САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1: Нужно написать свой boilerplate

Выполнил:

Кондратьев Алексей

Группа К33412

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

Чтобы работал проект требуется сделать команду make init, а также написать свой собственный файл. .env

```
#DATABASE
NAME=db
DIALECT=sqlite
USERNAME=root
PASSWORD=null
STORAGE=db.sqlite
DATABASE_DEVELOPMENT=database_development
DATABASE_TEST=database_test
DATABASE_PRODUCTION=database_production

# JWT
ACCESS_TOKEN_LIFETIME=3000000
REFRESH_TOKEN_LIFETIME=36000000

# SERVER
PORT=8000
HOST=127.0.0.1
```

И после всех этих действий с помощью команды make start запустить проект

Были настроены все файлы зависимостей и конфигураций, которые лежат в основной папке директории

О структуре src:

- configs - содержит файлы конфигурации.
- controllers - содержит контроллеры для каждого ресурса.

- core - содержит файлы ядра приложения.
- index.ts - точка входа для запуска приложения.
- middlewares - содержит аутентификацию с использованием Passport.js.
- migrations - содержит файлы миграций для базы данных.
- models - содержит модели для модели пользователей и токенов.
- providers - содержит провайдеры, которые управляют зависимостями
- routes - содержит маршруты для каждого ресурса API
- seeders - содержит файлы начальных данных для базы данных.
- services - содержит службы, которые обрабатывают бизнес-логику приложения.
- utils - содержит утилиты, которые используются во всем приложении

Модели:

RefreshToken.ts – модель хранения токена

```
import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
import User from '../users/User'

@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string

    @ForeignKey(() => User)
    @Column
    userId: number
}

export default RefreshToken
```

User.ts – модель пользователя

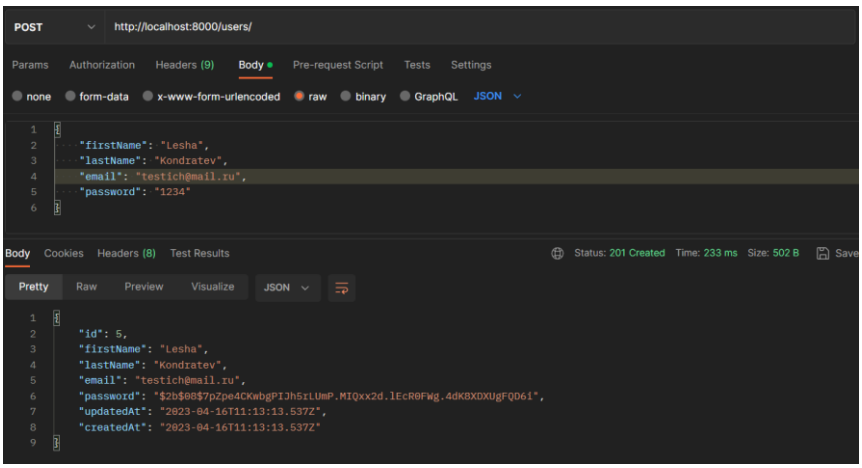
```ts
import { AllowNull, BeforeCreate, BeforeUpdate, Column, Model, Table, Unique } from 'sequelize-typescript'
import hashPassword from '../../utils/hashPassword'

@Table
class User extends Model {
    @AllowNull(false)
    @Column
    firstName: string

    @AllowNull(false)
    @Column
    lastName: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance

        if (instance.changed('password')) {
            instance.password = hashPassword(password)
        }
    }
}

export default User
```

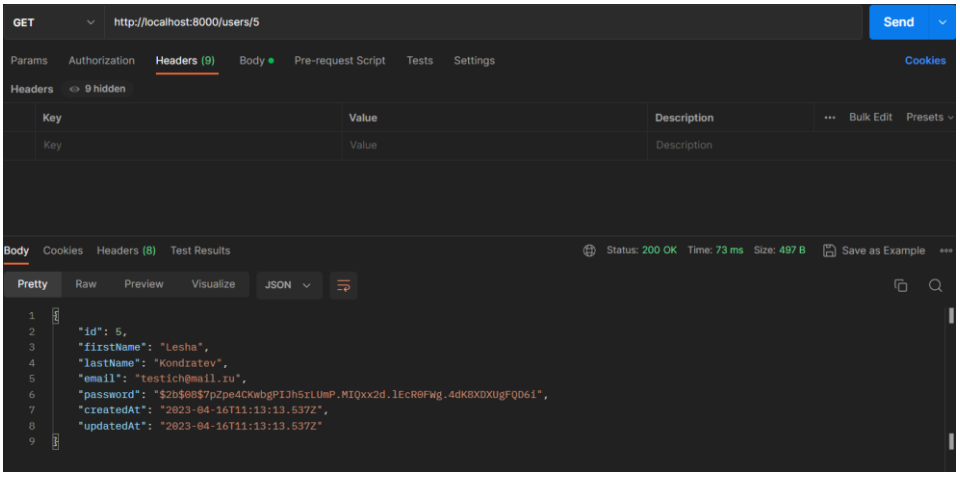
Маршруты:

Маршруты для пользователей /users

```ts
import express from "express"
import UserController from "../../../controllers/users/User"
import passport from "../../../middlewares/passport"

const router: express.Router = express.Router()

const controller: UserController = new UserController()

router.post('/', controller.create)
router.get('/profile', passport.authenticate('jwt', { session: false }), controller.me)
router.get('/:id', controller.get)
router.post('/login', controller.auth)
router.post('/refresh', controller.refreshToken)

export default router
```

Примеры взаимодействия:





Создание и получение пользователя по ID

Вывод

В ходе работы мы создали основную структуру приложения. Такой boilerplate позволяет быстро начать работу используя готовое решение. Разделение на большое количество файлов приводит к хорошей читаемости и поддерживаемости кода. Также познакомился как работать с sequelize/TypeOrm, typescript и отработал навыки работы с express.