

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2
Работа с TypeORM

Выполнил:
Даньшин Семён
К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Реализовать все модели данных, спроектированные в рамках ДЗ1, используя TypeORM. Реализовать набор CRUD-методов для работы с моделями данных средствами Express + TypeScript. Реализовать API-эндпоинт для получения пользователя по id/email.

Ход работы

1. Анализ текущей реализации

В проекте вместо TypeORM используется PostgreSQL с pgx драйвером для Go и чистый SQL с миграциями. Это решение обеспечивает большую производительность и контроль над запросами.

2. Реализация моделей данных

Модели данных реализованы в Go в файле backend/internal/domain/domain.go:

Основные модели:

```
type User struct {
    Model
    Email    string
    Password string
    FirstName string
    LastName  string
    DateOfBirth time.Time
    Height    float32
    Weight    float32
    ProfilePicURL string
}

type Exercise struct {
    Model
    Name        string
    Description  string
    VideoURL    string
    TargetMuscleGroups []MuscleGroup
}

type Workout struct {
    Model
    UserID      ID
    RoutineID   utils.Nullable[ID]
    Notes       string
    Rating      int
    FinishedAt  time.Time
    IsAIGenerated bool
    Reasoning   string
}
```

3. Реализация Repository слоя

CRUD-операции реализованы в слое репозитория `backend/internal/repository/`:

Примеры CRUD-методов для пользователей:

```
func (r *PGXRepository) CreateUser(ctx context.Context, user domain.User) (domain.User, error)
func (r *PGXRepository) GetUserByID(ctx context.Context, id domain.ID) (domain.User, error)
func (r *PGXRepository) GetUserByEmail(ctx context.Context, email string) (domain.User, error)
func (r *PGXRepository) UpdateUser(ctx context.Context, user domain.User) (domain.User, error)
```

4. Реализация Service слоя

Бизнес-логика реализована в `backend/internal/service/`:

```
func (s *Service) CreateUser(ctx context.Context, dto dto.CreateUserDTO) (domain.User, error)
func (s *Service) GetUserByID(ctx context.Context, id domain.ID) (domain.User, error)
func (s *Service) UpdateUser(ctx context.Context, id domain.ID, dto dto.UpdateUserDTO) (domain.User, error)
```

5. API эндпоинты

API эндпоинты реализованы с использованием `gRPC + gRPC-Gateway`:

Получение пользователя по ID:

```
rpc GetUser(GetUserRequest) returns (UserResponse)
```

Получение текущего пользователя:

```
rpc GetMe(google.protobuf.Empty) returns (UserResponse)
```

6. Структура API

[Скриншот Swagger документации API]

Основные эндпоинты:

- GET `/v1/users/{userId}` - получение пользователя по ID
- GET `/v1/users/me` - получение текущего пользователя
- POST `/v1/users` - создание пользователя
- PUT `/v1/users` - обновление пользователя

7. Примеры использования API

Получение пользователя по ID:

```
const user = await authApi.v1.userServiceGetUser(userId);
```

Получение текущего пользователя:

```
const currentUser = await authApi.v1.userServiceGetMe();
```

8. Валидация и обработка ошибок

Реализована система валидации входных данных и обработки ошибок:

- Валидация на уровне протобуфов
- Кастомные ошибки домена (например, domain.ErrAlreadyExists)
- HTTP статус коды в соответствии с REST принципами

9. Тестирование

The screenshot shows a REST client interface with the following sections:

- Method and URL:** GET /v1/users/me. A description below reads: "Метод для получения текущего пользователя".
- Parameters:** A section labeled "Parameters" with a "Cancel" button. Below it, it says "No parameters".
- Buttons:** "Execute" (blue) and "Clear" (white) buttons.
- Responses:** A section with a dropdown menu for "Response content type" set to "application/json".
- Curl:** A text area containing the curl command:

```
curl -X 'GET' \
  'http://5056723-cd72267.twc1.net:8080/api/v1/users/me' \
  -H 'accept: application/json' \
  -H 'x-access-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1eW9ja3NlbnR5ImlhdCI6MTc1MDc2OTY2NiwiYW5ja3NlbnR5IjoiaW5ja3NlbnR5IiwiaWF0Ij0iZGRjYjYyVnVhbnR5IiwiaWF0Ij0iZGRjYjYyVnVhbnR5InQ'`
```
- Request URL:** A text area containing the URL: `http://5056723-cd72267.twc1.net:8080/api/v1/users/me`
- Server response:** A section with a "Code" column showing "200" and a "Details" column showing the response body.
- Response body:** A JSON object:

```
{
  "user": {
    "id": "ddcb5a0c-19fd-41c6-8a9d-e5e9c878ff64",
    "createdAt": "2025-06-14T19:20:57.244068Z",
    "email": "semen.danshin@gmail.com",
    "firstName": "Семён",
    "lastName": "",
    "dateOfBirth": "1970-01-01T00:00:00Z",
    "height": 0,
    "weight": 52,
    "updatedAt": "2025-06-21T09:59:28.477093Z",
    "profilePictureUrl": "https://Fitness-thing-object-storage-dev.storage.yandexcloud.net/AA0581E5-2319-4F5D-8869-37197A55470D-profile.jpg"
  }
}
```
- Download:** A "Download" button with a download icon.

Вывод

Несмотря на то, что в проекте не используется TypeORM, реализована полнофункциональная система для работы с данными:

1. **Модели данных** реализованы с использованием Go structs с полной типизацией
2. **CRUD операции** реализованы в слое репозитория с использованием чистого SQL
3. **API эндпоинты** созданы с использованием gRPC + REST через gRPC-Gateway
4. **Получение пользователя по ID/email** полностью реализовано и работает
5. **Архитектура** соответствует принципам чистой архитектуры с разделением на слои

Такой подход обеспечивает лучшую производительность и контроль над базой данных по сравнению с ORM-решениями.