**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бек-энд разработка

Отчет

# Лабораторная работа “Typescript: Основы языка”

Выполнила:
Соколовская Арина

К333392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

**Задание**

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

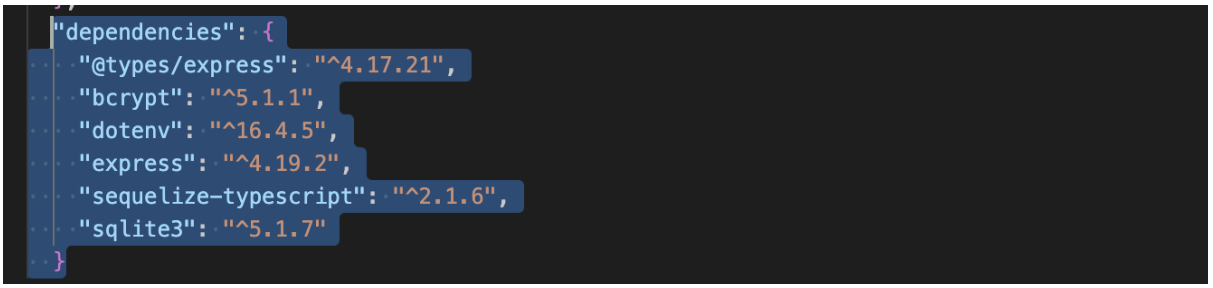
Должно быть явное разделение на:

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

**Ход работы**

1. **Инициализация проекта, установка зависимостей**

Для начала я проинициализировала проект и добавила следующие зависимости:




2. **Настройка Typescript**

Я использовала команды

`npm i -D typescript` – установка компилятора Typescript

`./node_modules/.bin/tsc --init` – конфигурирование Typescript

После этого в директории появился автоматически сгенерированный файл tsconfig.json, с помощью которого можно сконфигурировать TS для себя.

3. **Создание и конфигурация .env файла**

В .env файле находятся следующие переменные:

```
#service
port="8080"

#database
database="database"
username="admin"
password="password"
storage="db.sqlite"
```

Для использования .env файла требуется импортировать dotenv/config:

```
import express from "express"
import { createServer, Server } from "http"
import routes from "../routes/user.route"
import 'dotenv/config'  7.5k (gzipped: 3.2k)
import sequelize from "../db/db";
```

4. **Создание БД**

```
const sequelize = new Sequelize({
    database: process.env.database,
    dialect: "sqlite",
    host: process.env.host,
    username: process.env.username,
    password: process.env.password,
    storage: process.env.storage,
    logging: console.log
});
```

## 5. Создание модели пользователя

```
export @Table
class User extends Model {

   @PrimaryKey
   @AutoIncrement
   @Column
    id: number;

    @Unique
    @Column
    nickname: string

    @Column
    name: string

    @Unique
    @Column
    email: string


    @Unique
    @Column
    phone: string

    @Column
    password: string;
```

Для шифрования пароля я использовала bcrypt:

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

    if (instance.changed('password')) {
        instance.password = bcrypt.hashSync(password, bcrypt.genSaltSync(5));
    }
}
```

## 6. Добавление и синхронизация модели

```
const models = [User];

sequelize.addModels(models);
```

```
sequelize
  .sync()
  .then(() => {
    //something here
    console.log('Models synced successfully');
  })
  .catch((e) => console.log(e));

async function testConnection() {
    try {
        await sequelize.authenticate();
        console.log('Connection has been established successfully.');
    } catch (error) {
        console.error('Unable to connect to the database:', error);
    }
}

testConnection();

export default sequelize;
```

## 7. Создание репозитория

```
export class UserService {
    async create(creationData: any): Promise<User> {
      try {
        return (await User.create(creationData)).toJSON();
      } catch (e) {
        console.error(creationData);
        throw e;
      }
    }

    async get(id: number): Promise<User> {
      try {
        const user = await User.findByPk(id);
        return user;
      } catch {
        throw new UserNotFound(
          `User with id=${id} not found}`
        )
      }
    }
  }
```

## 8. Создание контроллера

```
class UserController {
  service: UserService

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

  get = async (req: Request, res: Response) => {
    try {
      const id = Number(req.params.id);
      console.log(id);
      res.send((await this.service.get(id)).toJSON())
    } catch {
      res
        .status(404)
        .send({ error: 'User with the specified id not found' })
    }
  }
  post = async (req: Request, res: Response) => {
    try {
      console.log(req.body)
      res.send(this.service.create(req.body as User))
    } catch {
      res.status(400).send({ error: 'Invalid data' })
    }
  }
}

export default UserController
```

## 9. Создание роутера

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

const userController = new UserController()

router
  .route('/user/:id')
  .get(userController.get)
router
  .route('/user')
  .post(userController.post)
export default router
```

## 10. Создание приложения

```
const app = express()
app.use(express.json());
app.use('/api', router);

app.listen(process.env.port, () => {
  sequelize
  console.log(`Running on port ${process.env.port}`)
})
```

## 11. Проверка работоспособности