**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

# **Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Беломытцев Андрей

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

**Задача**

Нужно написать свой boilerplate на express + TypeORM + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты

**Ход работы**

Разделение на модели, контроллеры и роуты, а также некоторые другие части написания boilerplate были заранее реализованы ещё при работе над ДЗ2.

Для использования переменных среды были созданы:

Файл config.ts получающий переменные среды с помощью dotenv

```
import dotenv from 'dotenv';
dotenv.config();

class Config {
  APP_PORT: number = parseInt(process.env.APP_PORT || '3000');

  DB_HOST = process.env.DB_HOST || 'localhost';
  DB_PORT = parseInt(process.env.DB_PORT || '5432');
  DB_NAME = process.env.DB_NAME || 'db';
  DB_USER = process.env.DB_USER || 'user';
  DB_PASSWORD = process.env.DB_PASSWORD || 'qwerty';

  JWT_SECRET_KEY = process.env.JWT_SECRET_KEY || 'secret';
  JWT_TOKEN_TYPE = process.env.JWT_SECRET_KEY || 'Bearer';
  JWT_ACCESS_TOKEN_LIFETIME: number =
parseInt(process.env.JWT_ACCESS_TOKEN_LIFETIME || '300');
}

const config = new Config();

export default config;
```

Файл .env

```
APP_PORT=3000

DB_HOST=localhost
DB_PORT=5432
DB_NAME=db
DB_USER=user
DB_PASSWORD=qwerty
```

Были отредактированы некоторые части кода, чтобы использовать переменные среды. Например, изменения в файле AppDataSource.ts отвечающем за подклчение БД:

```
import config from './config';

export const AppDataSource = new DataSource({
  type: 'postgres',
  host: config.DB_HOST,
  port: config.DB_PORT,
  username: config.DB_USER,
  password: config.DB_PASSWORD,
  database: config.DB_NAME,
  synchronize: true,
  logging: true,
  entities: [
    Category,
    Channel,
    Review,
    Role,
    Theme,
    User,
    Video,
  ],
  migrations: [],
  subscribers: [],
})
```

Затем надо было реализовать JWT авторизацию для этого установил jsonwebtoken и bcryptjs.

```
npm i jsonwebtoken
npm i bcryptjs
npm i --save-dev @types/jsonwebtoken @types/bcryptjs
```

В файл controllers/User.ts добавлены строчки

```
import bcryptjs from 'bcryptjs'
import signJWT from '../utils/signJWT'

const validateToken = async (req: Request, res: Response) => {
  res.json({ message: 'Authorized' })
}

const register = async (req: Request, res: Response) => {
  try {
    let { password } = req.body
    bcryptjs.hash(password, 10, async (err, hash) => {
      if (err){
        res.status(500).json({ err: err.message })
      }
      else {
        try {
          req.body.password = hash
          res.json(await repository.save(req.body))
        } catch (err: any) {
          res.status(500).json({ error: err.message })
```

```
        }
      }
    })
  }
  catch (error: any) {
    res.status(500).json({ error: error.message })
  }
}

const login = async (req: Request, res: Response) => {
  try {
    let { username, password } = req.body
    const user = await repository.findOne({ where: { username: username } })
    if (user){
      bcryptjs.compare(password, user?.password, (err, r) => {
        if (err){
          res.status(401).json ({ error: err.message })
          return
        }
        else if (r){
          signJWT(user, (error: any, token: any) => {
            if (error){
              res.status(401).json ({ error: error.message })
              return
            }
            else if (token){
              res.status(200).json({ message: 'Auth Successfull', token,
user: username })
              return
            }
          })
        }
      })
    }
    else{
      res.status(401).json ({ error: 'No user' })
    }
  }
  catch (error: any) {
    res.status(500).json({ error: error.message })
  }
}

export default { create, get, getOne, update, remove, getByEmail,
validateToken, register, login }
```

## В файл routers/User.ts добавлены строчки

```
import extractJWT from '../middleware/extractJWT'

router.get('/validate', extractJWT, controller.validateToken)
router.post('/register', controller.register)
router.post('/login', controller.login)
```

## Создан файл middleware/extractJWT.ts

```
import { Request, Response, NextFunction } from 'express'
import config from '../config';
import jwt from 'jsonwebtoken'
```

```
const extractJWT = async (req: Request, res: Response, next: NextFunction) =>
{
  let token = req.headers.authorization?.split(' ')[1]
  if (token){
    jwt.verify(token, config.JWT_SECRET_KEY, (err, decoded) => {
      if (err) {
        res.status(404).json({ message: err })
        return
      }
      else {
        res.locals.jwt = decoded
        next()
        return
      }
    })
  }
  else {
    res.status(401).json({ message: 'Unauthorized' })
  }
}

export default extractJWT
```

Создан файл utils/signJWT.ts

```
import config from '../config';
import jwt from 'jsonwebtoken'

type UserType = {
  username: string,
  password: string
}

const signJWT = (user: UserType, callback: any) => {
  try {
    jwt.sign(
      { username: user.username },
      config.JWT_SECRET_KEY,
      { algorithm: 'HS256', expiresIn: config.JWT_ACCESS_TOKEN_LIFETIME },
      (err, token) => {
        if (err) {
          callback(err, null)
        }
        else if (token) {
          callback(null, token)
        }
      }
    )
  } catch (err: any) {
    callback(err, null)
  }
}

export default signJWT
```

Работа JWT проверена с использованием расширения REST Client для VS Code.

**Вывод**

В результате написан свой boilerplate на express, TypeORM и typescript. Реализовано явное разделение на модели, контроллеры, роутеры. Реализована авторизация с помощью JWT. Использованы переменные среды.