

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнил:

Селезнев Илья

Группа К33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

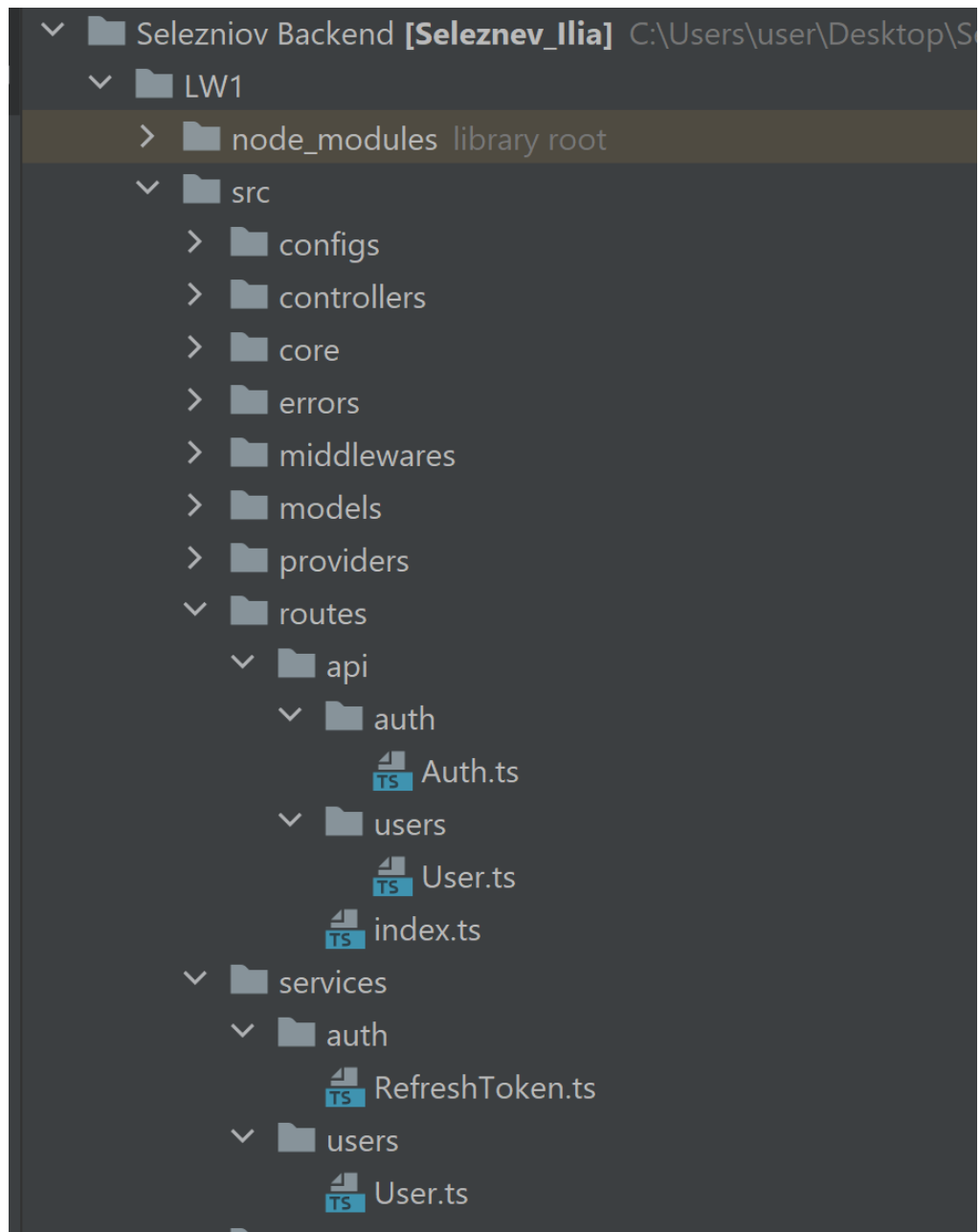
Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

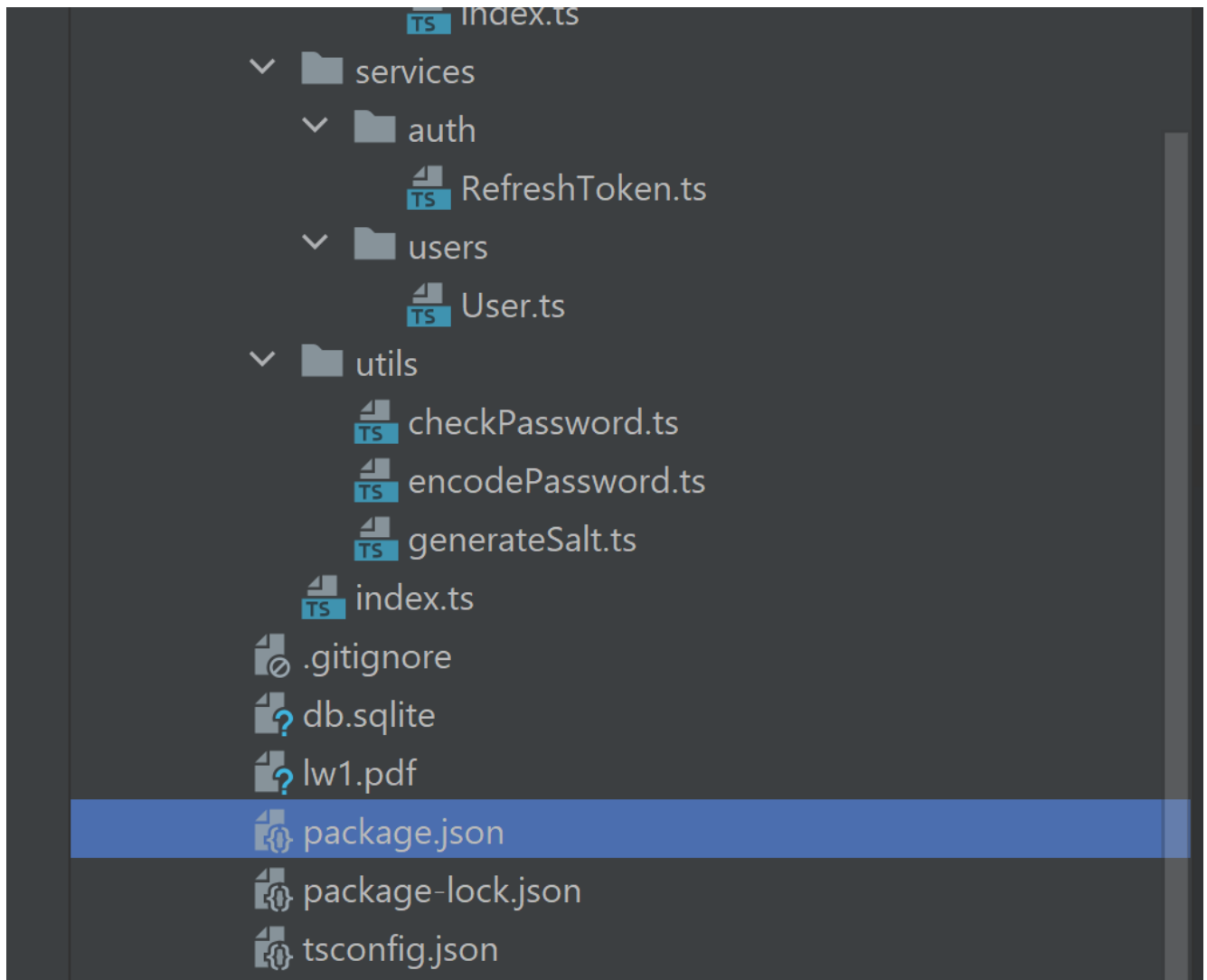
Должно быть явное разделение на:

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

Ход работы

Структура проекта:





Реализованы 2 модели: пользователь и refresh token и 5 эндпоинтов: логин, регистрация, создание пользователя, получение текущего пользователя, изменение текущего пользователя. Последние 2 эндпоинта требуют авторизации при помощи JWT.

Настройки проекта сохранены в файле .ini:

```
settings.ini x
1 [DATABASE]
2   name = "my_db"
3   dialect = "sqlite"
4   username = "root"
5   password = ""
6   storage = "db.sqlite"
7   host = ""
8   port = ""
9 [JWT]
10  accessTokenLifetime = 900000
11  refreshTokenLifetime = 3600000
12 [SERVER]
13  port = 5000
14  host = "localhost"
```

Файл настроек парсится следующим кодом:

```
index.ts x
1 const fs = require('fs')
2 const ini = require('ini')
3
4 const configFile = fs.readFileSync( path: 'src/configs/settings.ini', options: 'utf-8')
5 const dbConfig = ini.parse(configFile)['DATABASE']
6 const serverConfig = ini.parse(configFile)['SERVER']
7 const jwtConfig = ini.parse(configFile)['JWT']
8
9 export {
10   dbConfig,
11   serverConfig,
12   jwtConfig
13 }
```

Вспомогательные функции для авторизации (генерация соли, хэширование пароля, проверка пароля):

```
generateSalt.ts x
1 import {randomBytes} from 'crypto';
2
3 export default (): string => {
4   return randomBytes( size: 16).toString( encoding: 'base64');
5 }
```

```
encodePassword.ts ×
1 | import {createHash} from 'crypto';
2 |
3 | export default (password: string, salt: string): string => {
4 |     return createHash( algorithm: 'sha256').update( data: password + salt).digest( encoding: 'base64')
5 | }
```

```
checkPassword.ts ×
1 | import User from '../models/users/User'
2 | import encodePassword from './encodePassword'
3 |
4 | export default (user: User, password: string): boolean => {
5 |     return user.password === encodePassword(password, user.salt)
6 | }
```

Реализация модели пользователя с генерацией соли и хэшированием пароля:

```

1  import ...
4
5  @DefaultScope( scopeGetter: () => ({
6    attributes: ['id', 'firstName', 'lastName', 'email']
7  }))
8  @Table
9  class User extends Model {
10    @AllowNull( allowNull: false)
11    @Column
12    firstName: string
13
14    @AllowNull( allowNull: false)
15    @Column
16    lastName: string
17
18    @AllowNull( allowNull: false)
19    @Unique
20    @Column
21    email: string
22
23    @AllowNull( allowNull: false)
24    @Column
25    password: string
26
27    @Column
28    salt: string
29
30    @BeforeCreate
31    static setPasswordInitial(instance: User) {
32      instance.salt = generateSalt()
33      const {password} = instance
34      instance.password = encodePassword(password, instance.salt)
35      console.log(instance.salt)
36      console.log(instance.password)
37    }
38
39    @BeforeUpdate
40    static setPasswordUpdate(instance: User) {
41      const {password} = instance
42      if (instance.changed( key: 'password')) {
43        instance.password = encodePassword(password, instance.salt)
44      }
45    }
46  }
47
48  export default User

```

Модель refresh токена:

```
RefreshToken.ts
1 import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
2 import User from '../users/User'
3
4 @Table
5 class RefreshToken extends Model {
6   @Unique
7   @AllowNull( allowNull: false)
8   @Column
9   token: string
10
11   @ForeignKey( relatedClassGetter: () => User)
12   @Column
13   userId: number
14 }
15
16 export default RefreshToken
```

Сервис для пользователя:


```

1  import User from '../../../models/users/User'
2  import APIError from '../../../errors/APIError'
3  import checkPassword from '../../../utils/checkPassword'
4
5
6  class UserService {
7    async getById(id: number): Promise<User | APIError> {
8      const user = await User.findByPk(id)
9      if (user) {
10        return user.toJSON()
11      }
12      throw new APIError('User not found')
13    }
14
15    async create(userData: any): Promise<User | APIError> {
16      try {
17        const user = await User.create(userData)
18        return user.toJSON()
19      } catch (e: any) {
20        const errors = e.errors.map((error: any) => error.message)
21        throw new APIError(errors)
22      }
23    }
24
25    async update(id: number, userData: any): Promise<User | APIError> {
26      let user = await User.findByPk(id)
27      if (user) {
28        try {
29          user = await user.update(userData)
30          return user.toJSON()
31        } catch (e: any) {
32          const errors = e.errors.map((error: any) => error.message)
33          throw new APIError(errors)
34        }
35      }
36      throw new APIError('User not found')
37    }
38
39    async checkPassword(email: string, password: string): Promise<any> {
40      const user = await User.findOne( options: {where: {email}})
41      if (user) {
42        return {user: user.toJSON(), checkPassword: checkPassword(user, password)}
43      }
44      throw new APIError('Incorrect credentials')
45    }
46  }
47
48  export default UserService

```

Сервис для refresh токена:

```

7  class RefreshTokenService {
8      private user: User | null
9
10     constructor(user: User | null = null) {
11         this.user = user
12     }
13
14     generateRefreshToken = async (): Promise<string> => {
15         const token = randomUUID()
16         const userId = this.user?.id
17         await RefreshToken.create({ values: {token, userId}})
18         return token
19     }
20
21     isRefreshTokenExpired = async (token: string): Promise<{ userId: number | null, isExpired: boolean }> => {
22         const refreshToken = await RefreshToken.findOne({ options: {where: {token}}})
23         if (refreshToken) {
24             const tokenData = refreshToken.toJSON()
25             const currentDate = new Date()
26             const timeDelta = currentDate.getTime() - tokenData.createdAt.getTime()
27             if (timeDelta > 0 && timeDelta < jwtConfig.refreshTokenLifetime) {
28                 return {userId: tokenData.userId, isExpired: false}
29             }
30             return {userId: null, isExpired: true}
31         }
32         return {userId: null, isExpired: true}
33     }
34 }
35
36 export default RefreshTokenService

```

Контроллер для пользователей и авторизации:

```

8  class UserController {
9      private userService: UserService
10
11     constructor() {
12         this.userService = new UserService()
13     }
14
15     // get = async (request: any, response: any) => {
16     //     try {
17     //         const user: User | APIError = await this.userService.getById(
18     //             Number(request.params.id)
19     //         )
20     //         response.send(user)
21     //     } catch (error: any) {
22     //         response.status(404).send({'detail': error.message})
23     //     }
24     // }
25

```

```

26 post = async (request: any, response: any) => {
27     const {body} = request
28     try {
29         const user: User | APIError = await this.userService.create(body)
30         response.status(201).send(user)
31     } catch (error: any) {
32         response.status(400).send({'detail': error.message})
33     }
34 }
35
36 // Get only allowed for myself
37 get = async (request: any, response: any) => {
38     const {user} = request
39     if (user) {
40         response.send(user)
41     } else {
42         response.status(401).send({'detail': 'Not authenticated'})
43     }
44 }
45

```

```

46 // Put only allowed for myself
47 put = async (request: any, response: any) => {
48     const {body, user} = request
49     if (user) {
50         try {
51             const updatedUser: User | APIError = await this.userService.update(user.id, body)
52             response.status(200).send(updatedUser)
53         } catch (error: any) {
54             response.status(400).send({'detail': error.message})
55         }
56     } else {
57         response.status(401).send({'detail': 'Not authenticated'})
58     }
59 }
60

```

```

61 login = async (request: any, response: any) => {
62     const {body} = request
63     const {email, password} = body
64     try {
65         const {user, checkPassword} = await this.userService.checkPassword(email, password)
66
67         if (checkPassword) {
68             const payload = {id: user.id}
69             const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
70             const refreshTokenService = new RefreshTokenService(user)
71             const refreshToken = await refreshTokenService.generateRefreshToken()
72
73             response.send({accessToken, refreshToken})
74         } else {
75             response.status(400).send({'detail': 'Invalid credentials'})
76         }
77     } catch (e: any) {
78         response.status(400).send({'detail': e.message})
79     }
80 }
81

```

```

82 refreshToken = async (request: any, response: any) => {
83     const {body} = request
84     const {refreshToken} = body
85     const refreshTokenService = new RefreshTokenService()
86     try {
87         const {userId, isExpired} = await refreshTokenService
88             .isRefreshTokenExpired(refreshToken)
89
90         if (!isExpired && userId) {
91             try {
92                 const user = await this.userService.getById(userId)
93                 // @ts-ignore
94                 const payload = {id: user.id}
95                 const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
96                 // @ts-ignore
97                 const refreshTokenService = new RefreshTokenService(user)
98                 const refreshToken = await refreshTokenService.generateRefreshToken()
99                 response.send({accessToken, refreshToken})
100             } catch (e: any) {
101                 response.status(400).send({'detail': e.message})
102             }

```

```

103         } else {
104             response.status(401).send({'error': 'Invalid credentials'})
105         }
106     } catch (e: any) {
107         response.status(401).send({'error': e.message})
108     }
109 }
110 }
111
112 export default UserController

```

Работа эндпоинтов:

HTTP <http://127.0.0.1:5000/api/users/>

POST <http://127.0.0.1:5000/api/users/>

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL [JSON](#) [v](#)

```
1 {
2   ... "firstName": "ILIA",
3   ... "lastName": "Seleznirov",
4   ... "email": "example@test.com",
5   ... "password": "a123456"
6 }
```

Body Cookies Headers (8) Test Results [Status:](#)

[Pretty](#) [Raw](#) [Preview](#) [Visualize](#) [JSON](#) [v](#) [↕](#)

```
1 {
2   "id": 2,
3   "firstName": "ILIA",
4   "lastName": "Seleznirov",
5   "email": "example@test.com"
6 }
```

POST <http://127.0.0.1:5000/api/auth/login> [Send](#) [v](#)

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings [Cookies](#) [Beautify](#)

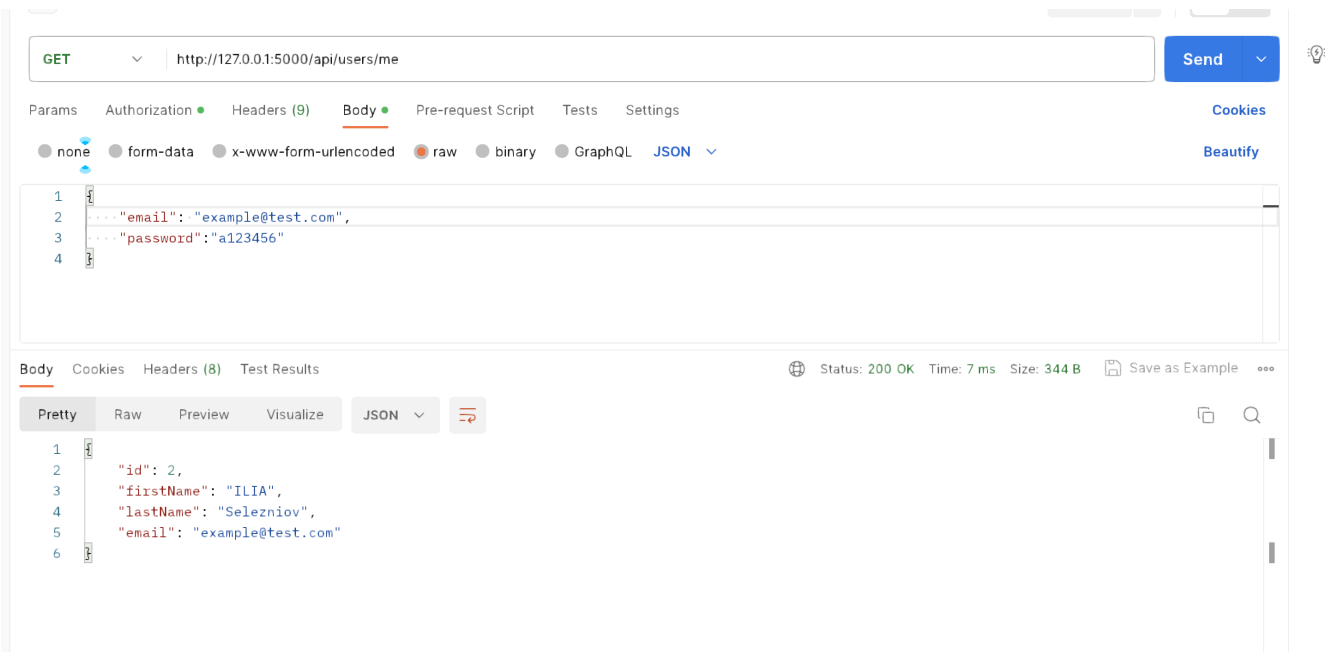
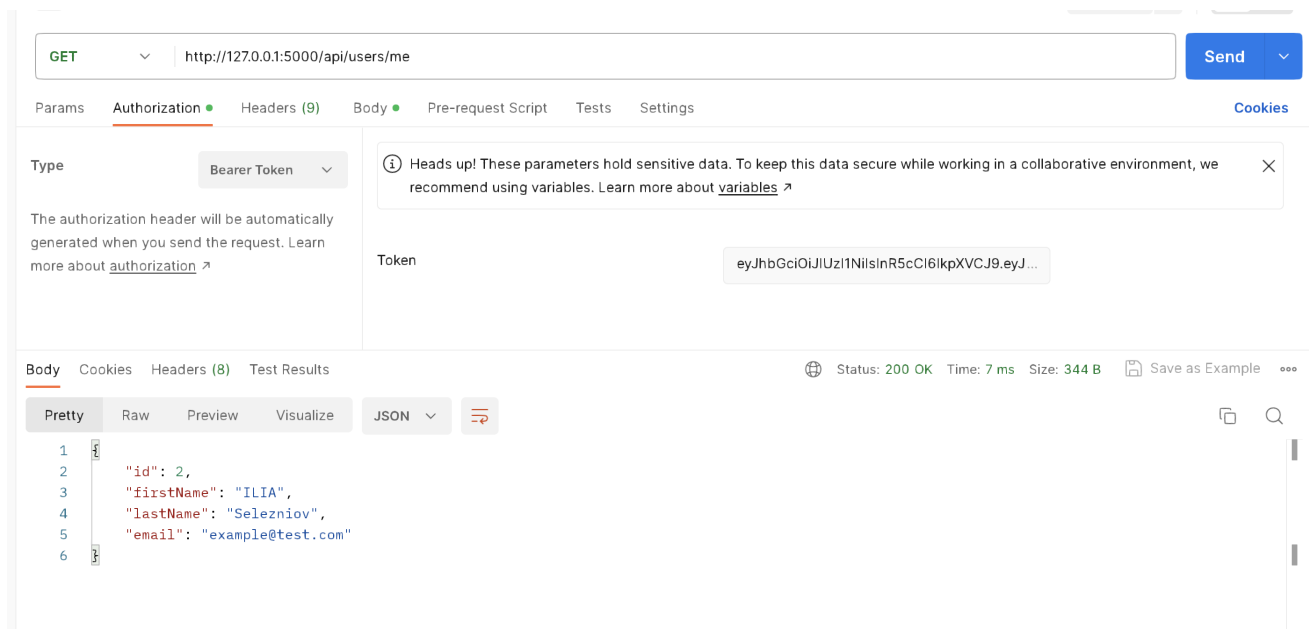
☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL [JSON](#) [v](#)

```
1 {
2   ... "email": "example@test.com",
3   ... "password": "a123456"
4 }
```

Body Cookies Headers (8) Test Results [Status: 200 OK](#) [Time: 13 ms](#) [Size: 455 B](#) [Save as Example](#) [...](#)

[Pretty](#) [Raw](#) [Preview](#) [Visualize](#) [JSON](#) [v](#) [↕](#) [📄](#) [🔍](#)

```
1 {
2   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNjg2NTIwMTI0fQ.tHt8r49BcvYVblgoinAi9rI5mG5-NzAbfaVftiAjC24",
3   "refreshToken": "845fb9a1-674d-4e54-aa4c-b246af4e1306"
4 }
```



Вывод

Я создал boilerplate проекта с реализованным функционалом пользователя и авторизации