МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯРАБОТА

на тему

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ ОБЛАЧНОГО ХРАНЕНИЯ ДАННЫХ НА УСТРОЙСТВАХ ПОД УПРАВЛЕНИЕМ ОС ANDROID

Студент _____ (подпись, дата) Табрисов Степан Андреевич

Руководитель _____ (подпись, дата) Юрова Ольга Викторовна 2.06.2023

Нормоконтролёр _____ (подпись, дата) Бычков Андрей Станиславович 2.06.2023

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 г. № 14)

Заведующий кафедрой _____ (подпись) М. А. Митрохин

Работа защищена с отметкой отлично (протокол заседания ГЭК от 06.06.2023г. № 4)

Секретарь ГЭК _____ (подпись) Е.И. Гурин

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» апреля 2023г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Табрисов Степан Андреевич гр.19ВВП2 факультета вычислительной техники.

2. Тема работы «Программное обеспечение для облачного хранения данных на устройствах под управлением ОС Android»

Тема утверждена приказом ПГУ №458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О. В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение разработки – клиент-серверное приложение для облачного хранения данных
Разработать программное обеспечение для облачного хранения данных на устройствах под управлением ОС Android с использованием клиент-серверной архитектуры. Клиентская часть программного обеспечения должна предоставлять пользователю возможность отправлять, удалять и загружать файлы на сервер. Серверная часть должна осуществлять обработку запросов от пользователя и отправку результата.

5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать инструменты для разработки, реализовать клент-серверное приложение.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10 %	26.04.23 - 29.04.23	
2.	Выбор архитектуры приложения и технологий, применяемых при проектировании и разработке	15 %	30.04.23 - 04.05.23	
3.	Разработка клиентской части	20 %	05.05.23 - 09.05.23	
4.	Разработка серверной части	25 %	10.05.23 - 21.05.23	
5.	Тестирование и отладка	15 %	22.05.23 - 24.05.23	
6.	Выполнение задания по БЖД	5 %	25.05.23 - 27.05.23	
7.	Оформление выпускной работы	10 %	28.05.23 - 31.05.23	

Дата выдачи задания "26" апреля 2023г.

Руководитель бакалаврской работы _____ Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент _____ Табрисов С.А.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 06.06.2023 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 88 листов, 19 рисунков, 1 таблицу, 7 источников, 4 приложения.

ОБЛАЧНОЕ ХРАНЕНИЕ ДАННЫХ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, СЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ, *ANDROID*, *JAVA*, *SPRING BOOT*, *RETROFIT*, *HTTP*

Цель работы – разработать программное обеспечение, позволяющее пользователям с мобильного устройства загружать, синхронизировать и управлять своими данными на удаленном сервере.

В ходе разработки необходимо реализовать загрузку данных с мобильного устройства, возможность просматривать и удалять загруженные файлы. Также мобильное приложение должно иметь окна авторизации, регистрации, предоставлять пользователю возможность создавать папки и перемещаться между ними. Серверная часть прогрпммного обеспечения должна осуществлять обработку запросов от и отправку результата на мобильное приложение.

В результате проведенной работы создано клиент-серверное приложение, соответствующее поставленной задаче.

Содержание

Перечень сокращений и обозначений 5

Введение 6

1 Разработка программного обеспечения для облачного хранения данных на устройствах под управлением ОС Android 7

1.1 Анализ предметной области 7

1.1.1 Обзор существующих решений 9

1.1.2 Постановка задачи 14

1.2 Выбор средств разработки 16

1.2.1 Среда разработки *Android Studio* 17

1.2.2 Язык программирования *Java* 17

1.2.3 Библиотека *Retrofit* 18

1.2.4 Библиотека *Jackson* 19

1.2.5 Фреймворк *Spring Boot* 19

1.2.6 СУБД *PostgreSQL* 20

1.3 Архитектура приложения 20

1.4 Реализация программного обеспечения 22

1.4.1 Описание работы клиент-серверного приложения 22

1.4.2 Реализация клиентской части 23

1.4.3 Реализация серверной части 29

1.5 Руководство пользоватедя 35

2 Безопасность жизнедеятельности 42

2.1 Требования к помещениям для работы с ПК 42

2.2 Требования к организации рабочих мест 43

Заключение 44

Список используемых источников 45

Приложение А – *UML*- диаграммы 46

Приложение Б – Листинг клиентской части 53

ПриложениеВ – Листинг серверной части 73

Приложение Г– Презентация 81

Перечень сокращений и обозначений

БД–база данных

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

ЯП – языкпрограммирования

API (*ApplicationProgrammingInterface*) – программныйинтерфейсприложения

HTTP (*HypertextTransferProtocol*) – протоколприкладногоуровня передачи данных

IDE (*Integrated development environment*) –интегрированнаясредаразработки

JSON (*JavaScriptObjectNotation*) – текстовыйформатобменаданными

REST (*REpresentational State Transfer*) – передачасостоянияпредставления

SQL (*StructuredQuery Language*) – язык структурированных запросов

UML (*UnifiedModelingLanguage*) – унифицированный язык моделирования

Введение

Использование цифровых технологий мобильных устройств стало неотъемлемой частью современного мира, и с каждым годом их популярность и востребованность только увеличиваются. Однако, с ростом объемов информации, которую мы создаем и обрабатываем каждый день, возникает проблема ее хранения и управления. Облачные хранилища данных на мобильных устройствах стали ответом на эту проблему, позволяя пользователям сохранять свои данные на удаленных серверах и получать к ним доступ с любого мобильного устройства, подключенного к интернету.

Технология облачного хранения данных стала неотъемлемой частью нашей повседневной жизни, и позволяет нам сохранять и обмениваться информацией безопасно и эффективно. Она также дает возможность организовать удаленную работу, сокращая необходимость использования физических носителей и устройств для передачи информации. Кроме того, облачные хранилища данных позволяют пользователям освободить место на мобильных устройствах, избежать потери данных и обеспечить их доступность в любое время и из любой точки мира.

В данной работе мы рассмотрим технологию облачного хранения данных на мобильных устройствах под управлением операционной системы *Android*, а также рассмотрим различные аспекты ее использования, включая достоинства и недостатки, виды облачных хранилищ, анализ предметной области и разработку программного обеспечения для облачного хранения данных.

1 Разработка программного обеспечения дляоблачного хранения данных на устройствах под управлением ОС Android

1.1 Анализ предметной области

Тема разработки программного обеспечения для облачного хранения данных на устройствах под управлением операционной системы *Android* является актуальной и важной в современном мире. Сегодня все больше людей используют мобильные устройства для хранения своих личных и деловых данных, и облачные хранилища данных являются наиболее удобным способом хранения этих данных.

Однако использование облачных хранилищ данных на мобильных устройствах имеет как достоинства, так и недостатки.

К достоинствам облачных хранилищ можно отнести:

- удобство использования.Облачные хранилища данных на мобильных устройствах обычно имеют простой интерфейс и могут быть легко использованы;
- доступность информации.Загруженные данные хранятся в облаке, что позволяет пользователям получать доступ к ним в любое время и в любом месте с помощью сети Интернет;
- экономия пространства на устройстве.Облачные хранилища позволяют пользователям освободить место на устройстве, избежав дополнительные затраты на оборудование.

Из недостатков можно выделить следующее:

- отсутсвие доступа к данным без подключения к сети.Пользователям для доступа к данным необходимо наличие сети Интернет, что может стать проблемой в местности с ограниченной связью;
- ограничение скорости загрузки и выгрузки данных на устройстве.Загрузка илиотправка данных в облако может быть замедлена из-за

малойскорости Интернет-соединения, что может стать проблемой при передаче больших объемов данных;

- ограниченныйбесплатынйобъем.Многие облачные хранилища предоставляют малый объем памяти для бесплатного использования.

Существует несколько разновидностей облачных хранилищ данных для мобильных устройств: общедоступные и корпоративные.

Общедоступные облачные хранилища — это платформы, которые предоставляют пользователю возможностьхранить данные на удаленном сервере. Данные сервисы позволяют пользователям загружать, хранить и синхронизировать свои данные с множеством разных устройств.

Корпоративные облачные хранилища — это платформы, которые предназначены для использования коммерческими организациями. Часто такие хранилища располагаются в самих организациях.Благодаря этому компании могут хранить и управлять данными своих сотрудников в безопасной облачной среде.

В настоящее время большинство пользователей использует общедоступные облачные хранилища. Существует немало причин, которые могут вынудить пользователя использовать облачное хранилище данных на своем мобильном устройстве. Рассмотрим некоторые из них:

- ограниченный объем памяти на устройстве.Если на мобильном устройстве небольшой объем встроенной памяти или заканчивается свободное место, то облачное хранилище данных может быть использовано для хранения редко используемых файлов;
- синхронизация данных.Облачное хранилище данных может быть использовано для синхронизации данных между различными устройствами пользователя, например, между смартфоном и планшетом;
- удобный доступ к данным.Облачное хранилище предоставляет пользователю удобный и быстрый доступ к своим данным в любое время и в любом месте;

– безопасность данных.Облачное хранилище данных с высоким уровнем защиты информации может обеспечить более надежное место для хранения информации, чем это возможно на мобильном устройстве.

1.1.1 Обзор существующих решений

Поскольку облачные сервисы развиваются достаточно долгое время, существует немало компаний, предоставляющих данные услуги.Для обзора систем, аналогичных проектируемой, сравним нескольких конкурентов создаваемому продукту, выделим достоинства и недостатки каждого сервиса.

Яндекс.Диск– это облачное хранилище, принадлежащее компании Яндекс. Сервис позволяет пользователям хранить свои данные на удаленном сервер. Интерфейс мобильного приложения представлен на рисунке 1.

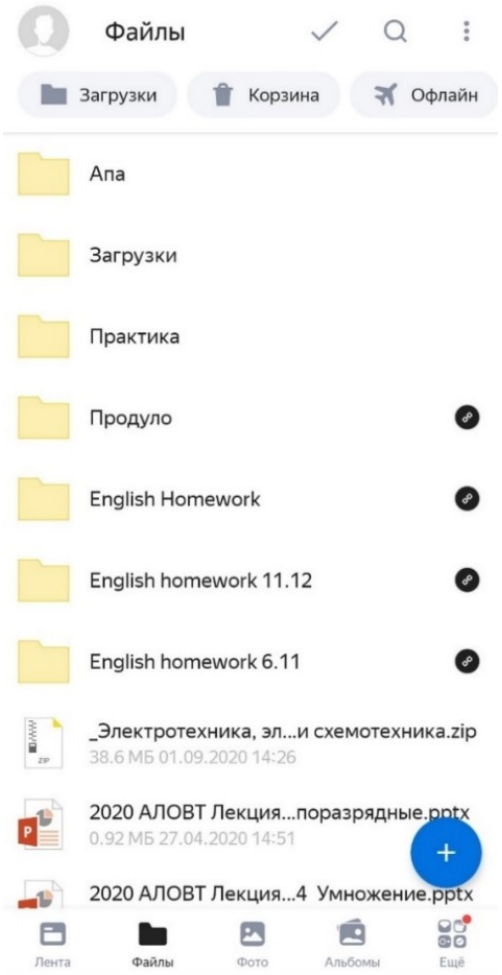


Рисунок 1 – Сервис Яндекс.Диск

К достоинтсвам сервиса можно отнести:

- возможность автоматической загрузки фото и видео с мобильного устройства в облачное хранилище;
- интеграция с другими сервисами Яндекс, такими как почта и поиск;
- поддержка совместной работы с файлами и папками.

Недостатками данного приложения являются:

- ограниченный бесплатный объем хранения - только 5 ГБ, после чего необходимо платить за дополнительное пространство;
- не всегда стабильная работа приложения на устройствах Android;
- реклама в бесплатной версии приложения.

Облако*Mail.ru*–облачное хранилище, предоставляемый компанией *Mail.ru Group*. Хранилище позволяет хранить музыку, видео и другие файлы не сервере и синхронизировать данные на смартфонах или планшетах. Пользоваетльский интерфейс изображен на рисунке 2.

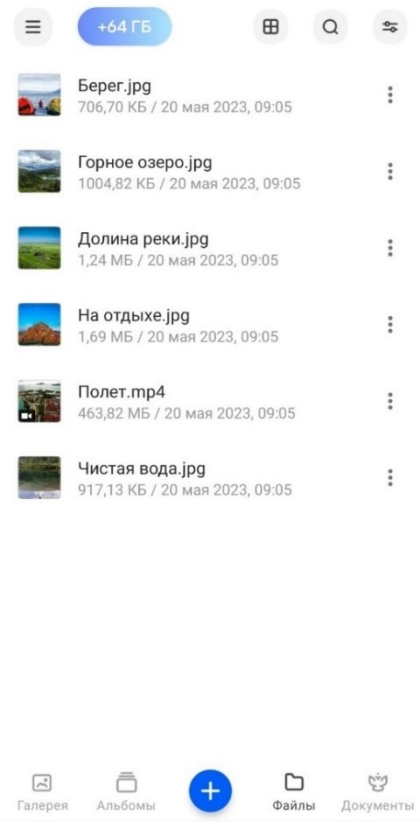


Рисунок 2 - Облако *Mail.ru*

Достоинствами облачного хранищила являются:

- бесплатное облачное хранилище до 8 ГБ;
- возможность совместной работы с документами;
- высокий уровень защиты данных и шифрование при передаче файлов.

В приложении можно выделить следующие недостатки:

- нет возможности загружать файлы размером более 2 ГБ;
- рекламные предложения и ограничения на загрузку файлов в бесплатной версии.

Google Drive – это облачныйсервис для хранения данных, разработанный компанией Google. В его функции входят загрузка, хранение, удаление файлов, а также их редактирование. Приложение представлено на рисунке 3.

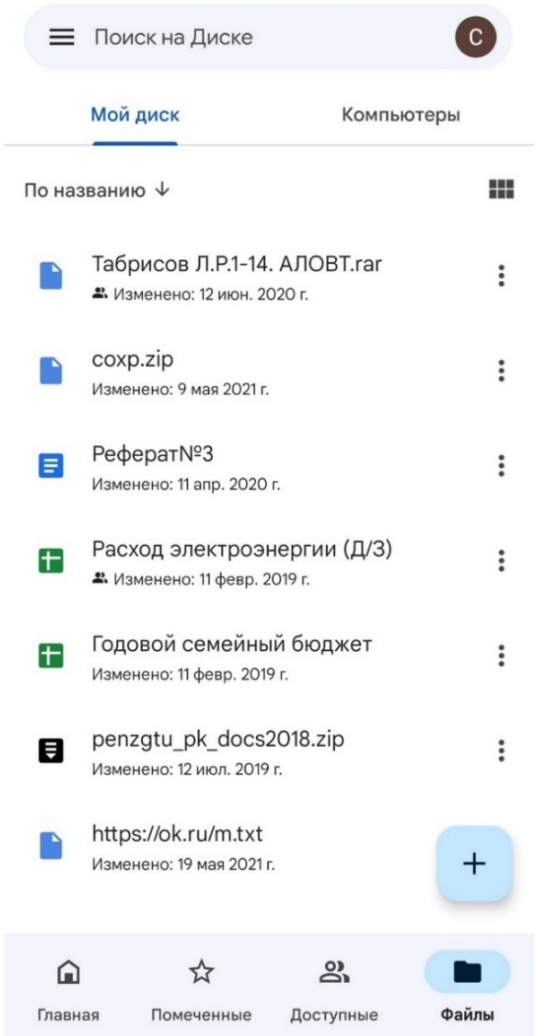


Рисунок 3 -Сервис *GoogleDrive*

Достоинства*GoogleDrive*:

- интеграция с другими сервисами *Google*, такими как *Gmail*, *GoogleDocs*, *GoogleSheets* и *Google Slides*;
- 15 ГБ бесплатного пространства для хранения;
- возможность отправки файлов другим пользователям *Google Drive*;

К недостаткам можно отнести:

- неудобное управление файлами при их большом количестве;
- ограничения на загрузку и скачивание файлов, особенно для бесплатной версии;
- проблемы с конфиденциальностью данных, так как *Google* может получить доступ к файлам пользователей.

DropBox– это облачное хранилище от компании *DropboxInc*. Сервис предоставляет возможность сохранять свои файлы и документы на удаленных серверах, предоставляя к ним доступ с любого устройства. Мобильное приложение сервиса представлено на рисунке 4.

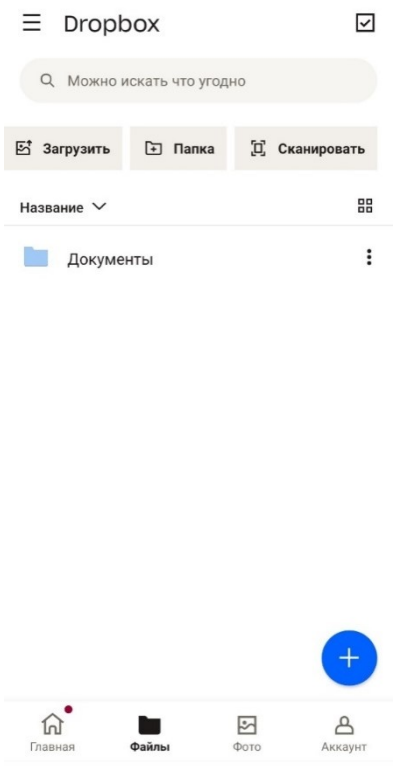


Рисунок 4 - Сервис *DropBox*

Среди достоинств хранилища можно выделить:

- простоту использования и наличие удобных функций, таких как автоматическая загрузка фотографий с устройства;
- хорошую синхронизацию файлов между устройствами.

Недостатки включают:

- ограниченное бесплатное пространство хранения 2 ГБ, что может быть недостаточно для больших объемов данных;
- высокую цену за дополнительное пространство по сравнению с другими облачными хранилищами;
- возможные проблемы с конфиденциальностью данных, так как DropBox имеет доступ к данным пользователей.

Анализ существующих решений показывает, что базовый функционал у сравниваемых сервисов находится примерно на одном уровне. Основной функционал включает в себя загрузку, скачивание и удаление файлов, а также возможность создания папок для организации своих данных (фотографии, видео, документы и т.д.). Кроме того, облачные хранилища обеспечивают резервное копирование и синхронизацию данных между устройствами.

Каждый пользователь выбирает облачное хранилище исходя из своих потребностей, возможностей оплаты того или иного тарифа и прочих факторов, таких как взаимодействие хранилища с другими сервисами, максимальный размер загружаемого файла, мнение других пользователей о надёжности сервиса или привязанность к определённому бренду. По этой причине создать идеальный сервис, который мог бы удовлетворить потребности любого пользователя, практически невозможно. Поэтому создание альтернативного решения, способного составить конкуренцию существующим, является оправданным.

1.1.2 Постановка задачи

По статистике среди мобильных устройств операционная система *Android* лидирует на мировом рынке и занимает около 70%. Именно поэтому было решено выбрать данную операционную систему для разработки программного обеспечения.

Необходимо разработать клиент-серверное приложение, позволяющее пользователю с мобильного устройства загружать, скачивать и удалять файлы, а также добавлять папки для организации своих данных.

Возможности и функционал программного обеспечения отображены в диаграммах прецедентов на рисунках 5 и 6.

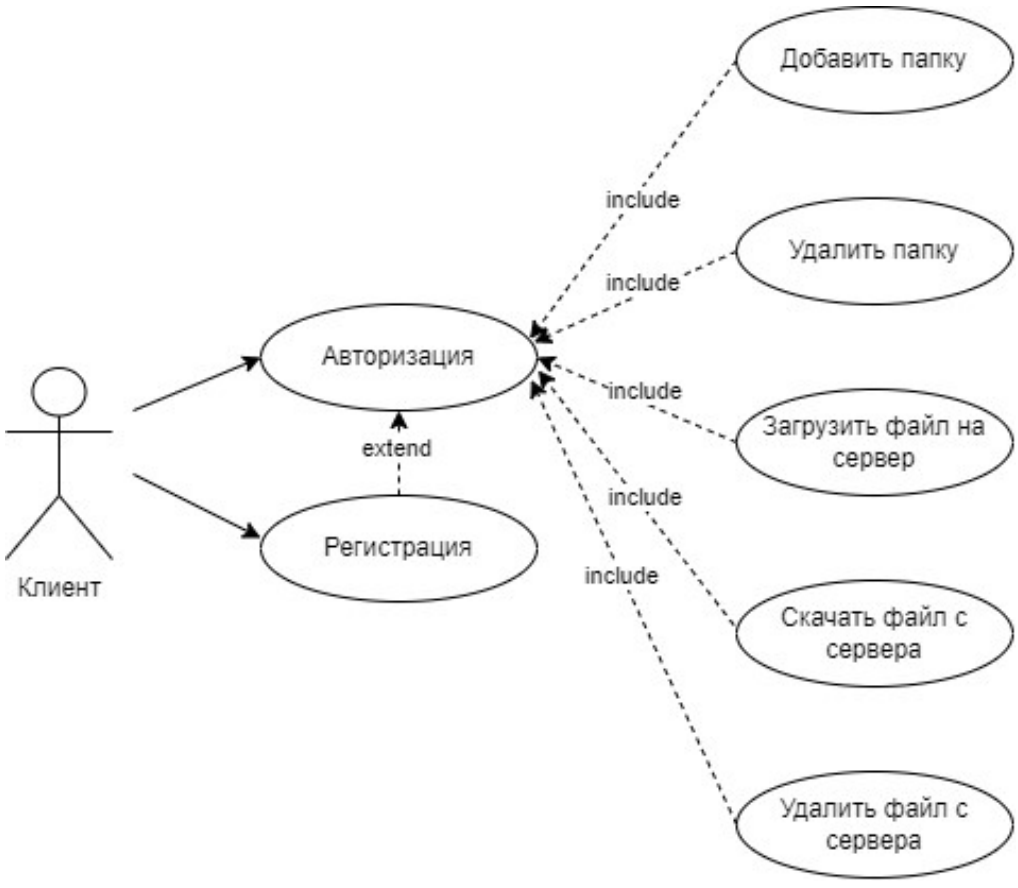


Рисунок 5 – *UML*-диаграмма прецедентов клиента

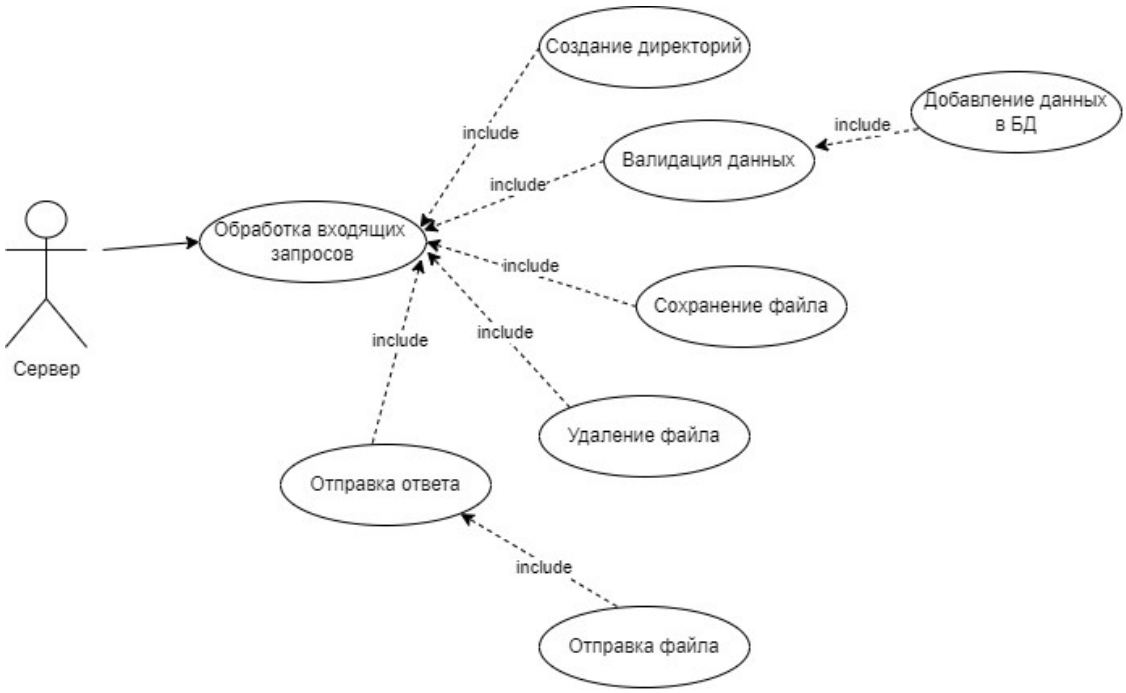


Рисунок 6 – *UML*-диаграмма прецедентов сервера

Функционал со стороны сервера:

- регистрация новых пользователей с проверкой данных. Сервер проверяет указанный пользователем логин в базе данных. Если логин уже хранится в БД, то это будет означать что данный логин зарезервирован другим пользователем;
- авторизация пользователя. Сервер осуществляет проверку введенного логина и пароля пользователя с данными из БД;
- создание директории клиента. После успешной регистрации пользователя на сервере создается каталог с его именем для хранения данных;
- добавление файла в хранилище. Выполняется загрузка отправленного файла и его сохранение в директорию авторизированного пользователя;
- отправка файла пользователю. Осуществляется отправка файла пользователю на мобильное устройство;
- удаление файла пользователя. Сервер выполняет удаление ранее загруженного файла из каталога пользователя;

- возможность обслуживания сразу нескольких клиентов.

Клиентская часть программного обеспечения предоставляет пользователю:

- простой и удобный графический интерфейс;
- возможность авторизации и регистрации пользователя;
- загрузку выбранного файла из локального хранилища на сервер;
- загрузку файлов с сервера. Клиентская часть приложения формирует запрос на скачивание файла пользователя. После чего выполняется загрузка файла в локальное хранилище данных мобильного устройства;
- удаление файлов с сервера;
- возможность просматривать загруженные файлы;
- создание папок для удобной организации своих данных.

1.2 Выбор средств разработки

Выбор инструментов разработки является важным этапом в создании любого программного продукта. Выбор правильных инструментов и технологий может существенно повысить качество и производительность приложения, снизить время разработки.

Для реализации программного обеспечения для облачного хранения данных на мобильных устройствах были рассмотрены следующие технологии и языки:

- *Android Studio* – интегрированная среда разработки для работы с платформой *Android*;
- *Java* – высокоуровневый, объектно-ориентированный язык программирования;
- *Retrofit* – библиотека для работы с сетевыми запросами на платформе Android;
- *Jackson* – библиотека для работы с данными в формате JSON в языке программирования Java;

- *Springboot* – фреймворк, основанный на платформе Spring, который предоставляет удобные инструменты для разработки приложений на платформе Java;
- *Tomcat* – популярный веб-сервер, реализующий обработку *HTTP*-запросов;
- *PostgreSQL* – мощная и расширяемая объектно-реляционная система управления базами данных (СУБД).

1.2.1 Среда разработки *Android Studio*

Android Studio — это интегрированная среда разработки (*IDE*), предназначенная специально для создания приложений под платформу *Android*. Разработанная компанией *Google*, *Android Studio* предлагает разработчикам удобные инструменты, богатый функционал и высокую производительность для создания приложений на *Android*.

Основными преимуществами данной среды разработки являются:

- проектная структура.*Android Studio* предоставляет удобную структуру проекта, включающую исходные коды, ресурсы, макеты пользовательского интерфейса и другие компоненты приложения;
- дизайнер пользовательского интерфейса.*Android Studio* включает в себя интуитивно понятный графический редактор для создания пользовательского интерфейса приложений;
- интеграция с *Android SDK* и другими инструментами.*Android Studio* тесно интегрирован с *Android SDK*, что обеспечивает доступ к богатому набору *API* и инструментов разработки для создания полнофункциональных приложений;
- поддержка языков программирования.Android Studio поддерживает несколько языков программирования, включая *Java*.

1.2.2 Язык программирования *Java*

Java — это объектно-ориентированный язык программирования, разработанный компанией *Sun Microsystems* (ныне принадлежит *Oracle*). Он

широко используется для создания различных типов приложений, включая веб-приложения, мобильные приложения, настольные приложения и игры [1].

Разработка проектов на языке *Java* имеет множество плюсов:

- кроссплатформенность.*Java* является платформа-независимым языком программирования,это означает, что программы, написанные на *Java*,могут работать на различных операционных системах, таких как *Windows*, *macOS*, *Linux*;
- многопоточность.*Java*предоставляет встроенную поддержку многопоточности, что позволяет эффективно использовать ресурсы процессора;
- большая библиотека.*Java* имеет обширную библиотеку классов, которая предоставляет множество готовых решений для различных задач. Она включает классы для работы с вводом-выводом, сетевыми операциями, работой с базами данных, многопоточностью и многим другим.

1.2.3 Библиотека *Retrofit*

Retrofit — это библиотека для работы с сетевыми запросами в приложениях на платформе *Android*. Она предоставляет удобный и эффективный способ взаимодействия с серверной частью приложения, основанной на архитектуре *REST API*.

Некоторые особенности *Retrofit*:

- простота использования.*Retrofit* предлагает простой и интуитивно понятный *API*, который позволяет легко выполнять *HTTP*-запросы и обрабатывать ответы от сервера;
- аннотации.*Retrofit* использует аннотации для определения типа запроса (*GET*, *POST*, *PUT*, *DELETE*), пути к ресурсу и параметров запроса;
- управление потоками.*Retrofit* интегрируется с библиотекой *RxJava*, позволяя выполнять сетевые запросы в асинхронном режиме;
- *Retrofit*значительно упрощает разработку клиентской части для взаимодействия сервером и обработку данных, полученных от него.

1.2.4 Библиотека*Jackson*

Jackson — это библиотека для работы с форматами данных, такими как *JSON*, в языке программирования *Java*. Она предоставляет мощные инструменты для сериализации и десериализации.

Библиотека *Jackson*имеет ряд преимуществ, таких как:

- простота использования.*Jackson* позволяет легко преобразовывать *Java*-объекты в *JSON* и обратно, скрывая сложности парсинга и генерации *JSON*;
- гибкость.*Jackson* поддерживает различные способы сопоставления свойств объектов с полями *JSON*. Он может работать с аннотациями, конфигурационными файлами, что позволяет гибко настраивать процесс сериализации и десериализации.
- высокая производительность.*Jackson* известен своей высокой производительностью. Он оптимизирован для эффективной работы с большими объемами данных и обладает низкими накладными расходами при сериализации и десериализации.

1.2.5 Фреймворк *Spring Boot*

Spring Boot — это фреймворк, основанный на платформе *Spring*, который упрощает и ускоряет процесс создания приложений на платформе *Java*. Он предоставляет все необходимые инструменты, конфигурации и конвенции для быстрой разработки приложений без необходимости настройки множества дополнительных компонентов.

Одной из ключевых особенностей фреймворка является то, что он предоставляет возможность включения встроенного сервера *Tomcat* в приложение с помощью нескольких простых шагов. Для этого необходимо добавить зависимость на *Tomcat* в файле конфигурации *Maven* или *Gradle* для проекта *Spring Boot*. После этого Spring Boot автоматически настроит и запустит *Tomcat* при запуске приложения.

Использование встроенного сервера *Tomcat* в *Spring Boot* упрощает разработку, серверной части приложения платформе *Java*, предлагая гибкость, простоту настройки, хорошую интеграцию с другими компонентами *Spring*[2].

1.2.6 СУБД *PostgreSQL*

PostgreSQL — это система управления базами данных (СУБД) реляционного типа с открытым исходным кодом. Она предназначена для хранения и управления структурированными данными. Поддерживает стандарт SQL и предлагает широкий набор функциональных возможностей, включая сложные запросы, транзакции, представления, триггеры, хранимые процедуры и многое другое.

СУБД отличается высокой производительностью и масштабируемостью, позволяя обрабатывать как небольшие базы данных, так и большие объемы данных. Она также обладает расширяемостью, позволяя создавать и интегрировать пользовательские типы данных, функции и операторы.

1.3 Архитектура приложения

Клиент-серверная архитектура широко используется в разработке программного обеспечения для облачного хранения данных на мобильных устройствах, потому что она позволяет разделить задачи между клиентом (мобильным устройством) и сервером (облачным хранилищем).

Клиентская часть программного обеспечения предназначена для пользователя. Она содержит графический интерфейс и позволяет пользователям создавать папки, загружать, скачивать и удалять файлы, отправляя соответствующие запросы на сервер.

Серверная часть представляет собой программу, которая работает в фоновом режиме. Сервер выполняет прием запросов, которые отправляются с клиентской части приложения, осуществляет их обработку и отправку результата пользователю.

Сетевое взаимодействие осуществляется с помощью *HTTP*-протокола и *REST*-архитектуры. *HTTP* и *REST* взаимосвязаны между собой при построении клиент-серверных приложений [3].

HTTP (*Hypertext Transfer Protocol*) является протоколом передачи данных, который определяет правила и формат обмена информацией между клиентом и сервером. Он определяет различные методы запросов, стандартные коды ответов сервера, такие как 200 (*OK*), 404 (*NotFound*) и другие. *HTTP* определяет также структуру сообщений, включая заголовки и тело запроса или ответа.

REST, с другой стороны, является архитектурным стилем для проектирования сетевых приложений. Он определяет набор ограничений и принципов для построения распределенных и масштабируемых систем. Один из ключевых принципов *REST* заключается в том, что клиент и сервер обмениваются представлениями состояния ресурсов.То есть каждый запросдолжен содержать всю необходимую информацию для его обработки на сервере. Сервер не должен сохранять состояние между запросами от одного клиента к другому. Клиенты должны предоставлять все данные, необходимые для выполнения операции.

REST использует протокол *HTTP* для передачи данных и выполнения операций над ресурсами. *REST API* использует различные методы *HTTP*, такие как *GET*, *POST*, *PUT* и *DELETE*, чтобы клиенты могли получать данные, создавать, обновлять и удалять ресурсы на сервере.

В соответствии с выбранной архитектурой были разработаны запросы для взаимодействия клиентской и серверной части. Ниже представлен пример запросов к серверной части программного обеспечения:

- *POST /api/upload/username* – запрос для отправки файла на сервер (файл в теле запроса);
- *POST /api/download/username* – запрос для загрузки файла с сервера (файл в теле ответа);

- *DELETE /api/delete/username* – запросдляудаленияфайла с сервера;
- *POST /users/check_user_password* – запрос для авторизации пользователя (данные в теле запроса);
- *POST /users/check_user* – запрос для регистрации пользователя. В тело запроса передается *JSON* – объект, который содержит информацию о пользователе.

1.4 Реализация программного обеспечения

1.4.1 Описание работы клиент-серверного приложения

Клиент-серверное приложение содержит клиентскую и серверную часть. Клиентская часть представлена в виде мобильного приложения на платформе *Android*. Мобильное приложение осуществляет взаимодействие с пользователем, предоставляя ему удобный графический интерфейс. После прохождения регистрации или авторизации, пользователь имеет возможность отправлять, загружать и удалять файлы с сервера.Серверная часть представляет собой программу, работающую в фоновом режиме. Сервер осуществляет обработку входящих запросов и отправку результата пользователю на мобильное устройство.

Сетевое взаимодействиемежду клиентом и сервером осуществляется с помощью *HTTP*-протокола. Диаграммы развертывания и последовательностей представлены на рисуках 8 и 9.

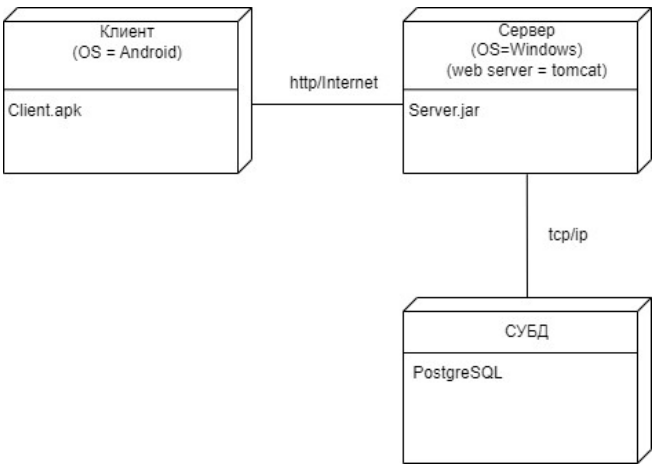


Рисунок 8 – *UML*-диаграмма развертывания

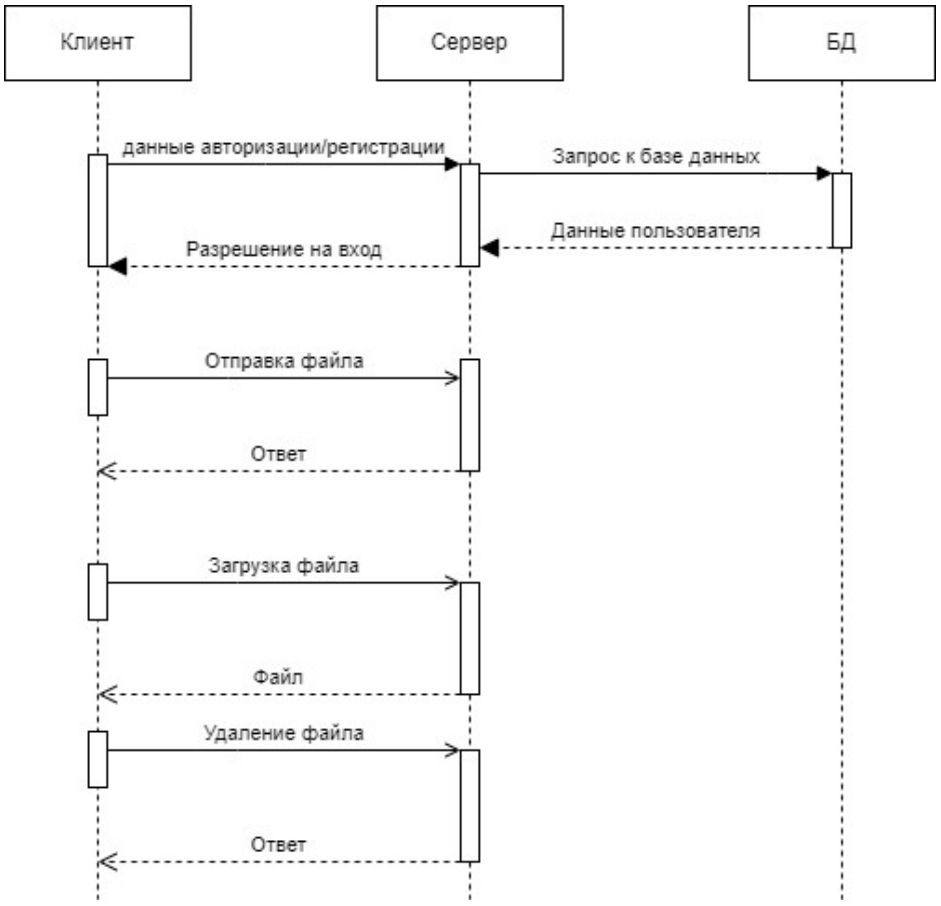


Рисунок 9 – *UML*-диаграмма последовательностей

1.4.2 Реализация клиентской части

Мобильное приложение будет работать с файлами и сетью, чтобы это стало возможным необходимо получить следующие разрешения:

- *READ_EXTERNAL_STORAGE* — это разрешение позволяет приложению читать данные из хранилища устройства;
- *WRITE_EXTERNAL_STORAGE* – разрешение предоставляет доступ к сохранению файлов на устройство;
- *INTERNET* – разрешение предоставляет приложению возможность отправки и загрузки данных по сети.

Для этого указали эти разрешения в файле *AndroidManifest.xml*, чтобы система могла уведомить пользователя о запрашиваемых разрешениях при первом запуске приложения (рисунок10). Фрагментфайла*AndroidManifest.xml*представленниже:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.INTERNET"/>
```

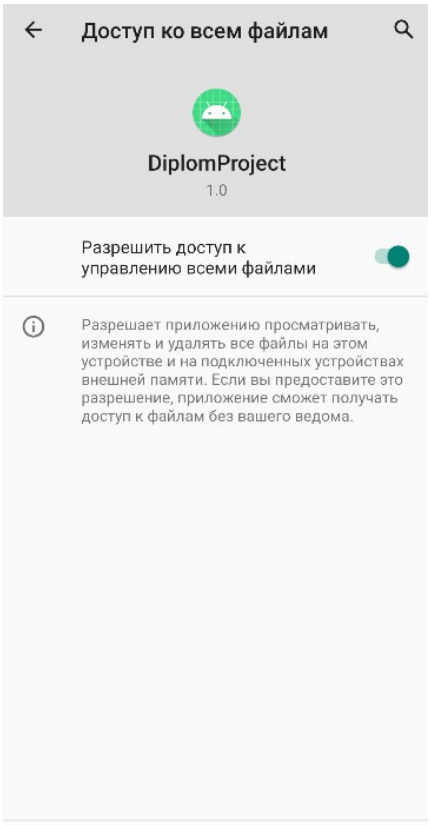


Рисунок10 – Запрос разрешений

Сетевое взаимодействие со стороны клиентского приложенияреализовано с помощью библиотеки*Retrofi*[4]. Был разработан класс *RetrofitService* для установления связи с сервером, а также создан интерфейс *NetworkApi*, который содержит функции для отправки запросов на сервер:

- метод *RegUser()*отправляет данные пользователя в *JSON*-формате на сервер для прохождения регистрации;
- метод *AuthUser()*аналогичен предыдущему, но предназначен для авторизации пользователя;
- метод *UploadFile()*осуществляет отправку файла на сервер;

- метод *DownloadFile()*выполняет загрузку файла с сервера;
- метод *DeleteFile()*выполняет удаление файла с сервера;
- методы*GetDatalist()*и*SaveDataList()*предназначены для сохранения и восстановления состояния графического интерфейса.

Для вышеперечисленных методов с помощью аннотаций указали *URI*-адрес конечной точки *API*и тип *HTTP*-запроса, который должен быть выполнен при вызове соответствующего метода. Нижепредставленфрагменткодаинтерфейса*NetworkApi*:

```
public interface NetworkApi {

    @POST("/users/check_user")
    Call<Boolean>RegUser(@Body UserData user);

    @POST("/users/check_user_password")
    Call<Boolean>AuthUser(@Body UserData user);

    @Multipart
    @POST("/api/upload/{username}")
    Call<ResponseBody>uploadFile(@Path("username") String
username, @Part MultipartBody.Part file);

    @Streaming
    @POST("/api/download/{username}")
    Call<ResponseBody>downloadFile(@Path("username") String
username, @Body String data);

}
```

Пользовательский интерфейс приложения реализован с помощью фрагментов. Фрагменты предоставляют более гибкий и модульный подход к разработке интерфейса приложения, позволяя повторно использовать и комбинировать различные части пользовательского интерфейса в разных активностях [5]. Они могут быть добавлены, удалены или заменены во время выполнения, что позволяет динамически изменять интерфейс приложения в зависимости от различных сценариев.

Для реализации страниц регистрации и авторизации были разработаны два класса: *RegisterFragment*и*EnterFragment*. Внутри каждого класса есть

кнопка, которая завершает процесс регистрации или авторизации. При нажатии на кнопку отправляется соответствующий запрос на сервер и ожидается результат. Для отслеживания нажатия на кнопку был использован метод *setOnClkickListener*.

Ниже представлен фрагмент кода авторизации пользователя:

```
enterButton.setOnClickListener(v -> {
        String login = ReadLogin();
        String password = ReadPassword();
userData.SetLogin(login);
userData.SetPassword(password);
        new Thread(() -> {
            try {
                Call<Boolean> call =
networkApi.AuthUser(userData);
                Response<Boolean> response = call.execute();
                if(Boolean.TRUE.equals(response.body())){
                    Intent intent = new Intent(getContext(),
MainActivity.class);
intent.putExtra(UserData.class.getSimpleName(), userData);
startActivity(intent);
                } else {
                    new Handler(Looper.getMainLooper()).post(() -
> {
Toast.makeText(getActivity(), "Неверныйлогинилипароль!",
Toast.LENGTH_SHORT).show();
inputPassword.setText("");
inputLogin.setText("");
                    });
                }
            } catch (IOException e) {
e.printStackTrace();
            }
        }).start();});}
```

Класс *FragmentFileSystem* отвечает за реализацию основного окна, где осуществляется работа с файлами и графическим интерфейсом. В данном классе созданы методы для загрузки и отправки файлов по сети.

Метод *downloadFile()* осуществляет загрузку файла с сервера с использованием*Retrofit*.

Кодметода*downloadFile()*:

```
public void downloadFile(String filename){
RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
```

```
retrofitService.getRetrofit().create(NetworkApi.class);
ExecutorServiceexecutorService =
Executors.newSingleThreadExecutor();
networkApi.downloadFile(userData.GetLogin(),
filename).enqueue(new Callback<ResponseBody>() {
        @SuppressLint("StaticFieldLeak")
        @Override
        public void onResponse(@NonNull Call<ResponseBody> call,
@NonNull Response<ResponseBody> response) {
executorService.execute(() -> {
                if(saveFile(filename,
Objects.requireNonNull(response.body()))){
                    new Handler(Looper.getMainLooper()).post(() -
> {
Toast.makeText(getActivity(), "Файлзагружен",
Toast.LENGTH_SHORT).show();
                    });
                }});
        }
}
```

Методы *saveFile()* и *UploadFile()* выполняют сохранение файла на устройстве и загрузку файла на сервер соответственно.Нижепредставленфрагментkодафункции*UploadFile()*:

```
private void UploadFile(Uri uri, String name) {
    File file = new
File(RealPathUtil.getRealPath(requireContext(), uri));
RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi
=retrofitService.getRetrofit().create(NetworkApi.class);
RequestBodyrequestFile = RequestBody.create(MediaType.
parse(getContext().getContentResolver().getType(uri)), file);
MultipartBody.Part body =
MultipartBody.Part.createFormData("file", name, requestFile);
networkApi.uploadFile(userData.GetLogin(), body).enqueue(new
Callback<ResponseBody>() {
        @Override
        public void onResponse(@NonNull Call<ResponseBody> call,
@NonNull Response<ResponseBody> response) {
            if(response.code() == 200){
                new Handler(Looper.getMainLooper()).post(() -> {
Toast.makeText(getActivity(), "Файлотправлен!",
Toast.LENGTH_SHORT).show();
                });
            }
     }
```

```
    })
;}
```

Для удобного отображения списка файлов и папок на экране был создан класс *RecyclerAdapter*. Этот адаптер позволяет связать данные, представляющие файлы и папки, с элементами пользовательского интерфейса, такими как *RecyclerView*, и обеспечивает правильное отображение и обновление списка.*FragmentFileSystem*реализует интерфейсы *View.OnClickListener* и *RecyclerAdapter.OnItemSelectedListener* для обработки нажатий на элементы списка.

Клиент может удобно перемещаться между папками, используя кнопку “назад”. Для реализации навигации был создан интерфейс *INavigator*, который содержит метод для поиска предыдущего фрагмента. За реализацию методов интерфейса отвечает класс*MainActivity*. Ниже представлен фрагмент кода поиска предыдущего фрагмента:

```
@Override
public void FindFragmentInStack(String tag) {
lastFragment = nextFragment;
nextFragment = (FileSystemFragment)getSupportFragmentManager()
.findFragmentByTag(tag);
ChangeFragment(nextFragment, nextFragment.tag);
}
```

Мобильное приложение реализует механизмсериализации и десериализации графического интерфейса пользователя. Сериализация списка происходит в методе жизненного цикла фрагмента *onStop()* в том случае если список был изменен. Список преобразуется в *JSON*объект с помощью библиотеки *Jackson*. После чего отправляется на сервер в теле запроса. Далее на сервере список сохраняется в файл.

При следующем запуске приложения список файлов восстанавливается. В конструкторе класса вызывается метод *initList()*, в котором приложение запрашивает данные с сервера. Полученные данные преобразуются из

*JSON*объекта в графический интерфейс. Нижепредставленфрагменткодаметода*initList()*:

```
RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);
networkApi.GetDataList(userData.GetLogin(), title).enqueue(new
Callback<Fields>() {
        @SuppressLint("NotifyDataSetChanged")
        @Override
        public void onResponse(Call<Fields> call,
Response<Fields> response) {
            if(response.body()!=null){
list.addAll(response.body().list);
adapter.notifyDataSetChanged();
}
        }
    });
```

1.4.3 Реализация серверной части

Разработка серверной части программного обеспечения осуществлялась с помощью фреймворка *SpringBoot.*

Файл *application.properties* в *Spring Boot* используется для конфигурации приложения [6]. Серверное приложение будет работать с базой данных и загрузкой файлов по сети. Исходя из этого в файле конфигурации установили порт 9000 для подключения, указали диалект *Hibernate*для работы с СУБД *PostgreSQL*, включили поддержку мультипарт-запросов, которые позволяют загружать файлы на сервер, установили максимальный размер загружаемого файла 200МБ и указали директорию для хранения файлов.

Фрагмент файла конфигурации представлен ниже:

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Pos
tgreSQLDialect
spring.datasource.platform=postgres
spring.datasource.url=jdbc:postgresql://localhost:5432/users
spring.datasource.username=postgres
spring.datasource.password=123456
spring.servlet.multipart.enabled=true
spring.servlet.multipart.file-size-threshold=2KB
spring.servlet.multipart.max-file-size=200MB
```

```
spring.servlet.multipart.max-request-size=215MB
file.upload.location=/SpringServerJava/uploads
server.port=9000
```

При разработке была использована СУБД *PostgreSQL*. В базе данных серверного приложения находится таблица *user_data*, котораяхранит персональные данные пользователя после прохождения регистрации. В таблице 1 представлено описание атрибутов таблицы базы данных.

Таблица 1 – Персональные данные пользователя

Название поля	Описание поля	Тип данных	Ограничения
id	Идентификатор пользователя	*IDENTITY*	*PK*
user_name	Имя пользователя	*VARCHAR(255)*	*NOT NULL*
user_login	Логин пользователя	*VARCHAR(255)*	*NOT NULL*
user_email	Emailпользователя	*VARCHAR(255)*	*NOT NULL*
user_password	Пароль	*VARCHAR(255)*	*NOT NULL*

Каждое поле имеет ограничение на 255 символов, за исключением поля *id*. Поле*id* с типом данных *IDENTITY*автоинкрементируется при каждой записи и не имеет минимального или максимального ограничения.

Тип данных *VARCHAR*позволяет хранить текстовые данные определенной длины, в данном случае 255 символов.

Ограничение *PK*определяет поле как первичный ключ. Это означает, что оно будет идентификатором каждой записи. Это необходимо для различения одинаковых записей в таблице.

Ограничение *NOT NULL* у полей *user_name*, *user_login*, *user_email*, *user_password*означает, что они не могут быть пустыми. К примеру, мы не можем добавить пользователя в базу данных без указания этих полей.

Для отображения объекта в таблицу в *Spring Boot*используется сущность (*Entity*). Сущность представляет собой класс, который проецируется на таблицу в базе данных. Серверсодержитсущность*UserData*, кодкоторойпредставленниже:

```
@Entity
@Table(name = "user_data")
public class UserData {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

    @Column(name = "USER_LOGIN")
    private String login;

    @Column(name = "USER_EMAIL")
    private String email;

    @Column(name = "USER_PASSWORD")
    private String password;
}
```

С помощью аннотации *@Column* поля классасопоставляются с таблицей. Сущность*UserData*имеетатрибуты*id*, *name*, *login*, *email*, *password*, которые отображаются на столбцы таблицы *user_data*в базе данных.

Для управления доступом к данным в БД S*pringBoot*предоставляет компонент – репозиторий [7].

Репозитории представляют собой интерфейсы или классы, которые обеспечивают абстракцию для доступа к информации, хранящаяся в базе данных.Репозитории используются для выполнения операций создания, чтения, обновления и удаления с объектами сущностей.

Основной целью репозиториев является скрытие деталей взаимодействия с базой данных и предоставление удобного *API* для работы с данными.

Интерфейс *UserRepository*, отвечает за выполнение операций чтения и записи в базу данных. Репозиторий наследуются от интерфейса *CrudRepository* и содержит в себе два запроса для извлечения данных. Нижепредставленкодинтерфейса*UserRepository*:

```
@Repository
public interface UserRepository extends CrudRepository<UserData,
Integer> {

    @Query(value = "SELECT * FROM User_Data WHERE USER_LOGIN =
?1", nativeQuery=true)
UserDatafindUserByLogin(String login);

    @Query(value = "SELECT USER_PASSWORD FROM User_Data WHERE
USER_LOGIN = ?1", nativeQuery=true)
    String findPasswordByLogin(String login);
}
```

Бизнес-логика серверного приложения реализована в сервисах.

Сервис в *SpringBoot* — это компонент или слой приложения, который выполняет конкретные задачи или операции. Сервисы отвечают за обработку данных, выполнение операций ввода-вывода и взаимодействие с другими компонентами системы.

Серверная часть программного обеспечения содержит два сервиса: *UserService*и *FileService*.

UserService взаимодействует с репозиторием *UserRepository* и реализует логику работы с данными пользователя:

- метод *CheckUserAndSave()*проверяет наличие пользователя в таблице по логину. Если логина нет в базе данных, то пользователь добавляется в таблицу;
- метод *CheckUserPassword()* осуществляет проверку логина и пароля пользователя с данными из БД.

*FileService*отвечает за работу с файлами пользователя:

- метод *CreateDir()*создает необходимые директории для хранения файлов пользователя;
- метод *SaveFile()*сохраняет загруженный файл в директорию пользователя;
- метод *DeleteFile()*удаляет ранее загруженный файл из хранилища;

– метод *LoadFile()* считывает выбранный файл для последующей отправки пользователю.

Для обработки веб-запросов *Spring Boot*предоставляет компонент – контроллер. Контроллеры отвечают за прием запросов от клиента, обработку запроса, взаимодействие с другими компонентами (например, сервисами) и возвращение ответа клиенту.

В серверной части приложения реализовано два контроллера: *UserController* и *FileController*.

*UserController*отвечает за прием запросов для регистрации или авторизации пользователя. Контроллер взаимодействует с сервисом *UserService* и включает в себя два метода, к каждому из которых клиент может обратиться, используя уникальный *URI*.

Нижепредставленфрагменткодаконтроллера:

```
@RestController
public class UserController {

    @Autowired
    private UserServiceuserService;

    @PostMapping("/users/check_user")
    public booleanCheckRegisterUser(@RequestBody
UserDatauserData){
        return userService.CheckUserAndSave(userData);
    }

    @PostMapping("/users/check_user_password")
    public booleanCheckAuthUser(@RequestBody UserDatauserData){
        return userService.CheckUserPassword(userData);
    }
}
```

*FileController*ничем не отличается от *UserController*,он обрабатывает запросы, связанные с операциями над файлами, и имеет собственные *URI*. Контроллер использует методы сервиса *FileService* для сохранения, загрузки и удаления файла.

Например, для загрузки файла с сервера, пользователь будет использовать *POST*-запрос по адресу *localhost:9000/api/download/username*, где *username*–логин пользователя, а имя файла передается в теле запроса.

Кодконтроллерапредставленниже:

```
@RestController
@RequestMapping("/api")
public class FileController {
    @Autowired
iFileSystemStoragefileSystemStorage;

    @PostMapping("/upload/{username}")
    public ResponseEntity<FileResponse>uploadSingleFileToDir
(@PathVariable String username, @RequestParam("file")
MultipartFile file) {
        String upfile = fileSystemStorage.saveFile(file,
username);
        String fileDownloadUri =
ServletUriComponentsBuilder.fromCurrentContextPath().path("/api/d
ownload/").path(upfile).toUriString();
        return ResponseEntity.status(HttpStatus.OK).body(new
FileResponse(upfile,fileDownloadUri,"File uploaded with
success!"));
    }

    @PostMapping("/download/{username}")
    public
ResponseEntity<Resource>downloadFileWithBodyAndPathVar(@PathVaria
ble String username, @RequestBody String filename) {
        Resource resource =
fileSystemStorage.loadFile(filename.trim().replaceAll("\"",""),
username);
        return
ResponseEntity.ok().header(HttpHeaders.CONTENT_DISPOSITION,
"attachment; filename=\"" + resource.getFilename() +
"\"").body(resource);
    }
 }
```

1.5 Руководство пользоватедя

В результате компиляции серверной программы был получен исполняемый файл «*SpringServer.jar*». Для его запуска можно использовать команду «*Java –jarSpringServer.jar*». На рисунке 11 представлен результат успешного запуска серверного приложения.

При первом запуске сервер создает папку *uploads*в том же каталоге, где и находится *JAR*-файл.Далее при регистрации пользователя создается каталог с его именем и два подкаталога *files*и *data*. В папке *files*хранятся файлы, загруженные пользователем, а в папке *data*хранятся данные о состоянии графического интерфейса мобильного приложении.

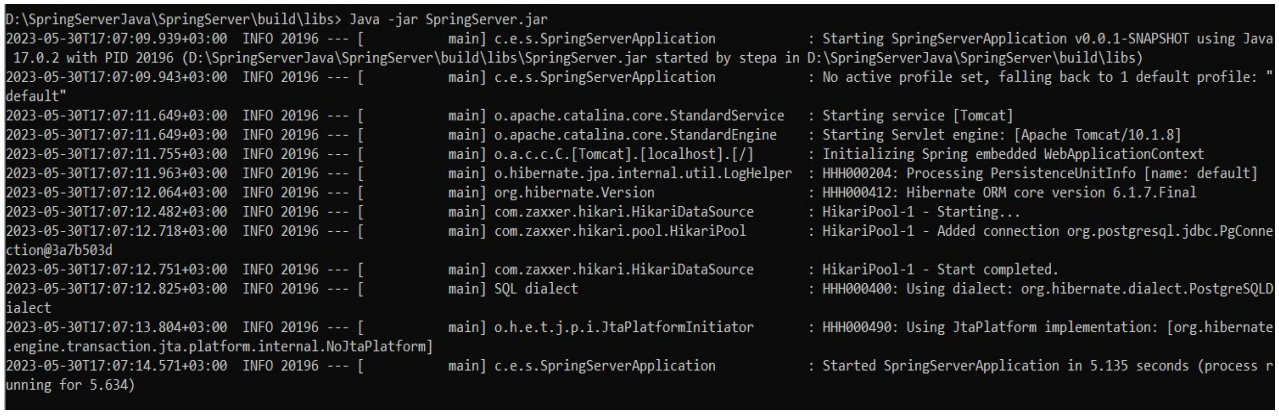


Рисунок11 – Запуск сервера

Работа клиентского приложения начинается с окна авторизации. После того как пользователь предоставил все необходимые разрешения для мобильного приложения, он попадает в окно авторизации. Если у пользователя еще нет аккаунта, то он может зарегистрироваться, перейдя в окно регистрации. На рисунке 12 представлены окна для регистрации и авторизации.

После успешной аутентификации или регистрации пользователь попадет в основное окно приложения (рисунок 13). В этом окне пользователь может

отправлять файлы на сервер, удалять и загружать их с него. Также доступно добавление папок для организации своих данных.

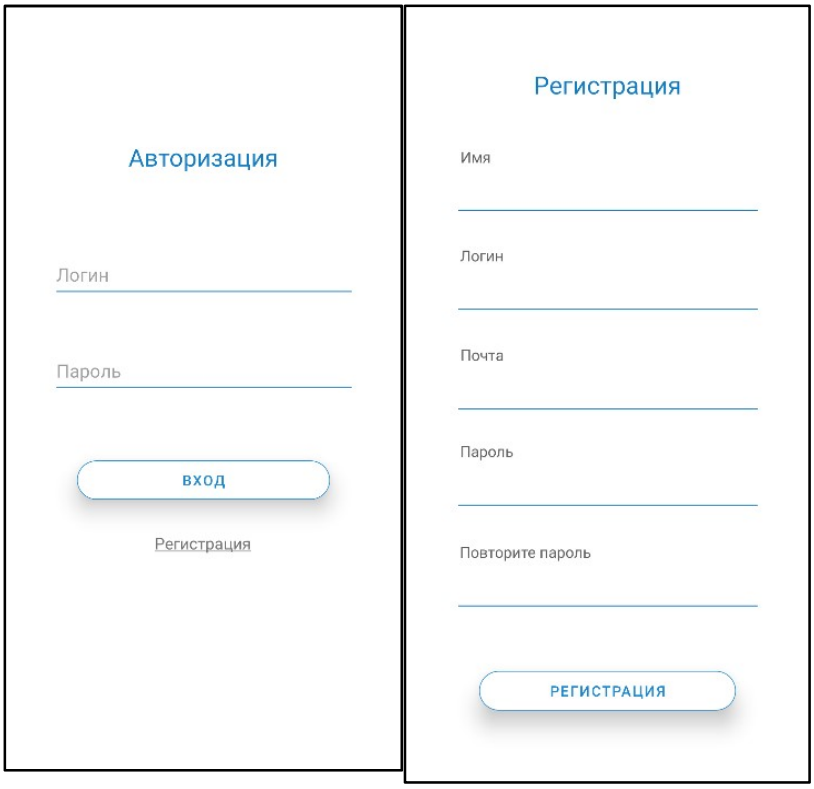


Рисунок12 – Окна регистрации и авторизации

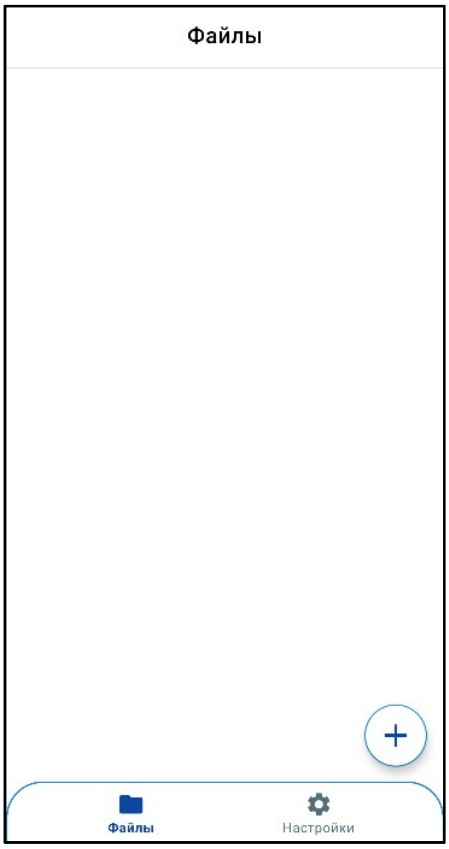


Рисунок13 – Основное окно

Чтобы добавить папку, пользователю необходимо нажать “+” и выбрать кнопку с изображением папки. Далее открывается диалоговое окно, куда необходимо ввести название папки. После подтверждения новая папка отображается на экране пользователя. Результат добавления папки показан на рисунке 14.

Попасть в созданную папку достаточно легко, необходимо просто кликнуть по ней. Результат открытия папки показан на рисунке 15.

Для того, чтобы загрузить файл на сервер пользователю необходимо нажать “+” и выбрать кнопку с изображением файла. После нажатия открывается окно для выбора файла из хранилища устройства (рисунок 16).

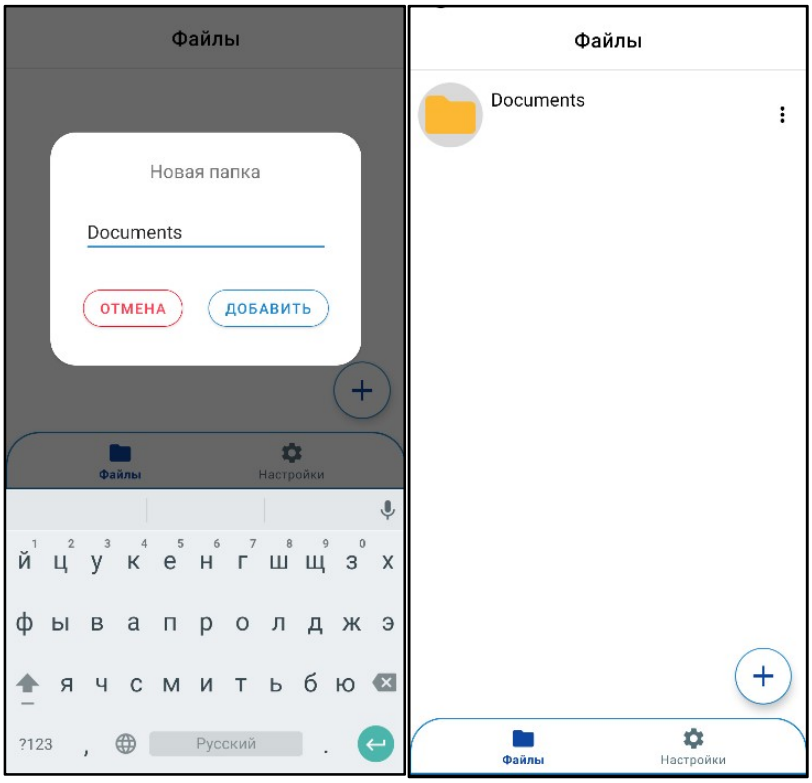


Рисунок14 – Добавление папки

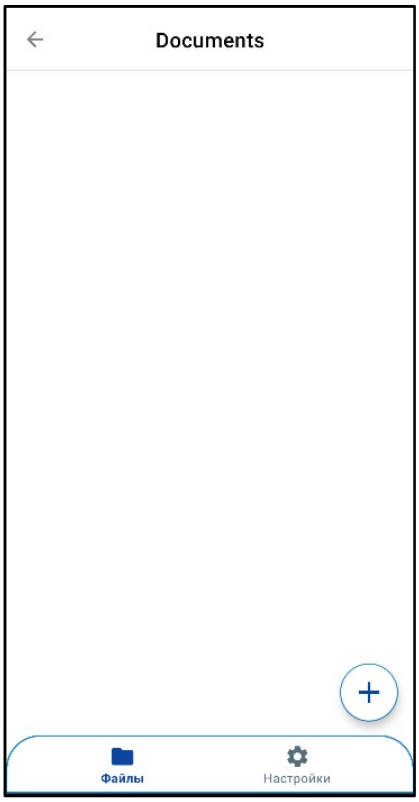


Рисунок 15 – Открытие папки

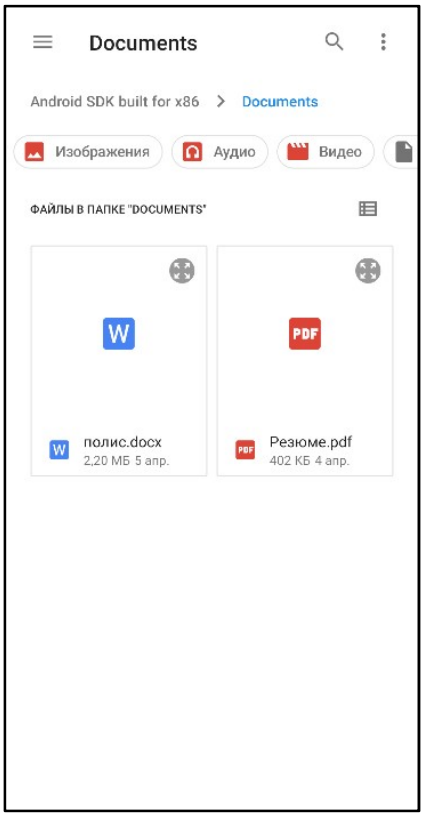


Рисунок 16 – Окно выбора файла

В случае успешной отправки файла, пользователю отображается соответствующее сообщение и загруженный файл на экране (рисунок 17).

Чтобы загрузить файл с сервера на мобильное устройство необходимо нажать на иконку “загрузить”. После завершения загрузки файл находится в папке “загрузки” на мобильном устройстве (рисунок 18).

Для того, чтобы удалить файл с сервера, пользователю необходимо в меню выбрать пункт“удалить”. В результате выполнения операции файл будет удален с сервера и графического интерфейса клиента (рисунок 19).

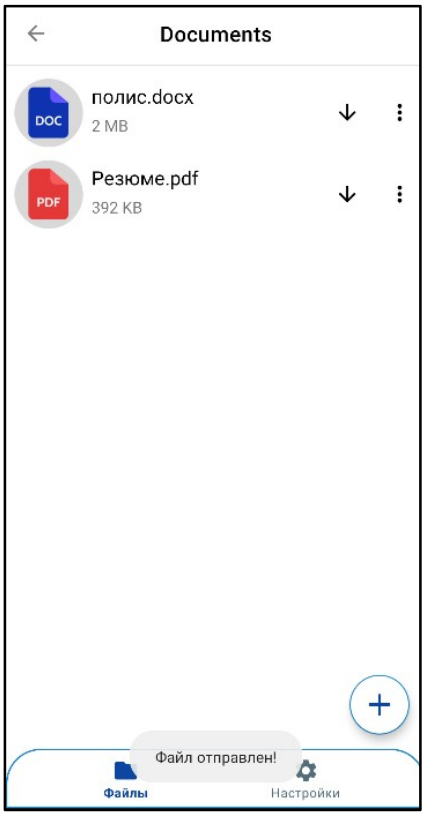


Рисунок 17 – Отображение загруженных файлов

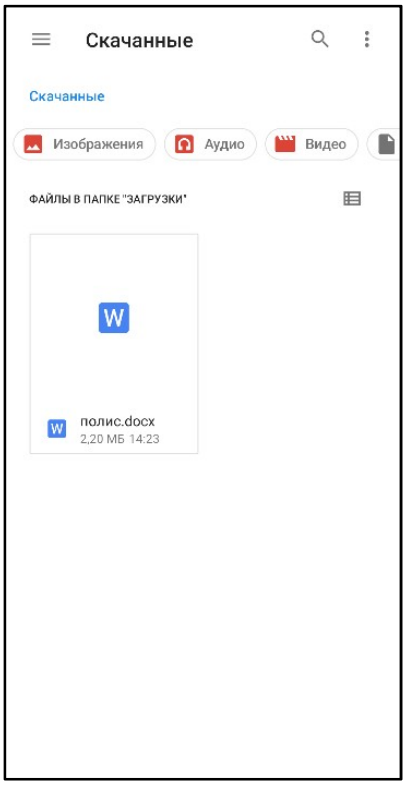


Рисунок 18– Загруженный файл

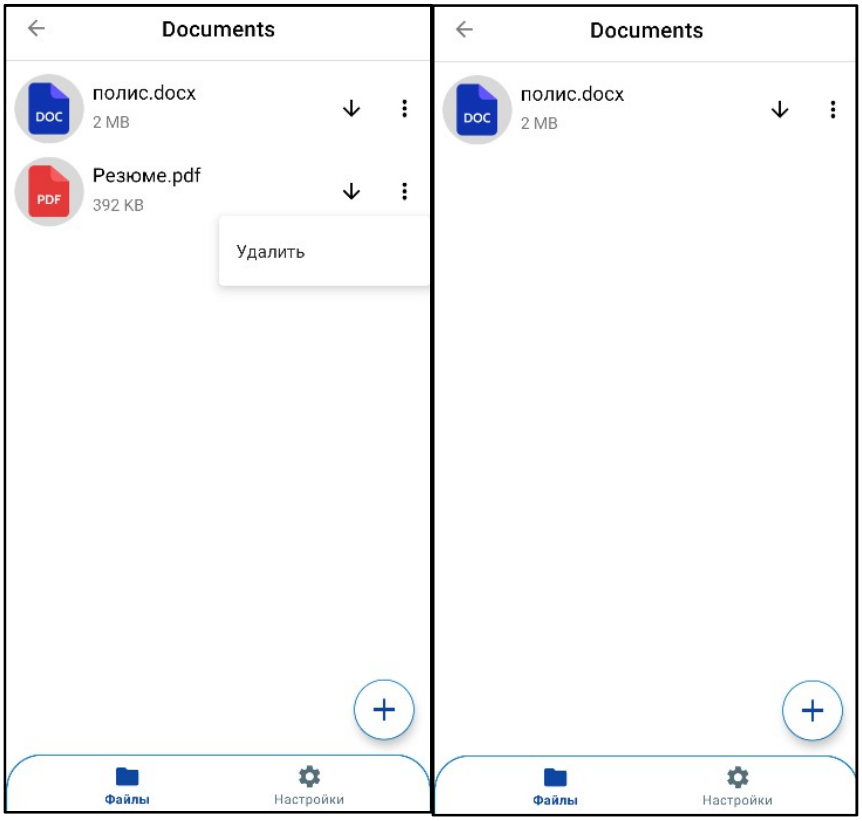


Рисунок 19 – Удаление файла

2 Безопасность жизнедеятельности

2.1 Требования к помещениям для работы с ПК

Помещения, предназначенные для эксплуатации ПК, должны иметь как естественное, так и искусственное освещение. Использование ПК в помещениях без естественного освещения разрешается только при наличии соответствующего обоснования и положительного санитарно-эпидемиологического заключения, выданного в установленном порядке.

Искусственное освещение в помещениях для эксплуатации ПК должно осуществляться системой общего равномерного освещения. В качестве источников света при искусственном освещении следует преимущественно использовать люминесцентные лампы с рассеивателями и экранирующими решетками. Применение светильников без рассеивателей и экранирующих решеток не допускается. В светильниках местного освещения допускается использование ламп накаливания, в том числе галогенных. Для поддержания требуемых значений освещенности в помещениях для использования ПК следует регулярно чистить стекла оконных рам и светильников не реже двух раз в год и своевременно заменять перегоревшие лампы.

Помещения, где располагаются рабочие места с ПК, должны быть оборудованы защитным заземлением. В этих помещениях следует ежедневно проводить влажную уборку и регулярно проветривать помещение после каждого часа работы на ПК. Также необходимо обеспечить наличие систем отопления, кондиционирования воздуха, чтобы обеспечить оптимальную температуру и влажность.

В помещении, где размещаются рабочие места с ПК, необходимо иметь аптечку первой медицинской помощи для оказания неотложной помощи при несчастных. Также, для обеспечения безопасности, необходимо иметь углекислотный огнетушитель, который может быть использован для тушения пожаров.

2.2 Требования к организации рабочих мест

При размещении ПК в помещениях культурно-развлекательных учреждений с использованием жидкокристаллических или плазменных экранов площадь на одно рабочее место должна быть не менее 4,5 м².

Для обеспечения комфортного пространства между рабочими столами с мониторами расстояние между ними должно быть не менее 2 метров, а расстояние между боковыми поверхностями мониторов - не менее 1,2 метра.

Рабочие столы следует располагать таким образом, чтобы мониторы были ориентированы боковой стороной к окнам, таким образом естественный свет падает преимущественно слева.

Для выполнения творческой работы, требующей значительного умственного напряжения, рекомендуется изолировать рабочие места с компьютерами друг от друга перегородками высотой 1,5–2,0 метра.

Экран монитора должен находиться на расстоянии 600–700 мм от глаз пользователя, с учетом размеров алфавитно-цифровых знаков и символов, но не ближе 500 мм. Клавиатура должна быть расположена на поверхности стола на расстоянии 100–300 мм от края.

Рабочий стол может иметь различные конструкции, отвечающие современным требованиям эргономики и позволяющие удобно разместить на рабочей поверхности необходимое оборудование. Желательно использование столов, которые имеют отдельную рабочую поверхность для размещения клавиатуры. Могут применяться столы с регулируемой и нерегулируемой высотой рабочей поверхности. Если стол не регулируется, его высота должна быть в пределах от 680 до 800 мм.

Для обеспечения физиологически правильной рабочей позы в течение рабочего дня рекомендуется использовать подъемно-поворотные рабочие стулья с регулируемыми по высоте и углам наклона сиденьями и спинками.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано программное обеспечение для облачного хранения данных на устройствах под управлением ОС *Android*с использованием клиент-серверной архитектуры. Мобильное приложение позволяет пользователям удобно сохранять, синхронизировать и получать доступ к своим данным с различных устройств.

Процесс разработки включал в себя анализ требований, выбор архитектуры программного обеспечения, разработку сервера, пользовательского интерфейса и реализацию функциональности для загрузки, синхронизации и управления данными.

Разработанное программное обеспечение представляет практическую ценность для пользователей, предоставляя им удобство при хранении и доступе к их данным на удаленном сервере.

В целом, выполнение данной работы позволило получить ценный опыт и глубже понять процессы, связанные с сетевыми технологиями и разработкой мобильных приложений.

Список используемых источников

1. Блох, Дж. *Java*: эффективное программирование / Дж. Блох. - М.: Диалектика, 2019. - 464 с.
2. Хабибуллин И. Ш. Создание распределенных приложений на *Java* 2. — СПб.: БХВ-Петербург, 2002. — 704 с.
3. Курняван, Б. Программирование *WEB*-приложений на языке *Java* / Б. Курняван. - М.: Лори, 2014. - 880 с.
4. *Retrofit*.: [Электронный ресурс] *URL*:*https://square.github.io/retrofit/*
5. Официальная документация*Android*.: [Электронный ресурс].*URL*: *https://developer.android.com/docs*
6. *Spring Boot*.: [Электронный ресурс] *URL*: *https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/*
7. Уоллс К. *Spring* в действии. – М.: ДМК Пресс, 2013. – 752 с.

Приложение А

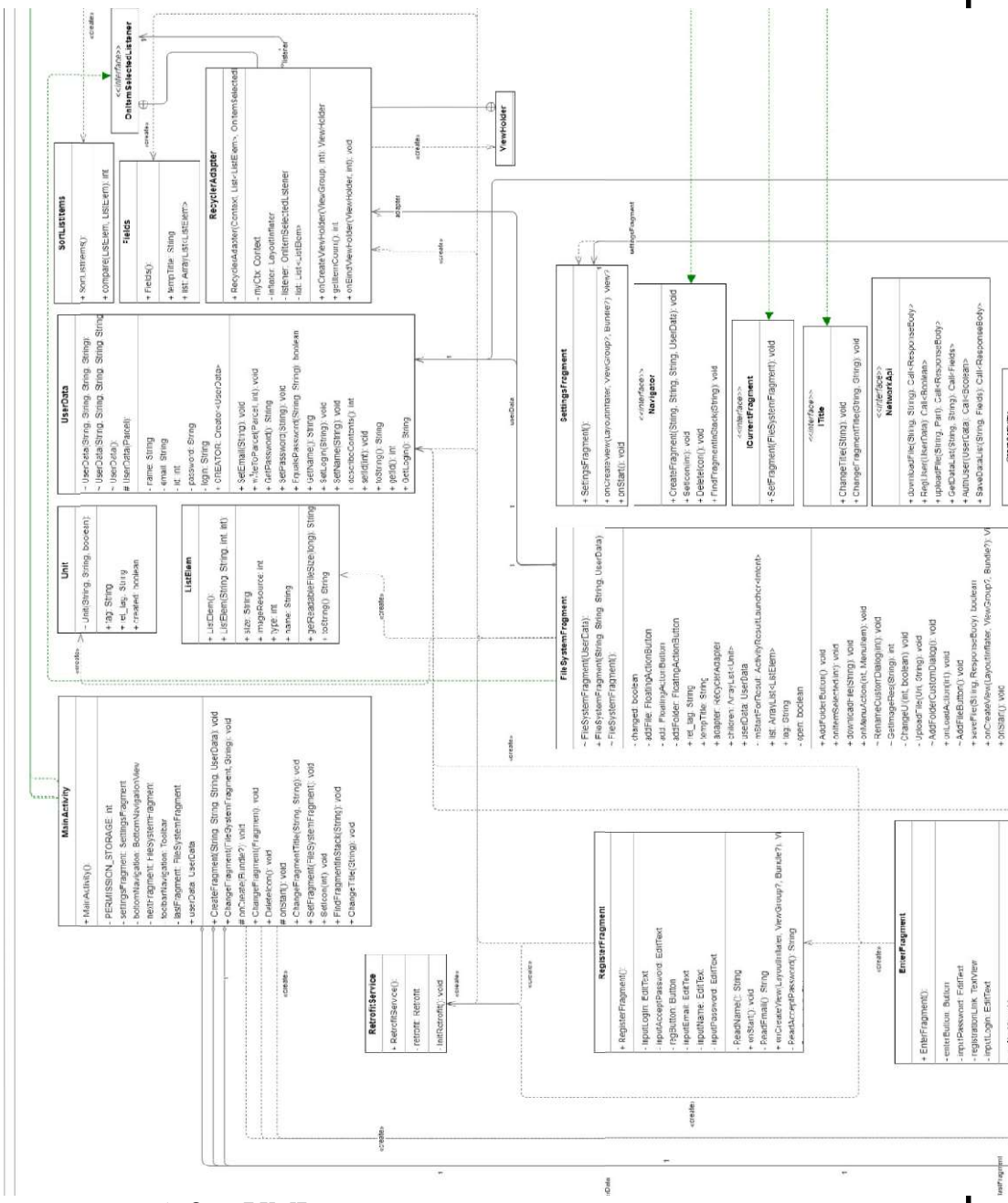
(обязательное)

Программное обеспечение для облачного хранения данных

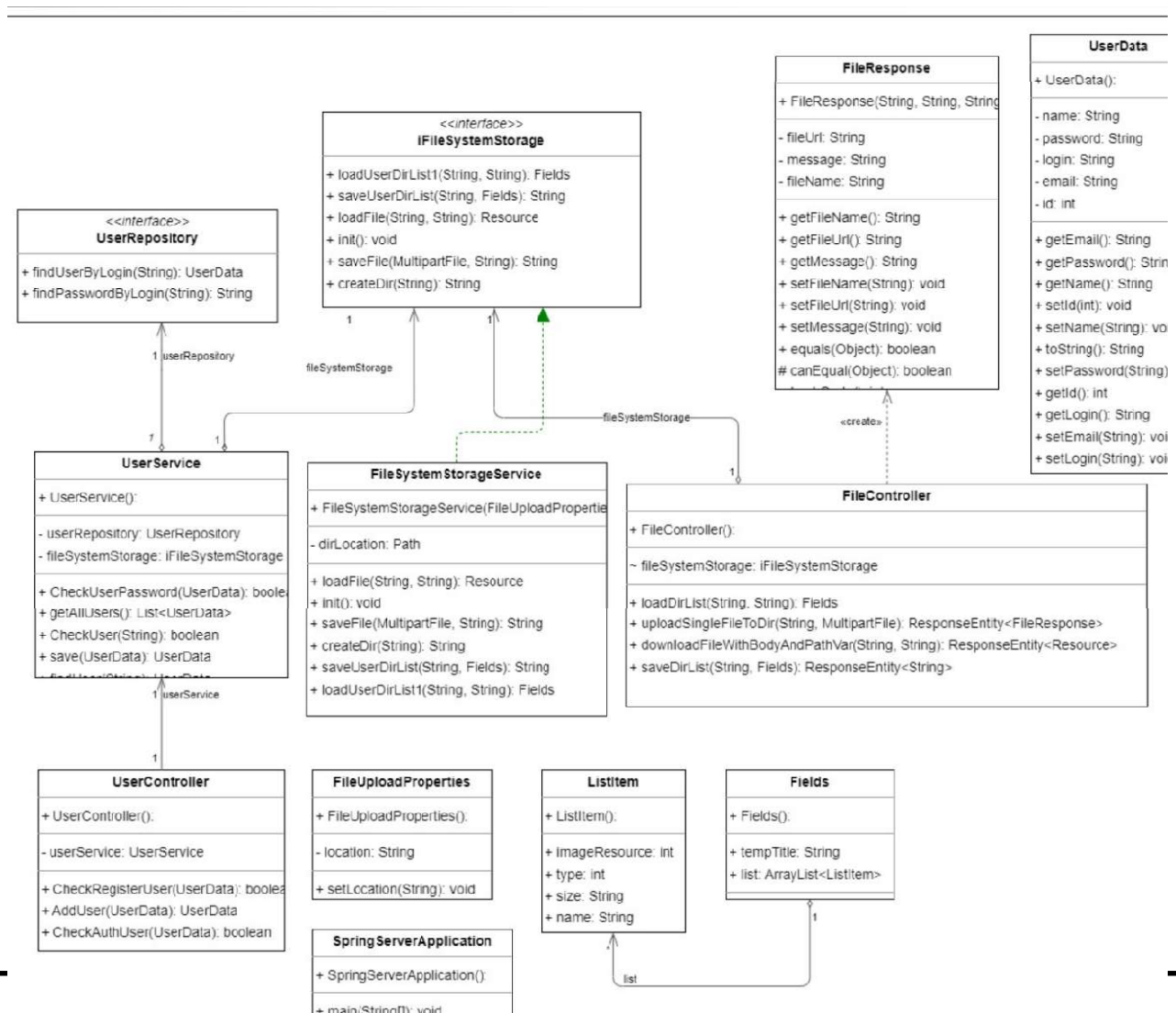
на устройствах под управлением ОС *Android*

UML-диаграммы

Приложение А.1 – *UML*–диаграмма классов клиента



Приложение А.2 – *UML*–диаграмма классов сервера

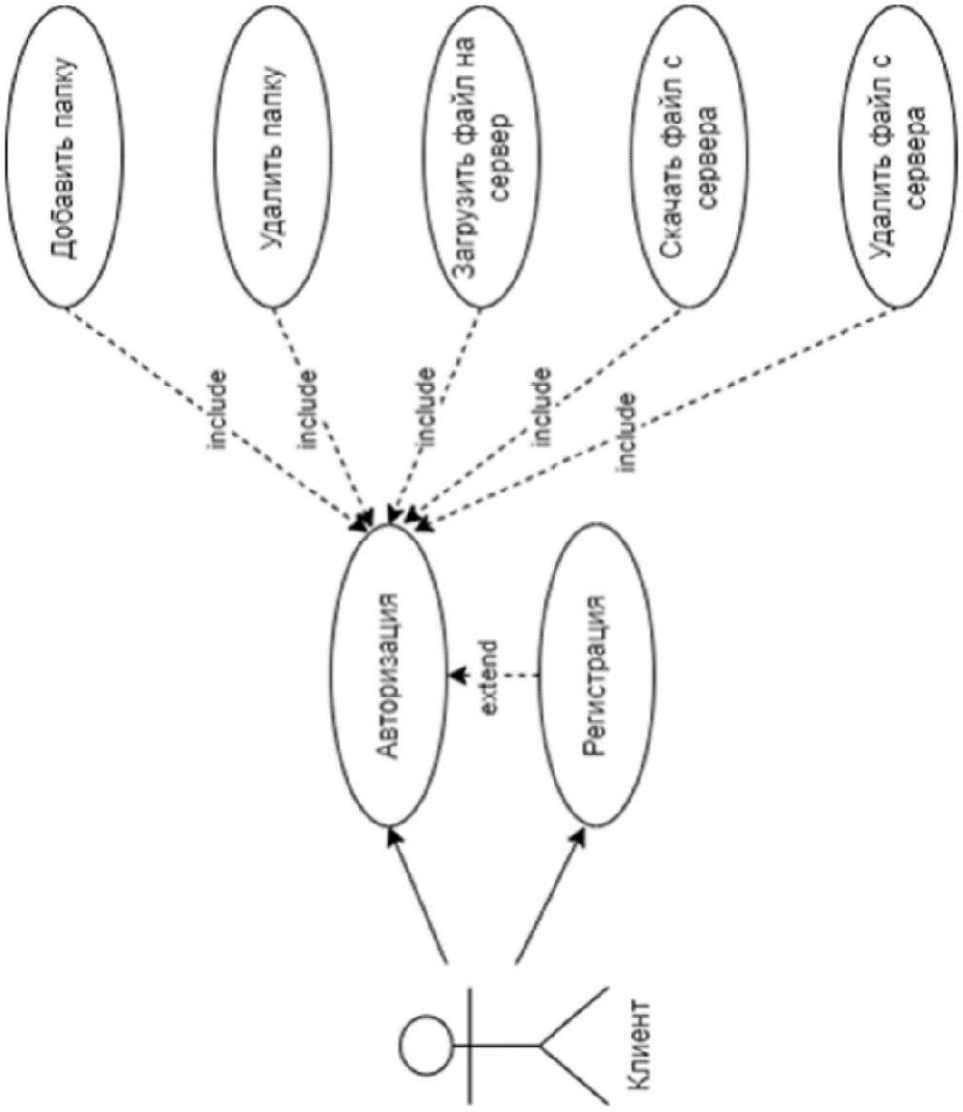


Изм.	Лист	№ докум.	Подпись	Дат

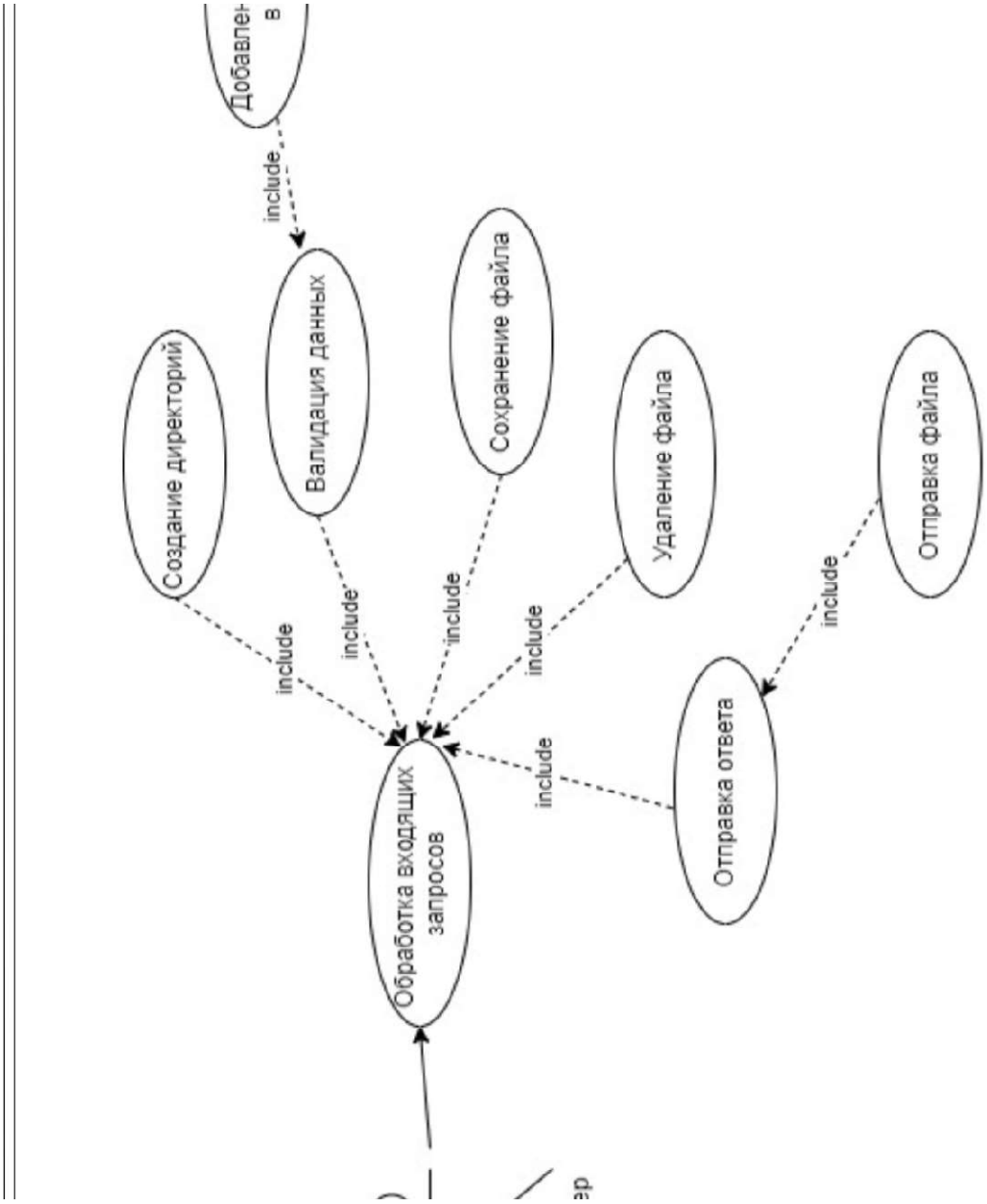
ПГУ1.090301.34.001 ПЗ

Лист 48

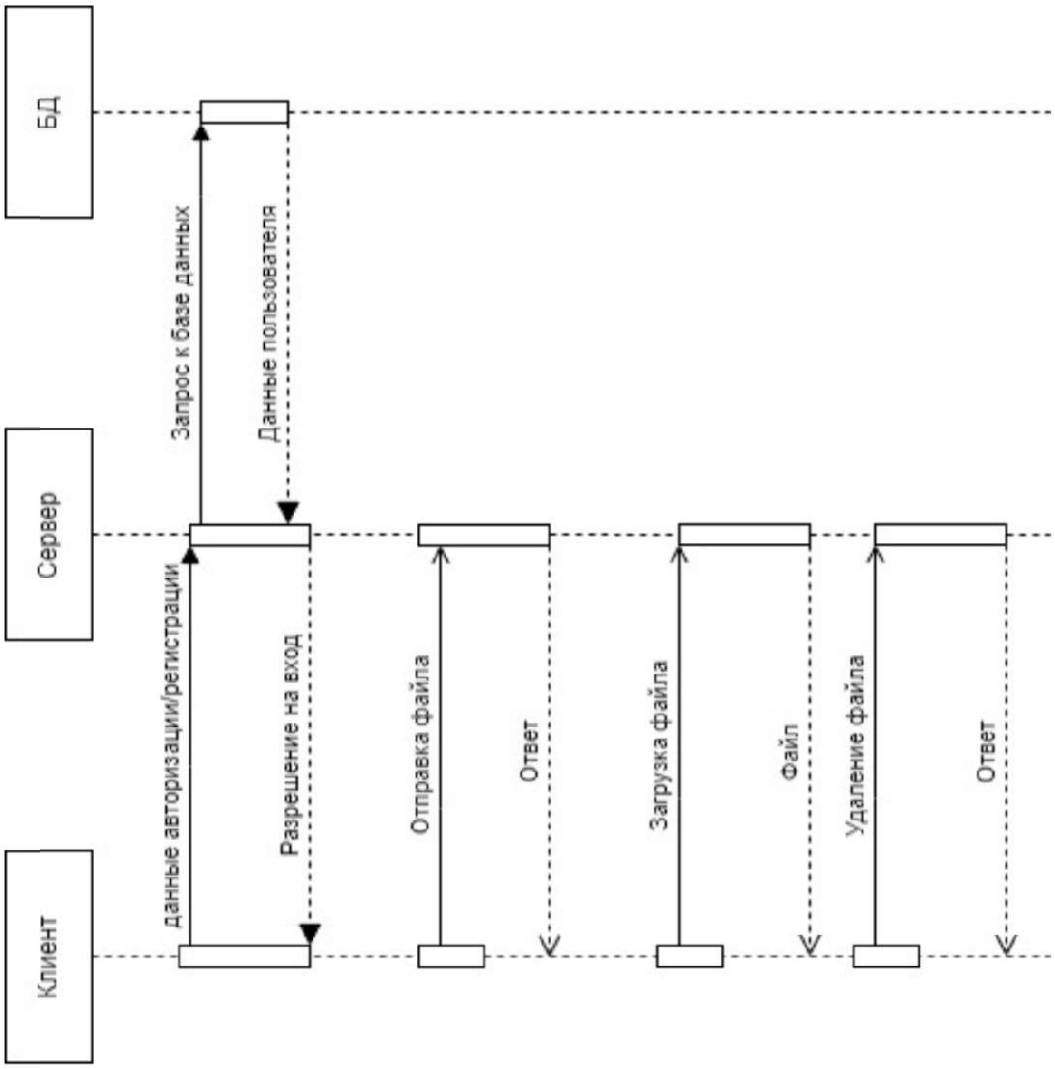
Приложение А.3 – *UML*–диаграмма прецедентов клиента



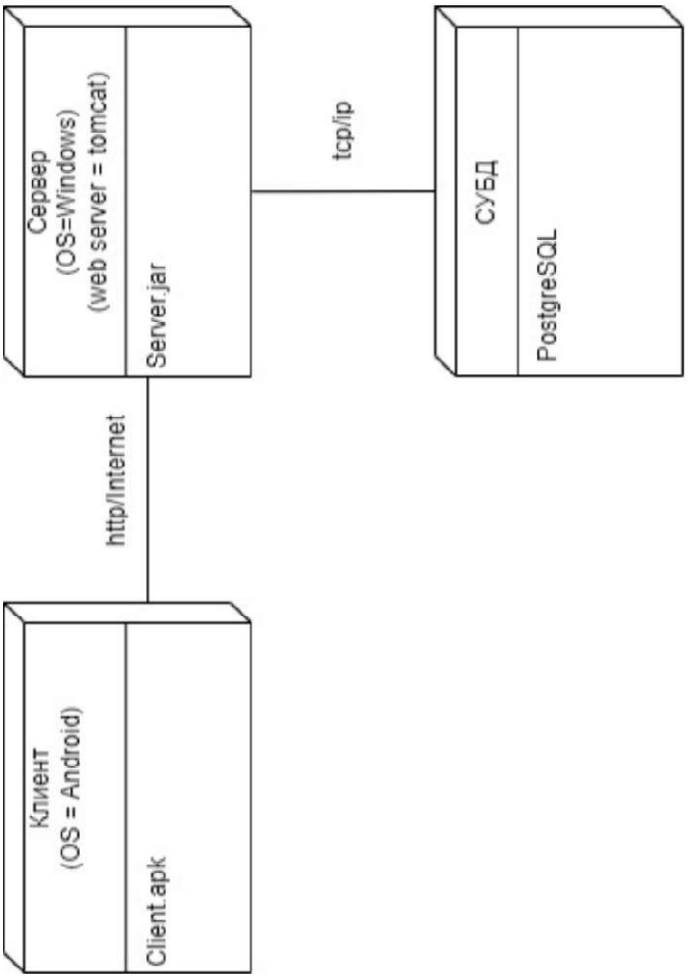
Приложение А.4 – *UML*–диаграмма прецедентов сервера



Приложение А.5 – *UML*–диаграмма последовательностей



Приложение А.6 – *UML*–диаграмма развертывания



ПриложениеБ

(обязательное)

Программное обеспечение для облачного хранения данных

на устройствах под управлением ОС *Android*

Листинг клиентского приложения

Файл «*EnterFragment.java*»

```
import ...

public class EnterFragment extends Fragment {

    private EditTextinputLogin;
    private EditTextinputPassword;
    private TextViewregistrationLink;
    private Button enterButton;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container, @Nullable Bundle
savedInstanceState) {
        View view =
inflater.inflate(R.layout.authorization_fragment, container,
false);
inputLogin = view.findViewById(R.id.a_input_login);
inputPassword = view.findViewById(R.id.a_input_password);
registrationLink = view.findViewById(R.id.a_registration_link);
enterButton = view.findViewById(R.id.a_enter_button);
        return view;
    }

    @Override
    public void onStart() {
super.onStart();

RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);

registrationLink.setOnClickListener(v -
>getActivity().getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .replace(R.id.fragmentContainerView, new
RegisterFragment())
                .commit());

enterButton.setOnClickListener(v -> {
            String login = ReadLogin();
            String password = ReadPassword();

UserDatauserData = new UserData();
userData.SetLogin(login);
userData.SetPassword(password);

            new Thread(() -> {
                try {
                    Call<Boolean> call =
```

```
networkApi.AuthUser(userData);
                    Response<Boolean> response = call.execute();
                    if(Boolean.TRUE.equals(response.body())){
                        Intent intent = new Intent(getContext(),
MainActivity.class);
intent.putExtra(UserData.class.getSimpleName(), userData);
startActivity(intent);
                    } else {
                        new
Handler(Looper.getMainLooper()).post(() -> {
Toast.makeText(getActivity(), "Неверныйлогинилипароль!",
Toast.LENGTH_SHORT).show();
inputPassword.setText("");
inputLogin.setText("");
                        });
                    }
                } catch (IOException e) {
e.printStackTrace();
                }
            }).start();
        });
    }

    public String ReadPassword() {
        return inputPassword.getText().toString().trim();
    }

    private String ReadLogin() {
        return inputLogin.getText().toString().trim();
    }
}
```

Файл «*RegisterFragment.java*»

```
import ...
public class RegisterFragment extends Fragment {

    private EditTextinputName;
    private EditTextinputLogin;
    private EditTextinputEmail;
    private EditTextinputPassword;
    private EditTextinputAcceptPassword;
    private Button regButton;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container, @Nullable Bundle
savedInstanceState) {
        View view =
inflater.inflate(R.layout.registration_fragment, container,
false);
```

```
inputName = view.findViewById(R.id.r_input_name);
inputLogin = view.findViewById(R.id.r_input_login);
inputEmail = view.findViewById(R.id.r_input_email);
inputPassword = view.findViewById(R.id.r_input_password);
inputAcceptPassword =
view.findViewById(R.id.r_input_accept_password);
regButton = view.findViewById(R.id.registration_button);
        return view;
    }

    @Override
    public void onStart() {
super.onStart();

RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);

regButton.setOnClickListener(v -> {
            String name = ReadName();
            String login = ReadLogin();
            String email = ReadEmail();
            String password = ReadPassword();
            String accept = ReadAcceptPassword();

            if(UserData.EqualsPassword(password, accept)){
UserDatauserData = new UserData();
userData.SetName(name);
userData.SetLogin(login);
userData.SetEmail(email);
userData.SetPassword(password);

                new Thread(() -> {
                    try {
                        Call<Boolean> call =
networkApi.RegUser(userData);
                        Response<Boolean> response =
call.execute();
                        if(Boolean.TRUE.equals(response.body())){
                            Intent intent = new
Intent(getContext(), MainActivity.class);
intent.putExtra(UserData.class.getSimpleName(), userData);
startActivity(intent);
                        } else {
                            new
Handler(Looper.getMainLooper()).post(() -> {
Toast.makeText(getActivity(), "Пользователь уже существует!",
Toast.LENGTH_SHORT).show();
inputLogin.setText("");
                            });
                        }
                    } catch (IOException e) {
```

```
e.printStackTrace();
                    }
                }).start();

            }else{
Toast.makeText(getActivity(), "Паролинесовпадают!",
Toast.LENGTH_SHORT).show();
inputPassword.setText("");
inputAcceptPassword.setText("");
            }
        });
    }

    private String ReadAcceptPassword() {
        return inputAcceptPassword.getText().toString().trim();
    }

    private String ReadPassword() {
        return inputPassword.getText().toString().trim();
    }

    private String ReadEmail() {
        return inputEmail.getText().toString().trim();
    }

    private String ReadLogin() {
        return inputLogin.getText().toString().trim();
    }

    private String ReadName() {
        return inputName.getText().toString().trim();
    }
}
```

Файл «*RetrofitService.java*»

```
public class RetrofitService {

    private Retrofit retrofit;

    public RetrofitService(){
InitRetrofit();
    }

    public Retrofit getRetrofit() {
        return retrofit;
    }


 private void InitRetrofit() {
        retrofit = new Retrofit.Builder()
                .baseUrl("http://192.168.0.104:9000")
```

```
.addConverterFactory(GsonConverterFactory.create(new Gson()))
                .build();
    }
}
```

Файл «*NetworkApi.java*»

```
import ...
public interface NetworkApi {


    @POST("/users/check_user")
    Call<Boolean>RegUser(@Body UserData user);

    @POST("/users/check_user_password")
    Call<Boolean>AuthUser(@Body UserData user);

    @POST("/api/load/{username}")
    Call<Fields>GetDataList(@Path("username") String username,
@Body String data);

    @POST("/api/save/{username}")
    Call<ResponseBody>SaveDataList(@Path("username") String
username, @Body Fields data);

    @Multipart
    @POST("/api/upload/{username}")
    Call<ResponseBody>uploadFile(@Path("username") String
username, @Part MultipartBody.Part file);

    @Streaming
    @POST("/api/download/{username}")
    Call<ResponseBody>downloadFile(@Path("username") String
username, @Body String data);

    @DELETE("/api/delete/{username}")
    Call<ResponseBody>deleteFile(@Path("username") String
username, @Body String data);

}
```

Файл «*FileSystemFragment.java*»

```
public class FileSystemFragment extends Fragment
        implements View.OnClickListener,
RecyclerAdapter.OnItemSelectedListener {

    private FloatingActionButton add;
    private FloatingActionButtonaddFile;
    private FloatingActionButtonaddFolder;
```

```
    public String tag;
    public String ret_tag;
    public String tempTitle;

    public ArrayList<ListElem> list = new ArrayList<>();
    public ArrayList<Unit> children = new ArrayList<>();
    public RecyclerAdapter adapter;

    private boolean open = false;
    private boolean changed = false;
    public UserDatauserData;

FileSystemFragment(UserDatauserData){
        super(R.layout.parent_fs_fragment);
        tag = "parent";
ret_tag = "";
tempTitle = "Файлы";
this.userData = userData;
InitList(tempTitle);
    }

    public FileSystemFragment(String tag, String ret_tag, String
tempTitle, UserDatauserData){
this.tag = tag;
this.ret_tag = ret_tag;
this.tempTitle = tempTitle;
this.userData = userData;
InitList(tempTitle);
    }

    @SuppressLint("NotifyDataSetChanged")
    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container, @Nullable Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.parent_fs_fragment,
container, false);
        add = view.findViewById(R.id.f_add_fab);
addFile = view.findViewById(R.id.f_add_file_fab);
addFolder = view.findViewById(R.id.f_add_folder_fab);
RecyclerViewrecyclerView = view.findViewById(R.id.recycler_view);
recyclerView.setLayoutManager(new
LinearLayoutManager(getContext()));
        adapter = new RecyclerAdapter(getContext(), list,  this);
recyclerView.setAdapter(adapter);
adapter.notifyDataSetChanged();
        return view;
    }

    @Override
    public void onStart() {
```

```
super.onStart();
        if(Objects.equals(tag, "parent")){

Objects.requireNonNull((ITitle)getActivity()).ChangeTitle("Файлы"
);
            ((Navigator)getActivity()).DeleteIcon();
        }else{

Objects.requireNonNull((ITitle)getActivity()).ChangeTitle(tempTit
le);

((Navigator)getActivity()).SetIcon(R.drawable.ic_baseline_arrow_b
ack_24);
        }
add.setOnClickListener(this);
addFile.setOnClickListener(this);
addFolder.setOnClickListener(this);
    }


    @SuppressLint("NonConstantResourceId")
    @Override
    public void onClick(View v) {
        switch(v.getId()){
            case R.id.f_add_fab:
this.ShowButton();
                break;
            case R.id.f_add_file_fab:
this.AddFileButton();
                break;
            case R.id.f_add_folder_fab:
this.AddFolderButton();
                break;
        }
    }


    @Override
    public void onItemSelected(int position) {
ListElemcurr = list.get(position);
        if(curr.type == 1){

Objects.requireNonNull((ICurrentFragment)getActivity()).SetFragme
nt(this);
            if(children.get(position).created){

((Navigator)getActivity()).FindFragmentInStack(children.get(posit
ion).tag);
            }else{

((Navigator)getActivity()).CreateFragment(children.get(position).
tag, children.get(position).ret_tag, curr.name, userData);
```

```
children.get(position).created = true;
            }
        }
    }


    @SuppressLint({"NonConstantResourceId",
"NotifyDataSetChanged"})
    @Override
    public void onMenuAction(int position, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.delete_menu:
                if(list.get(position).type == 1){
children.remove(position);
                }
list.remove(position);
adapter.notifyDataSetChanged();
                break;
            case R.id.rename_menu:
RenameCustomDialog(position);
                break;
        }
    }


    @Override
    public void onLoadAction(int position) {
        String name = list.get(position).name;
downloadFile(name);
    }

    public void downloadFile(String filename){
RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);
ExecutorServiceexecutorService =
Executors.newSingleThreadExecutor();
networkApi.downloadFile(userData.GetLogin(),
filename).enqueue(new Callback<ResponseBody>() {
            @SuppressLint("StaticFieldLeak")
            @Override
            public void onResponse(@NonNull Call<ResponseBody>
call, @NonNull Response<ResponseBody> response) {
executorService.execute(() -> {
                    if(saveFile(filename,
Objects.requireNonNull(response.body()))){
                        new
Handler(Looper.getMainLooper()).post(() -> {
Toast.makeText(getActivity(), "Файлзагружен",
Toast.LENGTH_SHORT).show();
                        });
                    }
```

```
                });
            }
            @Override
            public void onFailure(@NonNull Call<ResponseBody>
call, @NonNull Throwable t) {
t.printStackTrace();
            }
        });
    }

    public booleansaveFile(String filename, ResponseBody body){
        File newFile = new
File(Environment.getExternalStoragePublicDirectory(Environment.DI
RECTORY_DOWNLOADS), filename);
InputStreaminputStream = null;
OutputStreamoutputStream = null;
        try {
            byte[] fileReader = new byte[4096];
inputStream = body.byteStream();
outputStream = new FileOutputStream(newFile);

            while (true) {
                int read = inputStream.read(fileReader);
                if (read == -1) break;
outputStream.write(fileReader, 0, read);
            }
outputStream.flush();
            return true;
        } catch (IOException e) {
e.printStackTrace();
            try{
inputStream.close();
                if (outputStream != null) {
outputStream.close();
                }
                return false;
            }catch (IOException a){
e.printStackTrace();
                return false;
            }
        }
    }


    public void AddFolderButton(){
AddFolderCustomDialog();
addFile.setVisibility(View.INVISIBLE);
addFolder.setVisibility(View.INVISIBLE);
        open = false;
    }

ActivityResultLauncher<Intent>mStartForResult =
```

```
registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
            new ActivityResultCallback<ActivityResult>() {
                @SuppressLint("NotifyDataSetChanged")
                @Override
                public void onActivityResult(ActivityResult
result) {
                    if (result.getResultCode() ==
Activity.RESULT_OK) {
                        Uri uri =
Objects.requireNonNull(result.getData()).getData();

                        Cursor returnCursor =
requireContext().getContentResolver().query(uri, null, null,
null, null);
                        int nameIndex =
returnCursor.getColumnIndex(OpenableColumns.DISPLAY_NAME);
                        int sizeIndex =
returnCursor.getColumnIndex(OpenableColumns.SIZE);
returnCursor.moveToFirst();

                        String name =
returnCursor.getString(nameIndex);
                        long size =
returnCursor.getLong(sizeIndex);
                        int resourceImage = GetImageRes(name);

UploadFile(uri, name);

list.add(new ListElem(name, ListElem.getReadableFileSize(size),
0, resourceImage));
list.sort(new SortListItems());
adapter.notifyDataSetChanged();
returnCursor.close();
                        changed = true;
                    }
                }
            });

    private void UploadFile(Uri uri, String name) {
        File file = new
File(RealPathUtil.getRealPath(requireContext(), uri));

RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);

RequestBodyrequestFile =
RequestBody.create(MediaType.parse(getContext().getContentResolve
r().getType(uri)), file);
MultipartBody.Part body =
MultipartBody.Part.createFormData("file", name, requestFile);
```

```
networkApi.uploadFile(userData.GetLogin(), body).enqueue(new
Callback<ResponseBody>() {
            @Override
            public void onResponse(@NonNull Call<ResponseBody>
call, @NonNull Response<ResponseBody> response) {
                if(response.code() == 200){
System.out.println(response.body());
                    new Handler(Looper.getMainLooper()).post(() -
> {
Toast.makeText(getActivity(), "Файлотправлен!",
Toast.LENGTH_SHORT).show();
                    });
                }
            }

            @Override
            public void onFailure(@NonNull Call<ResponseBody>
call, @NonNull Throwable t) {
t.printStackTrace();
            }
        });
    }


    void AddFileButton(){
Toast.makeText(getActivity(),"добавитьфайл",Toast.LENGTH_SHORT).s
how();
ChangeUI(View.INVISIBLE, false);
        Intent myFiles = new Intent(Intent.ACTION_GET_CONTENT);
myFiles.setType("*/*");
myFiles.addCategory(Intent.CATEGORY_OPENABLE);
mStartForResult.launch(myFiles);
    }


    void ShowButton(){
        if(open)
ChangeUI(View.INVISIBLE, false);
        else
ChangeUI(View.VISIBLE, true);
    }

    private void ChangeUI(int invisible, boolean b) {
addFile.setVisibility(invisible);
addFolder.setVisibility(invisible);
        open = b;
    }

    @SuppressLint("NotifyDataSetChanged")
    void AddFolderCustomDialog() {
        Dialog dialog = new Dialog(getContext());
```

```
dialog.setContentView(R.layout.custom_dialog);
dialog.setCancelable(true);
dialog.getWindow().setBackgroundDrawable(new
ColorDrawable(Color.TRANSPARENT));

        Button d_add_folder =
dialog.findViewById(R.id.dialog_add_button);
        Button cancel =
dialog.findViewById(R.id.dialog_cancel_button);
EditTextnameFolder = dialog.findViewById(R.id.folder_name);
TextView title = dialog.findViewById(R.id.dialog_text);

title.setText("Новаяпапка");

d_add_folder.setOnClickListener(v -> {
            String name_folder = nameFolder.getText().toString();
list.add(new ListElem(name_folder, "12kb", 1,
R.drawable.folder_icon2)); // добавлениепапки в список
list.sort(new SortListItems());
adapter.notifyDataSetChanged();
children.add(new Unit(UUID.randomUUID().toString(), tag, false));
            changed = true;
dialog.cancel();
        });
cancel.setOnClickListener(v ->dialog.cancel());
dialog.show();
    }


    @SuppressLint("NotifyDataSetChanged")
    void RenameCustomDialog(int position) {
        Dialog dialog = new Dialog(getContext());
dialog.setContentView(R.layout.custom_dialog);
dialog.setCancelable(true);
dialog.getWindow().setBackgroundDrawable(new
ColorDrawable(Color.TRANSPARENT));

        Button d_add_folder =
dialog.findViewById(R.id.dialog_add_button);
        Button cancel =
dialog.findViewById(R.id.dialog_cancel_button);
EditTextnameFolder = dialog.findViewById(R.id.folder_name);
TextView title = dialog.findViewById(R.id.dialog_text);

title.setText("Переименовать");

d_add_folder.setOnClickListener(v -> {
            String name_folder = nameFolder.getText().toString();
list.get(position).name = name_folder;  // измененииеназвания
adapter.notifyDataSetChanged();
            if(children.get(position).created){
```

```
Objects.requireNonNull((ITitle)getActivity()).ChangeFragmentTitle
(name_folder, children.get(position).tag);
            }
dialog.cancel();
        });
cancel.setOnClickListener(v ->dialog.cancel());
dialog.show();
    }


    int GetImageRes(String name){
        if (name.endsWith("png")) return R.drawable.png_icon;
        if (name.endsWith("doc") || name.endsWith("docx")) return
R.drawable.doc_icon;
        if (name.endsWith("mp3")) return R.drawable.mp3_icon;
        if (name.endsWith("mp4")) return R.drawable.mp4_icon;
        if (name.endsWith("pdf")) return R.drawable.pdf_icon;
        if (name.endsWith("ppt")) return R.drawable.ppt_icon;
        if (name.endsWith("txt")) return R.drawable.txt_icon;
        if (name.endsWith("xls")) return R.drawable.xls_icon;
        return 0;
    }

    void InitList(String title){
RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);
networkApi.GetDataList(userData.GetLogin(), title).enqueue(new
Callback<Fields>() {
                @SuppressLint("NotifyDataSetChanged")
                @Override
                public void onResponse(Call<Fields> call,
Response<Fields> response) {
                    if(response.body()!=null){
list.addAll(response.body().list);
adapter.notifyDataSetChanged();
IntStream.range(0, list.size()).forEach(i ->children.add(new
Unit(UUID.randomUUID().toString(), tag, false)));
                    }
                }
                @Override
                public void onFailure(Call<Fields> call,
Throwable t) {
t.printStackTrace();
                }
            });
    }

    @Override
    public void onStop() {
super.onStop();
        if(changed){
```

```
            changed = false;
            Fields a = new Fields();
a.tempTitle = tempTitle;
a.list = list;
RetrofitServiceretrofitService  = new RetrofitService();
NetworkApinetworkApi =
retrofitService.getRetrofit().create(NetworkApi.class);
networkApi.SaveDataList(userData.GetLogin(), a).enqueue(new
Callback<ResponseBody>() {
                @Override
                public void onResponse(Call<ResponseBody> call,
Response<ResponseBody> response) {
System.out.println(response.body());
                }
                @Override
                public void onFailure(Call<ResponseBody> call,
Throwable t) {
t.printStackTrace();
                }
            });
        }
    }
}
```

Файл «*RecyclerAdapter.java*»

```
public class RecyclerAdapter extends
RecyclerView.Adapter<RecyclerAdapter.ViewHolder> {

    private OnItemSelectedListener listener;
    private final LayoutInflater inflater;
    private final List<ListElem> list;
    private final Context myCtx;

    public RecyclerAdapter(Context myCtx, List<ListElem> list,
OnItemSelectedListener listener) {
this.listener   = listener;
this.inflater   = LayoutInflater.from(myCtx);
this.list       = list;
this.myCtx      = myCtx;
    }

    @NonNull
    @Override
    public RecyclerAdapter.ViewHolderonCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
        View view = inflater.inflate(R.layout.list_item, parent,
false);
        return new ViewHolder(view);
    }

    @Override
```

```
    public void onBindViewHolder(@NonNull
RecyclerAdapter.ViewHolder holder, int position) {
ListElemelem = list.get(position);

holder.imageItem.setImageResource(elem.imageResource);
holder.nameItem.setText(elem.name);
holder.sizeItem.setText(elem.size);

        if(elem.type == 0){
holder.imageItem.setImageResource(elem.imageResource);
holder.nameItem.setText(elem.name);
holder.sizeItem.setText(elem.size);
        }else{
holder.imageItem.setImageResource(elem.imageResource);
holder.nameItem.setText(elem.name);
holder.sizeItem.setVisibility(View.INVISIBLE);
holder.loadItemButton.setVisibility(View.INVISIBLE);
        }

holder.loadItemButton.setOnClickListener(v -> {
            if (listener == null) return;
            int position1 = holder.getAdapterPosition();
listener.onLoadAction(position1);
        });

holder.menuItemButton.setOnClickListener(v -> {
PopupMenu popup = new PopupMenu(myCtx, holder.menuItemButton);
popup.inflate(R.menu.item_menu);

popup.setOnMenuItemClickListener(item -> {
                if (listener == null) return false;
                int position1 = holder.getAdapterPosition();
listener.onMenuAction(position1, item);
                return false;
            });
popup.show();
        });
    }

    @Override
    public int getItemCount() {
        return list.size();
    }

    public interface OnItemSelectedListener {
        void onItemSelected(int position);
        void onMenuAction(int position, MenuItem item);
        void onLoadAction(int position);
    }

    public class ViewHolder extends RecyclerView.ViewHolder
implements View.OnClickListener {
```

```
        final ImageViewimageItem;
        final TextViewnameItem;
        final TextViewsizeItem;
        final ImageButtonloadItemButton;
        final ImageButtonmenuItemButton;

        public ViewHolder(@NonNull View itemView) {
            super(itemView);
imageItem = itemView.findViewById(R.id.image_list_item);
nameItem = itemView.findViewById(R.id.name_list_item);
sizeItem = itemView.findViewById(R.id.size_list_item);
loadItemButton = itemView.findViewById(R.id.load_list_item);
menuItemButton = itemView.findViewById(R.id.menu_list_item);
itemView.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            int position = getAdapterPosition();
            if (listener == null) return;
listener.onItemSelected(position);
        }
    }
}
```

Файл «*MainActivity.java*»

```
public class MainActivity extends AppCompatActivity implements
ITitle, ICurrentFragment, Navigator {

    private static final int PERMISSION_STORAGE = 101;

    private BottomNavigationViewbottomNavigation;
    private Toolbar toolbarNavigation;
    private FileSystemFragmentlastFragment;
    private FileSystemFragmentnextFragment;
    private SettingsFragmentsettingsFragment;

    public UserDatauserData = new UserData();

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main_activity);

        Bundle arg = getIntent().getExtras();
        if(arg!=null){
userData = arg.getParcelable(UserData.class.getSimpleName());
        }
```

```
settingsFragment = new SettingsFragment();
nextFragment = new FileSystemFragment(userData);
lastFragment = nextFragment;

toolbarNavigation = findViewById(R.id.nav_toolbar);
toolbarNavigation.setTitle("Файлы");
setSupportActionBar(toolbarNavigation);

bottomNavigation = findViewById(R.id.bottomNavigationView);
ChangeFragment(nextFragment, "parent");

        if (!PermissionsUtils.hasPermissions(this)) {
            if
(PermissionsUtils.hasPermissions(MainActivity.this)) return;
PermissionsUtils.requestPermissions(MainActivity.this,
PERMISSION_STORAGE);
        }

    }

    @SuppressLint("NonConstantResourceId")
    @Override
    protected void onStart() {
super.onStart();
bottomNavigation.setOnItemSelectedListener(item -> {
            switch (item.getItemId()){
                case R.id.home:
ChangeFragment(nextFragment);
                    return true;
                case R.id.settings:
ChangeFragment(settingsFragment);
                    return true;
            }
            return false;
        });

toolbarNavigation.setNavigationOnClickListener(v -> {
FindFragmentInStack(nextFragment.ret_tag);
bottomNavigation.setSelectedItemId(R.id.home);
        });
    }

    @Override
    public void FindFragmentInStack(String tag) {
lastFragment = nextFragment;
nextFragment = (FileSystemFragment)
getSupportFragmentManager().findFragmentByTag(tag);
ChangeFragment(nextFragment, nextFragment.tag);
    }

    @Override
    public void CreateFragment(String tag, String ret_tag, String
```

```
title, UserDatauserData) {
lastFragment = nextFragment;
nextFragment = new FileSystemFragment(tag, ret_tag, title,
userData);
ChangeFragment(nextFragment, nextFragment.tag);
    }

    public void ChangeFragment(Fragment fragment){
getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .replace(R.id.fragment_container, fragment)
                .addToBackStack(null)
                .commit();
    }

    public void ChangeFragment(FileSystemFragment fragment,
String tag){
getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .replace(R.id.fragment_container, fragment, tag)
                .addToBackStack(null)
                .commit();
    }

    @Override
    public void ChangeTitle(String newTitle) {
toolbarNavigation.setTitle(newTitle);
    }

    @Override
    public void ChangeFragmentTitle(String newTitle, String tag)
{
nextFragment = (FileSystemFragment)
getSupportFragmentManager().findFragmentByTag(tag);
nextFragment.tempTitle = newTitle;
    }


    @Override
    public void SetFragment(FileSystemFragmentfr) {
lastFragment = fr;
    }


    @Override
    public void SetIcon(int resId) {
toolbarNavigation.setNavigationIcon(resId);
    }

    @Override
    public void DeleteIcon() {
toolbarNavigation.setNavigationIcon(null);
```

```
    }
}
```

Файл «*Navigator.java*»

```
public interface Navigator {
    void SetIcon(@DrawableRes int resId);
    void DeleteIcon();
    void FindFragmentInStack(String tag);
    void CreateFragment(String tag, String ret_tag, String title,
UserDatauserData);
}
```

ПриложениеВ

(обязательное)

Программноеобеспечениедляоблачногохраненияданных

наустройствахподуправлениемОС*Android*

Листингсерверногоприложения

Файл «*UserData.java*»

```
import jakarta.persistence.*;

@Entity
@Table(name = "User_Data")
public class UserData {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

    @Column(name = "USER_LOGIN")
    private String login;

    @Column(name = "USER_EMAIL")
    private String email;

    @Column(name = "USER_PASSWORD")
    private String password;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
this.login = login;
    }

    public String getEmail() {
        return email;
    }
```

```
    public void setEmail(String email) {
this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
this.password = password;
    }

    @Override
    public String toString() {
        return "UserData{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", login='" + login + '\'' +
                ", email='" + email + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

Файл «*UserRepository.java*»

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends CrudRepository<UserData,
Integer> {

    @Query(value = "SELECT * FROM User_Data WHERE USER_LOGIN =
?1", nativeQuery=true)
UserDatafindUserByLogin(String login);

    @Query(value = "SELECT USER_PASSWORD FROM User_Data WHERE
USER_LOGIN = ?1", nativeQuery=true)
    String findPasswordByLogin(String login);
}
```

Файл «*UserService.java*»

```
@Service
public class UserService {

    @Autowired
    private UserRepositoryuserRepository;
```

```
    @Autowired
    private iFileSystemStoragefileSystemStorage;

    public UserData save(UserDatauserData){
       return userRepository.save(userData);
    }

    public List<UserData>getAllUsers(){
        List<UserData> result = new ArrayList<>();
Streamable.of(userRepository.findAll()).forEach(result::add);
        return result;
    }

    public UserDatafindUser(String login){
        return userRepository.findUserByLogin(login);
    }

    public boolean CheckUser(String login){
UserDatauserData = userRepository.findUserByLogin(login);
        return userData != null;
    }

    public booleanCheckUserAndSave(UserDatauserData){
UserData temp =
userRepository.findUserByLogin(userData.getLogin());
        if(temp != null) return false;
this.save(userData);
fileSystemStorage.createDir(userData.getLogin());
        return true;
    }

    public booleanCheckUserPassword(UserDatauserData){
        if(!CheckUser(userData.getLogin())) return false;
        String temp =
userRepository.findPasswordByLogin(userData.getLogin());
        return temp.equals(userData.getPassword());
    }
}
```

Файл «*UserController.java*»

```
@RestController
public class UserController {

    @Autowired
    private UserServiceuserService;

    @PostMapping("/users/add")
    public UserDataAddUser(@RequestBody UserDatauserData){
        return userService.save(userData);
    }
```

```
    @PostMapping("/users/check_user")
    public booleanCheckRegisterUser(@RequestBody
UserDatauserData){
        return userService.CheckUserAndSave(userData);
    }

    @PostMapping("/users/check_user_password")
    public booleanCheckAuthUser(@RequestBody UserDatauserData){
        return userService.CheckUserPassword(userData);
    }
}
```

Файл «*FileSystemStorageService.java*»

```
@Service
public class FileSystemStorageService implements
iFileSystemStorage {
    private final Path dirLocation;

    @Autowired
    public
FileSystemStorageService(FileUploadPropertiesfileUploadProperties
) {
this.dirLocation = Paths.get(fileUploadProperties.getLocation())
                .toAbsolutePath()
                .normalize();
    }

    @Override
    @PostConstruct
    public void init() {
        try {
Files.createDirectories(this.dirLocation);
        }
        catch (Exception ex) {
ex.printStackTrace();
        }
    }

    @Override
    public String saveFile(MultipartFile file, String username) {
        try {
            String fileName = file.getOriginalFilename();
System.out.println(fileName);
            Path pFile = null;
            if(Files.exists(Paths.get(this.dirLocation + "\\"+
username + "\\files"))){
pFile = Paths.get(this.dirLocation + "\\"+ username +
"\\files").resolve(Objects.requireNonNull(fileName));
            }
Files.copy(file.getInputStream(), pFile,
StandardCopyOption.REPLACE_EXISTING);
```

```
System.out.println("File  saved");
            return fileName;
        } catch (Exception e) {
e.printStackTrace();
        }
        return null;
    }

    @Override
    public Resource loadFile(String fileName, String username) {
        try {
            Path file = Paths.get(this.dirLocation + "\\"+
username + "\\files").resolve(fileName).normalize();
            Resource resource = new UrlResource(file.toUri());
            if (resource.exists() || resource.isReadable())
return resource;
        }
        catch (MalformedURLException e) {
e.printStackTrace();
        }
        return null;
    }

    @Override
    public String createDir(String username) {
        try {
            Path dir =
Files.createDirectory(Paths.get(this.dirLocation + "\\"+ username
));
Files.createDirectories(Paths.get(dir + "\\"  + "data"));
Files.createDirectories(Paths.get(dir + "\\"  + "files"));
        }catch (Exception e){
e.printStackTrace();
        }
        return null;
    }

    @Override
    public String saveUserDirList(String username, Fields data) {
        try {
ObjectMapperobjectMapper = new ObjectMapper();
            Path dir = Paths.get(this.dirLocation + "\\" +
username);
            Path file = Paths.get(dir + "\\" + "data" + "\\" +
data.tempTitle + ".json").normalize();
            if(!Files.exists(file)) Files.createFile(file);
objectMapper.writeValue(new File(file.toString()), data);
        }catch (Exception e){
e.printStackTrace();
        }
        return null;
    }
```

```
    @Override
    public Fields loadUserDirList1(String username, String title)
{
        try {
            Path dir = Paths.get(this.dirLocation + "\\" +
username);
            Path file = Paths.get(dir + "\\" + "data" + "\\" +
title + ".json").normalize();
            if(!Files.exists(file)) return null;
ObjectMapperobjectMapper = new ObjectMapper();
            Fields result = objectMapper.readValue(new
File(file.toString()), Fields.class);
            return result;
        } catch (IOException e) {
e.printStackTrace();
        }
        return null;
    }
}
```

Файл «*FileController.java*»

```
@RestController
@RequestMapping("/api")
public class FileController {

    @Autowired
iFileSystemStoragefileSystemStorage;

    @PostMapping("/upload/{username}")
    public ResponseEntity<FileResponse>uploadSingleFileToDir
(@PathVariable String username, @RequestParam("file")
MultipartFile file) {
        String upfile = fileSystemStorage.saveFile(file,
username);
        String fileDownloadUri =
ServletUriComponentsBuilder.fromCurrentContextPath().path("/api/d
ownload/").path(upfile).toUriString();
        return ResponseEntity.status(HttpStatus.OK).body(new
FileResponse(upfile,fileDownloadUri,"File uploaded with
success!"));
    }

    public
ResponseEntity<Resource>downloadFileWithBodyAndPathVar(@PathVaria
ble String username, @RequestBody String filename) {
        Resource resource =
fileSystemStorage.loadFile(filename.trim().replaceAll("\"",""),
username);
        return
ResponseEntity.ok().header(HttpHeaders.CONTENT_DISPOSITION,
```

```
"attachment; filename=\"" + resource.getFilename() +
"\"").body(resource);
    }

    @PostMapping("/save/{username}")
    public ResponseEntity<String>saveDirList (@PathVariable
String username, @RequestBody Fields data) {
fileSystemStorage.saveUserDirList(username, data);
        return ResponseEntity.status(HttpStatus.OK).body("Save");
    }

    @PostMapping("/load/{username}")
    public Fields loadDirList (@PathVariable String username,
@RequestBody String data) {
        return fileSystemStorage.loadUserDirList1(username,
data.trim().replaceAll("\"",""));
    }

    @DeleteMapping("/delete/{username}")
    public ResponseEntity<FileResponse>deleteFile (@PathVariable
String username, @RequestBody String data) {
        String upfile = fileSystemStorage.delete(data, username);
        String fileDownloadUri =
ServletUriComponentsBuilder.fromCurrentContextPath().path("/api/d
ownload/").path(upfile).toUriString();
        return ResponseEntity.status(HttpStatus.OK).body(new
FileResponse(upfile,fileDownloadUri,"File delete with
success!"));
    }
}
```

ПриложениеГ

(обязательное)

Программноеобеспечениедляоблачногохраненияданных наустройствахподуправлениемОС*Android*

Презентация

Программное обеспечение для облачного хранения данных на устройствах под управлением ОС Android

Выполнил:
студент группы 19ВВП2 Табрисов С. А.
Руководитель:
к.т.н., доцент кафедры ВТ ПГУ Юрова О. В.

Актуальность работы

- Рост популярности облачных хранилищ
- Удобство использования
- Рост объемов данных
- Синхронизация данных

Цель работы

Цель работы – разработать программное обеспечение, позволяющее пользователям с мобильного устройства загружать, синхронизировать и управлять своими данными на удаленном сервере.

Функционал

Со стороны клиента:

- Простой и удобный графический интерфейс
- Возможность авторизации и регистрации пользователя
- Загрузка выбранного файла из локального хранилища на сервер
- Загрузка файлов с сервера
- Удаление файлов с сервера
- Создание папок
- Возможность просматривать список загруженных файлов

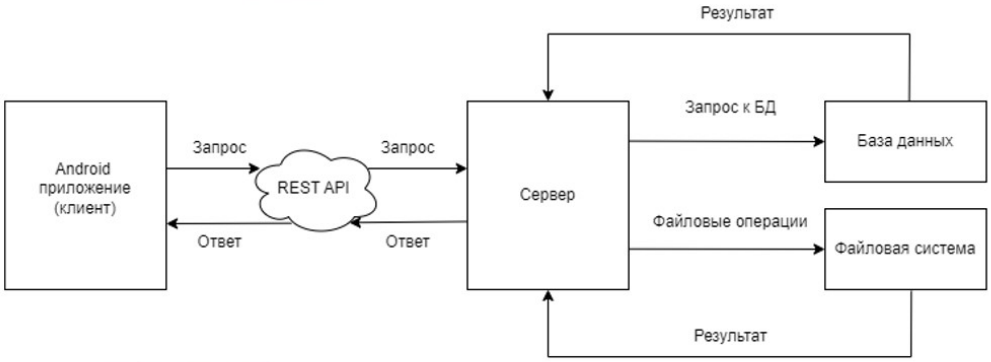
Со стороны сервера:

- Регистрация новых пользователей с валидацией данных
- Авторизация пользователя
- Создание директории клиента
- Добавление файла в хранилище
- Отправка файла пользователю
- Удаление файла

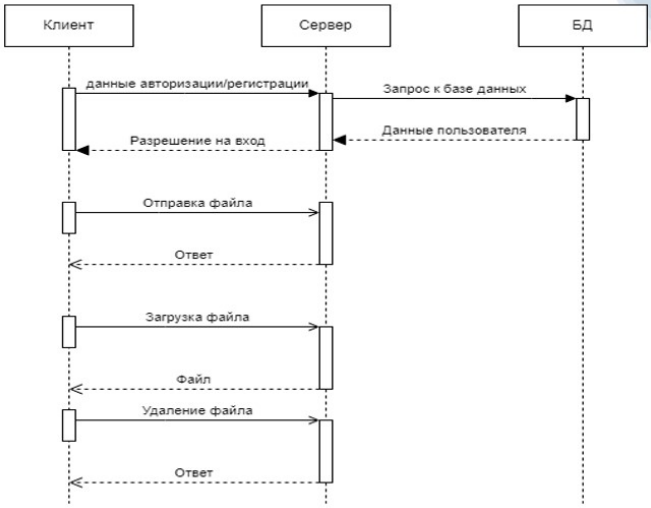
Средства разработки

- Среда разработки Android Studio
- Язык программирования Java
- Библиотека Retrofit
- Библиотека Jackson
- Фреймворк Spring Boot
- СУБД PostgreSQL

Структура проекта



UML ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТЕЙ



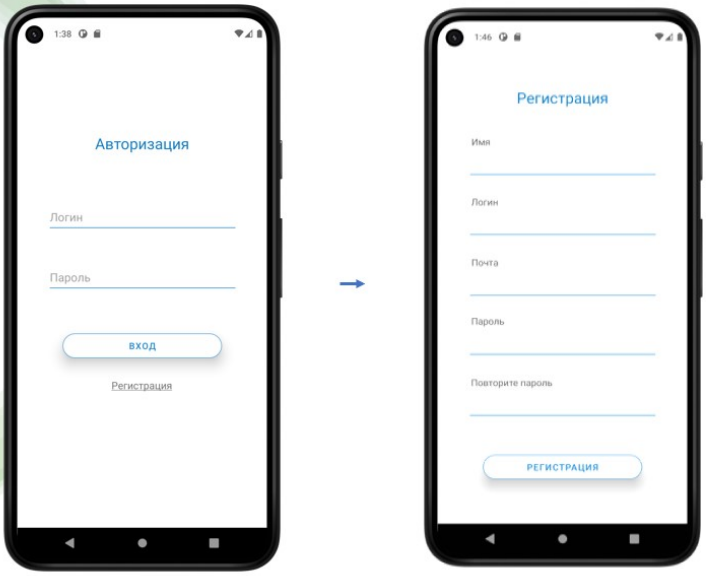
Реализация серверной части

Запуск сервера

```
D:\SpringServerJava\SpringServer\build\libs> Java -jar SpringServer.jar
2023-05-30T17:07:09.939+03:00  INFO 20196 --- [           main] c.e.s.SpringServerApplication            : Starting SpringServerApplication v0.0.1-SNAPSHOT using Java
 17.0.2 with PID 20196 (D:\SpringServerJava\SpringServer\build\libs\SpringServer.jar started by stepa in D:\SpringServerJava\SpringServer\build\libs)
2023-05-30T17:07:09.943+03:00  INFO 20196 --- [           main] c.e.s.SpringServerApplication            : No active profile set, falling back to 1 default profile: "
default"
2023-05-30T17:07:11.649+03:00  INFO 20196 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-05-30T17:07:11.649+03:00  INFO 20196 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-05-30T17:07:11.755+03:00  INFO 20196 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-05-30T17:07:11.963+03:00  INFO 20196 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2023-05-30T17:07:12.064+03:00  INFO 20196 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.1.7.Final
2023-05-30T17:07:12.482+03:00  INFO 20196 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-05-30T17:07:12.718+03:00  INFO 20196 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection org.postgresql.jdbc.PgConne
ction@3a7b503d
2023-05-30T17:07:12.751+03:00  INFO 20196 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-05-30T17:07:12.825+03:00  INFO 20196 --- [           main] SQL dialect                              : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLD
ialect
2023-05-30T17:07:13.804+03:00  INFO 20196 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate
.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-05-30T17:07:14.571+03:00  INFO 20196 --- [           main] c.e.s.SpringServerApplication            : Started SpringServerApplication in 5.135 seconds (process r
unning for 5.634)
```

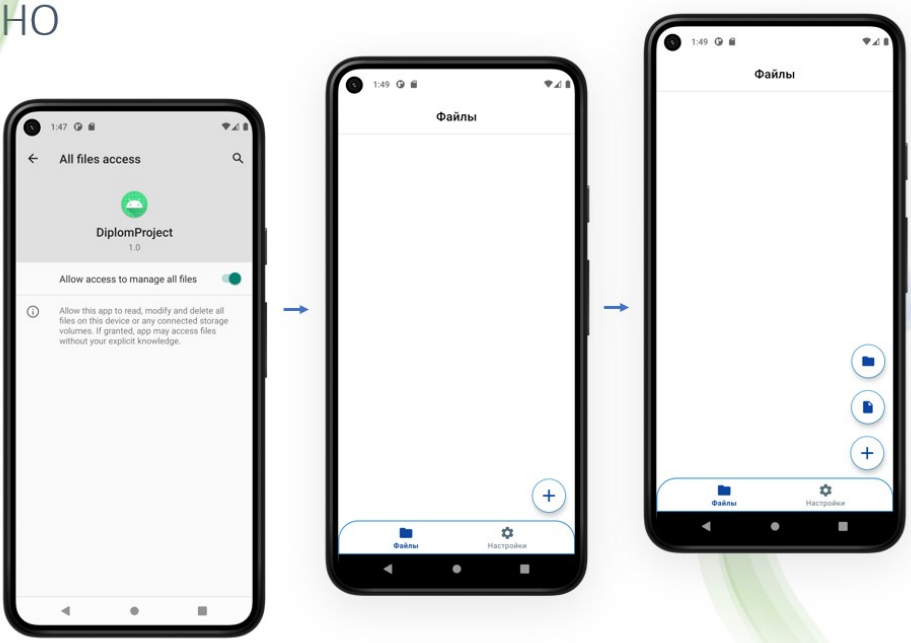
Реализация клиентской части

- Окно авторизации
- Окно регистрации

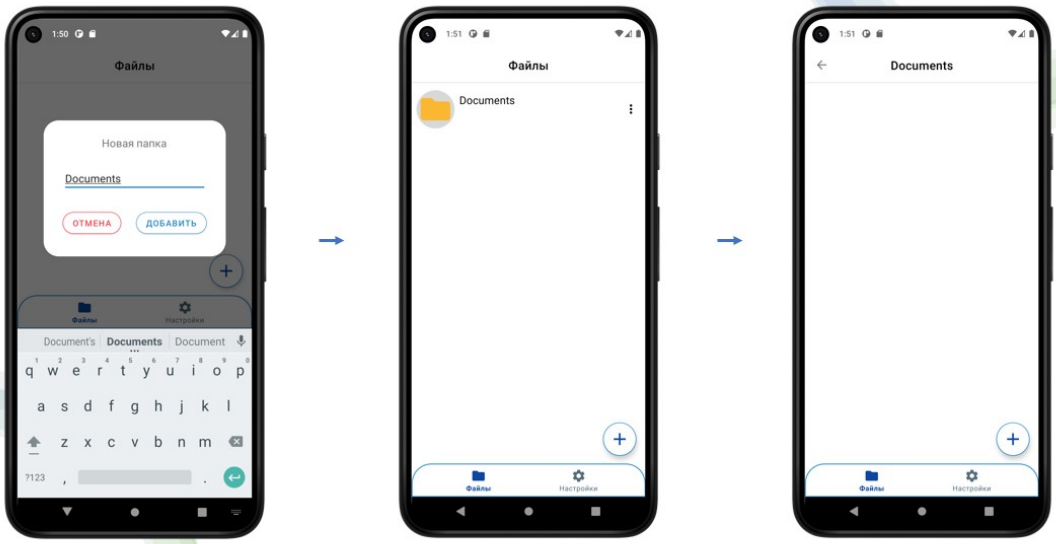


ОСНОВНОЕ ОКНО

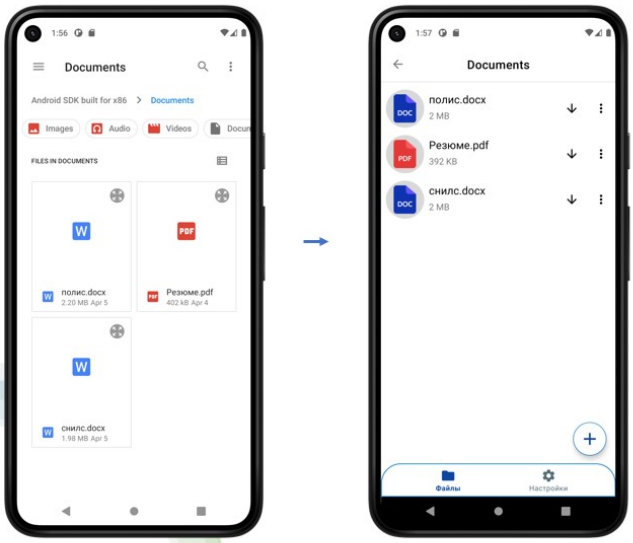
- Запрос разрешений
- Добавление папок
- Добавление файлов



ДОБАВЛЕНИЕ ПАПОК

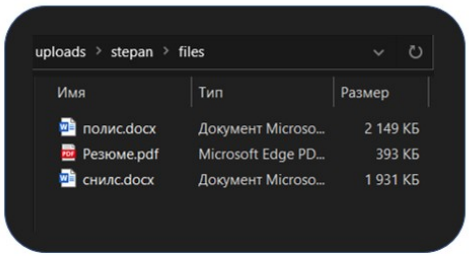


ОТПРАВКА ФАЙЛОВ



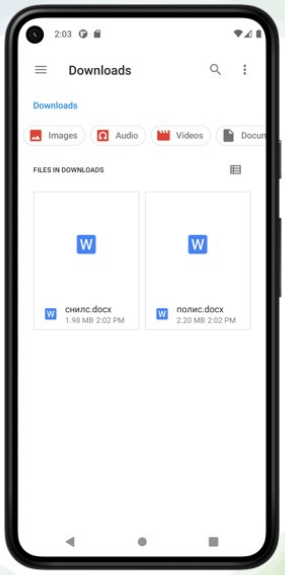
- Окно выбора файлов
- Отображение результата в приложении

Сохранение на сервере

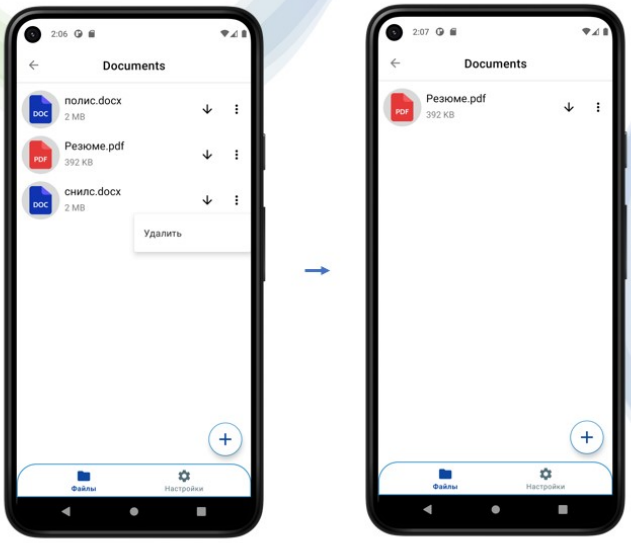


ЗАГРУЗКА ФАЙЛОВ С СЕРВЕРА

Скачанные файлы хранятся в папке «Downloads» на мобильном устройстве

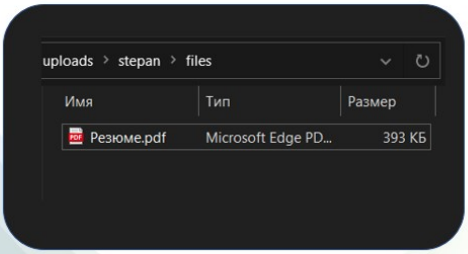


УДАЛЕНИЕ ФАЙЛОВ



- Удаление файлов
- Отображение результата на экране

Удаление файлов на сервере



ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы было разработано программное обеспечение для облачного хранения данных на устройствах под управлением ОС Android с использованием клиент-серверной архитектуры. Мобильное приложение позволяет пользователям удобно сохранять, синхронизировать и получать доступ к своим данным с различных устройств.

Спасибо за внимание!