

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №1

“Написание boilerplate для Express.js”

Выполнил:

Кондрашов Е. Ю.

Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2022 г.

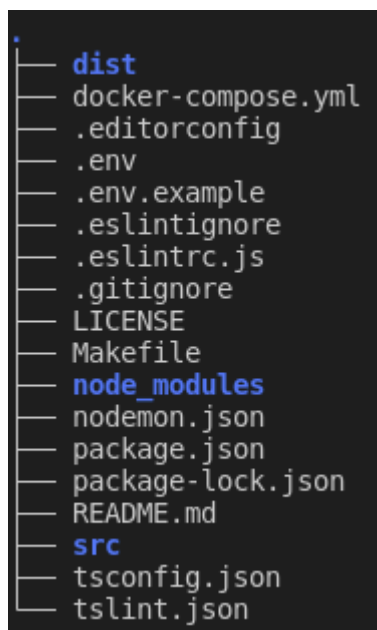
Задача

1. Нужно написать свой boilerplate на express + sequelize + typescript.
2. Должно быть явное разделение на: модели, контроллеры, роуты, сервисы для работы с моделями (реализуем паттерн “репозиторий”).

Ход работы

В качестве ORM я решил использовать TypeORM. За основу для шаблона я взял <https://github.com/kantegory/express-sequelize-boilerplate> и <https://github.com/mkosir/express-typescript-typeorm-boilerplate>.

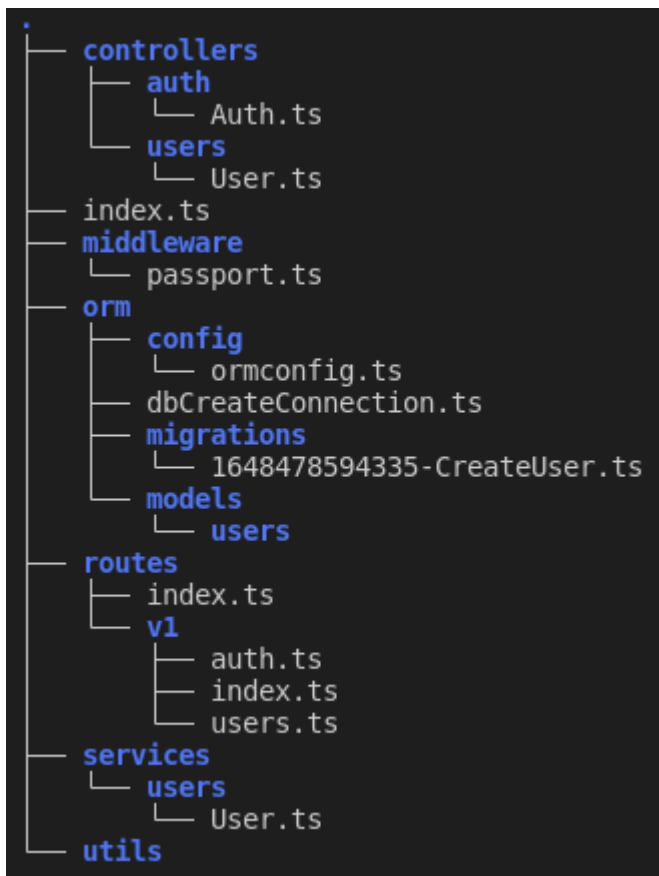
Корневая папка шаблона:



```
.
├── dist
├── docker-compose.yml
├── .editorconfig
├── .env
├── .env.example
├── .eslintignore
├── .eslintrc.js
├── .gitignore
├── LICENSE
├── Makefile
├── node_modules
├── nodemon.json
├── package.json
├── package-lock.json
├── README.md
├── src
├── tsconfig.json
└── tslint.json
```

В ней лежат конфигурационные файлы (для линтеров, для пакета npm, для typescript), мета-файлы (readme и license), а также docker-compose для локального запуска БД.

Рассмотрим папку src:



В папке controllers располагаются контроллеры, которые получают запрос и возвращают ответы. Реализованы как методы в классе:

```
import UserService from '../services/users/User'

class UserController {
  private userService: UserService

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

  retrieve = async (request: any, response: any) => {
    try {
      const user = await this.userService.getById(
```

```

        Number(request.params.id)

    )

    response.send(user)

} catch (error: any) {

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

}

}

me = async (request: any, response: any) => {

    response.send(request.user)

}

}

export default UserController

```

В файле `index.ts` располагается общая конфигурация express-приложения.

В папке `middleware` располагаются файлы, относящиеся к связующему ПО, в данном случае настройки пакета `passport.js`.

В папке `ORM` располагаются файлы, связанные с ORM: конфигурация, миграции и модели.

```

import { Column, CreateDateColumn, Entity,
PrimaryGeneratedColumn, UpdateDateColumn } from 'typeorm';

@Entity('users')

export class User {

    @PrimaryGeneratedColumn()

```

```

    id: number;

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

    @Column()
    password: string;

    @Column()
    @CreateDateColumn()
    created_at: Date;

    @Column()
    @UpdateDateColumn()
    updated_at: Date;
}

export default User

```

В папке routes располагаются файлы маршрутизации, которые сопоставляют путь и соответствующий контроллер.

```

import { Router } from 'express';

import UserController from "../../controllers/users/User";

import passport from "../../middleware/passport";

```

```

const router = Router();

const controller = new UserController()

router.route('/me')
    .get(passport.authenticate('jwt', { session: false }),
controller.me)

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

export default router

```

В папке services располагается описание классов, которые реализуют бизнес-логику взаимодействия с сущностями приложения:

```

import bcrypt from 'bcryptjs';

import { getRepository } from 'typeorm';

import User from '../../orm/models/users/User';

class UserService {

    async getById(id: number) : Promise<User> {

        const user = await getRepository(User).findOne(id);

        if (user) return user;

        throw new Error('User with specified ID is not found');

    }

    async getByEmail(email: string) : Promise<User> {

```

```

        const user = await getRepository(User).findOne({ where: {
email } });

        if (user) return user;

        throw new Error('User with specified email is not found');
    }

    async create(userData: {email: string, password: string}) :
Promise<User> {
        try {
            userData.password = bcrypt.hashSync(userData.password,
8)

            const user = await getRepository(User).save(userData)

            return user
        } catch (e: any) {
            throw new Error(e)
        }
    }

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

        if (!user) {
            throw new Error('Does not exist')
        }

        const passwordMatch = bcrypt.compareSync(password,
user.password);

```

```
        return { user: user, passwordMatch: passwordMatch }  
  
    }  
}  
  
export default UserService
```

В папке `utils` лежат различные вспомогательные классы и функции, не относящиеся к остальным папкам.

Вывод

В ходе работы, используя несколько готовых решений, я собрал шаблон репозитория `Express.js + TypeScript + TypeORM`, готовый к созданию полноценного бэкенд-приложения. Понятная организация файлов в репозитории ускоряет скорость разработки, а также упрощает понимание создаваемой архитектуры приложения другими разработчиками.