Festival Ticket API

Penjelasan Proyek

Festival Ticket API merupakan proyek fiktif sebagai studi kasus selama pelatihan Back-End Developer yang diselenggarakan oleh Dicoding, KADIN, dan IDCamp. Melalui proyek ini, siswa pelatihan diharapkan mampu mengimplementasikan pencapaian belajar melalui praktik langsung.

Menjalankan Proyek Awal

Instruksi yang diberikan di bawah ini dibuat tepat dan ringkas. Jika Anda memiliki pertanyaan atau ingin berdiskusi detail dari langkah yang disampaikan, bertanyalah pada salah satu instruktur.

Persiapan Database

Sebelum menjalankan proyek awal Festival Ticket API, Anda perlu menyiapkan database sebagai tempat penyimpanan yang digunakan pada proyek tersebut. Lakukan langkah-langkah berikut untuk membuat database-nya:

1. Buka Terminal dan masuk ke database postgres dengan perintah: `psql --username postgres --dbname postgres`.
2. Setelah masuk ke dalam database postgres, buat database bernama `festival_ticket` dengan perintah: `CREATE DATABASE festival_ticket`.
3. Atur permission terhadap database tersebut agar dapat dikelola oleh user developer. Jalankan perintah di bawah ini.
   ```
   GRANT ALL ON DATABASE festival_ticket TO developer;
   ALTER DATABASE festival_ticket OWNER TO developer;
   ```
4. Setelah database dibuat dan permission diatur, keluar dari user postgres dengan perintah: `exit`.

Mengunduh dan Menjalankan Proyek Awal

Untuk menjalankan proyek awal Festival Ticket API lakukan langkah-langkah berikut.

1. Unduh proyek starter melalui tautan: festival-ticket-api.zip.
2. Buka proyek dengan menggunakan VSCode atau Text Editor yang Anda biasa gunakan.
3. Buka Terminal dan pasang seluruh dependencies dengan menggunakan perintah: `npm install`.
4. Setelah itu, lakukan migrasi database dengan menggunakan perintah: `npm run migrate up`.

5. Jalankan proyek dengan menggunakan nodemon (development mode) dengan perintah: `npm run dev`.

Menguji Proyek Awal dengan Postman

Setelah aplikasi Festival Ticket API berjalan, kita perlu menguji untuk memastikan semuanya berjalan dengan baik. Lakukan langkah-langkah berikut.

1. Unduh Postman Collection pada tautan berikut: postman-testing.zip.
2. Import Postman Collection hingga collection Festival Ticket API Test muncul pada aplikasi Postman.
3. Jalankan Postman Collection dan pastikan seluruh request yang berada di dalam folder *Users*, *Authentications*, dan *Festivals* lolos pengujian.

Eksplorasi Mandiri

Langkah selanjutnya adalah eksplorasi mandiri. Bacalah kode-kode yang disediakan pada proyek awal karena beberapa kode di dalamnya memiliki penjelasan dalam bentuk komentar. Jika Anda memiliki pertanyaan atau ingin berdiskusi detail dari kode yang disediakan, bertanyalah pada salah satu instruktur.

Message Broker: Membuat Fitur Booking Festival

Instruksi yang diberikan di bawah ini dibuat tepat dan ringkas. Jika Anda memiliki pertanyaan atau ingin berdiskusi detail dari langkah yang disampaikan, bertanyalah pada salah satu instruktur.

Penjelasan Fitur

Fitur Booking merupakan fitur yang memperbolehkan pengguna melakukan pemesanan tiket pada sebuah festival. Fitur ini menggunakan komunikasi secara asynchronous yang memanfaatkan RabbitMQ message broker dalam melakukan perhitungan total tiket yang harus dibayar dan tautan konfirmasi yang dikirim melalui email pengguna.

Jika pengguna mengakses tautan konfirmasi, itu artinya ia sudah melakukan pembayaran dan status booking akan berubah menjadi terkonfirmasi.

Selain memesan tiket, pengguna juga bisa membatalkan pesanan tersebut dan akan mendapatkan pemberitahuan pembatal via email.

Berikut spesifikasi endpoint yang harus Anda bangun.

Method	Path	Auth	Body	Penjelasan

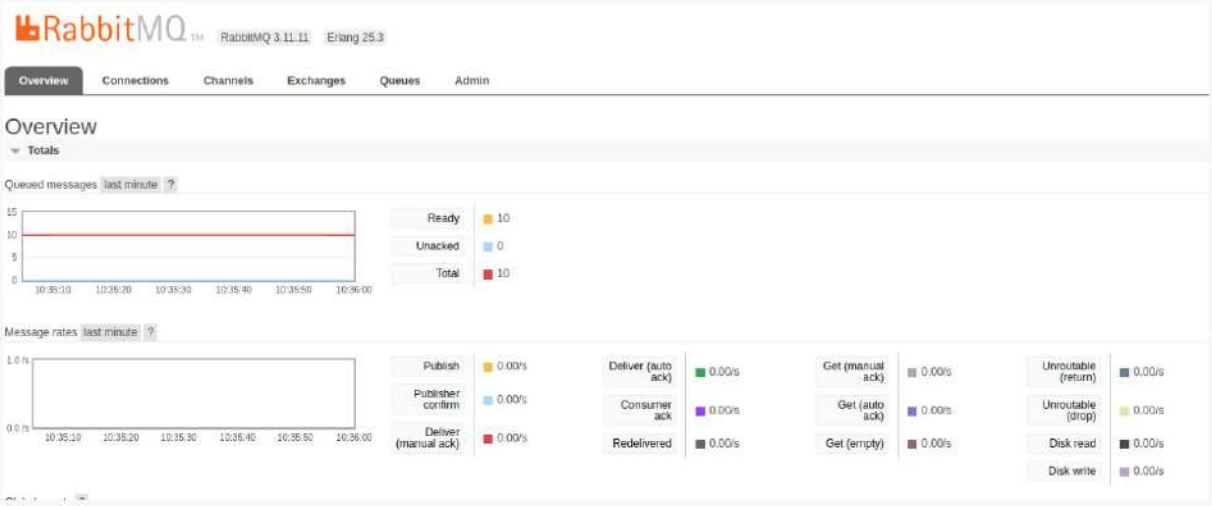
POST	/bookings	JWT	{
			festivalId: string (required),
			quantity: number (required),
			bookingDate: date iso format (required)
			}
				Digunakan untuk memesan tiket festival. Pengguna harus mengirimkan infrormasi festivalId, quantity, dan bookingDate pada body request.
				
				Jika permintaan berhasil, pengguna akan dikirim email berupa konfirmasi pemesanan.

GET	/bookings/confirms/{confirmationCode}	JWT		Digunakan untuk konfirmasi pemesanan. Konfirmasi kode dikirim via email. Jika pengguna mengirim permintaan dengan kode yang valid, pesanan akan terkonfirmasi.
GET	/bookings/{id}	JWT		Mendapatkan detail dari booking seperti id, festivalId, userId, hingga *status booking**.
DELETE	/bookings/{id}	JWT		Digunakan untuk membatalkan booking.

*) Status booking terdiri dari tiga status: 0 -> belum terkonfirmasi, 1-> terkonfirmasi, -1 -> dibatalkan.

Persiapan Infrastruktur

Untuk persiapan Infrastruktur, pastikan komputer/laptop Anda sudah terpasang RabbitMQ secara lokal.
Pastikan Anda bisa mengkases RabbitMQ dashboard melalui URL http://localhost:15672 dengan username: **guest** dan password: **guest**.



Jika belum, silakan pasang terlebih dulu. Jangan sungkan untuk meminta bantuan salah satu instruktur jika mengalami kendala.

Membuat Tabel Bookings

Ikuti langkah-langkah berikut untuk membuat tabel `bookings` di database postgres dengan teknik migrations.

1. Buka Terminal (VSCode) dan tulis perintah berikut untuk membuat migrasi baru: `npm run migrate create "create table bookings"`.
2. Buka berkas migrasi yang baru dibuat di dalam folder *migrations/xxx_create-table-bookings.js*.
3. Tulis kode migrasi di bawah ini.

```
exports.up = (pgm) => {
  pgm.createTable('bookings', {
    id: {
      type: 'VARCHAR(50)',
      primaryKey: true,
    },
    user_id: {
      type: 'VARCHAR(50)',
      notNull: true,
    },
    festival_id: {
      type: 'VARCHAR(50)',
      notNull: true,
    },
    booking_date: {
      type: 'DATE',
```

```
      notNull: true,
    },
    quantity: {
      type: 'SMALLINT',
      notNull: true,
    },
    total_price: {
      type: 'DECIMAL(10,2)',
    },
    status: {
      type: 'SMALLINT',
      notNull: true,
    },
    created_at: {
      type: 'DATE',
      notNull: true,
    },
    confirmation_code: {
      type: 'VARCHAR(50)',
    },
  });

  pgm.addConstraint('bookings', 'fk_bookings.user_id_users.id',
'FOREIGN KEY(user_id) REFERENCES users(id) ON DELETE CASCADE');
  pgm.addConstraint('bookings',
'fk_bookings.festival_id_festivals.id', 'FOREIGN KEY(festival_id)
REFERENCES festivals(id) ON DELETE CASCADE');
};

exports.down = (pgm) => {
  pgm.dropTable('bookings');
};
```

4. Jalankan up migration untuk mengeksekusi kode tersebut dengan perintah: `npm run migrate up`.
5. Tabel `bookings` berhasil dibuat.

Menambahkan Environment Variable dan Konfigurasi Terkait RabbitMQ

Ikuti langkah-langkah berikut untuk menambahkan environment variable dan konfigurasi terkait RabbitMQ:

1. Buka berkas `.env` dan tambahkan baris kode berikut untuk menambahkan nilai `RABBITMQ_SERVER` pada environment variable.

```
# RabbitMQ
RABBITMQ_SERVER=amqp://localhost
```

2. Buka berkas *src/utils/config.js* dan tambahkan properti `rabbitMQ.server` di dalam objek `config`.

```
rabbitMq: {
  server: process.env.RABBITMQ_SERVER,
},
```

Membuat QueueService

QueueService merupakan object yang digunakan untuk berkomunikasi dengan message broker via Node.js. Di sini kita akan membuat method `sendMessage()` yang digunakan untuk mengirimkan pesan ke queue.

Ikuti langkah-langkah berikut untuk membuat QueueService.

1. Buka Terminal (VSCode) dan pasang package amqplib dengan menggunakan perintah: `npm install amqplib`.
2. Buat folder dan berkas baru pada alamat src/services/**rabbitmq/QueueService.js**.
3. Di dalamnya, tulis kode berikut.

```
const amqp = require('amqplib');
const config = require('../../utils/config');

// this class contains all the methods needed to handle rabbitmq
server
class QueueService {
 /**
  * This method is used to send message to a queue in rabbitmq server
  */
 async sendMessage(queue, message) {
   // connect to rabbitmq server
   const connection = await amqp.connect(config.rabbitMq.server);
   // create channel that will be used to send message
   const channel = await connection.createChannel();

   // assert the queue to make sure the queue is exist
   // if the queue is not exist, it will be created automatically
   await channel.assertQueue(queue, {
     durable: true,
   });
```

```
    // send message to the queue
    await channel.sendToQueue(queue, Buffer.from(message));

    // close the connection after the message is sent
    // we give it 1-second delay to make sure the message is sent
before the connection is closed
    setTimeout(() => {
      connection.close();
    }, 1000);
  }
}

module.exports = QueueService;
```

4. Amati dan pahami kode tersebut melalui komentar yang disediakan. Silakan tanyakan kepada salah satu instruktur jika ada yang ingin ditanyakan.

Membuat BookingsService

BookingsService merupakan objek yang digunakan untuk menangani seluruh operasi database yang berhubungan dengan tabel `bookings`.

Ikut langkah-langkah berikut untuk membuat BookingsService.

1. Buat berkas baru pada alamat: src/services/postgres/BookingsService.js
2. Tulislah kode di bawah ini.

```
const { nanoid } = require('nanoid');
const { createPool } = require('./pool');

// this class will be used to handle all the database operations
related to bookings table
class BookingsService {
  constructor() {
    this._pool = createPool();
  }

  /**
  * this method will be used to add booking to database
  */
  async persistBooking({
    userId, festivalId, bookingDate, quantity,
```

```
  }) {
    const bookingsId = `booking-${nanoid(16)}`;

    const query = {
      text: 'INSERT INTO bookings VALUES($1, $2, $3, $4, $5, $6, $7,
$8, $9)',
      values: [bookingsId, userId, festivalId, bookingDate, quantity,
null, 0, new Date(), null],
    };

    await this._pool.query(query);

    return bookingsId;
  }

  /**
  * this method will be used to confirm bookings
  */
  async confirmBooking(confirmationCode) {
    const query = {
      text: 'UPDATE bookings SET status = 1 WHERE confirmation_code =
$1',
      values: [confirmationCode],
    };

    await this._pool.query(query);
  }

  /**
  * this method will be used to get user id by confirmation code
  */
  async getUserIdByConfirmationCode({ confirmationCode }) {
    const query = {
      text: 'SELECT user_id FROM bookings WHERE confirmation_code =
$1',
      values: [confirmationCode],
    };

    const { rows } = await this._pool.query(query);

    if (!rows.length) {
      return null;
    }
```

```
  return rows[0].user_id;
}

/**
* this method will be used to get user id by booking id
*/
async getUserIdByBookingId(bookingId) {
  const query = {
    text: 'SELECT user_id FROM bookings WHERE id = $1',
    values: [bookingId],
  };

  const { rows } = await this._pool.query(query);

  if (!rows.length) {
    return null;
  }

  return rows[0].user_id;
}

/**
* this method will be used to get booking by id
*/
async getBookingById(bookingId) {
  const query = {
    text: 'SELECT * FROM bookings WHERE id = $1',
    values: [bookingId],
  };

  const { rows } = await this._pool.query(query);

  const [booking] = rows;

  // delete the confirmation code from the booking object
  delete booking.confirmation_code;

  return booking;
}

/**
* this method will be used to softly delete booking
```

```
    */
    async softDeleteBooking(bookingId) {
      const query = {
        text: 'UPDATE bookings SET status = -1 WHERE id = $1',
        values: [bookingId],
      };

      await this._pool.query(query);
    }
}

module.exports = BookingsService;
```

3. Amati dan pahami kode tersebut melalui komentar yang disediakan. Silakan tanyakan kepada salah satu instruktur jika ada yang ingin ditanyakan.

Membuat Bookings Plugin

Bookings Plugin menyimpan dan mengagregasikan seluruh fitur yang berhubungan dengan fitur bookings. Di dalam plugin ini, kita akan terdiri dari routes, validator, dan handler. Di dalam plugin ini juga kita akan menggunakan service-service yang sudah kita buat terkait fitur bookings dan queue.

Ikuti langkah-langkah berikut untuk membuat Bookings plugin:

1. Buatlah folder baru bernama *bookings* pada *src/api*.
2. Di dalamnya, buatlah berkas *handler.js*, *index.js*, *routes.js*, dan *validator.js*.
3. Kita mulai dengan berkas handler.js terlebih dulu. Di dalam berkas handler.js, kita menuliskan logika bisnis dalam menangani permintaan yang masuk terkait fitur bookings. Bukalah berkas tersebut dan tulis kode berikut ini.

```
const ForbiddenError = require('../../exceptions/ForbiddenError');
const InvariantError = require('../../exceptions/InvariantError');

// this handler will be used to handle requests related to bookings
feature
class BookingsHandler {
 constructor(bookingsService, festivalsService, queueService, validator)
{
   this._bookingsService = bookingsService;
   this._festivalsService = festivalsService;
   this._queueService = queueService;
   this._validator = validator;
 }
```

```
 // this handler will be used to handle POST /bookings
 async postBookingHandler(request, h) {
   // get user id from request.auth.credentials
   const { id: userId } = request.auth.credentials;

   // validate the payload
   // if the payload is not valid, then the validator will throw an error
   // if the payload is valid, then the validator will return the payload
   const {
     festivalId,
     quantity, bookingDate,
   } = this._validator.validatePostBookingPayload(request.payload);

   // check if the festival is available
   const isFestivalAvailable = await
this._festivalsService.isFestivalAvailable(festivalId);

   // if the festival is not available, then throw an error
   if (!isFestivalAvailable) {
     throw new InvariantError('festival tidak valid');
   }

   // persist booking to database and get booking id for sending
confirmation email to queue
   const bookingId = await this._bookingsService.persistBooking({
     userId,
     festivalId,
     quantity,
     bookingDate,
   });

   // payload for sending confirmation email to queue
   const queueMessage = JSON.stringify({ bookingId });

   // send confirmation email to queue
   await this._queueService.sendMessage('booking:send_confirmation',
queueMessage);

   // return response with booking id
   return h.response({
     status: 'success',
     message: 'konfirmasi booking akan dikirimkan melalui email',
     data: {
       bookingId,
```

```
      },
    }).code(201);
  }

  // this handler will be used to handle GET
/bookings/confirm/{confirmationCode}
  async getBookingConfirmHandler(request) {
    const { id: userId } = request.auth.credentials;
    const { confirmationCode } = request.params;

    // get booking owner by confirmation code
    const owner = await
this._bookingsService.getUserIdByConfirmationCode(confirmationCode);

    // if the booking owner is not the same as the user id, then throw an
error
    if (owner !== userId) {
      throw new ForbiddenError('anda tidak berhak mengakses resource
ini');
    }

    // confirm booking to database
    const booking = await
this._bookingsService.confirmBooking(confirmationCode);

    // return response with booking data
    return {
      status: 'success',
      data: {
        booking,
      },
    };
  }

  // this handler will be used to handle GET /bookings
  async getBookingByIdHandler(request, h) {
    const { id: userId } = request.auth.credentials;
    const { id: bookingId } = request.params;

    // get booking owner by booking id
    const owner = await
this._bookingsService.getUserIdByBookingId(bookingId);

    // if the booking owner is not the same as the user id, then throw an
error
```

```
    if (owner !== userId) {
      throw new ForbiddenError('anda tidak berhak mengakses resource
ini');
    }

    // get booking data by booking id
    const booking = await this._bookingsService.getBookingById(bookingId);

    // create response with booking data
    const response = h.response({
      status: 'success',
      message: 'booking berhasil ditemukan',
      data: {
        booking,
      },
    });

    return response;
  }

  // this handler will be used to handle DELETE /bookings/{id}
  async deleteBookingByIdHandler(request) {
    const { id: userId } = request.auth.credentials;
    const { id: bookingId } = request.params;

    // get booking owner by booking id
    const owner = await
this._bookingsService.getUserIdByBookingId(bookingId);

    // if the booking owner is not the same as the user id, then throw an
error
    if (owner !== userId) {
      throw new ForbiddenError('anda tidak berhak mengakses resource
ini');
    }

    // soft delete booking by booking id
    await this._bookingsService.softDeleteBooking(bookingId);

    // send message to queue to delete booking
    await this._queueService.sendMessage('booking:delete',
JSON.stringify({ bookingId }));

    return {
      status: 'success',
```

```
      message: 'booking berhasil dihapus',
      data: {},
    };
  }
}

module.exports = BookingsHandler;
```

4. Buka berkas *routes.js* dan tulis kode di bawah ini.

```
const config = require('../../utils/config');

const routes = (handler) => [
  {
    method: 'POST',
    path: '/bookings',
    // the reason we used anonymous function here is because ...
    // we want to keep the context of this on the class handler
    handler: (request, h) => handler.postBookingHandler(request, h),
    options: {
      // set the authentication strategy to 'festival-ticket__api'
      auth: config.application.authenticationName,
    },
  },
  {
    method: 'GET',
    path: '/bookings/confirms/{confirmationCode}',
    handler: (request, h) => handler.getBookingConfirmHandler(request, h),
    options: {
      auth: config.application.authenticationName,
    },
  },
  {
    method: 'GET',
    path: '/bookings/{id}',
    handler: (request, h) => handler.getBookingByIdHandler(request, h),
    options: {
      auth: config.application.authenticationName,
    },
  },
  {
    method: 'DELETE',
    path: '/bookings/{id}',
    handler: (request, h) => handler.deleteBookingByIdHandler(request, h),
    options: {
```

```
      auth: config.application.authenticationName,
    },
  },
];

module.exports = routes;
```

5. Buka berkas *validator.js* dan tulis kode di bawah ini.

```
const Joi = require('joi');
const InvariantError = require('../../exceptions/InvariantError');

// Joi schema for POST /bookings
const PostBookingPayloadSchema = Joi.object({
 festivalId: Joi.string().required(),
 quantity: Joi.number().min(1).required(),
 bookingDate: Joi.date().iso().required(),
});

// this validator will be used to validate the payload from client
related to bookings feature
const BookingsValidator = {
 validatePostBookingPayload(payload) {
   const validationResult = PostBookingPayloadSchema.validate(payload);

   // if the payload is not valid, then the validator will throw an error
   if (validationResult.error) {
     throw new InvariantError(validationResult.error.message);
   }

   return validationResult.value;
 },
};

module.exports = BookingsValidator;
```

6. Terakhir, buka berkas index.js dan tulis kode di bawah ini.

```
const BookingsHandler = require('./handler');
const BookingsValidator = require('./validator');
const routes = require('./routes');

// this plugin will be used to create bookings feature
const bookings = {
```

```
 name: 'bookings',
 version: '1.0.0',
 register: async (server, { bookingsService, festivalsService,
queueService }) => {
   const bookingsHandler = new BookingsHandler(
     bookingsService,
     festivalsService,
     queueService,
     BookingsValidator,
   );

   server.route(routes(bookingsHandler));
 },
};

module.exports = bookings;
```

7. Amati dan pahami kode tersebut melalui komentar yang disediakan. Silakan tanyakan kepada salah satu instruktur jika ada yang ingin ditanyakan.

Menggunakan Bookings Plugin di HTTP Server

Setelah membuat Bookings Plugin, selanjutnya kita akan gunakan plugin tersebut di HTTP server agar pengguna (dan Anda) dapat menggunakannya melalui HTTP request (Postman).

Ikut langkah-langkah berikut untuk menggunakan Bookings Plugin di HTTP Server.

1. Buka berkas *src/http/createServer.js*.
2. Di dalam fungsi `createServer()`, buat instance `bookingsService` dan `queueService`.

```
// .. other code
const BookingsService =
require('../services/postgres/BookingsService');
const QueueService = require('../services/rabbitmq/QueueService');

async function createServer() {
  // create services that will be used by the plugin
  // .. other code
  const bookingsService = new BookingsService();
  const queueService = new QueueService();

  // .. other code
}
```

3. Kemudian daftarkan plugin bookings seperti di bawah ini.

```
// .. other code
const bookings = require('../api/bookings');

async function createServer() {
  // .. other code

  // register internal plugin
  await server.register([
    // .. other code,
    {
      plugin: bookings,
      options: {
        bookingsService,
        festivalsService,
        queueService,
      },
    },
  ]);
}
```

4. Jalankan aplikasi (server) menggunakan perintah: `npm run dev`.
5. Pastikan aplikasi berjalan lancar tanpa masalah.

Membuat Program Consumer untuk Queue Bookings

Di langkah sebelumnya, Anda sudah tahu bahwa pemesanan tiket dan pembatalan tiket mengirimkan sebuah pesan ke dalam queue. Program consumer yang akan Anda buat kali ini merupakan program yang akan menkonsumsi pesan, mengolahnya, dan mengirimkan email ke pemesan tiket.

Ada dua pesan yang perlu dikonsumsi, yakni ketika pemesanan tiket dan pembatalan tiket. Untuk pemesanan tiket, pesan yang ada di queue perlu diolah untuk proses perhitungan harga total, pembuatan tautan konfirmasi, kemudian mengirimkan kedua informasi tersebut melalui email ke pemesan. Untuk pembatalan tiket, pesan yang ada cukup diolah untuk mengirimkan pemberitahuan pembatalan ke pemesan melalui email.

Ikuti langkah-langkah berikut untuk membuat program consumer untuk queue bookings.

1. Di komputer Anda, buatlah folder baru bernama *festival-ticket-api-consumer*.
2. Buka folder tersebut menggunakan VSCode.
3. Di dalam VSCode, buka Terminal dan tulis perintah berikut untuk menginisialisasi proyek: `npm init --y`.

4. Pasang package yang dibutuhkan untuk menjadi dependencies menggunakan perintah berikut.
   ```
   npm install amqplib dotenv nanoid@3 nodemailer pg
   ```
5. Pasang juga package yang dibutuhkan untuk menjadi development dependencies menggunakan perintah berikut.
   ```
   npm install nodemon --save-dev
   ```
6. Buka berkas package.json dan ubah properti `"scripts"` menjadi seperti ini.
   ```
   "scripts": {
     "start:confirmation": "node src/consumer-confirmation.js",
     "start:deletion": "node src/consumer-deletion.js",
     "dev:confirmation": "nodemon src/consumer-confirmation.js",
     "dev:deletion": "nodemon src/consumer-deletion.js"
   },
   ```
7. Buat berkas **.env** yang digunakan untuk menampung nilai environment variable dan nantinya digunakan untuk konfigurasi beberapa hal terkait infrastruktur database dan queue.
8. Di dalam berkas .env, tulislah kode berikut.
   ```
   # Application
   APPLICATION_PORT=9000

   # Postgres Database
   PGUSER=developer
   PGHOST=localhost
   PGPASSWORD=supersecretpassword
   PGDATABASE=festival_ticket
   PGPORT=5432

   # RabbitMQ
   RABBITMQ_SERVER=amqp://localhost

   # Mail
   SMTP_HOST=sandbox.smtp.mailtrap.io
   SMTP_PORT=2525
   SMTP_USER=TODO
   SMTP_PASSWORD=TODO
   ```

   Nilai TODO nantinya kita akan ganti dengan SMTP user dan password dari layanan pengujian email, yakni mailtrap.io.

9. Buat folder bernama *src* untuk menampung seluruh kode program konsumer.
10. Di dalam folder src, buat folder dan berkas JavaScript baru yang berlokasi di *src/utils/config.js*.

11. Di dalam berkas config.js, tulis kode berikut.

```
const dotenv = require('dotenv');
dotenv.config();

// this is the config object that will be used throughout the
application
// the values are taken from the .env file
const config = {
  application: {
    port: process.env.APPLICATION_PORT,
  },
  postgres: {
    host: process.env.PGHOST,
    database: process.env.PGDATABASE,
    port: process.env.PGPORT,
    user: process.env.PGUSER,
    password: process.env.PGPASSWORD,
  },
  rabbitMq: {
    server: process.env.RABBITMQ_SERVER,
  },
  mail: {
    host: process.env.SMTP_HOST,
    port: Number(process.env.SMTP_PORT),
    user: process.env.SMTP_USER,
    password: process.env.SMTP_PASSWORD,
  }
}

module.exports = config;
```

12. Kembali ke folder src. Di dalam folder tersebut, buat dua buah berkas JavaScript bernama *consumer-confirmation.js* dan *consumer-deletion.js*. Kedua berkas tersebut nantinya akan mengonsumsi queue `bookings:send_confirmation` dan `bookings:delete`.
13. Untuk sementara, di dalam kedua berkas tersebut, tulislah kode TODO berikut.

```
async function run() {
  // @TODO
}

run();
```

14. Buat folder dan berkas JavaScript baru pada *src/services/DatabaseService.js*. Di dalamnya tulis kode berikut.

```
const { Pool } = require('pg');
const config = require("../utils/config");

// this is the service that will handle all database operations
class DatabaseService {
  constructor() {
    // create a new pool of connections
    this._pool = new Pool({
      host: config.postgres.host,
      database: config.postgres.database,
      port: config.postgres.port,
      user: config.postgres.user,
      password: config.postgres.password,
    });
  }

  // this method will be used to get the user information
  async getTicketPriceByFestivalId(festivalId) {
    const query = {
      text: `SELECT price FROM festivals WHERE id = $1`,
      values: [festivalId],
    };

    const result = await this._pool.query(query);

    return result.rows[0].price;
  }

  // this method will be used to get the booking information
  async getBookingInformationByBookingId(bookingId) {
    const query = {
      text: `SELECT bookings.quantity, bookings.festival_id
             FROM bookings
             WHERE bookings.id = $1`,
      values: [bookingId],
    };

    const result = await this._pool.query(query);

    return result.rows[0];
  }
```