

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бек-энд разработка

Отчет

Лабораторная Работа №1

Выполнил:

Кобелев Л.К.

К33401

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

Определим структуру проекта:

```
.
├── controllers
├── core
├── database
│   └── database.sqlite
├── middlewares
├── models
├── repositories
├── routes
└── services
```

- Core — основной класс для работы с приложением, в нём происходит подключение роутов и базы данных;
- Database — класс для подключения к БД;
- Routes — роуты;
- Middlewares — middleware, например, проверка токенов;
- Models — модели;
- Controllers, services, repositories — классы для управления данными, реализуют одноименный паттерн.

Дополнительно есть файл с переменными окружениям:

```
PORT=3000
HOST=localhost
DB_NAME=database
JWT_SECRET=hello
JWT_LIFETIME=1h
```

Рассмотрим пример каждого типа класса по отдельности.

database:

```
export const AppDataSource = new DataSource({
  type: "sqlite",
  database: path.join(__dirname, process.env.DB_NAME + ".sqlite"),
  synchronize: true,
  logging: false,
  entities: [__dirname + "../models/*.js"],
  migrations: [],
  subscribers: [],
});
```

Было решено использовать TypeORM. Путь к БД строится через .env, а в сущности попадают все модели.

Модель пользователя (User):

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

  @Column()
  username: string

  @Column()
  password: string

  @Column()
  tokenVersion: number

  @ManyToMany(() => RandomEntity)
  @JoinTable()
  randomEntities: RandomEntity[]

  hashPassword() {
    this.password = bcrypt.hashSync(this.password, 8)
  }

  checkIfUnencryptedPasswordIsValid(unencryptedPassword: string) {
    return bcrypt.compareSync(unencryptedPassword, this.password)
  }
}
```

Есть все стандартные поля, плюс:

- Версия токена, чтобы только один токен был валидным;
- Некая М-М связь, чтобы проверить возможности TypeORM;

- Парочка методов для работы с паролями.

Роуты User:

```
import express from "express"
import AuthController from "../controllers/AuthController"
import { checkJWT } from "../middlewares/checkJWT"

const router: express.Router = express.Router()
const authController = new AuthController()

router
  .route('/login')
  .post(authController.login)

router
  .route('/signup')
  .post(authController.signup)

router
  .route('/me')
  .get(checkJWT, authController.me)

export default router
```

Рассмотрим checkJWT:

```
const authService = new AuthService()

export const checkJWT = async (request: Request, response: Response, next: NextFunction) => {
  const token = <string>request.headers["auth"]
  let jwtPayload

  try {
    jwtPayload = <any>jwt.verify(token, process.env.JWT_SECRET as string)
    response.locals.jwtPayload = jwtPayload
    const user = await authService.getUserTokenVersion(jwtPayload.userId)
    if (user.tokenVersion !== jwtPayload.v) {
      throw {
        status: 404,
        message: "Not Found",
      };
    }
  } catch (error) {
    response.status(401).send({
      error: "Invalid token"
    })
    return
  }

  next()
}
```

Как видно, он обращается к AuthService:

```

const userRepository = new UserRepository
const randomEntityService = new RandomEntityService
|
class UserService {
  async getAll() {
    return userRepository.readAll()
  }

  async getById(id: number) {
    return userRepository.readById(id)
  }

  async getByUsername(username: string) {
    return userRepository.readByUsername(username)
  }

  async addRandomEntity(username: string, randomEntityId: number) {
    let randomEntity = await randomEntityService.getById(randomEntityId)
    return userRepository.updateUserRandomEntities(username, randomEntity, true)
  }

  async deleteRandomEntity(username: string, randomEntityId: number) {
    let randomEntity = await randomEntityService.getById(randomEntityId)
    return await userRepository.updateUserRandomEntities(username, randomEntity, false)
  }
}

export default UserService

```

Который в свою очередь обращается к UserRepository:

```

import { AppDataSource } from "../database/data-source"
import { User } from "../models/User"

const userRepository = AppDataSource.getRepository(User)

class UserRepository {
  async readAll() {
    return await (userRepository.find({
      select: ["id", "username"]
    }))
  }

  async readById(id: number) {
    return await userRepository.findOneOrFail({
      select: ["id", "username"],
      relations: {
        randomEntities: true,
      },
      where: { id: id }
    })
  }

  async readTokenVersion(id: number) {
    return await userRepository.findOneOrFail({
      select: ["id", "username", "tokenVersion"],
      where: { id: id }
    })
  }
}

```

Таким образом происходит “разграничение ответственности”, уменьшение связности и дублирования кода.

Дополнительно есть контроллеры, которые тоже обращаются к сервисам:

```
import { Request, Response } from "express"
import UserService from "../services/User"

const userService = new UserService

class UserController {
  getAllUsers = async (request: Request, response: Response) => {
    const allUsers = userService.getAll()
    return response.send(allUsers)
  }

  addUserRandomEntity = async (request: Request, response: Response) => {
    let randomEntityId = Number(request.params.id)
    let username = response.locals.jwtPayload.username
    let user = await userService.addRandomEntity(username, randomEntityId)
    return response.send(user)
  }

  removeUserRandomEntity = async (request: Request, response: Response) => {
    let randomEntityId = Number(request.params.id)
    let username = response.locals.jwtPayload.username
    let user = await userService.deleteRandomEntity(username, randomEntityId)
    return response.send(user)
  }

  getMyRandomEntities = async (request: Request, response: Response) => {
    let username = response.locals.jwtPayload.username
    let user = await userService.getByUsername(username)
    return response.send(user)
  }
}

export default UserController
```

Вывод

По итогу был создан boilerplate на express + TS + TypeORM. Тем не менее, нет предела совершенства:

1. Хотелось бы добавить работу с миграциями;
2. Объединить репозитории под один абстрактный класс, чтобы еще больше уменьшить дублирование кода;
3. Добавить линтер.