Отчет по лабораторной работе 1 - Разработка базы данных

Цель работы
Целью данной работы являлась разработка базы данных для приложения, связанного с организацией и управлением походами. Основные задачи включали настройку PostgreSQL в Docker, проектирование ORM-моделей, реализацию функционала для работы с данными, а также оформление отчета с описанием структуры базы данных и используемых технологий.

База данных

Настройка базы данных
Для работы с базой данных был выбран PostgreSQL с расширением PostGIS, которое позволяет хранить и обрабатывать геопространственные данные. База данных была развернута в Docker-контейнере с использованием образа `postgis/postgis:17-3.5`. Конфигурация контейнера включала настройки окружения, такие как имя базы данных, пользователь и пароль, которые были вынесены в `.env` файл для удобства управления. Для доступа к базе данных извне контейнера был настроен порт `5433`.

```
# Database Service (PostgreSQL with PostGIS)
postgres:
  image: postgis/postgis:17-3.5
  container_name: hikemap-postgres
  env_file:
    - .env  # общий .env файл в корне
  environment:
    POSTGRES_DB: ${POSTGRES_DB}
    POSTGRES_USER: ${POSTGRES_USER}
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
  ports:
    - "5433:5432"
  volumes:
    - postgres-data:/var/lib/postgresql/data
  networks:
    - hikemap-network
```

Миграции базы данных выполнялись с помощью Liquibase, который также был запущен в Docker-контейнере. Liquibase обеспечивал управление изменениями структуры базы данных через файлы миграций, расположенные в директории `migrations`. Это позволило легко отслеживать и применять изменения в схеме базы данных.

Сущности

Пример таблицы `hike`

Таблица `hike` является центральной в структуре и содержит основную информацию о походе:

```
CREATE TABLE hike (
    id              SERIAL PRIMARY KEY,
    title           VARCHAR(255) NOT NULL,
    description     TEXT,
    photo_path      VARCHAR(255),
    start_date      DATE         NOT NULL,
    end_date        DATE         NOT NULL,
    track_gpx_path  VARCHAR(255),
    track_geometry  geometry(LineString, 4326),
    report_pdf_path VARCHAR(255),
    created_at      TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at      TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    organizer_id    INTEGER      NOT NULL REFERENCES user_account (id),
    area_id         INTEGER      NOT NULL REFERENCES area (id),
    difficulty      INTEGER      NOT NULL,
    is_categorical  BOOLEAN      DEFAULT FALSE,
    hike_type_id    INTEGER      NOT NULL REFERENCES hike_type (id),
    CONSTRAINT valid_dates CHECK (end_date >= start_date)
);
```

Пояснение:

- `title` — название похода;
- `description` — описание;
- `photo_path`, `track_gpx_path`, `report_pdf_path` — пути к загруженным файлам;
- `start_date` / `end_date` — даты начала и окончания похода;
- `track_geometry` — геометрия трассы типа `LineString` в системе координат WGS84 (SRID=4326), поддерживается PostGIS;
- `organizer_id` — внешний ключ на организатора (пользователя);
- `area_id` — регион проведения;

- `difficulty` — уровень сложности;
- `is_categorical` — флаг, указывающий, является ли поход категорийным;
- `hike_type_id` — тип похода (водный, пеший и т.д.);
- Проверка `valid_dates` гарантирует корректность временного диапазона.

Таблица ролей `role`

Хранит доступные роли пользователей.

```
CREATE TABLE role (
    id   SERIAL PRIMARY KEY,
    name VARCHAR(20) UNIQUE NOT NULL
);
```

Таблица пользователей `user_account`

Содержит данные о пользователях системы.

```
CREATE TABLE user_account (
    id            SERIAL PRIMARY KEY,
    username      VARCHAR(50) UNIQUE  NOT NULL,
    email         VARCHAR(255) UNIQUE NOT NULL,
    password_hash CHAR(60)            NOT NULL,
    role_id       INTEGER             NOT NULL REFERENCES Role (id) DEFAULT
1,
    created_at    TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at    TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

Таблица типов походов `hike_type`

Список возможных типов походов.

```
CREATE TABLE hike_type (
    id   SERIAL PRIMARY KEY,
    name VARCHAR(20) UNIQUE NOT NULL
);
```

Таблица регионов `area`

Регионы или зоны проведения походов.

```sql
CREATE TABLE area (
    id   SERIAL PRIMARY KEY,
    name VARCHAR(20) UNIQUE NOT NULL
);
```

Таблица лайков `user_hike_like`

Фиксирует факт "лайка" похода пользователем.

```sql
CREATE TABLE user_hike_like (
    id         SERIAL PRIMARY KEY,
    user_id    INTEGER NOT NULL REFERENCES user_account (id),
    hike_id    INTEGER NOT NULL REFERENCES hike (id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

ORM, запросы и бизнес логика

Разработка ORM-моделей

Основной сущностью приложения является `Hike` (поход), которая содержит информацию о названии, описании, датах начала и окончания, уровне сложности, типе похода и других атрибутах. Для хранения геоданных маршрута использовался тип `LineString` из библиотеки `geolatte-geom`, который сохраняется в PostgreSQL как геометрия с помощью аннотации `@Column(columnDefinition = "geometry(LineString,4326)")`.

```java
@Getter
@Setter
@Accessors(chain = true)
@Entity
@Table(name = "hike")
public class Hike {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(name = "title")
  private String title;

  @Column(name = "description")
```

```java
    private String description;

    @Column(name = "photo_path")
    private String photoPath;

    @Column(name = "start_date")
    private LocalDate startDate;

    @Column(name = "end_date")
    private LocalDate endDate;

    @Column(name = "track_gpx_path")
    private String trackGpxPath;

    @JsonIgnore
    @Basic(fetch = FetchType.LAZY)
    @Column(columnDefinition = "geometry(LineString,4326)")
    private LineString trackGeometry;

    @Column(name = "report_pdf_path")
    private String reportPdfPath;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "organizer_id", nullable = false)
    private User organizer;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "area_id", nullable = false)
    private Area area;

    @Column(name = "is_categorical", nullable = false)
    private boolean isCategorical;

    @Column(name = "difficulty", nullable = false)
    private int difficulty;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "hike_type_id", nullable = false)
```

```
  private HikeType hikeType;
}
```

Связи между сущностями были реализованы через JPA-аннотации. Например, поход (`Hike`) связан с пользователем (`User`) как организатором через отношение `@ManyToOne` , а также с сущностями `Area` (регион) и `HikeType` (тип похода). Для обеспечения целостности данных были добавлены ограничения `nullable = false` на обязательные поля.

Для преобразования данных между сущностями и DTO использовался MapStruct. Например, в классе `HikeMapper` были определены методы для конвертации `Hike` в `HikeResponse` , где поля `area` , `hikeType` и `organizer` маппились в их строковые представления (название региона, тип похода и имя организатора).

```
@Mapper(componentModel = MappingConstants.ComponentModel.SPRING)
public interface HikeMapper {
  @Mapping(target = "area", source = "hike.area.name") // например, берем
название региона
  @Mapping(target = "hikeType", source = "hike.hikeType.name") // берем
строковое значение типа
  @Mapping(target = "organizer", source = "hike.organizer.username")
  HikeResponse toResponse(Hike hike);

  Hike toEntity(HikeRequest hikeRequset);
}
```

Функционал для работы с данными

Сервис `HikeService` предоставляет методы для создания, чтения, обновления и удаления походов. Особое внимание было уделено валидации данных. Например, при создании или обновлении похода проверяется, что дата начала не позже даты окончания, а название похода не пустое. Для фильтрации походов использовался `Specification` из Spring Data JPA, что позволило гибко комбинировать условия поиска (по датам, сложности, региону и т. д.).

```
@Transactional(readOnly = true)
public List<HikeResponse> getAllHikes() {
  return hikeRepository
    .findAll()
    .stream()
```

```
    .map(hikeMapper::toResponse)
    .toList();
}

@Transactional
public Long createHike(HikeRequest hikeRequest) {
  Area area = areaRepository
    .findById(hikeRequest.areaId())
    .orElseThrow(() -> new IllegalArgumentException("Area not found"));

  HikeType hikeType = hikeTypeRepository
    .findById(hikeRequest.hikeTypeId())
    .orElseThrow(() -> new IllegalArgumentException("Hike type not found"));

  User organizer = userRepository
    .findById(hikeRequest.organizerId())
    .orElseThrow(() -> new IllegalArgumentException("Organizer not found"));

  Hike hike = hikeMapper.toEntity(hikeRequest);

  hike.setArea(area);
  hike.setHikeType(hikeType);
  hike.setOrganizer(organizer);

  hikeRepository.save(hike);
  return hike.getId();
}
```

Для работы с файлами (фотографии, треки GPX, отчеты PDF) был реализован сервис `FileServiceImpl`. Он использует стратегию `FileProcessor` для обработки файлов разных типов. Например, треки GPX конвертируются в `LineString` с помощью класса `GpxConverter`, который парсит XML-файл и извлекает координаты точек маршрута.

```
@Service
public class FileServiceImpl implements FileService {

  private final Map<String, FileProcessor> processors;
  private final HikeRepository hikeRepository;

  public FileServiceImpl(
    @Qualifier("fileProcessorMap") Map<String, FileProcessor> processors,
    HikeRepository hikeRepository
  ) {
    this.processors = processors;
    this.hikeRepository = hikeRepository;
```

```
  }

  @Override
  public String saveFile(Long hikeId, MultipartFile file, String fileType) {
    FileProcessor processor = processors.get(fileType.toUpperCase());

    if (processor == null) {
      throw new IllegalArgumentException(
        "File type is not supported: " + fileType
      );
    }

    String fileName;

    try {
      fileName = processor.processFile(hikeId, file);
    } catch (Exception e) {
      throw new RuntimeException(e);
    }

    return fileName;
  }

  @Override
  public Resource loadFile(Long hikeId, String fileType) {
    FileProcessor processor = processors.get(fileType.toUpperCase());

    if (processor == null) {
      throw new IllegalArgumentException("Unsupported file type: " +
fileType);
    }

    try {
      Resource resource = processor.loadFile(hikeId);
      if (resource == null || !resource.exists()) {
        throw new IllegalStateException("Failed to load file: " + hikeId);
      }
      return resource;
    } catch (Exception e) {
      throw new RuntimeException("Error loading file: " + hikeId, e);
    }
  }



  public void removeFile(Long hikeId, String fileType) {
    FileProcessor processor = processors.get(fileType.toUpperCase());
```

```
    if (processor == null) {
      throw new IllegalArgumentException("Unsupported file type: " +
fileType);
    }

    processor.removeFile(hikeId);
  }
}
```

Инструменты и технологии

В работе были использованы следующие технологии:

- **PostgreSQL + PostGIS** для хранения данных, включая геопространственные.
- **Spring Boot** как основа backend-части приложения.
- **Liquibase** для управления миграциями базы данных.
- **Docker** для контейнеризации сервисов (PostgreSQL и Liquibase).