun karyawan dengan id tertentu
/karyawan/mitra/{mitraId}	GET	Endpoint untuk mengambil seluruh karyawan terdaftar pada aplikasi dan id mitra tertentu

4. Mitra Endpoint Endpoint mitra

Tabel 4. Mitra Endpoint Table

Path	Method	Description
/mitra	GET	Endpoint untuk mengambil seluruh mitra terdaftar pada aplikasi
/mitra	POST	Endpoint untuk membuat akun mitra baru
/mitra/{id}	GET	Endpoint untuk mengambil mitra dengan id tertentu
/mitra/{id}	PUT	Endpoint untuk mengaupdate mitra dengan id tertentu
/mitra/{id}	DELETE	Endpoint untuk menghapus mitra dengan id tertentu

5. Produk Endpoint Endpoint produk

Tabel 5. Produk Endpoint Table

Path	Method	Description
/produk	GET	Endpoint untuk mengambil seluruh produk yang terdaftar pada aplikasi
/produk	POST	Endpoint untuk membuat produk baru pada mitra tertentu
/produk/{id}	GET	Endpoint untuk mengambil prduk dengan id tertentu
/produk/{id}	PUT	Endpoint untuk memperbarui produk dengan id tertentu
/produk/{id}	DELETE	Endpoint untuk menghapus produk dengan id tertentu
/produk/mitra/{mitraId}	GET	Endpoint untuk mengambil produk dari id mitra tertentu

6. Pesanan Endpoint pesanan endpoint

Tabel 6. Pesanan Endpoint Table

Path	Method	Description
/pesanan	GET	Endpoint untuk mengambil seluruh pesanan
/pesanan	POST	Endpoint untuk membuat pesanan baru
/pesanan/{invoice}	GET	Endpoint untuk mengambil pesanan dengan invoice tertentu
/pesanan/{invoice}	PUT	Endpoint untuk memperbarui pesanan dengan invoice tertentu
/pesanan/{invoice}	DELETE	Endpoint untuk menghapus pesanan dengan invoice tertentu
/pesanan/mitra/{mitraId}	GET	Endpoint untuk mengambil seluruh pesanan dari id mitra tertentu

7. OrderList Endpoint Orderlist endpoint

Tabel 7. OrderList Endpoint Table

Path	Method	Description
/orderlist	POST	Endpoint untuk membuat orderlist baru
/orderlist/{id}	GET	Endpoint untuk mengambil orderlist dengan id tertentu
/orderlist/{id}	PUT	Endpoint untuk memperbarui orderlist dengan id tertentu
/orderlist/{id}	DELETE	Endpoint untuk menghapus orderlist dengan id tertentu
/orderlist/invoice/{invoice}	GET	Endpoint untuk mengambil orderlist dengan invoice pesanan tertentu

8. Antrian Endpoint Antrian endpoint

Tabel 8. Antrian Endpoint Table

Path	Method	Description
/antrian	POST	Endpoint untuk membuat antrian baru
/antrian/{id}	GET	Endpoint untuk mengambil antrian dengan id tertentu
/antrian/{id}	PUT	Endpoint untuk memperbarui antrian dengan id tertentu
/antrian/{id}	DELETE	Endpoint untuk menghapus antrian dengan id tertentu
/antrian/mitra/{mitraId}	GET	Endpoint untuk mengambil antrian pada mitra tertentu

9. Reviews Endpoint Reviews endpoint

Tabel 9. Reviews Endpoint Table

Path	Method	Description
/reviews	GET	Endpoint untuk mengambil seluruh review
/reviews	POST	Endpoint untuk membuat review baru
/reviews/mitra/{mitraId}	GET	Endpoint untuk mengambil review pada mitra tertentu
/reviews/{mitraId}/{pelangganId}	GET	Endpoint untuk mengambil review pelanggan pada mitra tertentu
/reviews/{mitraId}/{pelangganId}	PUT	Endpoint untuk memperbarui review pelanggan pada mitra tertentu

Continued on next page

Tabel 9 – continued from previous page

Path	Method	Description
/reviews/{mitraId}/{pelangganId}	DELETE	ndpoint untuk menghapus review pelanggan pada mitra tertentu

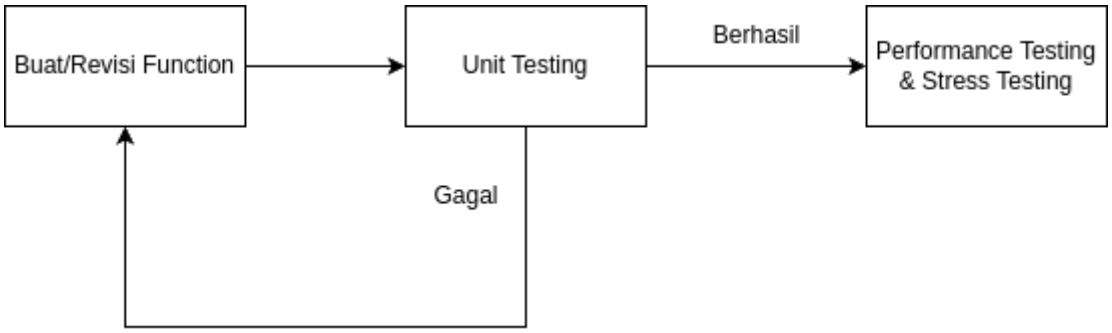
10. Image server endpoint endpoint berikut untuk menyajikan gambar ke client

Tabel 10. Image server Endpoint Table

Path	Method	Description
/image/{fileName}	GET	Endpoint untuk menampilkan gambar dengan filename tertentu

3.4 Alur Pengujian

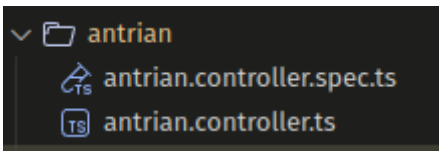
Pengujian menggunakan metode white box testing yaitu unit testing. Testing dilakukan pada setiap fungsi yang telah dibuat tanpa terhubung ke komponen lain. Setelah semua fungsi lulus unit testing, dilakukan performance dan load testing menggunakan library K6 pada node.js untuk mnegukur performansi aplikasi. Environment Testing menggunakan lokal build pada laptop penulis dan menggunakan tunnelling Cloudflare untuk mengukur performa network dan cpu load nya. Diagram alur pengujian dapat dilihat pada gambar 4.



Gambar 4. Alur Testing

3.4.1 Unit Testing

Pengujian dimulai dengan membuat file yang bernama ¡component¿.spec.ts seperti terlihat pada gambar 5. Lalu pada file tersebut penulis mendeskripsikan test apa yang akan dibuat seperti contoh pada listing 11, pertama difinisikan parameter yang diperlukan function lalu expect hasil keluaran fungsi sesuai dengan yang ditentukan.



Gambar 5. File spec.ts

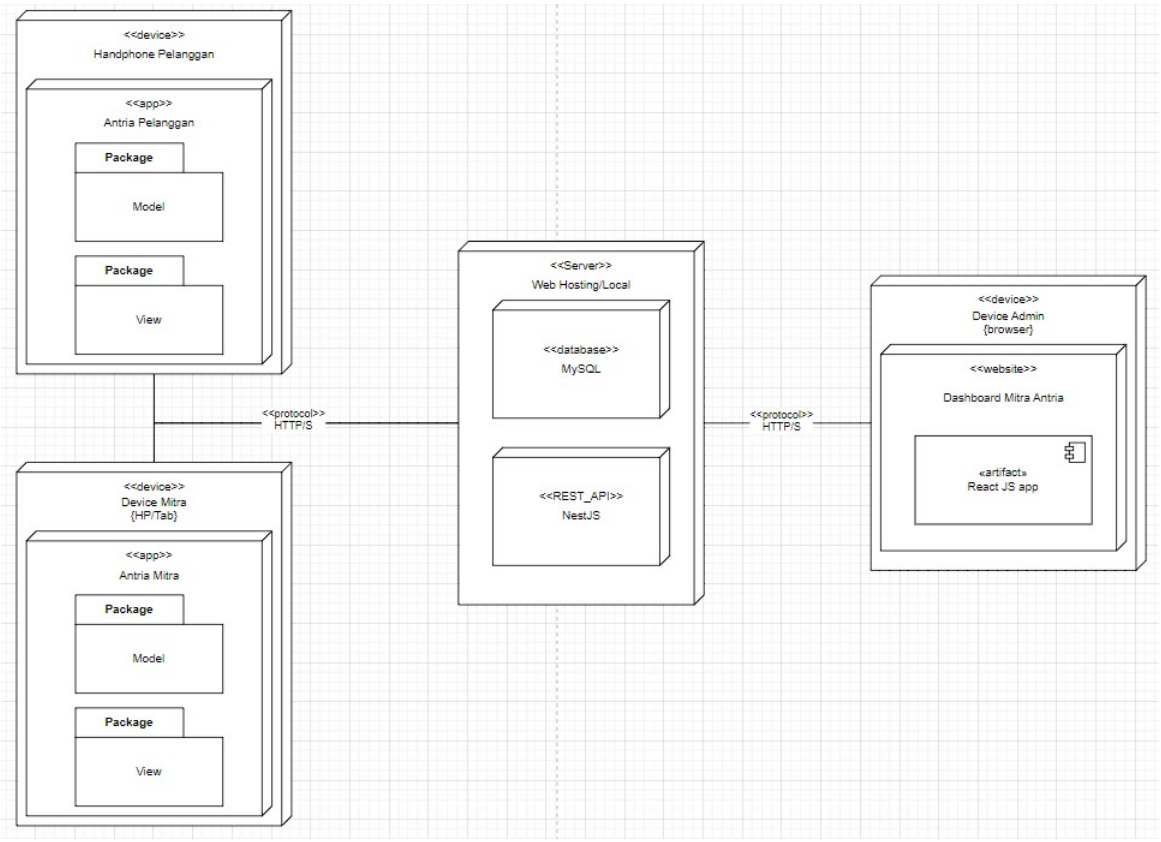
Listing 11: Contoh Testing Menggunakan Jest

```
describe('canActivate', () => {
it('should return true if user role is karyawan', async () => {
  const mockRequest = { user: { role: 'karyawan' } };
  const mockContext = { switchToHttp: () => ({ getRequest: () => mockRequest }) } as unknown
      as ExecutionContext;

  expect(await guard.canActivate(mockContext)).toBe(true);
```

```
  });

  it('should return false if user role is not karyawan', async () => {
    const mockRequest = { user: { role: 'admin' } };
    const mockContext = { switchToHttp: () => ({ getRequest: () => mockRequest }) } as unknown
        as ExecutionContext;

    expect(await guard.canActivate(mockContext)).toBe(false);
  });
});
```

3.5 Deployment

Ketika backend selesai di develope dan siap dipakai maka akan dilakukan deployment. Pada diagram 6 menjelaskan bagaimana server backend berkomunikasi dengan Aplikasi lain melalui protokol HTTPS.



Gambar 6. Deployment Diagram

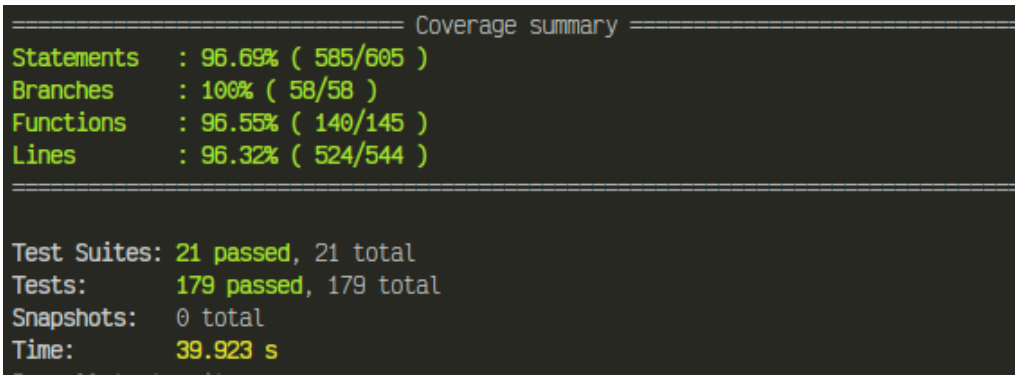
4. Evaluasi

4.1 Hasil Pengujian

Terdapat 2 Test suite pada setiap entity kecuali auth dimana auth memiliki 3 test suite yang ditotal sebanyak 21 test suite yang perlu dibuat dan dilakukan.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	96.69	100	96.55	96.32	
antrian	100	100	100	100	
antrian.controller.ts	100	100	100	100	
antrian.service.ts	100	100	100	100	
auth	100	100	100	100	
auth.controller.ts	100	100	100	100	
auth.guards.ts	100	100	100	100	
auth.service.ts	100	100	100	100	
constants.ts	100	100	100	100	
auth/dto	100	100	100	100	
loginMitra.dto.ts	100	100	100	100	
loginPelanggan.dto.ts	100	100	100	100	
image	100	100	100	100	
image.controller.ts	100	100	100	100	
image.service.ts	100	100	100	100	
karyawan	93.1	100	93.75	92.59	
karyawan.controller.ts	88.23	100	87.5	87.5	45-48
karyawan.service.ts	100	100	100	100	
mitra	94.59	100	93.33	94.02	
mitra.controller.ts	88.57	100	87.5	87.87	51-54
mitra.service.ts	100	100	100	100	
orderlist	100	100	100	100	
orderlist.controller.ts	100	100	100	100	
orderlist.service.ts	100	100	100	100	
pelanggan	92	100	92.3	91.3	
pelanggan.service.ts	100	100	100	100	
pelangganController.ts	87.87	100	83.33	87.09	47-50
pelanggan/dto	100	100	100	100	
createPelanggan.dto.ts	100	100	100	100	
pesanan	100	100	100	100	
pesanan.controller.ts	100	100	100	100	
pesanan.service.ts	100	100	100	100	
produk	88.23	100	87.5	86.66	
produk.controller.ts	78.94	100	77.77	77.77	37-40,61-64
produk.service.ts	100	100	100	100	
review	100	100	100	100	
review.controller.ts	100	100	100	100	
review.service.ts	100	100	100	100	

Gambar 7. Hasil unit testing menggunakan jest



Gambar 8. Coverage Summary

4.2 Analisis Hasil Pengujian

Analisis merupakan salah satu bagian yang penting untuk TA. Pada TA S1 tidak dituntut untuk mendapatkan hasil performasi yang lebih bagus dibandingkan dengan baseline yang populer, yang dituntut adalah membuat analisis yang lengkap. Menganalisis pengaruh kondisi-kondisi yang berbeda (seperti parameter, jenis data, threshold, dan sub-sistem) yang digunakan.

Cara sitasi adalah sebagai berikut: [**?**] untuk buku, [**?**] untuk *paper*, dan [**?**] untuk website.

5. Kesimpulan

Bagian Kesimpulan memuat kesimpulan dan Saran (*Future Work*), bisa dituliskan dalam poin-poin ataupun paragraf-paragraf. Semua poin kesimpulan diambil dari hasil pengujian dan analisis hasil pengujian sehingga tidak ada kesimpulan dari teori ataupun nalar semata. Sebagaimana sudah disebutkan pada bagian sebelumnya, pengujian dan analisis harus sesuai dengan tujuan TA. Jadi kesimpulan-kesimpulan yang dituliskan selaras dengan seluruh tujuan TA.

Daftar Pustaka

[1] E. Aghajani, C. Nagy, G. Bavota, and M. Lanza. A large-scale empirical study on linguistic antipatterns affecting apis. In *Soft System Stakeholder Analysis Methodology*, pages 25–35. Institute of Electrical and Electronics Engineers Inc., 11 2018.

[2] F. S. Alshraiedeh and N. Katuk. A uri parsing technique and algorithm for anti-pattern detection in restful web services. *International Journal of Web Information Systems*, 17:1–17, 1 2021.

[3] M. I. Beer and M. F. Hassan. Adaptive security architecture for protecting restful web services in enterprise computing environment. *Service Oriented Computing and Applications*, 12:111–121, 6 2018.

[4] M. Ghazal, R. Hamouda, and S. Ali. A smart mobile system for the real-time tracking and management of service queues. *International Journal of Computing and Digital Systems*, 5:305–313, 7 2016.

[5] P. Giessler, M. Gebhart, D. Sarancin, R. Steinegger, and S. Abeck. Best practices for the design of restful web services. In *International Conferences of Software Advances (ICSEA)*, pages 392–397, 2015.

[6] K. Gos and W. Zabierowski. The comparison of microservice and monolithic architecture. In *2020 IEEE XVI-th International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE, 2020.

[7] P. Jatkiewicz and S. Okrój. Differences in performance, scalability, and cost of using microservice and monolithic architecture. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 1038–1041, 2023.

[8] Y. L. Khong, B. C. Ooi, K. E. Tan, S. A. B. Ibrahim, and P. L. Tee. E-queue mobile application. In *SHS Web of Conferences*, volume 33, page 00033. EDP Sciences, 2017.

[9] M. Lorenz, J.-P. Rudolph, G. Hesse, M. Uflacker, and H. Plattner. Object-relational mapping revisited-a quantitative study on the impact of database technology on o/r mapping strategies. *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[10] K. Mysliwiec. Nestjs documentation.

[11] Q. Odeniran, H. Wimmer, and C. M. Rebman. Node.js or php? determining the better website server backend scripting language. *Issues in Information Systems*, 24:328–341, 2023.

[12] A. D. Pham. Developing back-end of a web application with nestjs framework: Case: Integrify oy's student management system. *Theseus*, 2020.

[13] I. Prisma Data. Prismajs documentation.

[14] A. Rahmatullo, A. P. Aldya, and M. N. Arifin. Stateless authentication with json web tokens using rsa-512 algorithm. *JURNAL INFOTEL*, 11(2):36–42, 2019.

[15] P. Runeson. A survey of unit testing practices. *IEEE software*, 23(4):22–29, 2006.

[16] H. Shah and T. R. Soomro. Node. js challenges in implementation. *Global Journal of Computer Science and Technology*, 17(2):73–83, 2017.

[17] M. N. Uddin, M. Rashid, M. Mostafa, S. Salam, N. Nithe, and S. Z. Ahmed. Automated queue management system, 2016.

[18] G. William, R. Anthony, and J. Purnama. Development of nodejs based backend system with multiple storefronts for batik online store. *ACM International Conference Proceeding Series*, 2020. Cited by: 0.

[19] D. Zmaranda, L.-L. Pop-Fele, C. Győrödi, R. Győrödi, and G. Pecherle. Performance comparison of crud methods using net object relational mappers: A case study, 2020.

Lampiran

Lampiran dapat berupa detil data dan contoh lebih lengkapnya, data-data pendukung, detail hasil pengujian, analisis hasil pengujian, detail hasil survey, surat pernyataan dari tempat studi kasus, screenshot tampilan sistem, hasil kuesioner dan lain-lain.