Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе № 1 «Создание boilerplate на Express + TypeORM + TypeScript»

по дисциплине «Бэкенд-разработка»

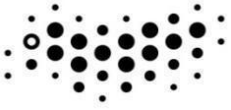
Автор: Власов М. И.

Факультет: ИКТ

Группа: K33402

Преподаватель: Добряков Д. И.

Дата: 05.04.22



УНИВЕРСИТЕТ ИТМО

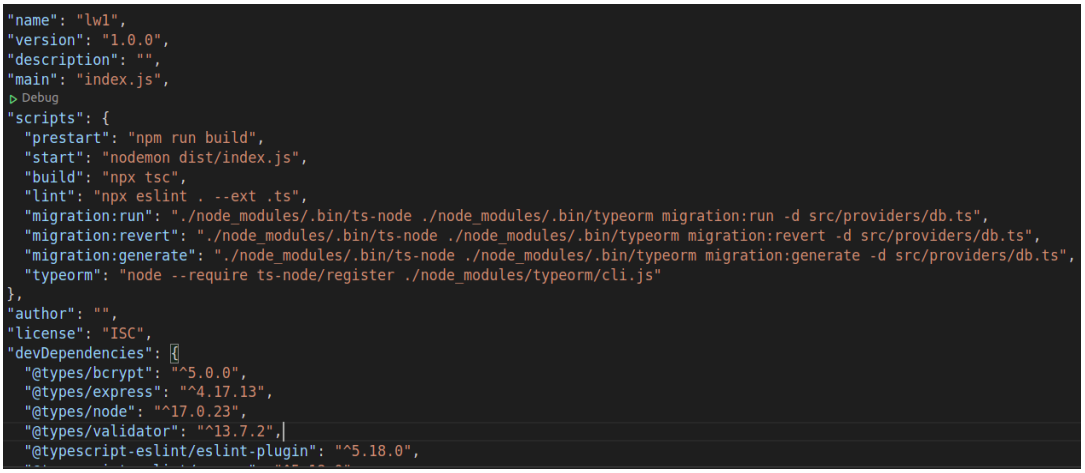
Санкт-Петербург, 2022

Цель: создать boilerplate на Express + TypeORM + TypeScript с явным разделением на:

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

Основной код созданного boilerplate

Файл *package.json*, где находятся скрипты и зависимости



```
"name": "lw1",
"version": "1.0.0",
"description": "",
"main": "index.js",
▷ Debug
"scripts": {
  "prestart": "npm run build",
  "start": "nodemon dist/index.js",
  "build": "npx tsc",
  "lint": "npx eslint . --ext .ts",
  "migration:run": "./node_modules/.bin/ts-node ./node_modules/.bin/typeorm migration:run -d src/providers/db.ts",
  "migration:revert": "./node_modules/.bin/ts-node ./node_modules/.bin/typeorm migration:revert -d src/providers/db.ts",
  "migration:generate": "./node_modules/.bin/ts-node ./node_modules/.bin/typeorm migration:generate -d src/providers/db.ts",
  "typeorm": "node --require ts-node/register ./node_modules/typeorm/cli.js"
},
"author": "",
"license": "ISC",
"devDependencies": {
  "@types/bcrypt": "^5.0.0",
  "@types/express": "^4.17.13",
  "@types/node": "^17.0.23",
  "@types/validator": "^13.7.2",
  "@typescript-eslint/eslint-plugin": "^5.18.0",
```

TypeORM datasource

```
const dataSource = new DataSource({
    type: "postgres",
    host: "localhost",
    port: 5432,
    username: "test",
    password: "test",
    database: "test",
    synchronize: true,
    logging: true,
    entities: ['dist/models/**/*.js'],
    subscribers: [],
    migrations: ['dist/migrations/**/*.js'],
})

dataSource
    .initialize()
    .then(() => {
        console.log("Data Source has been initialized successfully")
    })
    .catch((err) => {
        console.error("Error during Data Source initialization:", err)
    })
```

Ядро приложения

```
class App {
  public port: number
  public host: string

  private app: express.Application
  private server: Server

  constructor(port = 8080, host = "localhost") {
      this.port = port
      this.host = host

      this.app = this.createApp()
      this.server = this.createServer()
  }

  private createApp(): express.Application {
      const app = express()
      const bodyParser = require('body-parser')

      app.use(bodyParser.urlencoded({ extended: false }))
      app.use(bodyParser.json())
      app.use('/', routes)

      return app
    }

  private createServer(): Server {
      const server = createServer(this.app)

      return server
  }

  public start(): void {
      this.server.listen(this.port, () => {
          console.log(`Running server on port ${this.port}`)
          const all_routes = require('express-list-endpoints')
          console.log(all_routes(this.app))
      })
  }
}
```

Модель *User*

```
@Entity({ name: "users" })
export class User {
    @PrimaryGeneratedColumn()
    id!: number

    @Column()
    firstName!: string

    @Column()
    lastName!: string

    @Column({ unique: true })
    email!: string

    @Column()
    password!: string

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

        instance.password = hashPassword(password)
    }
}

export default User
```

Соответствующий контроллер

```
class UserController {
    private userService: UserService

    constructor() {
        this.userService = new UserService()
    }

    create = async (request: any, response: any) => {
        try {
            const { body } = request
            body.password = hashPassword(body.password)

            const user: User | Error = await this.userService.create(body)

            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }

    retrieve = async (request: any, response: any) => {
        try {
            const user: User | Error = await this.userService.getById(Number(request.params.id))

            response.send(user)
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }

    login = async (request: any, response: any) => {
        const { body } = request

        const { email, password } = body

        try {
            const { user, checkPassword } = await this.userService.checkPassword(email, password)
            if (checkPassword) {
                response.send(`You successfully logged in as ${user.firstName} ${user.lastName}`)
            } else {
                response.send("Your email/password is incorrect!")
            }
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }
```

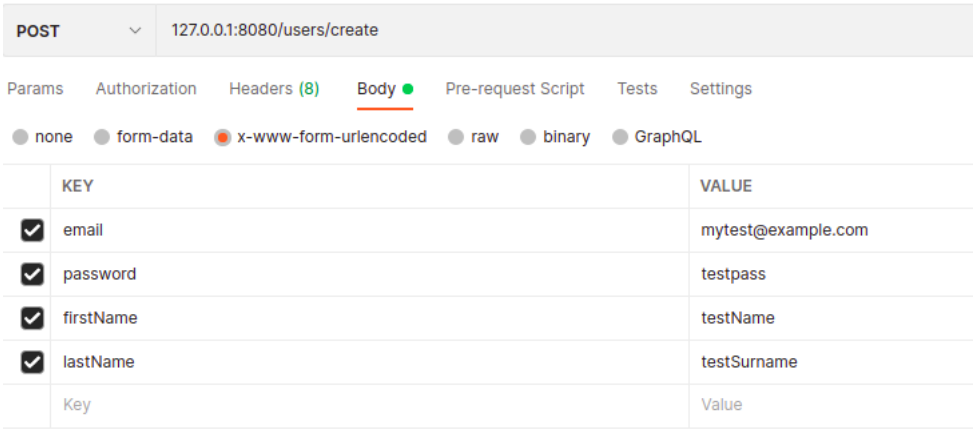
Routers

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

const controller: UserController = new UserController()

router.route("/create").post(controller.create)

router.route("/:id").get(controller.retrieve)

router.route("/login").post(controller.login)
```

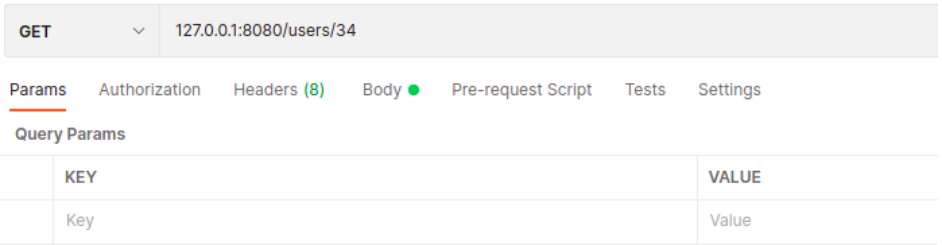
Сервисы

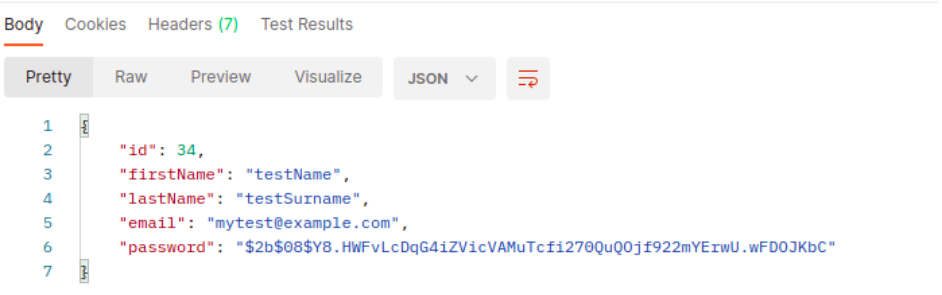
```
class UserService {
    async getById(id: number) : Promise<User> {
        const user = await dataSource.getRepository(User).findOneBy({ 'id': id })

        if (user) return user

        throw new Error(`User with id ${id} not found`)
    }

    async getByEmail(email: string) : Promise<User> {
        const user = await dataSource.getRepository(User).findOneBy({ 'email': email })

        if (user) return user

        throw new Error(`User with email ${email} not found`)
    }

    async create(userData: object) : Promise<User> {
        try {
            const user = await dataSource.getRepository(User).save(userData)
            return user
        } catch (e: any) {
            throw new Error(e)
        }
    }

    async checkPassword(email: string, password: string) : Promise<any> {
        const user = await dataSource.getRepository(User).findOneBy({ 'email': email })

        if (user) return { user: user, checkPassword: checkPassword(user, password) }

        throw new Error(`User with email ${email} not found`)
    }
}
```

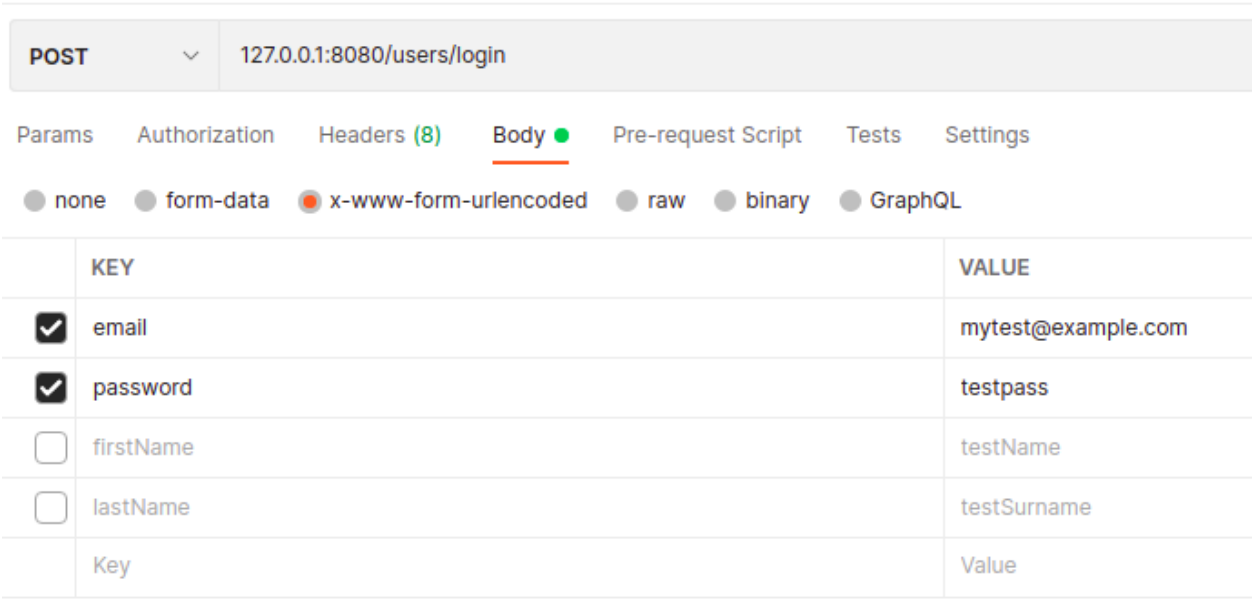
Примеры запросов

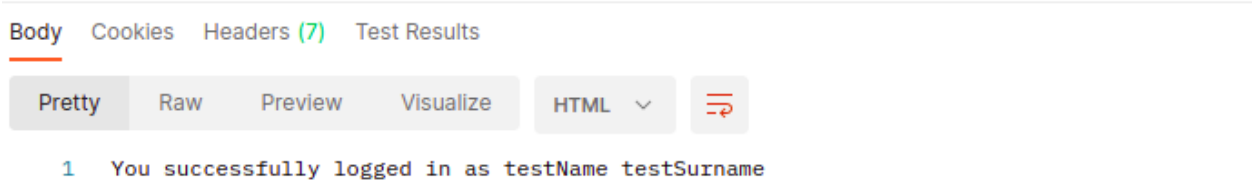


127.0.0.1:8080/users/34









Вывод: в ходе лабораторной работы мы создали boilerplate на Express, TypeORM и TypeScript, который включает в себя такие элементы, как модели, контроллеры, роуты и сервисы для работы с моделями.