Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Удаленное файловое хранилище

Студент: гр. 951007
Воривода М.А.

Руководитель:
Царук А.И.

Минск 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Никогда в истории человечества информация не была столь ценным ресурсом для столь большого количества людей. Две тысячи лет назад все накопленные человеком знания помещались в одной библиотеке, а бытовая информация и вовсе помещалась в голове. С каждой сотней лет скорость накопления информации увеличивалась. После того как были изобретены различные записывающие устройства количество знаний начало увеличиваться экспоненциально, но именно после широкого распространения фотоаппаратов, диктофонов и телефонов наступил информационный взрыв. Общий объём информации растёт на 30% каждый год, то есть каждые несколько лет этот объём увеличивается вдвое.

Как и две тысячи лет назад, так и сейчас огромная часть производимой информации бесполезна. Однако 1% знаний тогда и сейчас – это на порядки отличающиеся объёмы. Поэтому в современном мире твёрдо укрепилась проблема хранения информации. Если десять лет назад человеку хватало DVD-диска, то сейчас ему, скорее всего, не будет хватать хранилищ на десятки и сотни, а иногда даже тысячи, гигабайт. Также увеличивается и количество людей с такой проблемой. Отсюда появляется запрос на ёмкие и практичные хранилища данных. Диски – очень хрупкий и не очень компактный носитель. Твердотельные накопители, подключаемые по USB – либо слишком маленький объём, либо высокая стоимость, к минусам также можно отнести то, что для чтения нужен USB-порт, которого нет в телефоне, очень важном гаджете в жизни современного человека. SD-карта – компактный, поддерживается большинством смартфонов, но недостатки те же, что и у USB-flash. Решение: удалённое файловое хранилище.

1 АНАЛИЗ АНАЛОГОВ ПРОГРАММНОГО СРЕДСТВА

1.1 «Google Диск» - сервис хранения, редактирования и синхронизации файлов, разработанный компанией Google

Перед началом анализа аналогов программного средства необходимо разобрать основные понятия и суть архитектуры, на базе которой построено удаленное хранилище данных.

Архитектура клиент – сервер (*client-server architecture*) – это концепция информационной сети, в которой основная часть ее ресурсов сосредоточена в серверах, обслуживающих своих клиентов. Рассматриваемая архитектура определяет два типа компонентов: *серверы* и *клиенты*.

Сервер – это объект, предоставляющий сервис другим объектам сети по их запросам. Сервис – это процесс обслуживания клиентов. Сервер работает по заданиям клиентов и управляет выполнением их заданий. После выполнения каждого задания сервер посылает полученные результаты клиенту, пославшему это задание. Сервисная функция в архитектуре клиент – сервер описывается комплексом прикладных программ, в соответствии с которым выполняются разнообразные прикладные процессы.

Процесс, который вызывает сервисную функцию с помощью определенных операций, называемых *клиентом.* Им может быть программа или пользователь.

Перейдем к анализу наиболее известного аналога: «Google Диск» (рисунок 1.1). безопасное хранилище для резервного копирования файлов и работы с ними на любых устройствах. Вы можете приглашать других пользователей просматривать, редактировать или комментировать контент в любых файлах и папках.

Вы можете хранить и передавать коллегам файлы и папки, а также работать над ними вместе с другими пользователями на компьютере или любом мобильном устройстве

Диск полностью совместим с Документами, Таблицами и Презентациями. Эти облачные продукты помогут вам и вашим коллегам эффективно взаимодействовать в режиме реального времени. Вы можете сразу создавать файлы и открывать к ним доступ. Для этого не потребуется переносить материалы из сервисов, с которыми вы работали.

Диск совместим с технологиями, которыми пользуется ваша команда, и дополняет их. Для совместной работы над файлами Microsoft Office не требуется преобразовывать их в другие форматы. Вы можете редактировать и хранить файлы более чем 100 других типов, включая PDF, CAD и т. д.

Специальные средства, реализованные в Google Диске, обеспечивают непревзойденную скорость, эффективность и надежность поиска. А различные функции, например, вкладка "Важные", используют технологии

искусственного интеллекта, чтобы определять, что именно вас интересует, и показывать вам наиболее подходящие результаты. Благодаря этому вы будете тратить на поиск нужных материалов на 50 % меньше времени.

Мобильное приложение «Google Диск» для Android поможет легко отсканировать документы, визитные карточки, квитанции и другие бумажки. Всё это превратится в удобочитаемые PDF-файлы и загрузится в облако.

Нажмите на кнопку со значком плюса в клиенте и выберите опцию «Сканировать», затем наведите камеру на текст. Приложение автоматически обрежет пустые края и осветлит фон. При необходимости вы можете нажать на кнопку «Кадрировать» и указать нужный фрагмент текста вручную.

У «Google Диска» есть официальное расширение, которое позволяет отправлять веб-страницы, файлы и изображения прямо в ваше хранилище.

Нужно просто щёлкнуть правой кнопкой мыши на картинку и выбрать «Сохранить изображение в „Google Диск“». Либо открыть какой-нибудь документ PDF из интернета в браузере и нажать на иконку расширения. Всё отправится в облако.

Веб-страницы расширение сохраняет в виде скриншота, HTML, MHT или же конвертирует их в формат Google Docs целиком.

В Windows или macOS разместить на рабочем столе ярлык на документ или картинку из хранилища Google очень просто, если у вас установлен клиент «Автозагрузка и синхронизация». Выберите в вашей папке «Google Диска» нужный файл, щёлкните правой кнопкой мыши, создайте ярлык в Windows или псевдоним в macOS, и готово. Это довольно очевидное действие.

Но точно так же вы можете создавать ссылки на файлы и на мобильных устройствах. Найдите нужный файл или папку в приложении «Google Диск», нажмите на многоточие и выберите «Добавить на главный экран». Теперь нужные данные всегда будут у вас под рукой.

Если вы храните на «Google Диске» какие-то особо ценные данные и хотите обеспечить им дополнительную защиту, установите бесплатное дополнение Secure File Encryption. Оно шифрует файлы по алгоритму AES-256.

Нажмите «Создать» → «Ещё» → Secure File Encryption. Затем придумайте пароль и перетащите файл в окно браузера. В дальнейшем его нельзя будет скачать или просмотреть без ввода пароля.

Вы пишете письмо и хотите приложить к нему документ или картинку из своего хранилища? Не надо переключаться на другую вкладку и открывать «Google Диск». Нажмите на значок внизу окошка составления письма и выберите нужный файл прямо в интерфейсе Gmail.

Таким образом можно отправлять вложения значительных размеров. По умолчанию Gmail не позволяет прикрепить файл тяжелее 25 МБ. Но вложения с «Google Диска» могут достигать 10 ГБ.

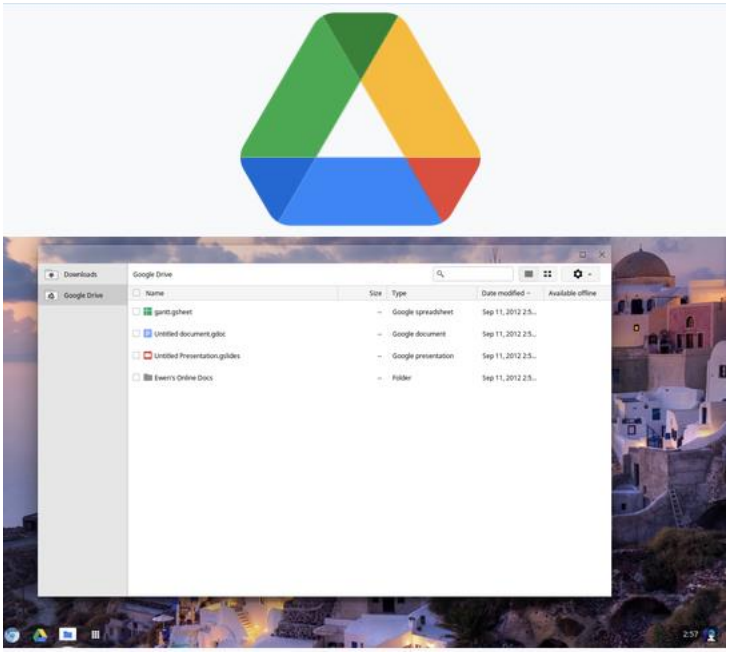

Рисунок 1.1 – программное средство «Google Диск»

2 ЦЕЛИ И ЗАДАЧИ

2.1 Цель курсового проекта

Целью данного курсового проекта является разработка программного средства: удаленное файловое хранилище.

2.2 Задачи для курсового проекта

Проанализировав аналоги и архитектуру клиент-серверного чата, можно выделить следующие задачи:

- Разработка модуля, раюотающего с файловой системой.
- Разработка модулей, принимающих HTTP-запросы.
- Поддержка авторизации и аутентификации.
- Разработка создания базы данных.
- Создание программного слоя, разделяющего базу данных и бизнес-логику.
- Написание графического интерфейса.
- Создание модуля, отправляющего запросы.
- Разделение клиента по схеме model-view-controller.
- Интеграция spring в JavaFX.

2.3 Необходимые ресурсы для курсового проекта

Для разработки программного средства использован язык программирования Java. Среда разработки – IntelliJ IDEA. Это одна из самых мощных и популярных интегрированных сред разработки. Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса.

3 ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА КУРСОВОЙ РАБОТЫ

3.1 Структура программного средства

Данный проект представляет собой совокупность двух модулей – сервера и клиента.

Сервер содержит веб-службу построенную в соответствии с REST-архитектурой (далее Remote Storage Server). Веб-служба отвечает за авторизацию и аутентификацию, регистрацию, управление связанных баз данных PostgreSQL и управление личным хранилищем пользователя. Данный модуль спроектирован с помощью Spring Framework для Java. За сборку отвечает Apache Maven. Классы-логгеры предоставляет SLF4J.

Клиент представляет Desktop-приложение, интерфейс для которого был написан с помощью JavaFX (далее Remote Storage Client). За управление некоторыми зависимостями отвечает Spring Boot, а за отправку запросов – Spring Framework Web Client, конкретно – класс RestTemplate.

Модуль клиента спроектирован по схеме Model-View-Controller.

3.2 Структура серверной веб-службы

Основная задача Remote Storage Server – управление пользовательским хранилищем. Для каждого пользователя выделено своё хранилище, находящееся на диске машины, где развернута веб-служба. Для получения личного хранилища пользователю необходимо зарегистрироваться, а для получения доступа к хранилищу пользователь должен пройти аутентификацию. Здесь можно выделить три части – регистрация, аутентификация и авторизация, доступ к ресурсам удалённого хранилища.

Spring позволяет передавать управление объектам в зависимости от URL по которому обратились к веб-службе. Так называемые контроллеры должны быть помечены соответствующими аннотациями (@Controller), а их методы или они сами должны быть помечены аннотациями, отвечающими за маппинг (@GetMapping, @PutMapping, @RequestMapping и так далее). Таким образом Spring знает какие вызывать методы у каких контроллеров при получении запроса на определённый URL. Для REST API можно использовать аннотацию @RestController – сочетание @Controller и @ResponseBody, сигнализирующая о том, что объекты данного класса могут возвращать тело в HTTP-ответе. Remote Storage Server обрабатывает запросы на «/registration», «/login» и «/storage/**». Соответственно в этом модуле есть три контроллера: RegistrationController, LoginController и StorageController.

3.2.1 RegistrationController

Данный класс содержит в себе экземпляр логгера, один метод для обработки запроса и UserService.

UserService – это класс обёртка позволяющий удобно работать с подключённой базой данных пользователей. Пользователя представляет класс User. Для того, чтобы Spring мог использовать данный сервис он помечен аннотацией Service. Для использования его в процессе аутентификации сам класс имплементирует интерфейс UserDetailsService, а User – UserDetails. В себе объекты класса UserService содержат JPA-репозитории – сущности, непосредственно работающие с базой данных. В данном случае присутствуют два репозитория: TokenRepository и UserRepository.

```
private final Logger logger = LoggerFactory.getLogger(UserService.class);

private final TokenRepository tokenRepository;
private final UserRepository userRepository;
private final PasswordEncoder passwordEncoder;

@Autowired
public UserService(UserRepository userRepository, TokenRepository tokenRepository, PasswordEncoder
passwordEncoder) {
   this.userRepository = userRepository;
   this.tokenRepository = tokenRepository;
   this.passwordEncoder = passwordEncoder;
}
```

TokenRepository связан с таблицей, хранящей токены аутентификации.

UserRepositoty связан с таблицей, в которой отражается сущность User.

Метод регистрирующий пользователя последовательно выполняет следующие действия: получает имя пользователя и пароль из параметров запроса, проверяет на наличие имя пользователя в системе, проверяет имя пользователя и пароль на корректность, добавляет сохраняет пользователя в базу данных. Перед сохранением в БД UserService хеширует пароль с помощью PasswordEncoder.

```
@PostMapping(GlobalConstants.REGISTRATION_PATH)
public ResponseEntity<String> addUser(@RequestParam("username") String username,
                  @RequestParam("password") String password) throws Exception {

   logger.info("Registration of new user...");
   if (userService.loadUserByUsername(username) != null) throw new
SuchUserAlreadyExistsException("Such user already exists.");
   String errorsReport = getRegistrationErrorsReport(username, password);
   if (errorsReport.length() > 0) throw new InvalidRegistrationArgumentsException(errorsReport);

   userService.create(new User(username, password));
   logger.info("Registered.");
   return new ResponseEntity<>("", HttpStatus.OK);
}
```

```
public boolean create(User user) {
  logger.info("Trying to create user in repository...");
  if (userRepository.findByUsername(user.getUsername()) != null) {
    logger.warn("Such user already exists.");
    return false;
  }

  user.setPassword(passwordEncoder.encode(user.getPassword()));
  logger.info("Creating user " + user + "...");
  userRepository.save(user);
  logger.info("User created.");
  return true;
}
```

3.2.2 LoginController

LoginController также содержит UserService для работы со списком пользователей и имеет один метод для обработки запроса. Аутентификация происходит по имени пользователя и паролю, которые пересылаются в заголовке «Authorization» с подзаголовком «Basic» в формате «имя:пароль» и закодированные в Base64. Так как REST подразумевает общение клиента и сервера без установки соединения, при каждом запросе, требующем аутентификации необходимо будет указывать имя пользователя и пароль. Чтобы избежать этого LoginController при удачной аутентификации возвращает токен безопасности, который можно в дальнейшем использовать для доступа к хранилищу. Токен имеет время жизни – 60 минут. На практике объект SecurityToken хранит время, в которое время жизни токен истечёт. Зная это время, клиент может запросить новый токен в случае если текущий уже истёк. В запросе токен находится в заголовке «Authorization» с подзаголовком «Bearer».

```
@PostMapping(value = GlobalConstants.LOGIN_PATH)
public ResponseEntity<String> getToken(@RequestHeader("Authorization") String credentials) {
  logger.info("Trying to signing in, credentials: " + credentials + "...");

  credentials = new String(Base64.getDecoder().decode(credentials.substring("Basic".length()).trim()));
  String[] parsed = credentials.split(":", 2);
  logger.debug("Username: " + parsed[0] + ", Password: " + parsed[1]);

  SecurityToken token = userService.signIn(parsed[0], parsed[1]);
  if (token == null) throw new AuthenticationServiceException("Wrong username or password");

  logger.info("Signed in.");
  return new ResponseEntity<>(token.getExpiredDate().getTime() + " " + token.getValue(), HttpStatus.OK);
}
```

Классы необходимые для обеспечения безопасности хранятся в пакете by.vorivoda.matvey.security. Среди них: SecurityToken, AuthenticationFilter – класс, извлекающий токен и передающий в AuthenticationProvider, который проверяет полученный токен, SecurityConfig – класс конфигурации, помеченный аннотацией @Configuration. SecurityConfig реализует методы,

отвечающие за настройку авторизации, то есть за предоставление ресурсов в зависимости от того, прошёл ли запрос аутентификацию. Этот класс указывает Spring, что необходимо требовать аутентификацию для доступа к ресурсам только по адресам, начинающимся с «/storage». Запросы на другие адреса – разрешить для всех.

```
@Override
public void configure(WebSecurity webSecurity) {
  webSecurity.ignoring()
       .antMatchers("/login/**", "/registration/**");
}

@Override
public void configure(HttpSecurity http) throws Exception {
  http.sessionManagement()
       .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
       .and().exceptionHandling()
       .and()
       .authenticationProvider(authenticationProvider)
       .addFilterBefore(authenticationFilter(), AnonymousAuthenticationFilter.class)
       .authorizeRequests()
       .antMatchers("/login/**", "/registration/**").permitAll()
       .antMatchers("/storage/**").authenticated()
       .and()
       .csrf().disable()
       .formLogin().disable()
       .httpBasic().disable()
       .logout().disable();
}
```

3.2.3 StorageController

После аутентификации и авторизации клиент может получать ресурсы по адресу «/storage», этот адрес корнем хранилища. StorageController содержит несколько методов для обработки запросов с различными HTTP-методами. Так как Spring Web Framework использует сервлеты и Tomcat в качестве контейнера сервлета, он ограничен в наборе HTTP-методов спецификацией Tomcat. Поэтому для обработки специфических запросов таких как копирование, перемещение, получение размера и так далее запрос должен содержать заголовок «Operation-Type» с именем соответствующей операции. Для тех методов, который поддерживает Spring предусмотрен стандартный функционал.

```
@PostMapping
public ResponseEntity<String> postRequest(
                        @RequestHeader(value = "Operation-Type", required = false) String operationType,
                        @RequestUrl String url,
                        @RequestBody byte[] content) throws HttpRequestMethodNotSupportedException,
IOException { … }
```

Общими для всех методов являются: декодирование URL, выделение пути относительно корня хранилища, получения менеджера хранилища авторизированного пользователя, получение необходимой операции, запуск операции и возврат значения.

```
private String getStoragePath(String url) {
   String path= url.substring(url.indexOf(GlobalConstants.STORAGE_PATH) +
GlobalConstants.STORAGE_PATH.length());
   path = path.startsWith("/") ? path : "/" + path;

   StringBuilder decodedPath = new StringBuilder();
   int prevSlash = 0;
   for (int i = 1; i < path.length() + 1; i++) {
      if (i ==path.length() || path.charAt(i) == '/') {
         decodedPath.append("/").append(URLDecoder.decode(path.substring(prevSlash + 1, i),
StandardCharsets.UTF_8));
         prevSlash = i;
      }
   }

   return decodedPath.toString();
}
```

Для создания более гибкой архитектуры действия над хранилищем представлены двумя видами операций: FileStorageAPIOperation и FileStorageAPIContentOperation. Первый вид операций не требует тела запроса и может содержать заголовок «Additional», в котором указан дополнительный путь, например, для операции копирования. Второй вид операций может содержать тело запроса. Также было создано перечисление APIOperation хранящее виды операций, из специфических присутствуют SIZE – получение размера, GET_ALL – получение списка всех путей в папке, в том числе и вложенных. StorageController хранит две карты – для операций двух видов, ключ в которых – это элемент APIOperation.

```
private final Map<APIOperation, FileStorageAPIContentOperation> contentOperations = new HashMap<>()
{{

   put(APIOperation.PUT, (storage, path, content) -> { // Put or create and put file
      logger.info("Executing \"PUT\" APIOperation...");
      storage.writeFile(path, content);
      logger.info("Executed successfully.");
      return new ResponseEntity<>("OK", HttpStatus.OK);
   });

   put(APIOperation.POST, (storage, path, content) -> { // Post file
      logger.info("Executing \"POST\" APIOperation...");
      storage.appendFile(path, content);
      logger.info("Executed successfully.");
      return new ResponseEntity<>("OK", HttpStatus.OK);
   });

   put(APIOperation.CREATE_FOLDER, (storage, path, ignored) -> { // Create folder
      logger.info("Executing \"CREATE_FOLDER\" APIOperation...");
      storage.createFolder(path);
```

```
        logger.info("Executed successfully.");
        return new ResponseEntity<>("OK", HttpStatus.OK);
    });
});
```

3.2.4 FileStorage

Экземпляр класса FileStorage содержит путь к директории, с которой он производит операции. FileStorage работает непосредственно с файловой системой.

```
public List<String> getAllPaths(String path) throws IOException {
    logger.info("Getting all paths in storage...");
    path = getFullPath(path);
    List<String> answer;
    if (Files.exists(Paths.get(path))) {
        if (Files.isDirectory(Paths.get(path))) {
            answer = getListOfContent(path, true);
        } else {
            throw new NotADirectory("Requested path does not point to directory.");
        }
    } else {
        throw new NoSuchFileException("No such folder.");
    }
    logger.info("Getting list of all paths handled.");
    return answer;
}
```

3.2.5 Вспомогательные классы

Для обеспечения полной и корректной работы веб-службы в модуле существуют вспомогательные классы.

GlobalExceptionHandler – глобальный обработчик ошибок, отлавливающий исключения выброшенные в пределах зоны ответственности Spring, помечен аннотацией @ControllerAdvice, методы помечены аннотацией @ExceptionHandler.

```
@ExceptionHandler(value = {
        NotADirectory.class,
        NoSuchFileException.class,
        NotAFile.class,
        NoSuchElementException.class})
public ResponseEntity<String> handleBadRequestExceptions(Exception e) {
    logError(e);
    return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
}
```

WebMvcConfing и RequestUrlArgumentResolver необходимы для передачи адреса запроса в параметры методов контроллера помеченные @RequestUrl. По умолчанию Spring не предоставляет такой возможности,

поэтому данные классы имплементируют интерфейсы Spring для отработки действий на уровне сервлетов.

```
@Override
public boolean supportsParameter(MethodParameter methodParameter) {
  return methodParameter.getParameterAnnotation(RequestUrl.class) != null;
}

@Override
public Object resolveArgument(
    MethodParameter methodParameter,
    ModelAndViewContainer modelAndViewContainer,
    NativeWebRequest nativeWebRequest,
    WebDataBinderFactory webDataBinderFactory) throws Exception {

  HttpServletRequest request
      = (HttpServletRequest) nativeWebRequest.getNativeRequest();

  return request.getRequestURL().toString();
}
```

User, UserInfo, StorageInfo – сущности отражающиеся в базе данных, для этого сами классы и их поля помечены соответствующими аннотациями. В пакете by.vorivoda.matvey.model.dao.entity.repository хранятся интерфейсы расширяющие JpaRepository<T, ID> и помеченные аннотацией @Repository. Репозитории определяют стандартные и специфические запросы для БД.

```
@Id
@Column(name = "ID")
@GeneratedValue
private Long id;

@Column(name = "USERNAME")
private String username;
@Column(name = "ALTERNATIVE_USERNAME")
private String alternativeUsername;
@Column(name = "PASSWORD")
private String password;

@OneToOne(optional = false, cascade = CascadeType.ALL)
@JoinColumn(name = "USER_INFO_ID")
private UserInfo userInfo;
@OneToOne(optional = false, cascade = CascadeType.ALL)
@JoinColumn(name = "STORAGE_INFO_ID")
private StorageInfo storageInfo;
@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "TOKEN_ID")
private SecurityToken token;

public User() {
  id = 0L;
  username = alternativeUsername = password = "";
  setUserInfo(new UserInfo());
  setStorageInfo(new StorageInfo());
}
```

3.3 Структура модуля клиента

Пользовательский интерфейс был написан с помощью JavaFX и соответствующего языка разметки FXML. Интерфейс не содержит графических элементов операционной системы. FXML файлы хранятся в папке «resources». В этой папке также содержатся файлы стилей CSS и ассеты.

Приложение имеет три основных и три вспомогательных окна. Основные окна: «login.fxml» – окно с формой для аутентификации, «registration.fxml» – окно с формой для регистрации, «storageManager.fxml» – главное окно с менеджером хранилища. Вспомогательные окна: окно загрузки, окно с сообщением, окно с запросом о вводе. Для переключения между окнами создано перечисление ApplicationScene, каждый элемент которого хранит путь к соответствующему файлу с разметкой, а также статический метод «switchRoot» в классе с точкой входа в приложение, который осуществляет переключение между окнами.

3.3.1 Spring Boot

Для использования таких преимуществ Spring, как Dependency Injection в приложении JavaFX, необходимо перед запуском инициализировать контекст Spring. Пакет by.vorivoda.matvey.spring.support содержит классы необходимые для интеграции JavaFX и Spring Boot. Класс AbstractApplicationWithSpring расширяет стандартный JavaFX класс Application и переопределяет методы «init» и «stop», для инициализации и закрытия контекста Spring, соответственно. Класс ConfigurationController содержит бины контекста.

```
@Override
public void init() throws Exception {
   Platform.runLater(this::showLoadingScene);
   context = SpringApplication.run(getClass(), savedArgs);
   context.getAutowireCapableBeanFactory().autowireBean(this);
   Platform.runLater(this::closeLoadingScene);
}
```

3.3.2 Controller

Пакет by.vorivoda.matvey.controller содержит классы управляющие интерфейсом. Для создания уникального UI/UX были созданы два графических компонента расширяющие компоненты JavaFX: FolderView и FolderStructureView. Также этот пакет содержит специфичный класс CurrentOS, который определяет операционную систему и позволяет запускать файлы в ней.

```
public static boolean open(File file) {
   try {
      if (CurrentOS.isWindows()) {
```

```
            openInWinApi(file.getAbsolutePath());
            return true;
        } else if (CurrentOS.isLinux() || CurrentOS.isMac()) {
            Runtime.getRuntime().exec(new String[]{"/usr/bin/open", file.getAbsolutePath()});
            return true;
        } else {
            // Unknown OS, try with desktop
            if (Desktop.isDesktopSupported()) {
                Desktop.getDesktop().open(file);
                return true;
            } else {
                return false;
            }
        }
    } catch (Exception e) {
        e.printStackTrace(System.err);
        return false;
    }
}

public static native void openInWinApi(String path);
```

Данный класс содержит нативный метод, который вызывается при помощи механизмов JNI. Реализация этого метода содержит вызов функции ShellExecute с параметрами, позволяющими открыть файл в ассоциированной с ним программе.

```
#include "CurrentOS.h"
#include <windows.h>

JNIEXPORT void JNICALL
Java_by_vorivoda_matvey_controller_util_operating_system_CurrentOS_openInWinApi
(JNIEnv * env, jclass class, jstring path) {
    const char *nativeString = (*env)->GetStringUTFChars(env, path, 0);
    ShellExecute(NULL, "open", nativeString, NULL, NULL, SW_SHOWNORMAL);
    (*env)->ReleaseStringUTFChars(env, path, nativeString);
}
```

3.3.2 Model

Пакет by.vorivoda.matvey.model содержит бизнес-логику, в частности основные классы – RemoteStorageClient, User, SecurityToken, StorageStateBindings. Вспомогательные классы FilesMethods, FileInfo.

Объекты типа User хранят информацию о текущем аутентифицированном пользователе. В случае истечения срока действия токена данные о пользователе используются для повторной аутентификации.

Объекты типа SecurityToken хранят информацию о текущем токене безопасности.

```
public class SecurityToken {

    private final String value;
    private final Date expiredDate;
```

```
    public SecurityToken(String value, Date expiredDate) {
        this.value = value;
        this.expiredDate = expiredDate;
    }

    public String getValue() {
        return value;
    }

    public Date getExpiredDate() {
        return expiredDate;
    }
}
```

Объекты типа FileInfo хранят информацию о выбранном файле: адрес ресурса, реальный путь к соответствующему временному файлу на компьютере пользователя, размер.

Класс FilesMethods содержит статический методы для работы с файлами.

```
public class FilesMethods {

    public static String getFileExtension(String path) {
        int slashIndex = path.lastIndexOf("/");
        int dotIndex = path.lastIndexOf(".");

        if (slashIndex == path.length()) return "/";
        return dotIndex > -1 && dotIndex > slashIndex ? path.substring(dotIndex) : "";
    }

    public static String getFileName(String path) {
        int slashIndex = path.lastIndexOf("/");
        slashIndex = slashIndex == path.length() ? path.lastIndexOf("/", slashIndex - 1) : slashIndex;

        int dotIndex = path.lastIndexOf(".");
        dotIndex = dotIndex == -1 ? path.length() : dotIndex;

        return path.substring(slashIndex + 1, dotIndex);
    }

    public static String getFullName(String path) {
        int slashIndex = path.lastIndexOf("/");
        slashIndex = slashIndex == path.length() ? path.lastIndexOf("/", slashIndex - 1) : slashIndex;

        return path.substring(slashIndex + 1);
    }
}
```

StorageStateBindings хранит состояние менеджера хранилища. Предназначен для того, чтобы исключить зависимость RemoteStorageClient от JavaFX. Содержит поля типа Property, на значения которых можно устанавливать слушатели, а сами значения связать с внешними по средствам метода «bind».

```
public StorageStateBindings() {
   currentFolder = new SimpleStringProperty();
   currentFolderElements = new SimpleListProperty<>(FXCollections.observableArrayList());
   currentFile = new SimpleStringProperty();
   allPaths = new SimpleListProperty<>(FXCollections.observableArrayList());
}
```

RemoteStorageClient – ядро клиентской части проекта и представляет собой файловый менеджер. Имплементирует интерфейс IRemoteStorageClient, который представляет методы для работы с удалённым файловым хранилищем.

```
@Override
public BigInteger sizeOf(String path) {
   preOperation();
   logger.info("Getting size of resource: " + path + "...");

   HttpHeaders headers = getDefaultHeaders();
   headers.add("Operation-Type", "SIZE");

   ResponseEntity<String> response = exchange(getHttpUrl(path), HttpMethod.GET, headers);

   logger.info("Got: " + response.getBody());
   return response.getBody() == null ? null : new BigInteger(response.getBody());
}
```

Ключевым элементом RemoteStorageClient является RestTemplate – часть Spring, которая позволяет просто отправить запрос и получить ответ.

```
@Bean
public RestTemplate restTemplate(RestTemplateBuilder builder) {
   return builder
         .setConnectTimeout(Duration.ofSeconds(2))
         .setReadTimeout(Duration.ofSeconds(30))
         .build();
}
```

Методы RemoteStorageClient вызывают изменения в состоянии менеджера, например, метод «open» отправляет GET запрос и может получить два различных ответа: если адрес указывает на папку, то список имён элементов в этой папке, или содержание файла, если адрес указывает на файл. Такое поведение трудно заранее предусмотреть в контроллере пользовательского интерфейса, ведь для этого необходимо делать дополнительный запрос о типе ресурса. Вместо этого контроллер устанавливает обработчики на изменение состояния менеджера и метод «open» установив значение в нужное поле StorageStateBinding, вызовет таким образом нужный обработчик. Такая прослойка между Model и Controller позволяет облегчить код, сделать архитектуру гибче и насколько это возможно уменьшить взаимозависимость.

```
@Override
public void open(String path) throws IOException {
    preOperation();
    logger.info("Opening resource: " + path + "...");

    HttpHeaders headers = getDefaultHeaders();

    ResponseEntity<String> response = exchange(getHttpUrl(path), HttpMethod.GET, headers);

    MediaType responseContentType = response.getHeaders().getContentType();
    if (responseContentType != null && responseContentType.equals(MediaType.APPLICATION_JSON)) {
        ObjectMapper mapper = new ObjectMapper();
        String[] folderElements = mapper.readValue(response.getBody(), String[].class);
        state.setCurrentFolderElements(Arrays.asList(folderElements));
        state.currentFolderProperty().set(path);
    } else {
        BigInteger size = response.getBody() == null
                ? BigInteger.ZERO
                : BigInteger.valueOf(response.getBody().length());

        Path temporaryFile = createTemporaryFile(path);
        try {
            if (response.getBody() != null)
                Files.writeString(temporaryFile, response.getBody());
        } catch (IOException e) {
            Files.delete(temporaryFile);
            throw e;
        }

        addTemporaryFile(temporaryFile.toAbsolutePath().toString());
        state.setCurrentFile(path, new FileInfo(temporaryFile.toAbsolutePath().toString(), path, size));
    }
    logger.info("Opened.");
}
```

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

После проектирования и разработки программного средства необходимо провести тестирование основных функций и возможностей удаленного хранилища файлов на платформе Windows 10. Были проведены следующие тесты:

1. Инициализация Spring может проходить достаточно долго и даже при текущем объёме проекта данный процесс занимает около секунды, поэтому чтобы пользователь знал, что приложение запустилось на экране в первую очередь появляется окно загрузки

Результат: При запуске приложения сначала показывается окно загрузки, после чего стартовое окно приложения.

Тест прошел успешно (рисунок 4.1).

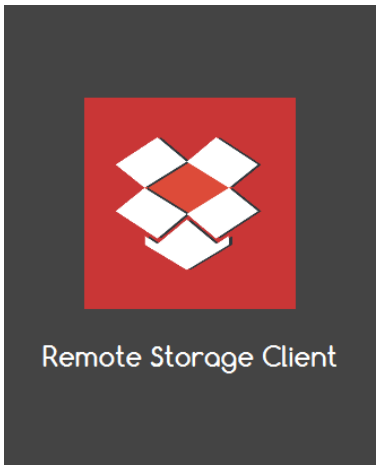

Рисунок 4.1 – окно загрузки

2. При нажатии соответствующих кнопок окно «Sign in» должно переключаться на окно «Sing up» и обратно.

Результат: При запуске приложения отображается окно «Sign in» после чего его можно переключить на окно «Sign up» и обратно.

Тест прошел успешно (рисунок 4.2).

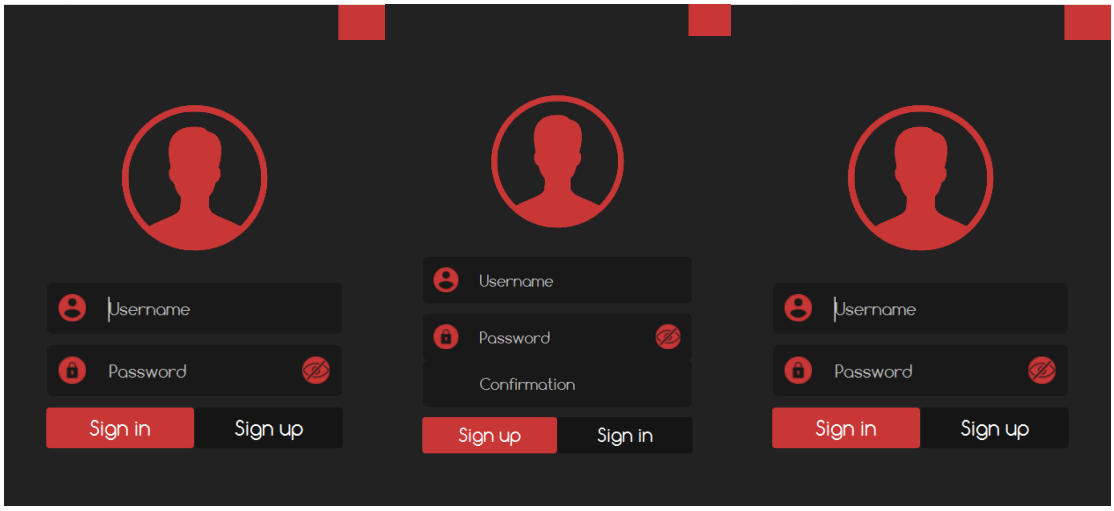

Рисунок 4.2 – окно «Sign in», окно «Sign up», окно «Sign in»

3. При попытке регистрации с неправильным именем пользователя или паролем выводится сообщение об ошибке.

Результат: При попытке отправить на регистрацию слишком короткое имя пользователя и пароль выводится соответствующая ошибка.

Тест прошел успешно (рисунок 4.3).

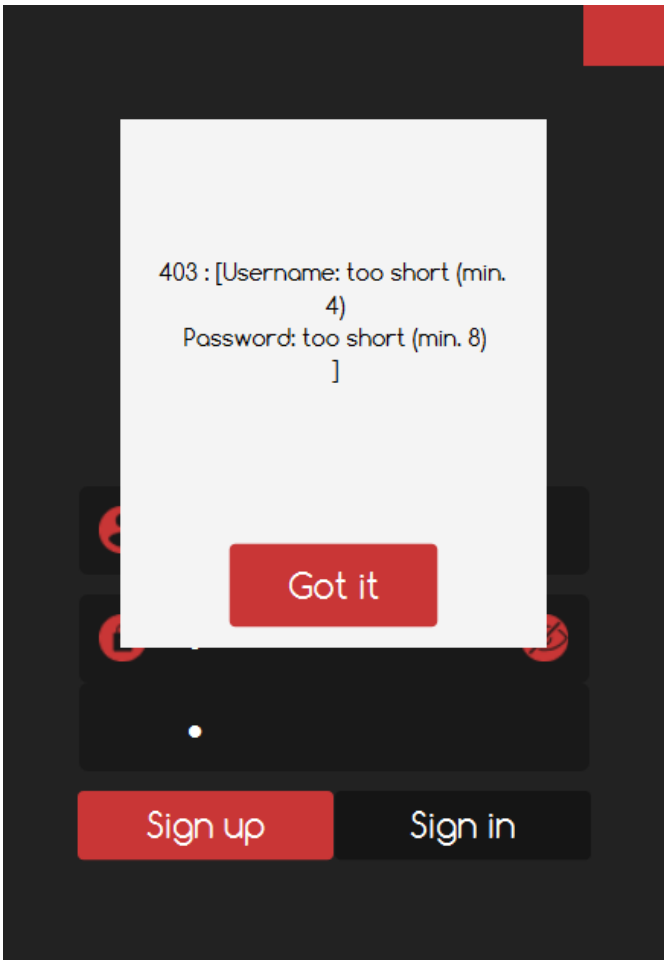

Рисунок 4.3 – окно регистрации с ошибкой

4. При введении правильного имени пользователя и пароля должно открываться окно с менеджером удалённого хранилища.

Результат: При введении в поле «Username» значения «Dewey» и в поле «Password» значения «password» открывается окно с менеджером удалённого хранилища.

Тест прошел успешно (рисунок 4.4).

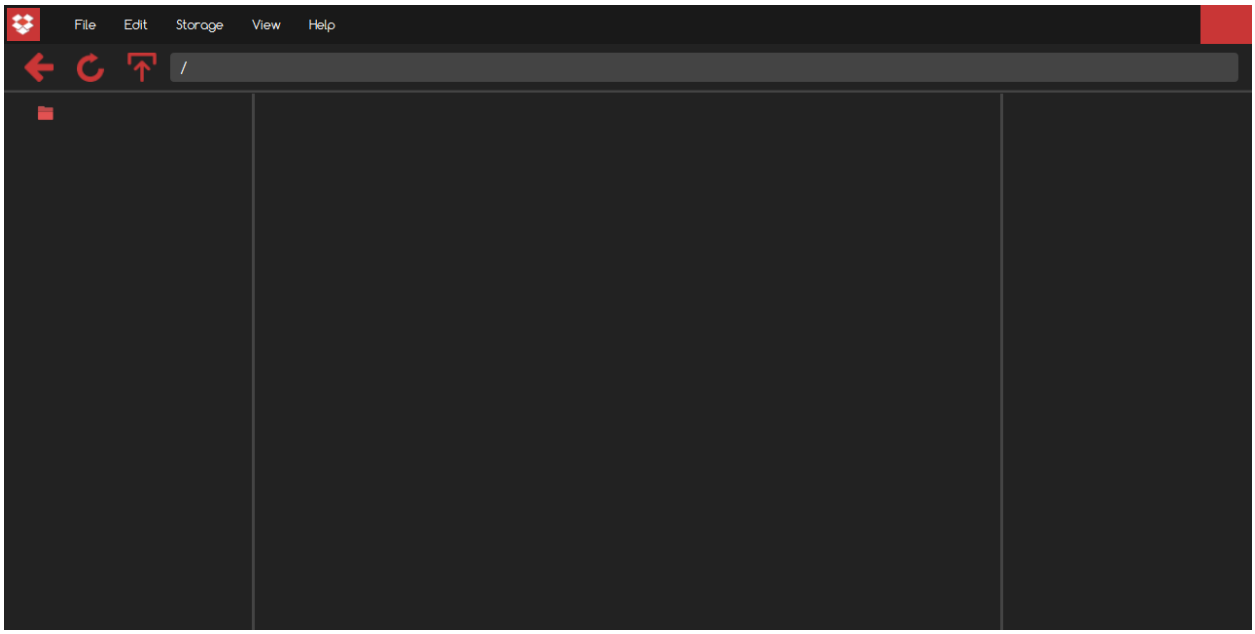

Рисунок 4.4 – окно с менеджером удалённого хранилища

5. В приложении предусмотрены горячие клавиши, например, при нажатии сочетания «Ctrl+N» должен появится запрос на имя папки и после ввода можно создать новую папку.

Результат: За нажатием «Ctrl+N» последовало появление окна с текстовым полем и после ввода с подтверждением была создана новая папка в удалённом файловом хранилище.

Тест прошел успешно (рисунок 4.5, 4.6).

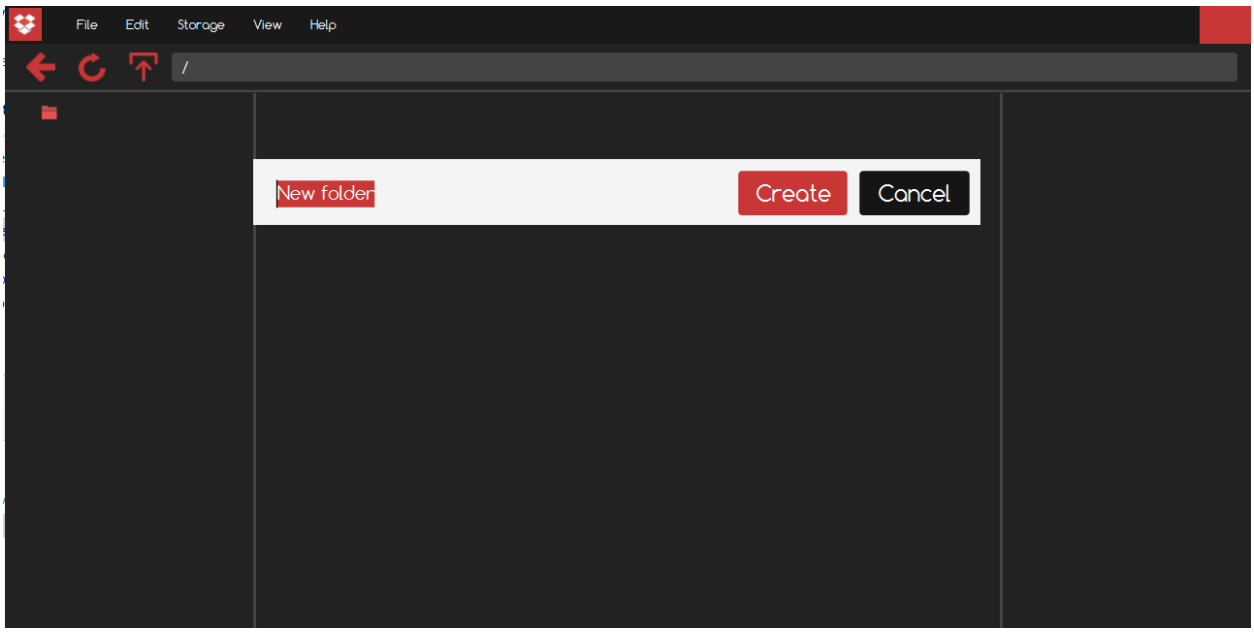

Рисунок 4.5 – запрос с именем новой папки

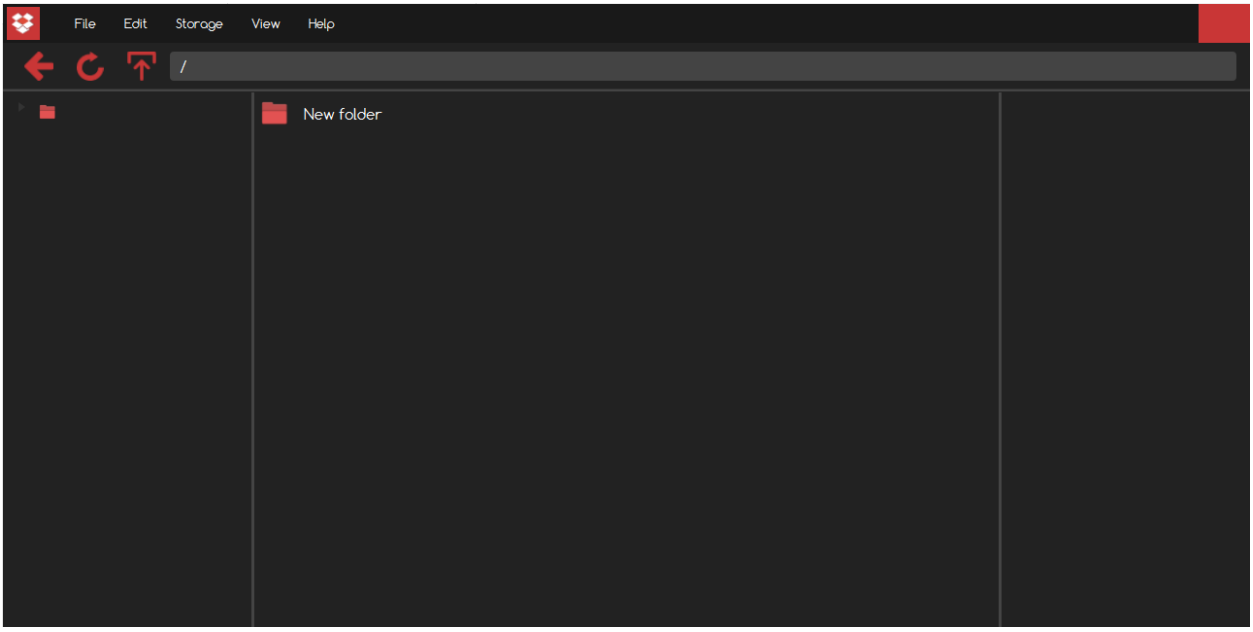

Рисунок 4.6 – папка создана

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Начало работы

Для использования удалённого файлового хранилища необходима работа двух программных средств – серверной веб-службы и клиентского приложения. Работа веб-службы подразумевается на постоянной основе после развертывания на сервере. Руководство пользователя относится только к клиентскому приложению.

5.2 Запуск сервера

Веб-служба запускается единожды – при развертывании, информация о процессе запуска службы логгируется (Рисунок 5.1).

Рисунок 5.1 – вывод в консоль информации о процессе запуска

5.3 Запуск клиента

Как уже было сказано работа серверной части подразумевается на постоянной основе, поэтому всё, что нужно сделать пользователю это пройти регистрацию и войти в свой аккаунт, введя правильные имя пользователя и пароль (Рисунок 5.2). После чего пользователь получает полный доступ к своему хранилищу до перезапуска приложения (Рисунок 5.3).

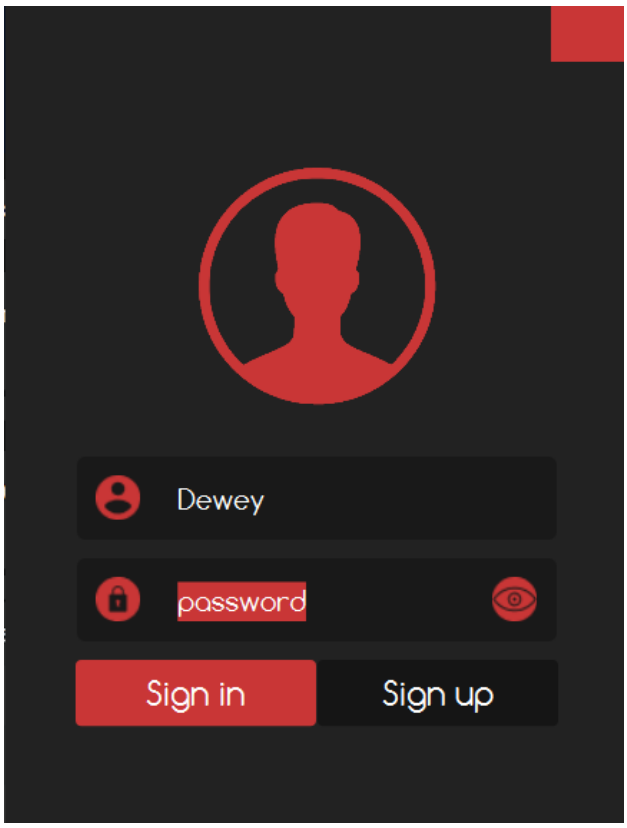

Рисунок 5.2 – окно входа

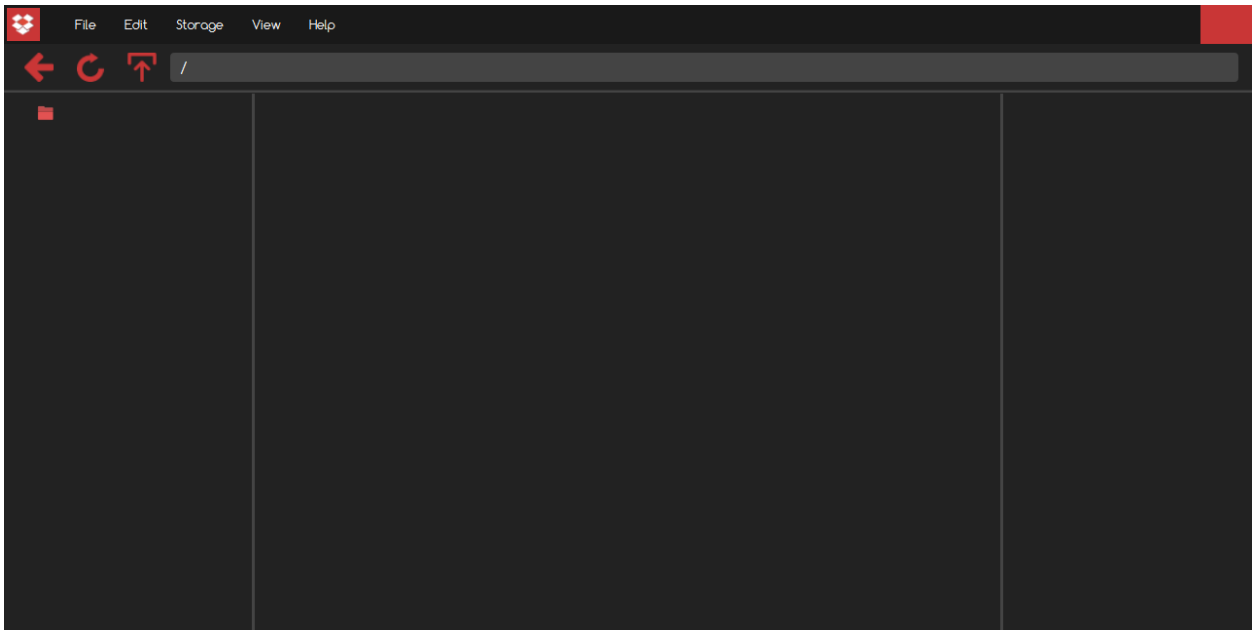

Рисунок 5.3 – окно менеджера

Пользователю доступно большое количество операций с хранилищем: создание папки, загрузка и скачивание файлов, копирование, перемещение, удаление, получение размера. Есть несколько способов ввода: через

контекстное меню, главное меню, горячие клавиши и для некоторых операций выделены отдельные кнопки.

По центру расположен обозреватель текущей открытой папки, слева находиться структура хранилища, а справа обозреватель текущего выбранного файла: его можно скачать или открыть в системе пользователя (Рисунок 5.5).

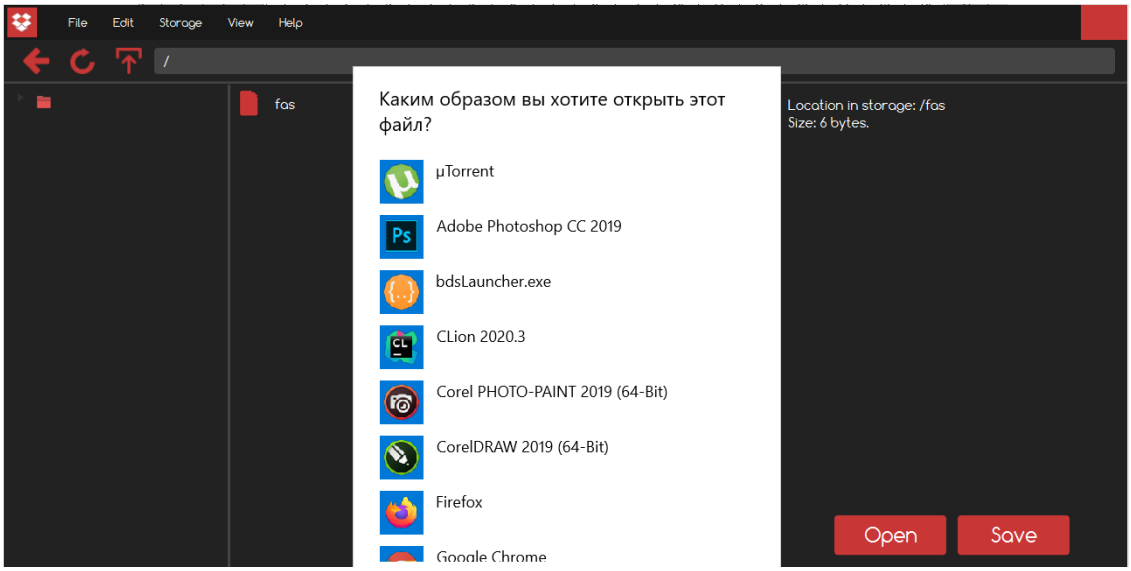

Рисунок 5.5 – открытие файла на компьютере пользователя

ЗАКЛЮЧЕНИЕ

Разработка данного приложения требовала познаний в области архитектуры клиент-сервер, понимание взаимодействия клиента и сервера, HTTP-протоколов, паттерна проектирования «наблюдатель», архитектурного стиля REST.

В ходе разработки были изучены возможности визуальной среды разработки IntelliJ IDEA, получены навыки создания графического пользовательского интерфейса с использованием JavaFx, Spring, maven, FXML.

В результате выполнения данной курсовой работы разработано удаленное файловое хранилище.

Приложение имеет возможность совершенствования функционала и расширения возможностей. Со временем в данное приложение будут добавлены шифрование, протокол HTTPs, возможность просматривать файлы, авторизация с помощью почты. Неотъемлемым плюсом разработки является огромное количество возможностей для обновления и улучшения хранилища, которые будут использованы в скором будущем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кэти Сьерра, Берт Бэйтс: Head First Java, Изучаем Java. - М.: Эксмо, 2015
2. Роберт Лафоре: Структуры данных и алгоритмы Java. - М.: Питер Мейл, 2018
3. J.F DiMarzio: Quick Start Guide to JavaFx - M.: Oracle, 2014
4. Руководства JavaFx. - (электронный ресурс). Электронные данные. Режим доступа: https://o7planning.org/ru/11009/javafx
5. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2018. – 464 с.: ил.
6. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003.
7. Уилсон, С. Принципы проектирования и разработки программного обеспечения, учебн. курс. – СПб, 2003.

ПРИЛОЖЕНИЕ А

Исходный код программы

```
    package by.vorivoda.matvey.controller;

import by.vorivoda.matvey.ApplicationScene;
import by.vorivoda.matvey.ClientApplication;
import by.vorivoda.matvey.controller.component.folder.structure.view.FolderStructureView;
import by.vorivoda.matvey.controller.component.folder.view.FolderView;
import by.vorivoda.matvey.controller.util.operating.system.CurrentOS;
import by.vorivoda.matvey.model.IRemoteStorageClient;
import by.vorivoda.matvey.model.RemoteStorageClient;
import by.vorivoda.matvey.model.StorageStateBindings;
import by.vorivoda.matvey.model.util.FilesMethods;
import javafx.application.Platform;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyCombination;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class StorageFXController extends CommonController {

   private static final String ABOUT_PROGRAM = "Remote storage can save your files in a safe place.";
   private static final String ABOUT_AUTHOR = "Vorivoda Matvey (Student of group 951007, BSUIR -
Belarus)";

   @FXML
   private SplitPane splitPane;

   @FXML
   private AnchorPane structurePane;

   @FXML
   private TreeView<HBox> storageStructure;

   @FXML
   private ScrollPane folderElementsScrollBar;

   @FXML
   private VBox folderElements;

   @FXML
```

```
private AnchorPane filePreviewPane;

@FXML
private VBox fileInfoVBox;

@FXML
private Button openFileBtn;

@FXML
private Button saveFileBtn;

@FXML
private AnchorPane closeBtn;

@FXML
private HBox titleHBox;

@FXML
private MenuItem mmOpen;

@FXML
private MenuItem mmSave;

@FXML
private MenuItem mmExit;

@FXML
private MenuItem mmCopy;

@FXML
private MenuItem mmMove;

@FXML
private MenuItem mmPaste;

@FXML
private MenuItem mmDelete;

@FXML
private MenuItem mmNewFolder;

@FXML
private MenuItem mmUpload;

@FXML
private MenuItem mmRefresh;

@FXML
private MenuItem mmBack;

@FXML
private CheckMenuItem mmShowStructure;

@FXML
private CheckMenuItem mmShowFilePreview;

@FXML
private MenuItem mmAboutProgram;

@FXML
```

```
private MenuItem mmAboutAuthor;

@FXML
private VBox backBtn;

@FXML
private VBox refreshBtn;

@FXML
private VBox uploadBtn;

@FXML
private TextField currentFolderPath;

private IRemoteStorageClient storage;
private StorageStateBindings storageState;
private StringProperty folderCreatingName;
private StringProperty src;
private final double structureShowingOnThisDividerPosition = 0.2;
private final double previewShowingOnThisDividerPosition = 0.75;
private final double widthOfOpenedPane = 150;

private enum COPY_MOVE {NOTHING, COPY, MOVE}

private COPY_MOVE currentCopyMoveState;

@FXML
void initialize() {
   folderCreatingName = new SimpleStringProperty();
   src = new SimpleStringProperty();
   currentCopyMoveState = COPY_MOVE.NOTHING;

   filePreviewPane.setVisible(false);

   storage = (RemoteStorageClient) ClientApplication.getContext().getBean("StorageClient");
   storageState = storage.getState();

   folderElements.prefWidthProperty().bind(folderElementsScrollBar.widthProperty().multiply(0.9));
   folderElements.prefHeightProperty().bind(folderElementsScrollBar.heightProperty().multiply(0.9));

   currentFolderPath.textProperty().bind(storageState.currentFolderProperty());

   FolderView folderView = new FolderView(folderElements);
   folderView.currentElementsProperty().bind(storageState.currentFolderElementsProperty());

   FolderStructureView folderStructureView = new FolderStructureView(storageStructure);
   folderStructureView.currentElementsProperty().bind(storageState.allPathsProperty());

   currentFolderPath.focusedProperty().addListener((observableValue, oldValue, newValue) -> {
      if (newValue) {
         currentFolderPath.textProperty().unbind();
      } else {
         if (!currentFolderPath.getText().equals(storageState.getCurrentFolder())) {
            storageState.setCurrentFolder(currentFolderPath.getText());
         }
         currentFolderPath.textProperty().bind(storageState.currentFolderProperty());
      }
   });

   currentFolderPath.setOnKeyPressed(keyEvent -> {
```

```
    if (keyEvent.getCode() == KeyCode.ENTER)
        refreshBtn.requestFocus();
});

EventHandler<MouseEvent> refreshEvent = mouseEvent -> {
    try {
        storage.open(storageState.getCurrentFolder());
    } catch (IOException e) {
        e.printStackTrace();
    }
};

EventHandler<MouseEvent> backEvent = mouseEvent -> {
    try {
        String current = storageState.getCurrentFolder();
        if (current.lastIndexOf("/") == 0) {
            current = "/";
        } else {
            current = current.substring(0, current.lastIndexOf("/"));
        }
        storage.open(current);
    } catch (IOException e) {
        e.printStackTrace();
    }
};

EventHandler<MouseEvent> storageStructureClickEvent = mouseEvent -> {
    try {
        if (mouseEvent.getButton() == MouseButton.SECONDARY)
            storage.refreshAllPaths();
    } catch (IOException e) {
        e.printStackTrace();
    }
};

EventHandler<MouseEvent> uploadRequest = mouseEvent -> {
    FileChooser fc = new FileChooser();
    fc.setTitle("Choose file for uploading");
    File file = fc.showOpenDialog(uploadBtn.getScene().getWindow());

    if (file == null) return;

    try {
        storage.uploadFile(storage.getStoragePath(file.getName()), file);
    } catch (IOException e) {
        e.printStackTrace();
    }
};

EventHandler<ActionEvent> openFileRequest = actionEvent -> {
    boolean isOpened = CurrentOS.open(new File(storageState.getCurrentFileInfo().getRealPath()));
    if (!isOpened) alert("Error when opening file.", openFileBtn.getScene().getWindow());
};

EventHandler<ActionEvent> saveFileRequest = actionEvent -> {
    FileChooser fc = new FileChooser();
    fc.setTitle("Choose safe place :)");
    File file = fc.showSaveDialog(saveFileBtn.getScene().getWindow());
    if (file == null) return;
```

```
            try {
                Files.write(
                        file.toPath(),
                        Files.readAllBytes(Paths.get(storageState.getCurrentFileInfo().getRealPath()))
                );
            } catch (IOException e) {
                e.printStackTrace();
            }
        };

        storageState.currentFileProperty().addListener((observableValue, oldValue, newValue) -> {
            fileInfoVBox.getChildren().clear();
            fileInfoVBox.getChildren().add(new Label("Location in storage: " +
storageState.getCurrentFileInfo().getResourcePath()));
            fileInfoVBox.getChildren().add(new Label("Size: " + storageState.getCurrentFileInfo().getSize() + "
bytes."));
            // TODO Resource Preview

            filePreviewPane.setVisible(true);
        });

        EventHandler<ActionEvent> createFolderRequest = actionEvent -> {
            showModal(folderCreatingName, ApplicationScene.FOLDER_NAME_REQUESTER,
folderElements.getScene().getWindow());
            if (folderCreatingName.get().length() == 0) return;

            storage.createFolder(storage.getStoragePath(folderCreatingName.get()));
        };

        EventHandler<ActionEvent> folderPasteRequest = actionEvent -> {
            if(currentCopyMoveState != COPY_MOVE.NOTHING)
                pasteRequestHandled(storage.getStoragePath(FilesMethods.getFullName(src.get())));
        };

        ContextMenu folderViewMenu = new ContextMenu();
        folderViewMenu.setStyle("-fx-background-color: #000");
        folderViewMenu.setOnCloseRequest(windowEvent -> folderViewMenu.hide());

        MenuItem createFolderItem = new MenuItem("Create folder");
        createFolderItem.setOnAction(createFolderRequest);
        folderViewMenu.getItems().add(createFolderItem);

        MenuItem uploadFileItem = new MenuItem("Upload file");
        uploadFileItem.setOnAction(actionEvent -> uploadRequest.handle(null));
        folderViewMenu.getItems().add(uploadFileItem);

        MenuItem pasteItem = new MenuItem("Paste");
        pasteItem.setOnAction(actionEvent ->
                folderPasteRequest.handle(null));
        folderViewMenu.getItems().add(pasteItem);

        folderElements.setOnContextMenuRequested(contextMenuEvent -> {
            if (contextMenuEvent.getSource().equals(contextMenuEvent.getTarget()))
                folderViewMenu.show(folderElements,
                        contextMenuEvent.getSceneX() + folderElements.getScene().getWindow().getX(),
                        contextMenuEvent.getSceneY() + folderElements.getScene().getWindow().getY());
        });
        folderViewMenu.setOnCloseRequest(windowEvent -> folderViewMenu.hide());

        folderView.setOnMouseClicked((mouseEvent, name) -> {
```

```
            try {
                if (mouseEvent.getButton() == MouseButton.PRIMARY)
                    if(mouseEvent.getClickCount() == 2) {
                        storage.open(storage.getStoragePath(name));
                        openFileRequest.handle(null);
                    } else {
                        storage.open(storage.getStoragePath(name));
                    }
            } catch (IOException e) {
                e.printStackTrace();
            }
        });

        refreshBtn.setOnMouseClicked(refreshEvent);
        backBtn.setOnMouseClicked(backEvent);
        storageStructure.setOnMouseClicked(storageStructureClickEvent);
        saveFileBtn.setOnAction(saveFileRequest);
        openFileBtn.setOnAction(openFileRequest);
        uploadBtn.setOnMouseClicked(uploadRequest);

        folderView.setOnCopyRequest((actionEvent, name) -> {
            currentCopyMoveState = COPY_MOVE.COPY;
            src.set(storage.getStoragePath(name));
        });
        folderView.setOnMoveRequest((actionEvent, name) -> {
            currentCopyMoveState = COPY_MOVE.MOVE;
            src.set(storage.getStoragePath(name));
        });
        folderView.setOnPasteRequest((actionEvent, name) ->
pasteRequestHandled(storageState.getCurrentFolder() + "/" + name));
        folderView.setOnDeleteRequest((actionEvent, name) -> storage.delete(storage.getStoragePath(name)));
        folderView.setOnOpenRequest((actionEvent, name) -> {
            try {
                storage.open(storage.getStoragePath(name));
                openFileRequest.handle(actionEvent);
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        folderView.setOnSaveRequest((actionEvent, name) -> {
            try {
                storage.open(storage.getStoragePath(name));
                saveFileRequest.handle(actionEvent);
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        folderView.setOnSizeRequest((actionEvent, name) -> {
            alert("Size of " + name + ": " + storage.sizeOf(storage.getStoragePath(name)) + "",
folderElements.getScene().getWindow());
        });

    // ---------- MENU INITIALIZING ----------

        mmOpen.setOnAction(openFileRequest);
        mmSave.setOnAction(saveFileRequest);
        mmExit.setOnAction(actionEvent -> Platform.exit());
        mmCopy.setOnAction(actionEvent -> {
            String currentFile = storageState.getCurrentFile();
            if (currentFile != null && currentFile.length() > 0) {
```

```
            currentCopyMoveState = COPY_MOVE.COPY;
            src.set(storageState.getCurrentFileInfo().getResourcePath());
        }
    });
    mmMove.setOnAction(actionEvent -> {
        String currentFile = storageState.getCurrentFile();
        if (currentFile != null && currentFile.length() > 0) {
            currentCopyMoveState = COPY_MOVE.MOVE;
            src.set(storageState.getCurrentFileInfo().getResourcePath());
        }
    });
    mmPaste.setOnAction(folderPasteRequest);
    mmDelete.setOnAction(actionEvent -> {
        String currentFile = storageState.getCurrentFile();
        if (currentFile != null && currentFile.length() > 0)
            storage.delete(storageState.getCurrentFileInfo().getResourcePath());
    });
    mmNewFolder.setOnAction(createFolderItem.getOnAction());
    mmUpload.setOnAction(uploadFileItem.getOnAction());
    mmRefresh.setOnAction(actionEvent -> refreshEvent.handle(null));
    mmBack.setOnAction(actionEvent -> backEvent.handle(null));
    mmAboutProgram.setOnAction(actionEvent -> alert(ABOUT_PROGRAM,
folderElements.getScene().getWindow()));
    mmAboutAuthor.setOnAction(actionEvent -> alert(ABOUT_AUTHOR,
folderElements.getScene().getWindow()));

    mmOpen.setAccelerator(KeyCombination.keyCombination("Ctrl+O"));
    mmSave.setAccelerator(KeyCombination.keyCombination("Ctrl+S"));

    mmCopy.setAccelerator(KeyCombination.keyCombination("Ctrl+C"));
    mmMove.setAccelerator(KeyCombination.keyCombination("Ctrl+X"));
    mmPaste.setAccelerator(KeyCombination.keyCombination("Ctrl+V"));
    mmDelete.setAccelerator(KeyCombination.keyCombination("Delete"));

    mmNewFolder.setAccelerator(KeyCombination.keyCombination("Ctrl+N"));
    mmUpload.setAccelerator(KeyCombination.keyCombination("Ctrl+U"));
    mmRefresh.setAccelerator(KeyCombination.keyCombination("Ctrl+R"));
    mmBack.setAccelerator(KeyCombination.keyCombination("Esc"));

    mmShowStructure.setAccelerator(KeyCombination.keyCombination("Ctrl+Shift+S"));
    mmShowFilePreview.setAccelerator(KeyCombination.keyCombination("Ctrl+Shift+F"));
    mmAboutProgram.setAccelerator(KeyCombination.keyCombination("Ctrl+P"));
    mmAboutAuthor.setAccelerator(KeyCombination.keyCombination("Ctrl+A"));

    mmShowStructure.selectedProperty().addListener((observableValue, oldValue, newValue) ->
        splitPane.setDividerPosition(0, newValue ? structureShowingOnThisDividerPosition : 0.0));
    structurePane.widthProperty().addListener((observableValue, oldValue, newValue) ->
        mmShowStructure.setSelected(newValue.doubleValue() > widthOfOpenedPane)
    );

    mmShowFilePreview.selectedProperty().addListener((observableValue, oldValue, newValue) ->
        splitPane.setDividerPosition(1, newValue ? previewShowingOnThisDividerPosition : 1.0)
    );
    filePreviewPane.widthProperty().addListener((observableValue, oldValue, newValue) ->
        mmShowFilePreview.setSelected(newValue.doubleValue() > widthOfOpenedPane)
    );

    mmShowStructure.setSelected(true);
    mmShowFilePreview.setSelected(true);
```

```
// ---------- END MENU INITIALIZING ----------

        try {
            storage.open("/");
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            storage.refreshAllPaths();
        } catch (IOException e) {
            e.printStackTrace();
        }
        setCloseBtn(closeBtn);
        setWindowGrabber(titleHBox);
    }

    private void pasteRequestHandled(String dest) {
        switch (currentCopyMoveState) {
            case COPY:
                storage.copy(src.get(), dest);
                break;
            case MOVE:
                storage.move(src.get(), dest);
                currentCopyMoveState = COPY_MOVE.NOTHING;
                break;
        }
    }
}
package by.vorivoda.matvey.model;

import by.vorivoda.matvey.model.security.SecurityToken;
import by.vorivoda.matvey.model.security.User;
import by.vorivoda.matvey.model.util.FileInfo;
import by.vorivoda.matvey.model.util.FilesMethods;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.http.*;
import org.springframework.stereotype.Component;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

import java.io.File;
import java.io.IOException;
import java.math.BigInteger;
import java.net.URI;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;

@Component("StorageClient")
public class RemoteStorageClient implements IRemoteStorageClient {
```

```
private final Logger logger = LoggerFactory.getLogger(RemoteStorageClient.class);
private final RestTemplate requester;
private final String serverAddress;

private User currentUser;
private SecurityToken currentToken;
private final StorageStateBindings state;

private final List<String> temporaryFiles;
private @Value("${maximum.temp.files}")
int maxTempFiles;

@Autowired
public RemoteStorageClient(RestTemplate requester,
                    @Value("${server.address}") String serverAddress) {
   this.requester = requester;
   this.serverAddress = serverAddress;
   temporaryFiles = new ArrayList<>();
   state = new StorageStateBindings();

   state.currentFolderProperty().addListener((observable, oldValue, newValue) -> {
      try {
         open(newValue);

         if (newValue.equals("/")) return;
         if (newValue.endsWith("/")) newValue = newValue.substring(0, newValue.length() - 1);
         if (!isDirectory(newValue)) {
            String parent = newValue.substring(0, newValue.lastIndexOf("/"));
            parent = parent.length() == 1 ? parent : parent.substring(0, parent.length() - 1);
            state.setCurrentFolder(parent);
         }
      } catch (Exception e) {
         state.setCurrentFolder(oldValue);
      }
   });
}

private boolean isDirectory(String path) {
   if (path.equals("/")) return true;
   for (String element : state.getAllPaths()) {
      if (element.equals(path + "/")) return true;
   }
   return false;
}

@Override
public Map<String, String> registration(String username, String password) throws IOException {
   logger.info("Attempting to register user with username \"" + username + "\"...");

   HttpHeaders headers = new HttpHeaders();
   headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

   MultiValueMap<String, String> parameters = new LinkedMultiValueMap<>();
   parameters.add("username", username);
   parameters.add("password", encode(password));

   try {
      HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(parameters, headers);
      ResponseEntity<String> response = requester.postForEntity(serverAddress + "/registration", request,
```

```
String.class);
        logger.debug(response.toString());

        String errorReport = response.getBody();
        Map<String, String> errors = parseErrorReport(errorReport);

        if (errors.size() > 0) logger.error("Errors occurred: " + errorReport);
        else logger.info("Registered.");

        return errors;
      } catch (Exception e) {
        throw new IOException(e.getMessage());
      }
    }

    private static Map<String, String> parseErrorReport(String errorReport) {
      if (errorReport == null || errorReport.length() == 0) return new HashMap<>();

      errorReport = errorReport.toLowerCase(Locale.ROOT);
      Scanner scanner = new Scanner(errorReport);

      Map<String, String> errors = new HashMap<>();
      while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        String key = line.substring(0, line.indexOf(":"));
        String value = line.substring(line.indexOf(":") + 1).trim();

        errors.merge(key, value, (currentErrors, newError) -> currentErrors + "; " + newError);
      }

      return errors;
    }

    private String encode(String password) {
      logger.info("Encoding password...");
      char[] encoded = new char[password.length()];

      for (int i = 0; i < encoded.length; i++) {
        encoded[i] = (char) ((password.charAt(i) * (i + 1)) % 31);
      }

      logger.info("Password " + password + " encoded to " + String.valueOf(encoded));
      return String.valueOf(encoded);
    }

    @Override
    public boolean signIn(String username, String password) {
      logger.info("Attempting to sign in user with username \"" + username + "\"...");

      HttpHeaders headers = new HttpHeaders();
      headers.setBasicAuth(username, encode(password));

      HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(headers);
      logger.info(request.toString());
      try {
        ResponseEntity<String> response = requester.postForEntity(serverAddress + "/login", request,
String.class);
        logger.debug(response.toString());

        if (response.getBody() == null) return false;
```

```
            if (response.getStatusCode().is2xxSuccessful()) {
                logger.info("Signed in.");
                String[] tokenFields = response.getBody().split(" ", 2);
                currentToken = new SecurityToken(tokenFields[1], new Date(Long.parseLong(tokenFields[0])));
                currentUser = new User(username, password);
                return true;
            }
            return false;
        } catch (Exception e) {
            logger.info("Error: " + e.getMessage());
            return false;
        }
    }

    @Override
    public void open(String path) throws IOException {
        preOperation();
        logger.info("Opening resource: " + path + "...");

        HttpHeaders headers = getDefaultHeaders();

        ResponseEntity<String> response = exchange(getHttpUrl(path), HttpMethod.GET, headers);

        MediaType responseContentType = response.getHeaders().getContentType();
        if (responseContentType != null && responseContentType.equals(MediaType.APPLICATION_JSON)) {
            ObjectMapper mapper = new ObjectMapper();
            String[] folderElements = mapper.readValue(response.getBody(), String[].class);
            state.setCurrentFolderElements(Arrays.asList(folderElements));
            state.currentFolderProperty().set(path);
        } else {
            BigInteger size = BigInteger.ZERO;

            Path temporaryFile = createTemporaryFile(path);
            try {
                if (response.getBody() != null) {
                    byte[] bytes = Base64.getDecoder().decode(response.getBody());
                    Files.write(temporaryFile, bytes);
                    size = BigInteger.valueOf(bytes.length);
                }
            } catch (IOException e) {
                Files.delete(temporaryFile);
                throw e;
            }

            addTemporaryFile(temporaryFile.toAbsolutePath().toString());
            state.setCurrentFile(path, new FileInfo(temporaryFile.toAbsolutePath().toString(), path, size));
        }
        logger.info("Opened.");
    }

    @Override
    public void refreshAllPaths() throws IOException {
        preOperation();
        logger.info("Refreshing storage...");

        HttpHeaders headers = getDefaultHeaders();
        headers.add("Operation-Type", "GET_ALL");
        ResponseEntity<String> response = exchange(getHttpUrl("/"), HttpMethod.GET, headers);
        ObjectMapper mapper = new ObjectMapper();
        String[] folderElements = mapper.readValue(response.getBody(), String[].class);
```

```
        state.setAllPaths(Arrays.asList(folderElements));
        logger.info("Refreshed.");
    }

    private void addTemporaryFile(String path) {
        while (temporaryFiles.size() > maxTempFiles - 1) {
            try {
                Files.delete(Paths.get(temporaryFiles.get(0)));
            } catch (IOException e) {
                e.printStackTrace();
            }
            temporaryFiles.remove(0);
        }

        temporaryFiles.add(path);
    }

    private Path createTemporaryFile(String path) throws IOException {
        return Files.createTempFile(FilesMethods.getFileName(path), FilesMethods.getFileExtension(path));
    }

    @Override
    public BigInteger sizeOf(String path) {
        preOperation();
        logger.info("Getting size of resource: " + path + "...");

        HttpHeaders headers = getDefaultHeaders();
        headers.add("Operation-Type", "SIZE");

        ResponseEntity<String> response = exchange(getHttpUrl(path), HttpMethod.GET, headers);

        logger.info("Got: " + response.getBody());
        return response.getBody() == null ? null : new BigInteger(response.getBody());
    }

    private ResponseEntity<String> exchange(String url, HttpMethod method, HttpHeaders headers) {
        return exchange(url, method, headers, "");
    }

    private ResponseEntity<String> exchange(String url, HttpMethod method, HttpHeaders headers, String
body) {
        RequestEntity<String> request = new RequestEntity<>("=" + body, headers, method, URI.create(url));
        logger.info(request.toString());
        ResponseEntity<String> response = requester.exchange(request, String.class);
        logger.debug(response.toString());
        return response;
    }

    @Override
    public void uploadFile(String path, File file) throws IOException {
        uploadFile(path, Files.readAllBytes(file.toPath()));
    }

    @Override
    public void uploadFile(String path, byte[] content) {
        uploadFile(path, Base64.getEncoder().encodeToString(content));
    }

    @Override
    public void uploadFile(String path, String content) {
```

```
    preOperation();
    logger.info("Uploading file on path: " + path + "...");
    HttpHeaders headers = getDefaultHeaders();
    exchange(getHttpUrl(path), HttpMethod.PUT, headers, content);
    logger.info("Uploaded.");
    postOperation();
}

@Override
public void appendFile(String path, File file) throws IOException {
    appendFile(path, Files.readAllBytes(file.toPath()));
}

@Override
public void appendFile(String path, byte[] content) {
    appendFile(path, Base64.getEncoder().encodeToString(content));
}

@Override
public void appendFile(String path, String content) {
    preOperation();
    logger.info("Appending file on path: " + path + "...");
    HttpHeaders headers = getDefaultHeaders();
    if (content != null && content.length() > 0)
        exchange(getHttpUrl(path), HttpMethod.POST, headers, content);
    logger.info("Appended.");
    postOperation();
}

@Override
public void createFolder(String path) {
    preOperation();
    path = getValidateDestination(path);
    logger.info("Creating folder on path: " + path + "...");
    HttpHeaders headers = getDefaultHeaders();
    headers.add("Operation-Type", "CREATE_FOLDER");
    exchange(getHttpUrl(path), HttpMethod.PUT, headers);
    logger.info("Created.");
    postOperation();
}

@Override
public void move(String src, String dest) {
    preOperation();
    dest = getValidateDestination(dest);
    logger.info("Moving element from " + src + " to " + dest + "...");
    HttpHeaders headers = getDefaultHeaders();
    headers.add("Additional", dest);
    headers.add("Operation-Type", "MOVE");
    exchange(getHttpUrl(src), HttpMethod.GET, headers);
    logger.info("Moved.");
    postOperation();
}

@Override
public void copy(String src, String dest) {
    preOperation();
    dest = getValidateDestination(dest);
    logger.info("Copying element from " + src + " to " + dest + "...");
    HttpHeaders headers = getDefaultHeaders();
```

```
    headers.add("Additional", dest);
    headers.add("Operation-Type", "COPY");
    exchange(getHttpUrl(src), HttpMethod.GET, headers);
    logger.info("Copied.");
    postOperation();
}

@Override
public void delete(String path) {
    preOperation();
    logger.info("Deleting element on path: " + path + "...");
    HttpHeaders headers = getDefaultHeaders();
    exchange(getHttpUrl(path), HttpMethod.DELETE, headers);
    logger.info("Delete.");
    postOperation();
}

private HttpHeaders getDefaultHeaders() {
    HttpHeaders headers = new HttpHeaders();
    headers.setBearerAuth(currentToken.getValue());
    return headers;
}

private void preOperation() {
    final long timeGapToExpired = 20000;
    if(currentToken == null
            || new Date().getTime() + timeGapToExpired > currentToken.getExpiredDate().getTime())
        refreshToken();
}

private void refreshToken() {
    signIn(currentUser.getUsername(), currentUser.getPassword());
}

private void postOperation() {
    logger.info("Finalizing...");
    try {
        open(state.getCurrentFolder());
        refreshAllPaths();
    } catch (IOException e) {
        e.printStackTrace();
    }
    logger.info("Finalized.");
}

@Qualifier("StorageState")
@Bean
public StorageStateBindings getState() {
    return state;
}

@Override
public String getStoragePath(String name) {
    return state.getCurrentFolder().length() == 1 ? "/" + name : state.getCurrentFolder() + "/" + name;
}

private String getHttpUrl(String path) {
    StringBuilder encodedPath = new StringBuilder();
    int prevSlash = 0;
    for (int i = 1; i < path.length() + 1; i++) {
```

```
            if (i == path.length() || path.charAt(i) == '/') {
                encodedPath.append("/").append(URLEncoder.encode(path.substring(prevSlash + 1, i),
StandardCharsets.UTF_8));
                prevSlash = i;
            }
        }

        return serverAddress + "/storage" + encodedPath;
    }

    private String getValidateDestination(String dest) {
        int dotIndex = dest.lastIndexOf(".");
        int slashIndex = dest.lastIndexOf("/");
        int suffixInsertIndex = dotIndex;

        if (dotIndex < slashIndex) {
            suffixInsertIndex = slashIndex == dest.length() ? slashIndex : dest.length();
        }

        String validatedDest = dest;
        int currentIteration = 0;
        while (state.getAllPaths().contains(validatedDest) || state.getAllPaths().contains(validatedDest + "/")) {
            validatedDest = dest.substring(0, suffixInsertIndex) +
                    "(" + currentIteration++ + ")" + dest.substring(suffixInsertIndex);
        }
        return validatedDest;
    }
}
```