Оглавление

[Теория 2](#__RefHeading___Toc4608_950171436)

[Типы JSON в PostgreSQL 2](#__RefHeading___Toc4610_950171436)

[Подготовка проекта 4](#__RefHeading___Toc4612_950171436)

[Зависимости 4](#__RefHeading___Toc4638_950171436)

[Подготовка базы данных 4](#__RefHeading___Toc4614_950171436)

[Скрипт миграции 4](#__RefHeading___Toc4616_950171436)

[Сущность «Post» 4](#__RefHeading___Toc4618_950171436)

[DAO 5](#__RefHeading___Toc4620_950171436)

[Свойства файлов 5](#__RefHeading___Toc4622_950171436)

[file\_config.properties 5](#__RefHeading___Toc4624_950171436)

[Конфигурация 5](#__RefHeading___Toc4626_950171436)

[FileConfig 5](#__RefHeading___Toc4628_950171436)

[Сервисный слой 6](#__RefHeading___Toc4630_950171436)

[PostImageService 6](#__RefHeading___Toc4632_950171436)

[PostService 6](#__RefHeading___Toc4634_950171436)

[Data Transfer Objects (DTO) 7](#__RefHeading___Toc5222_950171436)

[FileUploadResponse 7](#__RefHeading___Toc5224_950171436)

[PostBody 7](#__RefHeading___Toc5226_950171436)

[Blocks 8](#__RefHeading___Toc5228_950171436)

[Контроллеры 8](#__RefHeading___Toc5230_950171436)

[FileRestController 8](#__RefHeading___Toc5232_950171436)

[PostRestController 8](#__RefHeading___Toc5234_950171436)

[Postman 9](#__RefHeading___Toc5236_950171436)

[Загрузка изображения 9](#__RefHeading___Toc5238_950171436)

[Получение изображения 10](#__RefHeading___Toc5240_950171436)

[Создание поста 11](#__RefHeading___Toc5242_950171436)

[Просмотр постов пользователя 11](#__RefHeading___Toc5244_950171436)

[Лабораторная работа №5 13](#__RefHeading___Toc5246_950171436)

## 6 Типы JSON в PostgreSQL

Типы JSON предназначены для хранения данных JSON (JavaScript Object Notation, Запись объекта JavaScript) согласно стандарту [RFC 7159](https://tools.ietf.org/html/rfc7159). Такие данные можно хранить и в типе text, но типы JSON лучше тем, что проверяют, соответствует ли вводимое значение формату JSON. Для работы с ними есть также несколько специальных функций и операторов;

Существуют два типа данных JSON: json и jsonb. Они принимают на вход почти одинаковые наборы значений, а отличаются главным образом с точки зрения эффективности. Тип json сохраняет точную копию введённого текста, которую функции обработки должны разбирать заново при каждом выполнении запроса, тогда как данные jsonb сохраняются в разобранном двоичном формате, что несколько замедляет ввод из-за преобразования, но значительно ускоряет обработку, не требуя многократного разбора текста. Кроме того, jsonb поддерживает индексацию, что тоже может быть очень полезно.

Так как тип json сохраняет точную копию введённого текста, он сохраняет семантически незначащие пробелы между элементами, а также порядок ключей в JSON-объектах. И если JSON-объект внутри содержит повторяющиеся ключи, этот тип сохранит все пары ключ/значение. (Функции обработки будут считать действительной последнюю пару.) Тип jsonb, напротив, не сохраняет пробелы, порядок ключей и значения с дублирующимися ключами. Если во входных данных оказываются дублирующиеся ключи, сохраняется только последнее значение.

Для большинства приложений предпочтительнее хранить данные JSON в типе jsonb (если нет особых противопоказаний, например важны прежние предположения о порядке ключей объектов).

PostgreSQL позволяет использовать только одну кодировку символов в базе данных, поэтому данные JSON не будут полностью соответствовать спецификации, если кодировка базы данных не UTF-8. При этом нельзя будет вставить символы, непредставимые в кодировке сервера, и наоборот, допустимыми будут символы, представимые в кодировке сервера, но не в UTF-8.

RFC 7159 разрешает включать в строки JSON спецпоследовательности Unicode в виде \uXXXX. В функцию ввода для типа json эти спецпоследовательности допускаются вне зависимости от кодировки базы данных, и проверяется только правильность их синтаксиса (за \u должны следовать четыре шестнадцатеричных цифры). Однако функция ввода для типа jsonb более строгая: она не допускает спецпоследовательности Unicode для не ASCII-символов (символов после U+007F), если кодировка базы данных не UTF8. Тип jsonb также не принимает \u0000 (так как это значение не может быть представлено в типе text PostgreSQL) и требует, чтобы суррогатные пары Unicode использовались для представления символов вне основной многоязыковой плоскости (BMP) правильно. Корректные спецпоследовательности Unicode преобразуются для хранения в соответствующий символ ASCII или UTF8 (это подразумевает сворачивание суррогатных пар в один символ) (рис. 6.1).

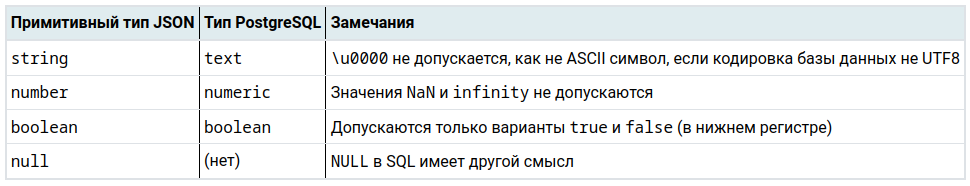


Рис. 6.1 - Примитивные типы JSON и соответствующие им типы PostgreSQL

## Загрузка файлов.

Для получения файла в запросе контроллера используется объект MultipartFile.

Тип содержимого **multipart/form-data** — это составной тип содержимого, чаще всего использующийся для отправки HTML-форм с бинарными (не-ASCII) данными методом POST протокола HTTP. Указывается в поле заголовка Content-Type (тип содержимого) и следует правилам для составных MIME-данных в соответствии с RFC 2045.

# 6.1 Подготовка проекта

## 6.1.2. Зависимости

Чтобы правильно работать с этим типом данных, нужно добавить следующую зависимость:

<dependency>

<groupId>com.vladmihalcea</groupId>

<artifactId>hibernate-types-52</artifactId>

<version>2.3.4</version>

</dependency>

## 

## 6.1.2 Подготовка базы данных

## Скрипт миграции:

CREATE TABLE if not exists posts  
(  
 id UUID NOT NULL,  
 title VARCHAR(50) NOT NULL,  
 author\_id UUID NOT NULL,  
 body JSONB,  
 created\_at TIMESTAMP WITHOUT TIME ZONE,  
 CONSTRAINT pk\_posts PRIMARY KEY (id)  
);  
  
ALTER TABLE posts  
 ADD CONSTRAINT FK\_POSTS\_ON\_AUTHOR FOREIGN KEY (author\_id) REFERENCES users (id);

## Сущность «Post»:

@Getter  
@Setter  
@Entity(name = "post")  
@Table(name = "posts")  
@TypeDef(name = "jsonb", typeClass = JsonType.class)  
public class Post implements Serializable{  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private UUID id;  
  
 @Column(nullable = false, length = 50)  
 private String title;  
  
 @JoinColumn(name = "author\_id", nullable = false)  
 private UUID authorId;  
  
 @Type(type = "jsonb")  
 @Column(columnDefinition = "jsonb", name = "body")  
 private List<PostBody> body;  
  
 @CreationTimestamp  
 private Date createdAt;  
}

## DAO:

public interface PostRepository extends JpaRepository<Post, UUID> {

List<Post> findAllByAuthorId(UUID id);

}

## 6.1.3 Свойства файлов

## file\_config.properties:

images.path.storeDir = .../src/main/resources

Свойство *images.path.storeDir*хранит абсолютный путь к директории, в которой будет храниться директория *images* для хранения изображений. Сохранить файл *file\_config.properties* рядом с *application.properties*.

## 6.1.4 Конфигурация

## FileConfig:

@PropertySource("classpath:file\_config.properties")

@Slf4j

@Configuration

public class FileConfig {

@Value(value = "${images.path.storeDir}")

private String filesDirPath;

@Bean

public File imagesDir() {

File file = new File(filesDirPath+"/images/");

file.mkdir();

return file;

}

}

@PropertySource — указывает на файл со свойствами, из которых будет парситься нужное нам значение для строки *fileDirPath*.

## 6.1.5 Сервисный слой

## PostImageService:

@Service

public class PostImageService {

private final File imagesDir;

@Autowired

public PostImageService(File imagesDir) {

this.imagesDir = imagesDir;

}

public byte[] download(String url){

try {

FileInputStream fileInputStream = new FileInputStream(url);

return fileInputStream.readAllBytes();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

public FileUploadResponse upload(MultipartFile file){

try {

return new FileUploadResponse(createFile(file));

} catch (IOException e) {

throw new RuntimeException(e);

}

}

private String createFile(MultipartFile file) throws IOException {

String filePath = imagesDir.getPath()

+ '/'

+ generateFileName(file.getOriginalFilename());

FileOutputStream fileOutputStream = new FileOutputStream(filePath,true);

fileOutputStream.write(file.getBytes());

fileOutputStream.close();

return filePath;

}

private String generateFileName(String fileName){

int indexOfLastDot = fileName.lastIndexOf('.');

String newName = fileName.substring(0,indexOfLastDot);

return newName + new SimpleDateFormat("dd:MM:yyyy-hh:mm:ss").format(new Date()) + fileName.substring(indexOfLastDot);

}

}

## PostService:

@Service

public class PostService {

private final PostRepository repository;

private final UserService userService;

@Autowired

public PostService(PostRepository repository,

UserService userService){

this.repository = repository;

this.userService = userService;

}

public Post create(Blocks blocks){

Post post = new Post();

post.setTitle(blocks.getTitle());

post.setCreatedAt(new Date());

post.setAuthorId(userService

.getByUsername(

SecurityContextHolder

.getContext()

.getAuthentication()

.getName())

.getId()

);

post.setBody(blocks.getBlocks());

return repository.save(post);

}

public List<Post> getAllByAuthorId(UUID id){

return repository.findAllByAuthorId(id);

}

}

## 6.1.6 Data Transfer Objects (DTO)

## FileUploadResponse:

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class FileUploadResponse {

private String url;

}

Является оберткой для ссылки на загруженное изображение. Требуется для ответа на запрос загрузки изображения

## PostBody:

@Getter

@Setter

public class PostBody implements Serializable {

private String type;

private String data;

}

Хранит тип и данные для одного блока в посте.

## Blocks:

@Getter

@Setter

public class Blocks implements Serializable {

private String title;

private List<PostBody> blocks;

}

## 6.1.7 Контроллеры

## FileRestController:

@RestController

@RequestMapping("/posts/images/")

public class FileRestController {

private final PostImageService postImageService;

public FileRestController(PostImageService postImageService) {

this.postImageService = postImageService;

} @PostMapping

public FileUploadResponse create(@RequestParam("img")

MultipartFile multipartFile)

throws InvalidContentTypeException {

if(!multipartFile.getContentType().contains("image")){

throw new InvalidContentTypeException("File isn't img");

}

return postImageService.upload(multipartFile);

}

@GetMapping

@RequestMapping

public byte[] download(@RequestParam("img\_path") String imgPath){

return postImageService.download(imgPath);

}

}

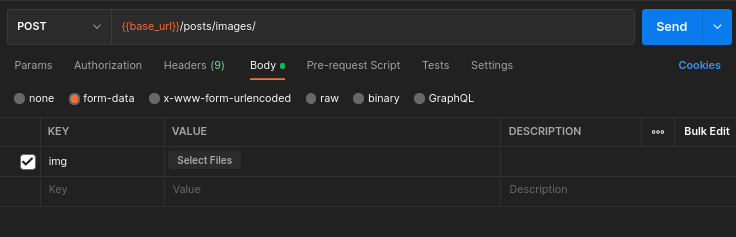
## PostRestController:

@RestController  
public class PostRestController  
{  
 private final PostService postService;  
 @Autowired  
 public PostRestController(PostService postService) {  
 this.postService = postService;  
 }  
 @PostMapping  
 @RequestMapping("/posts/")  
 public Post create(@RequestBody Blocks blocks){  
 return postService.create(blocks);  
 }  
 @GetMapping  
 @RequestMapping("/users/{id}/posts/")  
 public List<Post> getAllByUserId(@PathVariable("id") UUID id){  
 return postService.getAllByAuthorId(id);  
 }  
}

## 6.1.8 Postman

## Загрузка изображения:

1. Создать коллекцию Post.
2. Создать в колекции папку img.
3. Создать запрос с именем create(рис. 6.2).
4. Создать запрос с именем download.

Рис. 6.2 — Post запрос для загрузки изображения с именем «create»

1. Ввести адрес маршрута как на рис. 6.2.
2. Установить метод запроса POST.
3. Для загрузки изображения перейти в Body → form-data.
4. В поле столбца key у правого края поменять значение text на file.
5. Дать имя «img» параметру.
6. Выбрать изображение для загрузки.
7. В результате выполнения запроса должны получить ответ как на рис. 6.3.

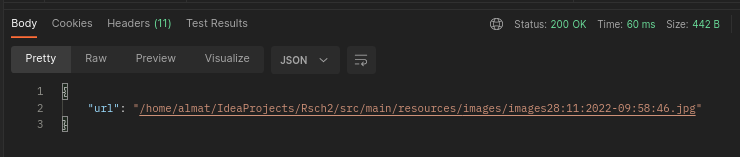


Рис. 6.3 — Ответ на запрос «create»

Получение изображения. Чтобы получить изображение в виде массива byte опишите следующий запрос (рис. 6.4):

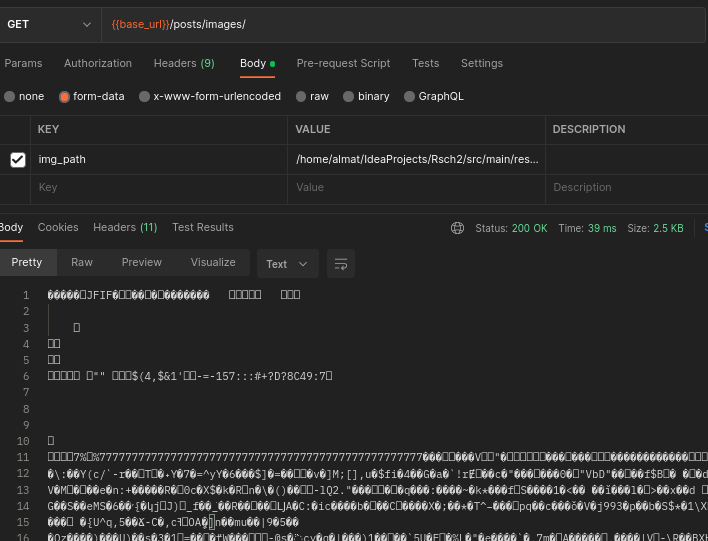


Рис. 6.4 — Запрос и ответ на скачивание изображения

1. Установить метод запроса GET.
2. Указать в качестве адреса маршрут как на рис. 6.3.
3. Добавить параметр «img\_path», значением которого служит ссылка на изображение, которое мы получили в результате запроса «create».
4. В качестве ответа получаем массив байтов изображения.

### Создание поста (рис. 6.5):

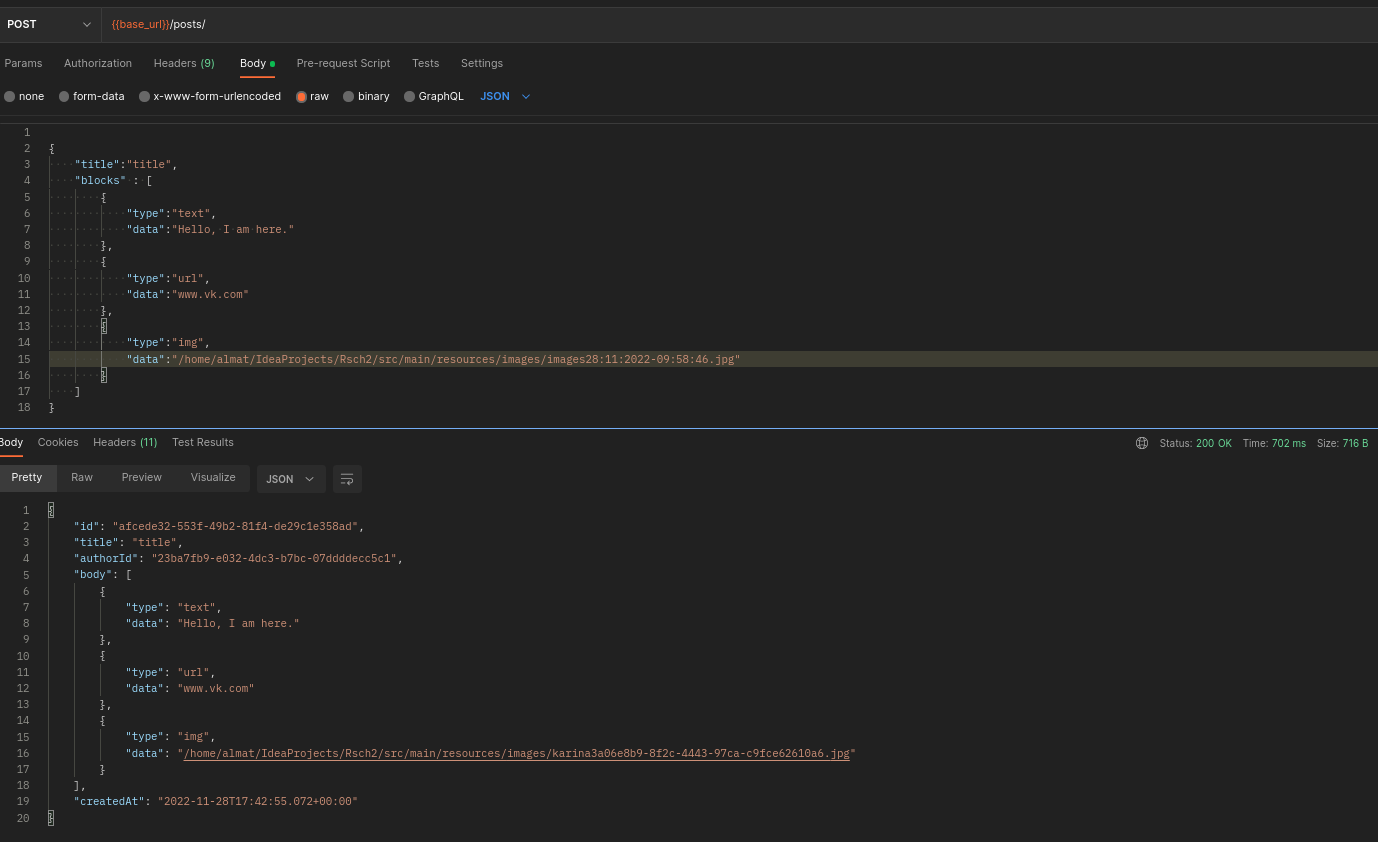


Рис. 6.5 — Запрос и ответ создания поста

1. Внутри коллекции post следует создать запрос «create».
2. В теле запроса описать наименование поста и блоки различных типов.
3. Ссылку на изображение, которое мы получаем в результате загрузки, мы вписываем в блок с типом «img» в поле «data».
4. Также можно добавить поле «url», которое будет помечать то, что этот блок является ссылкой.

### Просмотр постов пользователя (рис. 6.6):

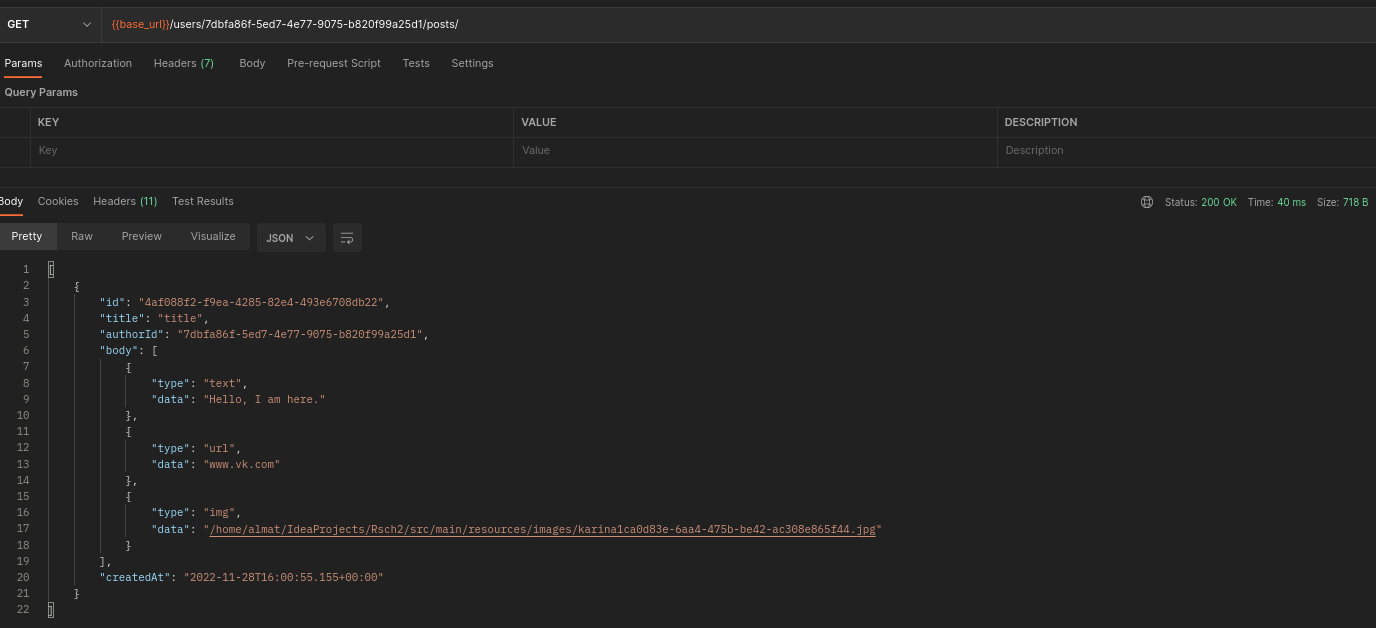
****

Рис. 6.6 — запрос и ответ на получение постов пользователя с указанием его id

1. Создать запрос «get by user id»в колекции post.
2. Установить метод GET.
3. Указать адрес как на рис. 6.5.
4. После «users/» в адресе следует указать id целевого пользователя.
5. В ответе получаем список постов данного пользователя.

# 6.2 Лабораторная работа №5 - Лента постов. Загрузка файлов

Цель лабораторной работы: научиться работать JSON в PostgreSQL, добавлять ленту постов с наименованиями, текстом, ссылками и изображениями.

Задание:

1. Реализовать ленту постов с добавление наименования, текста, ссылки и изображения.
2. Реализовать хранение json в PostgreSQL.
3. Реализовать загрузку изображений.

Для создания карточки поста предварительно необходимо загружать изображение, затем, используя ссылку на данное изображение, создать пост.

В массиве blocks указать список блоков поста. Каждый блок является объектом PostBody.

Контрольные вопросы:

Вопросы к защите лабораторной работы включают процесс установки и настройки по каждому пункту задания, а также теоретический материал к данной работе.