МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕНЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.Шухова»
(БГТУ им. В.Г.Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

КУРСОВАЯ РАБОТА

По дисциплине: Интерфейсы вычислительных систем

Тема: “Разработка клиент-серверного приложения.”

Выполнил: Руденький А.О.
Проверил: Торопчин Д.А.

Белгород 2020

Содержание

Введение

Сегодня все крупные и успешные предприятия перестают использовать нативные решения различных прикладных задач и активно переходят на использование веб-приложений, которые в последнее время всё чаще реализуются по принципу микросервисной архитектуры. Плюсы и минусы, есть как у нативного ПО, так и у современных веб-приложений, но в любом случае они как правило работают не автономно, а в связке с сервером, который производит большую часть обработки данных на себе и является наиболее важным звеном работы всей платформы.

Реализовать сервер можно на огромном количестве языков программирования, например, сервер можно написать на ЯП Golang, Python, Java, NodeJs, C++ и т.д., но чаще всего не просто с использованием элементарных объектов tcp взаимодействия, а с помощью специальных фрэймворков, которые позволяют избавится от boilerplate кода и бесконечных велосипедов с костылями.

В курсовой работе мы подробно рассмотрим тему разработки веб-приложения, на примере приложения “облачного файлового хранилища”, для этого мы рассмотрим необходимый для этого теоретический материал, создадим сервер на языке программирования java с использованием фрэймворка spring, клиентскую сторону реализуем с использованием ReactJS и в результате работы сделаем вывод о полученных результатах.

Глава 1. Основная часть

1.1. Java

Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems. Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины.

1.2. MVC

Model-View-Controller (Модель-Представление-Контроллер) — это подход разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

- ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
- ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

1.3. Spring

Spring — это простая в использовании среда Java MVC с открытым исходным кодом, обеспечивающая всестороннюю поддержку для простой и быстрой разработки приложений Java.

Spring — это облегченный фреймворк, который можно рассматривать как фреймворк фреймворков, поскольку он предлагает поддержку различных фреймворков, таких как Hibernate, Struts, Tapestry, JSF и т. д.

Стоит отметить, что наиболее важной особенностью Spring Framework, является Dependency Injection.

Spring Framework предоставляет 20 модулей, которые можно использовать в зависимости от требований приложения.

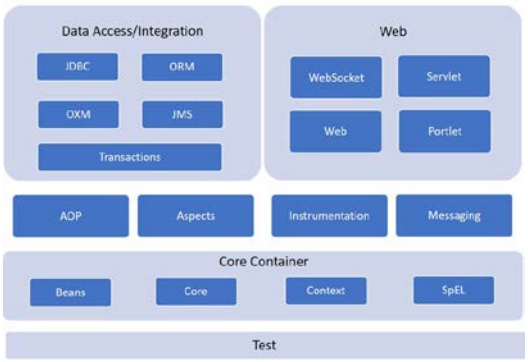

Рис. 1.1. Модули Spring Framework.

Core и Bean обеспечивают фундаментальную часть платформы, включая IoC и DI.

1.4. БД, ORM и Hibernate

Ни одно современное веб-приложение не обходится без хранения огромного количества различной информации. Эту задачу обычно возлагают на специальные программы — Систему Управления Базами Данных СУБД. По схеме организации базы данных делятся на несколько видов, но самым распространенным видом оказались реляционные.

В реляционных базах, данные организованны в виде сущностный (таблиц) и связей между ними. Программисты, работающие с объектно-ориентированными языками программирования, зачастую сталкиваются с задачей преобразования данных из формы, понятной СУБД в привычную объектную форму и решением этой проблемы является технология Object-Relational Mapping (ORM), которую реализует популярная библиотека Hibernate. Hibernate берет на себя задачу преобразования данных их реляционного вида в объектный, для чтения, и из объектного вида в реляционный — для записи. Кроме того, библиотека позволяет легко настроить подключение к СУБД и с помощью нее очень легко управлять транзакциями.

Глава 2. Разработка веб-приложения

2.1. Проектирование БД

В качестве база данных мы выберем объектно-реляционную систему управления базами данных PostgreSQL.

Для взаимодействия сервера и БД используем библиотеку Hibernate реализующую ORM, а также паттерн “репозиторий” и Data Transfer Object (DTO).

Следует отметить, что Data Transfer Object (DTO) — один из шаблонов проектирования, используется для передачи данных между подсистемами приложения. Data Transfer Object, в отличие от business object или data access object не должен содержать какого-либо поведения.

Для работы нашего облачного хранилища файлов выделим следующие наиболее значимые объекты:

- Клиент
- Файловое хранилище
- Промо-код
- Сессия
- Расшаренный файл

Определим следующую диаграмму базы данных:

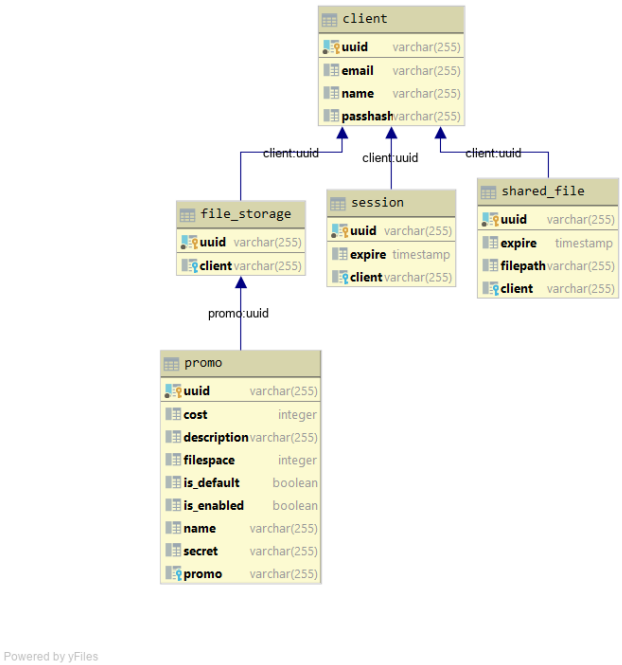

Рис. 2.1. Диаграммы базы данных

DDL описание сущностей:

- Для client:

```
create table client
(
        uuid varchar(255) not null
                constraint client_pkey
                        primary key,
        email varchar(255),
        name varchar(255),
        passhash varchar(255)
);
```

- Для file_storage:

```
create table file_storage
(
        uuid varchar(255) not null
                constraint file_storage_pkey
                        primary key,
        client varchar(255)
                constraint fk1p6tccobbljm79sltymiils0h
                        references client
);
```

- Для promo:

```
create table promo
(
        uuid varchar(255) not null
                constraint promo_pkey
                        primary key,
        cost integer,
        description varchar(255),
        filespace integer,
        is_default boolean,
        is_enabled boolean,
        name varchar(255),
        secret varchar(255),
        promo varchar(255)
                constraint fkjo2f4kp53hq1qgnrimbh26w0a
                        references file_storage
);
```

- Для session:

```
create table session
(
	uuid varchar(255) not null
		constraint session_pkey
			primary key,
	expire timestamp,
	client varchar(255)
		constraint fkljne74ivrs5a4bbyym1lvbuta
			references client
);
```

- Для shared_file:

```
create table shared_file
(
	uuid varchar(255) not null
		constraint shared_file_pkey
			primary key,
	expire timestamp,
	filepath varchar(255),
	client varchar(255)
		constraint shared_file_client_uuid_fk
			references client
);
```

2.2. Описание api функций

Выделим следующие api функции которые будут предоставлять java spring сервер:

- **Регистрация пользователя в системе.**

  Сигнатура: /api/v1/client/register

  Входные данные:

  - Логин
  - Почта
  - Пароль
  - Промо-код

  Выходные данные:

  - Токен
  - Логин

- **Авторизация пользователя в системе.**

  Сигнатура: /api/v1/client/login

  Входные данные:

  - Логин
  - Пароль
  - Запомнить (да/нет)

  Выходные данные:

  - Токен
  - Логин

- **Получение содержимого облачной папки пользователя по относительному пути /****

  Сигнатура: /api/v1/mycloud/**

  Входные данные:

  - Токен

  Выходные данные:

  - Название файла
  - Путь к файлу
  - Является ли папкой (да/нет)

- **Загрузка файла в облачную папку по относительному пути /****

  Сигнатура: /api/v1/upload/mycloud/**

  Входные данные:

  - Токен
  - Файл

- **Скачивание файла с облачной папки по относительному пути /****

  Сигнатура: /api/v1/download/mycloud/**

  Входные данные:

  - Токен

  Выходные данные:

  - Файл

- **Создание подпапки в облачной папке по относительному пути /****

  Сигнатура: /api/v1/mkdir/mycloud/**

  Входные данные:

  - Токен

- **Удаление файла/подпапки в облачной папке по относительному пути /****

  Сигнатура: /api/v1/delete/mycloud/**

  Входные данные:

  - Токен

- **Проверка актуальности токена.**

  Сигнатура: /api/v1/session/checkToken

  Входные данные:

  - Токен

  Выходные данные:

  - Истинна в случае актуальности переданного токена.

2.3. Реализация документирования RestApi посредством SpringDoc

В данной реализации сервера, для документации использовали инструмент автодокументации restapi функций SpringDoc, которые мы в процессе описания контроллеров обозначали специальными аннотациями SpringDoc. Следует отметить, что SpringDoc это инструмент, который упрощает создание и обслуживание документов API на основе спецификации OpenAPI 3.

После того как мы запустим сервер, наша документация будет доступна по следующему адресу:

http://localhost:8181/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config

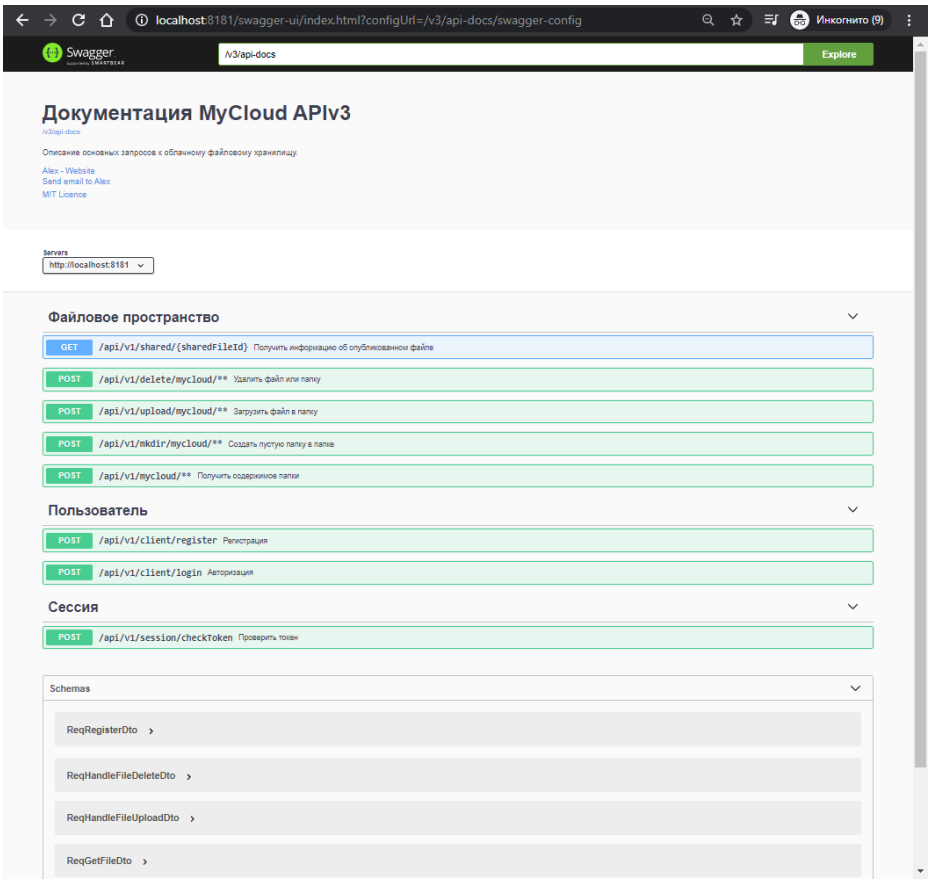

Рис 2.2. Документация MyCloud RestAPI.

Рис 2.3. Пример описания структуры dto.

Скриншоты

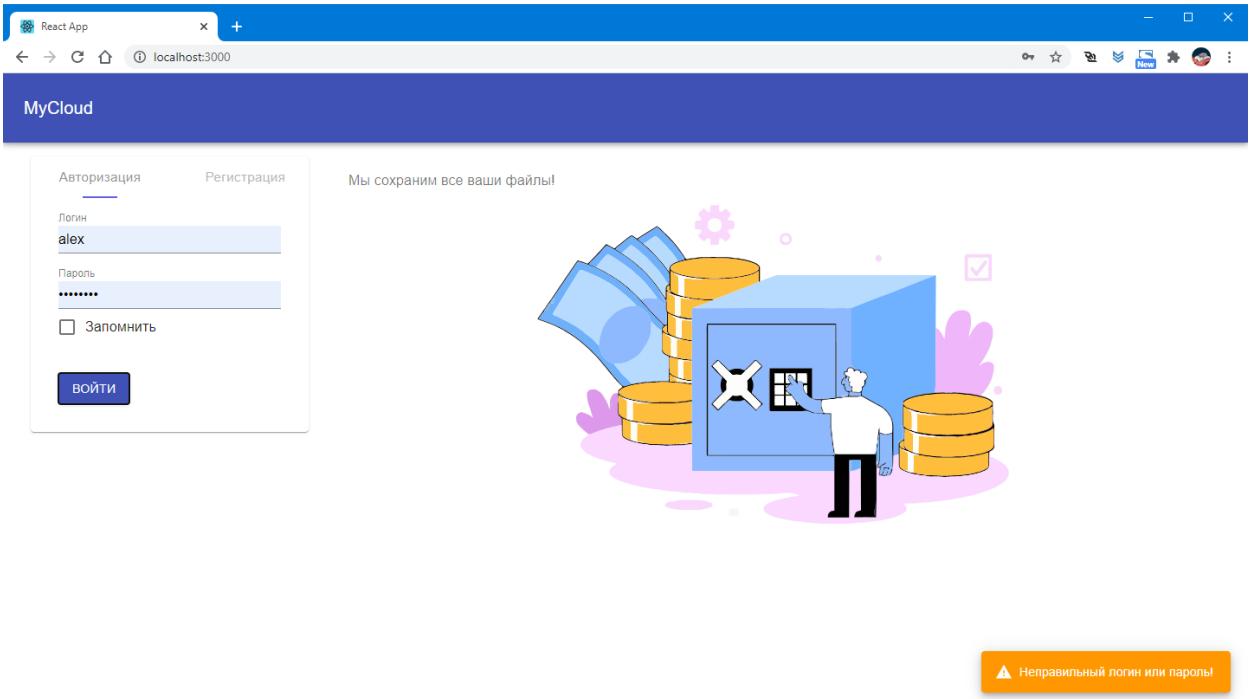

Рис 3.1. Скриншот сайта. Форма авторизации.

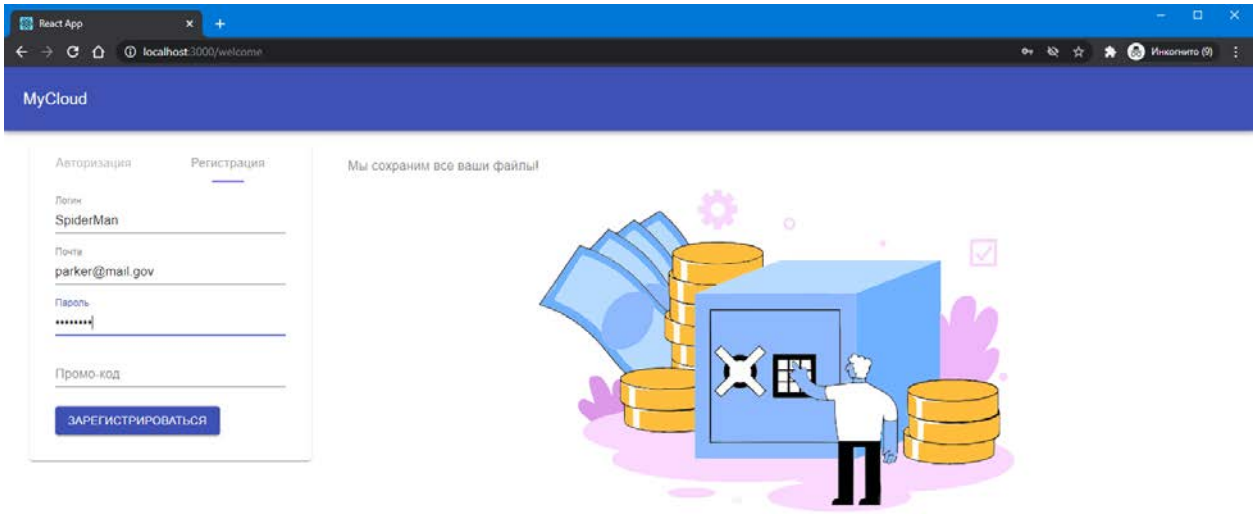

Рис 3.2. Скриншот сайта. Форма регистрации.

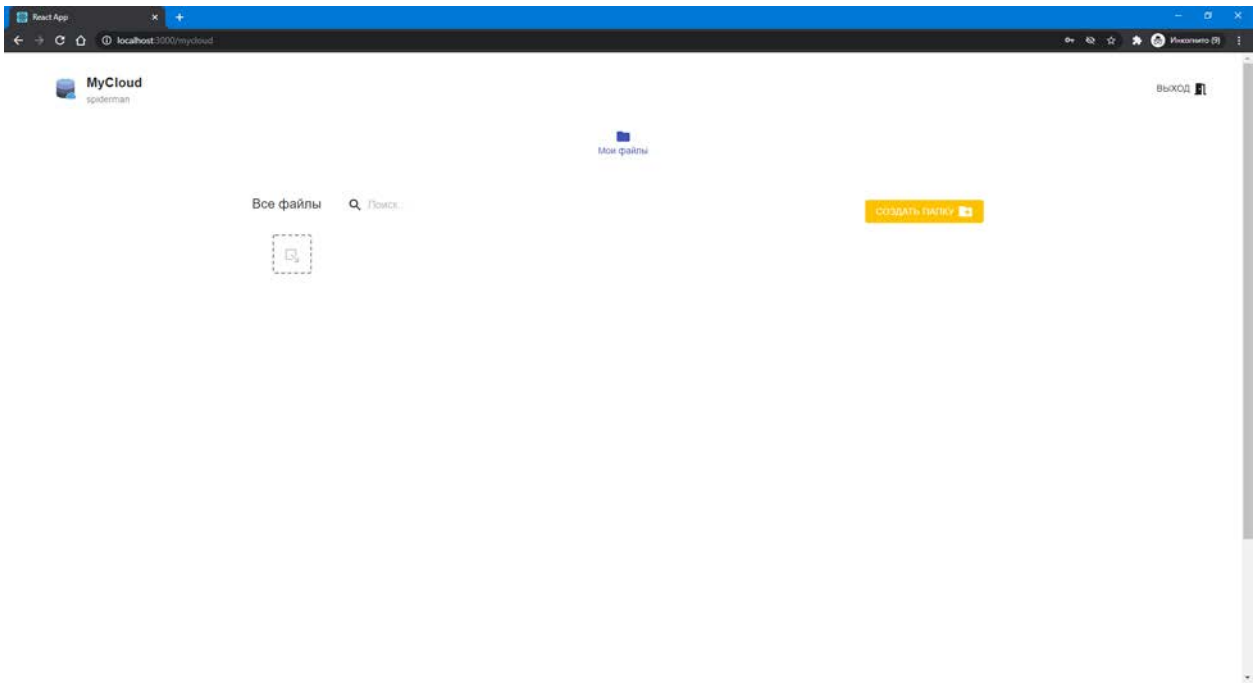

Рис 3.3. Скриншот сайта. Главная форма.

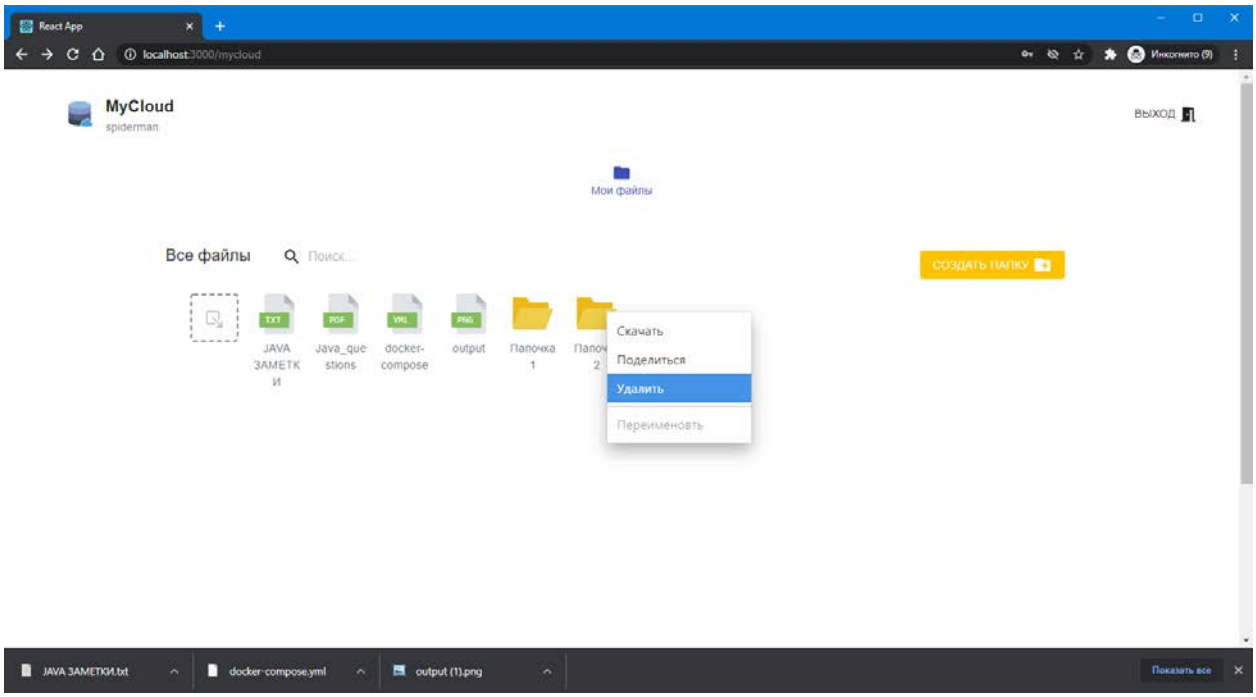

Рис 3.4. Скриншот сайта. Главная форма с загруженными файлами и открытым выпадающим меню.

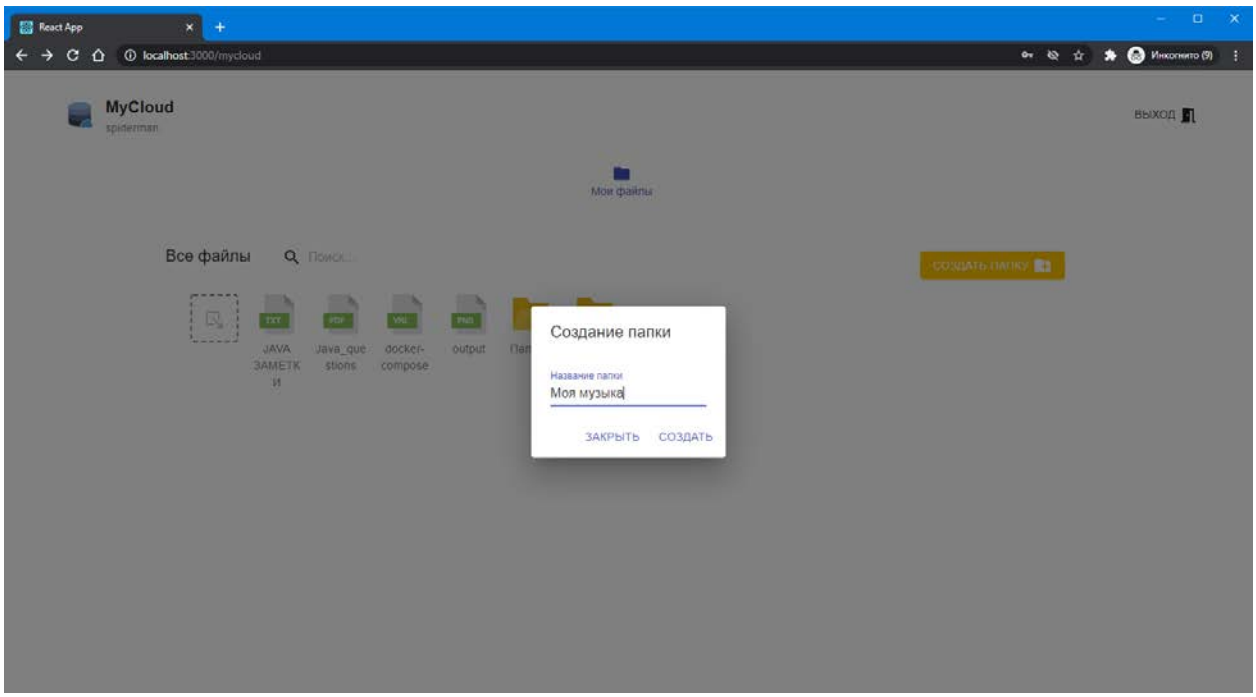

Рис 3.5. Скриншот сайта. Главная форма с модальным окном, отвечающим за создание папки.

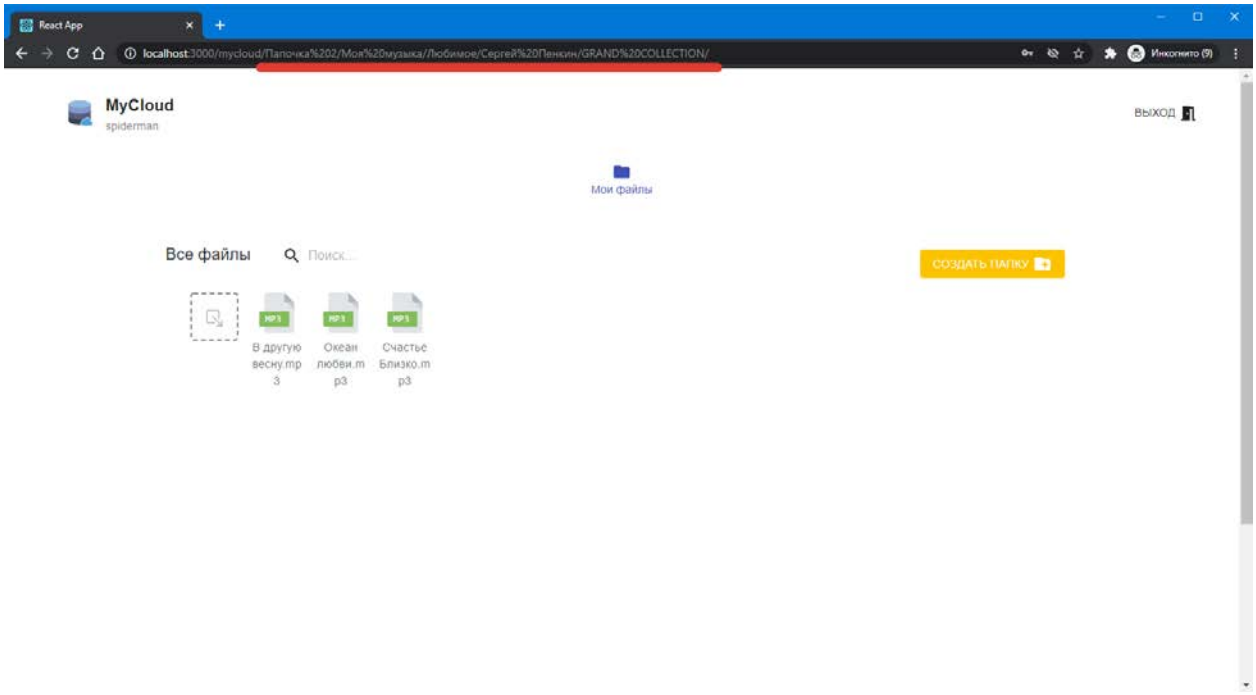

Рис 3.6. Скриншот сайта. Демонстрация маршрута, сформированного при переходе в подпапки.

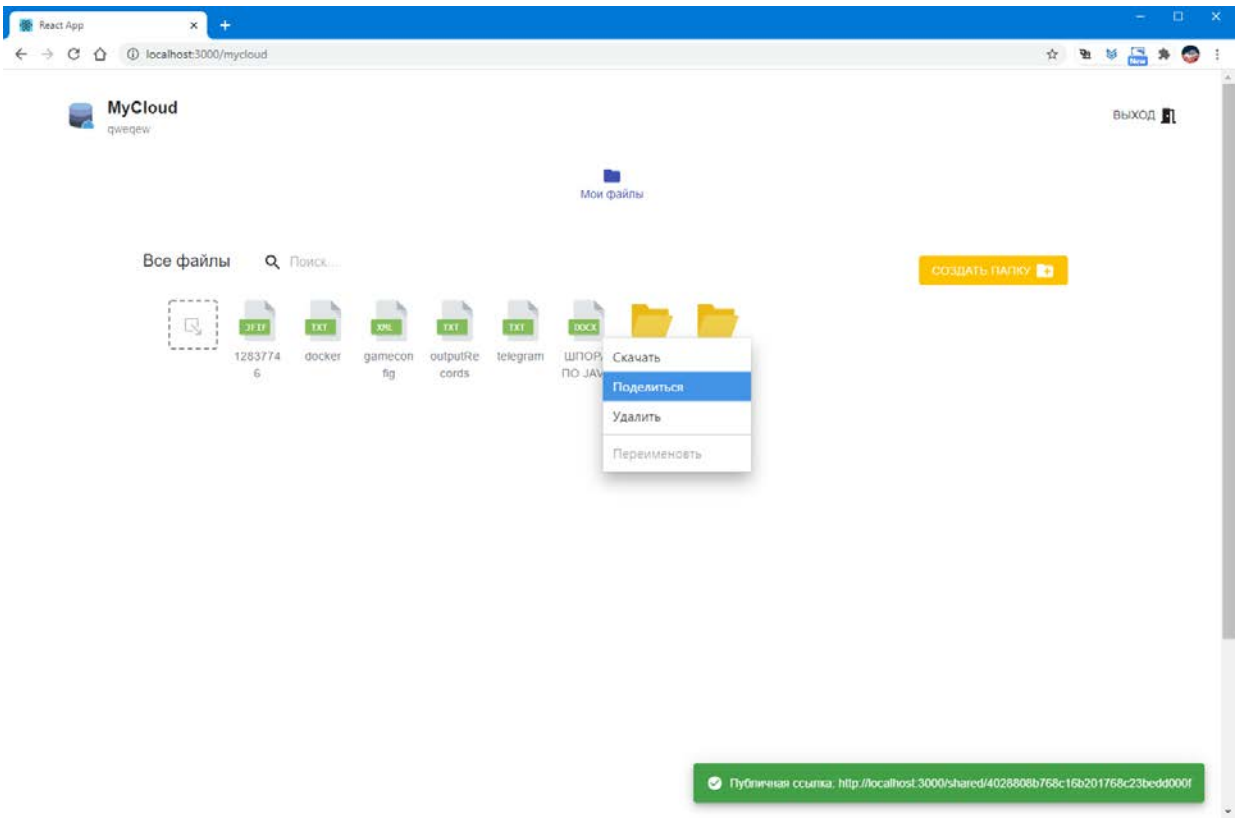

Рис 3.7. Скриншот сайта. Публикация файла в открытый доступ и получение её ссылки.

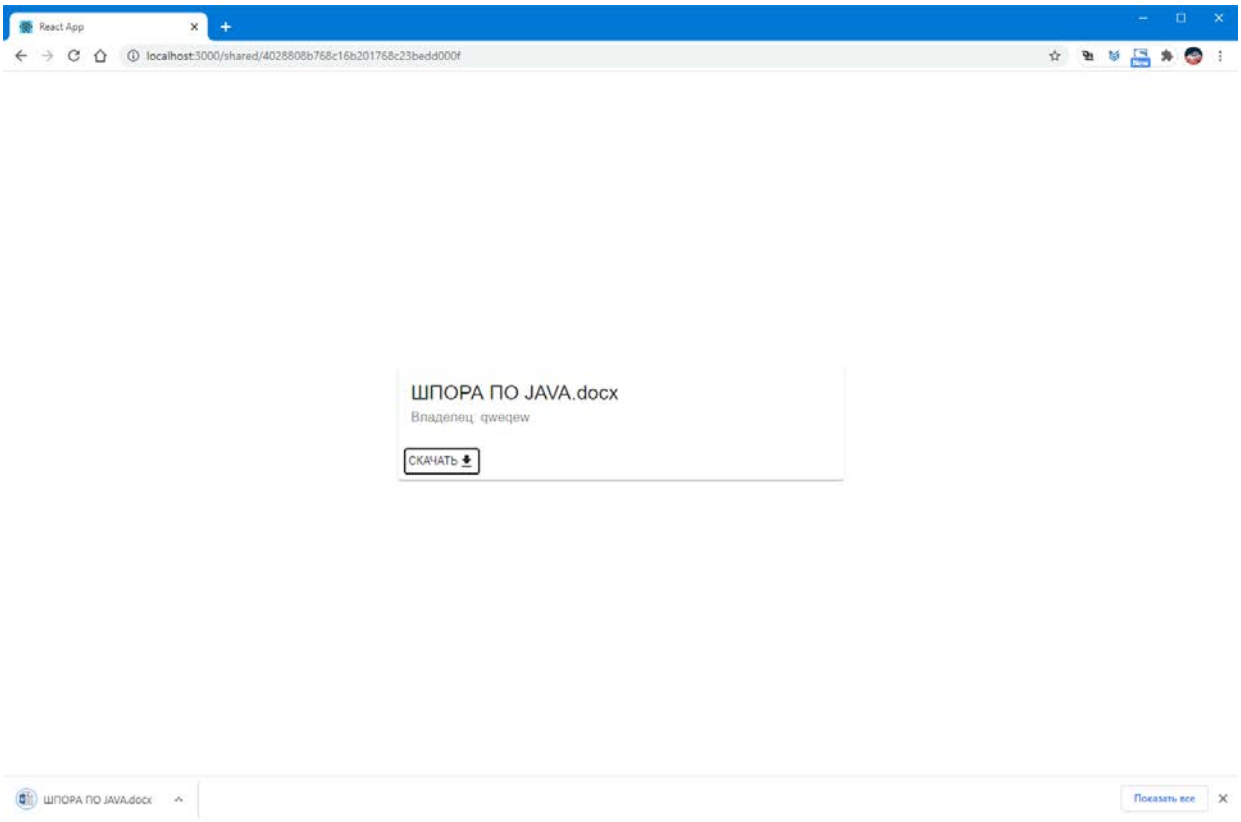

Рис 3.8. Скриншот сайта. Скачивание файла с облака по публичной ссылке.

Заключение

В данной работе мы познакомились с клиент-серверной архитектурой, определили актуальность данной технологии, после чего рассмотрели кратко теоретический материал и разработали веб-приложение, которое было создано по этому принципу. В частности, мы разработали клиентскую часть на ReactJS и серверную на Java Spring, которые взаимодействуют между собой посредством архитектурного стиля взаимодействия компонентов распределённого приложения в сети RestAPI.

Процесс разработки мы начали с описания необходимых спецификаций моделей сущностей, обозначения наиболее необходимых обработчиков запросов. Практическая разработки сервера началась с установки базовых модулей и сопутствующей базовой конфигурацией сервера. Большая часть времени понадобилась на описание контроллеров и реализацией сервисов. Для взаимодействия нашего сервера с БД мы использовали объектно-реляционное отображение, которое реализовано в библиотеке Hibernate. Для большей возможности масштабируемости и независимости от серверной ОС, мы использовали микросервисную технологию, которую прекрасно реализует Docker и соответственно сервис базы данных и файловый сервис семейства Amazon S3 сделали в самостоятельных контейнерах. После завершения процесса разработки, мы приступили к этапу “тестирования”. Тестирование api функций мы производили посредством специального google chrome расширения “Servistate HTTP Editor”, который позволили наиболее быстро и удобно отладить все api функции, а затем выявить, и устранить множество недостатков, связанных с логическими ошибками в коде.

Список литературы

1. Герберт, Шилдт Java 8. Руководство для начинающих / Шилдт Герберт. - М.: Диалектика / Вильямс, 2015. - 899 с.

2. Джошуа, Блох Java. Эффективное программирование / Блох Джошуа. - М.: ЛОРИ, 2014. - 292 с.

3. Официальная документация Spring. URL: https: // spring. io/docs (Дата обращения: 18.12.2020)

4. Spring IO Platform Reference Guide. URL: http://docs. spring. io/spring/docs/4.3.0. RC2/spring-framework-reference/htmlsingle/ (Дата обращения 14.12.2020)

5. Seth Ladd, Darren Davison, Expert Spring MVC and Web Flows [2006]

6. Кларенс Хо, Роб Харроп Spring 3 для профессионалов [2013]

7. Быстрый старт spring. URL: http://spring-projects.ru/projects/spring-framework/ (Дата обращения: 14.12.2020)

8. Давыдов, Станислав IntelliJ IDEA. Профессиональное программирование на Java / Станислав Давыдов , Алексей Ефимов. - М.: БХВ-Петербург, 2015. - 800 с.

9. Хортон А., Вайс Р. - Разработка веб-приложений в ReactJS - Издательство "ДМК Пресс" - 2016 - 254с. - ISBN: 978-5-94074-819-9 - Текст электронный // ЭБС ЛАНЬ - URL: https://e.lanbook.com/book/97339

10. Алекс, Бэнкс React и Redux. Функциональная веб-разработка. Руководство / Бэнкс Алекс. - М.: Питер, 2018. - 458 с.

11. Ньюмен, Сэм Создание микросервисов / Сэм Ньюмен. - М.: Питер, 2015. - 497 с.

12. Сэм, Ньюмен Создание микросервисов. Руководство / Ньюмен Сэм. - М.: Питер, 2016. - 145 с.

ПРИЛОЖЕНИЕ А

Листинг А.1.1.1 – Листинг CorsConfiguration.java

```
package com.bstu.cloudserver;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
public class CorsConfiguration implements WebMvcConfigurer
{
   @Override
   public void addCorsMappings(CorsRegistry registry) {
      registry.addMapping("/**")
            .allowedMethods("GET", "POST","Delete");
   }
}
```

Листинг А.1.1.2 – Листинг OpenAPIConfiguration.java.

```
package com.bstu.cloudserver;

import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.info.Contact;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.info.License;
import io.swagger.v3.oas.annotations.servers.Server;

@OpenAPIDefinition(
      info = @Info(
            title = "Документация MyCloud APIv3",
            description = "" +
                  "Описание основных запросов к облачному файловому хранилищу.",
            contact = @Contact(
                  name = "Alex",
                  url = "https://github.com/alex-rudenkiy",
                  email = "alex-rudenkiy@mail.ru"
            ),
            license = @License(
                  name = "MIT Licence",
                  url = "https://opensource.org/licenses/MIT")),
      servers = @Server(url = "http://localhost:8181")
)
class OpenAPIConfiguration {}
```

Листинг А.1.1.3 – Листинг application.properties

```
 server.port=8181

spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=true
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL95Dialect

spring.datasource.driverClassName=org.postgresql.Driver


spring.datasource.url= jdbc:postgresql://localhost:5432/docker?createDatabaseIfNotExist=true
spring.datasource.username=docker
spring.datasource.password=docker

spring.jpa.show-sql=true
spring.session.store-type=none

spring.servlet.multipart.max-file-size=25MB
spring.servlet.multipart.max-request-size=25MB

springdoc.api-docs.enabled=true
springdoc.api-docs.path= /v3/api-docs

springdoc.swagger-ui.disable-swagger-default-url=true
springdoc.swagger-ui.url=/v3/api-docs
springdoc.swagger-ui.configUrl=/v3/api-docs/swagger-config
```

Листинг А.1.1.5 – Листинг Client.java

```
package com.bstu.cloudserver.models.Client;
import lombok.*;
import org.hibernate.annotations.GenericGenerator;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.io.Serializable;

@Entity
@RequiredArgsConstructor
@NoArgsConstructor

public class Client implements Serializable
{
   @Id
   @Column(name = "uuid")
   @GeneratedValue(generator = "system-uuid")
```

```
    @GenericGenerator(name = "system-uuid", strategy = "uuid")
    @Getter
    @Setter
    private String id;

    @Column(name = "name")
    @Getter
    @Setter
    @NonNull
    private String name;

    @Column(name = "email")
    @Getter
    @Setter
    @NonNull
    private String email;

    @Column(name = "passhash")
    @Getter
    @Setter
    @NonNull
    private String passhash;
}
```

Листинг А.1.2.1 – Листинг ClientController.java

```
package com.bstu.cloudserver.models.Client;

import com.bstu.cloudserver.Response;

import com.bstu.cloudserver.models.Client.dto.ReqLoginDto;

import com.bstu.cloudserver.models.Client.dto.ReqRegisterDto;

import com.bstu.cloudserver.models.Session.Session;

import com.google.gson.Gson;

import io.minio.errors.*;

import io.swagger.v3.oas.annotations.Operation;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.ApplicationContext;

import org.springframework.stereotype.Controller;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;


@Controller
@Transactional
public class ClientController {
    @Autowired
    ApplicationContext context;


    @Operation(summary = "Регистрация", description = "", tags = { "Пользователь" })
    @RequestMapping(value = "/api/v1/client/register", method = RequestMethod.POST)
    public @ResponseBody String userReg(@RequestBody ReqRegisterDto data) throws
IOException, InvalidResponseException, InvalidKeyException, NoSuchAlgorithmException,
ServerException, ErrorResponseException, XmlParserException, InsufficientDataException,
InternalException {
        Client v = context.getBean(ClientService.class).register(data);
        return new Gson().toJson(new Response(v!=null?"ok":"failed",v));
    }
    @Operation(summary = "Авторизация", description = "", tags = { "Пользователь" })
    @RequestMapping(value = "/api/v1/client/login", method = RequestMethod.POST)
    public @ResponseBody String userLogin(@RequestBody ReqLoginDto data) {//,
@RequestParam("remember") String remember
        Session v = context.getBean(ClientService.class).login(data);
        return new Gson().toJson(new Response(v!=null?"ok":"failed",v));
    }
}
```

Листинг А.1.2.2. – Листинг ClientJPA.java

```
package com.bstu.cloudserver.models.Client;
import org.hibernate.annotations.Entity;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

@Entity
public interface ClientJPA extends JpaRepository<Client, Long>
{
   Boolean existsByName(String name);
   Boolean existsByEmail(String email);
   Boolean existsByPasshash(String passhash);
   List<Client> findClientByNameEquals(String name);
}
```

Листинг А.1.2.3 – Листинг ClientService.java

```
package com.bstu.cloudserver.models.Client;
import com.bstu.cloudserver.models.Client.dto.ReqLoginDto;
import com.bstu.cloudserver.models.Client.dto.ReqRegisterDto;
import com.bstu.cloudserver.models.FileStorage.FileStorageService;
import com.bstu.cloudserver.models.Session.Session;
import com.bstu.cloudserver.models.Session.SessionJPA;
import io.minio.MakeBucketArgs;
import org.apache.commons.codec.digest.DigestUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;
import java.time.LocalDateTime;
import static com.bstu.cloudserver.models.FileStorage.Minio.minioClient;

@Service
public class ClientService {
   @Autowired
   ApplicationContext context;
```

```
    Client register(ReqRegisterDto data){
        Client result = null;
        Client client = new Client(data.login, data.email, DigestUtils.md5Hex( data.password ));

        if(!context.getBean(ClientJPA.class).existsByName(data.login) &&
!context.getBean(ClientJPA.class).existsByEmail(data.email)) {

            result = context.getBean(ClientJPA.class).save(client);

            try {
                minioClient.makeBucket(
                        MakeBucketArgs.builder()
                                .bucket(context.getBean(FileStorageService.class).getBucketName(result))
                                .build());
            }catch (Exception e){
                context.getBean(ClientJPA.class).delete(client);
            }

        }
        return result;
    }

    Session login(ReqLoginDto data){
        Client u = null;

        if(context.getBean(ClientJPA.class).existsByName(data.login) &&
context.getBean(ClientJPA.class).existsByPasshash(DigestUtils.md5Hex(data.password))) {
            u = context.getBean(ClientJPA.class).findClientByNameEquals(data.login).get(0);

            LocalDateTime actualDateTime = LocalDateTime.now();
```

```
            if (data.remember) {
                actualDateTime = actualDateTime.plusHours(1);
            } else {
                actualDateTime = actualDateTime.plusYears(1);
            }

            context.getBean(SessionJPA.class).deleteAllByClientEquals(u);
            return context.getBean(SessionJPA.class).save(new Session(u, actualDateTime));
        }else {
            return null;
        }
    }
}
```

Листинг А.1.3.1 – Листинг SharedFile.java

```
package com.bstu.cloudserver.models.FileStorage.SharedFile;
import com.bstu.cloudserver.models.Client.Client;
import lombok.*;
import org.hibernate.annotations.GenericGenerator;
import javax.persistence.*;
import java.io.Serializable;
import java.time.LocalDateTime;

@Entity
@RequiredArgsConstructor
@NoArgsConstructor
public class SharedFile implements Serializable {

    @Id
    @Column(name = "uuid")
    @GeneratedValue(generator = "system-uuid")
    @GenericGenerator(name = "system-uuid", strategy = "uuid")
```

```
    @Getter
    @Setter
    private String id;

    @OneToOne
    @JoinColumn(name = "Client")
    @Getter
    @Setter
    @NonNull
    private Client client;

    @Column(name = "Expire")
    @Getter
    @Setter
    @NonNull
    private LocalDateTime expires;

    @Column(name = "Filepath")
    @Getter
    @Setter
    @NonNull
    private String filepath;
}
```

Листинг А.1.3.2 – Листинг SharedFileController.java

```
package com.bstu.cloudserver.models.FileStorage.SharedFile;
import com.bstu.cloudserver.Response;
import com.google.gson.Gson;
import io.swagger.v3.oas.annotations.Operation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import javax.servlet.http.HttpServletResponse;

@Controller
public class SharedFileController {
    @Autowired
    ApplicationContext context;

    @Operation(summary = "Получить информацию об опубликованном файле", description
= "", tags = { "Файловое пространство" })
    @RequestMapping(value = "/api/v1/shared/{sharedFileId}", method = RequestMethod.GET)
    public @ResponseBody
    String getSharedFileInfo(@PathVariable String sharedFileId) {
        SharedFile t =
context.getBean(SharedFileJPA.class).getSharedFileByIdEquals(sharedFileId);
        return new Gson().toJson(new Response(t!=null?"ok":"failed",t));
    }

    @Operation(summary = "Скачать опубликованный файл", description = "", tags = {
"Файловое пространство" })
    @RequestMapping(value = "/api/v1/shared/{sharedFileId}", method = RequestMethod.POST)
    public void downloadSharedFile(@PathVariable String sharedFileId, HttpServletResponse
response) {
        SharedFile t =
context.getBean(SharedFileJPA.class).getSharedFileByIdEquals(sharedFileId);
        context.getBean(SharedFileService.class).getFile(t.getClient(), t.getFilepath(), response);
    }
}
```

Листинг А.1.3.3 – Листинг SharedFileService.java

```
package com.bstu.cloudserver.models.FileStorage.SharedFile;
import com.bstu.cloudserver.models.Client.Client;
import com.bstu.cloudserver.models.FileStorage.FileStorageService;
import io.minio.GetObjectArgs;
import io.minio.GetObjectResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.core.io.InputStreamResource;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import javax.servlet.http.HttpServletResponse;
import java.io.InputStream;
import static com.bstu.cloudserver.models.FileStorage.Minio.minioClient;

@Service
public class SharedFileService {
    @Autowired
    ApplicationContext context;

    public ResponseEntity<Resource> getFile(
            Client owner,
            String filePath,
            HttpServletResponse response) {
        try {
            try {
                InputStream is = minioClient.getObject(
                        GetObjectArgs.builder()
```

```
                    .bucket(context.getBean(FileStorageService.class).getBucketName(owner))
                    .object(filePath)
                    .build());

            HttpHeaders header = new HttpHeaders();
            header.add(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename="+
((GetObjectResponse) is).object());
            header.add("Cache-Control", "no-cache, no-store, must-revalidate");
            header.add("Pragma", "no-cache");
            header.add("Expires", "0");

            InputStreamResource resource = new InputStreamResource(is);

            return ResponseEntity.ok()
                    .headers(header)
                    .contentLength(((GetObjectResponse) is).headers().byteCount())
                    .contentType(MediaType.APPLICATION_OCTET_STREAM)
                    .body(resource);
          } catch (Exception e) {
          }
      } catch (Exception ex) {
          throw new RuntimeException("IOError writing file to output stream");
      }
      return null;
   }
}
```

Листинг А.1.3.4 – Листинг SharedFileJPA.java

```
package com.bstu.cloudserver.models.FileStorage.SharedFile;
import org.hibernate.annotations.Entity;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
@Entity
public interface SharedFileJPA extends JpaRepository<SharedFile, Long>
{
   SharedFile getSharedFileByIdEquals(String id);
}
```

Листинг А.1.4.1 – Листинг Session.java

```
package com.bstu.cloudserver.models.Session;
import com.bstu.cloudserver.models.Client.Client;
import lombok.*;
import org.hibernate.annotations.GenericGenerator;
import javax.persistence.*;
import java.io.Serializable;
import java.time.LocalDateTime;

@Entity
@RequiredArgsConstructor
@NoArgsConstructor
public class Session implements Serializable {
   @Id
   @Column(name = "uuid")
   @GeneratedValue(generator = "system-uuid")
   @GenericGenerator(name = "system-uuid", strategy = "uuid")
   @Getter
   @Setter
   private String token;

   @OneToOne
   @JoinColumn(name = "Client")
   @Getter
   @Setter
```

```
    @NonNull
    private Client client;
    @Column(name = "Expire")
    @Getter
    @Setter
    @NonNull
    private LocalDateTime expires;
}
```

Листинг А.1.4.2 – Листинг SessionController.java

```
package com.bstu.cloudserver.models.Session;
import com.bstu.cloudserver.Response;
import com.google.gson.Gson;
import io.swagger.v3.oas.annotations.Operation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class SessionController {
    @Autowired
    ApplicationContext context;

    @Operation(summary = "Проверить токен", description = "", tags = { "Сессия" })
    @RequestMapping(value = "/api/v1/session/checkToken", method=RequestMethod.POST)
    public @ResponseBody String auth(@RequestParam("token") String token) {
        return new Gson().toJson(new Response("ok",
context.getBean(SessionService.class).checkToken(token)));
```

```
    }
}
```

Листинг А.1.4.3 – Листинг SessionJPA.java

```
package com.bstu.cloudserver.models.Session;
import com.bstu.cloudserver.models.Client.Client;
import org.hibernate.annotations.Entity;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

@Entity
public interface SessionJPA extends JpaRepository<Session, Long>
{
    List<Session> findSessionByTokenEquals(String token);
    void deleteAllByClientEquals(Client client);
}
```

Листинг А.1.4.4 – Листинг SessionService.java

```
package com.bstu.cloudserver.models.Session;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;

@Service
public class SessionService {
    @Autowired
    ApplicationContext context;

    public Boolean checkToken(String token){
        return !context.getBean(SessionJPA.class).findSessionByTokenEquals(token).isEmpty();
    }
}
```

Листинг А.1.5.1 – Листинг FileStorage.java

```
package com.bstu.cloudserver.models.FileStorage;
import com.bstu.cloudserver.models.Client.Client;
import com.bstu.cloudserver.models.Promo.Promo;
import lombok.*;
import org.hibernate.annotations.GenericGenerator;
import javax.persistence.*;
import java.io.Serializable;
import java.util.List;

@Entity
@RequiredArgsConstructor
@NoArgsConstructor
public class FileStorage implements Serializable {

    @Id
    @Column(name = "uuid")
    @GeneratedValue(generator = "system-uuid")
    @GenericGenerator(name = "system-uuid", strategy = "uuid")
    @Getter
    @Setter
    private String id;

    @OneToOne
    @JoinColumn(name = "client")
    @NonNull
    private Client client;

    @OneToMany
    @JoinColumn(name = "promo")
    @NonNull
```

```
    private List<Promo> promo;
}
```

Листинг А.1.5.2 – Листинг FileStorageController.java

```
package com.bstu.cloudserver.models.FileStorage;
import com.bstu.cloudserver.Response;
import com.bstu.cloudserver.models.Client.Client;
import com.bstu.cloudserver.models.FileStorage.Dto.ReqGetFileDto;
import com.bstu.cloudserver.models.FileStorage.Dto.ReqHandleFileDeleteDto;
import com.bstu.cloudserver.models.FileStorage.Dto.ReqHandleFileUploadDto;
import com.bstu.cloudserver.models.FileStorage.Dto.ReqProvideListInfoDto;
import com.bstu.cloudserver.models.Session.Session;
import com.bstu.cloudserver.models.Session.SessionJPA;
import com.google.gson.Gson;
import io.minio.PutObjectArgs;
import io.minio.errors.*;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Schema;
import org.apache.commons.codec.DecoderException;
import org.apache.commons.codec.net.URLCodec;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.security.InvalidKeyException;
```

```
import java.security.NoSuchAlgorithmException;

import java.util.List;

import static com.bstu.cloudserver.models.FileStorage.Minio.minioClient;


@Controller

public class FileStorageController {

   @Autowired

   ApplicationContext context;


   @Operation(summary = "Получить содержимое папки", description = "", tags = { "Файловое пространство" })

   @RequestMapping(value="/api/v1/mycloud/**", method=RequestMethod.POST)

   public @ResponseBody String provideListInfo(@RequestBody ReqProvideListInfoDto data, HttpServletRequest request) throws DecoderException {

      String dirpath = "";

      if(request.getRequestURL().toString().split("/mycloud/").length>1) dirpath = new URLCodec().decode(request.getRequestURL().toString().split("/mycloud/")[1]);


      return new Gson().toJson(new Response("ok",context.getBean(FileStorageService.class).provideListInfo(data.getToken(), dirpath)));

   }


   @Operation(summary = "Загрузить файл в папку", description = "", tags = { "Файловое пространство" })

   @RequestMapping(value="/api/v1/upload/mycloud/**", method=RequestMethod.POST, consumes = MediaType.MULTIPART_FORM_DATA_VALUE)

   public @ResponseBody String handleFileUpload(

         @ModelAttribute ReqHandleFileUploadDto data,

                              HttpServletRequest request) throws DecoderException {

      String dirpath = "";

      if(request.getRequestURL().toString().split("/mycloud/").length>1) {

         String tpath = request.getRequestURL().toString().split("/mycloud/")[1];
```

```
            dirpath = new URLCodec().decode(tpath);
        }


        return new Gson().toJson(new
Response("ok",context.getBean(FileStorageService.class).handleFileUpload(data.getToken(),
data.getFile(), dirpath)));
    }


    @Operation(summary = "Скачать файл в папку", description = "", tags = { "Файловое
пространство" })
    @RequestMapping(value = "/api/v1/download/mycloud/**", method =
RequestMethod.POST)
    public void getFile(
            @RequestBody ReqGetFileDto data,
            HttpServletRequest request,
            HttpServletResponse response) throws DecoderException {
        String filePath = "";
        if(request.getRequestURL().toString().split("/mycloud/").length>1) filePath = new
URLCodec().decode(request.getRequestURL().toString().split("/mycloud/")[1]);


        Client owner = null;
        List<Session> t =
context.getBean(SessionJPA.class).findSessionByTokenEquals(data.getToken());
        if(!t.isEmpty()){
            owner = t.get(0).getClient();
        }


        context.getBean(FileStorageService.class).getFile(owner, filePath, response);
    }


    @Operation(summary = "Создать пустую папку в папке", description = "", tags = {
"Файловое пространство" })
    @RequestMapping(value = "/api/v1/mkdir/mycloud/**", method = RequestMethod.POST)
    public @ResponseBody String mkDirectory(
```

```
    @RequestBody ReqGetFileDto data,
    HttpServletRequest request,
    HttpServletResponse response) throws DecoderException {

String dirpath = "";
if(request.getRequestURL().toString().split("/mycloud/").length>1) dirpath = new
URLCodec().decode(request.getRequestURL().toString().split("/mycloud/")[1]);

Client c = null;

List<Session> t =
context.getBean(SessionJPA.class).findSessionByTokenEquals(data.getToken());
if(!t.isEmpty()){
   c = t.get(0).getClient();
}

if (c!=null &&!dirpath.isEmpty()) {
   try {
      minioClient.putObject(
            PutObjectArgs.builder()
                  .bucket(context.getBean(FileStorageService.class).getBucketName(c))
                  .object(dirpath)
                  .stream(
                  new ByteArrayInputStream(new byte[] {}), 0, -1)
                  .build());
      new Gson().toJson(new Response("ok",""));
   } catch (Exception e) {
      return null;
   }
} else {
   return null;
}
```

```
        return null;
    }


    @Operation(summary = "Удалить файл или папку", description = "", tags = { "Файловое пространство" })
    @RequestMapping(value="/api/v1/delete/mycloud/**", method = RequestMethod.POST)
    public @ResponseBody String handleFileDelete(@Parameter(description="UserSessionToken",required=true, schema=@Schema(implementation = ReqHandleFileDeleteDto.class))
                                                  @RequestBody ReqHandleFileDeleteDto data,
                                                  HttpServletRequest request) throws IOException, InvalidKeyException, InvalidResponseException, InsufficientDataException, NoSuchAlgorithmException, ServerException, InternalException, XmlParserException, ErrorResponseException, DecoderException {
        String filepath = "";
        if(request.getRequestURL().toString().split("/mycloud/").length>1) filepath = new URLCodec().decode(request.getRequestURL().toString().split("/mycloud/")[1]);


        return new Gson().toJson(new Response("ok",context.getBean(FileStorageService.class).handleFileDelete(data.getToken(), filepath)));
    }
}
```

Листинг А.1.5.3 – Листинг FileStorageService.java

```
package com.bstu.cloudserver.models.FileStorage;

import com.bstu.cloudserver.models.Client.Client;

import com.bstu.cloudserver.models.FileStorage.Dto.RespProvideListInfoDto;

import com.bstu.cloudserver.models.Session.Session;

import com.bstu.cloudserver.models.Session.SessionJPA;

import com.mpatric.mp3agic.ID3v2;

import com.mpatric.mp3agic.Mp3File;

import io.minio.*;

import io.minio.errors.*;

import io.minio.messages.Item;
```

```
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.tomcat.util.http.fileupload.IOUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.LinkedList;
import java.util.List;
import static com.bstu.cloudserver.models.FileStorage.Minio.minioClient;

@Service
public class FileStorageService {
    @Autowired
    ApplicationContext context;

    public String getBucketName(Client client) {
        System.out.println(client.getName());
        return client.getName().replaceAll("[^A-Za-z0-9]", "");
    }

    public List<RespProvideListInfoDto> provideListInfo(String token, String dirpath) {
        Client c = null;
        List<RespProvideListInfoDto> result = new LinkedList<>();

        List<Session> t = context.getBean(SessionJPA.class).findSessionByTokenEquals(token);
        if (!t.isEmpty()) {
            c = t.get(0).getClient();
```

```
        Iterable<Result<Item>> results = minioClient.listObjects(

ListObjectsArgs.builder().bucket(context.getBean(FileStorageService.class).getBucketName(c)).
prefix(dirpath).build());


        results.forEach(e -> {
            try {
                result.add(new RespProvideListInfoDto(e.get().objectName(), e.get().objectName(),
e.get().isDir()));
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        });
    }
    return result;
  }


  public Boolean handleFileUpload(String token, MultipartFile file, String dirpath) {
    Client c = null;


    List<Session> t = context.getBean(SessionJPA.class).findSessionByTokenEquals(token);
    if (!t.isEmpty()) {
      c = t.get(0).getClient();
    }


    if (c != null && !file.isEmpty()) {
        File randTmpFileName = new File(String.format("tmp\\%s.dat",
RandomStringUtils.random(15, true, true)));
        try {
          byte[] bytes = file.getBytes();
          BufferedOutputStream stream =
                new BufferedOutputStream(new FileOutputStream(randTmpFileName));
          stream.write(bytes);
```

```
            stream.close();

            minioClient.uploadObject(
                  UploadObjectArgs.builder()
                        .bucket(context.getBean(FileStorageService.class).getBucketName(c))
                        .object(dirpath + file.getOriginalFilename())
                        .filename(randTmpFileName.getAbsolutePath())
                        .build());

            randTmpFileName.delete();
            return true;

         } catch (Exception e) {
            return false;
         }
      } else {
         return false;//return "Вам не удалось загрузить " + name + " потому что файл пустой.";
      }
   }

   public Boolean handleFileDelete(String token, String filepath) throws IOException, InvalidKeyException, InvalidResponseException, InsufficientDataException, NoSuchAlgorithmException, ServerException, InternalException, XmlParserException, ErrorResponseException {
      Client c = null;
      List<Session> t = context.getBean(SessionJPA.class).findSessionByTokenEquals(token);
      if (!t.isEmpty()) {
         c = t.get(0).getClient();
      }

minioClient.removeObject(RemoveObjectArgs.builder().bucket(context.getBean(FileStorageService.class).getBucketName(c)).object(filepath).build());
```

```
        return true;
    }

    public void getFile(
            Client owner,
            String filePath,
            HttpServletResponse response) {
        try {
            try {
                InputStream is = minioClient.getObject(
                        GetObjectArgs.builder()
                                .bucket(context.getBean(FileStorageService.class).getBucketName(owner))
                                .object(filePath)
                                .build());
                response.setContentType("application/x-download");
                response.setHeader("Content-disposition", "attachment; filename=" +
((GetObjectResponse) is).object());

                System.out.println(((GetObjectResponse) is).object().toString());
                IOUtils.copy(is, response.getOutputStream());
                response.flushBuffer();
            } catch (Exception e) {}
        } catch (Exception ex) {
            throw new RuntimeException("IOError writing file to output stream");
        }
    }

    public Boolean checkToken(String token) {
        return false;
    }
}
```