Geekbrains

**Разработка web-сервиса для обеспечения топо-геодезических работ исходными данными для построения опорных и съёмочных сетей.**

IT-специалист:

Веб-разработка на Java

Низовкин А.В.

Шахты

2024

# Содержание.

1. Титульный лист

2. Содержание

3. Введение

4. Основная часть

5. Заключение

6. Список литературы и ресурсов

7. Приложения

# Введение.

Целью топо-геодезических работ является создание цифровой модели местности, которая в свою очередь определяется в пространстве набором точек с трёхмерными координатами.

Для этих целей используется различное современное оптико-электронное и спутниковое оборудование, но в любом случае исходными данными для этих работ является опорная геодезическая сеть.

Геодезическая сеть — совокупность специально обозначенных (закреплённых) точек земной поверхности (геодезических пунктов), положение которых определено в общей для них системе координат. Применяется в целях установления, распространения и связи предусмотренных геодезических систем координат и высот. Геодезические сети создают по принципу перехода от сетей более высокоточных и масштабных к сетям с меньшими расстояниями и менее точными измерениями

Геодезический пункт — точка, особым образом закреплённая на местности (в грунте, на строении или другом искусственном сооружении) и являющаяся носителем координат, определённых геодезическими методами.

Настоящий проект ставит перед собой целью разработку бэкэнда веб-приложения для работы с базой данных опорных геодезических пунктов, со следующей функциональностью:

* получение списка всех пунктов
* добавление пункта в базу данных
* изменение атрибутов пункта с заданным идентификатором
* удаление пункта с заданным идентификатором
* получение пункта по уникальному идентификатору
* получение списка пунктов по названию
* получение списка пунктов по номенклатуре листа
* получение списка пунктов в заданном районе

В проекте используются следующие инструменты и технологии:

**-** рабочая машина под управлением операционной системы **Ubuntu 20.04**

- разработка пояснительной записки к проекту в программе **LibreOffice Writer**

- формат пояснительной записки Word 2007-365(.docx)

- среда разработки **Intellij Idea(Community Edition)**

- система контроля версий **git**

- хранение файлов проекта публичный репозиторий на онлайн платформе **Github**

- реализация клиент-серверной архитектуры программы средствами фрэймворка **Spring**

- база данных **Postgres**

- эмуляция клиентской части средствами **Postman**

- документирование API средствами **Swagger**

- разработка Unit-тестов средствами библиотеки **Junit-5**

- разработка интеграционных тестов средствами фрэймворка **Mockito**

# Основная часть.

Разделы основной части.

## Хранение данных.

Для хранения данных о геодезических пунктах проектом предусматривается использовать базу данных Postgres.

Чтобы развернуть базу данных на рабочей машине, будем использовать контейнер docker. Для этого подбираем подходящий образ на сайте hub.docker.com.

Команду, запускающую наш контейнер прописываем в файле run-postgres.sh:

docker run -d --name postgres-container -e POSTGRES\_DB=basepointsdb -e POSTGRES\_USER=andrew -e POSTGRES\_PASSWORD=1234 -p 5432:5432 -v /home/andrew/basepointsdb:/var/lib/postgresql/data postgres:alpine3.19

Здесь:

-d – ключ, запускающий контейнер в режиме демона

--name postgres-container – название создаваемого контейнера

-e POSTGRES\_DB=basepointsdb – переменная окружения, определяющая название создаваемой базы данных

-e POSTGRES\_USER=andrew – переменная окружения, определяющая имя пользователя

-e POSTGRES\_PASSWORD=1234 – переменная окружения, определяющая пароль для входа в субд для пользователя andrew

-p 5432:5432 – указываем соответствие порта в контейнере порту на хост-машине

-v /home/andrew/basepointsdb:/var/lib/postgresql/data – указываем место на хост-машине, в котором будет сохраняться наша база данных. Это позволит сохранять данные между сеансами работы с бд

postgres:alpine3.19 – название выбранного на docker.hub образа

Для запуска созданного скрипта необходимо добавить права доступа на выполнение для файла. Для этого выполним в терминале команду:

sudo chmod +x run-postgres.sh

После этого запускаем скрипт терминальной командой из папки в которой он расположен:

./run-postgres.sh

Результат работы скрипта можно посмотреть с помощью терминальной команды, позволяющей увидеть список запущенных контейнеров:

docker ps

После этого, входим в контейнер в интерактивном режиме и убеждаемся в работе субд:

docker exec -it postgres-container psql -U andrew basepointsdb

\dt

Здесь:

\dt – отображает содержимое базы данных

## Разработка приложения.

### 1. Проектирование API.

Проектом предусматривается наличие в приложении следующих эндпоинтов для работы с базой данных геодезических пунктов:

1. GET /point: Получение списка всех геодезических опорных пунктов (далее пунктов), хранящихся в базе данных
2. GET /point/{id}: Получение информации о конкретном пункте. Здесь {id} – это уникальный идентификатор пункта.
3. GET /point/name/{name}: Получение списка всех пунктов с названием, совпадающим с переданным параметром {name}. Название пункта не является уникальным. Пункты с одинаковым названием могут иметь разные координаты, разный класс точности и расположены на разных листах карт.
4. GET /point/sheet/{sheet}: Получение списка всех пунктов, расположенных на листах карты с номенклатурой (см. Приложение А), совпадающей с переданным параметром {sheet}.
5. PUT /point/{id}: Обновление информации о существующем пункте. Здесь {id} – это уникальный идентификатор пункта, а обновлённые данные о пункте будут отправляться в теле запроса в формате JSON.
6. POST /point: Добавление нового пункта. Здесь данные о новом пункте будут отправляться в теле запроса в формате JSON.
7. POST/point/area: Получение списка всех пунктов, расположенных в конкретном районе. Здесь данные о районе, заданном координатами юго-западного угла и его размерами, отправляются в теле запроса в формате JSON.
8. DELETE /point/{id}: Удаление пункта из базы данных. Здесь {id} – это уникальный идентификатор пункта.

### 2. Проектирование структуры данных для пункта.

Каждый геодезический пункт будет иметь следующие атрибуты:

* id: Уникальный идентификатор пункта.
* name: Название пункта.
* x: Координата X пункта.
* y: Координата Y пункта.
* z: Координата Z пункта.
* sheet: Номенклатура листа карты масштаба 1:100 000, на котором расположен пункт (или наименование объекта для крупномасштабных съёмок).
* accuracyClass: Класс точности сети пункта.(см. Приложение Б)
* coordinateSystem: Система координат пункта.

### 3. Создание и настройка нового Spring проекта.

На сайте Spring Initializr выбираем следующие параметры:

- Project: Maven Project

- Language: Java

-Spring Boot: Выбираем последнюю стабильную версию

- Project Metadata: Вводим информацию о своём проекте

Добавляем следующие зависимости:

- Spring Web: Для создания веб-приложения с использованием Spring MVC.

- Spring Data JPA: Для работы с базой данных через JPA(Java Persistence API). Это спецификация для управления, доступа и сохранения Java объектов в базу данных. Это способ взаимодействия с базой данных с помощью объектно-ориентированного подхода.

- Lombok: Библиотека аннотаций Java, которая помогает сократить количество шаблонных кодов.

- PostgreSQL: Драйвер JDBC и R2 ODBC, который позволяет Java-программам подключаться к базе данных PostgreSQL, используя стандартный, независимый от базы данных Java-код.

После выбора всех параметров, генерируем наш проект, нажав кнопку «Generate», скачиваем и распаковываем архив в рабочую папку проекта.

Для подключения нашего проекта к базе данных, сервер которой уже запущен в docker-контейнере добавляем в файл srс/main/resources/application.yml следующие строки:

datasource:  
 url: jdbc:postgresql://localhost:5432/basepointsdb  
 username: andrew  
 password: 1234  
 driver-class-name: org.postgresql.Driver  
jpa:  
 hibernate:  
 ddl-auto: create-drop  
 show-sql: true

Здесь мы указываем параметры, использованные при запуске контейнера с базой данных:

url: адрес, порт и название базы данных

username: имя пользователя

password: пароль

ddl-auto: параметр библиотеки Hibernate, контролирующий автоматическое создание и обновление схемы базы данных. В нашем случае, при разработке и отладке приложения используем значение «create-drop», при котором Hibernate будет удалять существующую схему базы данных при каждом запуске и после завершения работы.

show-sql: этот параметр позволит отображать sql-запросы, что удобно при разработке и отладке приложения.

### 4. Слой model

Для работы с базой данных через JPA, создадим класс-сущность, который соответствует таблице в базе данных.

Класс BasePoint.java

import jakarta.persistence.\*;  
import lombok.Data;  
import lombok.NoArgsConstructor;

@Entity  
@Data  
@Table(name = "points")  
@NoArgsConstructor  
public class BasePoint {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private long id;  
  
 @Column(name = "name", nullable = false)  
 private String name;  
  
 @Column(name = "x", nullable = false)  
 private long x;  
  
 @Column(name = "y", nullable = false)  
 private long y;  
  
 @Column(name = "z", nullable = false)  
 private long z;  
  
 @Column(name = "sheet")  
 private String sheet;  
  
 @Column(name = "accuracy\_class")  
 private String accuracyClass;  
  
 @Column(name = "coordinate\_system")  
 private String coordinateSystem;

Здесь используются следующие аннотации:

@Entity: указывает, что этот класс является JPA сущностью и должен быть отображён на таблицу базы данных.

@Table(name = «points»): указывает название таблицы в базе данных, с которой связана наша сущность.

@Data: генерирует шаблонный код, добавляя геттеры для всех полей, сеттеры для всех нефинальных полей, правильные реализации toString, equals, hashCode, конструктор для всех нефинальных полей, формируя таким образом объект POJO, необходимый для работы с JPA технологией.

@NoArgsConstructor: генерирует пустой конструктор.

@Id: указывает на то, что это поле является идентификатором, первичным ключом для базы данных.

@GeneratedValue(strategy = GenerationType.IDENTITY): этой аннотацией обычно помечается то же поле, что и аннотацией @Id. У неё есть четыре возможных стратегии (AUTO, IDENTITY, SEQUENCE, TABLE). В нашем случае ( GenerationType.IDENTITY), Hibername делегирует установку ID на уровень базы данных. При этом используется колонка, помеченная как PRIMARY KEY, AUTOINCREMENT.

@Column: позволяет задать имя колонки и другие параметры, такие как nullable и length. В нашем случае (nullable = false) означает, что для этих полей невозможно пустое значение.

### 5. Слой Repositoty

JpaRepositoty предоставляет множество полезных встроенных методов для работы с базой данных, а также позволяет создать пользовательские методы на базе SQL-запросов.

Интерфейс BasePointRepository

public interface BasePointRepository extends JpaRepository<BasePoint, Long> {

@Query("select b from BasePoint b where b.name like %?1%")  
 List<BasePoint> findByName(String name);

@Query("select b from BasePoint b where b.sheet like %?1%")  
 List<BasePoint> findBySheet(String sheet);

@Query("select p from BasePoint p where (p.x between :xSouth and :xNorth) and (p.y between :yWest and :yEast)")  
 List<BasePoint> findByArea(@Param("xSouth") long xSouth, @Param("yWest") long yWest, @Param("xNorth") long xNorth, @Param("yEast") long yEast);

}

Здесь:

@Query: аннотация, позволяющая создать SQL-запрос к базе данных и внедрить в него параметр, передаваемый в пользовательский метод.

С помощью конструкции ?1, указываем порядковый номер параметра в описании пользовательского метода, который передаём в SQL-запрос.

С помощью конструкции :xNorth, передаём в SQL-запрос параметр метода, помеченный аннотацией @Param(«xNorth»).

Дополняем JpaRepository следующими методами:

findByName: Выбирает в базе данных записи о пунктах, название которых содержит фрагмент, переданный аргументом.

findBySheet: Выбирает в базе данных записи о пунктах, название листа карты (название объекта) содержит фрагмент, переданный аргументом.

findByArea: Выбирает в базе данных записи о пунктах, которые попадают в область, параметры которой передаются аргументами. Прямоугольная область описывается плановыми (xSouth, yWest) координатами юго-западного угла и координатами северо-восточного угла (xNorth, yEast).

### 6. Слой Service

Слой сервиса – место, где выполняется основная бизнес-логика нашего приложения. В соответсвии с принципами SOLID, вначале создадим интерфейс, что сделает наш код более гибким, менее связанным.

Здесь мы определяем основные операции, которые проектом предусматривается выполнять над нашими данными.

Интерфейс BasePointService

public interface BasePointService {

// Следующий метод возвращает все записи из базы данных.

List<BasePointDto> getAll();

// Следующий метод возвращает запись с указанным id.

BasePointDto getById(long id);

// Следующий метод возвращает список геодезических пунктов, название которых содержит указанный фрагмент «name».

List<BasePointDto> getByName(String name);

// Следующий метод возвращает список геодезических пунктов, номенклатура листа М 1: 100 000 или название объекта которых содержит указанный фрагмент «sheet».

List<BasePointDto> getBySheet(String sheet);

// Следующий метод возвращает список геодезических пунктов, внутри области, описанной объектом «areaDto».

List<BasePointDto> getByArea(AreaDto areaDto);

// Следующий метод создаёт в базе данных запись с параметрами, переданными объектом «basePointDto»

BasePointDto createBasePoint(BasePointDto basePointDto);

// Следующий метод удаляет из базы данных запись с идентификатором «id».

BasePointDto removeById(long id);

// Следующий метод обновляет в базе данных поля записи с идентификатором «id», присваивая ей значения переданные объектом «basePointDto».

BasePointDto updatePoint(long id, BasePointDto basePointDto);

}

Методы, описанные в интерфейсе BasePointService реализуем в классе BasePointServiceImpl:

@Service  
@RequiredArgsConstructor  
public class BasePointServiceImpl implements BasePointService{  
 private static final String NOT\_FOUND\_POINT\_MESSAGE = "Couldn't find the point with the ";

private final BasePointRepository basePointRepository;

private final BasePointMapper basePointMapper;

@Override  
 public List<BasePointDto> getAll() {  
 return basePointMapper.toListBasePointDto(basePointRepository.findAll());  
 }

@Override  
 public BasePointDto getById(long id) {  
 BasePoint basePoint = basePointRepository.findById(id).orElse(null);  
 if (basePoint != null) {  
 return basePointMapper.toBasePointDto(basePoint);  
 } else {  
 throw new NoSuchElementException(NOT\_FOUND\_POINT\_MESSAGE + "id = " + id);  
 }  
 }

@Override  
 public List<BasePointDto> getByName(String name) {   
 List<BasePoint> basePoints = basePointRepository.findByName(name);  
 if (basePoints.isEmpty()) {  
 throw new NoSuchElementException(NOT\_FOUND\_POINT\_MESSAGE + "name = " + name);  
 }  
 return basePointMapper.toListBasePointDto(basePoints);  
 }

@Override  
 public List<BasePointDto> getBySheet(String sheet) {   
 List<BasePoint> basePoints = basePointRepository.findBySheet(sheet);  
 if (basePoints.isEmpty()) {  
 throw new NoSuchElementException(NOT\_FOUND\_POINT\_MESSAGE + "sheet = " + sheet);  
 }  
 return basePointMapper.toListBasePointDto(basePoints);  
 }

@Override  
 public List<BasePointDto> getByArea(AreaDto areaDto) {  
 long xNorthEast = areaDto.getX() + areaDto.getAreaHeight();  
 long yNorthEast = areaDto.getY() + areaDto.getAreaWidth();  
 List<BasePoint> basePoints = basePointRepository  
 .findByArea(areaDto.getX(),  
 areaDto.getY(),  
 xNorthEast,  
 yNorthEast);  
 if (basePoints.isEmpty()) {  
 throw new NoSuchElementException("No points were found inside the specified area");  
 }  
 return basePointMapper.toListBasePointDto(basePoints);  
 }

@Override  
 @Transactional  
 public BasePointDto createBasePoint(BasePointDto basePointDto) {  
 BasePoint basePoint = basePointMapper.toBasePoint(basePointDto);  
 return basePointMapper.toBasePointDto(basePointRepository.saveAndFlush(basePoint));  
 }

@Override  
 @Transactional  
 public BasePointDto removeById(long id) {  
 BasePoint basePoint = basePointRepository.findById(id).orElse(null);  
 if (basePoint == null) {  
 throw new NoSuchElementException(NOT\_FOUND\_POINT\_MESSAGE + "id = " + id);  
 }  
 basePointRepository.deleteById(id);  
 return basePointMapper.toBasePointDto(basePoint);  
 }

@Override  
 @Transactional  
 public BasePointDto updatePoint(long id, BasePointDto basePointDto) {  
 BasePoint basePoint = basePointRepository.findById(id).orElse(null);  
 if (basePoint == null) {  
 throw new NoSuchElementException(NOT\_FOUND\_POINT\_MESSAGE + "id = " + id);  
 }  
 basePoint.setName(basePointDto.getName());  
 basePoint.setX(basePointDto.getX());  
 basePoint.setY(basePointDto.getY());  
 basePoint.setZ(basePointDto.getZ());  
 basePoint.setSheet(basePointDto.getSheet()); basePoint.setAccuracyClass(basePointDto.getAccuracyClass());  
 return basePointMapper.toBasePointDto(basePointRepository.saveAndFlush(basePoint));  
 }  
}

Здесь:

@Service: сообщает фреймворку, что класс является компонентом приложения и указывает, что он содержит бизнес-логику

@RequiredArgsConstructor: позволяет получить конструктор с final полями. Таким образом, в нашем случае, внедряем в сервис BasePointRepository и BasePointMapper.

@Override: маркерная аннотация, указывающая компилятору на то, что компонент подкласса переопределяет элемент родительского класса, в нашем случае интерфейса BasePointService.

@Transactional: помечает метод, который исполняется в рамках транзакции и позволяет установить различные параметры, такие как Propagation и Isolation. В нашем случае устанавливаются значения по умолчанию.

propogation = Propagation.REQUIRED: Spring проверяет, есть ли активная транзакция, и если ничего не существует, создает новую. В противном случае бизнес-логика добавляется к текущей активной транзакции.

Isolation это одно из свойств ACID. Описывает как изменения, применяемые параллельными транзакциями, видны друг другу.

Isolation = Isolation.READ\_COMMITTED: уровень изоляции по умолчанию для Postgres. Этот уровень изоляции предтвращает dirty reads (грязное чтение). Остальные побочные эффекты параллелизма все еще могут иметь место. Таким образом, незафиксированные изменения в параллельных транзакциях никак на нас не влияют, но если транзакция зафиксирует свои изменения, наш результат может измениться при повторном запросе. Здесь речь идёт о следующих, возможных эффектах параллелизма:

- Dirty read: считывание незафиксированного изменения параллельной транзакции.

- Nonrepeatable read: получение другого значения при повторном чтении строки, если параллельная транзакция обновляет ту же строку и фиксирует.

-Phantom read: получение других строк после повторного выполнения запроса диапазона, если другая транзакция добавляет или удаляет некоторые строки в диапазоне и фиксирует.

В случае возникновения ошибки, например отсутствия или некорректного запроса имени пункта сервис пробрасывает исключение на уровень контроллера.

Для большей гибкости, меньшей связности и взаимодействия между слоями добавляем в проект классы BasePointDto и AreaDto.

Класс BasePointDto:

@Data  
public class BasePointDto {

private long id;

private String name;

private long x;

private long y;

private long z;

private String sheet;

private String accuracyClass;

private String coordinateSystem;

}

Здесь:

@Data: генерирует шаблонный код, добавляя геттеры для всех полей, сеттеры для всех нефинальных полей, правильные реализации toString, equals, hashCode, конструктор для всех нефинальных полей, формируя таким образом объект POJO, необходимый для работы с JPA технологией.

Класс AreaDto:

@Data  
@NoArgsConstructor  
public class AreaDto {

private long x;

private long y;

private long areaHeight;

private long areaWidth;

}

Здесь:

x: Координата X юго-западного угла прямоугольной области.

y: Координата Y юго-западного угла прямоугольной области.

areaHeight: Высота области.

areaWidth: Ширина области.

@NoArgsConstructor: генерирует пустой конструктор.

Для преобразования объектов между слоями Model и Dto служит класc BasePointMapperImpl, который, следуя принципам SOLID, внедряем в реализацию сервиса через интерфейс BasePointMapper.

Интерфейс BasePointMapper:

public interface BasePointMapper {

// Следующий метод возвращает экземпляр класса BasePoint, значения полей которого соответствуют полям объекта «BasePointDto».

BasePoint toBasePoint(BasePointDto basePointDto);

// Следующий метод возвращает экземпляр класса BasePointDto, значения полей которого соответствуют полям объекта «BasePoint».

BasePointDto toBasePointDto(BasePoint basePoint);

// Следующий метод преобразует список экземпляров класса BasePoint в список экземпляров класса BasePointDto и возвращает его.

List<BasePointDto> toListBasePointDto(List<BasePoint> basePoints);

// Следующий метод преобразует список экземпляров класса BasePointDto в список экземпляров класса BasePoint и возвращает его.

List<BasePoint> toListBasePoint(List<BasePointDto> basePointsDto);

}

Класс BasePointMapperImpl:

@Component  
public class BasePointMapperImpl implements BasePointMapper{

public BasePoint toBasePoint(BasePointDto basePointDto) {

BasePoint basePoint = new BasePoint();

basePoint.setName(basePointDto.getName());

basePoint.setX(basePointDto.getX());

basePoint.setY(basePointDto.getY());

basePoint.setZ(basePointDto.getZ());

basePoint.setSheet(basePointDto.getSheet());

basePoint.setAccuracyClass(basePointDto.getAccuracyClass());

basePoint.setCoordinateSystem(basePointDto.getCoordinateSystem());

return basePoint;

}

public BasePointDto toBasePointDto(BasePoint basePoint) {

BasePointDto basePointDto = new BasePointDto();

basePointDto.setId(basePoint.getId());

basePointDto.setName(basePoint.getName());

basePointDto.setX(basePoint.getX());

basePointDto.setY(basePoint.getY());

basePointDto.setZ(basePoint.getZ());

basePointDto.setSheet(basePoint.getSheet());

basePointDto.setAccuracyClass(basePoint.getAccuracyClass());

basePointDto.setCoordinateSystem(basePoint.getCoordinateSystem());

return basePointDto;

}

public List<BasePointDto> toListBasePointDto(List<BasePoint> basePoints) {

return basePoints.stream().map(this::toBasePointDto).toList();

}

public List<BasePoint> toListBasePoint(List<BasePointDto> basePointsDto) {

return basePointsDto.stream().map(this::toBasePoint).toList();

}

}

Здесь:

@Component: Указывает, что этот класс является компонентом. Такие классы автоматически сканируются Spring и регистрируются в контейнере IoC.

В методах toListBasePointDto и toListBasePoint используем возможности Java Stream Api, отображая и преобразуя данные с помощью функции map в нужный нам тип. В качестве функционального интерфейса типа Function, передаём аргументом ссылку на соответствующий метод класса.

### Слой Controller.

Слой контроллеров отвечает за обработку HTTP-запросов и взаимодействует со слоем сервисов для выполнения бизнес-логики.

Класс BasePointController:

@RestController

@RequestMapping("point")

@RequiredArgsConstructor

public class BasePointController {

private final BasePointService basePointService;

@GetMapping

public ResponseEntity<List<BasePointDto>> getAllBasePoints() {

return ResponseEntity.ok().body(basePointService.getAll());

}

@GetMapping("{id}")

public ResponseEntity<BasePointDto> getBasePointById(@PathVariable long id) {

try {

return ResponseEntity.ok().body(basePointService.getById(id));

} catch (NoSuchElementException e) {

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, e.getMessage());

}

}

@GetMapping("name/{name}")

public ResponseEntity<List<BasePointDto>> getBasePointByName(@PathVariable String name) {

try {

return ResponseEntity.ok().body(basePointService.getByName(name));

} catch (NoSuchElementException e) {

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, e.getMessage());

}

}

@GetMapping("sheet/{sheet}")

public ResponseEntity<List<BasePointDto>> getBasePointBySheet(@PathVariable String sheet) {

try {

return ResponseEntity.ok().body(basePointService.getBySheet(sheet));

} catch (NoSuchElementException e) {

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, e.getMessage());

}

}

@DeleteMapping("{id}")

public ResponseEntity<BasePointDto> removeBasePointById(@PathVariable long id) {

try {

return ResponseEntity.ok().body(basePointService.removeById(id));

} catch (NoSuchElementException e) {

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, e.getMessage());

}

}

@PostMapping

public ResponseEntity<BasePointDto> createBasePoint(@RequestBody BasePointDto basePointDto) {

return new ResponseEntity<>(basePointService.createBasePoint(basePointDto),HttpStatus.CREATED);

}

@PostMapping("area")

public ResponseEntity<List<BasePointDto>> getByArea(@RequestBody AreaDto areaDto) {

try {

return ResponseEntity.ok().body(basePointService.getByArea(areaDto));

} catch (NoSuchElementException e) {

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, e.getMessage());

}

}

@PutMapping("{id}")

public ResponseEntity<BasePointDto> updateBasePoint(@PathVariable long id, @RequestBody BasePointDto basePointDto) {

try {

return ResponseEntity.ok().body(basePointService.updatePoint(id, basePointDto));

} catch (NoSuchElementException e) {

throw new ResponseStatusException(HttpStatus.NOT\_FOUND, e.getMessage());

}

}

}

Здесь:

**@RestController**: объединяет в себе аннотации @Controller и @ResponseBody. Она не только помечает класс как Spring MVC Controller, но и автоматически преобразует возвращаемые данные в формат JSON или XML. REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Таким образом наш микросервис легко внедрить в другие системы.

**@RequestMappping(«point»)**: эта аннотация задаёт базовый маршрут для всех методов в этом контроллере. Все эндпоинты будут начинаться с «/point».

**@GetMapping, @PostMapping, @PutMapping, @DeleteMapping**: Эти аннотации задают HTTP-метод для каждого из методов контроллера. Главным образом, они соответствуют стандартным CRUD-операциям: получение, создание, обновление и удаление.

**@PathVariable**: эта аннотация используется для передачи элемента пути запроса в виде параметра метода контроллера.

**@RequestBody**: эта аннотация помечает параметр в методе контроллера, сопоставляя его с телом запроса.

Методы контроллера обращаются к соответствующим методам сервиса для выполнения операции и возвращает результат в теле объекта ResponseEntity<T>.

Контроллер обрабатывает исключения вбрасываемые на уровне сервисного слоя и возвращает исключение типа ResponseStatusException.

**ResponseStatusException**: является программной альтернативой @ResponseStatus и базовым классом для исключений, используемых для применения кода состояния к HTTP-ответу. Это RuntimeException, и, следовательно, его не требуется явно добавлять в сигнатуру метода.

## Разработка тестов.

### 1. Настройка проекта.

Для снижения вероятности наличия дефектов, которые могут находиться в разрабатываемом проектом микро-сервисе, предусматривается разработка различных типов тестов, среди которых:

- unit-тесты

- интеграционные тесты

- сквозное тестирование

Для использования возможностей Junit-5, Mockito в проект внедряется Spring- boot в файл pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

Для целей интеграционного и сквозного тестирования проект использует базу данных H2. Это открытая, кроссплатформенная СУБД, имеющая режим хранения данных в памяти. Добавляем зависимость в pom.xml:

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

Для выполнения сквозных HTTP-тестов проект использует WebTestClient. Для этого добавляем следующую зависимость в pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-webflux</artifactId>

</dependency>

Для работы тестов с внедрённой базой данных H2, в папку test/resources добавляем файл application-test.yaml, со следующим содержимым:

server:

port: 8180

spring:

datasource:

url: jdbc:h2:mem:test

username: andrew

password:

driver-class-name: org.h2.Driver

jpa:

hibernate:

ddl-auto: create-drop

Здесь:

port: 8180 – свободный порт, на котором будет запускаться наш сервер БД

url: jdbc:h2:mem:test – параметр mem указывает, что хранение данных будет в оперативной памяти

Остальные параметры сходны с основными настройками.

В соответствии с принципом DRY(Don`t Repeat Yourself), нацеленного на снижение повторения информации различного рода, добавляем в проект класс PointComparator, методы которого сравнивают поля экземпляров классов BasePoint и BasePointDto:

@Component

public class PointComparator {

// Следующий метод возвращает истину при равенстве соответствующих полей экземпляров класса BasePoint и BasePointDto.

public boolean compareBasePointBasePointDTO(BasePoint basePoint, BasePointDto basePointDto) {

return basePoint.getId() == basePointDto.getId()&& basePoint.getName().equals(basePointDto.getName()) &&

basePoint.getX() == basePointDto.getX() &&

basePoint.getY() == basePointDto.getY() &&

basePoint.getZ() == basePointDto.getZ() && basePoint.getSheet().equals(basePointDto.getSheet()) && basePoint.getAccuracyClass().equals(basePointDto.getAccuracyClass()) && basePoint.getCoordinateSystem().equals(basePointDto.getCoordinateSystem());

}

// Следующий метод возвращает истину при равенстве соответствующих полей экземпляров класса BasePointDto и BasePoint, за исключением поля id.

public boolean compareBasePointDtoBasePoint(BasePointDto basePointDto, BasePoint basePoint) {

return basePointDto.getName().equals(basePoint.getName()) &&

basePointDto.getX() == basePoint.getX() &&

basePointDto.getY() == basePoint.getY() &&

basePointDto.getZ() == basePoint.getZ() && basePointDto.getSheet().equals(basePoint.getSheet()) && basePointDto.getAccuracyClass().equals(basePoint.getAccuracyClass()) && basePointDto.getCoordinateSystem().equals(basePoint.getCoordinateSystem());

}

// Следующий метод возвращает истину, при условии равенства всех полей двух экземпляров класса BasePoinDto, за исключением поля id.

public boolean compareBasePointDtoBasePointDto(BasePointDto expectPoint, BasePointDto actualPoint) {

return expectPoint.getName().equals(actualPoint.getName()) && expectPoint.getX() == actualPoint.getX() && expectPoint.getY() == actualPoint.getY() && expectPoint.getZ() == actualPoint.getZ() && expectPoint.getSheet().equals(actualPoint.getSheet()) && expectPoint.getAccuracyClass().equals(actualPoint.getAccuracyClass()) && expectPoint.getCoordinateSystem().equals(actualPoint.getCoordinateSystem());

}

}

### 2. Unit-тесты.

Для тестирования методов класса PointComparator создаём класс PointComparatorTest:

import static org.junit.jupiter.api.Assertions.\*;

// Используем статический импорт для уменьшения повторяющегося кода.

@SpringBootTest(classes = PointComparator.class)

class PointComparatorTest {

// @SptingBootTest: аннотация, которая указывает Spring, что тестовый класс является тестом Spring Boot. Это позволяет автоматически конфигурировать приложение, поднимает встроенный сервер, создаёт реальный контекст приложения, что позволяет тестировать его так, как если бы оно работало в продакшене. Для тестирования этого класса весь контекст не нужен, поэтому указываем параметром:

(classes = PointComparator.class): необходимый для тестов класс.

@Autowired

private PointComparator pointComparator;

private BasePoint basePoint;

private BasePointDto basePointDto;

// @Autowired: автоматически связывает бин с классом.

Вводим в класс две private-переменные, с которыми будем производить сравнения.

@BeforeEach

// эта аннотация обеспечит выполнение метода перед каждым тестом, который выполняет инициализацию сравниваемых переменных.

public void setUp() {

basePoint = new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

);

basePointDto = new BasePointDto();

basePointDto.setId(basePoint.getId());

basePointDto.setName(basePoint.getName());

basePointDto.setX(basePoint.getX());

basePointDto.setY(basePoint.getY());

basePointDto.setZ(basePoint.getZ());

basePointDto.setSheet(basePoint.getSheet()); basePointDto.setAccuracyClass(basePoint.getAccuracyClass()); basePointDto.setCoordinateSystem(basePoint.getCoordinateSystem());

}

// Следующий тест проверяет работу метода, сравнивающего все поля экземпляров классов BasePoint и BasePointDto.

@Test

void compareBasePointBasePointDTOTest() {

assertTrue(pointComparator.compareBasePointBasePointDTO(

basePoint, basePointDto));

}

// Следующий тест проверяет работу метода, сравнивающего все поля двух экземпляров класса BasePointDto, за исключением поля id.

@Test

void compareBasePointDtoBasePoint() {

assertTrue(pointComparator.compareBasePointDtoBasePointDto(

basePointDto, basePointDto

));

}

// Следующий тест проверяет работу метода, сравнивающего все поля экземпляров классов BasePointDto и BasePoint, за исключением поля id.

@Test

void compareBasePointDtoBasePointDto() {

assertTrue(pointComparator.compareBasePointDtoBasePoint(

basePointDto, basePoint

));

}

}

Для тестирования методов класса BasePointMapperImpl служит тестовый класс BasePointMapperImplTest:

import static org.junit.jupiter.api.Assertions.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

@SpringBootTest(classes = {

BasePointMapperImpl.class,

PointComparator.class

})

// Параметрами аннотации @SpringBootTest указываем необходимые для тестирования классы, чтобы не загружать весь контекст приложения

class BasePointMapperImplTest {

Random random = new Random();

// Внедряем экземпляр класса Random для создания тестируемых объектов.

@Autowired

BasePointMapper basePointMapper;

@Autowired

PointComparator pointComparator;

// Внедряем тестируемый BasePointMapper и необходимый для сравнения PointComparator

// Следующий тест проверяет корректность преобразования экземпляра BasePointDto в BasePoint.

@Test

void toBasePointTest() {

BasePointDto expectBasePointDto = new BasePointDto();

expectBasePointDto.setName("1001");

expectBasePointDto.setX(999999999999L);

expectBasePointDto.setY(888888888888L);

expectBasePointDto.setZ(9000000000L);

expectBasePointDto.setSheet("L-37-32");

expectBasePointDto.setAccuracyClass("2кл");

expectBasePointDto.setCoordinateSystem("МСК-61-2");

BasePoint actualBasePoint = basePointMapper.toBasePoint(expectBasePointDto);

assertNotNull(actualBasePoint); assertTrue(pointComparator.compareBasePointDtoBasePoint(expectBasePointDto, actualBasePoint));

}

// Следующий тест проверяет корректность преобразования экземпляра BasePointDto в BasePoint.

@Test

void toBasePointDtoTest() {

BasePoint expectBasePoint = new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

);

BasePointDto actualBasePointDto = basePointMapper.toBasePointDto(expectBasePoint);

assertNotNull(actualBasePointDto); assertTrue(pointComparator.compareBasePointBasePointDTO(expectBasePoint, actualBasePointDto));

}

// Следующий тест проверяет корректность преобразования списка экземпляров BasePoint в список BasePointDto

@Test

void toListBasePointDtoTest() {

List<BasePoint> expectBasePointList = new ArrayList<>();

for (int i = 0; i < 10; i++) {

expectBasePointList.add(new BasePoint(

"Point\_N\_" + random.nextInt(100),

random.nextLong(1000000000),

random.nextLong(1000000000),

random.nextLong(1000000),

"L-37-" + random.nextInt(144),

random.nextInt(4) + "кл",

String.valueOf(random.nextInt(10000))

));

}

List<BasePointDto> actualBasePointDtoList = basePointMapper.toListBasePointDto(expectBasePointList);

assertNotNull(actualBasePointDtoList);

assertEquals(expectBasePointList.size(), actualBasePointDtoList.size());

for (int i = 0; i < expectBasePointList.size(); i++) {

boolean result = pointComparator.compareBasePointBasePointDTO(expectBasePointList.get(i), actualBasePointDtoList.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет корректность преобразования списка экземпляров BasePointDto в список BasePoint

@Test

void toListBasePointTest() {

List<BasePointDto> expectBasePointDtoList = new ArrayList<>();

for (int i = 0; i < 10; i++) {

BasePointDto basePointDto = new BasePointDto();

basePointDto.setName("Point\_N\_" + random.nextInt(100));

basePointDto.setX(random.nextLong(1000000000));

basePointDto.setY(random.nextLong(1000000000));

basePointDto.setZ(random.nextLong(1000000));

basePointDto.setSheet("L-37-" + random.nextInt(144));

basePointDto.setAccuracyClass(random.nextInt(4) + "кл");

basePointDto.setCoordinateSystem(String.valueOf(random.nextInt(10000)));

expectBasePointDtoList.add(basePointDto);

}

List<BasePoint> actualBasePointList = basePointMapper.toListBasePoint(expectBasePointDtoList);

assertNotNull(actualBasePointList);

assertEquals(expectBasePointDtoList.size(), actualBasePointList.size());

for (int i = 0; i < expectBasePointDtoList.size(); i++) {

boolean result = pointComparator.compareBasePointDtoBasePoint(expectBasePointDtoList.get(i), actualBasePointList.get(i));

assertTrue(result);

}

}

}

### 3. Интеграционное тестирование сервиса.

Интеграционные тесты дадут нам уверенность в том, что разные части нашего приложения корректно работают вместе, как и ожидалось.

Для тестирования класса BasePointService и проверки взаимодействия различных слоёв приложения служит класс BasePointServiceTest:

import static org.junit.jupiter.api.Assertions.\*;

// Чтобы не загружать весь контекст приложения, указываем необходимые для тестирования классы:

@SpringBootTest(classes = {

BasePointMapperImpl.class,

BasePointServiceImpl.class,

PointComparator.class,

})

class BasePointServiceTest {

// Здесь внедряем необходимые зависимости. BasePointRepository заменяем Mock-объектом, что позволит при тестировании определять поведение его методов:

@Autowired

private BasePointService basePointService;

@Autowired

private BasePointMapper basePointMapper;

@Autowired

private PointComparator pointComparator;

@MockBean

private BasePointRepository basePointRepository;

@Test

void getAllTest() {

List<BasePoint> expectBasePointList = createDemoPoints();

Mockito.when(basePointRepository.findAll()).thenReturn(expectBasePointList);

List<BasePointDto> actualBasePointList = basePointService.getAll();

assertNotNull(actualBasePointList);

assertEquals(expectBasePointList.size(), actualBasePointList.size());

boolean result;

for (int i = 0; i < expectBasePointList.size(); i++) {

result = pointComparator.compareBasePointBasePointDTO(expectBasePointList.get(i), actualBasePointList.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет работу метода getById(), в случае, если база данных содержит запись с указанным id.

@Test

void getByIdTest() {

BasePoint expectBasePoint = new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

);

Optional<BasePoint> basePointOptional = Optional.of(expectBasePoint);

Mockito.when(basePointRepository.findById(1L)).thenReturn(basePointOptional);

BasePointDto actualBasePointdto = basePointService.getById(1);

assertNotNull(actualBasePointdto);

boolean result = pointComparator.compareBasePointBasePointDTO(expectBasePoint, actualBasePointdto);

assertTrue(result);

}

// Следующий тест проверяет работу метода getById(), в случае, если в базе данных нет записи с указанным Id.

@Test

void getByIdThrowException() {

Mockito.when(basePointRepository.findById(1L)).thenReturn(null);

assertThrows(RuntimeException.class, () -> basePointService.getById(1L));

}

// Следующий тест проверяет работу метода getByName(), в том случае, когда база данных содержит записи, соответствующие переданному шаблону.

@Test

void getByNameTest() {

List<BasePoint> expectBasePointList = createDemoPoints();

Mockito.when(basePointRepository.findByName("Гор")).thenReturn(expectBasePointList);

List<BasePointDto> actualBasePointDtoList = basePointService.getByName("Гор");

assertNotNull(actualBasePointDtoList);

assertEquals(expectBasePointList.size(), actualBasePointDtoList.size());

boolean result;

for (int i = 0; i < expectBasePointList.size(); i++) {

result = pointComparator.compareBasePointBasePointDTO(expectBasePointList.get(i), actualBasePointDtoList.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет работу метода getByName(), в том случае, когда база данных не содержит записей, соответствующих переданному шаблону.

@Test

void getByNameThrowsException() {

List<BasePoint> expectBasePointList = new ArrayList<>();

Mockito.when(basePointRepository.findByName("Гор")).thenReturn(expectBasePointList);

assertThrows(RuntimeException.class, () -> basePointService.getByName("Гор"));

}

// Следующий тест проверяет работу метода getBySheet(), в том случае, когда база данных содержит записи, соответствующие переданному шаблону.

@Test

void getBySheet() {

List<BasePoint> expectBasePointList = createDemoPoints();

Mockito.when(basePointRepository.findBySheet("L-37")).thenReturn(expectBasePointList);

List<BasePointDto> actualBasePointDtoList = basePointService.getBySheet("L-37");

assertNotNull(actualBasePointDtoList);

assertEquals(expectBasePointList.size(), actualBasePointDtoList.size());

boolean result;

for (int i = 0; i < expectBasePointList.size(); i++) {

result = pointComparator.compareBasePointBasePointDTO(expectBasePointList.get(i), actualBasePointDtoList.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет работу метода getBySheet(), в том случае, когда база данных не содержит записей, соответствующих переданному шаблону.

@Test

void getBySheetThrowsException() {

List<BasePoint> expectBasePointList = new ArrayList<>();

Mockito.when(basePointRepository.findBySheet("L-37")).thenReturn(expectBasePointList);

assertThrows(RuntimeException.class, () -> basePointService.getBySheet("L-37"));

}

// Следующий тест проверяет работу метода getByArea(), в том случае, когда база данных содержит геодезические пункты внутри описанной прямоугольной области.

@Test

void getByAreaTest() {

List<BasePoint> expectBasePointList = createDemoPoints();

AreaDto areaDto = new AreaDto();

areaDto.setX(0);

areaDto.setY(0);

areaDto.setAreaHeight(1000);

areaDto.setAreaWidth(1000);

Mockito.when(basePointRepository.findByArea(0, 0, 1000, 1000))

.thenReturn(expectBasePointList);

List<BasePointDto> actualBasePointDtoList = basePointService.getByArea(areaDto);

assertNotNull(actualBasePointDtoList);

assertEquals(expectBasePointList.size(), actualBasePointDtoList.size());

boolean result;

for (int i = 0; i < expectBasePointList.size(); i++) {

result = pointComparator.compareBasePointBasePointDTO(expectBasePointList.get(i), actualBasePointDtoList.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет работу метода getByArea(), в том случае, когда база данных не содержит геодезических пунктов внутри описанной прямоугольной области.

@Test

void getByAreaThrowException() {

List<BasePoint> expectBasePointList = new ArrayList<>();

AreaDto areaDto = new AreaDto();

areaDto.setX(0);

areaDto.setY(0);

areaDto.setAreaHeight(1000);

areaDto.setAreaWidth(1000);

Mockito.when(basePointRepository.findByArea(0, 0, 1000, 1000))

.thenReturn(expectBasePointList);

assertThrows(RuntimeException.class, () -> basePointService.getByArea(areaDto));

}

// Этот тест проверяет корректную работу сервиса при добавлении новой записи в базу данных.

@Test

void createBasePoint() {

BasePointDto expectBasePointDto = new BasePointDto();

expectBasePointDto.setName("1001");

expectBasePointDto.setX(999999999999L);

expectBasePointDto.setY(888888888888L);

expectBasePointDto.setZ(9000000000L);

expectBasePointDto.setSheet("L-37-32");

expectBasePointDto.setAccuracyClass("2кл");

expectBasePointDto.setCoordinateSystem("МСК-61-2");

BasePoint expectBasePoint = basePointMapper.toBasePoint(expectBasePointDto);

Mockito.when(basePointRepository.saveAndFlush(Mockito.any())).thenReturn(expectBasePoint);

BasePointDto actualBasePointDto = basePointService.createBasePoint(expectBasePointDto);

assertNotNull(actualBasePointDto);

boolean result = pointComparator.compareBasePointDtoBasePointDto(expectBasePointDto, actualBasePointDto);

assertTrue(result);

}

// Следующий тест проверяет корректную работу сервиса по удалению записи из базы данных в случае, если запись с указанным Id существует.

@Test

void removeById() {

BasePoint expectBasePoint = new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

);

Mockito.when(basePointRepository.findById(expectBasePoint.getId())).thenReturn(Optional.of(expectBasePoint));

BasePointDto actualBasePointDto = basePointService.removeById(expectBasePoint.getId());

assertNotNull(actualBasePointDto);

boolean result = pointComparator.compareBasePointBasePointDTO(expectBasePoint, actualBasePointDto);

assertTrue(result);

}

// Следующий тест проверяет корректную работу сервиса при попытке удалить запись с несуществующим Id.

@Test

void removeByIdThrowsException() {

Mockito.when(basePointRepository.findById(Mockito.any())).thenReturn(null);

assertThrows(RuntimeException.class, () -> basePointService.removeById(1));

}

// Следующий вспомогательный метод возвращающает список пунктов для использования в тестах. Внедрён в тестовый класс для избежания большого количества повторяющегося кода, в соответствии с принципом DRY.

private List<BasePoint> createDemoPoints() {

return List.of(

new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"),

new BasePoint(

"Горняк",

5212456400L,

7575789450L,

1122280L,

"L-37-10",

"3кл",

"1942г")

);

}

}

### 4. Сквозное тестирование приложения.

Этот вид тестирования проверяет работу приложения от начала до конца. Это позволит убедиться, все компоненты системы могут работать в реальных условиях. Для взаимодействия с приложением используем методы экземпляра класса WebTestClient

Для тестирования эндпоинтов приложения служит класс BasePointControllerTest:

import static org.junit.jupiter.api.Assertions.\*;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED\_PORT)

// Параметр аннотации определяет использование порта, указанного в файле application-test.yaml

@AutoConfigureWebTestClient

// эта аннотация привязывает WebTestClient непосредственно к приложению. В этом случае тесты не зависят от HTTP-сервера и используют фиктивные запросы и ответы.

@ActiveProfiles("test")

// эта аннотация определяет активный профиль при загрузке контекста.

class BasePointControllerTest {

@Autowired

private WebTestClient webTestClient;

@Autowired

private BasePointRepository basePointRepository;

@Autowired

BasePointMapper basePointMapper;

@Autowired

PointComparator pointComparator;

// Внедряем необходимые зависимости

@BeforeEach

public void setUp() {

basePointRepository.deleteAll();

}

// @BeforeEach определяет метод, который будет выполняться перед каждым тестом.

// Следующий тест проверяет эндпоинт «GET: /point» для получения всех записей из базы данных

@Test

void getAllBasePointsTest() {

List<BasePoint> expectedBasePointList = createDemoPoints(); basePointRepository.saveAll(expectedBasePointList);

List<BasePointDto> responseBody = webTestClient.get()

.uri("point")

.exchange()

.expectStatus().isOk()

.expectBodyList(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

assertEquals(expectedBasePointList.size(), responseBody.size());

boolean result;

for (int i = 0; i < expectedBasePointList.size(); i++) {

result = pointComparator.compareBasePointBasePointDTO(expectedBasePointList.get(i), responseBody.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет эндпоинт «GET: /point/{id}», в случае успешного обнаружения в базе данных записи с указанным id.

@Test

void getBasePointByIdTestSuccess() {

BasePoint expectBasePoint = basePointRepository.save(new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

));

BasePointDto responseBody = webTestClient.get()

.uri("point/" + expectBasePoint.getId())

.exchange()

.expectBody(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

boolean result = pointComparator.compareBasePointBasePointDTO(expectBasePoint, responseBody);

assertTrue(result);

}

// Следующий тест проверяет эндпоинт «GET: /point/{id}», в случае отсутствия в базе данных записи с указанным id.

@Test

void getBasePointByIdNotFound() {

basePointRepository.saveAll(createDemoPoints());

webTestClient.get()

.uri("point/4")

.exchange()

.expectStatus().isNotFound();

}

// Следующий тест проверяет эндпоинт «GET: /point/{name}», в случае успешного обнаружения в базе данных геодезических пунктов, название которых содержит фрагмент «name»

@Test

void getBasePointByNameTestSuccess() {

List<BasePoint> expectedBasePointList = createDemoPoints();

basePointRepository.saveAll(expectedBasePointList);

List<BasePointDto> responseBody = webTestClient.get()

.uri("point/name/Гор")

.exchange()

.expectStatus().isOk()

.expectBodyList(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

assertEquals(expectedBasePointList.size(), responseBody.size());

boolean result;

for (int i = 0; i < expectedBasePointList.size(); i++) {

result = pointComparator

.compareBasePointBasePointDTO(expectedBasePointList.get(i),

responseBody.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет эндпоинт «GET: /point/{name}», в случае отсутствия в базе данных геодезических пунктов, название которых содержит фрагмент «name»

@Test

void getBasePointByNameNotFound() {

basePointRepository.saveAll(createDemoPoints());

webTestClient.get()

.uri("point/name/1234")

.exchange()

.expectStatus().isNotFound();

}

// Следующий тест проверяет эндпоинт «GET: /point/{sheet}», в случае успешного обнаружения в базе данных геодезических пунктов, название листа M 1: 100 000 или объекта которых содержит фрагмент «sheet»

@Test

void getBasePointBySheetTestSuccess() {

List<BasePoint> expectedBasePointList = createDemoPoints();

basePointRepository.saveAll(expectedBasePointList);

List<BasePointDto> responseBody = webTestClient.get()

.uri("point/sheet/L-37")

.exchange()

.expectStatus().isOk()

.expectBodyList(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

assertEquals(expectedBasePointList.size(), responseBody.size());

boolean result;

for (int i = 0; i < expectedBasePointList.size(); i++) {

result = pointComparator.compareBasePointBasePointDTO(expectedBasePointList.get(i), responseBody.get(i));

assertTrue(result);

}

}

// Следующий тест проверяет эндпоинт «GET: /point/{sheet}», в случае отсутствия в базе данных геодезических пунктов, название листа M 1: 100 000 или объекта которых содержит фрагмент «sheet»

@Test

void getBasePointBySheetNotFound() {

basePointRepository.saveAll(createDemoPoints());

webTestClient.get()

.uri("point/sheet/1234")

.exchange()

.expectStatus().isNotFound();

}

// Следующий тест проверяет эндпоинт «DELETE: /point/{id}», в случае успешного обнаружения в базе данных записей с идентификатором id.

@Test

void removeBasePointByIdTest() {

BasePoint expectBasePoint = basePointRepository.save(new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

));

BasePointDto responseBody = webTestClient.delete()

.uri("point/" + expectBasePoint.getId())

.exchange()

.expectStatus().isOk()

.expectBody(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

boolean result = pointComparator.compareBasePointBasePointDTO(

expectBasePoint, responseBody

);

assertTrue(result);

}

// Следующий тест проверяет эндпоинт «POST: /point».

@Test

void createBasePointTest() {

BasePointDto expectBasePointDto = new BasePointDto();

expectBasePointDto.setName("1001");

expectBasePointDto.setX(999999999999L);

expectBasePointDto.setY(888888888888L);

expectBasePointDto.setZ(9000000000L);

expectBasePointDto.setSheet("L-37-32");

expectBasePointDto.setAccuracyClass("2кл");

expectBasePointDto.setCoordinateSystem("МСК-61-2");

BasePointDto responseBody = webTestClient.post()

.uri("point")

.contentType(MediaType.APPLICATION\_JSON)

.body(Mono.just(expectBasePointDto),BasePointDto.class)

.exchange()

.expectStatus().isCreated()

.expectBody(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

boolean result = pointComparator.compareBasePointDtoBasePointDto(

expectBasePointDto, responseBody);

assertTrue(result);

}

// Следующий тест проверяет эндпоинт «POST: /point/area}», в случае успешного обнаружения в базе данных геодезических пунктов внутри прямоугольной области, описанной в теле запроса.

@Test

void getByAreaTest() {

Random random = new Random();

List<BasePoint> expectedBasePointList = new ArrayList<>();

for (int i = 0; i < 10; i++) {

expectedBasePointList.add(new BasePoint(

"basePoint\_" + (i + 1),

random.nextInt(1000),

random.nextInt(1000),

random.nextInt(100),

"Empty",

"Empty",

"Empty"

));

basePointRepository.saveAll(expectedBasePointList);

AreaDto areaDto = new AreaDto();

areaDto.setX(0);

areaDto.setY(0);

areaDto.setAreaHeight(1000);

areaDto.setAreaWidth(1000);

List<BasePointDto> responseBody = webTestClient.post()

.uri("point/area")

.contentType(MediaType.APPLICATION\_JSON)

.body(Mono.just(areaDto), AreaDto.class)

.exchange()

.expectStatus().isOk()

.expectBodyList(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

assertEquals(expectedBasePointList.size(), responseBody.size());

boolean result;

for (int j = 0; j < expectedBasePointList.size(); j++) {

result = pointComparator.compareBasePointBasePointDTO(

expectedBasePointList.get(i),

responseBody.get(i));

assertTrue(result);

}

}

}

// Следующий тест проверяет эндпоинт «PUT: /point/{id}», в случае успешного обнаружения в базе данных записей с идентификатором id.

@Test

void updateBasePointTest() {

BasePoint expectBasePoint = basePointRepository.save(new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"

));

BasePointDto expectBasePointDto = new BasePointDto();

expectBasePointDto.setName("1001");

expectBasePointDto.setX(999999999999L);

expectBasePointDto.setY(888888888888L);

expectBasePointDto.setZ(9000000000L);

expectBasePointDto.setSheet("L-37-32");

expectBasePointDto.setAccuracyClass("2кл");

expectBasePointDto.setCoordinateSystem("МСК-61-2");

BasePointDto responseBody = webTestClient.put()

.uri("point/" + expectBasePoint.getId())

.contentType(MediaType.APPLICATION\_JSON)

.body(Mono.just(expectBasePointDto), BasePointDto.class)

.exchange()

.expectStatus().isOk()

.expectBody(BasePointDto.class)

.returnResult()

.getResponseBody();

assertNotNull(responseBody);

boolean result = pointComparator.compareBasePointDtoBasePointDto(

expectBasePointDto, responseBody);

assertTrue(result);

}

1. // Следующий вспомогательный метод служит для получения тестового списка экземпляров BasePoint.

private List<BasePoint> createDemoPoints() {

return List.of(

new BasePoint(

"Горная",

5212365400L,

7575821450L,

1159780L,

"L-37-09",

"3кл",

"1942г"),

new BasePoint(

"Горняк",

5212456400L,

7575789450L,

1122280L,

"L-37-10",

"3кл",

"1942г")

);

}

}

## Документация.

Для описания работы API в проект используется Swagger.

Swagger - это Open Source фреймворк для спецификации RESTful API.

Для внедрения Swagger в проект добавим зависимости в файл pom.xml:

<dependency>

<groupId>org.springdoc</groupId> <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.3.0</version>

</dependency>

<dependency>

<groupId>io.swagger.core.v3</groupId>

<artifactId>swagger-annotations</artifactId>

<version>2.1.6</version>

</dependency>

Для описания эндпоинтов нашего сервиса, и ответов сервиса добавим аннотации в класс BasePointController для каждого метода:

@RestController

@RequestMapping("point")

@RequiredArgsConstructor

public class BasePointController {

private final BasePointService basePointService;

@Operation(summary = "gets all the basic geodetic points", description =

"Загружает данные обо всех геодезических пунктах, хранящихся в базе данных")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Success"),

@ApiResponse(responseCode = "404", description = "Not found"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

@GetMapping

public ResponseEntity<List<BasePointDto>> getAllBasePoints() {}

Здесь:

@Operation: Описывает метод. Параметр summary представляет короткое описание, а description – развёрнутое описание.

@ApiResponses: Предоставляет массив описаний для ответов метода.

Для описания модели, в классы AreaDto и BasePointDto добавим аннотации Swagger для документирования входных и выходных данных.

Класс AreaDto:

@Data

@NoArgsConstructor

@Schema(name = "Прямоугольная область поиска")

public class AreaDto {

/\*\*

\* X coordinate of the southwest corner of the search area

\*/

@NotNull

@Schema(name = "Юго-западный угол X, мм.")

private long x;

Класс BasePointDto:

@Data

@NoArgsConstructor

@Schema(name = "Пункт геодезической сети")

public class BasePointDto {

private long id;

/\*\*

\* Points name

\*/

@NotNull

@Schema(name = "Название пункта")

private String name;

Здесь:

@Shema: Позволяет определять входные и выходные данные.

Внедрение Swagger в проект добавляет к проекту следующие эндпоинты:

localhost:8080/swagger-ui/index.html: графический интерфейс Swagger, который позволяет тестировать сервис и предоставляет документацию. Даёт возможность просмотреть какие типы запросов есть, описание моделей и их типов данных.

Localhost:8080/v3/api-docs: генерирует документацию на основе существующего кода, основываясь на Java Annotation.

Работа с Swagger представлена в приложении C.

# Заключение.

Таким образом, в результате вышеописанной работы в нашем распоряжении микросервис, который можно запустить одним из нижеописанных способов:

### Способ 1.

- Открываем терминал в папке geocatalog/target

- Запускаем контейнер с Postgres из командной строки:

docker run -d --name postgres-container -e POSTGRES\_DB=basepointsdb -e POSTGRES\_USER=andrew -e POSTGRES\_PASSWORD=1234 -p 5432:5432 postgres:alpine3.19

- Запускаем приложение из командной строки:

java -jar geocatalog.jar

Демонстрация работы с сервисом с помощью программы Postman продемонстрирована в приложении D.

# Приложения.

## Приложение А. Номенклатура топографических карт.

В основу номенклатуры топографических карт положена карта масштаба 1:1 000 000 (10 км в 1см; так называемые «миллионки»).

Вся поверхность Земли делится параллелями на ряды (через 4°), а меридианами — на колонны (через 6°); стороны образовавшихся трапеций служат границами листов карты масштаба 1:1000 000. Ряды обозначаются заглавными латинскими буквами от А до V, начиная от экватора к обоим полюсам, а колонны — арабскими цифрами, начиная от меридиана 180°с запада на восток. Номенклатура листа карты состоит из буквы ряда и номера колонны. Например, лист с городом Москва обозначается N-37.

Приполярные круглые области (с широтой свыше 88°) обозначаются буквой Z без указания номера колонны.

Карты масштаба 1:100 000 (1 км в 1 см) получаются делением листа миллионной карты на 144 части (рис. 3); их номенклатура состоит из обозначения листа карты масштаба 1:1000000 с добавлением одного из чисел 1, 2, 3, 4, …, 143, 144. Например, лист стотысячной карты с городом Рязань имеет номенклатуру N-37-56.

Во избежание путаницы с картами масштабов 1:500 000 и 1:200 000 (для которых существуют сходные альтернативные обозначения), карты этого масштаба могут обозначаться трёхзначными арабскими цифрами без потери начальных нулей, например: О-37-050.

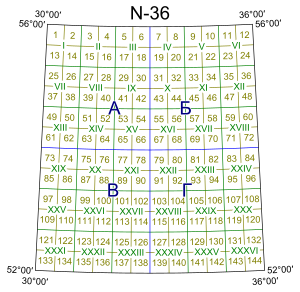


Рис. 3. Пример разграфки листа миллионной карты на карты масштаба 1:500 000 (синие), 1:200 000 (зелёные) и 1:100 000 (жёлтые)

## Приложение Б. Классификация геодезических сетей.

Все геодезические сети по назначению и точности построения подразделяются на три большие группы:

* государственные геодезические сети (ГГС),
* геодезические сети сгущения (ГСС),
* геодезические съемочные сети.

Государственные плановые сети имеют плотность в среднем 1 пункт на 5 — 15 км², высотные — 1 пункт на 5 — 7 км², дальнейшее сгущение геодезических сетей осуществляется путем создания сетей местного значения — сетей сгущения и съемочных сетей. Минимальное число пунктов съемочного обоснования для незастроенной территории 12 шт. на 1 км

У пунктов геодезической сети есть такая характеристика как класс точности.

В порядке убывания от более точных к менее точным:

* 1 класс
* 2 класс
* 3 класс
* 4 класс
* I разряд
* II разряд

С точки зрения геометрии любая геодезическая сеть – это группа зафиксированных на местности точек, для которых определены плановые координаты (X и Y или B и L) в принятой двухмерной системе координат и отметки H в принятой системе высот или три координаты X, Y и Z в принятой трехмерной системе пространственных координат.

По своей сущности геодезические пункты подразделяются на грунтовые и стенные. Грунтовые (закладываемые в грунт на незастроенной территории) состоят непосредственно из самого центра, являющегося носителем координат, наружного знака — обозначающего положение центра на местности и обеспечивающего взаимную видимость смежных пунктов сети и подземного сооружения (скрытая часть). Основная часть — Центр. Предназначен надежно и долговременно сохранять координаты пункта всех классов кроме I-го. Пункты I-го класса представляют группы центров, т. н. «кусты». Стенные г. п. закладываются в стены сооружений на застроенной территории и предназначены для сохранения координат III, IV классов, 1 и 2 разрядов. В отдельный класс выделяются пункты технических сетей, не имеющие капитального закрепления и наружного знака. В таких случаях применяются разборные знаки (переносимые или перевозимые), а сам пункт имеет характер временного (утрачивается в течение незначительного отрезка времени — 1 или 2 сезонов). Все виды геодезических пунктов не противопоставляются друг другу, а взаимно дополняют — чем выше класс сети, тем солидней конструкция и выше надежность центра.



Рис. Наружный знак геодезического пункта

# Список использованной литературы.

1. Строительные нормы и правила
2. «Java полное руководство» Герберт Шилдт
3. «Разработка веб-приложений на платформе Spring» Суханов В.И.
4. «Чистая архитектура. Искусство разработки программного обеспечения.» Роберт Мартин.