

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

ЛР 1 - реализация boilerplate

Выполнил:

Сергеев Виктор

К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

## Задача

Требуется реализовать boilerplate на Express + TypeORM + typescript

Должно быть разделение на:

- модели
- контроллеры
- роуты

## Ход работы

Данный проект будет реализован по архитектуре MVCS. Текущая файловая структура проекта изображена на рисунке 1

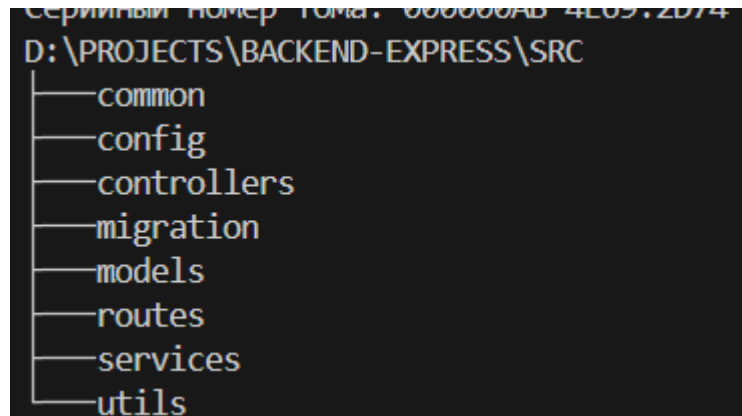


Рисунок 1 - файловая структура проекта

Помимо этого, в директории src находятся файлы index.ts - содержит инициализацию приложения - и data-source.ts - содержит инициализацию настроек базы данных. Назначение каждой директории следующее:

- common - содержит дженерик классы частей проекта для их успешного переиспользования
- config - содержит конфигурационные параметры проекта
- controllers - содержит контроллеры моделей
- migration - содержит файлы миграции базы данных
- models - содержит описания моделей
- routes - содержит роутеры для каждой модели
- services - содержит сервисы моделей
- utils - вспомогательные классы и функции

В этой структуре сервис отвечает за обращения к бд, контроллер для принятия данных, их валидации и взаимодействие с сервисом.

Для сервисов были реализованы дженерик классы BaseService и BaseCRUDService, от которых наследуются все дальнейшие сервисы. Это удобно, поскольку для большинства моделей надо сделать простой набор из CRUD-методов, и благодаря наследованию можно просто создать дочерний класс с нужной моделью, чтобы получить уже готовую реализацию. На рисунках 2 и 3 соответственно представлен их код.

```
src > common > TS BaseService.ts > BaseService
1  import { Repository } from "typeorm";
2  import { AppDataSource } from "../data-source";
3
4  export class BaseService<T> {
5      protected repository: Repository<T>;
6      protected model: new () => T;
7
8      constructor(model: new () => T) {
9          this.model = model;
10         this.repository = AppDataSource.getRepository(model);
11     }
12 }
```

Рисунок 2 - файл BaseService.ts

```

src > common > TS BaseCRUDService.ts > BaseCRUDService
1  import { DeepPartial } from "typeorm";
2  import { BaseService } from "../BaseService";
3
4  export class BaseCRUDService<T> extends BaseService<T> {
5      constructor (model: new() => T) {
6          super(model);
7      }
8
9      getAllEntities = async(): Promise<Array<T>> => {
10         return this.repository.find();
11     }
12
13     getEntityById = async (id: number): Promise<T | null> => {
14         return this.repository.findOneBy({id: id} as any);
15     }
16
17     createEntity = async (data: DeepPartial<T>): Promise<T> => {
18         const entity = this.repository.create(data);
19         return this.repository.save(entity);
20     }
21
22     updateEntity = async (id: number, data: DeepPartial<T>): Promise<T> => {
23         const entity = await this.getEntityById(id);
24         if (!entity) {
25             return null;
26         }
27         this.repository.merge(entity, data);
28         return this.repository.save(entity);
29     }
30
31     deleteEntity = async (id: number): Promise<boolean> => {
32         const result = await this.repository.delete(id);
33         return result.affected > 0;
34     }
35 }

```

Рисунок 3 - файл BaseCRUDService.ts

Далее были реализованы дженерики контроллеров: BaseController и BaseCRUDController. Их использование по аналогичной причине, что и для сервисов, также удобно поскольку позволяет переиспользовать код в разных контекстах. Они представлены на рисунках 4 и 5.

```

src > common > TS BaseController.ts > ...
1  import { BaseService } from "../BaseService";
2
3  export class BaseController<T> {
4      protected service: BaseService<T>
5      protected model: new () => T;
6
7      constructor(model: new () => T) {
8          this.model = model;
9          this.service = new BaseService<T>(model);
10     }
11 }

```

Рисунок 4 - файл BaseController.ts

```

src > routes > TS UserRouter.ts > ...
1  import { Router } from "express";
2  import { UserController } from "../controllers/UserController";
3
4  const router = Router();
5  const controller = new UserController();
6
7  router.get("/", controller.getUsers);
8  router.get("/:id", controller.getUserById);
9  router.post("/", controller.createUser);
10 router.put("/:id", controller.updateUser);
11 router.delete("/:id", controller.deleteUser);
12
13 export default router;
14

```

Рисунок 5 - файл BaseCRUDController.ts

Дальше, идёт инициализация конфигурационных параметров проекта. Это происходит в файлы config/settings.ts. Параметры подтягиваются либо из .env файла, либо берётся стандартное значение.

Наконец, была реализована возможность регистрации и авторизации на сервере. Для этого был реализован контроллер AuthController, который

содержит методы register и login, отвечающие за соответствующие функции. Они представлены на рисунках 6 и 7

```
register = async (request: Request, response: Response) => {
    const errors = new ValidationErrors();
    await validateRegister(request.body, errors, this.service);
    if (errors.isError) {
        response.status(422).json({errors: errors.errors});
        return;
    }

    const hash = hashPassword(request.body.password)
    const dto = {
        "username": request.body.username,
        "password": hash
    }
    try {
        const entity = await this.service.createEntity(dto);
        response.status(201).json(entity);
    } catch (error) {
        response.status(400).json({error: error.message});
    }
}
```

Рисунок 6 - метод register из AuthController

```

login = async (request: Request, response: Response) => {
  const errors = new ValidationErrors();
  validateLogin(request.body, errors);
  if (errors.isError) {
    response.status(422).json({errors: errors.errors});
    return;
  }

  const dto = {
    "username": request.body.username,
    "password": request.body.password
  };
  const user = await this.service.getEntityByUsername(dto.username);
  if (!user) {
    response.status(400).json({message: "Username or password incorrect"})
    return;
  }

  if (!checkPassword(dto.password, user.password)) {
    response.status(400).json({message: "Username or password incorrect"})
    return;
  }

  const accessToken = jwt.sign(
    {user: {
      id: user.id
    }},
    SETTINGS.JWT_SECRET_KEY,
    {
      expiresIn: SETTINGS.JWT_ACCESS_TOKEN_LIFETIME,
    }
  );

  response.json({"access_token": accessToken});
}

```

Рисунок 7 - метод login из AuthController

Остаётся проверить работоспособность проекта. Через роутеры контроллеры были подключены к приложению. После этого приложение было поднято локально. С помощью Burp Suite была проведена проверка работоспособности доступного API. На рисунках 8 и 9 представлена проверка методов, реализованных в AuthController, а на рисунках 10-12 - методов контроллера UserController.





Request		Response	
Pretty	Raw	Hex	Render
1	GET /api/user/21 HTTP/1.1	1	HTTP/1.1 200 OK
2	Host: 127.0.0.1:3000	2	X-Powered-By: Express
3		3	Content-Type: application/json; charset=utf-8
4		4	Content-Length: 175
		5	ETag: W/"af-bSOB8QEjkjKEjXCPSwb8Pd6H0rI"
		6	Date: Mon, 28 Apr 2025 13:45:30 GMT
		7	Connection: keep-alive
		8	Keep-Alive: timeout=5
		9	
		10	{
			"id": 21,
			"username": "username5",
			"password":
			"\$2b\$12\$fr3a6mv2qsCR3.CEdMtJUu3Z8xRkfhXGHJ57d0
			5N0TKSyEaTAPZEG",
			"avatar_url": null,
			"bio": null,
			"created_at": "2025-04-28T10:42:08.850Z"
			}

Рисунок 11 - получение данных о пользователе

Request		Response	
Pretty	Raw	Hex	Render
1	PUT /api/user/21 HTTP/1.1	1	HTTP/1.1 200 OK
2	Host: 127.0.0.1:3000	2	X-Powered-By: Express
3	Content-Type: application/json	3	Content-Type: application/json; charset=utf-8
4	Content-Length: 17	4	Content-Length: 180
5		5	ETag: W/"b4-s+gAyjlBLJYrcMoMDDpLK7s0nfk"
6	{	6	Date: Mon, 28 Apr 2025 13:46:38 GMT
	"bio": "UPDATED"	7	Connection: keep-alive
	}	8	Keep-Alive: timeout=5
		9	
		10	{
			"id": 21,
			"username": "username5",
			"password":
			"\$2b\$12\$fr3a6mv2qsCR3.CEdMtJUu3Z8xRkfhXGHJ5
			7d05N0TKSyEaTAPZEG",
			"avatar_url": null,
			"bio": "UPDATED",
			"created_at": "2025-04-28T10:42:08.850Z"
			}

Рисунок 12 - изменение пользователя

Request		Response	
Pretty	Raw	Hex	Render
1	DELETE /api/user/21 HTTP/1.1	1	HTTP/1.1 200 OK
2	Host: 127.0.0.1:3000	2	X-Powered-By: Express
3		3	Content-Type: application/json; charset=utf-8
4		4	Content-Length: 26
		5	ETag: W/"la-+BFCKZBji7vrbrwyE5s2YKViik"
		6	Date: Mon, 28 Apr 2025 13:47:15 GMT
		7	Connection: keep-alive
		8	Keep-Alive: timeout=5
		9	
		10	{
			"message": "User deleted"
			}

Рисунок 13 - удаление пользователя

## **Вывод**

В процессе работы был реализован boilerplate веб-приложения на стеке express + typeorm + typescript для более удобной дальнейшей реализации и поддержки. Были реализованы дженерик классы сервисов и контроллеров, реализован функционал аутентификации и проверены на работоспособность.