САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 2

Выполнил:
Байков Иван
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

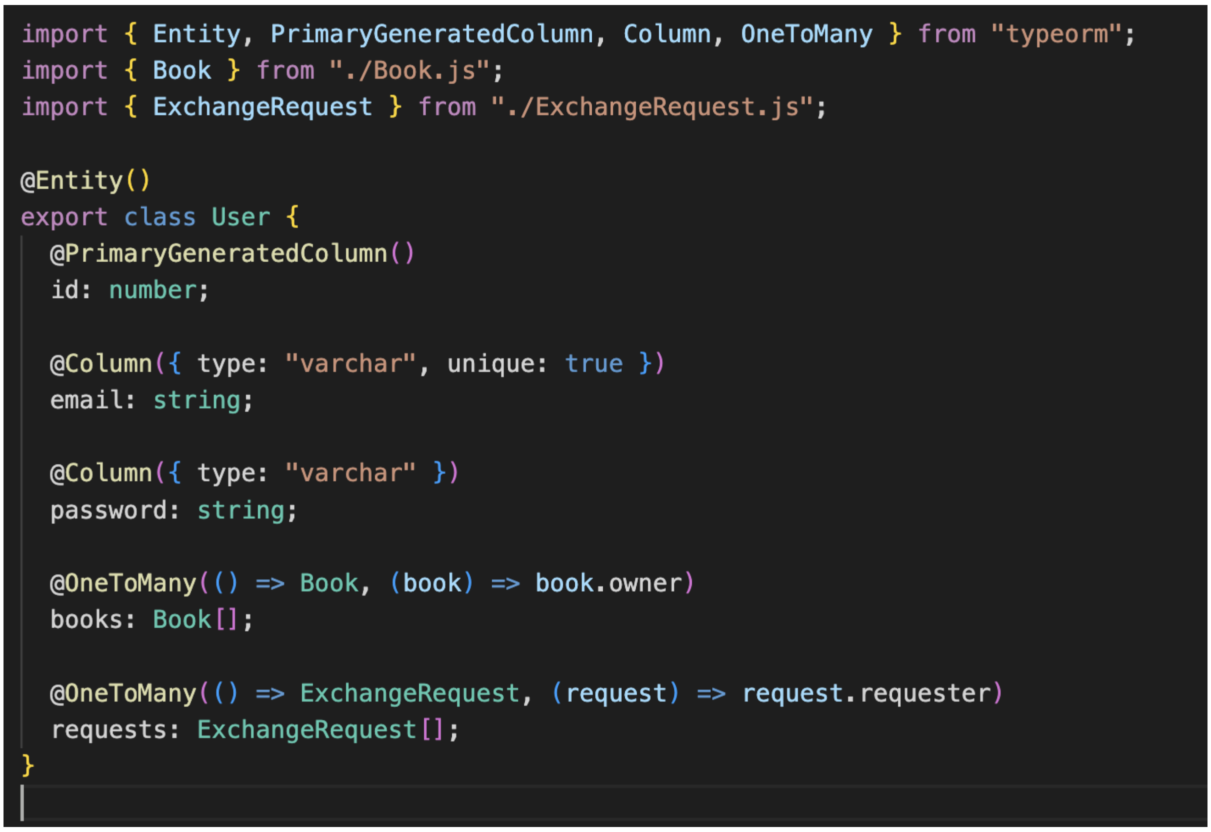
Задача

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

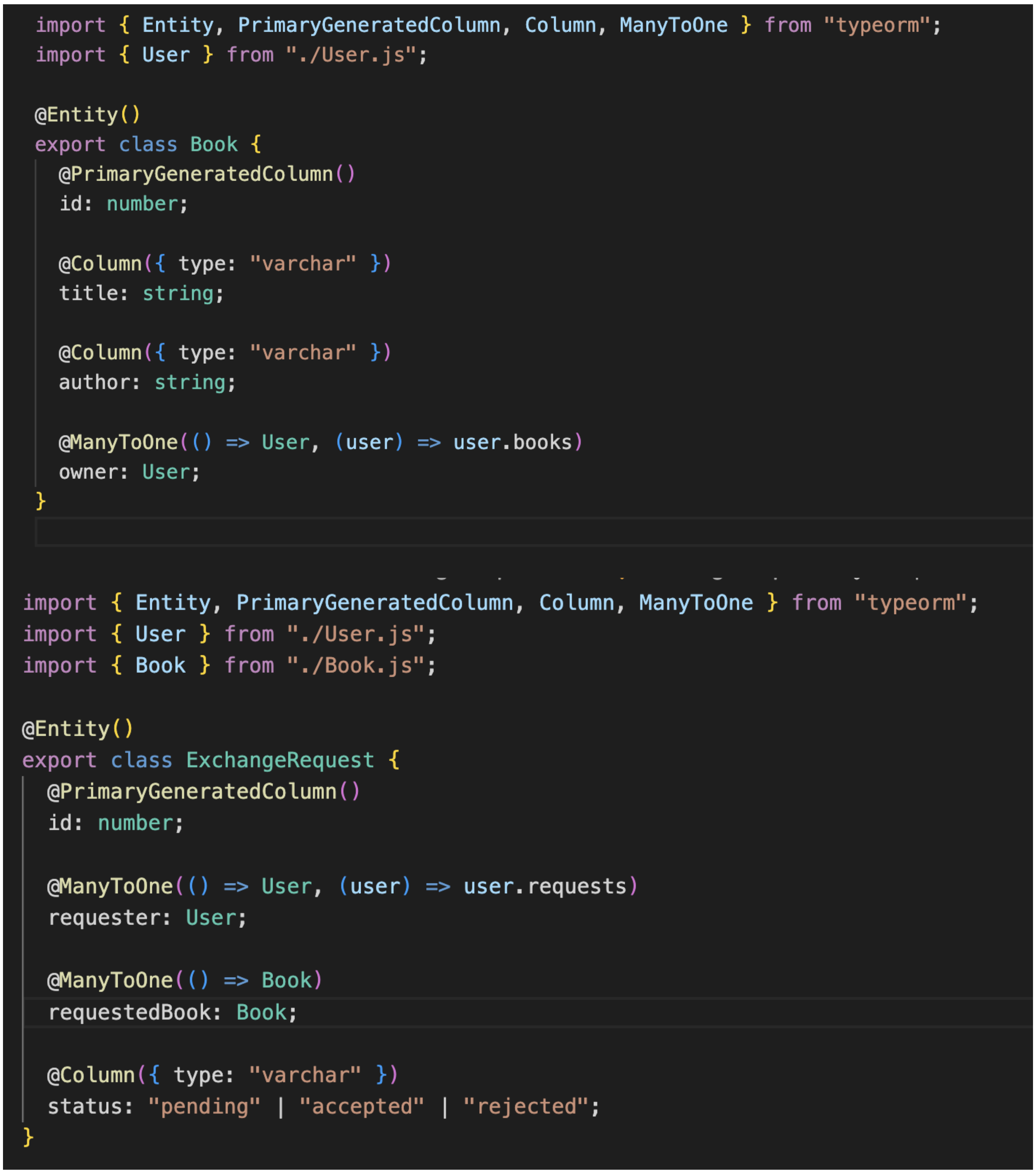
Сервис для буккроссинга. Требуемый функционал: регистрация, авторизация, создание списка своих книг, создание заявок на обмен книгами, работа с заявками на обмен.

Ход работы

1) Создадим все необходимые модели:



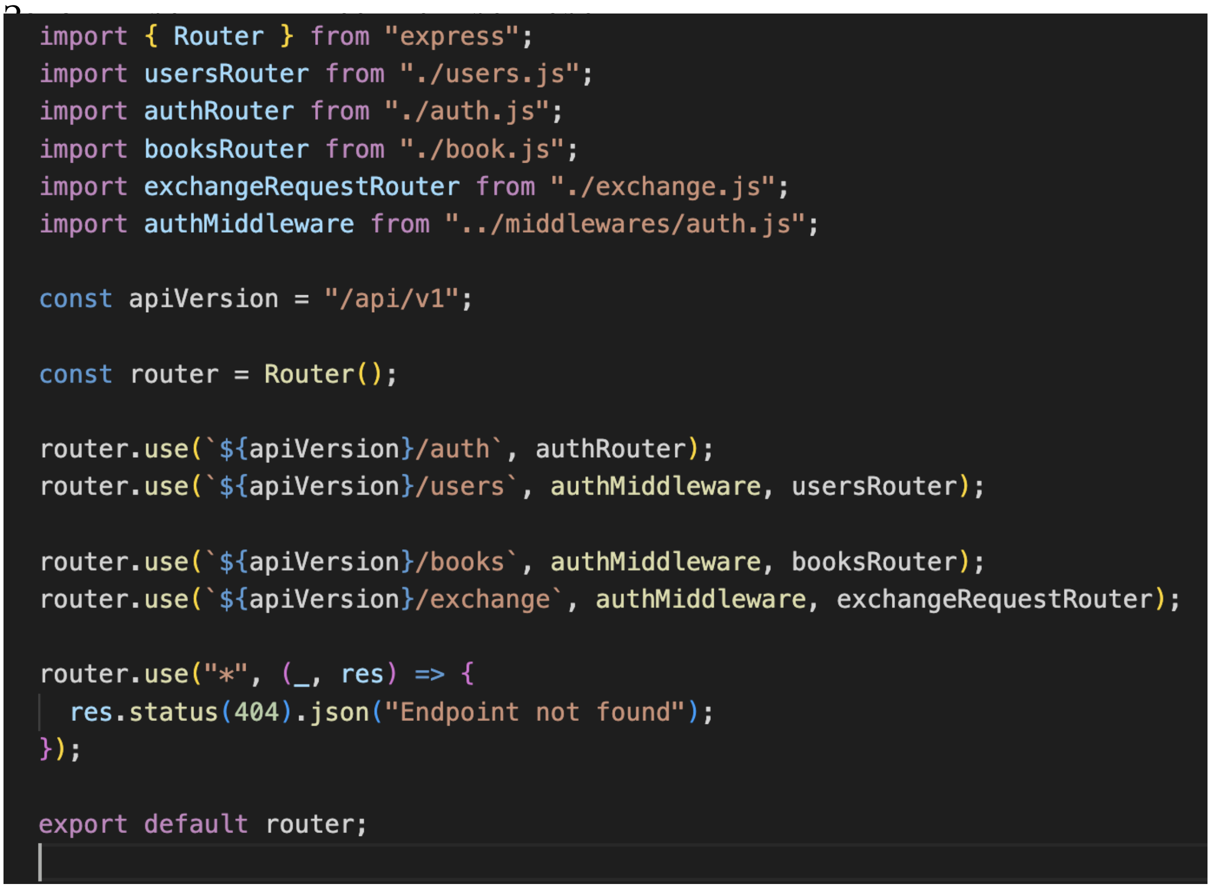
```
import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from "typeorm";
import { Book } from "./Book.js";
import { ExchangeRequest } from "./ExchangeRequest.js";

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ type: "varchar", unique: true })
  email: string;

  @Column({ type: "varchar" })
  password: string;

  @OneToMany(() => Book, (book) => book.owner)
  books: Book[];

  @OneToMany(() => ExchangeRequest, (request) => request.requester)
  requests: ExchangeRequest[];
}
```



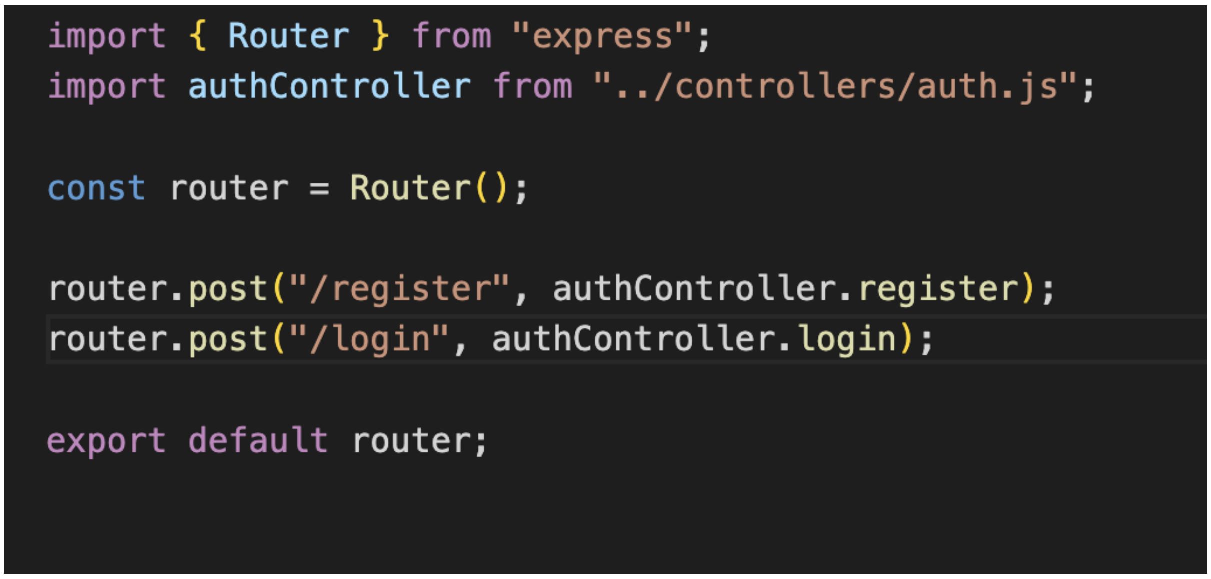
```typescript
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm";
import { User } from "./User.js";

@Entity()
export class Book {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ type: "varchar" })
  title: string;

  @Column({ type: "varchar" })
  author: string;

  @ManyToOne(() => User, (user) => user.books)
  owner: User;
}
```

```typescript
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm";
import { User } from "./User.js";
import { Book } from "./Book.js";

@Entity()
export class ExchangeRequest {
  @PrimaryGeneratedColumn()
  id: number;

  @ManyToOne(() => User, (user) => user.requests)
  requester: User;

  @ManyToOne(() => Book)
  requestedBook: Book;

  @Column({ type: "varchar" })
  status: "pending" | "accepted" | "rejected";
}
```

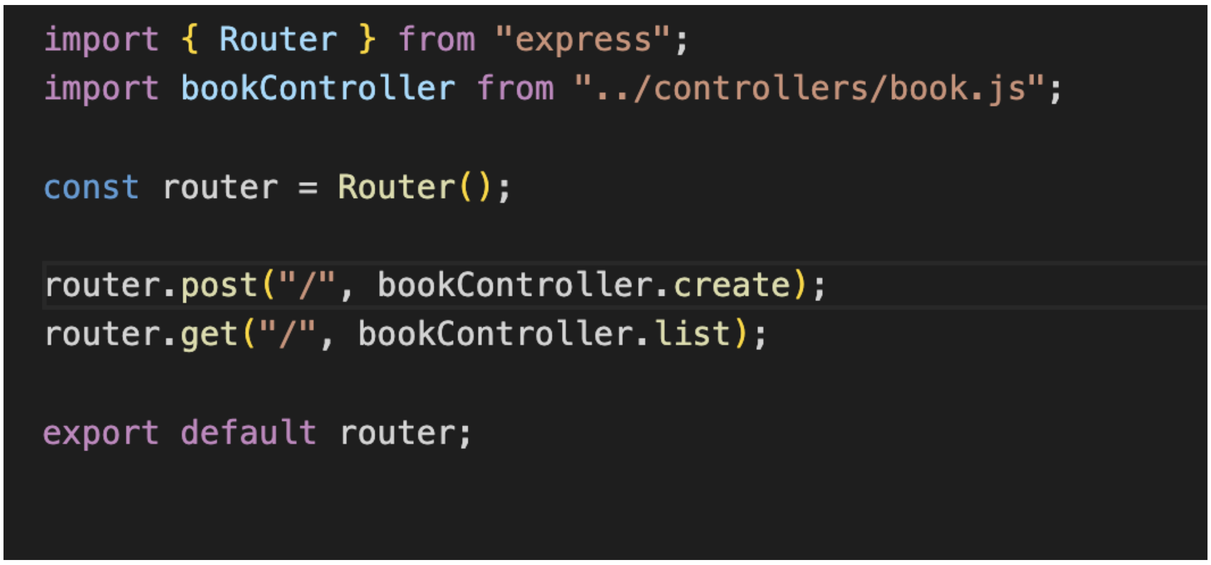
2)



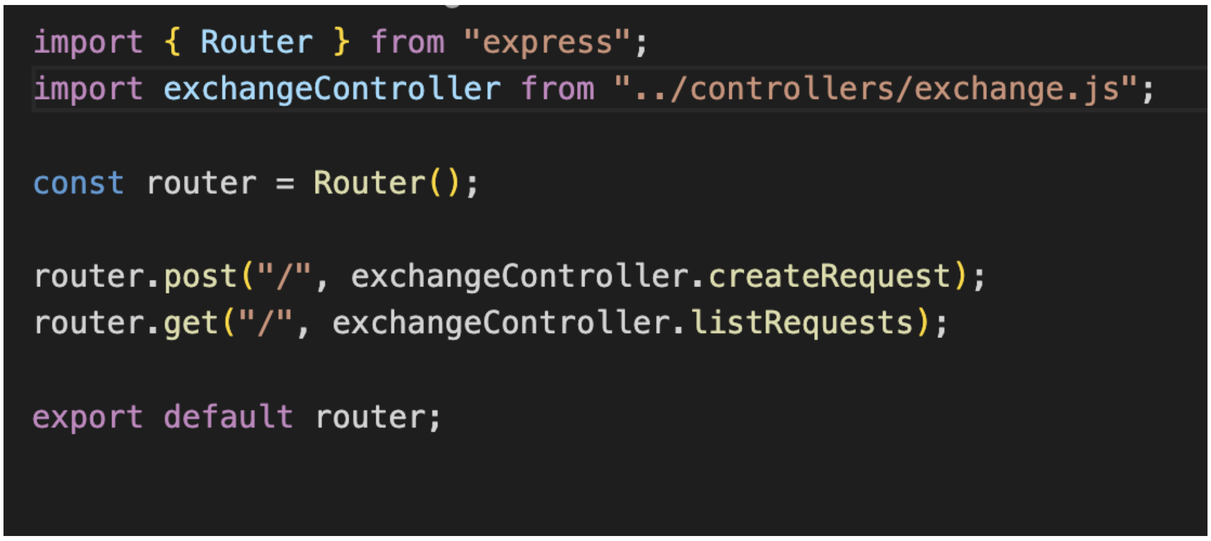
```
import { Router } from "express";
import usersRouter from "./users.js";
import authRouter from "./auth.js";
import booksRouter from "./book.js";
import exchangeRequestRouter from "./exchange.js";
import authMiddleware from "../middlewares/auth.js";

const apiVersion = "/api/v1";

const router = Router();

router.use(`${apiVersion}/auth`, authRouter);
router.use(`${apiVersion}/users`, authMiddleware, usersRouter);

router.use(`${apiVersion}/books`, authMiddleware, booksRouter);
router.use(`${apiVersion}/exchange`, authMiddleware, exchangeRequestRouter);

router.use("*", (_, res) => {
  res.status(404).json("Endpoint not found");
});

export default router;
```



```
import { Router } from "express";
import authController from "../controllers/auth.js";

const router = Router();

router.post("/register", authController.register);
router.post("/login", authController.login);

export default router;
```

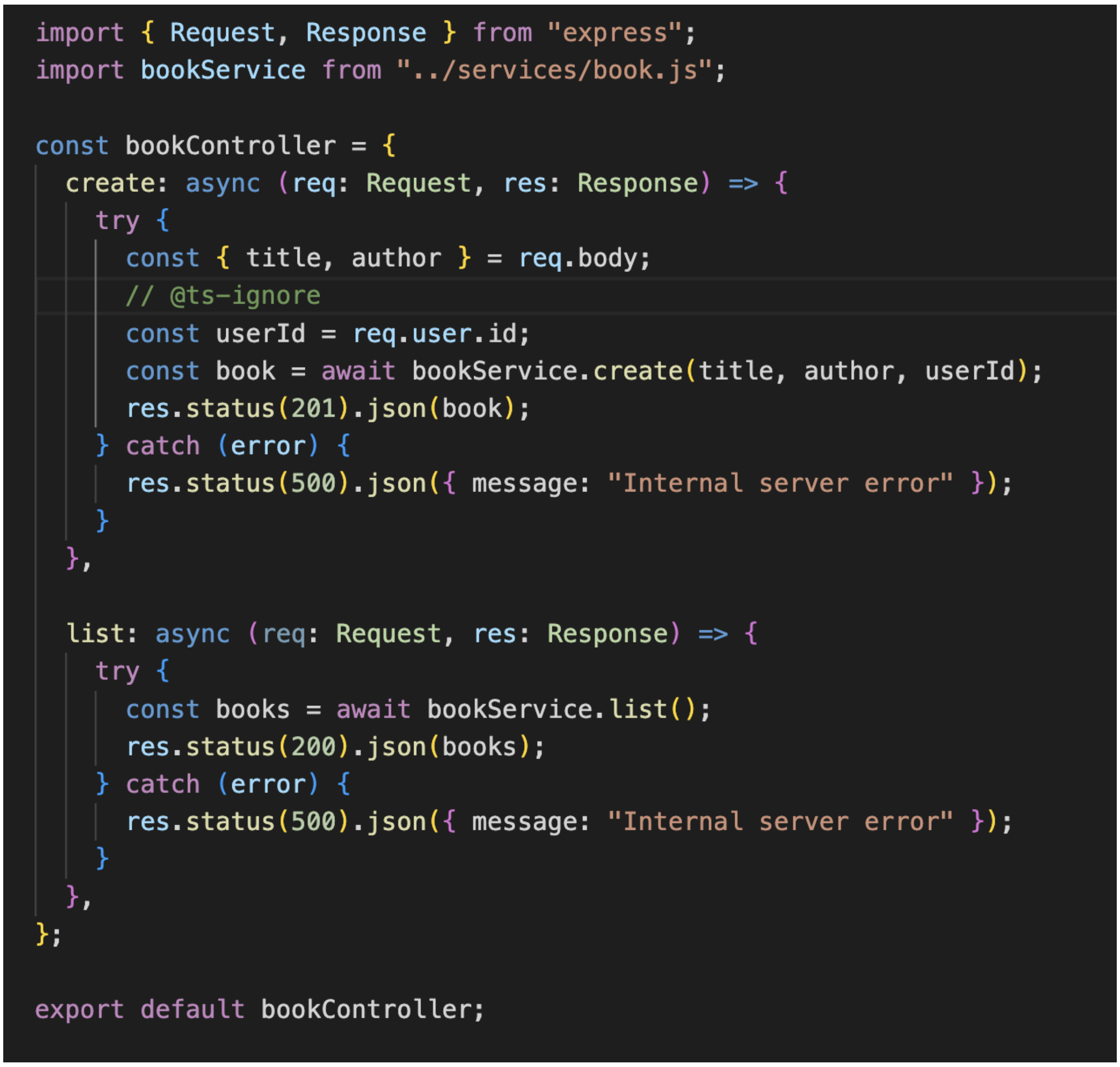


```
import { Router } from "express";
import bookController from "../controllers/book.js";

const router = Router();

router.post("/", bookController.create);
router.get("/", bookController.list);

export default router;
```

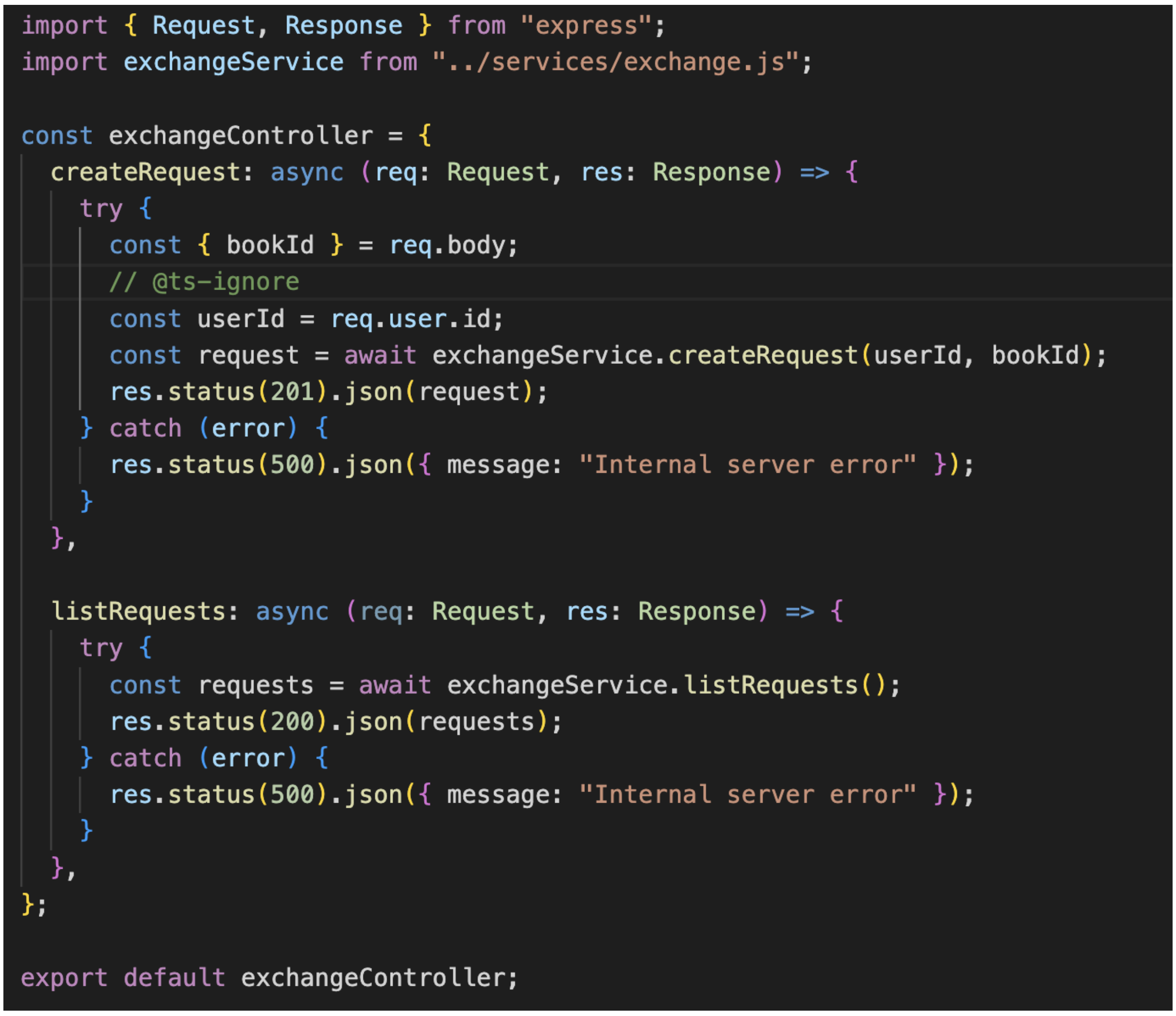


```
import { Router } from "express";
import exchangeController from "../controllers/exchange.js";

const router = Router();

router.post("/", exchangeController.createRequest);
router.get("/", exchangeController.listRequests);

export default router;
```

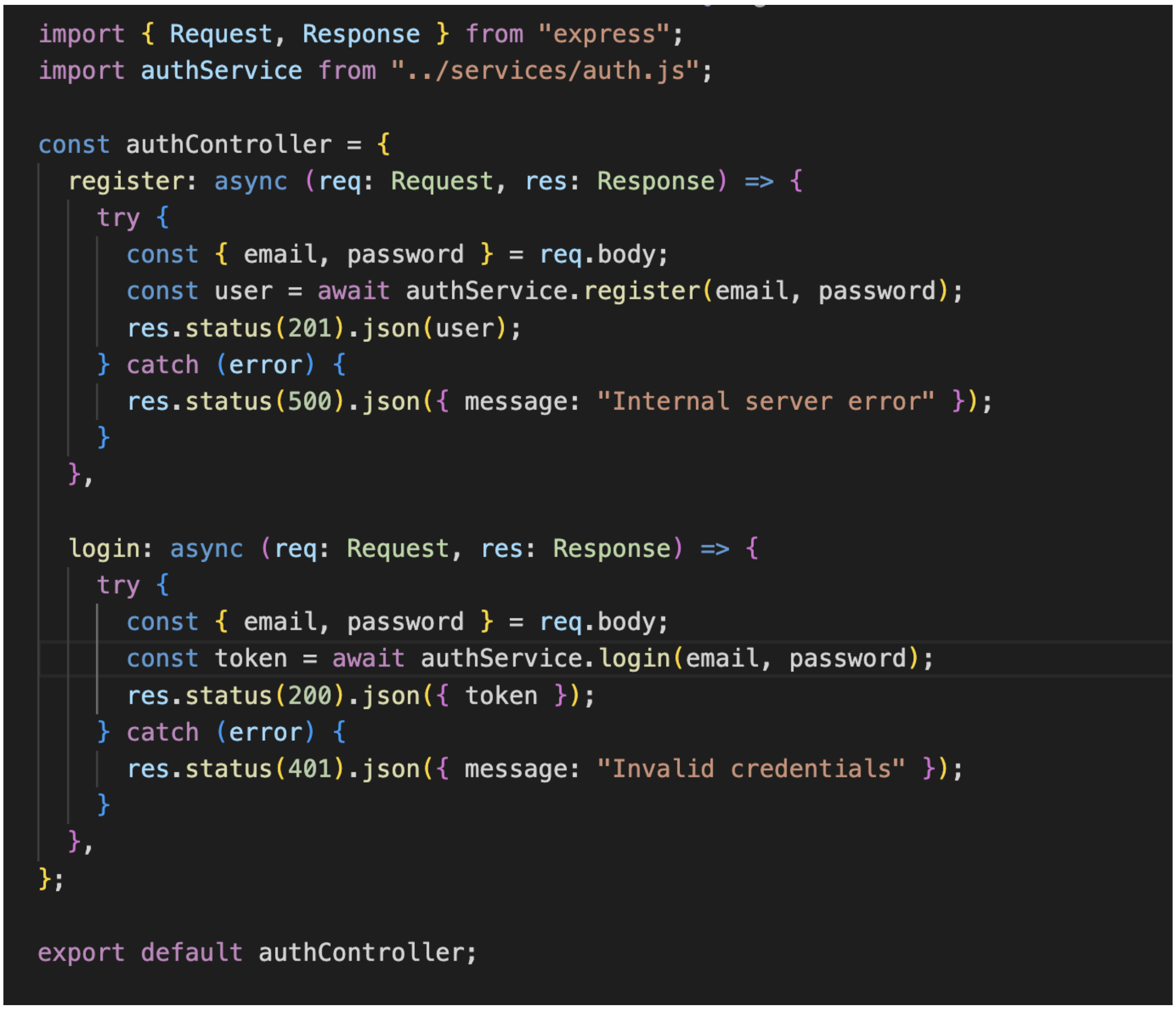
3) Напишем контроллеры для всех эндпоинтов



```typescript
import { Request, Response } from "express";
import bookService from "../services/book.js";

const bookController = {
  create: async (req: Request, res: Response) => {
    try {
      const { title, author } = req.body;
      // @ts-ignore
      const userId = req.user.id;
      const book = await bookService.create(title, author, userId);
      res.status(201).json(book);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },

  list: async (req: Request, res: Response) => {
    try {
      const books = await bookService.list();
      res.status(200).json(books);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },
};

export default bookController;
```

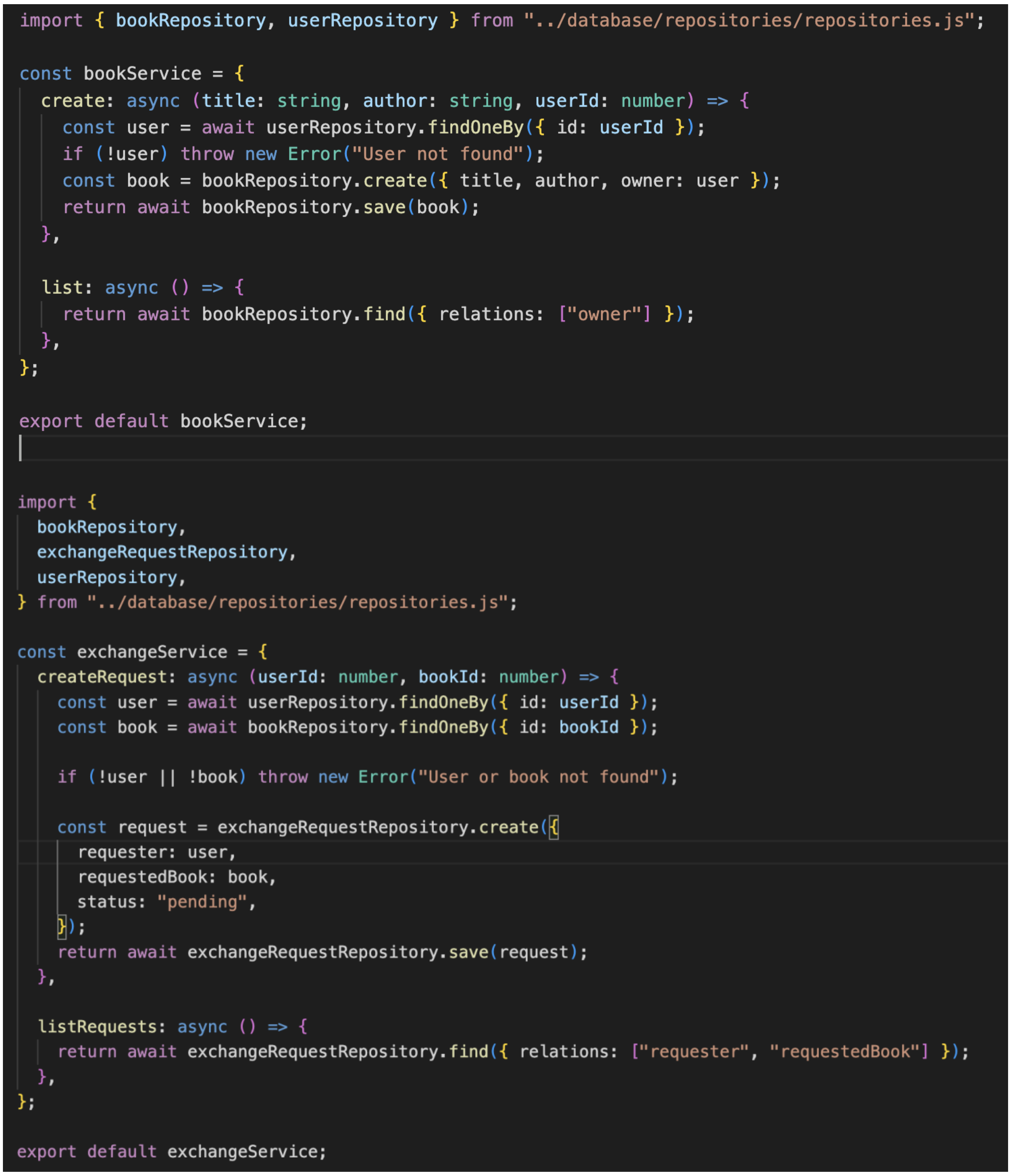


```
import { Request, Response } from "express";
import exchangeService from "../services/exchange.js";

const exchangeController = {
  createRequest: async (req: Request, res: Response) => {
    try {
      const { bookId } = req.body;
      // @ts-ignore
      const userId = req.user.id;
      const request = await exchangeService.createRequest(userId, bookId);
      res.status(201).json(request);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },

  listRequests: async (req: Request, res: Response) => {
    try {
      const requests = await exchangeService.listRequests();
      res.status(200).json(requests);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },
};

export default exchangeController;
```



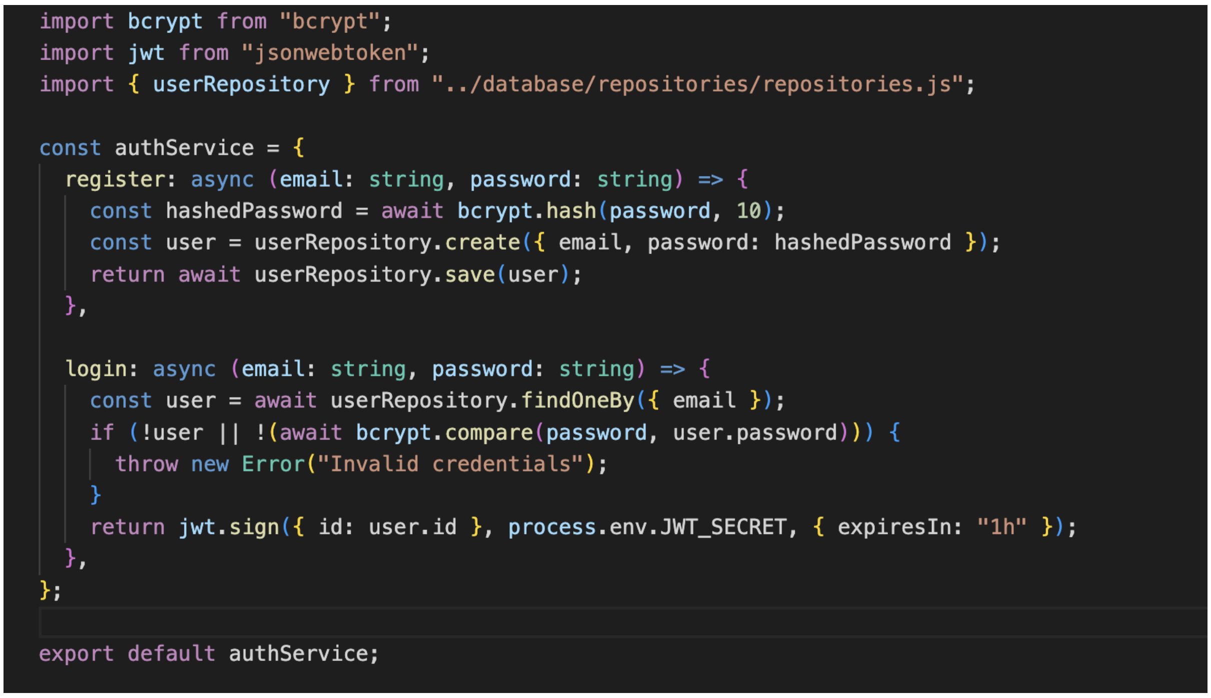
```
import { Request, Response } from "express";
import authService from "../services/auth.js";

const authController = {
  register: async (req: Request, res: Response) => {
    try {
      const { email, password } = req.body;
      const user = await authService.register(email, password);
      res.status(201).json(user);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },

  login: async (req: Request, res: Response) => {
    try {
      const { email, password } = req.body;
      const token = await authService.login(email, password);
      res.status(200).json({ token });
    } catch (error) {
      res.status(401).json({ message: "Invalid credentials" });
    }
  },
};

export default authController;
```

4) Создадим сервисы для всех моделей



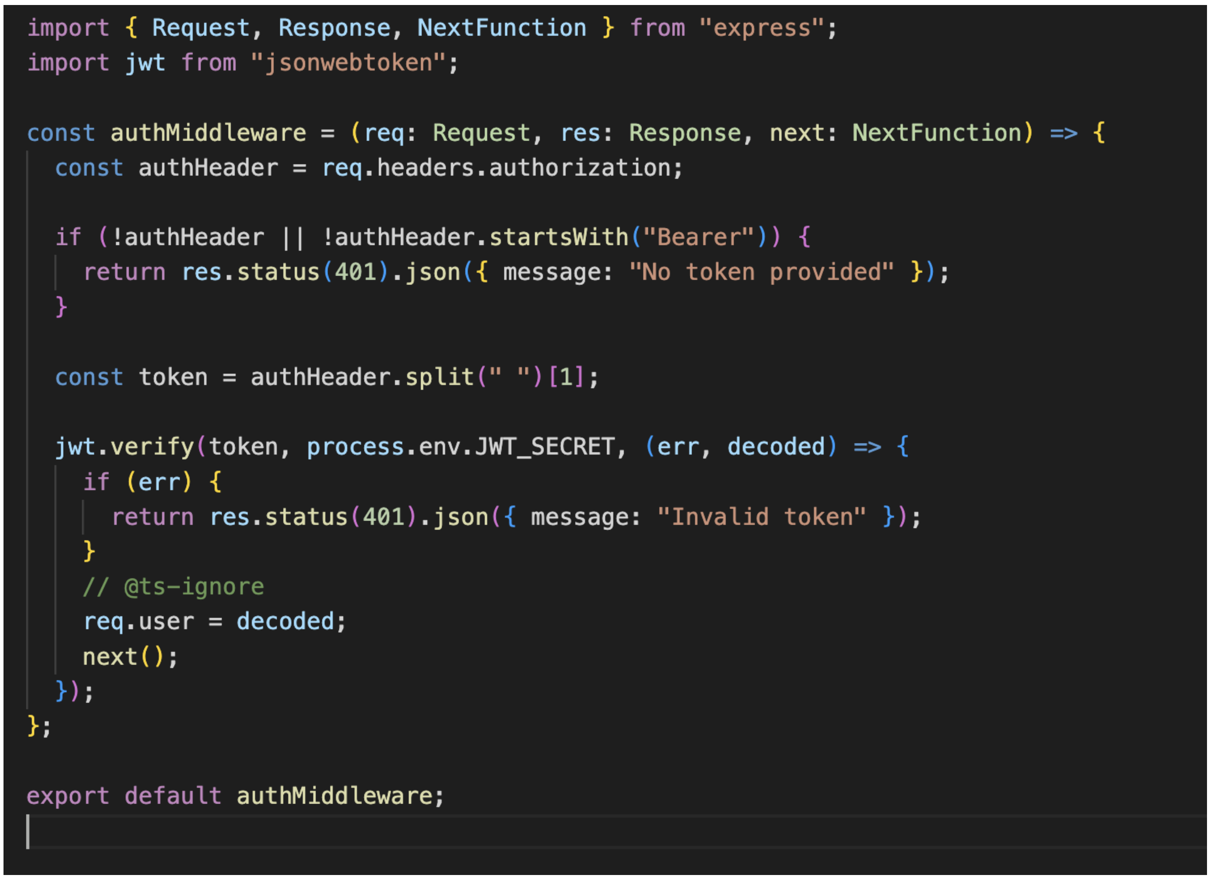
```
import { bookRepository, userRepository } from "../database/repositories/repositories.js";

const bookService = {
  create: async (title: string, author: string, userId: number) => {
    const user = await userRepository.findOneBy({ id: userId });
    if (!user) throw new Error("User not found");
    const book = bookRepository.create({ title, author, owner: user });
    return await bookRepository.save(book);
  },

  list: async () => {
    return await bookRepository.find({ relations: ["owner"] });
  },
};

export default bookService;

```

```
import {
  bookRepository,
  exchangeRequestRepository,
  userRepository,
} from "../database/repositories/repositories.js";

const exchangeService = {
  createRequest: async (userId: number, bookId: number) => {
    const user = await userRepository.findOneBy({ id: userId });
    const book = await bookRepository.findOneBy({ id: bookId });

    if (!user || !book) throw new Error("User or book not found");

    const request = exchangeRequestRepository.create({
      requester: user,
      requestedBook: book,
      status: "pending",
    });
    return await exchangeRequestRepository.save(request);
  },

  listRequests: async () => {
    return await exchangeRequestRepository.find({ relations: ["requester", "requestedBook"] });
  },
};

export default exchangeService;
```



```
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import { userRepository } from "../database/repositories/repositories.js";

const authService = {
  register: async (email: string, password: string) => {
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = userRepository.create({ email, password: hashedPassword });
    return await userRepository.save(user);
  },

  login: async (email: string, password: string) => {
    const user = await userRepository.findOneBy({ email });
    if (!user || !(await bcrypt.compare(password, user.password))) {
      throw new Error("Invalid credentials");
    }
    return jwt.sign({ id: user.id }, process.env.JWT_SECRET, { expiresIn: "1h" });
  },
};

export default authService;
```

5) Создадим auth middleware



```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";

const authMiddleware = (req: Request, res: Response, next: NextFunction) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith("Bearer")) {
    return res.status(401).json({ message: "No token provided" });
  }

  const token = authHeader.split(" ")[1];

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) {
      return res.status(401).json({ message: "Invalid token" });
    }
    // @ts-ignore
    req.user = decoded;
    next();
  });
};

export default authMiddleware;
```

Вывод

В ходе лабораторной работы был реализован RESTful api с использованием express, typescript, typeorm на базе boilerplate’а из предыдущей работы