

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Веб-приложение для синхронизированного
взаимодействия с ПДФ-документами в условиях
видеоконференции**

Студент

(подпись, дата)

Шандренко Артем Павлович

(ФИО полностью)

Руководитель

(подпись, дата)

Козлов А.Ю.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов «26»
декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Шандренко Артем Павлович гр. 22ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Веб-приложение для синхронизированного взаимодействия с ПДФ-документами в условиях видеоконференции

Тема утверждена приказом ПГУ № 1583/о от «30» декабря 2025г.

3. Руководитель работы д.т.н., профессор Козлов Андрей Юрьевич

4. Задание на работу (назначение разработки, исходные данные и т. п.) Разработать информационную систему для работы с PDF-документами в условиях видеоконференции, предназначенную для дистанционного обсуждения документов между пользователями. Система должна быть реализована в виде веб-приложения и обеспечивать создание комнаты, подключение участников, видеосвязь, загрузку PDF-документа, синхронизированный просмотр страниц и отображение действий ведущего у остальных участников.

В системе должны быть предусмотрены роли администратора, ведущего и участника. Администратор управляет учетными записями и ролями пользователей. Ведущий создает комнату, загружает PDF-документ и управляет его просмотром. Участник подключается к комнате, участвует в видеоконференции и просматривает синхронизированный документ.

Исходными данными для разработки являются тема выпускной квалификационной работы, результаты анализа предметной области, требования к функциональности системы, материалы по проектированию информационных систем и выбранный стек технологий. Для реализации системы используются Python, FastAPI, Uvicorn, Jinja2, HTML, CSS, JavaScript, WebRTC, WebSocket, PDF.js, Microsoft SQL Server и SQL Server Management Studio.

5. Перечень подлежащих разработке вопросов

- а) выполнить анализ предметной области, актуальности темы и существующих аналогов;
- б) сформулировать постановку задачи и требования к серверной и клиентской частям системы;
- в) выбрать и обосновать технологии разработки веб-приложения;
- г) спроектировать архитектуру системы, ролевую модель и базу данных;
- д) разработать структуру программного обеспечения и реализовать основные функции работы с пользователями, комнатами и PDF-документами;
- е) описать пользовательский интерфейс и основные сценарии работы с системой;
- ж) выполнить тестирование ключевых функций и оценку качества программной реализации;
- з) оформить пояснительную записку и приложения в соответствии с установленными требованиями.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова

Реферат

Пояснительная записка к выпускной квалификационной работе содержит 156 страниц, 58 рисунков, 18 таблиц, 24 источника и 3 приложения.

ИНФОРМАЦИОННАЯ СИСТЕМА, PDF-ДОКУМЕНТ, ВИДЕОКОНФЕРЕНЦИЯ, СИНХРОНИЗИРОВАННЫЙ ПРОСМОТР, WEBRTC, FASTAPI, UVICORN, WEBSOCKET, JINJA, HTML, CSS, JAVASCRIPT, PDF.JS, SQL SERVER, PYODBC.

Цель работы – разработать информационную систему для работы с PDF-документами в условиях видеоконференции. Система предназначена для дистанционного обсуждения документов между пользователями и обеспечивает создание комнаты, подключение участников, видеосвязь, загрузку PDF-документа и синхронизированный просмотр страниц.

В системе предусмотрены роли администратора, ведущего и зрителя. Администратор управляет пользователями, ведущий создает комнату и управляет просмотром документа, зритель подключается к комнате и просматривает документ в синхронизированном режиме.

Результатом работы является веб-приложение для работы с PDF-документами в условиях видеоконференции, разработанное в соответствии с требованиями выпускной квалификационной работы.

Для разработки использовались Python, FastAPI, Uvicorn, WebSocket, WebRTC, Jinja, HTML, CSS, JavaScript, PDF.js, SQL Server, SQL Server Management Studio и pyodbc.

					ВКР 09.03.02 – 08BP211.19 ПЗ			
Изм.		№ докум.	Подпись	Дата	Веб-приложение для синхронизированного взаимодействия с PDF-документами в условиях видеоконференции Пояснительная записка	Лит.	Лист	Листов
Разраб.		Шандренко А.П.					4	156
Провер.		Попова Н.А.				гр. 22ВИ1		
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и технического задания.....	9
1.1 Анализ предметной области	9
1.1.1 Анализ актуальности темы	10
1.1.2 Анализ аналогов.....	11
1.2 Анализ технического задания.....	17
1.2.1 Постановка задачи	17
1.2.2 Анализ требований к серверной и клиентской части на разработку ...	18
2 Проектирование информационной системы	21
2.1 Выбор технологий и инструментов разработки	21
2.1.1 Выбор языка программирования серверной части	21
2.1.2 Выбор фреймворка серверной части и веб-сервера	22
2.1.3 Выбор технологий формирования интерфейса	22
2.1.4 Выбор технологий видеосвязи и работы с PDF.....	23
2.1.5 Выбор базы данных и средств подключения.....	23
2.2 Проектирование ролевой модели и вариантов использования	24
2.3 Проектирование архитектуры веб-приложения	31
2.4 Проектирование программных классов и компонентов	36
2.5 Проектирование базы данных	40
2.5.1 Концептуальная модель данных	41
2.5.2 Логическая модель базы данных.....	48
2.5.3 Физическая модель базы данных	52
2.6 Моделирование поведения системы	55
2.7 Проектирование пользовательского интерфейса	60
2.8 Планирование разработки и оценка бюджета.....	63
3 Разработка информационной системы	67
3.1 Создание базы данных.....	67
3.2 Разработка структуры программного проекта.....	71
3.3 Реализация подключения к базе данных	73

3.4 Реализация ролевой модели и авторизации пользователей.....	74
3.5 Реализация работы с комнатами видеоконференции.....	76
3.6 Реализация загрузки PDF-документов и синхронизации действий.....	77
3.7 Реализация клиентской части веб-приложения.....	81
4 Тестирование и оценка качества информационной системы.....	89
4.1 Цель, объект и условия тестирования.....	89
4.2 Теоретические основы контроля качества	90
4.3 Тестирование методом черного ящика	92
4.4 Модульное тестирование	94
4.5 Анализ качества кода по метрикам	101
Заключение	106
Список используемых источников.....	109
Приложение А (обязательное) Листинг кода приложения.....	112
Приложение Б (рекомендуемое) Схемы и диаграммы информационной системы.....	140
Приложение В (рекомендуемое) Результаты тестирования.....	153

Введение

Современные образовательные и деловые процессы все чаще проводятся дистанционно или в смешанном формате. Участникам требуется не только голосовая или видеосвязь, но и совместная работа над обсуждаемыми документами. Особенно часто в таких сценариях применяется формат PDF, так как он сохраняет внешний вид документа и предназначен для обмена электронными документами независимо от среды их создания, просмотра или печати [1].

Стандартная демонстрация экрана во время видеоконференции лишь частично решает задачу презентации документа. Участники видят экран докладчика, но не имеют прямого доступа к состоянию документа: текущей странице, уровню масштабирования, выбранным областям и событиям, связанным с файлом. В результате обсуждение может замедляться из-за постоянных уточнений о том, какую страницу или раздел следует просмотреть.

При работе с PDF-документами важным является не только отображение содержимого файла, но и сохранение единого контекста обсуждения для всех участников встречи. Если каждый пользователь открывает документ самостоятельно, возникает риск рассинхронизации: один участник может находиться на другой странице, просматривать иной фрагмент или не видеть действия докладчика. Поэтому для эффективного дистанционного обсуждения требуется механизм, который связывает видеоконференцию с состоянием просматриваемого документа.

Актуальность данной работы обусловлена необходимостью разработки веб-приложения, в котором PDF-документ будет отображаться в видеоконференции как отдельный объект. В такой системе докладчик загружает файл в комнату, использует инструменты работы с текущей страницей, а другие участники получают синхронизированное состояние документа без необходимости вручную искать нужное местоположение.

Отдельное значение имеет то, что формат PDF является стандартизированным и широко применяемым способом представления

электронных документов. Это позволяет использовать его в учебных, организационных и деловых процессах, где требуется сохранить структуру, оформление и читаемость материала при передаче между пользователями [1]. Следовательно, создание средств для синхронизированной работы с PDF-документами во время видеоконференции представляет собой практически значимую задачу.

Цель данной выпускной квалификационной работы заключается в разработке информационной системы, предназначенной для работы с PDF-документами в условиях видеоконференции. Для достижения этой цели предусматривается анализ предметной области, формирование требований, проектирование архитектуры приложения и базы данных, разработка основных программных модулей, а также тестирование функций, связанных с обработкой PDF-файлов.

Исследование направлено на изучение процесса дистанционного обсуждения электронных документов. Предметом исследования выступает программная реализация веб-приложения, которое объединяет видеоконференцию, комнату для участников, загрузку PDF-файлов и синхронизацию страниц документа.

Практическая значимость работы заключается в том, что разработанное решение может быть использовано в консультациях преподавателей и студентов, при проверке выпускных и курсовых работ, при заключении контрактов, обсуждении технических спецификаций и проведении других встреч, где важен единый контекст для работы с PDF-документами.

1 Анализ предметной области и технического задания

Перед началом разработки программного продукта необходимо выполнить анализ предметной области и технического задания. Данный этап позволяет определить, какие задачи должна решать система, какие пользователи будут с ней взаимодействовать и какие ограничения необходимо учитывать при проектировании. Для веб-приложения, предназначенного для синхронизированного взаимодействия с PDF-документами в условиях видеоконференции, особенно важно рассмотреть не только сам процесс видеосвязи, но и работу с электронным документом как с основным объектом совместного обсуждения.

Разрабатываемая система должна объединять функции видеоконференции, просмотра PDF-файлов и синхронизации действий участников. Такой подход требуется в случаях, когда участникам важно не только поддерживать аудио- и видеосвязь, но и одновременно работать с одним документом. Это может быть отчет, договор, методические материалы, выпускная квалификационная работа, техническое задание, проектная документация или презентационные материалы, представленные в формате PDF.

1.1 Анализ предметной области

Анализ предметной области связан с изучением процессов, которые существуют до внедрения разрабатываемой системы, а также с определением основных проблем пользователей. В рамках данной работы предметной областью является дистанционное обсуждение PDF-документов во время видеоконференции. Такой процесс часто встречается как в образовательной среде, так и в деловой сфере.

В учебной деятельности PDF-документы применяются при проведении консультаций, защите отчетов, обсуждении курсовых и дипломных работ, проверке методических материалов и разборе заданий. В деловой среде данный формат используется при согласовании договоров, рассмотрении коммерческих предложений, анализе технических заданий, обсуждении проектной

документации и подготовке отчетности. Поэтому задача удобного совместного просмотра и обсуждения PDF-файлов является практически значимой.

1.1.1 Анализ актуальности темы

Формат PDF в настоящее время является одним из наиболее распространенных способов представления электронных документов. Согласно описанию стандарта ISO 32000-2, формат PDF предназначен для представления электронных документов таким образом, чтобы пользователи могли обмениваться ими и просматривать их независимо от среды создания, просмотра или печати [1]. Именно эта независимость от программной и аппаратной платформы делает PDF удобным форматом для передачи документов между разными организациями, учебными заведениями и пользователями.

В отличие от редактируемых офисных файлов, PDF чаще используется как итоговая форма документа. В таком виде отправляются отчеты, учебные материалы, положения, приказы, договоры, счета, акты, инструкции, презентации и техническая документация. Пользователь, открывающий PDF-файл, как правило, ожидает увидеть документ в том же виде, в котором его подготовил автор. Это особенно важно при обсуждении документов, где значение имеют нумерация страниц, расположение фрагментов, таблиц, рисунков и отдельных абзацев.

При очном взаимодействии участники обсуждения могут одновременно смотреть на распечатанный документ или на экран одного компьютера. В условиях дистанционного общения такой способ становится неудобным. Один пользователь демонстрирует экран, а остальные вынуждены ориентироваться только на изображение трансляции. Если участник открыл документ у себя, у него может быть другая страница, другой масштаб или другая область просмотра. В результате часть времени тратится не на обсуждение содержания, а на уточнение того, какой фрагмент документа рассматривается.

Проблема особенно заметна при проведении консультаций преподавателя со студентами. Например, студент может открыть отчет или пояснительную

записку, а преподаватель во время видеовстречи должен быстро указать на конкретную страницу, рисунок, таблицу или абзац. Если демонстрируется только экран, студент не всегда может удобно рассмотреть документ, а преподаватель не получает специализированных инструментов для управления совместным просмотром. При большом количестве участников ситуация усложняется еще сильнее.

Аналогичная задача возникает в бизнес-процессах. При обсуждении договора, коммерческого предложения или технического задания важно, чтобы все участники находились в одном контексте и видели один и тот же фрагмент документа. Обычная демонстрация экрана решает эту задачу лишь частично, поскольку не создает самостоятельного состояния документа внутри встречи. Участники фактически наблюдают за экраном ведущего, но не работают с PDF-файлом как с отдельным объектом системы.

Следовательно, актуальной является разработка веб-приложения, в котором видеоконференция будет дополнена синхронизированным просмотром PDF-документа. В такой системе ведущий сможет загрузить файл, управлять текущей страницей и показывать выделения, а остальные участники будут видеть эти действия в режиме реального времени. Это позволит снизить количество устных уточнений, ускорить обсуждение и сделать дистанционную работу с документами более наглядной.

1.1.2 Анализ аналогов

Для определения достоинств и ограничений существующих решений были рассмотрены сервисы, которые частично решают задачу дистанционного взаимодействия пользователей с документами. В качестве аналогов выбраны решения, относящиеся к двум группам: сервисы видеоконференций с демонстрацией экрана и инструменты совместной работы с PDF-файлами.

Первым аналогом является Zoom. Данный сервис предназначен для проведения видеоконференций и поддерживает демонстрацию содержимого экрана. В справочных материалах Zoom указано, что во время конференции

пользователь может показывать весь экран, выбранные приложения, часть экрана, доску, локально сохраненное видео и другие виды содержимого [2].

Преимуществом Zoom является развитая функциональность видеосвязи, наличие ролей организатора и участников, а также возможность быстро показать PDF-файл через демонстрацию окна программы просмотра. Однако работа с документом в этом случае строится вокруг трансляции экрана. PDF-файл не становится самостоятельным объектом комнаты, а участники не получают синхронизированную модель текущей страницы, выделений и состояния документа. Поэтому Zoom подходит для показа документа, но не полностью решает задачу управляемого совместного взаимодействия с PDF-файлом.

Интерфейс демонстрации PDF-документа в Zoom представлен на рисунке 1.



Рисунок 1 – Интерфейс демонстрации PDF-документа в Zoom

Вторым аналогом является Microsoft Teams. Сервис используется для онлайн-встреч, групповой коммуникации и совместной работы. В справке

Microsoft указано, что на собраниях Teams можно предоставлять общий доступ к экрану, рабочему столу, окну, файлу PowerPoint и другим материалам [3]. Кроме того, Teams поддерживает совместную работу с рядом встроенных инструментов Microsoft, что делает его удобным для корпоративной среды.

Сильной стороной Microsoft Teams является интеграция с экосистемой Microsoft 365, возможность проведения собраний, обмена файлами и работы в командах. Тем не менее основная модель обсуждения PDF в рамках видеовстречи также часто сводится к демонстрации экрана или открытию файла во внешнем просмотрщике. При этом специализированная синхронизация страницы PDF, передача выделений и управление документом в рамках отдельной комнаты не являются основной функцией сервиса.

Интерфейс демонстрации документа в Microsoft Teams представлен на рисунке 2.

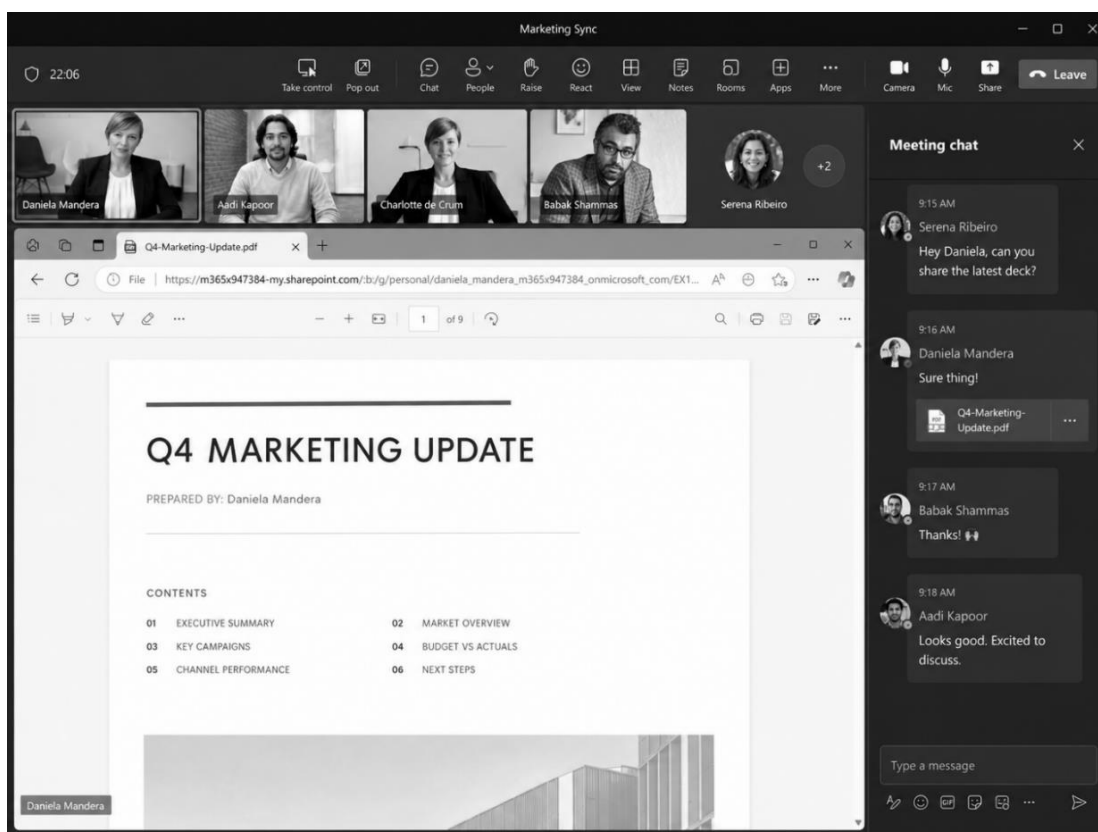


Рисунок 2 – Интерфейс демонстрации документа в Microsoft Teams

Третьим аналогом является Яндекс Телемост. Это сервис для проведения онлайн-встреч, который доступен пользователям через браузер и приложения. В

справке Яндексa указано, что участник встречи может показать другим пользователям свой экран, а для начала демонстрации необходимо выбрать соответствующую кнопку и указать, чем требуется поделиться [4].

Достоинствами Яндекс Телемоста являются простота запуска встречи, доступность через веб-браузер и удобство для быстрых созвонов. Сервис хорошо подходит для ситуации, когда необходимо оперативно показать собеседникам содержимое экрана. Однако в нем отсутствует специализированный режим совместного просмотра PDF-документа. Участники видят передаваемое изображение экрана, но не взаимодействуют с PDF-файлом как с отдельным синхронизируемым объектом.

Интерфейс демонстрации экрана в Яндекс Телемосте представлен на рисунке 3.

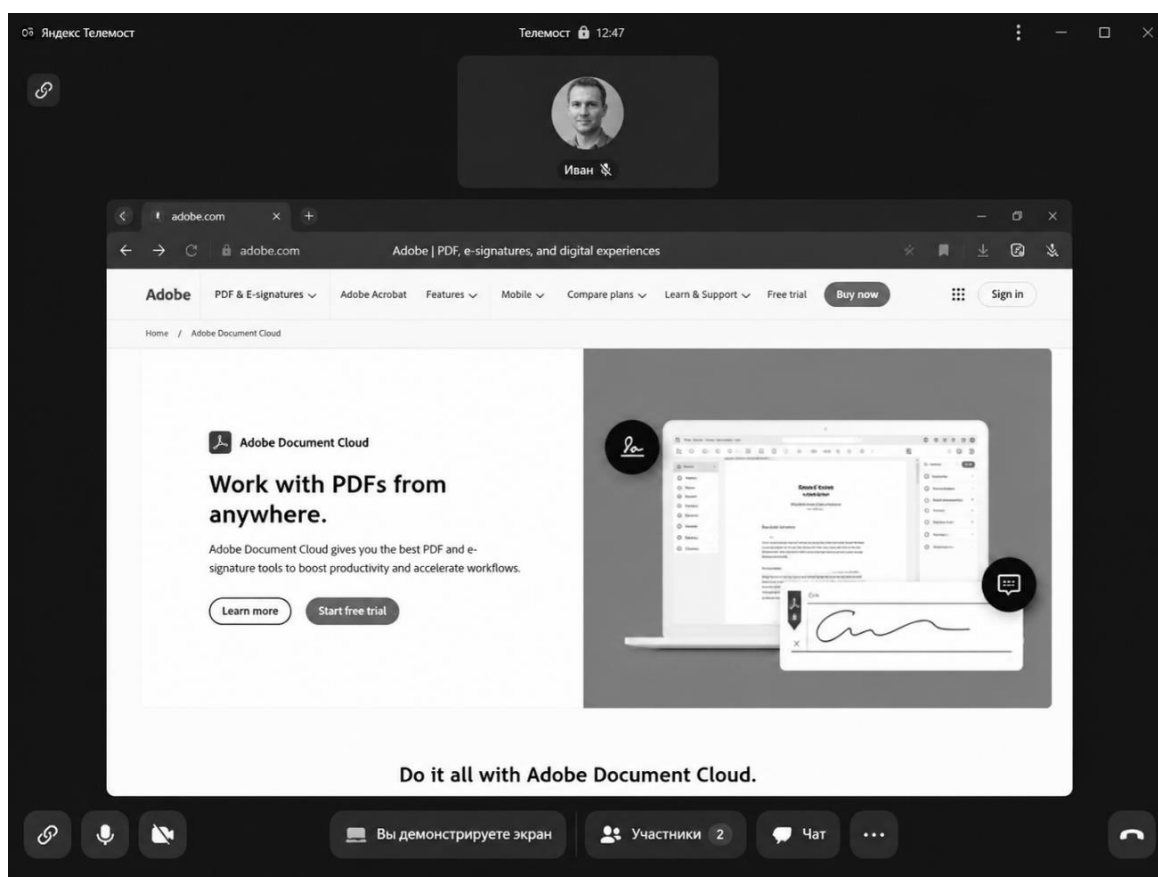


Рисунок 3 – Интерфейс демонстрации экрана в Яндекс Телемосте

Четвертым аналогом является Adobe Acrobat. В отличие от перечисленных выше решений, данный программный продукт ориентирован именно на работу с

PDF-документами. В справке Adobe указывается, что пользователи могут предоставлять доступ к PDF-файлу по ссылке, задавать разрешения и давать другим пользователям возможность просматривать и комментировать документ в браузере [5].

Преимуществом Adobe Acrobat является развитая работа с PDF: просмотр, комментирование, предоставление доступа, управление разрешениями и хранение документа в облаке. Однако Adobe Acrobat не является полноценной видеоконференц-системой, в которой обсуждение документа, видеосвязь и синхронизированное управление страницей объединены в одном простом сценарии. Пользователи могут совместно комментировать документ, но для живого обсуждения им часто требуется отдельный сервис связи.

Интерфейс общего доступа к PDF-документу в Adobe Acrobat представлен на рисунке 4.

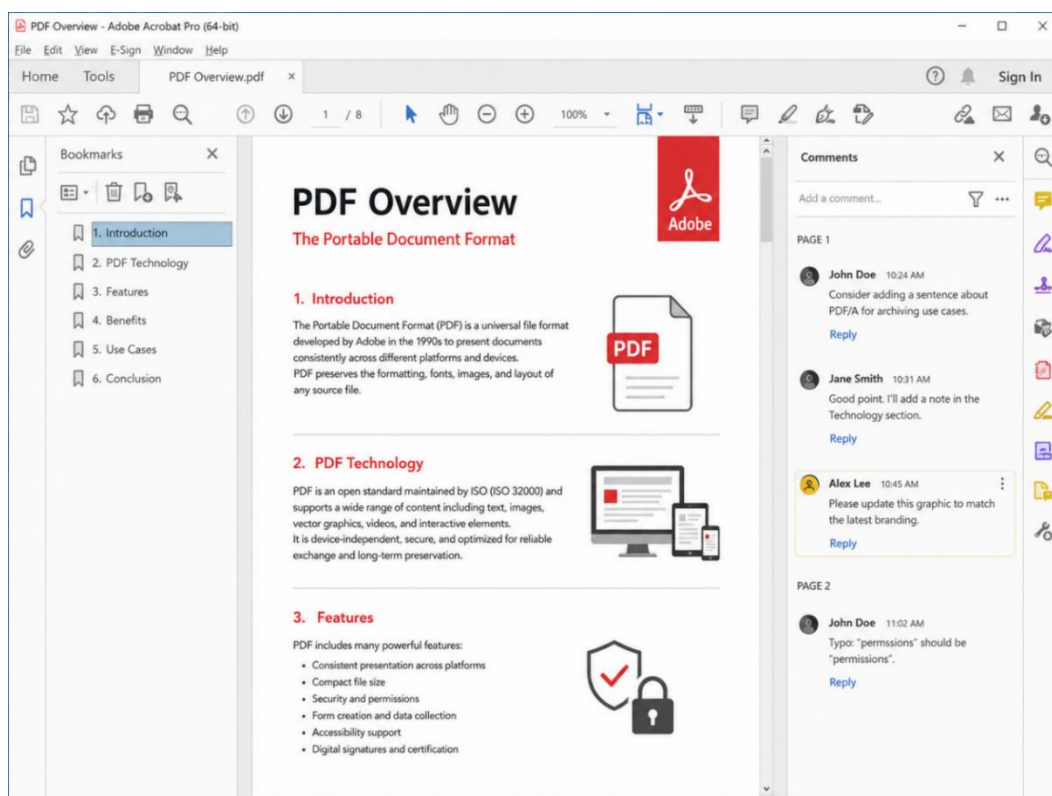


Рисунок 4 – Интерфейс общего доступа к PDF-документу в Adobe Acrobat

Результаты сравнения рассмотренных аналогов представлены в таблице 1. Для оценки используется обозначение «+», если возможность присутствует, и «-», если возможность отсутствует или не является основной функцией сервиса.

Таблица 1 – Сравнительная таблица аналогов разрабатываемой системы

Критерий	Zoom	Microsoft Teams	Яндекс Телемост	Adobe Acrobat	Разрабатываемая система
Проведение видеоконференции	+	+	+	–	+
Работа с PDF-документом	+	+	+	+	+
Загрузка PDF в среду совместной работы	–	+	–	+	+
Синхронизация страницы PDF-документа	–	–	–	–	+
Отображение действий ведущего на документе	+	+	–	+	+
Запрос участника на загрузку PDF-документа	–	–	–	–	+
Ролевая модель: администратор, ведущий, участник	+	+	–	–	+
Учебные и деловые сценарии работы	+	+	+	+	+

Примечание – знак «+» означает наличие возможности, знак «–» означает ее отсутствие в требуемом для данной работы виде, знак «±» означает частичную реализацию или зависимость от настроек сервиса.

Анализ показал, что существующие решения закрывают отдельные части рассматриваемой задачи, но не объединяют их в одном специализированном сценарии. Zoom, Microsoft Teams и Яндекс Телемост удобны для видеосвязи и демонстрации экрана, однако при работе с PDF-документом они в основном передают изображение экрана ведущего. Adobe Acrobat, наоборот, предоставляет развитые средства работы с PDF, но не является самостоятельной системой видеоконференций.

Разрабатываемое веб-приложение должно объединить преимущества указанных подходов: видеосвязь, комнату для участников, загрузку PDF-документа, синхронизированное управление страницей и отображение выделений. За счет этого система будет лучше соответствовать задачам консультаций, разборов документов и деловых обсуждений, где важна не только связь между участниками, но и единый контекст работы с документом.

1.2 Анализ технического задания

Анализ технического задания направлен на уточнение назначения разрабатываемого программного продукта и определение набора требований к его реализации. На данном этапе необходимо установить, какие пользователи будут работать с системой, какие функции им должны быть доступны и какие данные должны передаваться между клиентской и серверной частями.

Разрабатываемое приложение рассматривается как клиент-серверная веб-система. Пользователь взаимодействует с интерфейсом через браузер, а серверная часть отвечает за хранение сведений о пользователях, комнатах, документах и событиях синхронизации. Дополнительно система должна обеспечивать видеосвязь между участниками и передачу изменений состояния PDF-документа в режиме реального времени.

1.2.1 Постановка задачи

По итогам анализа предметной области можно определить основную задачу работы: требуется создать веб-приложение для организации видеоконференций, в котором будет поддерживаться синхронизированная работа участников с PDF-документами. Система должна позволять пользователям создавать комнаты, подключаться к ним, участвовать в видеосвязи, загружать PDF-файлы и синхронно просматривать документ под управлением ведущего.

Основными пользователями системы являются администратор, ведущий и участник. Администратор управляет учетными записями и ролями

ользователей. Ведущий создает комнату видеоконференции, загружает PDF-документ и управляет его просмотром. Участник подключается к комнате, участвует в видеоконференции и видит синхронизированные действия ведущего с документом.

Цель разработки заключается в создании удобной цифровой среды, в которой обсуждение документа происходит без потери контекста. Пользователи должны видеть один и тот же PDF-файл, одну и ту же страницу и выделенные ведущим области. Это позволит использовать приложение как для образовательных консультаций, так и для деловых встреч, связанных с согласованием и анализом документов.

1.2.2 Анализ требований к серверной и клиентской части на разработку

Для реализации поставленной задачи необходимо определить требования к серверной и клиентской частям веб-приложения. Требования должны учитывать ролевую модель, работу с PDF-документами, проведение видеоконференции и синхронизацию действий пользователей.

Основные требования к серверной части:

- осуществлять аутентификацию и авторизацию пользователей с учетом ролей администратора, ведущего и участника;
- хранить сведения о пользователях, включая логин, хеш пароля, роль, контактные данные и путь к аватарке;
- предоставлять администратору функции создания, редактирования, удаления и блокировки учетных записей;
- создавать комнаты видеоконференций и формировать код или ссылку для подключения участников;
- хранить информацию о комнатах, их создателях, текущем статусе и подключенных участниках;
- хранить сведения о загруженных документах, включая имя файла, путь хранения, владельца и связанную комнату;

- обрабатывать запросы участников на загрузку собственного PDF-документа и передавать их ведущему или администратору;
- передавать участникам события изменения текущей страницы PDF-документа;
- передавать участникам события выделения фрагмента или области PDF-документа;
- обеспечивать обмен служебными данными, необходимыми для установления видеосвязи между участниками;
- ограничивать выполнение управляющих действий только пользователями с соответствующими правами;
- фиксировать события комнаты, сообщения и действия пользователей для последующего отображения в интерфейсе;
- обрабатывать ошибки подключения, загрузки документа и синхронизации без завершения работы приложения.

Основные требования к клиентской части:

- предоставлять пользователю интуитивно понятный интерфейс входа в систему и подключения к комнате;
- отображать список участников видеоконференции и состояние подключения;
- предоставлять область видеосвязи для отображения локального и удаленных видеопотоков;
- отображать PDF-документ непосредственно в интерфейсе веб-приложения;
- обновлять текущую страницу документа у участников при действиях ведущего;
- отображать выделенные ведущим фрагменты или области документа;
- предоставлять ведущему элементы управления загрузкой PDF-файла, сменой страницы и выделением фрагментов;

- предоставлять участнику режим просмотра синхронизированного документа без возможности несанкционированного управления комнатой;
- предоставлять участнику возможность отправки запроса на загрузку PDF-документа;
- предоставлять личный кабинет пользователя с возможностью просмотра данных, изменения контактной информации и загрузки аватарки;
- обеспечивать адаптивное отображение интерфейса на компьютерах и мобильных устройствах;
- обновлять данные комнаты без полной перезагрузки страницы;
- выводить понятные сообщения об ошибках при отсутствии доступа, неверном коде комнаты или ошибке загрузки файла.

В результате выполнения указанных требований разрабатываемое веб-приложение должно обеспечить синхронную работу участников с PDF-документом в рамках видеоконференции. При этом система должна оставаться достаточно простой для использования в учебных консультациях и достаточно универсальной для деловых обсуждений документов.

2 Проектирование информационной системы

Второй раздел пояснительной записки посвящен проектированию информационной системы для работы с PDF-документами в условиях видеоконференции. На данном этапе определяются технологии разработки, ролевая модель, архитектура приложения, структура программных компонентов, база данных, основные сценарии взаимодействия и пользовательский интерфейс.

Проектирование выполняется с учетом требований, сформулированных в первом разделе. Для описания системы применяются графические модели: схемы архитектуры, диаграммы вариантов использования, диаграммы классов, компонентов, последовательности, деятельности, кооперации, состояний, а также модели базы данных [6]. Такой набор диаграмм позволяет рассмотреть систему с разных сторон: с точки зрения пользователя, программной реализации, хранения данных и выполнения ключевых процессов.

2.1 Выбор технологий и инструментов разработки

Выбор технологического стека определяет способ реализации серверной части, клиентского интерфейса, механизма видеосвязи, синхронизации состояния PDF-документа и хранения данных. При выборе технологий учитывались доступность инструментов, поддержка веб-разработки, возможность работы в режиме реального времени, простота запуска на локальном компьютере и соответствие требованиям разрабатываемой системы.

2.1.1 Выбор языка программирования серверной части

В качестве языка программирования серверной части выбран Python. Данный язык имеет развитую стандартную библиотеку, широкую экосистему пакетов и применяется для разработки веб-приложений, сервисов автоматизации и прикладных информационных систем. Официальная документация Python содержит описание языка, стандартных модулей и средств выполнения программ [7].

Для данной работы Python удобен тем, что позволяет быстро реализовать

серверную часть, маршруты веб-приложения в режиме реального времени. Кроме того, выбранный язык хорошо подходит для создания прототипов и дальнейшего расширения системы.

2.1.2 Выбор фреймворка серверной части и веб-сервера

Для реализации серверной части используется фреймворк FastAPI. Он предназначен для разработки веб-приложений и программных интерфейсов на Python, поддерживает современный подход к описанию маршрутов, автоматическую обработку запросов и работу с асинхронными функциями [8]. В разрабатываемой системе FastAPI применяется для обработки страниц веб-приложения, запросов авторизации, управления комнатами, загрузки PDF-документов и WebSocket-подключений.

Для запуска FastAPI-приложения используется Uvicorn. Это ASGI-веб-сервер для Python-приложений, который обеспечивает обработку HTTP-запросов и поддерживает асинхронный режим работы [9]. В составе разработанной системы Uvicorn запускает серверную часть локально и обеспечивает доступ пользователей к веб-интерфейсу через браузер.

Отдельным требованием является передача событий между участниками комнаты без полной перезагрузки страницы. Для этого используется механизм WebSocket, поддерживаемый FastAPI. В официальной документации FastAPI WebSocket рассматривается как средство двустороннего обмена сообщениями между клиентом и сервером [10]. В данной системе WebSocket применяется для синхронизации страницы PDF-документа, передачи событий выделения, обновления списка участников, отправки сообщений чата и сигналинга WebRTC.

2.1.3 Выбор технологий формирования интерфейса

Для формирования HTML-страниц на серверной стороне используется шаблонизатор Jinja. Документация Jinja описывает его как инструмент для создания текстовых шаблонов с подстановкой данных и управляющими конструкциями [11]. В разрабатываемом приложении шаблоны применяются для

страниц входа, панели пользователя, списка комнат, профиля, управления пользователями и комнаты видеоконференции.

Клиентская часть веб-приложения строится на стандартных веб-технологиях. HTML используется для описания структуры страниц и элементов интерфейса [12]. CSS применяется для оформления внешнего вида страниц, адаптации интерфейса и визуального разделения функциональных областей [13]. JavaScript используется для динамического поведения интерфейса, обработки действий пользователя, работы с WebSocket, управления PDF-просмотрщиком и взаимодействия с WebRTC [14].

2.1.4 Выбор технологий видеосвязи и работы с PDF

Для реализации видеосвязи между участниками применяется WebRTC. Данная технология предназначена для передачи аудио, видео и других данных между браузерами в режиме реального времени [15]. В проектируемой системе WebRTC используется для организации медиасоединения между участниками комнаты, а WebSocket применяется для обмена служебными сообщениями сигналинга, такими как offer, answer и ice-candidate.

Для отображения PDF-документов в браузере используется PDF.js. Этот инструмент позволяет просматривать PDF-файлы средствами веб-интерфейса без необходимости устанавливать отдельное приложение на стороне пользователя [16]. В разрабатываемой системе PDF.js отвечает за отображение страниц документа, а собственная клиентская логика приложения обеспечивает синхронизацию текущей страницы и отображение действий ведущего.

2.1.5 Выбор базы данных и средств подключения

В качестве системы управления базами данных выбрана Microsoft SQL Server. Движок базы данных Microsoft SQL предназначен для хранения, обработки и защиты данных в приложениях и информационных системах [17]. Для проектируемой системы SQL Server используется для хранения пользователей, ролей, комнат, участников, документов, аннотаций, сообщений,

запросов, уведомлений и журнала действий.

Для управления базой данных используется SQL Server Management Studio. Данное средство предоставляет графический интерфейс для работы с SQL Server, выполнения SQL-запросов, просмотра объектов базы данных и управления структурами хранения данных [18]. В рамках разработанной системы база данных создается заранее при помощи SQL-скрипта, после чего приложение подключается к уже подготовленной структуре.

Для взаимодействия Python-приложения с SQL Server используется ODBC-драйвер Microsoft. Официальная документация Microsoft содержит сведения о загрузке ODBC Driver for SQL Server и его применении для подключения приложений к базе данных [19]. На уровне Python используется библиотека pyodbc, представляющая собой мост между Python и ODBC [20]. Такая связка позволяет серверной части приложения выполнять SQL-запросы, получать данные и сохранять изменения в базе данных.

Таким образом, выбранный технологический стек обеспечивает реализацию всех ключевых требований системы: веб-интерфейс, серверную логику, видеосвязь, WebSocket-синхронизацию, просмотр PDF-документов и хранение данных в реляционной базе.

2.2 Проектирование ролевой модели и вариантов использования

После определения технологического стека и общей архитектуры информационной системы необходимо перейти к моделированию поведения пользователей. Для этого используется диаграмма вариантов использования, позволяющая отразить функциональные требования к системе с точки зрения внешних действующих лиц. Вариант использования описывает действие, которое пользователь может выполнить при работе с системой, а связи между вариантами использования позволяют показать составные действия и особенности выполнения сценариев [6].

В разрабатываемой информационной системе предусмотрены три основные роли пользователей: зритель, ведущий и администратор. Ведущий

расширяет возможности зрителя функциями создания и управления комнатой, обработки вопросов и запросов на загрузку PDF. Администратор получает дополнительные функции управления, комнатами системы и общими запросами. Роль пользователя определяет доступные разделы интерфейса и набор функций, которые могут быть выполнены после входа в систему. При моделировании вариантов использования важно учитывать, что часть функций является общей для нескольких ролей. Для исключения повторения одинаковых вариантов использования в нескольких диаграммах применяется наследование ролей.

Схема наследования ролей пользователей представлена на рисунке 5.

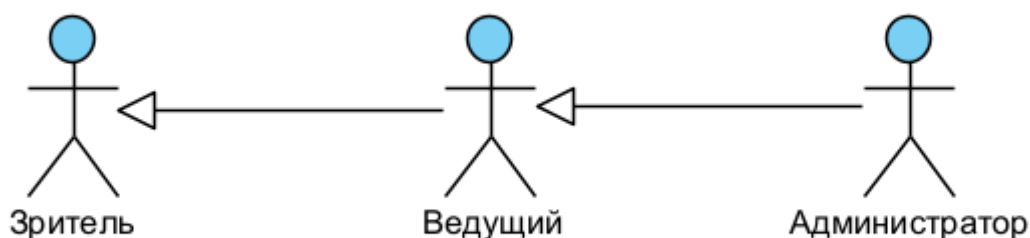


Рисунок 5 – Схема наследования пользователей по ролям

В рамках данной модели зритель рассматривается как базовая роль, включающая общие функции участника конференции. Ведущий расширяет возможности зрителя функциями создания и управления комнатой, обработки вопросов и запросов на загрузку PDF. Администратор получает дополнительные функции управления пользователями, комнатами системы и общими запросами. Такой подход позволяет не дублировать одинаковые функции на диаграммах и одновременно показать различия между ролями.

Первой рассматривается роль зрителя, она содержит основной набор пользовательских действий. Зритель может войти в систему, работать с профилем, подключиться к комнате по коду, участвовать в видеоконференции, просматривать PDF-документ, использовать чат, запрашивать загрузку PDF-документа и выйти из системы. Диаграмма вариантов использования зрителя представлена на рисунке 6.

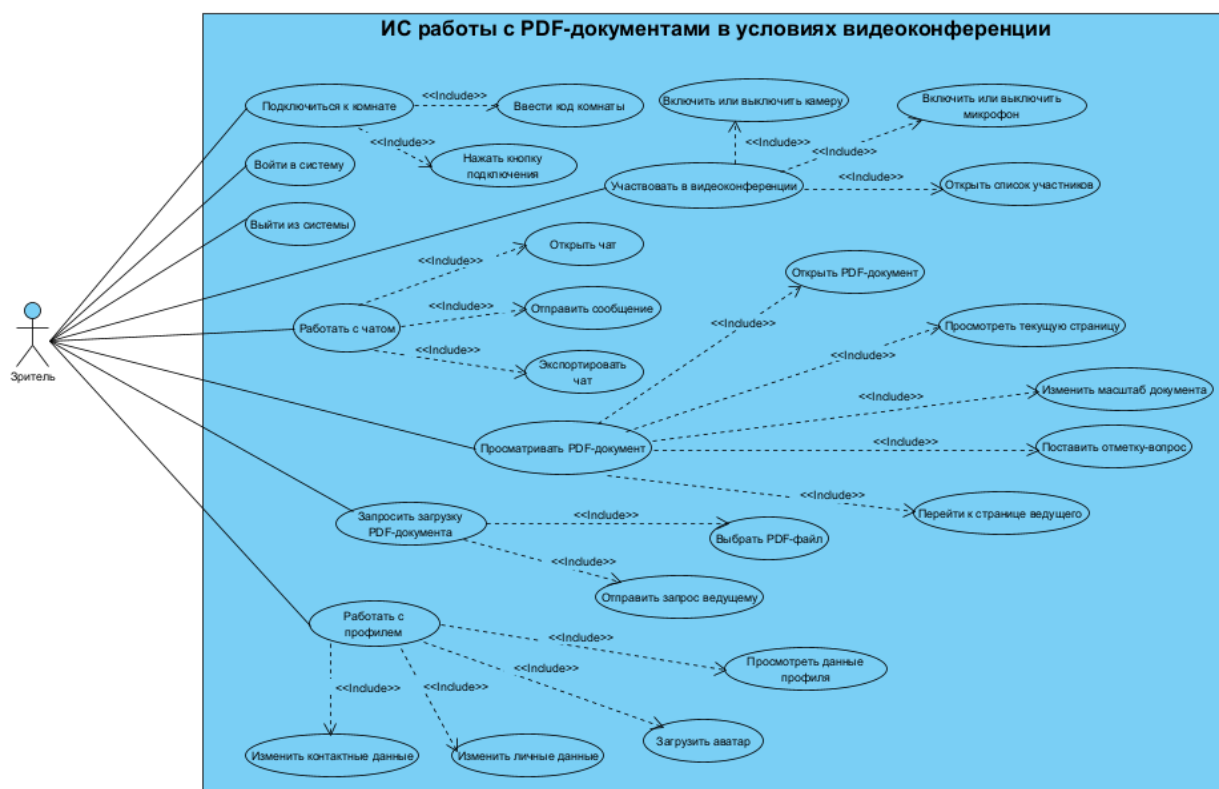


Рисунок 6 – Диаграмма вариантов использования зрителя

Вариант использования «Работать с профилем» включает просмотр данных профиля, изменение личных данных, изменение контактных данных и загрузку аватара. Эти действия связаны с разделом профиля пользователя и позволяют поддерживать актуальность персональной информации в системе.

Вариант использования «Подключиться к комнате» включает ввод кода комнаты и нажатие кнопки подключения. Данный сценарий используется в тех случаях, когда пользователь получает код от ведущего или другого участника и входит в уже созданную комнату. После успешного подключения пользователь становится участником комнаты и получает доступ к ее интерфейсу.

Вариант использования «Участвовать в видеоконференции» включает включение или выключение камеры, включение или выключение микрофона и открытие списка участников. Эти действия относятся к базовым функциям видеоконференции и обеспечивают взаимодействие пользователя с другими участниками комнаты.

Вариант использования «Просматривать PDF-документ» отражает

центральную часть системы, связанную с работой с документом. В него входят открытие PDF-документа, просмотр текущей страницы, изменение масштаба документа, постановка отметки-вопроса и переход к странице ведущего. Отметка-вопрос используется зрителем для обозначения места документа, которое требует внимания ведущего. Переход к странице ведущего необходим в случае, если зритель изменил просматриваемую страницу и затем должен вернуться к странице, которую демонстрирует ведущий.

Вариант использования «Работать с чатом» включает открытие чата, отправку сообщения и экспортирование чата. Чат используется как дополнительный канал текстового взаимодействия между участниками конференции. Экспорт чата позволяет сохранить сообщения для последующего просмотра.

Вариант использования «Запросить загрузку PDF-документа» включает выбор PDF-файла и отправку запроса ведущему. Эта функция нужна зрителю в том случае, если он должен предложить собственный документ для обсуждения. При этом зритель не может самостоятельно загрузить документ в комнату без разрешения, так как управление документом относится к функциям ведущего.

Разграничение полномочий зрителя и ведущего позволяет сохранить управляемый порядок совместной работы. Зритель может участвовать в обсуждении, просматривать документ и предлагать собственный PDF-файл, однако не изменяет общее состояние комнаты и демонстрируемого документа без разрешения ведущего. Это исключает случайное изменение материала, который рассматривают участники, и обеспечивает единый контекст видеоконференции.

Следующей рассматривается роль ведущего. Ведущий наследует общие пользовательские действия, поэтому на отдельной диаграмме отображаются только дополнительные функции, связанные с управлением комнатой, PDF-документом и действиями участников в процессе проведения видеоконференции.

Диаграмма вариантов использования ведущего представлена на рисунке 7.

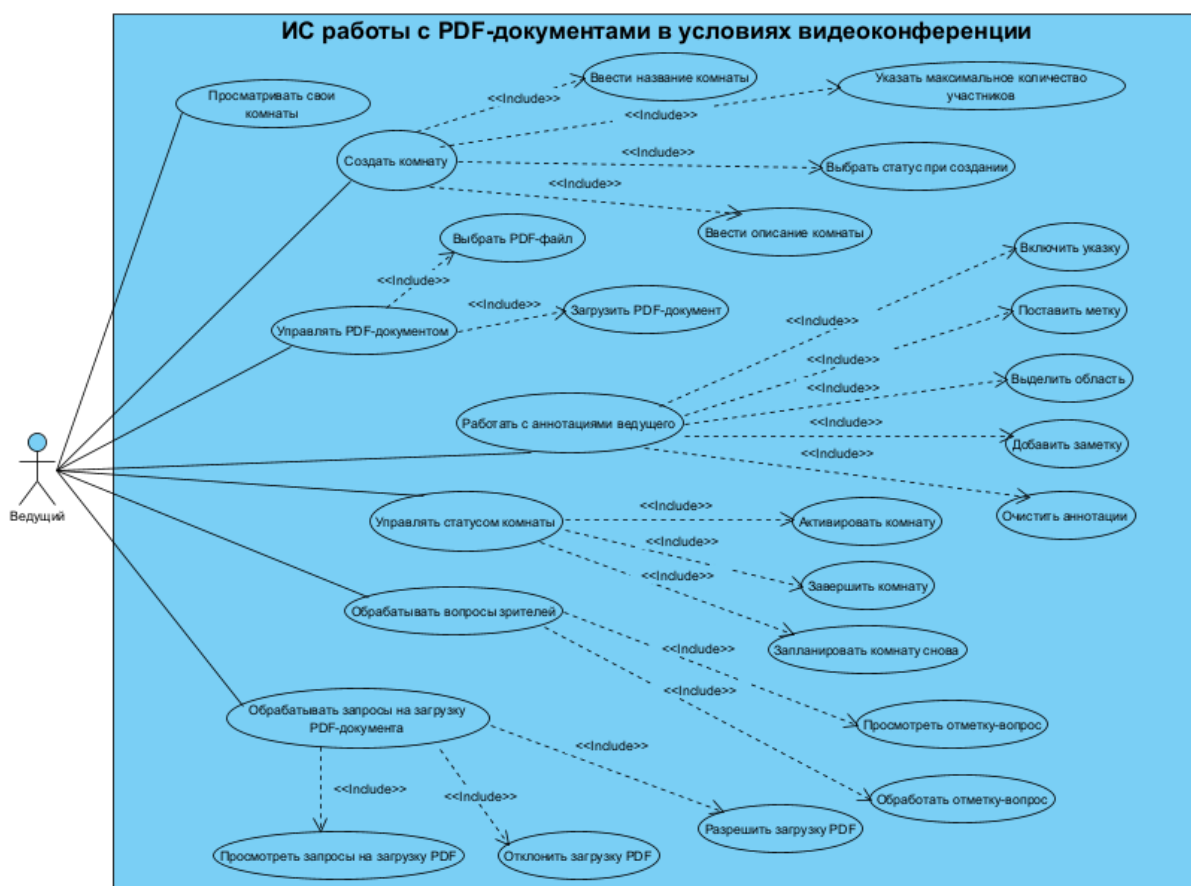


Рисунок 7 – Диаграмма вариантов использования ведущего

Вариант использования «Просматривать свои комнаты» предназначен для отображения списка комнат, созданных или доступных ведущему. В его состав входят поиск комнаты и открытие комнаты. Эта функция позволяет ведущему быстро перейти к нужной конференции и продолжить работу с ней.

Вариант использования «Создать комнату» включает ввод названия комнаты, указание максимального количества участников, выбор статуса при создании и ввод описания комнаты. В результате выполнения данного сценария в системе создается новая комната, которая может использоваться для проведения видеоконференции и совместной работы с PDF-документом.

Вариант использования «Управлять PDF-документом» включает выбор PDF-файла, загрузку PDF-документа и переключение страницы PDF-документа. Данный вариант использования является одним из ключевых для темы работы, так как ведущий не только открывает документ, но и управляет его отображением для участников. При переключении страницы текущее

состояние документа обновляется, после чего зрители могут перейти к странице ведущего.

Вариант использования «Работать с аннотациями ведущего» включает включение указки, постановку метки, выделение области, добавление заметки и очистку аннотаций. Эти инструменты позволяют ведущему акцентировать внимание участников на отдельных фрагментах PDF-документа. В отличие от зрителя, который может оставить отметку-вопрос, ведущий использует расширенный набор аннотационных инструментов для пояснения материала.

Вариант использования «Управлять статусом комнаты» включает активацию комнаты, завершение комнаты и повторное планирование комнаты. Статус комнаты определяет ее текущее состояние и возможность использования участниками. Например, активная комната доступна для подключения и проведения конференции, а завершенная комната фиксирует окончание работы.

Вариант использования «Обрабатывать вопросы зрителей» включает просмотр отметки-вопроса и обработку отметки-вопроса. После того как зритель оставляет вопрос на PDF-документе, ведущий может перейти к соответствующей отметке и отреагировать на нее в ходе обсуждения. Благодаря этому вопрос привязывается не только к сообщению в чате, но и к конкретному месту документа.

Вариант использования «Обрабатывать запросы на загрузку PDF-документа» включает просмотр запросов на загрузку PDF, разрешение загрузки PDF и отклонение загрузки PDF. Такая модель необходима для контроля содержимого комнаты. Зритель может предложить документ, но решение о загрузке принимает ведущий, что позволяет сохранить управление сценарием конференции.

Последней рассматривается роль администратора. Администратор выполняет функции управления системой и не дублирует базовые пользовательские сценарии, уже описанные для зрителя и ведущего. Диаграмма вариантов использования администратора представлена на рисунке 8.

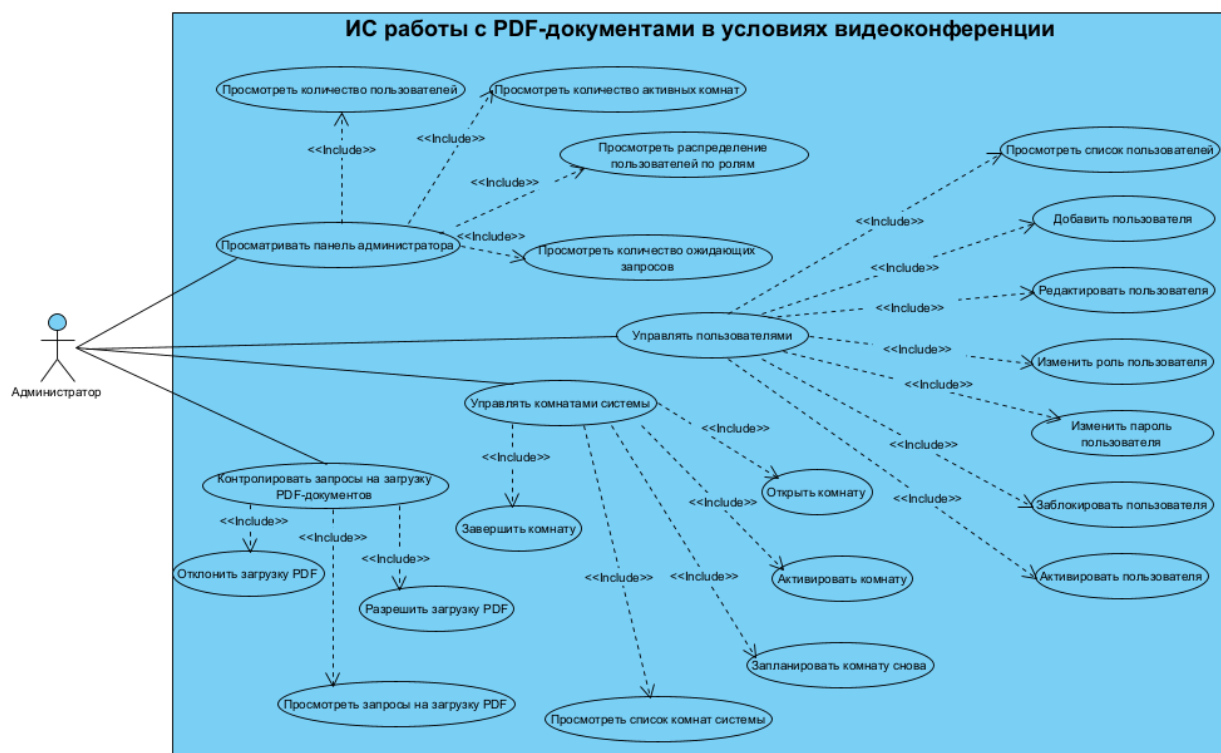


Рисунок 8 – Диаграмма вариантов использования администратора

Вариант использования «Просматривать панель администратора» включает просмотр количества пользователей, просмотр количества активных комнат, просмотр распределения пользователей по ролям и просмотр количества ожидающих запросов. Панель администратора предназначена для быстрого получения общей информации о состоянии системы.

Вариант использования «Управлять пользователями» включает просмотр списка пользователей, добавление пользователя, редактирование пользователя, изменение роли пользователя, изменение пароля пользователя, блокировку пользователя и активацию пользователя. Эта группа функций является основной для администратора, так как позволяет управлять учетными записями и правами доступа.

Вариант использования «Управлять комнатами системы» включает просмотр списка комнат системы, открытие комнаты, активацию комнаты, завершение комнаты и повторное планирование комнаты. В отличие от ведущего, который работает со своими комнатами, администратор управляет комнатами на уровне всей системы. Это необходимо для контроля активных

конференций и исправления возможных организационных ошибок.

Вариант использования «Контролировать запросы на загрузку PDF-документов» включает просмотр запросов на загрузку PDF, разрешение загрузки PDF и отклонение загрузки PDF. Данная функция пересекается с возможностями ведущего, но имеет административный характер, так как позволяет контролировать запросы в рамках системы.

Таким образом, разработанные диаграммы вариантов использования показывают функциональность системы с учетом трех ролей: зрителя, ведущего и администратора. Зритель описывает базовые действия участника конференции, ведущий расширяет их функциями управления комнатой и PDF-документом, а администратор получает средства контроля пользователей, комнат и запросов. Такое разделение соответствует фактической реализации интерфейса и позволяет связать проектную модель с дальнейшим описанием базы данных, компонентов системы и пользовательских сценариев.

2.3 Проектирование архитектуры веб-приложения

После выбора технологического стека и описания ролевой модели необходимо определить архитектуру разрабатываемой информационной системы. Приложение реализуется в виде веб-системы, в которой пользователь взаимодействует с интерфейсом через браузер, а основная обработка запросов, управление комнатами, работа с PDF-документами и хранение данных выполняются на серверной стороне.

Для разрабатываемой системы выбрана клиент-серверная архитектура. Такой подход позволяет централизованно обрабатывать данные, хранить сведения о пользователях, комнатах и документах, а также предоставлять доступ к системе без установки отдельного клиентского приложения. Пользователь получает доступ к функциональности через веб-браузер, а серверная часть обеспечивает обработку запросов, проверку прав доступа, работу с базой данных и управление загруженными PDF-документами.

Общая архитектура веб-приложения представлена на рисунке 9.



Рисунок 9 – Общая архитектура веб-приложения

На рисунке 9 показано взаимодействие основных частей системы: пользователя, клиентской части, серверной части, базы данных и файлового хранилища PDF-документов. Клиентская часть обеспечивает отображение интерфейса, просмотр PDF-документов, работу с WebSocket-соединением и участие пользователя в видеоконференции. HTTP/HTTPS используется для обычных запросов к серверу, включая авторизацию, переходы между страницами, создание и открытие комнат, а также загрузку PDF-документов.

WebSocket-соединение используется для передачи событий комнаты в реальном времени. К таким событиям относятся сообщения чата, синхронизация текущей страницы PDF-документа, изменение состояния комнаты и служебные данные сигналинга. Аудио- и видеопотоки между участниками передаются с использованием WebRTC. При этом серверная часть не хранит медиапотоки, а обеспечивает управление комнатой, авторизацию, обработку действий пользователей и передачу служебных событий.

База данных используется для хранения структурированной информации: учетных записей, ролей, комнат, участников, сообщений, сведений о PDF-

документах и запросах на загрузку файлов. В базе данных сохраняются метаданные файлов, включая имя, путь хранения, владельца, связанную комнату и дату загрузки. Такое разделение позволяет хранить бинарные файлы отдельно от структурированных данных.

Для внутренней организации приложения используется многоуровневая архитектура. Она позволяет разделить приложение на несколько логических уровней, каждый из которых отвечает за отдельную часть функциональности. Для представления архитектурных решений используются UML-диаграммы, так как они позволяют визуализировать структуру программной системы и зависимости между ее компонентами [6]. Необходимость отражения архитектуры и программных компонентов в проектной части выпускной квалификационной работы также соответствует методическим рекомендациям по разработке программного обеспечения [20].

UML-диаграмма компонентов многоуровневой архитектуры приложения представлена на рисунке 10.

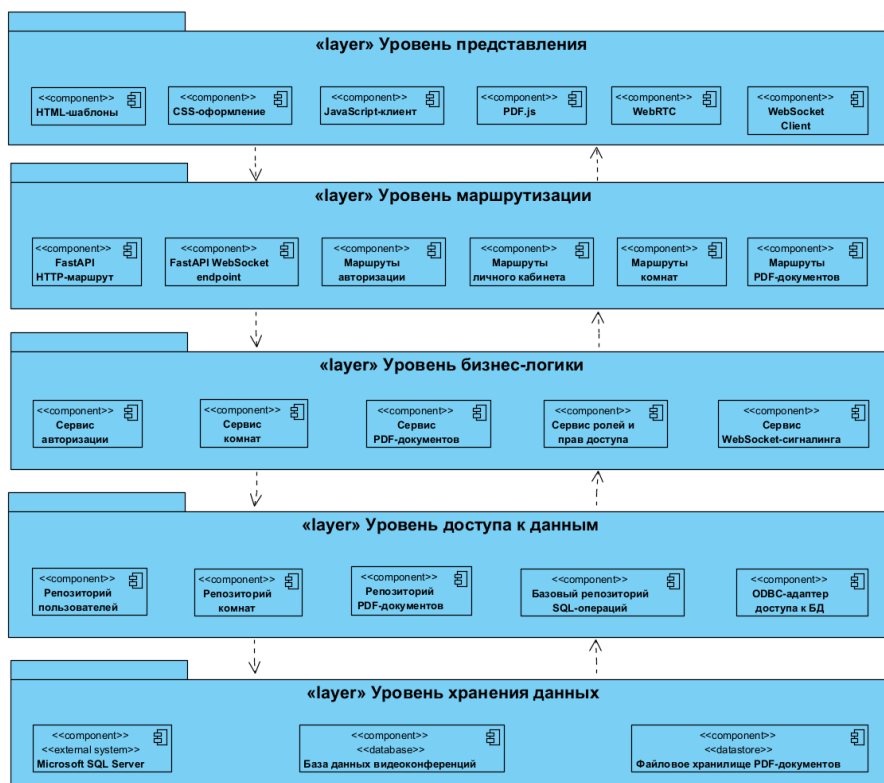


Рисунок 10 – UML-диаграмма компонентов многоуровневой архитектуры приложения

На рисунке 10 выделено пять уровней приложения: уровень представления, уровень маршрутизации, уровень бизнес-логики, уровень доступа к данным и уровень хранения данных.

Уровень представления отвечает за работу пользовательского интерфейса в браузере. На нем располагаются HTML-шаблоны, CSS-оформление, JavaScript-клиент, PDF.js, WebRTC и WebSocket Client. Через этот уровень пользователь выполняет вход в систему, он включает репозитории пользователей, работает с личным кабинетом, подключается к комнате, просматривает PDF-документ и участвует в видеоконференции.

Уровень маршрутизации принимает HTTP-запросы и WebSocket-подключения. Он связывает действия пользователя с соответствующими обработчиками серверной части. На данном уровне расположены HTTP-маршруты FastAPI, WebSocket endpoint, маршруты авторизации, личного кабинета, комнат и PDF-документов.

Уровень бизнес-логики содержит сервисы, реализующие основные правила работы приложения. К нему относятся сервис авторизации, сервис комнат, сервис PDF-документов, сервис ролей и прав доступа, а также сервис WebSocket-сигналинга. Выделение сервисного слоя позволяет не размещать прикладную логику непосредственно в маршрутах. Маршруты принимают запросы и передают их сервисам, а сервисы выполняют необходимые операции в соответствии с назначением пользователя и состоянием комнаты.

Уровень доступа к данным отвечает за выполнение операций с базой данных. Он включает репозитории пользователей, комнат и PDF-документов, базовый репозиторий SQL-операций и ODBC-адаптер доступа к базе данных. Репозитории изолируют SQL-запросы от сервисов и маршрутов, что снижает связанность компонентов приложения.

Уровень хранения данных включает Microsoft SQL Server, базу данных видеоконференций и файловое хранилище PDF-документов. В базе данных размещаются структурированные сведения, необходимые для работы системы, а файловое хранилище используется для сохранения загруженных документов.

Основное направление взаимодействия между уровнями проходит сверху вниз: клиентская часть обращается к уровню маршрутизации, маршруты вызывают сервисы бизнес-логики, сервисы используют репозитории, а репозитории обращаются к уровню хранения данных. Такое построение обеспечивает разделение ответственности между компонентами приложения и упрощает дальнейшее сопровождение системы.

После определения логической архитектуры необходимо показать, как она отражена в структуре программного проекта. Для этого используется диаграмма программных модулей серверной части. Она показывает физическую организацию каталогов проекта и зависимости между ними.

Структура программных модулей серверной части приложения представлена на рисунке 11.

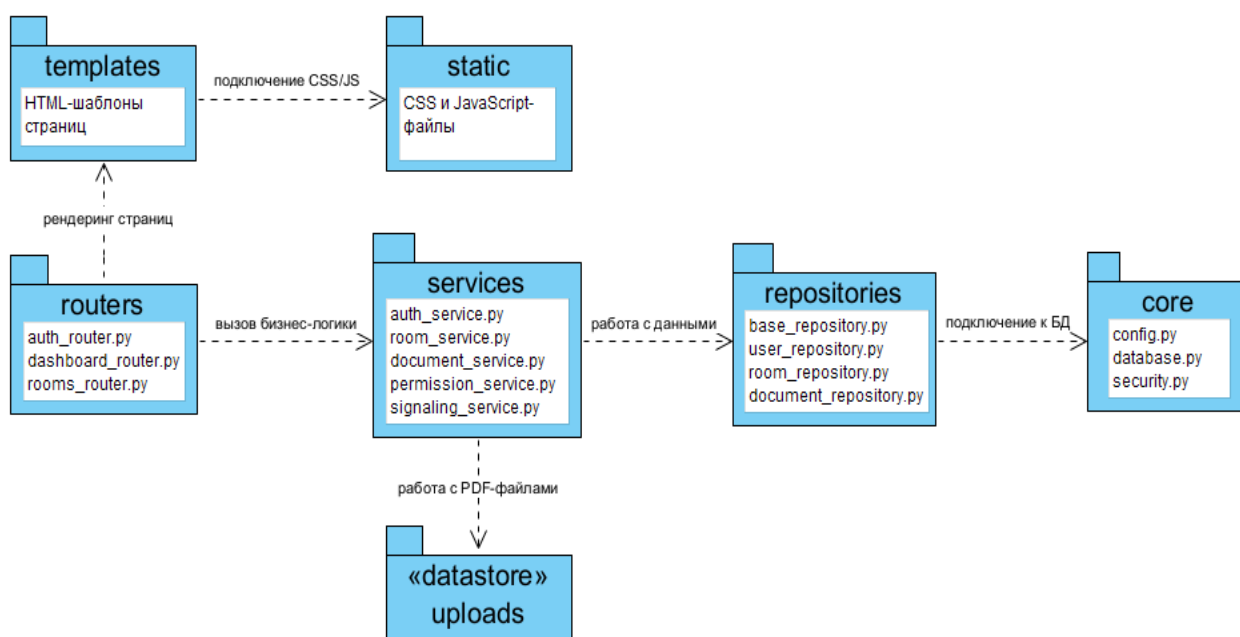


Рисунок 11 – Структура программных модулей серверной части приложения

На рисунке 11 показано разбиение серверной части на основные каталоги. Каталог **routers** содержит обработчики HTTP-запросов и WebSocket-подключений. Каталог **services** содержит сервисы бизнес-логики, отвечающие за авторизацию, работу с комнатами, обработку PDF-документов, проверку прав доступа и WebSocket-сигналинг. Каталог **repositories** содержит классы доступа к

данным. Каталог `core` включает конфигурацию, подключение к базе данных и механизмы безопасности.

Отдельно показаны каталоги `templates`, `static` и `uploads`. Каталог `templates` содержит HTML-шаблоны страниц, `static` содержит CSS- и JavaScript-файлы клиентской части, а `uploads` используется как файловое хранилище PDF-документов. Связи между каталогами отражают основное направление зависимостей: `routers`, `services`, `repositories`, `core`. Это показывает, что обработчики запросов не обращаются напрямую к базе данных, а используют сервисы и репозитории.

Связь `routers` и `templates` показывает формирование страниц интерфейса на основе HTML-шаблонов. Связь `templates` и `static` отражает подключение статических ресурсов к страницам. Связь `services` и `uploads` показывает, что работа с PDF-файлами выполняется через сервисный слой, а не напрямую из маршрутов.

Таким образом, разработанная архитектура разделяет пользовательский интерфейс, маршрутизацию, бизнес-логику, доступ к данным и хранение файлов. Такое построение соответствует назначению информационной системы, так как приложение должно одновременно обеспечивать работу веб-интерфейса, видеоконференции, синхронизированного просмотра PDF-документов, чата, ролевого доступа и хранения данных. Разделение на уровни и программные модули повышает читаемость проекта и упрощает его дальнейшее развитие.

2.4 Проектирование программных классов и компонентов

После определения общей архитектуры приложения необходимо уточнить состав основных программных классов серверной части. Для этого используется диаграмма классов UML. Диаграмма классов позволяет показать структуру программной системы, основные классы, их операции и отношения между ними [6]. Она детализирует внутреннюю организацию серверной части, показывая распределение ответственности между обработчиками запросов, сервисами и репозиториями.

Диаграмма классов серверной части приложения представлена на рисунке 12.

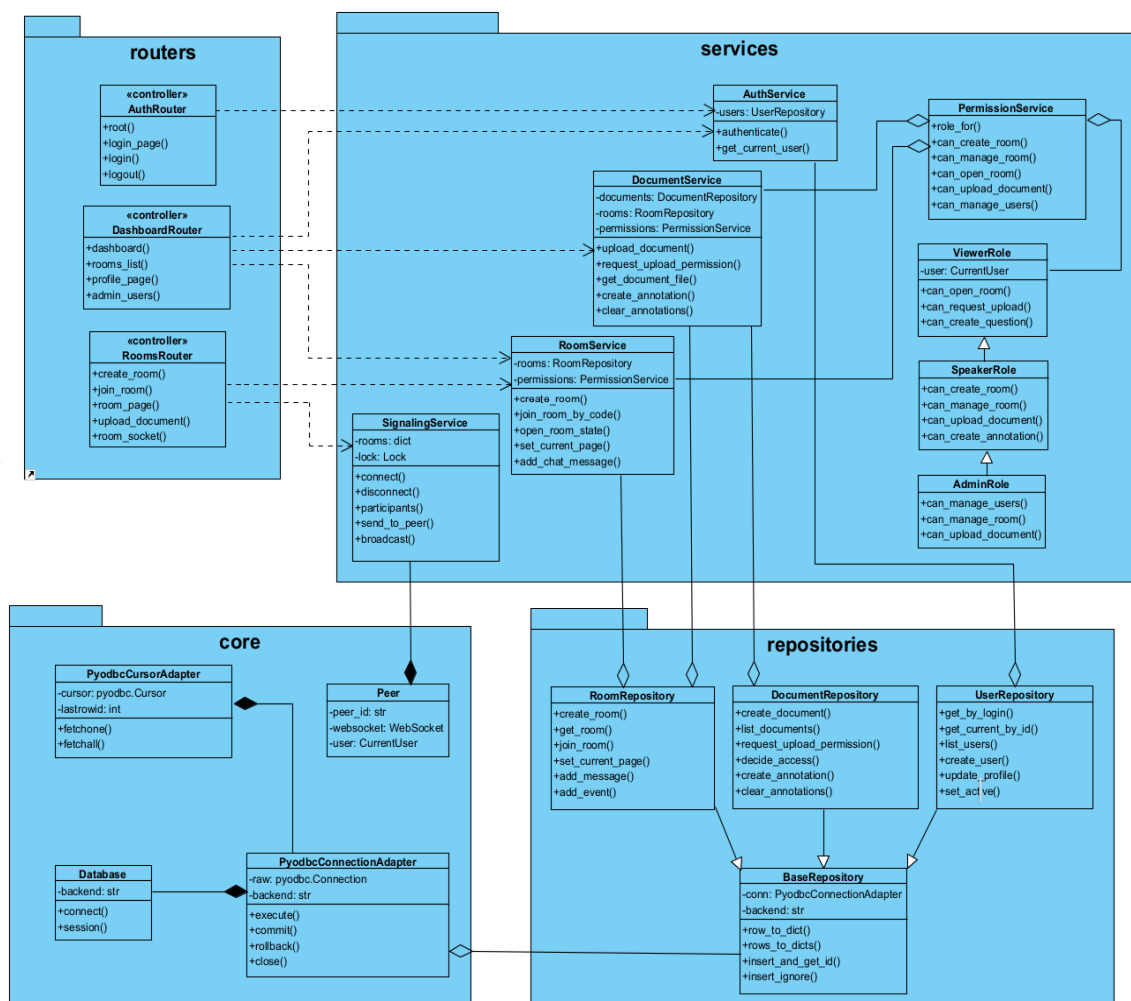


Рисунок 12 – Диаграмма классов серверной части приложения

На рисунке 12 серверная часть представлена через четыре группы классов: routers, services, repositories и core. Такое разбиение соответствует ранее выбранной многоуровневой архитектуре, но на данной диаграмме внимание уделяется не уровням в целом, а конкретным классам, реализующим обработку запросов, бизнес-логику, доступ к данным и инфраструктурные операции.

Группа routers содержит классы маршрутов AuthRouter, DashboardRouter и RoomsRouter. Данные классы принимают пользовательские запросы и передают их в соответствующие сервисы. AuthRouter связан с авторизацией пользователя, DashboardRouter – с личным кабинетом, профилем, списком комнат и административными страницами, а RoomsRouter – с созданием комнат,

подключением к ним, загрузкой PDF-документов и WebSocket-соединением комнаты.

Группа `services` содержит классы бизнес-логики. `AuthService` выполняет проверку учетных данных и получение текущего пользователя. `RoomService` реализует операции создания, открытия и управления комнатами. `DocumentService` отвечает за загрузку PDF-документов, обработку запросов на загрузку, получение текущего файла и работу с аннотациями. `PermissionService` выполняет проверку прав доступа с учетом роли пользователя. `SignalingService` управляет активными WebSocket-подключениями и рассылкой событий участникам комнаты.

Отдельно в сервисной части показаны классы ролевой модели: `ViewerRole`, `SpeakerRole` и `AdminRole`. Между ними используется наследование: `ViewerRole` является базовым классом, `SpeakerRole` расширяет его возможностями ведущего, а `AdminRole` добавляет административные действия. Такое решение позволяет отразить ролевую модель не только на диаграммах вариантов использования, но и на уровне программной структуры.

Группа `repositories` содержит классы доступа к данным. В основе находится `BaseRepository`, от которого наследуются `UserRepository`, `RoomRepository` и `DocumentRepository`. Базовый репозиторий содержит общие операции обработки результатов SQL-запросов и добавления записей, а специализированные репозитории инкапсулируют работу с конкретными сущностями предметной области: пользователями, комнатами и PDF-документами.

Группа `core` содержит инфраструктурные классы. `Database` отвечает за создание подключения к базе данных. `PyodbcConnectionAdapter` инкапсулирует работу с ODBC-подключением, а `PyodbcCursorAdapter` – получение результатов выполнения запросов. Выделение адаптеров позволяет отделить основную логику приложения от особенностей работы библиотеки подключения к базе данных.

Класс `Peer` используется для хранения сведений об активном участнике WebSocket-соединения.

Связи на диаграмме показывают основные зависимости между классами. Маршруты используют сервисы, сервисы обращаются к репозиториям, репозитории наследуются от базового репозитория и используют адаптер подключения к базе данных. Наследование репозитория позволяет повторно использовать общие операции доступа к данным, а наследование ролевых классов отражает расширение прав от зрителя к ведущему и администратору.

Назначение основных классов серверной части приведено в таблице 2.

Таблица 2 – Назначение основных классов серверной части приложения

Класс	Назначение
AuthRouter	обрабатывает маршруты входа, выхода и страницы авторизации
DashboardRouter	обрабатывает страницы личного кабинета, профиля, списка комнат и административного управления
RoomsRouter	обрабатывает создание комнат, подключение к комнатам, загрузку PDF-документов и WebSocket-соединение комнаты
AuthService	выполняет проверку учетных данных и получение текущего пользователя
RoomService	реализует бизнес-логику создания, открытия и управления комнатами
DocumentService	реализует бизнес-логику загрузки, получения и обработки PDF-документов
PermissionService	выполняет проверку прав доступа пользователя с учетом его роли
ViewerRole	содержит базовые разрешения зрителя
SpeakerRole	расширяет разрешения зрителя возможностями ведущего
AdminRole	расширяет разрешения ведущего административными возможностями
SignalingService	управляет WebSocket-подключениями и рассылкой событий участникам комнаты
Peer	хранит сведения об активном WebSocket-подключении участника

Продолжение таблицы 2

Класс	Назначение
BaseRepository	содержит общие операции доступа к данным и обработки результатов SQL-запросов
UserRepository	работает с пользователями, ролями, профилями и учетными данными
RoomRepository	работает с комнатами, участниками, сообщениями, событиями и состоянием комнаты
DocumentRepository	работает с PDF-документами, запросами на загрузку, аннотациями и вопросами
Database	создает подключение к SQL Server и предоставляет механизм работы с базой данных
PyodbcConnectionAdapter	инкапсулирует работу с ODBC-подключением к базе данных
PyodbcCursorAdapter	инкапсулирует получение результатов выполнения SQL-запросов

Таким образом, диаграмма классов показывает внутреннее устройство серверной части приложения на уровне программной реализации. Она уточняет, какие классы отвечают за обработку запросов, выполнение бизнес-логики, проверку прав, доступ к данным и работу с подключениями, а также показывает применение наследования, композиции и агрегации в структуре приложения.

2.5 Проектирование базы данных

Проектирование базы данных выполняется после определения функций системы, ролевой модели и архитектуры приложения. База данных должна обеспечивать хранение учетных записей пользователей, ролей, комнат видеоконференций, участников комнат, сведений о PDF-документах, сообщений, событий, уведомлений и служебной информации, необходимой для синхронизированной работы с документами.

Для описания структуры данных использованы модели «сущность – связь». В методических рекомендациях по выполнению выпускной квалификационной работы модели IDEF1X и другие графические решения

рассматриваются как допустимый способ представления структуры данных и проектных решений информационной системы [20]. Проектирование выполнено поэтапно: сначала сформирована концептуальная модель предметной области, затем построена логическая модель базы данных, после чего разработана физическая модель для Microsoft SQL Server.

2.5.1 Концептуальная модель данных

Концептуальная модель отражает состав основных сущностей предметной области и смысловые связи между ними без привязки к конкретной системе управления базами данных. На данном уровне используются русские наименования сущностей и атрибутов, так как модель предназначена для пояснения структуры предметной области.

Центральными сущностями модели являются пользователь, роль, комната и PDF-документ. Пользователь имеет роль, может создавать комнаты и подключаться к комнатам как участник. Комната объединяет участников, документы, сообщения, события и состояние просмотра. PDF-документ связан с аннотациями и запросами доступа, которые возникают при совместной работе с документом.

При построении модели учитывались как постоянные сведения, необходимые для функционирования системы, так и данные, изменяющиеся в ходе видеоконференции. К постоянным сведениям относятся учетные записи пользователей, назначенные роли и параметры комнат. В процессе работы комнаты формируются данные об участниках, сообщениях, событиях, загруженных PDF-документах, запросах и аннотациях. Выделение таких объектов позволяет обеспечить хранение истории взаимодействия пользователей с документом и состоянием видеоконференции.

Особое значение имеет связь комнаты с PDF-документом и состоянием совместного просмотра. Она позволяет определить, какой документ используется в конкретной конференции и какая страница в данный момент демонстрируется ведущим. Связи пользователя с комнатой, документами и

событиями обеспечивают разграничение прав доступа и фиксацию выполняемых действий.

Концептуальная ERD-модель данных представлена на рисунке 13.

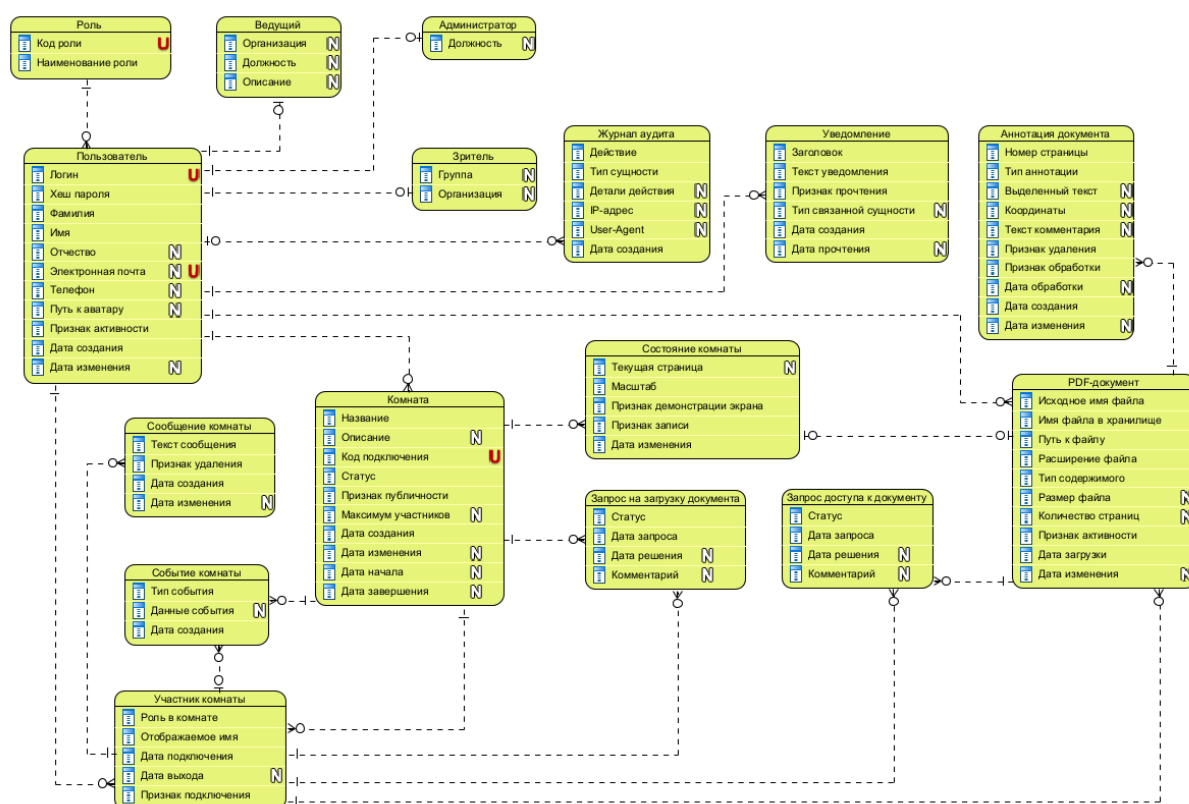


Рисунок 13 – Концептуальная ERD-модель данных

В модели выделены стержневые, характеристические и ассоциативные сущности, представленные в таблице 3. Стержневые сущности отражают основные объекты предметной области, характеристические уточняют их состояние или классификацию.

Таблица 3 – Описание сущностей концептуальной модели данных

№	Сущность	Описание	Категория	Режим обновления
1	Роль	Определяет уровень доступа пользователя в системе	Характеристическая	Редко
2	Пользователь	Хранит учетные и контактные данные	Стержневая	По мере регистрации

Продолжение таблицы 3

№	Сущность	Описание	Категория	Режим обновления
3	Администратор	Уточняет профиль пользователя с административными функциями	Характеристическая	По мере назначения роли
4	Ведущий	Уточняет профиль пользователя, управляющего комнатой и документом	Характеристическая	По мере назначения роли
5	Зритель	Уточняет профиль пользователя, участвующего в обсуждении документа	Характеристическая	По мере назначения роли
6	Комната	Описывает видеоконференцию, код подключения, статус и параметры	Стержневая	При создании и изменении комнаты
7	Участник комнаты	Фиксирует участие пользователя в конкретной комнате	Ассоциативная	При подключении и выходе
8	PDF-документ	Хранит сведения о загруженном PDF-документе	Стержневая	При загрузке и изменении документа
9	Запрос на загрузку документа	Фиксирует обращение участника на загрузку PDF-документа	Стержневая	По мере создания запросов
10	Запрос доступа к документу	Фиксирует запрос доступа к конкретному PDF-документу	Стержневая	По мере создания запросов

Окончание таблицы 3

№	Сущность	Описание	Категория	Режим обновления
11	Аннотация документа	Хранит выделения, комментарии и отметки на страницах PDF-документа	Ассоциативная	При работе с документом
12	Состояние комнаты	Хранит текущую страницу, масштаб и параметры синхронизации просмотра	Характеристическая	Часто
13	Сообщение комнаты	Хранит сообщения участников в чате комнаты	Стержневая	Часто
14	Событие комнаты	Фиксирует значимые действия и события комнаты	Стержневая	Часто
15	Уведомление	Используется для передачи пользователю системных сообщений	Стержневая	По мере событий
16	Журнал аудита	Хранит сведения о важных пользовательских и административных действиях	Стержневая	По мере действий

Сущности концептуальной модели соответствуют функциональности разрабатываемой системы и отражают хранение данных о пользователях, комнатах, для совместной работы с PDF-файлами. Связи между указанными сущностями определяют порядок их взаимодействия. Описание связей концептуальной модели приведено в таблице 4.

Таблица 4 – Описание связей концептуальной модели данных

№	Сущность 1	Сущность 2	Описание связи	Мощность
1	Роль	Пользователь	Одна роль может быть назначена многим пользователям	1 : M
2	Пользователь	Администратор	Профиль администратора уточняет учетную запись пользователя	1 : 0..1
3	Пользователь	Ведущий	Профиль ведущего уточняет учетную запись пользователя	1 : 0..1
4	Пользователь	Зритель	Профиль зрителя уточняет учетную запись пользователя	1 : 0..1
5	Пользователь	Комната	Пользователь может создавать комнаты	1 : M
6	Пользователь	Участник комнаты	Пользователь может участвовать в разных комнатах	1 : M
7	Комната	Участник комнаты	Комната содержит участников	1 : M
8	Комната	PDF-документ	В комнате могут быть загружены PDF-документы	1 : M
9	Участник комнаты	PDF-документ	Участник комнаты может загрузить документ	1 : M
10	Комната	Запрос на загрузку документа	Запрос на загрузку создается в рамках комнаты	1 : M
11	Участник комнаты	Запрос на загрузку документа	Запрос на загрузку отправляет участник комнаты	1 : M
12	PDF-документ	Запрос доступа к документу	Запрос доступа относится к конкретному документу	1 : M
13	Участник комнаты	Запрос доступа к документу	Запрос доступа создает участник комнаты	1 : M
14	PDF-документ	Аннотация документа	Документ может содержать несколько аннотаций	1 : M
15	Участник комнаты	Аннотация документа	Аннотацию создает участник комнаты	1 : M

Продолжение таблицы 4

№	Сущность 1	Сущность 2	Описание связи	Мощность
16	Комната	Состояние комнаты	Комната имеет текущее состояние просмотра	1 : 0..1
17	PDF-документ	Состояние комнаты	Состояние комнаты может ссылаться на активный документ	0..1 : 0..1
18	Участник комнаты	Сообщение комнаты	Сообщение отправляется участником комнаты	1 : M
19	Комната	Событие комнаты	Событие фиксируется в рамках комнаты	1 : M
20	Участник комнаты	Событие комнаты	Событие может быть связано с участником комнаты	0..1 : M
21	Пользователь	Уведомление	Пользователь может получать уведомления	1 : M
22	Пользователь	Журнал аудита	Запись аудита может быть связана с пользователем	0..1 : M

Ассоциативная сущность «Участник комнаты» уточняет действия пользователя в рамках конкретной комнаты.. Основные атрибуты сущностей концептуальной модели приведены в таблице 5.

Таблица 5 – Основные атрибуты сущностей концептуальной модели

Сущность	Атрибут	Тип данных	Роль	Обязательный	Описание
Пользователь	ID пользователя	целый	РК	да	Уникальный идентификатор учетной записи
Пользователь	Логин	строка		да	Используется для входа в систему
Пользователь	Хеш пароля	строка		да	Хранит преобразованное значение пароля
Пользователь	ID роли	целый	FK	да	Ссылка на роль пользователя

Продолжение таблицы 5

Сущность	Атрибут	Тип данных	Роль	Обязательный	Описание
Комната	ID комнаты	целый	РК	да	Уникальный идентификатор комнаты
Комната	Код комнаты	строка		да	Код подключения участников
Комната	Статус	строка		да	Запланирована, активна, завершена или отменена
PDF-документ	ID документа	целый	РК	да	Уникальный идентификатор документа
PDF-документ	Путь файла	строка		да	Путь к PDF-файлу на сервере
PDF-документ	ID комнаты	целый	FK	да	Комната, к которой относится документ
Аннотация	ID аннотации	целый	РК	да	Уникальный идентификатор аннотации
Аннотация	Номер страницы	целый		да	Страница документа, на которой создана аннотация
Сообщение	ID сообщения	целый	РК	да	Уникальный идентификатор сообщения
Сообщение	Текст сообщения	строка		да	Содержимое сообщения чата
Состояние комнаты	Текущая страница	целый		да	Номер страницы, синхронизируемый между участниками

Концептуальная модель задает состав данных, необходимых для выполнения основных сценариев системы. Она определяет объекты предметной области и связи между ними без учета особенностей выбранной СУБД. На

следующем этапе сущности предметной области преобразуются в отношения логической модели.

2.5.2 Логическая модель базы данных

На основе концептуальной модели разработана логическая модель базы данных. На данном уровне сущности преобразованы в отношения, определены ключевые атрибуты, обязательность связей и зависимость отношений друг от друга. В отличие от физической модели, логическая модель сохраняет русские наименования сущностей и полей, так как ее задача состоит в описании структуры данных без привязки к конкретным SQL-именам таблиц.

Логическая модель базы данных информационной системы представлена на рисунке 14.

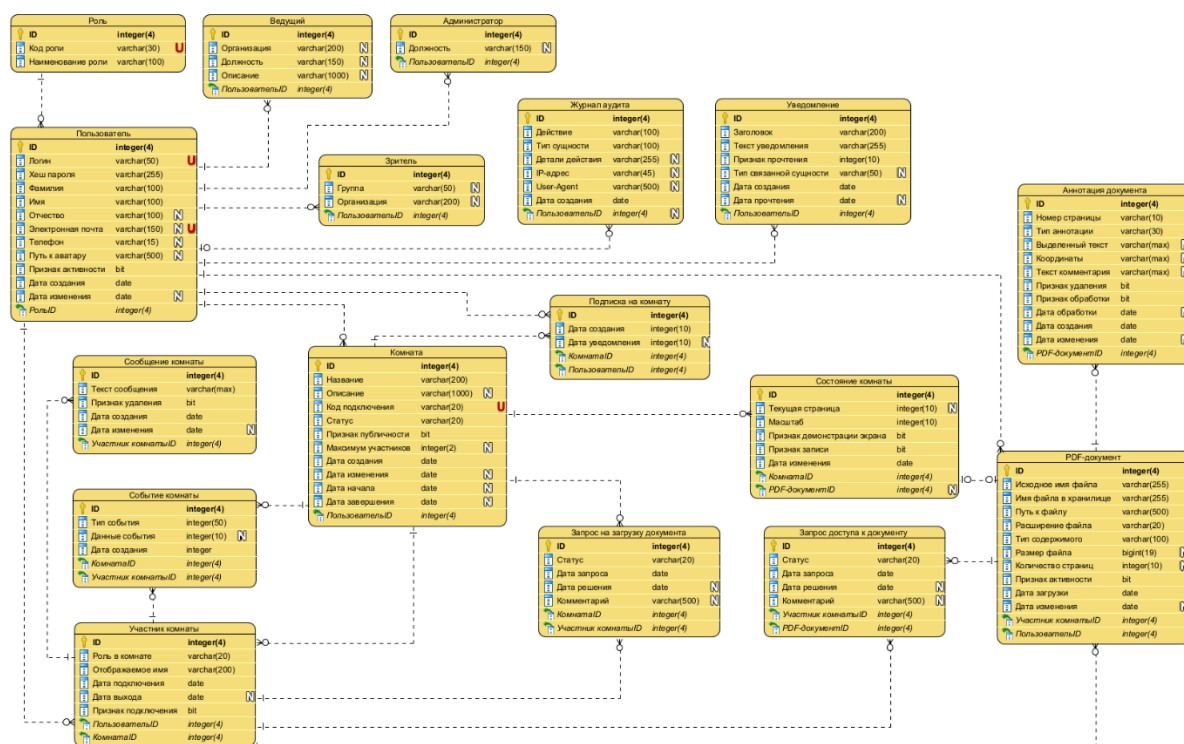


Рисунок 14 – Логическая модель базы данных информационной системы

В логической модели дополнительно показана сущность «Подписка на комнату». Она фиксирует связь пользователя с комнатой для предоставления доступа и отправки уведомлений. В концептуальной модели данная сущность не выделялась отдельно, однако в логической структуре она необходима для

реализации отношения многие ко многим. Такая организация данных позволяет хранить сведения о статусе подписки и времени ее создания. Спецификация отношений логической модели базы данных приведена в таблице 6.

Таблица 6 – Спецификация отношений логической модели базы данных

Отношение	Тип отношения	Назначение
Роль	Независимое	Справочник ролей пользователей
Пользователь	Независимое	Основная учетная запись пользователя
Администратор	Зависимое	Профиль администратора, связанный с пользователем
Ведущий	Зависимое	Профиль ведущего, связанный с пользователем
Зритель	Зависимое	Профиль зрителя, связанный с пользователем
Комната	Зависимое	Сведения о комнате видеоконференции
Участник комнаты	Зависимое	Состав участников комнаты и факт подключения
Подписка на комнату	Зависимое	Связь пользователя с комнатой для доступа и уведомления
PDF-документ	Зависимое	Сведения о загруженном PDF-документе
Запрос на загрузку документа	Зависимое	Запрос участника на загрузку PDF-документа
Аннотация документа	Зависимое	Аннотации, выделения и комментарии на страницах PDF-документа
Состояние комнаты	Зависимое	Текущая страница, активный документ и параметры просмотра
Сообщение комнаты	Зависимое	Сообщения участников в чате комнаты
Событие комнаты	Зависимое	События, возникающие в процессе работы комнаты
Уведомление	Зависимое	Уведомления пользователя
Журнал аудита	Зависимое	Журнал важных действий пользователей и системы

Спецификация отношений показывает зависимости сущностей от пользователя, комнаты, документа или участника комнаты. Основные связи отношений логической модели приведены в таблице 7.

Таблица 7 – Основные связи логической модели базы данных

Родительское отношение	Дочернее отношение	Мощность	Описание	Тип связи
Роль	Пользователь	1 : М	Пользователь имеет одну роль, роль может быть назначена многим пользователям	Неидентифицирующая
Пользователь	Администратор	1 : 0..1	Профиль администратора уточняет учетную запись пользователя	Идентифицирующая
Пользователь	Ведущий	1 : 0..1	Профиль ведущего уточняет учетную запись пользователя	Идентифицирующая
Пользователь	Зритель	1 : 0..1	Профиль зрителя уточняет учетную запись пользователя	Идентифицирующая
Пользователь	Комната	1 : М	Пользователь может создавать комнаты	Неидентифицирующая
Пользователь	Участник комнаты	1 : М	Пользователь может быть участником нескольких комнат	Неидентифицирующая
Комната	Участник комнаты	1 : М	Комната содержит записи об участниках	Неидентифицирующая
Пользователь	Подписка на комнату	1 : М	Пользователь может иметь несколько подписок на комнаты	Неидентифицирующая
Комната	Подписка на комнату	1 : М	Комната может иметь несколько подписок пользователей	Неидентифицирующая
Комната	PDF-документ	1 : М	Комната содержит загруженные документы	Неидентифицирующая
Участник комнаты	PDF-документ	1 : М	Участник комнаты может загрузить документ	Неидентифицирующая

Продолжение таблицы 7

Родительское отношение	Дочернее отношение	Мощность	Описание	Тип связи
Участник комнаты	Запрос на загрузку документа	1 : M	Запрос на загрузку создает участник комнаты	Неидентифицирующая
PDF-документ	Запрос доступа к документу	1 : M	Запрос доступа относится к документу	Неидентифицирующая
Участник комнаты	Запрос доступа к документу	1 : M	Запрос доступа создает участник комнаты	Неидентифицирующая
PDF-документ	Аннотация документа	1 : M	Документ содержит аннотации	Неидентифицирующая
Комната	Состояние комнаты	1 : 0..1	Комната имеет текущее состояние просмотра	Идентифицирующая
PDF-документ	Состояние комнаты	0..1 : 0..1	Состояние комнаты может ссылаться на активный документ	Неидентифицирующая
Участник комнаты	Сообщение комнаты	1 : M	Сообщение отправляется участником комнаты	Неидентифицирующая
Комната	Событие комнаты	1 : M	Событие фиксируется в рамках комнаты	Неидентифицирующая
Участник комнаты	Событие комнаты	0..1 : M	Событие может быть связано с участником комнаты	Неидентифицирующая
Пользователь	Уведомление	1 : M	Пользователь получает уведомления	Неидентифицирующая
Пользователь	Журнал аудита	0..1 : M	Запись аудита может быть связана с пользователем	Неидентифицирующая

Логическая модель уточняет структуру хранения данных и подготавливает основу для физического проектирования. На следующем этапе русские

наименования отношений и атрибутов заменяются реальными именами таблиц и столбцов, используемыми в SQL Server.

2.5.3 Физическая модель базы данных

Физическая модель базы данных разработана для Microsoft SQL Server. На данном уровне определены реальные имена таблиц и столбцов, типы данных, первичные и внешние ключи, признаки обязательности, уникальности и индексирования. Выбор Microsoft SQL Server согласуется с используемой серверной частью приложения и средствами подключения к базе данных [17].

Физическая модель базы данных представлена на рисунке 15.

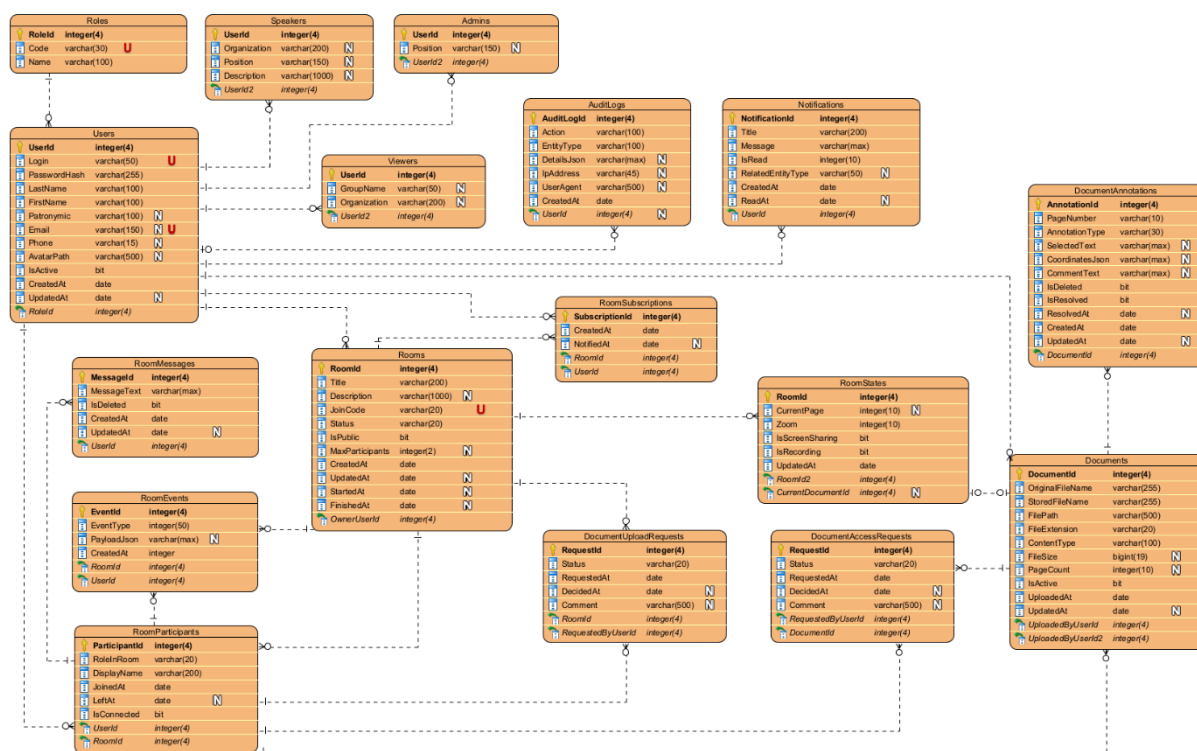


Рисунок 15 – Физическая модель базы данных

В физической модели используются латинские имена таблиц и столбцов, соответствующие SQL-скрипту создания базы данных. Такая модель непосредственно связана с реализацией хранилища и применяется при разработке SQL-запросов в репозиториях серверной части. Кроме наименований, в ней определяются типы данных полей, первичные и внешние ключи, а также ограничения целостности. Спецификация таблиц физической

модели базы данных приведена в таблице 8.

Таблица 8 – Спецификация таблиц физической модели базы данных

Таблица	Первичный ключ	Основные внешние ключи	Назначение
Roles	RoleId	–	Справочник ролей пользователей
Users	UserId	RoleId	Хранение учетных записей и общих данных пользователей
Admins	UserId	UserId	Профиль администратора
Speakers	UserId	UserId	Профиль ведущего
Viewers	UserId	UserId	Профиль зрителя
Rooms	RoomId	OwnerUserId	Хранение комнат, кодов подключения, статусов и параметров доступа
RoomParticipants	ParticipantId	RoomId, UserId	Фиксация состава участников и состояния подключения
RoomSubscriptions	SubscriptionId	RoomId, UserId	Хранение связи пользователя с комнатой для доступа и уведомлений
Documents	DocumentId	RoomId, UploadedByUserId	Хранение сведений о загруженных PDF-документах
Document UploadRequests	RequestId	RoomId, RequestedByUserId	Хранение запросов на загрузку PDF-документа
Document AccessRequests	RequestId	DocumentId, RequestedByUserId	Хранение запросов доступа к PDF-документам
Document Annotations	AnnotationId	DocumentId	Хранение выделений, комментариев и отметок на страницах PDF-документов
RoomStates	RoomId	RoomId, CurrentDocumentId	Хранение текущей страницы, активного документа и параметров синхронизации
RoomMessages	MessageId	ParticipantId	Хранение сообщений чата комнаты

Продолжение таблицы 8

Таблица	Первичный ключ	Основные внешние ключи	Назначение
Notifications	NotificationId	UserId	Хранение уведомлений пользователей
AuditLogs	AuditLogId	UserId	Фиксация административных и системных действий

Физическая модель предусматривает ограничения целостности, запрещающие создание дочерних записей без связанных родительских объектов. Например, участник комнаты не может существовать без пользователя и комнаты, документ не может быть связан с несуществующей комнатой, а запрос доступа должен относиться к конкретному PDF-документу. Основные типы данных физической модели и их использование приведены в таблице 9.

Таблица 9 – Основные типы данных физической модели

Тип данных	Использование
INT	Идентификаторы таблиц, внешние ключи, числовые параметры страниц и масштаба
NVARCHAR(n)	Логины, названия, статусы, имена файлов, пути хранения и другие строковые значения ограниченной длины
NVARCHAR(MAX)	Тексты сообщений, комментарии, координаты аннотаций и данные событий
BIT	Логические признаки активности, удаления, подключения, демонстрации экрана и записи
DATETIME2(0)	Даты создания, изменения, подключения, выхода, запроса, решения и загрузки
BIGINT	Размер загруженного файла в байтах

Использование перечисленных типов позволяет разделить идентификаторы, строковые значения, большие текстовые данные, логические признаки и временные отметки. Для ускорения доступа к данным в базе предусматриваются индексы по часто используемым полям: коду подключения комнаты, идентификаторам пользователя, комнаты, документа и датам создания

записей.

Таким образом, проектирование базы данных выполнено последовательно: концептуальная модель определяет предметную область и смысловые связи, логическая модель уточняет отношения и их зависимости, а физическая модель задает структуру таблиц для Microsoft SQL Server. Полученная структура обеспечивает хранение данных, необходимых для авторизации пользователей, управления комнатами, совместной работы с PDF-документами, обмена сообщениями, фиксации событий и формирования уведомлений.

2.6 Моделирование поведения системы

Для описания динамики работы информационной системы были разработаны UML-диаграммы последовательности, деятельности, кооперации и состояний. Они позволяют отразить не только статическую структуру приложения, но и порядок выполнения основных сценариев, связанных с подключением пользователя к комнате, работой с PDF-документом, синхронизацией аннотаций, согласованием загрузки документа и изменением состояния комнаты. Применение UML-диаграмм при проектировании информационной системы позволяет наглядно представить взаимодействие пользователей, клиентской части, серверных модулей, WebSocket-соединения, файлового хранилища и базы данных [6].

На рисунке 16 представлена диаграмма последовательности подключения пользователя к комнате. Перед выполнением данного сценария пользователь уже проходит авторизацию в системе, поэтому в диаграмме основное внимание уделено именно процессу входа в комнату по коду подключения. Пользователь вводит код комнаты в клиентской части приложения, после чего веб-клиент отправляет запрос на сервер. Серверная часть через маршрутизатор RoomsRouter передает данные в RoomService, где выполняется обращение к RoomRepository и базе данных для поиска комнаты по коду подключения. После получения сведений о комнате сервис проверяет ее статус и доступность, добавляет пользователя в список участников и возвращает результат подключения. Далее

клиентская часть выполняет переход на страницу комнаты и открывает WebSocket-соединение, необходимое для получения событий в режиме реального времени [10].

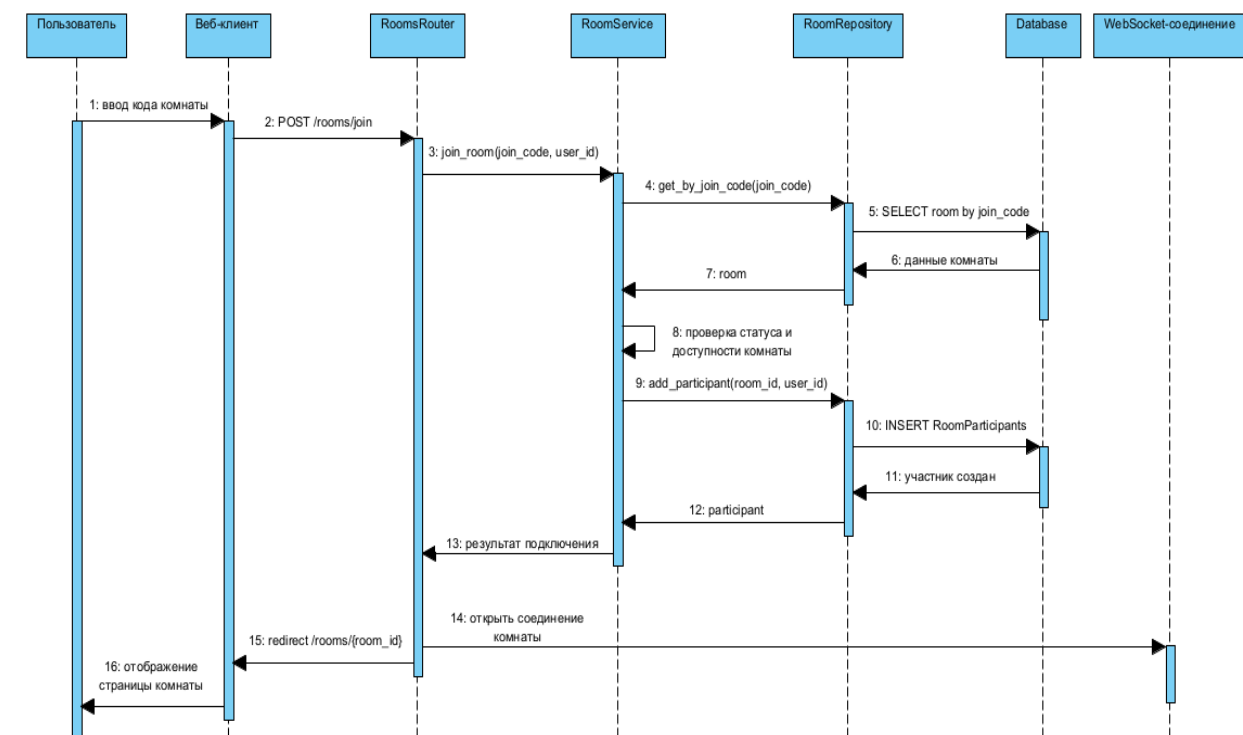


Рисунок 16 – Диаграмма последовательности подключения пользователя к комнате

Диаграмма последовательности синхронизации аннотаций PDF-документа представлена на рисунке 17. Данный сценарий отражает одну из ключевых функций разрабатываемой системы – совместную работу с PDF-документом в условиях видеоконференции. Ведущий добавляет аннотацию к документу в веб-клиенте, после чего клиент отправляет событие `annotation:add` через `WebSocketEndpoint`. Серверная часть передает данные аннотации в `DocumentService`, затем в `DocumentRepository`, где выполняется сохранение записи в таблице `DocumentAnnotations`. После успешного сохранения сервис подготавливает результат и инициирует рассылку события `annotation:created` другим участникам комнаты. В результате остальные участники получают обновление и видят созданную аннотацию на странице PDF-документа. Отображение PDF-документа в клиентской части выполняется средствами

PDF.js [16], а обмен событиями между участниками реализуется через WebSocket-соединение [10].

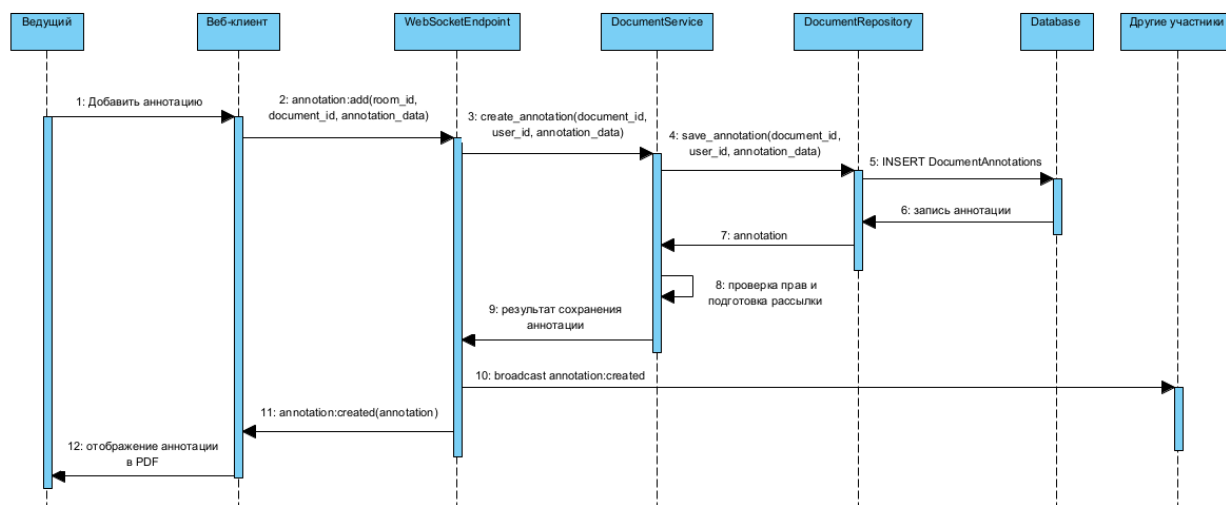


Рисунок 17 – Диаграмма последовательности синхронизации аннотаций PDF-документа

На рисунке 18 представлена диаграмма деятельности согласования загрузки PDF-документа зрителем. В отличие от диаграмм последовательности, данная диаграмма показывает не обмен сообщениями между программными компонентами, а общий процесс взаимодействия ролей в системе. Сценарий начинается с того, что зритель выбирает команду запроса загрузки PDF-документа и заполняет сведения о документе. Если данные заполнены некорректно, пользователь возвращается к исправлению формы. При корректном заполнении система регистрирует запрос и уведомляет ведущего. Ведущий просматривает запрос и принимает решение. Если запрос отклонен, зритель получает уведомление об отказе. Если запрос одобрен, зрителю право загрузки документа. После этого пользователь может загрузить PDF-файл либо отложить загрузку и вернуться к форме позднее. Завершается процесс обновлением состояния запроса и уведомлением участников комнаты.

Таким образом, диаграмма отражает контролируемый порядок передачи права на изменение общего документа. Решение ведущего исключает загрузку файлов без согласования с пользователем, управляющим комнатой.

Возможность отложить загрузку сохраняет принятое решение и позволяет зрителю вернуться к ней позднее. Это обеспечивает единый демонстрируемый документ для участников конференции.

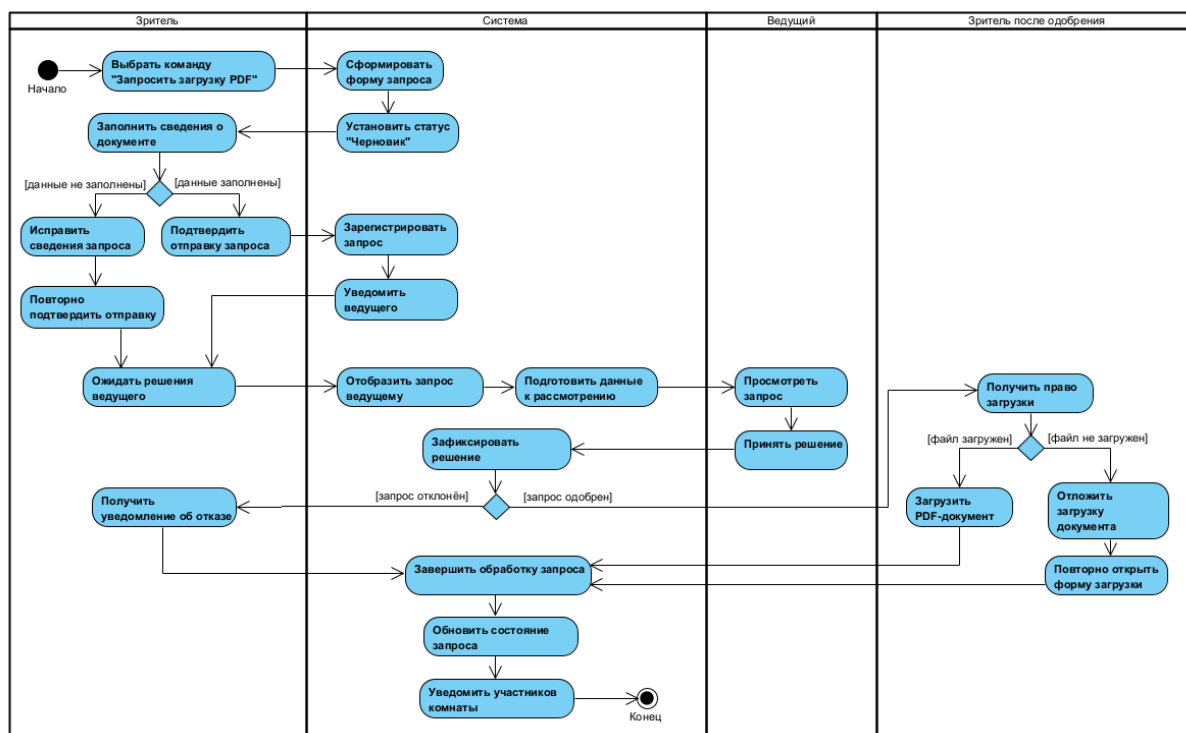


Рисунок 18 – Диаграмма деятельности согласования загрузки PDF-документа зрителем

Диаграмма кооперации взаимодействия компонентов комнаты представлена на рисунке 19. Она отображает не строгую временную последовательность действий, а общую структуру обмена сообщениями между основными компонентами комнаты. В состав взаимодействующих объектов входят веб-клиент комнаты, PDF.js, WebSocketEndpoint, SignalingService, RoomService, DocumentService, AnnotationService, репозитории, база данных и файловое хранилище. Веб-клиент использует PDF.js для открытия и отображения страниц PDF-документа, WebSocketEndpoint принимает и рассылает события комнаты, SignalingService отвечает за передачу событий синхронизации участникам, а сервисы RoomService, DocumentService и AnnotationService реализуют прикладную логику работы с комнатой, документами и аннотациями. Репозитории обеспечивают доступ к данным, база

данных хранит сведения о комнатах, участниках, документах и аннотациях, а файловое хранилище используется для сохранения загруженных PDF-файлов.

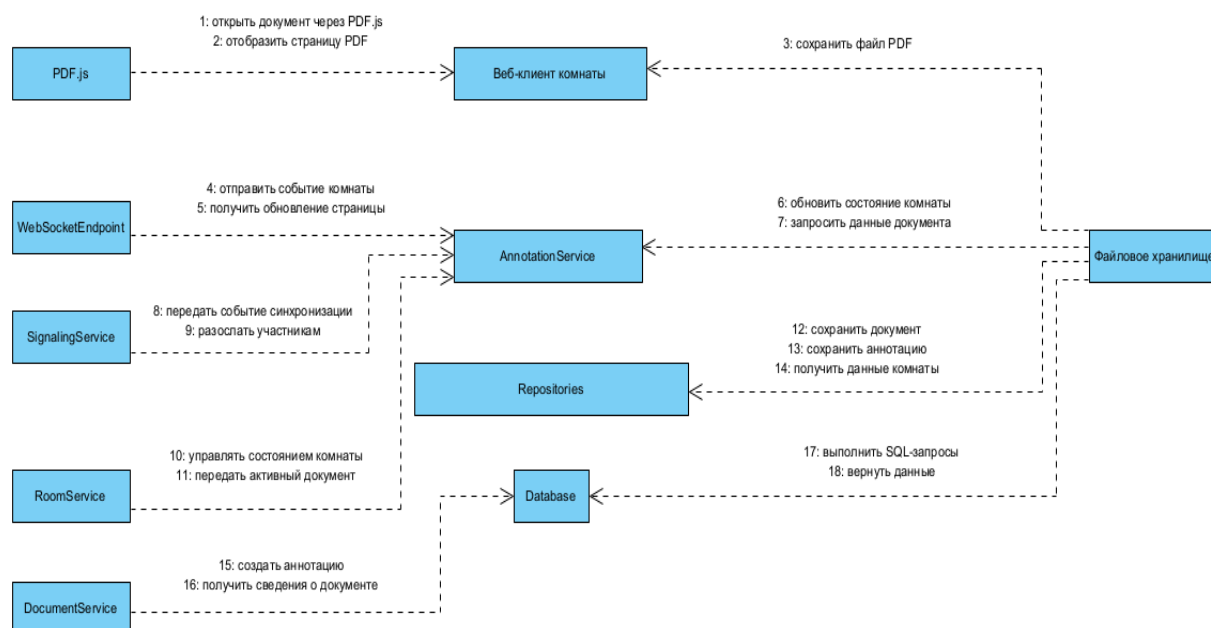


Рисунок 19 – Диаграмма кооперации взаимодействия компонентов комнаты

На рисунке 20 представлена диаграмма состояний комнаты. Данная диаграмма описывает жизненный цикл комнаты от момента создания до завершения или отмены. После создания комната переходит в состояние ожидания начала. Далее возможны два основных варианта: комната может быть запущена и перейти в активное состояние либо завершена без запуска. В активном состоянии в комнате могут выполняться видеосвязь, обмен событиями и взаимодействие участников. После загрузки PDF-документа комната переходит в состояние работы с PDF-документом, где выполняются просмотр страниц, синхронизация действий и работа с аннотациями. После окончания конференции комната переводится в завершенное состояние. Также предусмотрена возможность отмены комнаты, при которой фиксируется факт отмены и работа комнаты прекращается.

Переходы между состояниями определяются действиями ведущего и событиями, возникающими в процессе работы комнаты. Такая модель позволяет контролировать доступные функции на каждом этапе видеоконференции. При

этом завершенные и отмененные комнаты не допускают дальнейшего изменения данных.

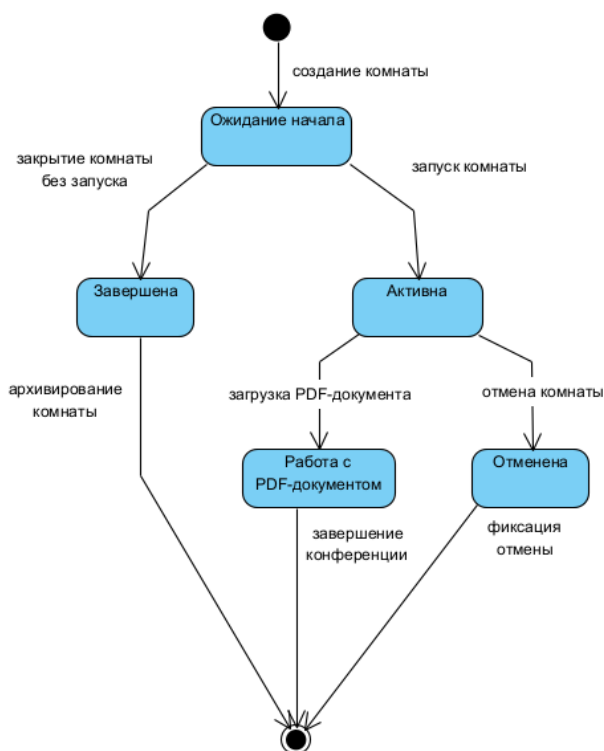


Рисунок 20 – Диаграмма состояний комнаты

Разработанные диаграммы поведения дополняют ранее построенные структурные и информационные модели системы. Они показывают, каким образом пользователи и программные компоненты взаимодействуют при выполнении ключевых сценариев: подключении к комнате, синхронизации действий с PDF-документом, согласовании загрузки файла и изменении состояния комнаты. Это позволяет уточнить логику работы приложения до этапа реализации и связать функциональные требования с проектными решениями серверной и клиентской частей.

2.7 Проектирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемого веб-приложения должен обеспечивать быстрый доступ к основным функциям системы: входу в приложение, просмотру профиля, созданию и подключению к комнате,

управлению пользователями, работе с PDF-документом и участию в видеоконференции. При проектировании интерфейса учитывалось, что набор доступных разделов и действий зависит от роли пользователя в системе.

Структура пользовательского интерфейса веб-приложения представлена на рисунке 21. После успешной авторизации система определяет роль пользователя и открывает соответствующий кабинет. Такой подход позволяет не перегружать интерфейс лишними элементами и отображать только те функции, которые доступны конкретному типу пользователя.

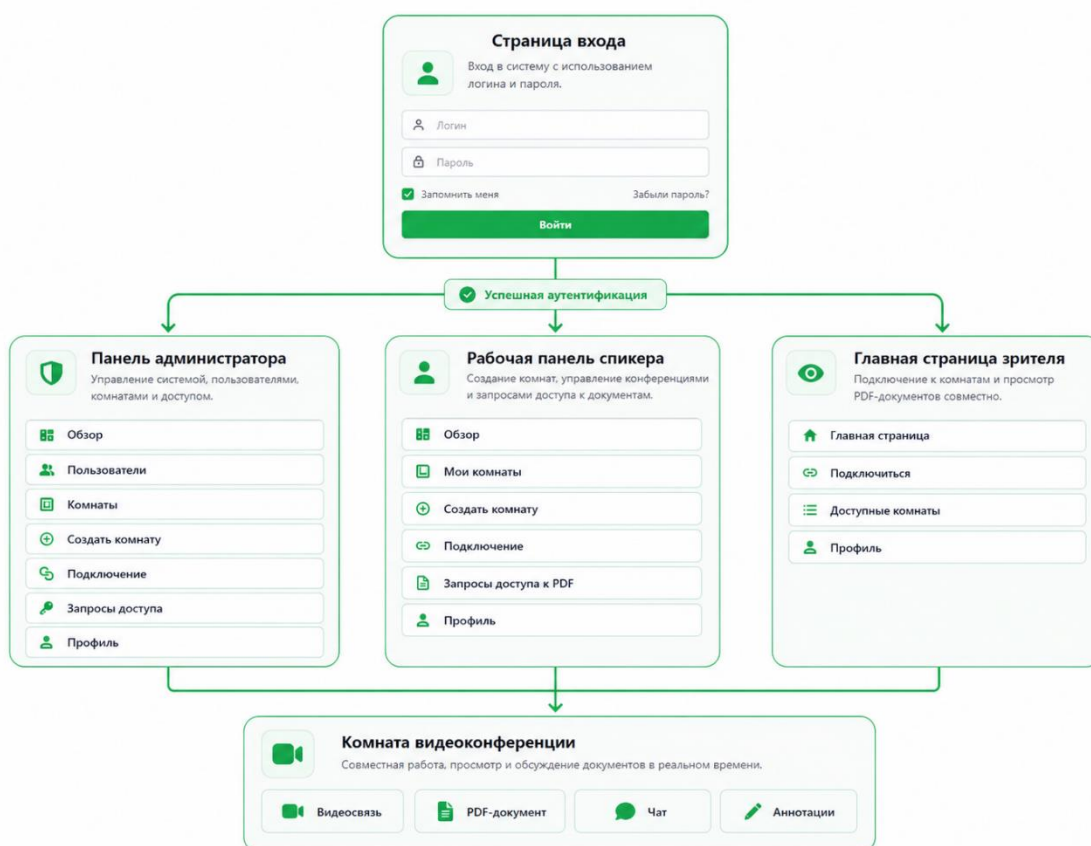


Рисунок 21 – Структура пользовательского интерфейса веб-приложения

Для администратора предусмотрена панель администратора, предназначенная для управления системой, пользователями, комнатами и доступом. В данном кабинете доступны разделы обзора, управления пользователями, просмотра комнат, создания комнаты, подключения, обработки запросов доступа и просмотра профиля. Основной задачей администратора является контроль учетных записей и общего состояния системы.

Для ведущего предусмотрена рабочая панель, ориентированная на создание и проведение видеоконференций. Ведущий может просматривать свои комнаты, создавать новую комнату, подключаться к комнате, обрабатывать запросы доступа к PDF-документам и открывать профиль. После перехода в комнату видеоконференции ведущему становятся доступны функции управления PDF-документом, синхронизации просмотра страниц, работы с аннотациями, чатом и участниками.

Для зрителя предусмотрена главная страница, предназначенная для подключения к комнате и участия в совместном просмотре PDF-документа. Зритель может подключиться к комнате по коду, просматривать доступные комнаты и открывать профиль. В комнате видеоконференции зрителю доступны видеосвязь, просмотр PDF-документа, чат, аннотации и отправка запроса на загрузку собственного PDF-документа.

Макет интерфейса комнаты видеоконференции представлен на рисунке 22. Данный макет отражает основную рабочую область системы, в которой пользователи одновременно участвуют в видеосвязи и работают с PDF-документом.

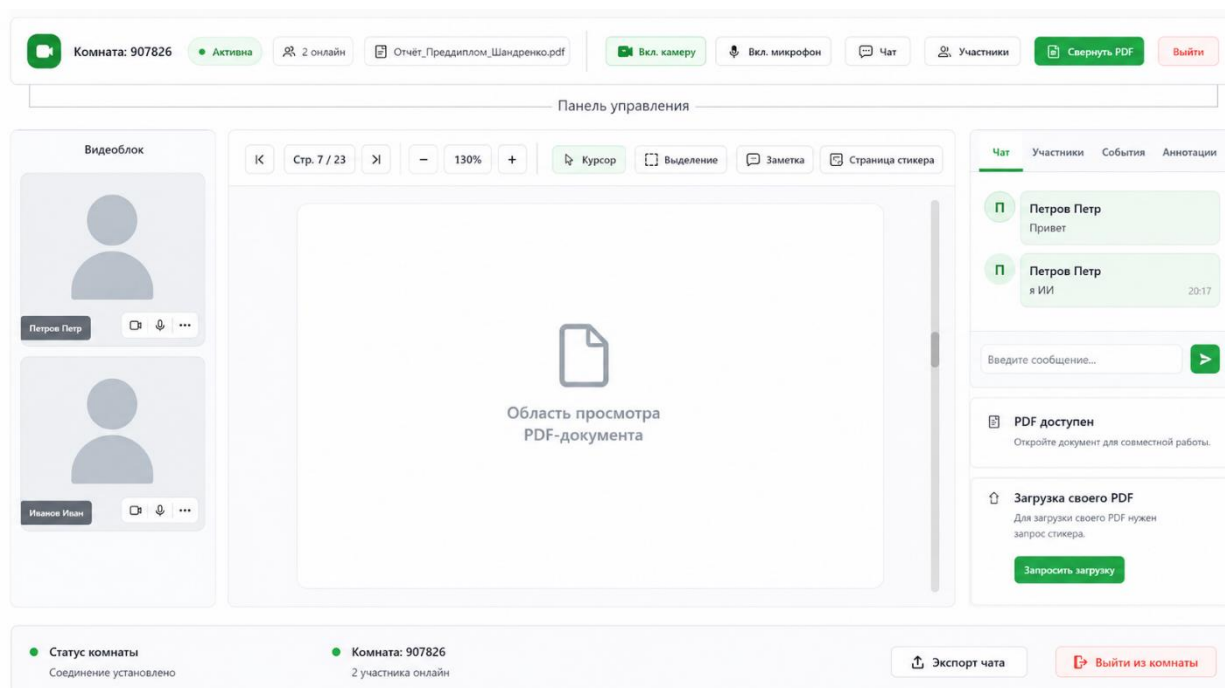


Рисунок 22 – Макет интерфейса комнаты видеоконференции

В верхней части интерфейса комнаты располагается панель с основной информацией о комнате: названием, кодом подключения, статусом, количеством участников онлайн и названием открытого PDF-документа. Также на верхней панели размещаются основные кнопки управления: включение камеры, включение микрофона, открытие чата, просмотр участников, сворачивание PDF-документа и выход из комнаты.

Левая часть экрана предназначена для видеоблока. В нем отображаются участники видеоконференции, их имена и элементы управления камерой и микрофоном. Такое расположение позволяет пользователю видеть участников обсуждения, не закрывая область работы с документом.

Центральная часть интерфейса отведена под область просмотра PDF-документа.

Правая часть интерфейса содержит блок взаимодействия участников. В нем предусмотрены вкладки чата, участников, событий и аннотаций. Ниже располагаются информационные блоки, связанные с доступностью PDF-документа и загрузкой собственного PDF-файла. Для зрителя отображается возможность отправить запрос на загрузку документа, а для ведущего в этом месте могут отображаться элементы рассмотрения и обработки таких запросов.

Таким образом, спроектированный пользовательский интерфейс разделен на отдельные кабинеты в зависимости от роли пользователя и общую рабочую страницу комнаты видеоконференции. Такая структура обеспечивает удобную навигацию, разграничение доступных действий и одновременную работу с видеосвязью, PDF-документом, чатом, аннотациями и запросами участников.

2.8 Планирование разработки и оценка бюджета

Планирование процесса разработки и расчет бюджета проекта были выполнены в Microsoft Excel. В рамках проекта определены ключевые этапы работ, их длительность, сроки начала и завершения, зависимости между задачами, задействованные ресурсы и стоимость выполнения. В качестве даты начала проекта принято 8 апреля 2026 года.

В результате планирования выделено 16 задач, охватывающих анализ предметной области и требований, анализ аналогов, выбор технологий, проектирование ролевой модели и UML-диаграмм, проектирование архитектуры приложения, моделирование базы данных, разработку SQL-скрипта, реализацию серверной и клиентской частей, интеграцию модулей, тестирование и оформление результатов. Список задач проекта представлен на рисунке 23.

Список задач проекта							
№	Название задачи	Длительность, дн.	Дата начала	Дата окончания	Предшественники	Ресурс	Стоимость, руб.
1	Анализ предметной области и требований	3	08.04.2026	10.04.2026		Аналитик	14 400 Р
2	Анализ аналогов и уточнение функциональности	3	13.04.2026	15.04.2026	1	Аналитик	14 400 Р
3	Выбор технологий и архитектурного подхода	2	16.04.2026	17.04.2026	2	Аналитик	9 600 Р
4	Проектирование ролевой модели и UML-диаграмм	3	20.04.2026	22.04.2026	3	Аналитик	14 400 Р
5	Проектирование архитектуры приложения	3	20.04.2026	22.04.2026	3	Разработчик	13 200 Р
6	Концептуальное и логическое моделирование БД	5	23.04.2026	29.04.2026	4,5	Аналитик	24 000 Р
7	Физическое моделирование БД и SQL-скрипт	3	30.04.2026	04.05.2026	6	Разработчик	13 200 Р
8	Реализация подключения к БД и репозитория	4	05.05.2026	08.05.2026	7	Разработчик	17 600 Р
9	Реализация авторизации и ролевого доступа	3	11.05.2026	13.05.2026	8	Разработчик	13 200 Р
10	Реализация комнат и WebSocket-взаимодействия	4	11.05.2026	14.05.2026	8	Разработчик	17 600 Р
11	Реализация видеосвязи WebRTC	3	15.05.2026	19.05.2026	10	Разработчик	13 200 Р
12	Реализация просмотра и синхронизации PDF	4	18.05.2026	21.05.2026	10	Разработчик	17 600 Р
13	Реализация аннотаций, вопросов и чата	4	22.05.2026	27.05.2026	12	Разработчик	17 600 Р
14	Разработка клиентского интерфейса и шаблонов	6	11.05.2026	18.05.2026	9	Разработчик	26 400 Р
15	Интеграция, отладка и исправление ошибок	3	28.05.2026	01.06.2026	13,14	Разработчик	13 200 Р
16	Тестирование и оформление результатов	4	02.06.2026	05.06.2026	15	Тестировщик	16 000 Р
Итого трудозатраты по задачам, руб.							2 55 600 Р
Консультации руководителя проекта (20 ч.), руб.							18 000 Р
Оценка бюджета проекта, руб.							2 73 600 Р

Рисунок 23 – Список задач проекта

Для выполнения задач проекта выделены трудовые ресурсы: аналитик, разработчик, тестировщик и консультант. Также учтен материальный ресурс – компьютер и программное обеспечение, используемые при разработке. В таблице ресурсов указаны коды ресурсов, тип, единицы измерения, ставка, количество, назначение и плановая стоимость. Список ресурсов проекта представлен на рисунке 24.

Список ресурсов							
Код	Наименование ресурса	Тип	Ед. изм.	Ставка, руб./ч	Количество	Назначение	Плановая стоимость, руб.
R1	Аналитик	Трудовой	ч	600 Р	1	Анализ требований, проектирование моделей	76 800 Р
R2	Разработчик	Трудовой	ч	550 Р	1	Реализация серверной и клиентской части	1 62 800 Р
R3	Тестировщик	Трудовой	ч	500 Р	1	Функциональное тестирование и проверка сценариев	16 000 Р
R4	Консультант	Трудовой	ч	900 Р	1	Консультации по постановке задачи и проверке результатов	18 000 Р
R5	Компьютер и программное обеспечение	Материальный	комплект	000 Р	1	Используется существующее оборудование и свободно доступные средства	000 Р
Итого по ресурсам, руб.							2 73 600 Р

Рисунок 24 – Список ресурсов проекта

На основе списка задач, сроков выполнения и предшествующих этапов была построена диаграмма Ганта, представленная на рисунке 25. Диаграмма отображает календарное распределение работ с 8 апреля по 5 июня 2026 года, а также показывает последовательность выполнения этапов и параллельную реализацию отдельных задач проекта. Для наглядности этапы работ выделены разными цветами: голубым цветом обозначены аналитические и проектные работы, зеленым цветом – задачи разработки и реализации программных модулей, оранжевым цветом – тестирование и оформление результатов.

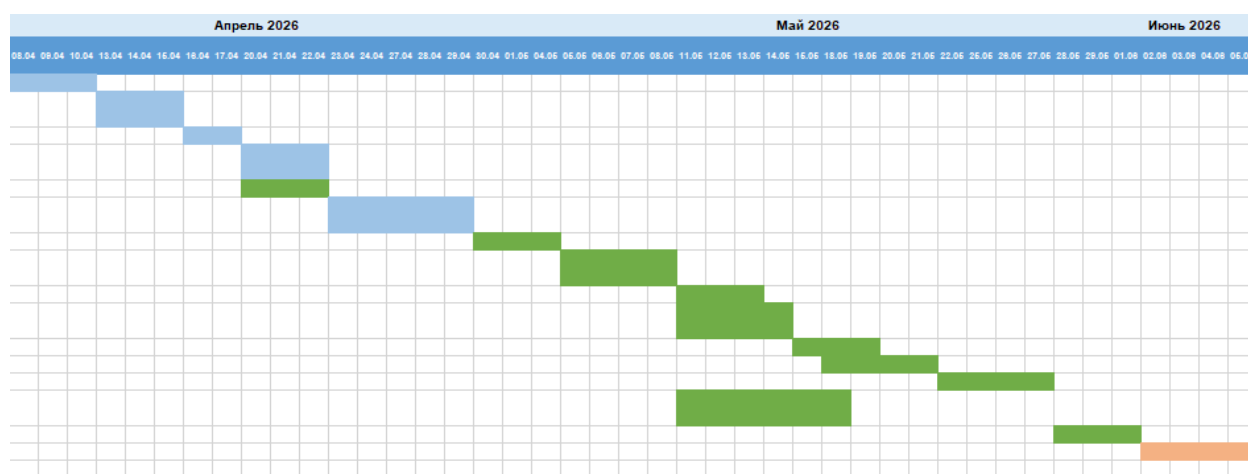


Рисунок 25 – Диаграмма Ганта для задач проекта

Согласно выполненному планированию, продолжительность проекта составляет 43 рабочих дня: с 08.04.2026 по 05.06.2026. Итоговая стоимость трудозатрат по задачам составляет 255 600 рублей. Дополнительно учтены консультации руководителя проекта в размере 18 000 рублей. Общая оценка бюджета проекта составляет 273 600 рублей. Полученные значения используются как предварительная оценка трудозатрат и стоимости разработки информационной системы для работы с PDF-документами в условиях видеоконференции.

Таким образом, в разделе проектирования были определены основные проектные решения для разрабатываемой информационной системы. Была описана ролевая модель пользователей с наследованием функциональных возможностей, разработаны диаграммы вариантов использования для зрителя, ведущего и администратора, определена клиент-серверная и многоуровневая

архитектура приложения, спроектированы программные компоненты и структура базы данных. Также были рассмотрены поведенческие модели системы, структура пользовательского интерфейса, план разработки и предварительная оценка бюджета проекта. Полученные проектные решения являются основой для последующей реализации серверной части, клиентского интерфейса, базы данных и механизмов синхронизации работы с PDF-документом в условиях видеоконференции.

Детализированные схемы и диаграммы, дополняющие проектные решения, приведены в приложении Б.

3 Разработка информационной системы

Третий раздел пояснительной записки посвящен реализации информационной системы для работы с PDF-документами в условиях видеоконференции. На данном этапе проектные решения, разработанные во втором разделе, преобразуются в программную реализацию: создается база данных, формируется структура программного проекта, реализуются подключение к базе данных, серверная логика, клиентский интерфейс, работа с комнатами, PDF-документами и событиями синхронизации.

Разработка выполнялась с учетом ранее выбранной архитектуры веб-приложения. Серверная часть реализована на языке Python с использованием FastAPI, клиентская часть построена на HTML, CSS и JavaScript, просмотр PDF-документов осуществляется средствами PDF.js, а обмен событиями комнаты выполняется через WebSocket. Для хранения данных используется Microsoft SQL Server, а работа с базой данных на стороне Python-приложения осуществляется через библиотеку pyodbc.

В ходе реализации особое внимание уделялось разделению программной логики по уровням. Обработка HTTP-запросов и WebSocket-соединений вынесена в маршруты, бизнес-логика реализована в сервисах, а SQL-запросы к базе данных размещены в репозиториях. Такой подход соответствует проектной архитектуре и позволяет упростить сопровождение программного кода.

3.1 Создание базы данных

Перед началом реализации серверной части была создана база данных, предназначенная для хранения сведений о пользователях, ролях, комнатах видеоконференций, участниках, PDF-документах, сообщениях, аннотациях, уведомлениях и событиях системы. Структура базы данных соответствует физической модели, разработанной в подразделе 2.5.3.

База данных создавалась в среде SQL Server Management Studio. Для этого был подготовлен SQL-скрипт VideoConferenceDb_full.sql, включающий создание базы данных, таблиц, первичных и внешних ключей, ограничений

целостности и индексов. Создание базы данных в SQL Server Management Studio представлено на рисунке 26.

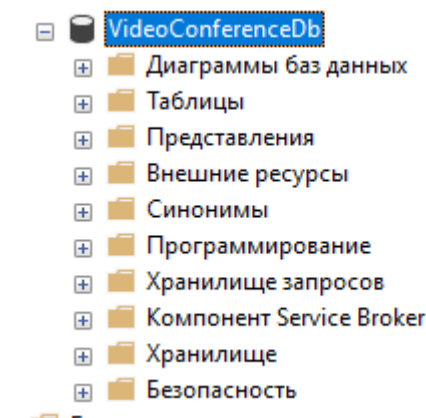


Рисунок 26 – Создание базы данных в SQL Server Management Studio

В начале SQL-скрипта выполняется проверка существования базы данных. Если база данных VideoConferenceDb отсутствует, она создается автоматически, после чего контекст выполнения запросов переключается на созданную базу. Далее формируются справочные и основные таблицы. Фрагмент SQL-скрипта создания базы данных представлен в листинге 1.

Листинг 1 – Фрагмент SQL-скрипта создания основных таблиц базы данных

```
USE VideoConferenceDb;

IF OBJECT_ID(N'dbo.Roles', N'U') IS NULL
BEGIN
    CREATE TABLE dbo.Roles (
        RoleId INT IDENTITY(1,1) NOT NULL,
        Code NVARCHAR(30) NOT NULL,
        Name NVARCHAR(100) NOT NULL,

        CONSTRAINT PK_Roles PRIMARY KEY (RoleId),
        CONSTRAINT UQ_Roles_Code UNIQUE (Code),
        CONSTRAINT CK_Roles_Code CHECK (Code IN (N'ADMIN',
N'SPEAKER', N'VIEWER'))
    );
END
GO

IF OBJECT_ID(N'dbo.Users', N'U') IS NULL
BEGIN
    CREATE TABLE dbo.Users (
        UserId INT IDENTITY(1,1) NOT NULL,
        RoleId INT NOT NULL,
```

```

Login NVARCHAR(50) NOT NULL,

PasswordHash NVARCHAR(255) NOT NULL,

LastName NVARCHAR(100) NOT NULL,
FirstName NVARCHAR(100) NOT NULL,
Patronymic NVARCHAR(100) NULL,

Email NVARCHAR(150) NULL,
Phone NVARCHAR(30) NULL,
AvatarPath NVARCHAR(500) NULL,

IsActive BIT NOT NULL CONSTRAINT DF_Users_IsActive DEFAULT
(1),

CreatedAt DATETIME2(0) NOT NULL CONSTRAINT
DF_Users_CreatedAt DEFAULT (SYSDATETIME()),
UpdatedAt DATETIME2(0) NULL,

CONSTRAINT PK_Users PRIMARY KEY (UserId),
CONSTRAINT UQ_Users_Login UNIQUE (Login),
CONSTRAINT UQ_Users_Email UNIQUE (Email),
CONSTRAINT FK_Users_Roles FOREIGN KEY (RoleId) REFERENCES
dbo.Roles (RoleId)
);
END

```

В данном фрагменте SQL-скрипта создаются основные таблицы базы данных: Roles, Users, Admins, Speakers, Viewers и Rooms. Они предназначены для хранения ролей, учетных данных пользователей, профилей по ролям и сведений о комнатах видеоконференций, включая коды подключения, статусы и параметры доступа.

После выполнения основного скрипта были созданы таблицы для участников комнат, подписок на комнаты, PDF-документов, запросов на загрузку, аннотаций, сообщений, событий, уведомлений и журнала аудита. Эти таблицы позволяют фиксировать состав участников, хранить загруженные файлы, передавать действия пользователей и сохранять служебную историю работы системы. Благодаря этому база данных поддерживает не только основные функции авторизации и комнат, но и дополнительные сценарии совместной работы с документами в режиме видеоконференции. Выполнение SQL-скрипта создания таблиц базы данных представлено на рисунке 27.

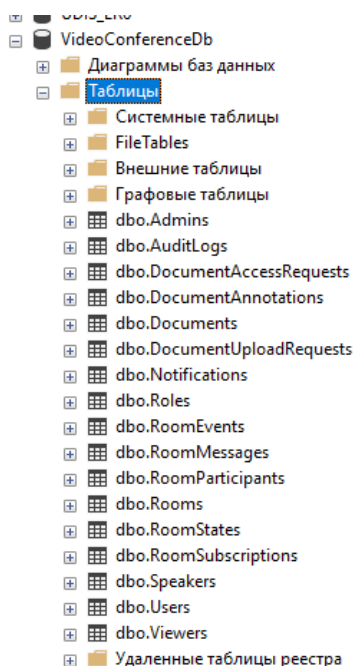


Рисунок 27 – Выполнение SQL-скрипта создания таблиц базы данных

Для контроля корректности созданной структуры была сформирована диаграмма базы данных средствами SQL Server Management Studio. Диаграмма позволяет визуально проверить наличие таблиц, первичных ключей, внешних ключей и связей между основными сущностями системы. Диаграмма базы данных представлена на рисунке 28.

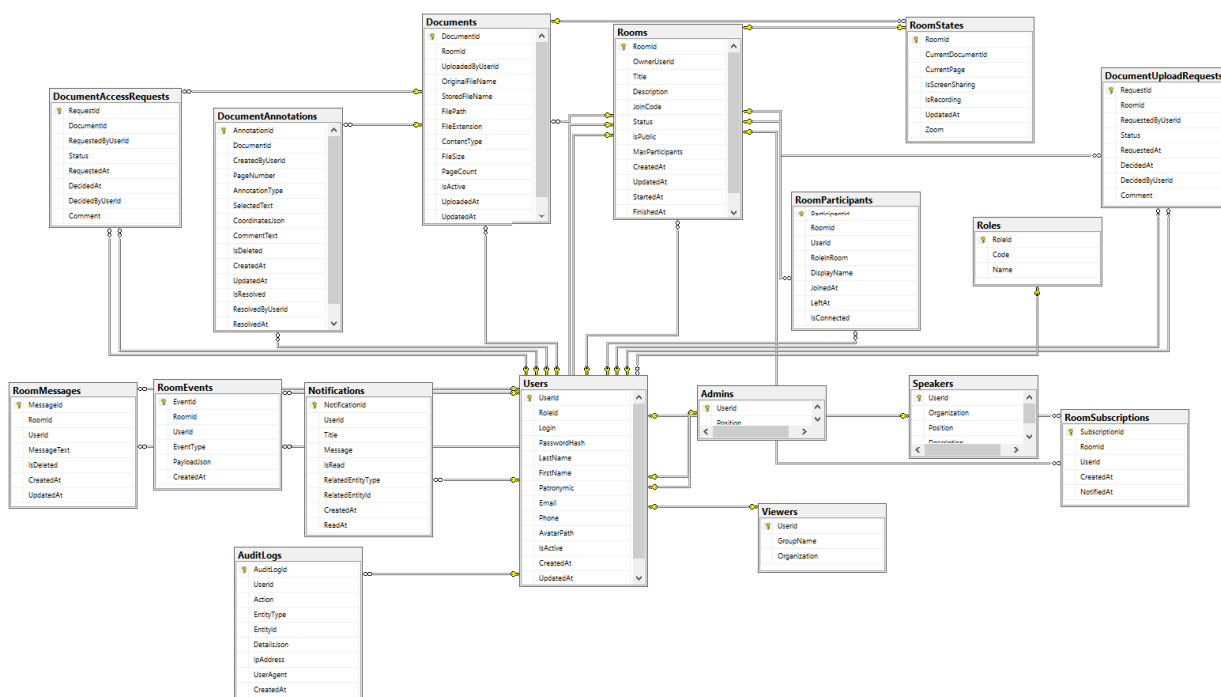


Рисунок 28 – Диаграмма базы данных в SQL Server Management Studio

Созданная структура базы данных обеспечивает хранение всех данных, необходимых для работы приложения. Таблицы пользователей и ролей используются при авторизации и разграничении доступа, таблицы комнат и участников обеспечивают проведение видеоконференций, таблицы документов и аннотаций поддерживают работу с PDF-файлами, а таблицы сообщений, уведомлений и событий позволяют фиксировать взаимодействие пользователей внутри комнаты.

Таким образом, на этапе создания базы данных была подготовлена физическая структура хранилища, соответствующая проектной модели. Полученная база данных используется серверной частью приложения для выполнения операций авторизации, управления комнатами, загрузки PDF-документов, синхронизации действий участников и информации.

3.2 Разработка структуры программного проекта

После создания базы данных была сформирована структура программного проекта. При разработке учитывалась многоуровневая архитектура, описанная в подразделе 2.3. Программный код разделен на каталоги в соответствии с назначением модулей: маршруты, сервисы, репозитории, инфраструктурные компоненты, HTML-шаблоны, статические файлы и каталог загружаемых PDF-документов.

Структура программного проекта представлена на рисунке 29.

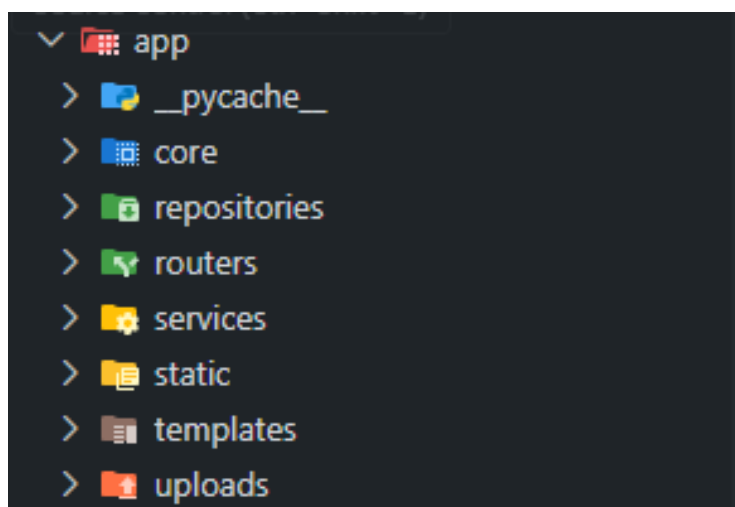


Рисунок 29 – Структура программного проекта

После формирования структуры программного проекта были определены основные каталоги и файлы приложения. Назначение ключевых элементов проекта, используемых при реализации серверной и клиентской частей, приведено в таблице 10.

Таблица 10 – Назначение основных каталогов программного проекта

Каталог или файл	Назначение
app/main.py	создание экземпляра FastAPI, подключение маршрутов, настройка статических файлов и обработка ошибок
app/core	конфигурация приложения, подключение к базе данных и функции безопасности
app/routers	маршруты авторизации, личного кабинета, комнат, PDF-документов и WebSocket-подключений
app/repositories	SQL-запросы и операции доступа к данным SQL Server
app/services	сервисная логика авторизации и рассылки WebSocket-событий
app/templates	HTML-шаблоны страниц веб-приложения
app/static	CSS- и JavaScript-файлы клиентской части
app/uploads	каталог для аватаров и загруженных PDF-документов
db/VideoConferenceDb_full.sql	SQL-скрипт создания базы данных и начальных справочных данных
config.json	параметры подключения к SQL Server и настройки запуска приложения

Основной файл приложения app/main.py содержит создание экземпляра FastAPI, подключение маршрутов, настройку статических файлов и каталогов загрузки. Каталог routers включает обработчики HTTP-запросов и WebSocket-подключений. Каталог repositories используется для выполнения SQL-запросов к базе данных, а каталог core содержит инфраструктурные файлы проекта: конфигурацию, подключение к базе данных и функции безопасности.

Каталог templates используется для хранения HTML-шаблонов страниц, а каталог static содержит CSS- и JavaScript-файлы клиентской части. Загруженные PDF-документы сохраняются в каталоге app/uploads/documents. В проекте также

предусмотрены служебные файлы для подготовки окружения и запуска серверной части, однако основной акцент реализации сделан на работе функциональных модулей системы.

Таким образом, структура проекта отражает ранее спроектированную архитектуру приложения. Разделение программного кода по каталогам позволяет выделить обработку запросов, бизнес-логику, доступ к данным, клиентский интерфейс и файловое хранилище PDF-документов.

3.3 Реализация подключения к базе данных

Для подключения серверной части к Microsoft SQL Server используется конфигурационный файл `config.json`, приведенный в листинге 2. В нем задаются параметры соединения: имя сервера, название базы данных, режим доверенного подключения, параметры тайм-аута и настройки приложения. Такой подход позволяет изменять параметры подключения без изменения исходного кода серверной части.

Листинг 2 – Фрагмент файла конфигурации подключения к базе данных

```
{
  "database": {
    "provider": "mssql",
    "server": "localhost",
    "database": "VideoConferenceDb",
    "driver": "",
    "trusted_connection": true,
    "username": "",
    "password": "",
    "trust_server_certificate": true,
    "connection_timeout": 15,
    "login_timeout": 15
  },
  "app": {
    "host": "127.0.0.1",
    "port": 8000,
    "reload": false,
    "access_log": false
  }
}
```

В данном фрагменте параметр `server` указывает на локальный сервер базы данных, а параметр `database` определяет имя заранее созданной базы `VideoConferenceDb`. Параметр `trusted_connection` используется для подключения

через учетную запись Windows. Также в конфигурации задаются хост и порт, на которых запускается веб-приложение.

На стороне Python-приложения подключение к базе данных выполняется через класс Database. Его реализация приведена в приложении А в листинге А.2. Класс отвечает за выбор доступного ODBC-драйвера, формирование строки подключения на основе параметров файла config.json, выполнение повторных попыток соединения и предоставление единого механизма работы с транзакциями.

Класс Database не создает таблицы при запуске серверной части. В соответствии с принятой организацией проекта база данных предварительно создается SQL-скриптом из каталога db, после чего приложение подключается к подготовленному хранилищу. Контекстный менеджер session обеспечивает подтверждение транзакции при успешном выполнении операций и откат изменений при возникновении исключения.

Таким образом, подключение к базе данных вынесено в отдельный инфраструктурный модуль. Это позволяет централизованно управлять параметрами соединения и использовать единый механизм доступа к SQL Server во всех репозиториях приложения.

3.4 Реализация ролевой модели и авторизации пользователей

Ролевая модель приложения основана на трех типах пользователей: администратор, ведущий и зритель. Роль определяет набор доступных действий пользователя после входа в систему. Администратор управляет учетными записями и параметрами системы, ведущий создает и проводит видеоконференции, управляет комнатой и совместной работой с PDF-документом, а зритель участвует в конференции в пределах предоставленных ему прав.

Сведения о пользователе и его роли хранятся в базе данных. После успешной проверки учетных данных они преобразуются в объект текущего пользователя CurrentUser. Объект содержит идентификатор, логин, код и

наименование роли, фамилию, имя, отчество, контактные данные и путь к аватару. Это позволяет использовать сведения об авторизованном пользователе при обработке запросов без повторного обращения к базе данных для получения основных данных профиля.

Реализация класса `CurrentUser`, функций хеширования и проверки пароля, а также механизма хранения активных пользовательских сессий приведена в приложении А в листинге А.3. Свойства объекта текущего пользователя используются для серверной проверки роли при выполнении операций. Например, создание комнаты доступно администратору и ведущему, а управление учетными записями пользователей – только администратору. Такой подход исключает выполнение управляющих действий пользователем, не обладающим необходимыми правами, даже при прямом обращении к маршруту серверной части.

Авторизация пользователя выполняется через маршрут `/login`. При отправке формы входа серверная часть устанавливает соединение с базой данных, создает сервис авторизации и передает ему введенные учетные данные для проверки. При успешной авторизации для пользователя формируется токен сессии, который сохраняется в cookie браузера. Фрагмент маршрута авторизации представлен в листинге 3.

Листинг 3 – Маршрут авторизации пользователя

```
@router.post('/login')
def login(request: Request, login: str = Form(...), password: str
= Form(...)):
    with db.session() as conn:
        service = AuthService(UserRepository(conn))
        user = service.authenticate(login.strip(), password)
    if not user:
        return templates.TemplateResponse('login.html',
{'request': request, 'error': 'Неверный логин или
        пароль'}, status_code=401)
    token = create_session(user)
    response = RedirectResponse('/dashboard', status_code=303)
    response.set_cookie(config.SESSION_COOKIE_NAME, token,
httponly=True, samesite='lax')
    return response
```

После входа в систему пользователь перенаправляется в личный кабинет.

В дальнейшем доступность отдельных действий определяется ролью пользователя. Например, создание комнаты разрешается администратору и ведущему, а управление пользователями доступно только администратору. Такой подход обеспечивает разграничение прав доступа на уровне серверной части и не зависит только от отображения элементов интерфейса.

Таким образом, реализация ролевой модели объединяет данные из базы, объект текущего пользователя, серверные проверки доступа и пользовательскую сессию. Это позволяет ограничивать выполнение управляющих действий в соответствии с назначенной ролью.

3.5 Реализация работы с комнатами видеоконференции

Комната видеоконференции является основным рабочим объектом системы. В ее рамках объединяются участники, сообщения чата, состояние демонстрируемого PDF-документа, аннотации и события синхронизации. Именно комната определяет контекст совместной работы: кто подключен к конференции, какой документ рассматривается в текущий момент и какие действия доступны пользователям в зависимости от назначенной роли и состояния комнаты.

Создание комнаты выполняется пользователем с ролью администратора или ведущего. При создании указываются название комнаты, описание, максимальное количество участников и начальный статус. Серверная часть получает данные формы, определяет текущего пользователя и проверяет его права на выполнение операции. При отсутствии авторизации пользователь перенаправляется на страницу входа, а при недостаточном уровне прав создание комнаты не выполняется. Реализация маршрутов создания и обработки комнат приведена в приложении А в листинге А.7.

После успешной проверки введенные данные передаются в репозиторий комнат, где создается запись в базе данных. Такой подход разделяет обработку HTTP-запроса, проверку прав доступа и операции с хранилищем данных. В результате маршрут отвечает за прием запроса и формирование ответа

пользователю, а репозиторий обеспечивает сохранение сведений о комнате и связанных с ней объектах.

Подключение к комнате выполняется по коду подключения. Если комната находится в активном состоянии, пользователь переходит на страницу конференции и получает доступ к ее функциям. Для запланированной комнаты пользователь добавляется в список ожидания, а при обращении к завершенной или отмененной комнате система выводит сообщение о невозможности подключения. Такая логика соответствует жизненному циклу комнаты, разработанному на этапе проектирования, и исключает выполнение действий в недопустимом состоянии конференции.

При открытии страницы комнаты серверная часть получает сведения о текущем состоянии комнаты, подключенных участниках, сообщениях чата, доступных документах, событиях синхронизации и ожидающих запросах на загрузку PDF-документов. Эти данные используются для первоначального формирования интерфейса до установления постоянного WebSocket-соединения. Для дальнейшей работы комнаты также формируется короткоживущий токен, позволяющий связать действия пользователя в интерфейсе с его участием в конкретной конференции при выполнении WebSocket- и AJAX-запросов.

Таким образом, серверная реализация комнат обеспечивает создание видеоконференции, подключение и учет участников, контроль статусов, разграничение доступа и подготовку данных для отображения общего рабочего пространства. Благодаря этому функции видеосвязи, работы с PDF-документом, чата и синхронизации выполняются в рамках единой комнаты и используют согласованное состояние данных.

3.6 Реализация загрузки PDF-документов и синхронизации действий

Одной из ключевых функций разрабатываемой системы является загрузка и назначение PDF-документа для совместного просмотра в комнате видеоконференции. Документ становится общим рабочим материалом

участников: ведущий управляет его демонстрацией, а зрители получают возможность просматривать одни и те же страницы и видеть результаты синхронизированных действий. Поэтому операция загрузки должна выполняться с учетом состояния комнаты, прав пользователя и требований к безопасности передаваемых файлов.

Загрузка PDF-документа реализована отдельным серверным маршрутом. Перед сохранением файла система определяет текущего пользователя, проверяет активность комнаты и наличие у него права на загрузку. Такая проверка позволяет исключить изменение общего документа неавторизованным пользователем, а также предотвратить загрузку файла в запланированную, завершенную или отмененную комнату. Реализация проверки и загрузки PDF-документа представлена в листинге 4.

Листинг 4 – Проверка и загрузка PDF-документа

```
@router.post('/rooms/{code}/documents')
async def upload_document(request: Request, code: str, pdf_file:
UploadFile = File(...)):
    user = _resolve_room_user(request)
    if not user:
        return JSONResponse({'error': 'unauthorized'},
status_code=401)
    code = code.upper()
    with db.session() as conn:
        room_repo = RoomRepository(conn)
        doc_repo = DocumentRepository(conn)
        room = room_repo.get_room_by_code(code)
        if not room or room['Status'] != 'ACTIVE':
            return JSONResponse({'error': 'room_not_active'},
status_code=403)
        if not doc_repo.can_upload_document(room, user):
            return JSONResponse({'error':
'upload_permission_required'}, status_code=403)
        original_name = Path(pdf_file.filename or
'document.pdf').name
        if not original_name.lower().endswith('.pdf'):
            return JSONResponse({'error': 'not_pdf'},
status_code=400)
        data = await pdf_file.read()
        if not data.startswith(b'%PDF'):
            return JSONResponse({'error': 'not_pdf'},
status_code=400)
        stored_name = f'{secrets.token_urlsafe(12)}.pdf'
        target = config.UPLOAD_DIR / 'documents' / stored_name
```

```

        target.parent.mkdir(parents=True, exist_ok=True)
        target.write_bytes(data)
        document = doc_repo.create_document(int(room['RoomId']),
user.id, original_name, stored_name,
            f'documents/{stored_name}', len(data))
        room_repo.set_current_document(int(room['RoomId']),
int(document['DocumentId']), user.id)
        room_repo.set_current_page(int(room['RoomId']), 1, None,
user.id)
        document_payload = {
            'id': int(document['DocumentId']),
            'fileName': document['OriginalFileName'],
            'url': f"/documents/{int(document['DocumentId'])}/file",
            'page': 1,
        }
        await signaling_service.broadcast(code, {'type': 'document-
updated', 'document': document_payload})
        await signaling_service.broadcast(code, {'type': 'room-event',
'event': {'EventType': f"Загружен PDF:
            {document['OriginalFileName']}", 'CreatedAt': '',
'PayloadJson': '{}'}})
        return {'document': document_payload}

```

В листинге 4 представлен маршрут `/rooms/{code}/documents`, предназначенный для загрузки и назначения PDF-документа для конкретной комнаты видеоконференции. Маршрут принимает код комнаты и файл, переданный пользователем через форму загрузки. Его назначение заключается не только в сохранении файла, но и в безопасном изменении общего документа, с которым работают участники конференции.

В начале обработки определяется текущий пользователь. При отсутствии авторизации сервер возвращает ответ с ошибкой, поэтому загрузка не может быть выполнена анонимным пользователем. Далее по коду подключения находится комната, проверяется ее существование и активный статус. Загрузка разрешается только в активную комнату, поскольку в запланированной, завершенной или отмененной конференции изменение общего документа недопустимо.

После проверки состояния комнаты вызывается метод `can_upload_document`, определяющий наличие у пользователя права на загрузку PDF-документа. Это позволяет учитывать роль пользователя и решение ведущего по ранее созданному запросу на загрузку. Таким образом, зритель не

может самостоятельно заменить общий документ без предоставленного разрешения, а ведущий и администратор получают возможность управлять демонстрируемыми материалами в пределах своих полномочий.

Затем выполняется проверка имени файла и его содержимого. Сначала контролируется расширение `.pdf`, после чего сервер проверяет сигнатуру файла `%PDF`. Двойная проверка предотвращает сохранение файлов другого формата, переименованных в PDF-документы. После успешной проверки файлу назначается случайное имя, исключающее совпадение имен в каталоге хранения, а исходное имя сохраняется отдельно для отображения пользователям в интерфейсе.

Сведения о загруженном документе, включая исходное и сохраненное имя, путь хранения, размер файла, идентификатор комнаты и пользователя, фиксируются в базе данных. Затем документ назначается текущим документом комнаты, а начальная отображаемая страница устанавливается равной единице. В результате после загрузки все участники получают единое исходное состояние для совместного просмотра PDF-файла.

После завершения операции сервер формирует объект с данными нового документа и передает событие `document-updated` всем подключенным участникам комнаты. Клиентская часть получает сведения об идентификаторе, имени, адресе файла и текущей странице, после чего обновляет область просмотра без перезагрузки страницы. Дополнительно в журнал событий комнаты добавляется информация о загрузке документа, что позволяет фиксировать существенные действия пользователей в ходе видеоконференции.

Передача событий между участниками комнаты выполняется через WebSocket-соединение. При подключении сервер проверяет пользователя и активность комнаты, регистрирует участника, передает начальное состояние конференции и обеспечивает дальнейший обмен событиями. Полная реализация маршрутов работы с комнатой, включая обработчик WebSocket-соединения, приведена в приложении А в листинге А.7.

Через WebSocket передаются служебные сообщения WebRTC,

необходимые для установления видеосвязи, сообщения чата, сведения о подключении и отключении участников, изменение текущей страницы PDF-документа, события указки и режима взаимодействия с документом. При получении события смены страницы сервер сохраняет новое значение в базе данных и рассылает его остальным участникам комнаты. Благодаря этому ведущий управляет общим просмотром документа, а зрители видят синхронно выбранную им страницу.

Для работы с аннотациями используется репозиторий DocumentRepository, полная реализация которого приведена в приложении А в листинге А.6. Репозиторий обеспечивает создание, получение и преобразование аннотаций PDF-документа для передачи в клиентскую часть. Каждая аннотация содержит тип, номер страницы, координаты размещения, текст комментария и сведения о пользователе, создавшем ее.

Для зрителя предусмотрен тип аннотации QUESTION, используемый для размещения вопроса на странице документа. Для ведущего доступны метки, выделения и комментарии, позволяющие пояснять содержание демонстрируемого материала. После создания аннотация сохраняется в базе данных и передается участникам комнаты как событие синхронизации, вследствие чего изменения становятся доступными другим пользователям в режиме реального времени.

Таким образом, реализация работы с PDF-документом включает контролируемую загрузку файлов, хранение сведений о документах, обновление состояния комнаты, синхронизацию страниц, работу с аннотациями и передачу событий участникам видеоконференции. Листинг 4 отражает центральный серверный маршрут данного процесса, а детальная реализация остальных операций вынесена в приложение А.

3.7 Реализация клиентской части веб-приложения

Клиентская часть веб-приложения реализована с использованием HTML-шаблонов, CSS-оформления и JavaScript-логики. HTML-шаблоны формируют

страницы входа, личных кабинетов, профиля, списка комнат, формы создания комнаты и интерфейса видеоконференции. CSS-файлы отвечают за внешний вид, расположение блоков и адаптацию интерфейса под разные размеры экрана.

JavaScript-модули получают данные о комнате и устанавливают соединение с серверной частью. Они обеспечивают подключение к WebSocket, обмен сообщениями WebRTC.

Страница входа в веб-приложение представлена на рисунке 30.

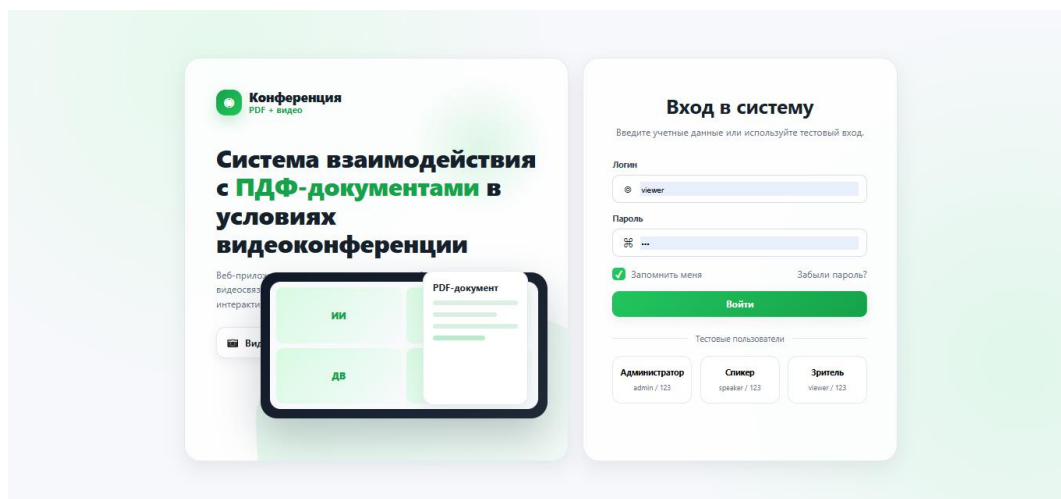


Рисунок 30 – Страница входа в веб-приложение

После успешной авторизации пользователь попадает в кабинет, состав которого зависит от роли. Для администратора отображаются сводные показатели, список пользователей, управление комнатами и запросами. Личный кабинет администратора представлен на рисунке 31.

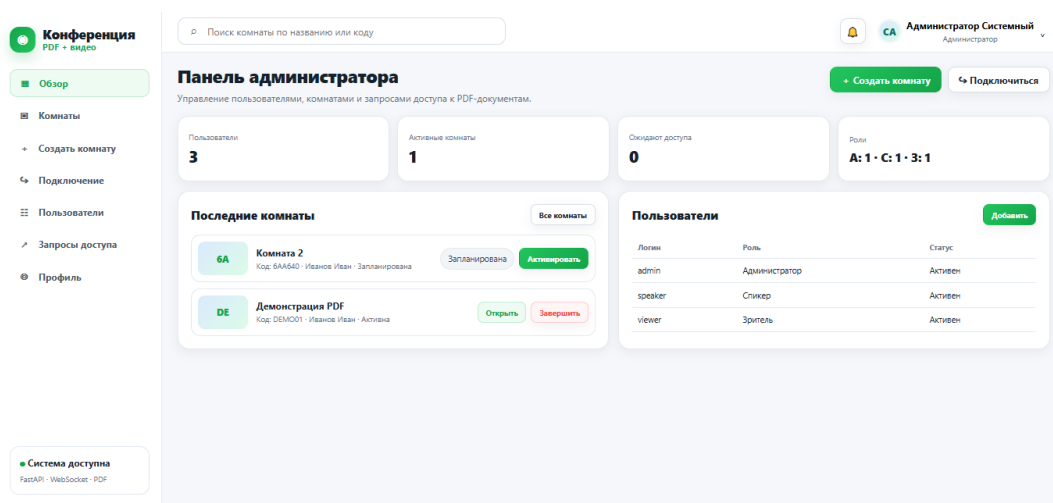


Рисунок 31 – Личный кабинет администратора

Для ведущего основными функциями являются создание комнаты, просмотр своих комнат, подключение к комнате и обработка запросов на загрузку PDF-документа. Личный кабинет ведущего представлен на рисунке 32.

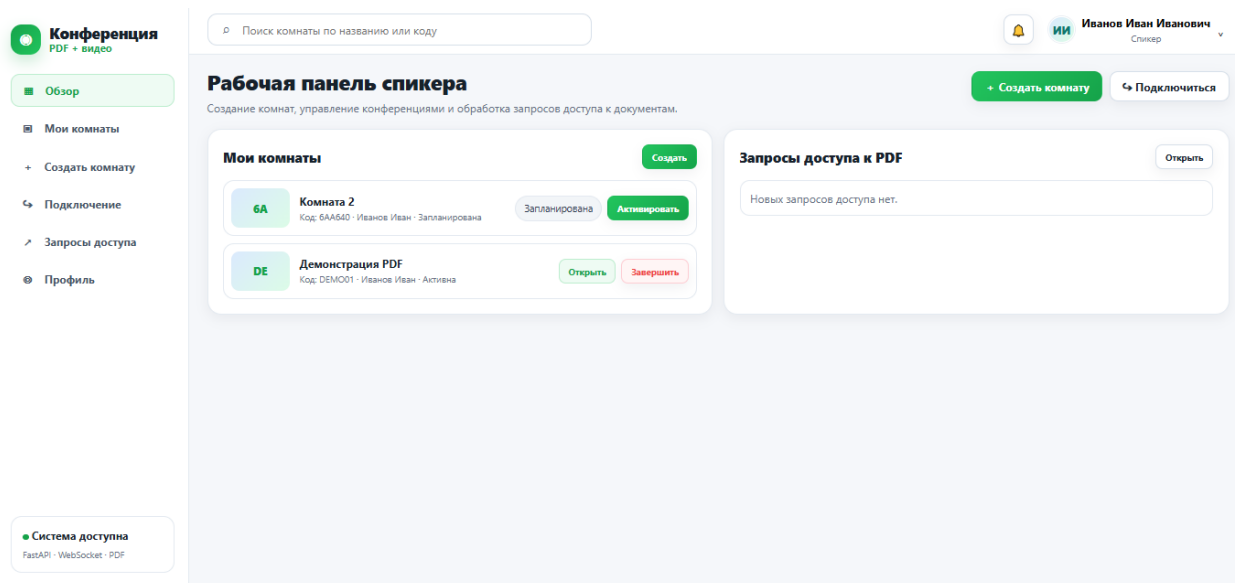


Рисунок 32 – Личный кабинет ведущего

Для зрителя интерфейс ориентирован на подключение к комнате, участие в видеоконференции и просмотр синхронизированного PDF-документа. Личный кабинет зрителя представлен на рисунке 33.

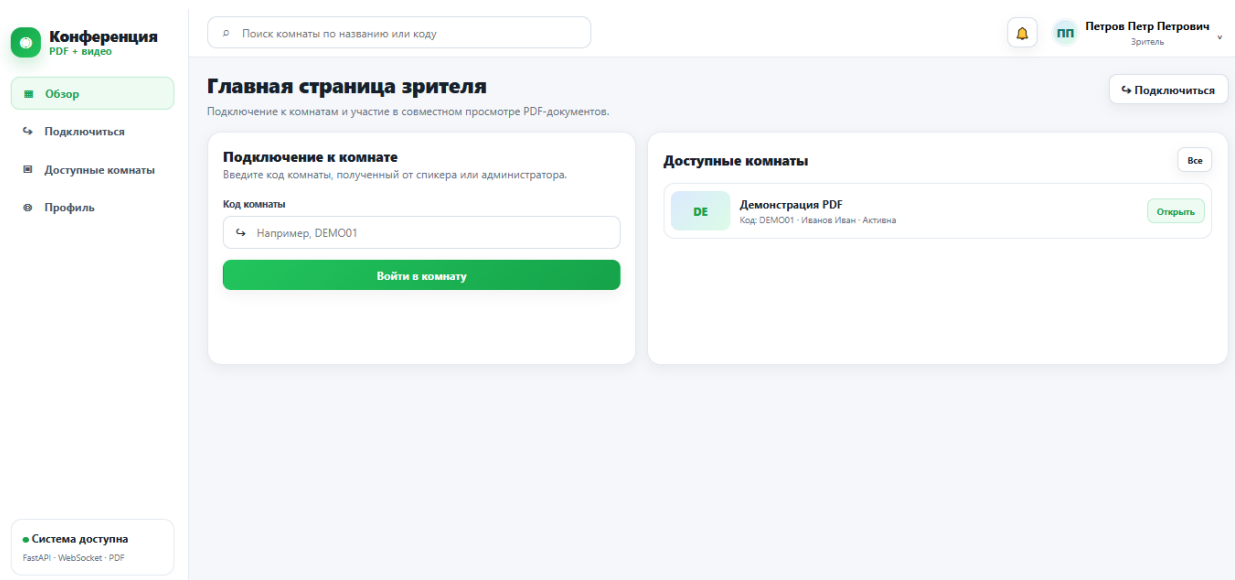


Рисунок 33 – Личный кабинет зрителя

Основная рабочая страница системы – интерфейс комнаты видеоконференции. Со стороны ведущего в комнате доступны загрузка PDF-

документа, переключение страниц, указка, аннотации, чат, список участников и обработка запросов зрителей. Интерфейс комнаты со стороны ведущего представлен на рисунке 34.

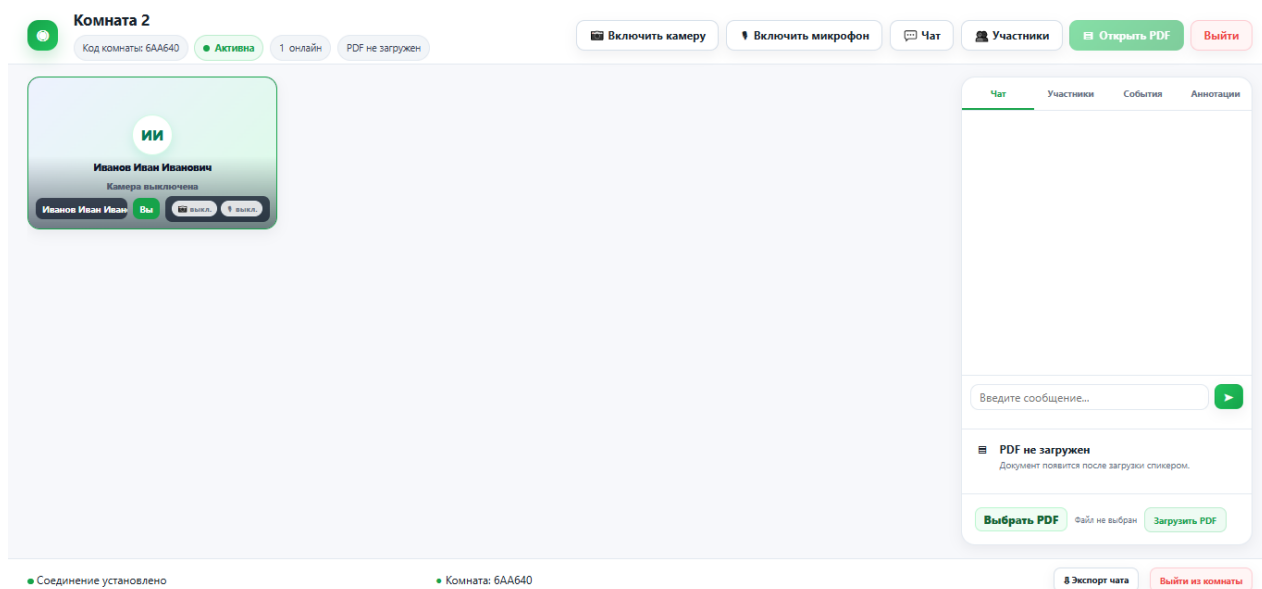


Рисунок 34 – Интерфейс комнаты видеоконференции со стороны ведущего

Со стороны зрителя интерфейс комнаты содержит видеосвязь, просмотр текущего документа, чат, список участников, возможность перехода к странице ведущего и отправку запроса на загрузку собственного PDF-документа. Интерфейс комнаты со стороны зрителя представлен на рисунке 35.

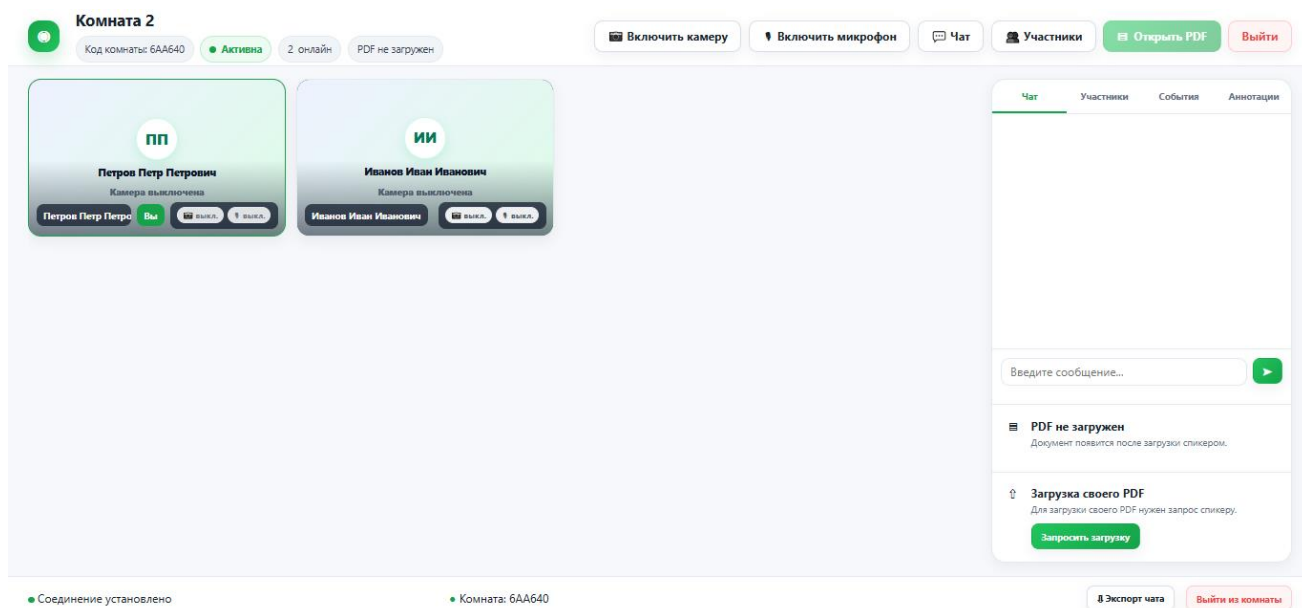


Рисунок 35 – Интерфейс комнаты видеоконференции со стороны зрителя

Для отображения PDF-документа на клиентской стороне используется библиотека PDF.js. Основные фрагменты клиентской логики работы с PDF-документом приведены в приложении А в листинге А.8. JavaScript-модуль получает ссылку на текущий документ комнаты, загружает библиотеку в браузере и формирует область просмотра на основе элемента canvas.

Функция `loadPdf` отвечает за получение PDF-документа по ссылке, переданной серверной частью. После успешной загрузки она создает элемент canvas и слой аннотаций, который располагается поверх отображаемой страницы. Это позволяет выводить PDF-документ непосредственно в интерфейсе видеоконференции и отображать связанные с ним действия участников без открытия отдельного приложения.

Функция `renderPdfPage` получает выбранную страницу документа, определяет ее размеры с учетом установленного масштаба и формирует изображение страницы на canvas. Одновременно обновляются сведения о текущей странице, общем количестве страниц документа и масштабе просмотра. После отрисовки страницы клиентская часть запрашивает аннотации, относящиеся к ней, и размещает их в соответствующих координатах поверх документа.

Переключение страниц выполняется функцией `changePage`. При изменении страницы ведущим новое значение передается через WebSocket-соединение и дополнительно сохраняется серверной частью. Благодаря этому подключившиеся пользователи получают актуальное состояние общего просмотра. Зритель также может перейти на другую страницу локально, однако такое действие не изменяет страницу, демонстрируемую ведущим. Для возврата к синхронному просмотру предусмотрена отдельная команда перехода к странице ведущего.

Таким образом, клиентская реализация обеспечивает загрузку PDF-документа, его постраничное отображение, изменение масштаба, вывод аннотаций и синхронизацию действий участников. Отображение PDF-документа и аннотаций в интерфейсе комнаты представлено на рисунке 36.

Отображение PDF-документа и аннотаций в комнате представлено на рисунке 36.

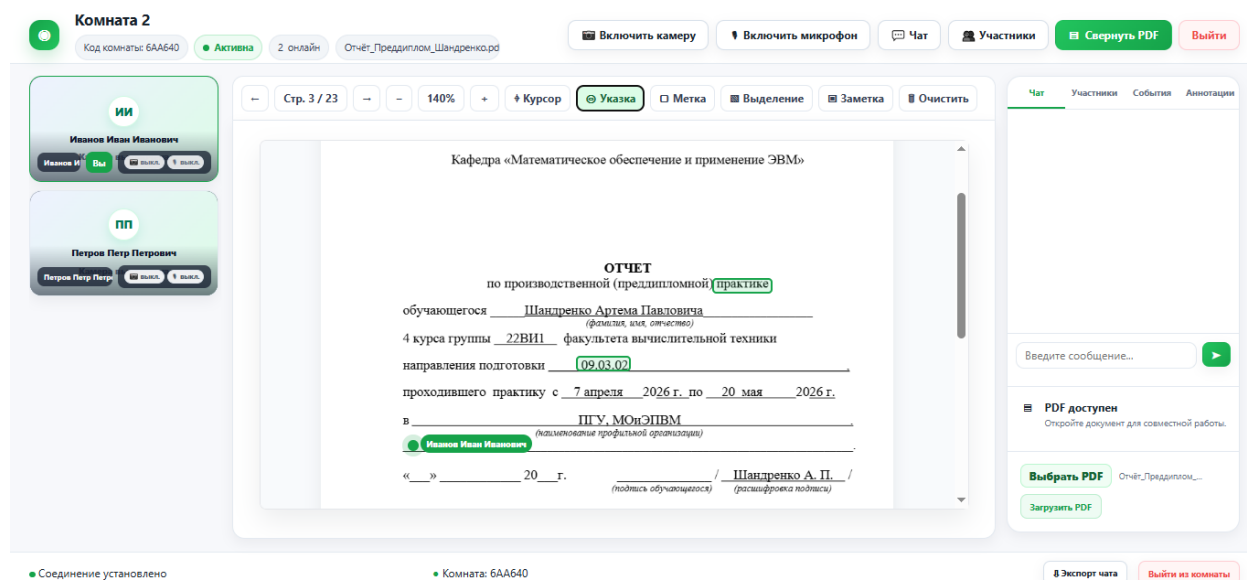


Рисунок 36 – Отображение PDF-документа и аннотаций в интерфейсе комнаты

Для получения событий комнаты клиент открывает WebSocket-соединение. Через него клиент получает сообщения о подключении участников, изменении страницы PDF-документа, новых аннотациях, обновлении документа, сообщениях чата и служебных данных WebRTC. Фрагмент обработки клиентских WebSocket-событий представлен в листинге 5.

Листинг 5 – Обработка WebSocket-событий на клиентской стороне

```
function connectSocket() {
    clearTimeout(reconnectTimer);

    const params = new URLSearchParams({ peer_id: peerId });
    if (ctx.roomToken) params.set('room_token', ctx.roomToken);

    socket = new WebSocket(
        `${wsScheme}://${location.host}/ws/rooms/${ctx.roomCode}?${params}`
    );

    socket.onopen = () => setConnection('Соединение установлено', true);
    socket.onerror = () => setConnection('Ошибка соединения');
    socket.onclose = () => {
        setConnection('Соединение закрыто');
        reconnectTimer = setTimeout(connectSocket, 1500);
    };
}
```

```

};

socket.onmessage = async ({ data }) => {
  const message = JSON.parse(data);

  if (message.participants)
    renderParticipants(message.participants);

  if (message.type === 'connected') {
    addEvent('Подключение к комнате выполнено');
    if (message.document) await
    applyDocumentUpdate(message.document);
  }

  if (message.type === 'peer-joined' && state.localStream) {
    const meta = normalizeParticipant(message.peer || {});
    if (meta.peerId && meta.peerId !== peerId) {
      callPeer(meta).catch(console.warn);
    }
  }

  if (message.type === 'peer-left')
    removeRemoteTile(message.peerId);
  if (message.type === 'offer') await handleOffer(message);
  if (message.type === 'answer') await
  handleAnswer(message);
  if (message.type === 'ice-candidate') await
  handleIce(message);

  if (message.type === 'pdf-page') {
    state.page = Number(message.page || 1);
    updatePdfLabels();
    if (state.pdfOpen) await renderPdfPage(state.page);
  }

  if (message.type === 'pdf-pointer') showPointer(message);
  if (message.type === 'chat') appendChat(message.message);
  if (message.type === 'document-updated') {
    await applyDocumentUpdate(message.document);
    toast('PDF обновлен');
  }
};
}

```

В обработчике WebSocket-сообщений клиент реагирует на события `connected`, `peer-joined`, `offer`, `answer`, `ice-candidate`, `chat`, `pdf-page`, `annotation-created` и `document-updated`. При изменении страницы ведущим `speakerPage` обновляется у всех участников. При создании аннотации она сразу отрисовывается поверх текущей страницы.

Таким образом, клиентская часть объединяет видеосвязь, синхронизированный просмотр PDF-документа, чат, аннотации и работу с участниками комнаты. Использование JavaScript, WebSocket, WebRTC и PDF.js позволяет реализовать взаимодействие в режиме реального времени без полной перезагрузки страницы.

В результате разработки была реализована информационная система, соответствующая проектным решениям второго раздела. Была создана база данных SQL Server, разработана структура программного проекта, реализованы подключение к базе данных, авторизация, ролевая модель, управление комнатами, загрузка PDF-документов, синхронизация страниц, работа с аннотациями и клиентский интерфейс. Полученная реализация является основой для дальнейшего и оценки качества программного обеспечения.

4 Тестирование и оценка качества информационной системы

Четвертый раздел посвящен проверке работоспособности разработанной информационной системы для синхронизированного взаимодействия с PDF-документами в условиях видеоконференции. Тестирование выполняется после реализации основных функций серверной и клиентской частей и направлено на подтверждение того, что система корректно обрабатывает работу пользователей с комнатой, PDF-документом, запросами на загрузку, синхронизацией страницы и аннотациями.

В соответствии с методическими рекомендациями по выпускной квалификационной работе раздел контроля качества должен включать проверку программных средств, описание тестовых сценариев и оценку качества кода по метрикам [20]. Поэтому в данном разделе рассматриваются тестирование методом черного ящика, модульное тестирование и расчет метрик качества для выбранного фрагмента серверной части. Фактические результаты выполнения проверок вынесены на рисунки В.1 - В.8, а в тексте раздела приведены сценарии, ожидаемые результаты и интерпретация полученных показателей.

4.1 Цель, объект и условия тестирования

Целью тестирования является проверка соответствия реализованной системы требованиям, сформулированным в предыдущих разделах пояснительной записки. Основное внимание уделяется не запуску приложения как таковому, а функциям, связанным с работой PDF-документа в комнате видеоконференции: загрузке файла, проверке прав пользователя, синхронизации текущей страницы, отображению аннотаций и обработке событий комнаты.

Объектом тестирования является веб-приложение, серверная часть которого реализована на Python с использованием FastAPI, а клиентская часть использует JavaScript, WebSocket, WebRTC и PDF.js. В качестве центрального проверяемого сценария выбран процесс работы с PDF-документом, условия проведения тестирования приложения представлено в таблице 11.

Таблица 11 – Условия проведения тестирования

Параметр	Значение
Объект тестирования	Информационная система для работы с PDF-документами в условиях видеоконференции
Основные роли	Администратор, ведущий, зритель
Проверяемые функции	Авторизация, подключение к комнате, загрузка PDF, запрос на загрузку, синхронизация страницы, аннотации, чат
Тестируемые уровни	Пользовательские сценарии, серверные маршруты, репозитории, обработка WebSocket-событий
Основной проверяемый модуль	Маршруты rooms_router.py и функции, связанные с PDF-документом
Критерии успешности	Корректный результат при допустимых данных, контролируемый отказ при ошибочных данных, отсутствие нарушения ролевых ограничений

Указанные условия позволяют ограничить объем тестирования наиболее значимыми функциями системы. Такой подход соответствует замечанию о необходимости не увеличивать количество однотипных тестов, но при этом сохраняет проверку ключевых сценариев, связанных с темой выпускной квалификационной работы.

4.2 Теоретические основы контроля качества

Качество программного обеспечения рассматривается как совокупность свойств, определяющих способность программы выполнять заданные функции в установленных условиях эксплуатации. Для разрабатываемой системы важны функциональная пригодность, надежность, защищенность, удобство использования и сопровождаемость, так как пользователь должен не только подключиться к комнате, но и корректно работать с общим PDF-документом.

Тестирование относится к динамическим методам верификации, поскольку проверка выполняется путем запуска программы или отдельных ее компонентов. Верификация позволяет установить соответствие программной реализации требованиям, а валидация показывает, насколько полученное поведение соответствует ожидаемому пользовательскому сценарию [21].

Тестирование методом черного ящика используется для проверки внешнего поведения системы без анализа внутренней структуры программы. Модульное тестирование позволяет оценить отдельные функции, маршруты и сервисы, а метрики кода применяются для количественной оценки сложности реализации [21]. Использование таблиц решений помогает сократить набор тестов при сохранении проверки существенных вариантов поведения [22]. Характеристики качества, значимые для тестируемой системы, приведены в таблице 12.

Таблица 12 – Характеристики качества, значимые для тестируемой системы

Характеристика	Проявление в системе	Способ проверки
Функциональная пригодность	PDF-документ загружается, открывается и синхронизируется между участниками комнаты	Тестирование методом черного ящика и таблица решений
Надежность	Система возвращает контролируемые ошибки при отсутствии авторизации, неверном статусе комнаты и некорректном файле	Модульные тесты и проверка ошибочных ветвей
Защищенность	Загрузка PDF и управление страницей доступны только пользователям с соответствующими правами	Проверка ролевых ограничений
Удобство использования	Пользователь получает понятные сообщения об ошибках и видит актуальное состояние документа	Функциональная проверка интерфейса
Сопровождаемость	Фрагмент загрузки PDF имеет измеримую сложность и понятные независимые маршруты	Метрики Холстеда, Маккейба и Чепина

Таким образом, выбранные характеристики качества связывают общие требования к программному обеспечению с конкретными функциями разработанной системы. Дальнейшие проверки сосредоточены на тех участках, где ошибка может привести к нарушению доступа к документу или рассинхронизации участников комнаты.

4.3 Тестирование методом черного ящика

Тестирование методом черного ящика выполняется на основе таблицы решений. В рассматриваемой системе поведение пользователя зависит от авторизации, состояния комнаты.. Модель причин и следствий для данного сценария представлена на рисунке 37.



Рисунок 37 – Модель причин и следствий для проверки работы с PDF-документом

На рисунке 37 причины обозначены буквой «П», а следствия – буквой «С».. Таблица решений для действий по данной схеме с PDF-документом приведена в таблице 13.

Таблица 13 – Таблица решений для доступа к действиям с PDF-документом

Условие/следствие	R1	R2	R3	R4	R5	R6	R7
П1 пользователь авторизован	0	1	1	1	1	1	1
П2 комната существует	1	0	1	1	1	1	1
П3 комната активна	1	1	0	1	1	1	1

Продолжение таблицы 13

П4 пользователь является администратором или владельцем комнаты	0	0	0	0	1	0	1
П5 пользователь имеет подтвержденный запрос на загрузку	0	0	0	0	0	1	0
П6 файл имеет расширение PDF	1	1	1	1	1	1	0
П7 содержимое файла имеет сигнатуру PDF	1	1	1	1	1	1	0
С1 разрешить загрузку PDF	0	0	0	0	1	1	0
С2 разрешить управление страницами	0	0	0	0	1	0	0
С3 создать или ожидать запрос на загрузку	0	0	0	1	0	0	0
С4 отказать и вывести сообщение	1	1	1	0	0	0	1
С5 передать событие участникам комнаты	0	0	0	0	1	1	0

В таблице 13 значение 1 означает выполнение условия или следствия, а значение 0 – его отсутствие. Правила R1 – R7 охватывают основные варианты поведения. Тест-кейсы по проверке методом черного ящика приведены в таблице 14.

Таблица 14 – Тест-кейсы по проверке методом черного ящика

Номер	Правило	Входные условия	Ожидаемый результат	Ссылка на результат
1	R1	Неавторизованный пользователь открывает комнату или обращается к загрузке PDF	Переход на страницу входа или ответ unauthorized	рисунок В.1
2	R2	Авторизованный пользователь вводит код несуществующей комнаты	Отображается сообщение о том, что комната не найдена	рисунок В.2

Продолжение таблицы 14

Номер	Правило	Входные условия	Ожидаемый результат	Ссылка на результат
3	R3	Пользователь обращается к комнате со статусом PLANNED или FINISHED	Подключение к рабочему интерфейсу не выполняется	рисунок В.3
4	R4	Зритель в активной комнате не имеет разрешения на загрузку PDF	Доступна отправка запроса на загрузку, прямая загрузка отклоняется	рисунок В.4
5	R5	Ведущий загружает корректный PDF-файл в активной комнате	Файл сохраняется, становится текущим документом, участники получают событие document-updated	рисунок В.5
6	R6	Зритель с подтвержденным запросом загружает корректный PDF-файл	Файл сохраняется, документ становится доступен участникам, управление страницей остается у ведущего	рисунок В.6
7	R7	Пользователь выбирает файл с неверным расширением или неверной сигнатурой	Загрузка отклоняется с ошибкой not_pdf	рисунок В.7

По результатам проверки методом черного ящика подтверждается, что работа с PDF-документом выполняется с учетом роли пользователя, состояния комнаты и корректности файла.

4.4 Модульное тестирование

Модульное тестирование направлено на проверку отдельных функций серверной части без прохождения полного пользовательского сценария через интерфейс. Оно позволяет изолированно оценить обработку входных данных, состояния комнаты и прав текущего пользователя, что особенно важно для операций работы с PDF-документом [21]. Перечень проверяемых модулей, функций и ожидаемых результатов приведен в таблице 15.

Для автоматизации проверок используется фреймворк `pytest` [23]. Тесты выполняются в отдельном каталоге `tests`, а обращения к хранилищу, репозиториям и сервису рассылки заменяются тестовыми объектами. Результат выполнения набора тестов приведен в приложении В на рисунке В.8.

Таблица 15 – Функциональная спецификация модульных тестов

Номер	Модуль/функция	Проверка	Ожидаемый результат	Ссылка на результат
1	<code>resolve_room_user</code>	Получение пользователя по токену страницы комнаты	Возвращается текущий пользователь или <code>None</code>	рисунок В.8
2	<code>can_upload_document</code>	Проверка права загрузки PDF для администратора, владельца и зрителя	Администратор и владелец получают <code>True</code> , зритель без <code>APPROVED</code> получает <code>False</code>	рисунок В.8
3	<code>upload_document</code>	Загрузка корректного PDF ведущим	Возвращается объект <code>document</code> и формируется <code>document-updated</code>	рисунок В.8
4	<code>upload_document</code>	Передача файла без расширения <code>.pdf</code>	Возвращается ошибка <code>not_pdf</code>	рисунок В.8
5	<code>set_page</code>	Попытка смены страницы зрителем	Возвращается ошибка <code>forbidden</code>	рисунок В.8
6	<code>create_annotation</code>	Создание вопроса зрителем	Создается аннотация типа <code>QUESTION</code>	рисунок В.8
7	<code>resolve_question</code>	Обработка вопроса ведущим	Аннотация получает признак обработки, участникам передается <code>question-resolved</code>	рисунок В.8

Таблица 15 определяет состав автоматизированных проверок. Набор охватывает получение текущего пользователя по токену страницы комнаты, контроль права загрузки PDF-документа, успешную и ошибочную обработку

файла, изменение страницы, создание вопроса зрителем и обработку вопроса ведущим. Таким образом, тестируются не только положительные сценарии, но и условия, при которых серверная часть обязана вернуть отказ и не изменить общее состояние комнаты.

Подготовка окружения включает установку pytest и расширения pytest-asyncio, для проверки маршрутов FastAPI. Запуск выполняется из каталога проекта командой `python -m pytest tests/test_rooms.py -v`. Ключ `-v` выводит имя каждой проверки и ее статус. Подключение библиотек и команда запуска автоматизированных тестов приведены в листинге 6.

Листинг 6 – Подключение pytest и запуск автоматизированных тестов

```
pytest==9.0.2
pytest-asyncio==1.3.0

python -m pip install -r tests/requirements-test.txt

python -m pytest tests/test_rooms.py -v
```

Тестовый модуль содержит функции формирования объекта пользователя, HTTP-запроса и контекстный менеджер `'StubDb'`. Он заменяет подключение к SQL Server тестовой реализацией, поэтому проверки не зависят от конкретного экземпляра базы данных. Реальные репозитории и сервис рассылки временно подменяются объектами с заранее определенными данными. Проверка получения пользователя и права загрузки PDF-документа приведена в листинге 7.

Листинг 7 – Проверка получения пользователя и права загрузки PDF-документа

```
def
test_resolve_room_user_returns_user_for_page_token(monkeypatch,
host):
    monkeypatch.setattr(rooms_router,
'get_user_by_room_page_token',
lambda token: host if token == 'token-1'
else None)
    monkeypatch.setattr(rooms_router, 'get_current_user', lambda
request: None)

    assert rooms_router._resolve_room_user(
make_request(query='room_token=token-1')) is host
    assert rooms_router._resolve_room_user(make_request()) is None
```

```

@pytest.mark.parametrize(('user', 'request_status', 'expected'), [
    (make_user(1, 'ADMIN'), None, True),
    (make_user(2, 'SPEAKER'), None, True),
    (make_user(3, 'VIEWER'), None, False),
    (make_user(3, 'VIEWER'), 'APPROVED', True),
])
def test_can_upload_document_checks_role_and_request_status(
    user, request_status, expected):
    room = {'RoomId': 5, 'OwnerUserId': 2}
    repo = DocumentRepository(ApprovalConnection(request_status))
    assert repo.can_upload_document(room, user) is expected

```

В листинге 7 представлены две базовые проверки. Первая подтверждает работу функции `resolve_room_user`, которая сначала анализирует токен страницы комнаты, а затем при его отсутствии использует данные обычной пользовательской сессии. Вторая проверка параметризована четырьмя наборами входных данных и последовательно оценивает права администратора, ведущего-владельца комнаты, зрителя без разрешения и зрителя с подтвержденным запросом. Такое построение теста уменьшает повторение кода и наглядно фиксирует ожидаемое правило доступа к общей загрузке документа.

Право загрузки определяет порядок работы с PDF-документом. Администратор получает доступ независимо от принадлежности к комнате, ведущий загружает документ в созданную комнату, а зритель — после утверждения запроса. При нарушении условия маршрут не выполняет сохранение файла и обновление состояния комнаты. Проверка успешной загрузки корректного PDF-документа приведена в листинге 8.

Листинг 8 – Проверка успешной загрузки корректного PDF-документа

```

@pytest.mark.asyncio
async def test_upload_document_saves_valid_pdf_and_broadcasts(
    monkeypatch, tmp_path, host):
    broadcasts = AsyncMock()
    monkeypatch.setattr(rooms_router, 'db', StubDb())
    monkeypatch.setattr(rooms_router, 'RoomRepository',
RoomRepoUpload)
    monkeypatch.setattr(rooms_router, 'DocumentRepository',
DocumentRepoUpload)
    monkeypatch.setattr(rooms_router, '_resolve_room_user',
                        lambda request: host)
    monkeypatch.setattr(rooms_router.config, 'UPLOAD_DIR',
tmp_path)

```

```

monkeypatch.setattr(rooms_router.signaling_service,
                    'broadcast', broadcasts)

result = await rooms_router.upload_document(
    make_request(), 'room01', UploadStub('report.pdf',
                                          b'%PDF-
1.7\ncontent'))

assert result['document']['fileName'] == 'report.pdf'
assert result['document']['page'] == 1
assert RoomRepoUpload.set_document_calls == [(5, 11, 2)]
assert RoomRepoUpload.set_page_calls == [(5, 1, None, 2)]
assert broadcasts.await_args_list[0].args[1]['type'] == \
    'document-updated'

```

Листинг 8 проверяет успешную ветвь маршрута `upload_document`. В тесте передается объект `UploadStub` с расширением `.pdf` и сигнатурой `%PDF`, а каталог хранения заменяется временной директорией `pytest`. После выполнения маршрута контролируется возвращаемый объект документа, вызовы установки текущего документа и первой страницы, а также отправка события `document-updated`. Поэтому тест подтверждает не только отсутствие ошибки при загрузке, но и обновление общего состояния комнаты, необходимое для синхронного просмотра PDF-файла участниками видеоконференции.

Особенностью успешной проверки является использование тестовых классов `RoomRepoUpload` и `DocumentRepoUpload` вместо реальных репозиторий. Они сохраняют сведения о переданных аргументах, но не выполняют SQL-запросы. Это позволяет проверить порядок вызова операций: сначала создается запись о документе, затем документ назначается текущим для комнаты и только после этого формируется сообщение для подключенных клиентов. Проверка отказных маршрутов функции `upload_document` приведена в листинге 9.

Листинг 9 – Проверка отказных маршрутов функции `upload_document`

```

@pytest.mark.asyncio
@pytest.mark.parametrize(
    ('user', 'room', 'allowed', 'file_name', 'file_data',
     'status', 'error'),
    [
        (None, active_room, True, 'report.pdf', b'%PDF-1.7',
         401, 'unauthorized'),
    ]

```

```

        (host, None, True, 'report.pdf', b'%PDF-1.7',
         403, 'room_not_active'),
        (host, planned_room, True, 'report.pdf', b'%PDF-1.7',
         403, 'room_not_active'),
        (viewer, active_room, False, 'report.pdf', b'%PDF-1.7',
         403, 'upload_permission_required'),
        (host, active_room, True, 'report.txt', b'%PDF-1.7',
         400, 'not_pdf'),
        (host, active_room, True, 'report.pdf', b'not-a-pdf',
         400, 'not_pdf'),
    ],
)
)
async def test_upload_document_rejects_invalid_routes(
    monkeypatch, user, room, allowed, file_name, file_data,
    status, error):
    configure_upload_dependencies(monkeypatch, user, room,
allowed)
    response = await rooms_router.upload_document(
        make_request(), 'room01', UploadStub(file_name,
file_data))
    assert response.status_code == status
    assert json.loads(response.body)['error'] == error

```

Листинг 9 содержит параметризованный тест ошибочных ветвей `upload_document`. В одном фрагменте задаются шесть независимых наборов данных: неавторизованный пользователь, отсутствующая комната, неактивная комната, отсутствие права загрузки, неправильное расширение и неверная сигнатура содержимого. Для каждого набора контролируются HTTP-статус и код ошибки, возвращаемый серверной частью. Проверка запрета смены страницы зрителем приведена в листинге 10.

Листинг 10 – Проверка запрета смены страницы зрителем

```

def test_set_page_rejects_viewer_without_room_management_rights(
    monkeypatch):
    viewer = make_user(3, 'VIEWER')
    monkeypatch.setattr(rooms_router, 'db', StubDb())
    monkeypatch.setattr(rooms_router, 'RoomRepository',
RoomRepoPage)
    monkeypatch.setattr(rooms_router, '_resolve_room_user',
        lambda request: viewer)

    response = rooms_router.set_page(
        make_request(), 'room01', {'page': 3})

    assert response.status_code == 403
    assert json.loads(response.body)['error'] == 'forbidden'

```

В листинге 10 проверяется маршрут `set_page`, который сохраняет номер текущей страницы PDF-документа. В качестве пользователя задается зритель, а тестовый репозиторий возвращает значение `False` для операции проверки прав управления комнатой. Ожидаемым результатом является ответ с кодом 403 и значением ошибки `forbidden`. Проверка подтверждает, что локальное переключение страницы зрителем не может изменить страницу, демонстрируемую ведущим всем участникам конференции.

Ограничение изменения общей страницы поддерживает синхронизацию участников. Ведущий передает номер страницы через `WebSocket`, а зритель может изучать документ самостоятельно только в клиентском интерфейсе. Серверная проверка сохраняет это правило и при прямом обращении к API-маршруту. Проверка создания вопроса и его обработки ведущим приведена в листинге 11.

Листинг 11 – Проверка создания вопроса и его обработки ведущим

```
@pytest.mark.asyncio
async def test_viewer_creates_question_and_host_resolves_it(
    monkeypatch):
    monkeypatch.setattr(rooms_router, 'db', StubDb())
    monkeypatch.setattr(rooms_router, 'RoomRepository',
RoomRepoQuestion)
    monkeypatch.setattr(rooms_router, 'DocumentRepository',
DocumentRepoQuestion)
    monkeypatch.setattr(rooms_router.signaling_service,
        'broadcast', broadcasts)
    monkeypatch.setattr(rooms_router, '_resolve_room_user',
        lambda request: viewer)

    created = await rooms_router.create_annotation(
        make_request(), 11, {'page': 2, 'type': 'QUESTION',
        'coordinates': {}, 'text': 'Нужен
комментарий'})
    assert created['annotation']['type'] == 'QUESTION'

    monkeypatch.setattr(rooms_router, '_resolve_room_user',
        lambda request: host)
    resolved = await rooms_router.resolve_question(
        make_request(), 11, 21)
    assert resolved['annotation']['type'] == 'QUESTION'
    assert broadcasts.await_args_list[-2].args[1]['type'] == \
        'question-resolved'
```

Листинг 11 объединяет две связанные операции работы с аннотациями. Сначала зритель вызывает `create_annotation` с типом `QUESTION` и получает созданный объект вопроса. Затем текущий пользователь заменяется на ведущего, который выполняет `resolve_question` для того же документа и аннотации. В результате проверяется не только создание записи с допустимым для зрителя типом, но и формирование события `question-resolved`, уведомляющего участников об обработке вопроса.

Тест исключает две потенциальные ошибки: создание зрителем произвольной аннотации, предназначенной для ведущего, и обработку вопроса пользователем, не управляющим комнатой. Эти ограничения реализованы на стороне сервера и не зависят от отображения элементов в интерфейсе. Поэтому даже при отправке специально сформированного HTTP-запроса пользователь не может выполнить действие, не соответствующее его роли.

Выполнение разработанного набора дает четырнадцать успешных проверок: часть тестов параметризована и запускается для нескольких комбинаций входных данных. Итог выполнения фиксируется в приложении В на рисунке В.8. Применение `pytest` позволило проверить критичные функции серверной части после внесения изменений в маршруты работы с комнатами и PDF-документами, а также подтвердить корректность обработки основных ограничений доступа и синхронизации [22].

Таким образом, модульное тестирование дополняет функциональную проверку методом черного ящика. Оно показывает корректность отдельных участков реализации, которые не всегда можно подробно оценить только по действиям пользователя в интерфейсе. Совместное использование обоих видов проверки обеспечивает подтверждение работоспособности алгоритма загрузки PDF-документа, управления страницами и взаимодействия ролей в условиях видеоконференции.

4.5 Анализ качества кода по метрикам

Для количественной оценки выбранного фрагмента кода применяются

метрики Холстеда, Маккейба и Чепина. Расчет выполнен для функции `upload_document`, так как она содержит несколько проверок входных данных, обращение к репозиториям, сохранение файла и отправку события обновления документа участникам комнаты. Метрики не заменяют тестирование, но позволяют оценить сопровождаемость и сложность фрагмента.

Для расчета цикломатической сложности построен управляющий граф алгоритма загрузки PDF-документа. При построении графа учитывались последовательные проверки авторизации, существования и активности комнаты, прав пользователя, расширения файла и сигнатуры PDF. Подобное представление соответствует подходу к описанию алгоритмов с использованием управляющих схем [24].

Управляющий граф алгоритма загрузки PDF-документа представлен на рисунке 38.

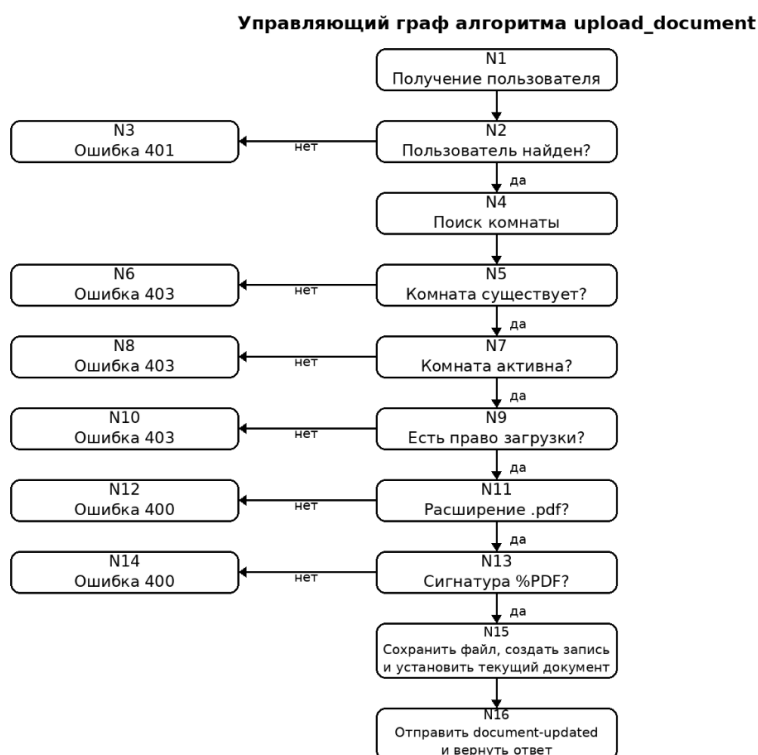


Рисунок 38 – Управляющий граф алгоритма загрузки PDF-документа

На управляющем графе выделены проверки, каждая из которых завершается ошибкой либо продолжает выполнение алгоритма. В актуальной

версии проверяется расширение имени файла и сигнатура содержимого PDF. Независимые маршруты алгоритма upload_document приведены в таблице 16.

Таблица 16 – Независимые маршруты алгоритма upload_document

Номер	Входные данные	Покрываемые узлы	Ожидаемый результат	Ссылка на результат
1	Пользователь не авторизован	N1-N3	Ошибка 401 unauthorized	рисунок В.8
2	Комната с указанным кодом отсутствует	N1, N2, N4-N6	Ошибка 403 room_not_active	рисунок В.8
3	Комната существует, но не активна	N1, N2, N4, N5, N7, N8	Ошибка 403 room_not_active	рисунок В.8
4	Зритель не имеет разрешения на загрузку	N1, N2, N4, N5, N7, N9, N10	Ошибка 403 upload_permission_required	рисунок В.8
5	Файл имеет неверное расширение	N1-N11, N12	Ошибка 400 not_pdf	рисунок В.8
6	Файл имеет расширение .pdf, но не содержит сигнатуру PDF	N1-N13, N14	Ошибка 400 not_pdf	рисунок В.8
7	Корректный PDF загружает пользователь с правом загрузки	N1-N16	Файл сохранен, создана запись документа, отправлено событие document-updated	рисунок В.8

Цикломатическая сложность управляющего графа рассчитывается по формуле $V(G) = E - N + 2P$, где E - количество дуг графа, N - количество вершин, P - число связности. Для графа на рисунке 38 принято $N = 16$, $E = 21$, $P = 1$. Тогда $V(G) = 21 - 16 + 2 = 7$. Исходные данные и результаты расчета метрик Холстеда для функции upload_document приведены в таблице 17.

Таблица 17 – Расчет метрик Холстеда для функции upload_document

Показатель	Обозначение/формула	Значение	Интерпретация
Уникальные операторы	η_1	37	Используются проверки, вызовы функций, обращения к атрибутам и операции возврата
Уникальные операнды	η_2	63	Фрагмент оперирует параметрами запроса, комнатой, файлом, документом и сообщениями ошибок
Общее число операторов	N_1	102	Количество операторов увеличено из-за проверок и вызовов репозитория
Общее число операндов	N_2	118	Операнды включают переменные, строковые константы и числовые коды ответов
Словарь программы	$\eta = \eta_1 + \eta_2$	100	Словарь фрагмента остается допустимым для отдельного маршрута
Длина программы	$N = N_1 + N_2$	220	Фрагмент имеет среднюю длину для серверного обработчика загрузки
Объем программы	$V = N \cdot \log_2(\eta)$	$\approx 1461,65$	Объем выше, чем у простого маршрута, из-за валидации и рассылки событий
Трудность	$D = (\eta_1 / 2) \cdot (N_2 / \eta_2)$	$\approx 34,65$	Фрагмент требует внимательной проверки ошибочных ветвей
Усилие	$E = D \cdot V$	$\approx 50647,28$	Основное усилие связано с условиями отказа и формированием результата

Расчет метрик Холстеда показывает, что обработчик загрузки PDF-документа имеет умеренный объем для серверного маршрута, однако содержит несколько независимых проверок. Это подтверждает необходимость отдельного

покрытия ошибочных сценариев. Результаты расчета метрик Маккейба и Чепина для функции `upload_document` приведены в таблице 18.

Таблица 18 – Метрики Маккейба и Чепина для функции `upload_document`

Метрика	Обозначение/формула	Значение	Интерпретация
Цикломатическая сложность Маккейба	$V(G) = E - N + 2P$	7	Требуется не менее семи независимых маршрутов тестирования
Количество параметров и выходных данных по Чепину	P	4	Учитываются <code>request</code> , <code>code</code> , <code>pdf_file</code> и результат <code>document_payload</code>
Количество модифицируемых данных по Чепину	M	8	Создаются и изменяются <code>user</code> , <code>room</code> , <code>original_name</code> , <code>data</code> , <code>stored_name</code> , <code>target</code> , <code>document</code> и <code>document_payload</code>
Количество управляющих данных по Чепину	C	6	Управление выполняют <code>user</code> , <code>room</code> , статус комнаты, право загрузки, расширение и сигнатура файла
Количество неиспользуемых данных по Чепину	T	0	Лишние переменные в выбранном фрагменте не выделены
Итоговая метрика Чепина	$Q = P + 2M + 3C + 0,5T$	38	Основная сложность связана с управляющими условиями проверки запроса

Значение цикломатической сложности 7 не указывает на критическую сложность, но показывает, что функция не должна проверяться одним успешным тестом. Метрика Чепина также подтверждает, что основная сложность сосредоточена в управляющих переменных, отвечающих за права пользователя, состояние комнаты и корректность PDF-файла.

По результатам тестирования и расчета метрик можно сделать вывод, что реализованная система корректно обрабатывает ключевые, однако текущий набор проверок является достаточным для подтверждения работоспособности основной функциональности.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана информационная система для синхронизированного взаимодействия с PDF-документами в условиях видеоконференции. Разработанное веб-приложение предназначено для дистанционного обсуждения документов между пользователями и обеспечивает совместную работу с PDF-файлом без необходимости использования сторонних средств демонстрации экрана.

В первой главе была рассмотрена предметная область, связанная с организацией удаленного взаимодействия пользователей при работе с документами. Были изучены особенности PDF-формата, рассмотрены существующие решения для проведения видеоконференций и демонстрации материалов, а также выявлены ограничения таких средств при совместной работе с документом. На основе проведенного анализа была обоснована необходимость разработки специализированного веб-приложения, ориентированного на синхронизированный просмотр PDF-документов, управление страницами и разграничение ролей пользователей.

Во второй главе было выполнено проектирование информационной системы. Были определены основные требования к функциональности веб-приложения, выделены роли администратора, ведущего и участника, а также описаны варианты использования системы. Также были разработаны архитектура приложения, структура базы данных, пользовательский интерфейс и основные варианты взаимодействия пользователей с системой. По итогам проектирования была подготовлена база для последующей программной реализации информационной системы.

В третьей главе была выполнена разработка информационной системы. Для реализации серверной части использовался язык программирования Python и фреймворк FastAPI. Хранение данных организовано с использованием реляционной базы данных MySQL. Клиентская часть веб-приложения

реализована с применением HTML, CSS и JavaScript. Для отображения PDF-документов использована библиотека PDF.js, для обмена событиями в реальном времени применены WebSocket-соединения, а для организации видеосвязи использована технология WebRTC. В ходе разработки были реализованы регистрация и авторизация пользователей, управление профилем, создание комнат, подключение участников, загрузка PDF-документа, синхронизация текущей страницы, работа с ролями и управление пользователями.

В четвертой главе было проведено тестирование и оценка качества разработанной информационной системы. Проверка выполнялась методом черного ящика, с использованием модульного тестирования и анализа метрик качества программного кода. В ходе функционального тестирования были проверены основные пользовательские сценарии: вход в систему, работа с профилем, создание комнаты, подключение участника, загрузка PDF-документа и синхронизация просмотра. Модульное тестирование позволило проверить отдельные программные компоненты, связанные с обработкой документов, управлением комнатами и проверкой прав доступа. Дополнительно были рассчитаны метрики сложности и качества кода, позволяющие оценить реализованный программный модуль с точки зрения структурной сложности и сопровождаемости.

В результате выполнения работы была достигнута поставленная цель – разработано веб-приложение для синхронизированного взаимодействия с PDF-документами в условиях видеоконференции. Разработанная система позволяет создавать комнаты, подключать участников, загружать PDF-документы, синхронно отображать страницы документа у пользователей и организовывать видеосвязь в рамках одной рабочей среды. Использование ролей позволяет разграничить права пользователей и обеспечить управление процессом обсуждения документа.

Практическая значимость разработанной системы заключается в том, что она может применяться для дистанционного обсуждения документов в образовательной, организационной и рабочей среде. В отличие от обычной

демонстрации экрана, система ориентирована именно на совместную работу с PDF-документом, что делает процесс обсуждения более управляемым и удобным для участников. Пользователи видят один и тот же документ, а изменение страницы у ведущего синхронно отображается у остальных участников комнаты, что снижает вероятность потери контекста при обсуждении.

В дальнейшем разработанное веб-приложение может быть расширено за счет добавления инструментов аннотирования документов, сохранения истории обсуждений, поддержки нескольких материалов в одной комнате, расширенной системы уведомлений и интеграции с внешними сервисами хранения файлов. Также перспективным направлением развития является поддержка других типов материалов помимо PDF-документов, например презентаций, текстовых документов, изображений и веб-ссылок. Это позволит использовать систему не только для обсуждения PDF-файлов, но и для совместной работы с различными цифровыми материалами в рамках одной видеоконференции. Кроме того, система может быть адаптирована для интеграции в корпоративные информационные системы, образовательные платформы, банковские и иные отраслевые системы, где требуется организованное дистанционное обсуждение документов и материалов между участниками.

Список используемых источников

1. ISO. Portable document format – ISO 32000-2:2020 [Электронный ресурс]. – URL: <https://www.iso.org/standard/75839.html> (дата обращения: 08.04.2026).
2. Zoom Support. Демонстрация экрана или рабочего стола в Zoom [Электронный ресурс]. – URL: https://support.zoom.com/hc/ru/article?id=zm_kb&sysparm_article=KB0060606 (дата обращения: 09.04.2026).
3. Microsoft. Приложение для демонстрации экрана | Microsoft Teams [Электронный ресурс]. – URL: <https://www.microsoft.com/ru-ru/microsoft-teams/screen-sharing> (дата обращения: 09.04.2026).
4. Яндекс 360 для бизнеса. Поделиться экраном | Телемост [Электронный ресурс]. – URL: <https://yandex.ru/support/yandex-360/business/telemost/web/ru/sharing-screen> (дата обращения: 10.04.2026).
5. Adobe Help Center. Предоставление общего доступа по ссылкам на файлы PDF [Электронный ресурс]. – URL: <https://helpx.adobe.com/ru/acrobat/desktop/share-and-review-documents/share-documents/share-pdfs.html> (дата обращения: 11.04.2026).
6. Буч Г., Рамбо Д., Джекобсон И. Язык UML. Руководство пользователя. 2-е изд. – М. : ДМК Пресс, 2006. – 496 с.
7. Python Software Foundation. Документация Python [Электронный ресурс]. – URL: <https://docs.python.org/ru/3/> (дата обращения: 15.06.2026).
8. FastAPI. Документация FastAPI [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/ru/> (дата обращения: 15.06.2026).
9. Uvicorn. An ASGI web server, for Python [Электронный ресурс]. – URL: <https://www.uvicorn.org/> (дата обращения: 15.06.2026).
10. FastAPI. WebSockets [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/ru/advanced/websockets/> (дата обращения: 15.06.2026).
11. Jinja. Jinja Documentation [Электронный ресурс]. – URL:

<https://jinja.palletsprojects.com/> (дата обращения: 15.06.2026).

12. MDN Web Docs. HTML: язык разметки гипертекста [Электронный ресурс]. – URL: <https://developer.mozilla.org/ru/docs/Web/HTML> (дата обращения: 15.06.2026).

13. MDN Web Docs. CSS: каскадные таблицы стилей [Электронный ресурс]. – URL: <https://developer.mozilla.org/ru/docs/Web/CSS> (дата обращения: 15.06.2026).

14. MDN Web Docs. Руководство по JavaScript [Электронный ресурс]. – URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide> (дата обращения: 15.06.2026).

15. MDN Web Docs. WebRTC API [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Web/API/WebRTC_API (дата обращения: 15.06.2026).

16. Mozilla. PDF.js [Электронный ресурс]. – URL: <https://mozilla.github.io/pdf.js/> (дата обращения: 16.06.2026).

17. Microsoft Learn. Компонент Database Engine SQL Server [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/sql/database-engine/sql-database-engine?view=sql-server-ver17> (дата обращения: 17.06.2026).

18. Microsoft Learn. SQL Server Management Studio (SSMS) [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/ssms/sql-server-management-studio-ssms> (дата обращения: 17.06.2026).

19. Microsoft Learn. Библиотеки базы данных SQL Azure для Python [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/python/api/overview/azure/sql> (дата обращения: 17.06.2026).

20. Выпускная квалификационная работа: стандарты и программные технологии : учебно-методическое пособие для бакалавров по направлениям 09.03.04 «Программная инженерия», 09.03.02 «Информационные системы и технологии» / под ред. П. П. Макарычева. – Пенза : Изд-во ПГУ, 2020. – 182 с.

21. Семахин А. М. Методы верификации и оценки качества программного обеспечения : учебное пособие. – Курган : Изд-во Курганского

государственного университета, 2018. – 150 с.

22. Майерс Г. Искусство тестирования программ. – М. : Финансы и статистика, 1982. – 176 с.

23. Microsoft Learn. Тестирование с помощью Pytest [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/training/modules/test-python-with-pytest/> (дата обращения: 17.06.2026).

24. ГОСТ 19.701–90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М. : Изд-во стандартов, 1991.

Приложение А
(обязательное)
Листинг кода приложения

В приложении приведены основные фрагменты серверной части разработанной информационной системы. В листинг включены файлы, необходимые для понимания работы приложения: инициализация FastAPI-приложения, подключение к базе данных, модель текущего пользователя, репозитории комнат и документов, маршруты работы с комнатой, PDF-документом, аннотациями и WebSocket-синхронизацией. Шаблоны HTML, файлы JavaScript и CSS в приложение не включены, так как они относятся к представлению интерфейса и не являются ключевыми для раскрытия серверной логики.

Листинг А.1 – Файл app/main.py

```
from __future__ import annotations

import logging

from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse, JSONResponse
from fastapi.staticfiles import StaticFiles

from app.core import config
from app.routers import auth_router, dashboard_router,
rooms_router

logger = logging.getLogger('conference_app')

app = FastAPI(title=config.APP_NAME)

config.UPLOAD_DIR.mkdir(parents=True, exist_ok=True)

app.mount('/static',
StaticFiles(directory=str(config.STATIC_DIR)), name='static')
app.mount('/uploads',
StaticFiles(directory=str(config.UPLOAD_DIR)), name='uploads')

app.include_router(auth_router.router)
app.include_router(dashboard_router.router)
app.include_router(rooms_router.router)

@app.exception_handler(RuntimeError)
async def runtime_error_handler(request: Request, exc:
RuntimeError):
    text = str(exc)
    if 'SQL Server' in text or 'config.json' in text or 'баз' in
text.lower():
```

```

        logger.error('Ошибка базы данных: %s', text)
        if request.url.path.startswith('/api/'):
            return JSONResponse({'ok': False, 'error': 'База
данных временно недоступна. Проверьте config.json и SQL Server.'},
                                status_code=503)
            return HTMLResponse(
                '<!doctype html><html lang="ru"><head><meta
charset="utf-8">'
                '<title>База данных недоступна</title>'
                '<style>body{font-family:Arial,sans-
serif;background:#f6f8fb;margin:0;padding:40px}'
                '.card{max-
width:780px;margin:auto;background:white;border-
radius:18px;padding:28px;box-shadow:0 14px 40px
rgba(20,40,80,.12)}'
                'code{background:#eef2f7;border-radius:8px;padding:2px
6px}</style></head><body><div class="card">'
                '<h1>База данных временно недоступна</h1>'
                '<p>Приложение запущено, но не удалось подключиться к
SQL Server.</p>'
                '<p>Проверьте файл <code>config.json</code>, имя
сервера из SSMS и состояние службы SQL Server.</p>'
                f'<p><small>{text}</small></p>'
                '</div></body></html>',
                status_code=503,
            )
        raise exc

@app.get('/api/health')
def health() -> dict:
    return {'ok': True, 'app': config.APP_NAME, 'db_backend':
config.DB_BACKEND}

```

Листинг А.2 – Файл app/core/database.py

```

from __future__ import annotations

import datetime as _dt
import time
from contextlib import contextmanager
from typing import Any, Iterator

from app.core import config

class RowDict(dict):
    """Словарь строки БД для единообразной работы репозитория"""

    def keys(self): # type: ignore[override]
        return super().keys()

    def _normalize_value(value: Any) -> Any:

```

```

    if isinstance(value, _dt.datetime):
        return value.isoformat(sep=' ')
    if isinstance(value, (_dt.date, _dt.time)):
        return value.isoformat()
    return value

class PyodbcCursorAdapter:
    def __init__(self, cursor: Any) -> None:
        self.cursor = cursor
        self.lastrowid: int | None = None

    def _row_to_dict(self, row: Any) -> RowDict | None:
        if row is None:
            return None
        columns = [str(col[0]) for col in (self.cursor.description
or [])]
        return RowDict({name: _normalize_value(value) for name,
value in zip(columns, row)})

    def fetchone(self) -> RowDict | None:
        return self._row_to_dict(self.cursor.fetchone())

    def fetchall(self) -> list[RowDict]:
        return [self._row_to_dict(row) for row in
self.cursor.fetchall()]

class PyodbcConnectionAdapter:
    backend = 'mssql'

    def __init__(self, raw: Any) -> None:
        self.raw = raw

    def execute(self, sql: str, params: tuple | list | None =
None) -> PyodbcCursorAdapter:
        cur = self.raw.cursor()
        cur.execute(sql, tuple(params or ()))
        return PyodbcCursorAdapter(cur)

    def commit(self) -> None:
        self.raw.commit()

    def rollback(self) -> None:
        self.raw.rollback()

    def close(self) -> None:
        self.raw.close()

class Database:
    """Единая точка доступа к SQL Server.
```

Таблицы создаются отдельным SQL-скриптом из папки db. При запуске сервера схема БД не изменяется: приложение только подключается к уже подготовленной базе данных.

```

"""

backend = 'mssql'

def _import_pyodbc(self):
    try:
        import pyodbc # type: ignore
    except ImportError as exc:
        raise RuntimeError(
            'Для работы с SQL Server требуется пакет pyodbc. '
            'Запустите run.bat, чтобы установить зависимости
через pip.'
        ) from exc
    return pyodbc

def _pick_driver(self, pyodbc: Any) -> str:
    if config.MSSQL_DRIVER:
        return config.MSSQL_DRIVER
    drivers = list(pyodbc.drivers())
    for driver in [
        'ODBC Driver 18 for SQL Server',
        'ODBC Driver 17 for SQL Server',
        'SQL Server Native Client 11.0',
        'SQL Server',
    ]:
        if driver in drivers:
            return driver
    raise RuntimeError(
        'Не найден ODBC-драйвер SQL Server. Установите
Microsoft ODBC Driver 17/18 for SQL Server.'
    )

def _connection_string(self, database: str | None = None, *,
autocommit: bool = False) -> tuple[str, bool]:
    if config.MSSQL_CONNECTION_STRING:
        return config.MSSQL_CONNECTION_STRING, autocommit
    pyodbc = self._import_pyodbc()
    driver = self._pick_driver(pyodbc)
    db_name = database or config.MSSQL_DATABASE
    parts = [
        f'DRIVER={{driver}}',
        f'SERVER={config.MSSQL_SERVER}',
        f'DATABASE={db_name}',
        f'Connection
Timeout={config.MSSQL_CONNECTION_TIMEOUT}',
        f'LoginTimeout={config.MSSQL_LOGIN_TIMEOUT}',
    ]
    if config.MSSQL_USER:

```

```

        parts.extend([f'UID={config.MSSQL_USER}',
f'PWD={config.MSSQL_PASSWORD}'])
    else:

parts.append(f'Trusted_Connection={config.MSSQL_TRUSTED_CONNECTION
}')
    if config.MSSQL_TRUST_SERVER_CERTIFICATE:

parts.append(f'TrustServerCertificate={config.MSSQL_TRUST_SERVER_C
ERTIFICATE}')
    return ';'.join(parts) + ';', autocommit

def _connect_with_retry(self, database: str, *, autocommit:
bool = False):
    pyodbc = self._import_pyodbc()
    conn_str, auto = self._connection_string(database,
autocommit=autocommit)
    attempts = 3
    last_error: Exception | None = None
    for attempt in range(1, attempts + 1):
        try:
            return pyodbc.connect(
                conn_str,
                autocommit=auto,
                timeout=max(1,
int(config.MSSQL_LOGIN_TIMEOUT)),
            )
        except Exception as exc:
            last_error = exc
            if attempt < attempts:
                time.sleep(1.0)
    raise RuntimeError(
        'Не удалось подключиться к SQL Server. Проверьте
config.json: '
        f'сервер "{config.MSSQL_SERVER}", база "{database}". '
        f'Исходная ошибка: {last_error}'
    ) from last_error

def connect(self) -> PyodbcConnectionAdapter:
    raw = self._connect_with_retry(config.MSSQL_DATABASE)
    return PyodbcConnectionAdapter(raw)

@contextmanager
def session(self) -> Iterator[PyodbcConnectionAdapter]:
    conn = self.connect()
    try:
        yield conn
        conn.commit()
    except Exception:
        conn.rollback()
        raise
    finally:
        conn.close()

```

```
db = Database()
```

Листинг А.3 – Файл app/core/security.py

```
from __future__ import annotations

import hashlib
import hmac
import secrets
from dataclasses import dataclass
from typing import Optional

from fastapi import Request

from app.core import config


@dataclass
class CurrentUser:
    id: int
    login: str
    role_code: str
    role_name: str
    first_name: str
    last_name: str
    patronymic: str | None
    email: str | None = None
    phone: str | None = None
    avatar_path: str | None = None

    @property
    def full_name(self) -> str:
        parts = [self.last_name, self.first_name, self.patronymic
or '']
        return ' '.join(p for p in parts if p).strip() or
self.login

    @property
    def initials(self) -> str:
        return ''.join([self.first_name[:1],
self.last_name[:1]]).upper() or self.login[:2].upper()

    @property
    def is_admin(self) -> bool:
        return self.role_code == 'ADMIN'

    @property
    def is_speaker(self) -> bool:
        return self.role_code == 'SPEAKER'

    @property
    def is_viewer(self) -> bool:
```

```

        return self.role_code == 'VIEWER'

sessions: dict[str, CurrentUser] = {}

def hash_password(password: str) -> str:
    salt = 'conference-demo-salt'
    return hashlib.sha256((salt + password).encode('utf-8')).hexdigest()

def verify_password(password: str, password_hash: str) -> bool:
    return hmac.compare_digest(hash_password(password), password_hash)

def create_session(user: CurrentUser) -> str:
    token = secrets.token_urlsafe(32)
    sessions[token] = user
    return token

def delete_session(token: str | None) -> None:
    if token:
        sessions.pop(token, None)

def get_current_user(request: Request) -> Optional[CurrentUser]:
    token = request.cookies.get(config.SESSION_COOKIE_NAME)
    if not token:
        return None
    return sessions.get(token)

# Краткоживущие токены страницы комнаты. Они позволяют тестировать
# несколько ролей
# в разных вкладках одного браузера: WebSocket и AJAX комнаты
# опираются на токен,
# который был выдан при открытии страницы, а не на текущую cookie
# после нового входа.
room_page_tokens: dict[str, CurrentUser] = {}

def create_room_page_token(user: CurrentUser) -> str:
    token = secrets.token_urlsafe(24)
    room_page_tokens[token] = user
    return token

def get_user_by_room_page_token(token: str | None) -> Optional[CurrentUser]:
    if not token:

```

```

        return None
    return room_page_tokens.get(token)

```

Листинг А.4 – Файл app/services/signaling_service.py

```

from __future__ import annotations

import asyncio
from dataclasses import dataclass
from typing import Any

from fastapi import WebSocket

from app.core.security import CurrentUser


@dataclass
class Peer:
    peer_id: str
    websocket: WebSocket
    user: CurrentUser


class SignalingService:
    def __init__(self) -> None:
        self._rooms: dict[str, dict[str, Peer]] = {}
        self._lock = asyncio.Lock()

    def _peer_payload(self, peer: Peer) -> dict[str, Any]:
        return {
            'peerId': peer.peer_id,
            'userId': peer.user.id,
            'fullName': peer.user.full_name,
            'initials': peer.user.initials,
            'role': peer.user.role_code,
            'roleName': peer.user.role_name,
        }

    async def connect(self, room_code: str, peer_id: str,
        websocket: WebSocket, user: CurrentUser) -> list[dict[str, Any]]:
        await websocket.accept()
        async with self._lock:
            peers = self._rooms.setdefault(room_code, {})
            existing = [self._peer_payload(peer) for pid, peer in
            peers.items() if pid != peer_id]
            old = peers.get(peer_id)
            if old:
                try:
                    await old.websocket.close(code=1000)
                except Exception:
                    pass
            peers[peer_id] = Peer(peer_id=peer_id,
            websocket=websocket, user=user)
            return existing

```

```

    async def disconnect(self, room_code: str, peer_id: str) ->
None:
        async with self._lock:
            room = self._rooms.get(room_code)
            if not room:
                return
            room.pop(peer_id, None)
            if not room:
                self._rooms.pop(room_code, None)

    async def participants(self, room_code: str) -> list[dict[str,
Any]]:
        async with self._lock:
            return [self._peer_payload(peer) for peer in
self._rooms.get(room_code, {}).values()]

    async def has_user_in_room(self, room_code: str, user_id: int)
-> bool:
        async with self._lock:
            return any(peer.user.id == user_id for peer in
self._rooms.get(room_code, {}).values())

    async def send_to_peer(self, room_code: str, target_peer_id:
str, payload: dict[str, Any]) -> None:
        async with self._lock:
            target = self._rooms.get(room_code,
{}).get(target_peer_id)
            if target:
                await target.websocket.send_json(payload)

    async def broadcast(self, room_code: str, payload: dict[str,
Any], exclude_peer_id: str | None = None) -> None:
        async with self._lock:
            peers = list(self._rooms.get(room_code, {}).values())
            for peer in peers:
                if exclude_peer_id and peer.peer_id ==
exclude_peer_id:
                    continue
                try:
                    await peer.websocket.send_json(payload)
                except Exception:
                    pass

```

```
signaling_service = SignalingService()
```

Листинг A.5 – Основные фрагменты файла app/repositories/room_repository.py

```
from __future__ import annotations
```

```
import json
import secrets
from typing import Any
```

```

from app.core.security import CurrentUser
from app.repositories.base_repository import BaseRepository

STATUS_LABELS = {
    'PLANNED': 'Запланована',
    'ACTIVE': 'Активна',
    'FINISHED': 'Завершена',
    'CANCELLED': 'Отменена',
}

class RoomRepository(BaseRepository):
    def _generate_code(self) -> str:
        while True:
            code = secrets.token_hex(3).upper()
            if not self.get_room_by_code(code):
                return code

    def _with_status_label(self, row: Any | dict[str, Any]) -> dict[str, Any]:
        item = dict(row)
        item['StatusLabel'] = STATUS_LABELS.get(item.get('Status'), item.get('Status'))
        item.setdefault('IsWaiting', 0)
        return item

    def create_room(self, owner: CurrentUser, title: str, description: str | None, max_participants: int | None, status: str = 'PLANNED') -> dict[str, Any]:
        status = status if status in STATUS_LABELS else 'PLANNED'
        code = self._generate_code()
        started_at = None
        room_id = self.insert_and_get_id(
            'Rooms',
            'RoomId',
            ['OwnerId', 'Title', 'Description', 'JoinCode', 'Status', 'IsPublic', 'MaxParticipants', 'StartedAt'],
            (owner.id, title, description, code, status, 0, max_participants, started_at),
        )
        if status == 'ACTIVE':
            self.conn.execute('UPDATE Rooms SET StartedAt=CURRENT_TIMESTAMP WHERE RoomId=?', (room_id,))
            self.conn.execute('INSERT INTO RoomStates (RoomId, CurrentPage, Zoom) VALUES (?, 1, 100)', (room_id,))
            self.add_event(room_id, owner.id, 'ROOM_CREATED', {'title': title, 'code': code, 'status': status})
            return self.get_room(room_id) # type: ignore[return-value]

    def set_room_status(self, room_id: int, status: str, user_id:

```

```

int) -> None:
    if status not in STATUS_LABELS:
        raise ValueError('unsupported room status')
    if status == 'ACTIVE':
        self.conn.execute("UPDATE Rooms SET Status='ACTIVE',
StartedAt=COALESCE(StartedAt, CURRENT_TIMESTAMP), FinishedAt=NULL,
UpdatedAt=CURRENT_TIMESTAMP WHERE RoomId=?", (room_id,))
        event = 'ROOM_ACTIVATED'
    elif status == 'FINISHED':
        self.conn.execute("UPDATE Rooms SET Status='FINISHED',
FinishedAt=CURRENT_TIMESTAMP, UpdatedAt=CURRENT_TIMESTAMP WHERE
RoomId=?", (room_id,))
        self.conn.execute('UPDATE RoomParticipants SET
IsConnected=0, LeftAt=CURRENT_TIMESTAMP WHERE RoomId=?',
(room_id,))
        event = 'ROOM_FINISHED'
    elif status == 'CANCELLED':
        self.conn.execute("UPDATE Rooms SET
Status='CANCELLED', FinishedAt=CURRENT_TIMESTAMP,
UpdatedAt=CURRENT_TIMESTAMP WHERE RoomId=?", (room_id,))
        event = 'ROOM_CANCELLED'
    else:
        self.conn.execute("UPDATE Rooms SET Status='PLANNED',
UpdatedAt=CURRENT_TIMESTAMP WHERE RoomId=?", (room_id,))
        event = 'ROOM_PLANNED'
    self.add_event(room_id, user_id, event, {'status':
status})

def activate_room(self, room_id: int, user_id: int) -> None:
    self.set_room_status(room_id, 'ACTIVE', user_id)

def finish_room(self, room_id: int, user_id: int) -> None:
    self.set_room_status(room_id, 'FINISHED', user_id)

def can_manage_room(self, room: dict[str, Any], user:
CurrentUser) -> bool:
    return user.is_admin or user.id ==
int(room['OwnerUserId'])

def can_open_room(self, room: dict[str, Any], user:
CurrentUser) -> bool:
    return room['Status'] == 'ACTIVE'

def join_room(self, room: dict[str, Any], user: CurrentUser) -
> None:
    role = 'HOST' if user.id == int(room['OwnerUserId']) else
('SPEAKER' if user.is_speaker else 'VIEWER')
    active = self.conn.execute('SELECT ParticipantId FROM
RoomParticipants WHERE RoomId=? AND UserId=? AND IsConnected=1',
(int(room['RoomId']), user.id)).fetchone()
    if active:
        self.conn.execute('UPDATE RoomParticipants SET
DisplayName=?, RoleInRoom=?, LeftAt=NULL WHERE ParticipantId=?',

```

```

(user.full_name, role, int(active['ParticipantId']))
    else:
        self.conn.execute(
            '''INSERT INTO RoomParticipants (RoomId, UserId,
RoleInRoom, DisplayName, IsConnected)
            VALUES (?, ?, ?, ?, 1)''',
            (int(room['RoomId']), user.id, role,
user.full_name),
        )
        self.add_event(int(room['RoomId']), user.id,
'PARTICIPANT_JOINED', {'name': user.full_name})

    def get_room_state(self, room_id: int) -> Any:
        state = self.conn.execute('SELECT * FROM RoomStates WHERE
RoomId=?', (room_id,)).fetchone()
        if not state:
            self.conn.execute('INSERT INTO RoomStates (RoomId,
CurrentPage, Zoom) VALUES (?, 1, 100)', (room_id,))
            state = self.conn.execute('SELECT * FROM RoomStates
WHERE RoomId=?', (room_id,)).fetchone()
        return state

    def set_current_document(self, room_id: int, document_id: int,
user_id: int) -> None:
        self.conn.execute('UPDATE RoomStates SET
CurrentDocumentId=?, CurrentPage=1, Zoom=100,
UpdatedAt=CURRENT_TIMESTAMP WHERE RoomId=?', (document_id,
room_id))
        self.add_event(room_id, user_id, 'PDF_UPLOADED',
{'documentId': document_id})

    def set_current_page(self, room_id: int, page: int, zoom: int
| None, user_id: int) -> None:
        if zoom is None:
            self.conn.execute('UPDATE RoomStates SET
CurrentPage=?, UpdatedAt=CURRENT_TIMESTAMP WHERE RoomId=?', (page,
room_id))
        else:
            self.conn.execute('UPDATE RoomStates SET
CurrentPage=?, Zoom=?, UpdatedAt=CURRENT_TIMESTAMP WHERE
RoomId=?', (page, zoom, room_id))
            self.add_event(room_id, user_id, 'PDF_PAGE_CHANGED',
{'page': page, 'zoom': zoom})

    def add_message(self, room_id: int, user_id: int, text: str) -
> Any:
        msg_id = self.insert_and_get_id('RoomMessages',
'MessageId', ['RoomId', 'UserId', 'MessageText'], (room_id,
user_id, text))
        full_name = self.full_name_sql('u')
        return self.conn.execute(
            f'''SELECT m.*, {full_name} AS FullName FROM

```

```

RoomMessages m JOIN Users u ON u.UserId=m.UserId WHERE
m.MessageId=?'',
    (msg_id,),
    ).fetchone()

def list_messages(self, room_id: int, limit: int = 50) ->
list[Any]:
    full_name = self.full_name_sql('u')
    if self.is_mssql:
        return self.conn.execute(
            f'''SELECT TOP (?) m.*, {full_name} AS FullName
            FROM RoomMessages m JOIN Users u ON
u.UserId=m.UserId
            WHERE m.RoomId=? AND m.IsDeleted=0 ORDER BY
m.CreatedAt ASC''',
            (limit, room_id),
            ).fetchall()
        return self.conn.execute(
            f'''SELECT m.*, {full_name} AS FullName
            FROM RoomMessages m JOIN Users u ON
u.UserId=m.UserId
            WHERE m.RoomId=? AND m.IsDeleted=0 ORDER BY
m.CreatedAt ASC LIMIT ?''',
            (room_id, limit),
            ).fetchall()

    def add_event(self, room_id: int, user_id: int | None,
event_type: str, payload: dict | None = None) -> None:
        self.conn.execute('INSERT INTO RoomEvents (RoomId, UserId,
EventType, PayloadJson) VALUES (?, ?, ?, ?)', (room_id, user_id,
event_type, json.dumps(payload or {}, ensure_ascii=False)))

    def list_events(self, room_id: int, limit: int = 50) ->
list[Any]:
        if self.is_mssql:
            return self.conn.execute('SELECT TOP (?) * FROM
RoomEvents WHERE RoomId=? ORDER BY CreatedAt DESC', (limit,
room_id)).fetchall()
            return self.conn.execute('SELECT * FROM RoomEvents WHERE
RoomId=? ORDER BY CreatedAt DESC LIMIT ?', (room_id,
limit)).fetchall()

```

Листинг А.6 – Файл app/repositories/document_repository.py

```

from __future__ import annotations

import json
from typing import Any

from app.core.security import CurrentUser
from app.repositories.base_repository import BaseRepository

class DocumentRepository(BaseRepository):

```

```

    def create_document(self, room_id: int, uploaded_by_user_id:
int, original_file_name: str, stored_file_name: str, file_path:
str, file_size: int) -> Any:
        document_id = self.insert_and_get_id(
            'Documents',
            'DocumentId',
            ['RoomId', 'UploadedByUserId', 'OriginalFileName',
'StoredFileName', 'FilePath', 'FileSize', 'PageCount'],
            (room_id, uploaded_by_user_id, original_file_name,
stored_file_name, file_path, file_size, None),
        )
        return self.get_document(document_id) # type:
ignore[return-value]

    def get_document(self, document_id: int | None) -> Any | None:
        if not document_id:
            return None
        return self.conn.execute('SELECT * FROM Documents WHERE
DocumentId=? AND IsActive=1', (int(document_id),)).fetchone()

    def list_documents(self, room_id: int) -> list[Any]:
        return self.conn.execute('SELECT * FROM Documents WHERE
RoomId=? AND IsActive=1 ORDER BY UploadedAt DESC',
(room_id,)).fetchall()

    def user_can_read_document(self, document: Any, user:
CurrentUser) -> bool:
        return user is not None and document is not None

    def can_upload_document(self, room: dict[str, Any], user:
CurrentUser) -> bool:
        if user.is_admin or user.id == int(room['OwnerUserId']):
            return True
        req = self.conn.execute(
            '''SELECT Status FROM DocumentUploadRequests
            WHERE RoomId=? AND RequestedByUserId=?''',
            (int(room['RoomId']), user.id),
        ).fetchone()
        return bool(req and req['Status'] == 'APPROVED')

    def upload_request_status(self, room_id: int, user:
CurrentUser) -> str:
        row = self.conn.execute(
            'SELECT Status FROM DocumentUploadRequests WHERE
RoomId=? AND RequestedByUserId=?',
            (room_id, user.id),
        ).fetchone()
        return row['Status'] if row else 'NOT_REQUESTED'

    def request_upload_permission(self, room_id: int, user_id:
int) -> Any:
        if self.is_mssql:
            self.conn.execute(

```

```

        '''IF EXISTS (SELECT 1 FROM DocumentUploadRequests
WHERE RoomId=? AND RequestedByUserId=?)
        UPDATE DocumentUploadRequests
        SET Status='PENDING',
RequestedAt=CURRENT_TIMESTAMP, DecidedAt=NULL,
DecidedByUserId=NULL
        WHERE RoomId=? AND RequestedByUserId=?
        ELSE
        INSERT INTO DocumentUploadRequests (RoomId,
RequestedByUserId, Status) VALUES (?, ?, 'PENDING')''',
        (room_id, user_id, room_id, user_id, room_id,
user_id),
    )
    else:
        self.conn.execute(
            '''INSERT INTO DocumentUploadRequests (RoomId,
RequestedByUserId, Status)
VALUES (?, ?, 'PENDING')
ON CONFLICT(RoomId, RequestedByUserId) DO
UPDATE SET Status='PENDING', RequestedAt=CURRENT_TIMESTAMP,
DecidedAt=NULL, DecidedByUserId=NULL''',
            (room_id, user_id),
        )
        return self.conn.execute(
            'SELECT * FROM DocumentUploadRequests WHERE RoomId=?
AND RequestedByUserId=?',
            (room_id, user_id),
        ).fetchone()

    def pending_requests_for_room(self, room_id: int) ->
list[Any]:
        full_name = self.full_name_sql('u')
        return self.conn.execute(
            f'''SELECT dur.*, {full_name} AS RequestedByName,
r.Title AS RoomTitle, r.JoinCode
FROM DocumentUploadRequests dur
JOIN Users u ON u.UserId=dur.RequestedByUserId
JOIN Rooms r ON r.RoomId=dur.RoomId
WHERE dur.RoomId=? AND dur.Status='PENDING'
ORDER BY dur.RequestedAt DESC''',
            (room_id,),
        ).fetchall()

    def decide_access(self, request_id: int, decider_id: int,
status: str) -> Any | None:
        self.conn.execute(
            '''UPDATE DocumentUploadRequests SET Status=?,
DecidedAt=CURRENT_TIMESTAMP, DecidedByUserId=? WHERE
RequestId=?''',
            (status, decider_id, request_id),
        )
        return self.conn.execute('SELECT * FROM
DocumentUploadRequests WHERE RequestId=?',

```

```
(request_id,)).fetchone()
```

```

    def create_annotation(self, document_id: int, user_id: int,
                           page_number: int, annotation_type: str, coordinates: dict,
                           comment_text: str | None = None, selected_text: str | None = None)
    -> Any:
        annotation_type = annotation_type.upper()
        if annotation_type == 'QUESTION':
            existing = self.conn.execute(
                '''SELECT AnnotationId FROM DocumentAnnotations
                    WHERE DocumentId=? AND CreatedByUserId=? AND
                    PageNumber=? AND AnnotationType='QUESTION' AND IsDeleted=0 AND
                    IsResolved=0''',
                (document_id, user_id, page_number),
            ).fetchone()
            if existing:
                self.conn.execute(
                    '''UPDATE DocumentAnnotations
                        SET CoordinatesJson=?, CommentText=?,
                        SelectedText=?, UpdatedAt=CURRENT_TIMESTAMP
                        WHERE AnnotationId=?''',
                    (json.dumps(coordinates, ensure_ascii=False),
                     comment_text, selected_text, int(existing['AnnotationId']),
                    )
                )
            return
        self.get_annotation(int(existing['AnnotationId']))
        annotation_id = self.insert_and_get_id(
            'DocumentAnnotations',
            'AnnotationId',
            ['DocumentId', 'CreatedByUserId', 'PageNumber',
            'AnnotationType', 'CoordinatesJson', 'CommentText',
            'SelectedText'],
            (document_id, user_id, page_number, annotation_type,
            json.dumps(coordinates, ensure_ascii=False), comment_text,
            selected_text),
        )
        return self.get_annotation(annotation_id)

    def get_annotation(self, annotation_id: int) -> Any:
        full_name = self.full_name_sql('u')
        return self.conn.execute(
            f'''SELECT a.*, {full_name} AS AuthorName
                FROM DocumentAnnotations a JOIN Users u ON
                u.UserId=a.CreatedByUserId
                WHERE a.AnnotationId=?''',
            (annotation_id,),
        ).fetchone()

    def list_annotations(self, document_id: int, page: int | None
    = None) -> list[dict]:
        full_name = self.full_name_sql('u')
        active_filter = "a.IsDeleted=0 AND
        (a.AnnotationType!='QUESTION' OR a.IsResolved=0)"

```

```

        if page is None:
            rows = self.conn.execute(
                f'''SELECT a.*, {full_name} AS AuthorName
                    FROM DocumentAnnotations a JOIN Users u ON
u.UserId=a.CreatedByUserId
                    WHERE a.DocumentId=? AND {active_filter} ORDER
BY a.CreatedAt ASC''',
                (document_id,),
            ).fetchall()
        else:
            rows = self.conn.execute(
                f'''SELECT a.*, {full_name} AS AuthorName
                    FROM DocumentAnnotations a JOIN Users u ON
u.UserId=a.CreatedByUserId
                    WHERE a.DocumentId=? AND a.PageNumber=? AND
{active_filter} ORDER BY a.CreatedAt ASC''',
                (document_id, page),
            ).fetchall()
        return [self.annotation_to_dict(r) for r in rows]

    def resolve_question(self, annotation_id: int,
resolver_user_id: int) -> Any | None:
        row = self.get_annotation(annotation_id)
        if not row or row['AnnotationType'] != 'QUESTION' or
int(row['IsDeleted']) == 1:
            return None
        self.conn.execute(
            '''UPDATE DocumentAnnotations
                SET IsResolved=1, ResolvedByUserId=?,
ResolvedAt=CURRENT_TIMESTAMP, UpdatedAt=CURRENT_TIMESTAMP
                WHERE AnnotationId=?''',
            (resolver_user_id, annotation_id),
        )
        return self.get_annotation(annotation_id)

    def clear_annotations(self, document_id: int, page: int, user:
CurrentUser, clear_questions: bool = True) -> None:
        question_filter = ' ' if clear_questions else " AND
AnnotationType!='QUESTION'"
        if user.is_admin:
            self.conn.execute(f'UPDATE DocumentAnnotations SET
IsDeleted=1, UpdatedAt=CURRENT_TIMESTAMP WHERE DocumentId=? AND
PageNumber=?{question_filter}', (document_id, page))
        else:
            self.conn.execute(f'UPDATE DocumentAnnotations SET
IsDeleted=1, UpdatedAt=CURRENT_TIMESTAMP WHERE DocumentId=? AND
PageNumber=? AND CreatedByUserId=?{question_filter}',
(document_id, page, user.id))

    @staticmethod
    def annotation_to_dict(row: Any) -> dict:
        try:
            coords = json.loads(row['CoordinatesJson'] or '{}')

```

```

except json.JSONDecodeError:
    coords = {}
return {
    'id': int(row['AnnotationId']),
    'documentId': int(row['DocumentId']),
    'authorId': int(row['CreatedByUserId']),
    'authorName': row['AuthorName'],
    'page': int(row['PageNumber']),
    'type': row['AnnotationType'],
    'coordinates': coords,
    'text': row['CommentText'] or '',
    'selectedText': row['SelectedText'] or '',
    'createdAt': str(row['CreatedAt']),
    'resolved': bool(row['IsResolved']) if 'IsResolved' in
row.keys() else False,
    'resolvedByUserId': int(row['ResolvedByUserId']) if
'ResolvedByUserId' in row.keys() and row['ResolvedByUserId'] is
not None else None,
    'resolvedAt': str(row['ResolvedAt']) if 'ResolvedAt'
in row.keys() and row['ResolvedAt'] is not None else None,
}

```

Листинг А.7 – Основные фрагменты файла app/routers/rooms_router.py

```

from __future__ import annotations

import json
import secrets
from pathlib import Path
from typing import Any
from urllib.parse import quote

from fastapi import APIRouter, Body, File, Form, Request,
UploadFile, WebSocket, WebSocketDisconnect
from fastapi.responses import FileResponse, JSONResponse,
RedirectResponse, PlainTextResponse
from fastapi.templating import Jinja2Templates

from app.core import config
from app.core.database import db
from app.core.security import get_current_user, sessions,
create_room_page_token, get_user_by_room_page_token
from app.repositories.document_repository import
DocumentRepository
from app.repositories.room_repository import RoomRepository,
STATUS_LABELS
from app.services.signaling_service import signaling_service

router = APIRouter()
templates = Jinja2Templates(directory=str(config.TEMPLATES_DIR))

def _resolve_room_user(request: Request):
    token = request.query_params.get('room_token') or

```

```

request.headers.get('X-Room-Token')
    return get_user_by_room_page_token(token) or
get_current_user(request)

@router.get('/rooms/{code}')
def room_page(request: Request, code: str):
    user = _resolve_room_user(request)
    if not user:
        return RedirectResponse('/login', status_code=303)
    with db.session() as conn:
        room_repo = RoomRepository(conn)
        doc_repo = DocumentRepository(conn)
        room = room_repo.get_room_by_code(code.upper())
        if not room:
            return templates.TemplateResponse('error.html',
{'request': request, 'user': user, 'message': 'Комната не
найдена'}, status_code=404)
        if not room_repo.can_open_room(room, user):
            if room['Status'] == 'PLANNED':

room_repo.subscribe_user_to_room(int(room['RoomId']), user.id)
            return templates.TemplateResponse('error.html',
{'request': request, 'user': user, 'message': 'Комната
запланирована. Она добавлена в список ожидания и станет доступна
после активации.'}, status_code=403)
            return templates.TemplateResponse('error.html',
{'request': request, 'user': user, 'message': f"Комната
недоступна. Текущий статус: {room['StatusLabel']}"},
status_code=403)
        state = room_repo.get_room_state(int(room['RoomId']))
        documents = doc_repo.list_documents(int(room['RoomId']))
        current_document =
doc_repo.get_document(state['CurrentDocumentId']) if
state['CurrentDocumentId'] else None
        messages = room_repo.list_messages(int(room['RoomId']))
        events = room_repo.list_events(int(room['RoomId']))
        pending_requests =
doc_repo.pending_requests_for_room(int(room['RoomId'])) if
room_repo.can_manage_room(room, user) else []
        can_upload_pdf = doc_repo.can_upload_document(room, user)
        upload_request_status =
doc_repo.upload_request_status(int(room['RoomId']), user)
        room_token = create_room_page_token(user)
        document_url =
f"/documents/{current_document['DocumentId']}/file?room_token={roo
m_token}" if current_document else None
    return templates.TemplateResponse('room.html', {
        'request': request,
        'user': user,
        'room': room,
        'state': state,
        'documents': documents,

```

```

        'current_document': current_document,
        'access_status': 'APPROVED' if current_document else
'NO_DOCUMENT',
        'room_token': room_token,
        'participants': [],
        'messages': messages,
        'events': events,
        'pending_requests': pending_requests,
        'can_upload_pdf': can_upload_pdf,
        'upload_request_status': upload_request_status,
        'document_url': document_url,
    })

@router.post('/rooms/{code}/documents')
async def upload_document(request: Request, code: str, pdf_file:
UploadFile = File(...)):
    user = _resolve_room_user(request)
    if not user:
        return JSONResponse({'error': 'unauthorized'},
status_code=401)
    code = code.upper()
    with db.session() as conn:
        room_repo = RoomRepository(conn)
        doc_repo = DocumentRepository(conn)
        room = room_repo.get_room_by_code(code)
        if not room or room['Status'] != 'ACTIVE':
            return JSONResponse({'error': 'room_not_active'},
status_code=403)
        if not doc_repo.can_upload_document(room, user):
            return JSONResponse({'error':
'upload_permission_required'}, status_code=403)
        original_name = Path(pdf_file.filename or
'document.pdf').name
        if not original_name.lower().endswith('.pdf'):
            return JSONResponse({'error': 'not_pdf'},
status_code=400)
        data = await pdf_file.read()
        if not data.startswith(b'%PDF'):
            return JSONResponse({'error': 'not_pdf'},
status_code=400)
        stored_name = f'{secrets.token_urlsafe(12)}.pdf'
        target = config.UPLOAD_DIR / 'documents' / stored_name
        target.parent.mkdir(parents=True, exist_ok=True)
        target.write_bytes(data)
        document = doc_repo.create_document(int(room['RoomId']),
user.id, original_name, stored_name, f'documents/{stored_name}',
len(data))
        room_repo.set_current_document(int(room['RoomId']),
int(document['DocumentId']), user.id)
        room_repo.set_current_page(int(room['RoomId']), 1, None,
user.id)
        document_payload = {

```

```

        'id': int(document['DocumentId']),
        'fileName': document['OriginalFileName'],
        'url': f"/documents/{int(document['DocumentId'])}/file",
        'page': 1,
    }
    await signaling_service.broadcast(code, {'type': 'document-
updated', 'document': document_payload})
    await signaling_service.broadcast(code, {'type': 'room-event',
'event': {'EventType': f"Загружен PDF:
{document['OriginalFileName']}", 'CreatedAt': '', 'PayloadJson':
'{}'}}))
    return {'document': document_payload}

@router.post('/rooms/{code}/documents/upload-request')
async def request_upload_permission(request: Request, code: str):
    user = _resolve_room_user(request)
    if not user:
        return RedirectResponse('/login', status_code=303)
    code = code.upper()
    with db.session() as conn:
        room_repo = RoomRepository(conn)
        doc_repo = DocumentRepository(conn)
        room = room_repo.get_room_by_code(code)
        if not room:
            return
            RedirectResponse('/dashboard?error=room_not_found',
status_code=303)
        req =
doc_repo.request_upload_permission(int(room['RoomId']), user.id)
        room_repo.add_event(int(room['RoomId']), user.id,
'UPLOAD_REQUESTED', {'requestId': int(req['RequestId'])})
        await signaling_service.broadcast(code, {'type': 'upload-
requested', 'message': f'{user.full_name} запросил разрешение на
загрузку PDF'})
        return RedirectResponse(f'/rooms/{code}', status_code=303)

@router.get('/documents/{document_id}/file')
def document_file(request: Request, document_id: int):
    user = _resolve_room_user(request)
    if not user:
        return RedirectResponse('/login', status_code=303)
    with db.session() as conn:
        repo = DocumentRepository(conn)
        document = repo.get_document(document_id)
        if not document or not
repo.user_can_read_document(document, user):
        return JSONResponse({'error': 'Нет доступа к
документу'}, status_code=403)
        path = config.UPLOAD_DIR / str(document['FilePath'])
        if not path.exists():
            return JSONResponse({'error': 'Файл не найден'},
status_code=404)
        encoded = quote(str(document['OriginalFileName']))

```

```

    return FileResponse(
        path,
        media_type='application/pdf',
        headers={'Content-Disposition': f"inline; filename*=UTF-8''{encoded}"},
    )

@router.post('/rooms/{code}/access/{request_id}/{decision}')
async def decide_access(request: Request, code: str, request_id: int, decision: str):
    user = _resolve_room_user(request)
    if not user:
        return RedirectResponse('/login', status_code=303)
    status = 'APPROVED' if decision == 'approve' else 'REJECTED'
    code = code.upper()
    with db.session() as conn:
        room_repo = RoomRepository(conn)
        doc_repo = DocumentRepository(conn)
        room = room_repo.get_room_by_code(code)
        if room and room_repo.can_manage_room(room, user):
            doc_repo.decide_access(request_id, user.id, status)
            room_repo.add_event(int(room['RoomId']), user.id,
                'UPLOAD_REQUEST_DECIDED', {'requestId': request_id, 'status':
                status})
            await signaling_service.broadcast(code, {'type': 'access-
                decided', 'requestId': request_id, 'status': status})
            return RedirectResponse(f'/rooms/{code}', status_code=303)

@router.post('/api/rooms/{code}/page')
def set_page(request: Request, code: str, payload: dict[str, Any]
    = Body(...)):
    user = _resolve_room_user(request)
    if not user:
        return JSONResponse({'error': 'unauthorized'},
            status_code=401)
    page = max(1, int(payload.get('page') or 1))
    with db.session() as conn:
        repo = RoomRepository(conn)
        room = repo.get_room_by_code(code.upper())
        if not room:
            return JSONResponse({'error': 'not found'},
                status_code=404)
        if not repo.can_manage_room(room, user):
            return JSONResponse({'error': 'forbidden'},
                status_code=403)
        repo.set_current_page(int(room['RoomId']), page, None,
            user.id)
    return {'ok': True, 'page': page}

@router.get('/api/documents/{document_id}/annotations')
def get_annotations(request: Request, document_id: int, page: int
    = 1):
    user = _resolve_room_user(request)

```

```

        if not user:
            return JSONResponse({'error': 'unauthorized'},
status_code=401)
        with db.session() as conn:
            repo = DocumentRepository(conn)
            document = repo.get_document(document_id)
            if not document or not
repo.user_can_read_document(document, user):
                return JSONResponse({'error': 'forbidden'},
status_code=403)
            annotations = repo.list_annotations(document_id, page)
            return {'annotations': annotations}

@router.post('/api/documents/{document_id}/annotations')
async def create_annotation(request: Request, document_id: int,
payload: dict[str, Any] = Body(...)):
    user = _resolve_room_user(request)
    if not user:
        return JSONResponse({'error': 'unauthorized'},
status_code=401)
    page = max(1, int(payload.get('page') or 1))
    kind = str(payload.get('type') or 'MARK').upper()
    type_map = {'MARKER': 'MARK', 'MARK': 'MARK', 'HIGHLIGHT':
'HIGHLIGHT', 'NOTE': 'COMMENT', 'COMMENT': 'COMMENT', 'QUESTION':
'QUESTION'}
    annotation_type = type_map.get(kind, 'MARK')
    coordinates = payload.get('coordinates') or {}
    text = str(payload.get('text') or '')[:500]
    with db.session() as conn:
        doc_repo = DocumentRepository(conn)
        room_repo = RoomRepository(conn)
        document = doc_repo.get_document(document_id)
        if not document or not
doc_repo.user_can_read_document(document, user):
            return JSONResponse({'error': 'forbidden'},
status_code=403)
        room = room_repo.get_room(int(document['RoomId']))
        if not room:
            return JSONResponse({'error': 'room_not_found'},
status_code=404)
        if user.is_viewer and annotation_type != 'QUESTION':
            return JSONResponse({'error':
'viewer_can_only_question'}, status_code=403)
        if not user.is_viewer and annotation_type == 'QUESTION':
            return JSONResponse({'error':
'speaker_uses_annotations'}, status_code=403)
        if not user.is_viewer and not
room_repo.can_manage_room(room, user):
            return JSONResponse({'error': 'forbidden'},
status_code=403)
        annotation = doc_repo.create_annotation(document_id,
user.id, page, annotation_type, coordinates, text)
        annotation_dict = doc_repo.annotation_to_dict(annotation)

```

```

        event_name = 'QUESTION_CREATED' if annotation_type ==
'QUESTION' else 'PDF_ANNOTATION_CREATED'
        room_repo.add_event(int(room['RoomId']), user.id,
event_name, {'annotationId': annotation_dict['id'], 'type':
annotation_type})
        await
signaling_service.broadcast(str(room['JoinCode']).upper(),
{'type': 'annotation-created', 'annotation': annotation_dict})
        await
signaling_service.broadcast(str(room['JoinCode']).upper(),
{'type': 'room-event', 'event': {'EventType': 'Задан вопрос' if
annotation_type == 'QUESTION' else 'Добавлена аннотация',
'CreatedAt': '', 'PayloadJson': '{}'}}})
        return {'annotation': annotation_dict}

@router.post('/api/documents/{document_id}/annotations/{annotation
_id}/resolve')
async def resolve_question(request: Request, document_id: int,
annotation_id: int):
    user = _resolve_room_user(request)
    if not user:
        return JSONResponse({'error': 'unauthorized'},
status_code=401)
    with db.session() as conn:
        doc_repo = DocumentRepository(conn)
        room_repo = RoomRepository(conn)
        document = doc_repo.get_document(document_id)
        if not document or not
doc_repo.user_can_read_document(document, user):
            return JSONResponse({'error': 'forbidden'},
status_code=403)
        room = room_repo.get_room(int(document['RoomId']))
        if not room or not room_repo.can_manage_room(room, user):
            return JSONResponse({'error': 'forbidden'},
status_code=403)
        annotation = doc_repo.get_annotation(annotation_id)
        if not annotation or int(annotation['DocumentId']) !=
int(document_id) or annotation['AnnotationType'] != 'QUESTION':
            return JSONResponse({'error': 'question_not_found'},
status_code=404)
        resolved = doc_repo.resolve_question(annotation_id,
user.id)
        if not resolved:
            return JSONResponse({'error': 'question_not_found'},
status_code=404)

@router.websocket('/ws/rooms/{code}')
async def room_socket(websocket: WebSocket, code: str):
    room_token = websocket.query_params.get('room_token')
    user = get_user_by_room_page_token(room_token)
    if user is None:
        token = websocket.cookies.get(config.SESSION_COOKIE_NAME)
        user = sessions.get(token or '')

```

```

    if user is None:
        await websocket.close(code=1008)
        return
    peer_id = websocket.query_params.get('peer_id') or
secrets.token_urlsafe(10)
    code = code.upper()
    with db.session() as conn:
        room_repo = RoomRepository(conn)
        doc_repo = DocumentRepository(conn)
        room = room_repo.get_room_by_code(code)
        if not room or room['Status'] != 'ACTIVE':
            await websocket.close(code=1008)
            return
        room_repo.join_room(room, user)
        state = room_repo.get_room_state(int(room['RoomId']))
        current_document =
doc_repo.get_document(state['CurrentDocumentId']) if
state['CurrentDocumentId'] else None
        messages = room_repo.list_messages(int(room['RoomId']))
        existing_peers = await signaling_service.connect(code,
peer_id, websocket, user)
        online = await signaling_service.participants(code)
        await websocket.send_json({
            'type': 'connected',
            'peerId': peer_id,
            'peers': existing_peers,
            'participants': online,
            'state': dict(state),
            'document': {'id': int(current_document['DocumentId']),
'fileName': current_document['OriginalFileName'], 'url':
f"/documents/{int(current_document['DocumentId'])}/file" if
current_document else None,
            'messages': [dict(m) for m in messages],
        })
        await signaling_service.broadcast(code, {'type': 'peer-
joined', 'peer': {'peerId': peer_id, 'userId': user.id,
'fullName': user.full_name, 'initials': user.initials, 'role':
user.role_code, 'roleName': user.role_name}, 'participants':
online}, exclude_peer_id=peer_id)
        await signaling_service.broadcast(code, {'type': 'room-event',
'event': {'EventType': f'{user.full_name} подключился',
'CreatedAt': '', 'PayloadJson': '{}'}})
    try:
        while True:
            raw = await websocket.receive_text()
            try:
                payload: dict[str, Any] = json.loads(raw)
            except json.JSONDecodeError:
                continue
            message_type = payload.get('type')
            target_peer_id = payload.get('targetPeerId')
            if message_type in {'offer', 'answer', 'ice-
candidate'} and target_peer_id:

```



```

        payload['sourceName'] = user.full_name
        await signaling_service.broadcast(code, payload,
exclude_peer_id=peer_id)
    except WebSocketDisconnect:
        pass
    finally:
        await signaling_service.disconnect(code, peer_id)
        online = await signaling_service.participants(code)
        if not await signaling_service.has_user_in_room(code,
user.id):
            with db.session() as conn:
                room_repo = RoomRepository(conn)
                room = room_repo.get_room_by_code(code)
                if room:
                    room_repo.leave_room(int(room['RoomId']),
user)
            await signaling_service.broadcast(code, {'type': 'peer-
left', 'peerId': peer_id, 'userId': user.id, 'participants':
online})
            await signaling_service.broadcast(code, {'type': 'room-
event', 'event': {'EventType': f'{user.full_name} вышел',
'CreatedAt': '', 'PayloadJson': '{}'}})

```

Приведенные фрагменты отражают основную серверную логику приложения: создание и открытие комнат, разграничение прав пользователей, загрузку PDF-документа, обработку запросов на загрузку, синхронизацию страницы, работу с аннотациями и рассылку событий участникам комнаты.

Приложение Б
(рекомендуемое)
Схемы и диаграммы информационной системы

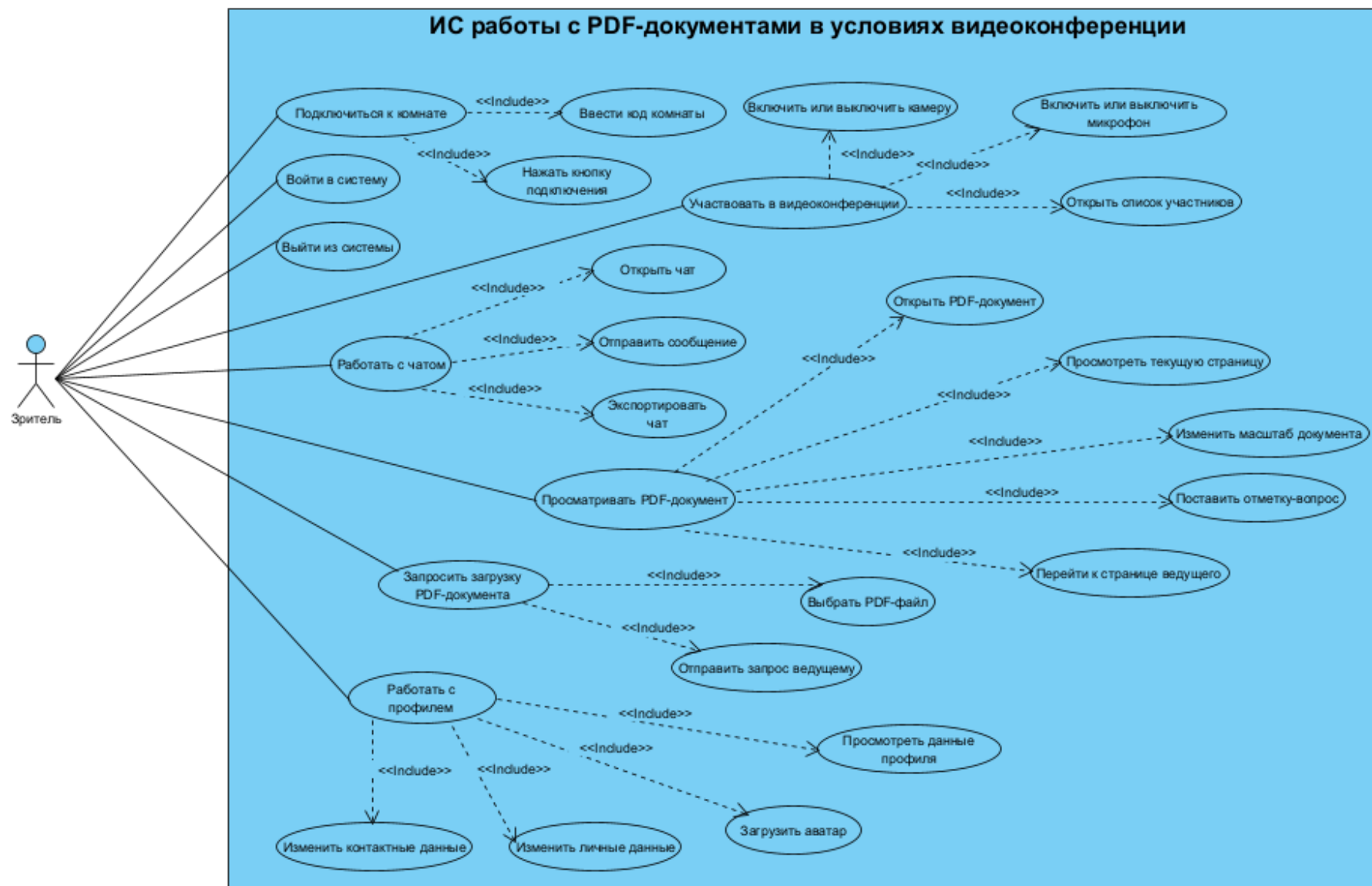


Рисунок Б.1 – Диаграмма вариантов использования зрителя

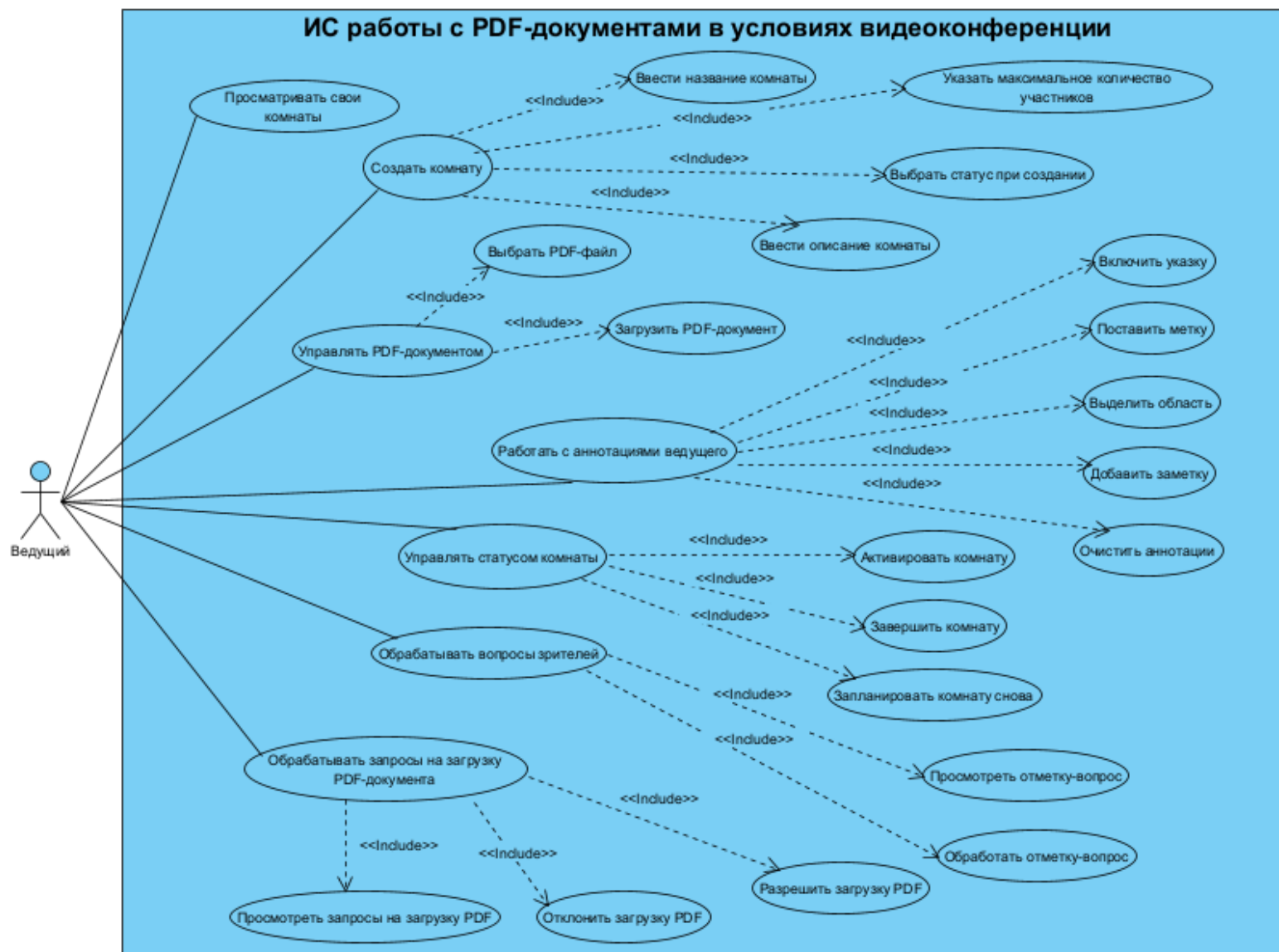


Рисунок Б.2 – Диаграмма вариантов использования ведущего

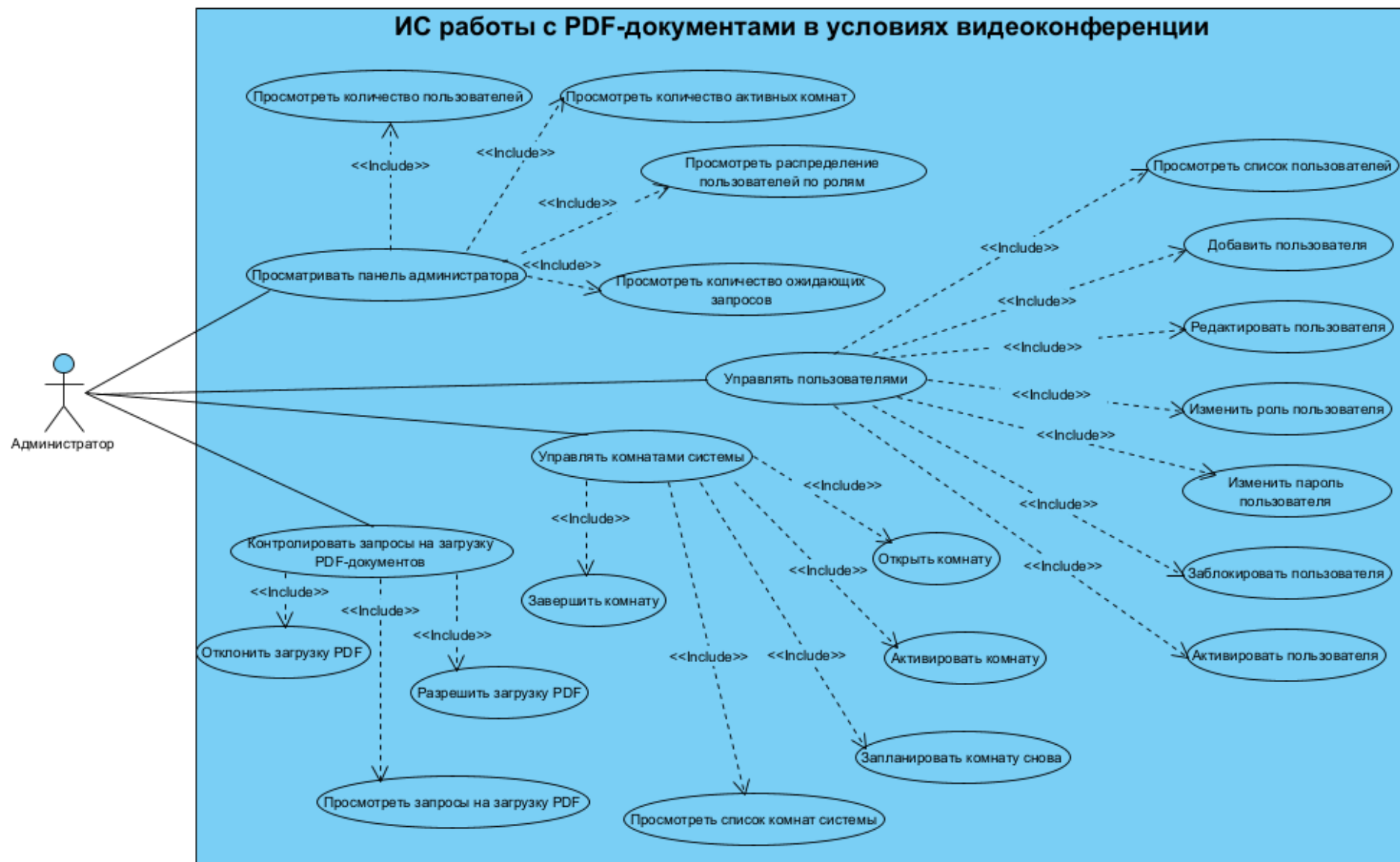


Рисунок Б.3 – Диаграмма вариантов использования администратора

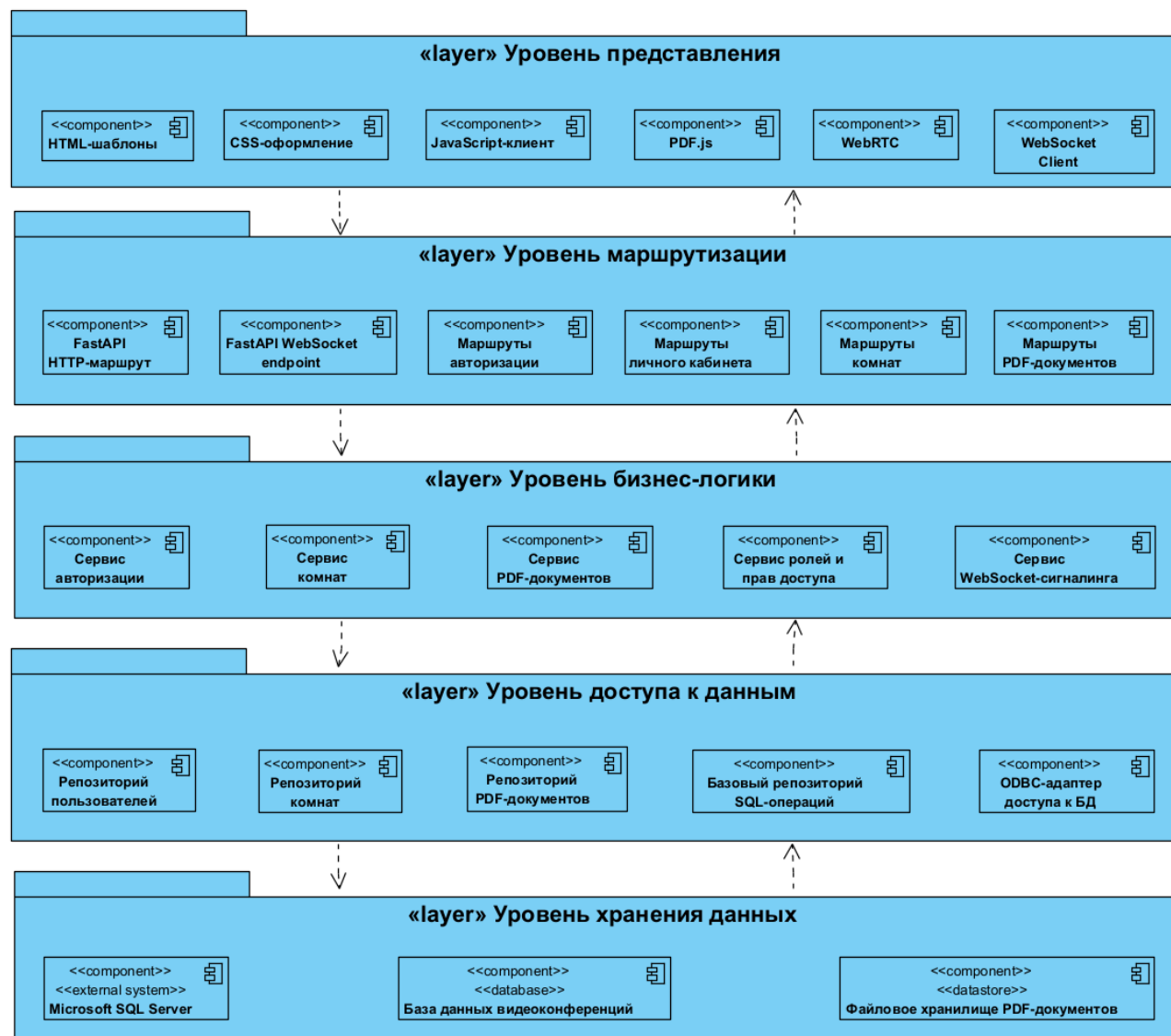


Рисунок Б.4 – Архитектура слоев информационной системы

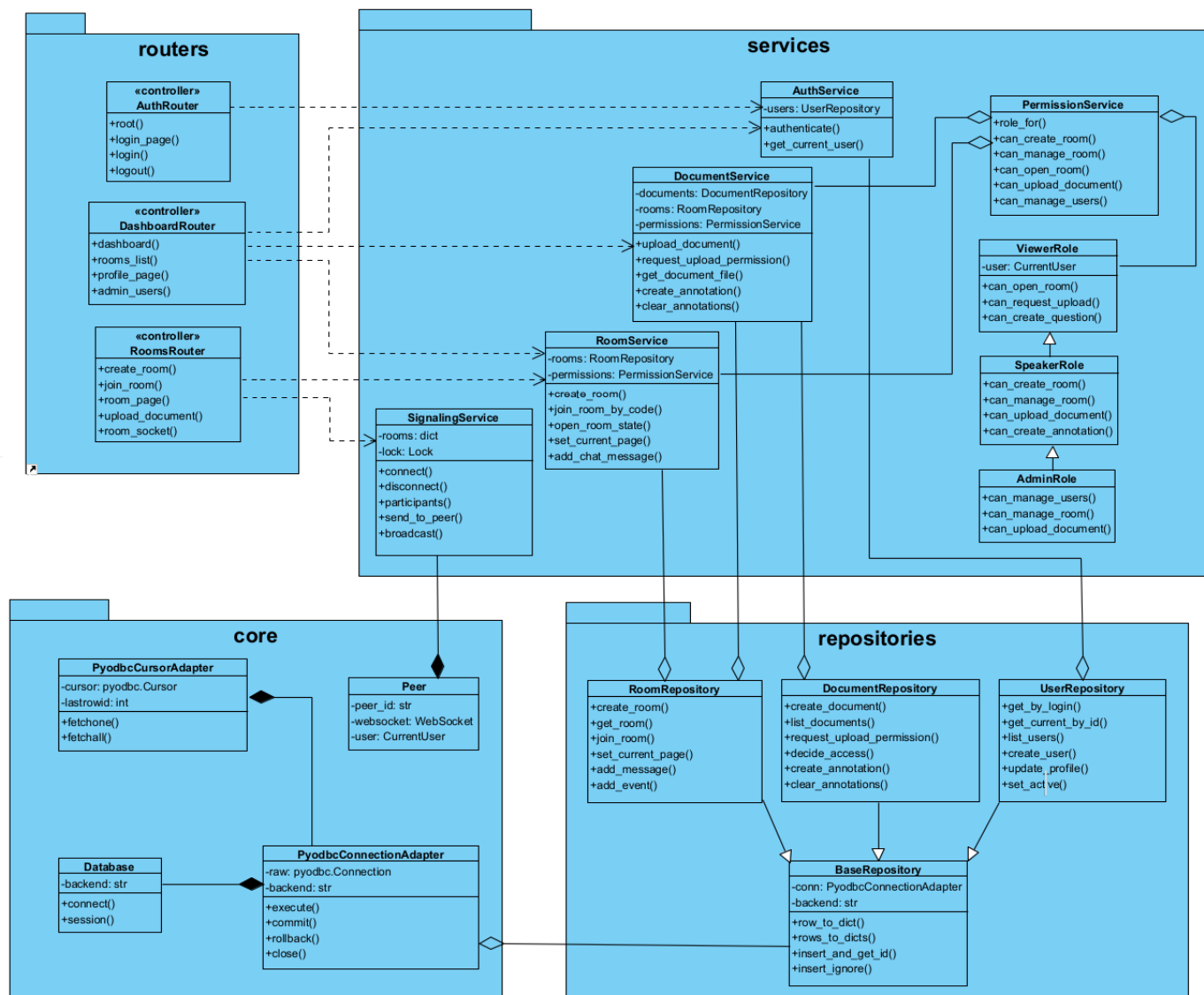


Рисунок Б.5 – Диаграмма классов информационной системы

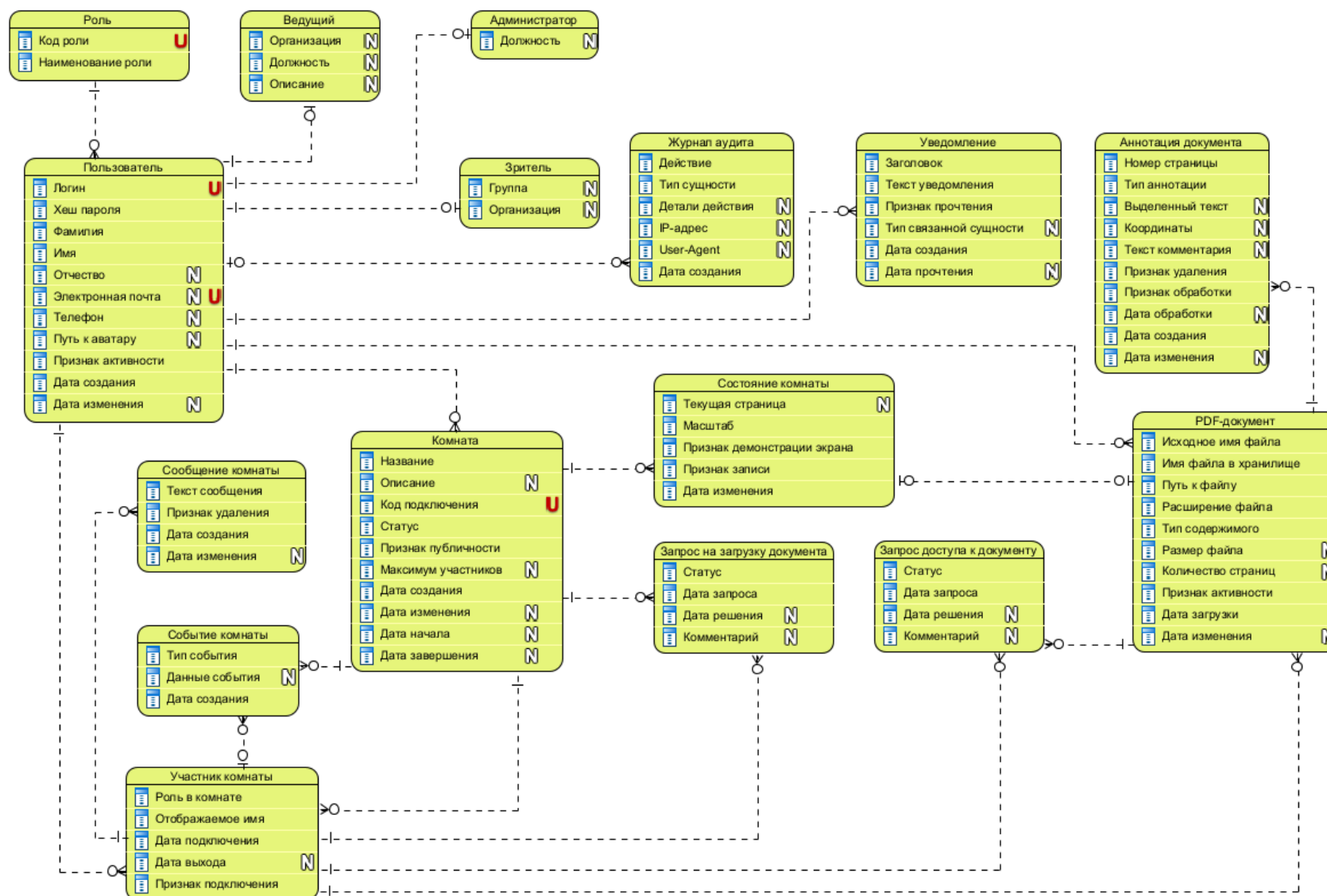


Рисунок Б.6 – ER-диаграмма концептуальной модели данных

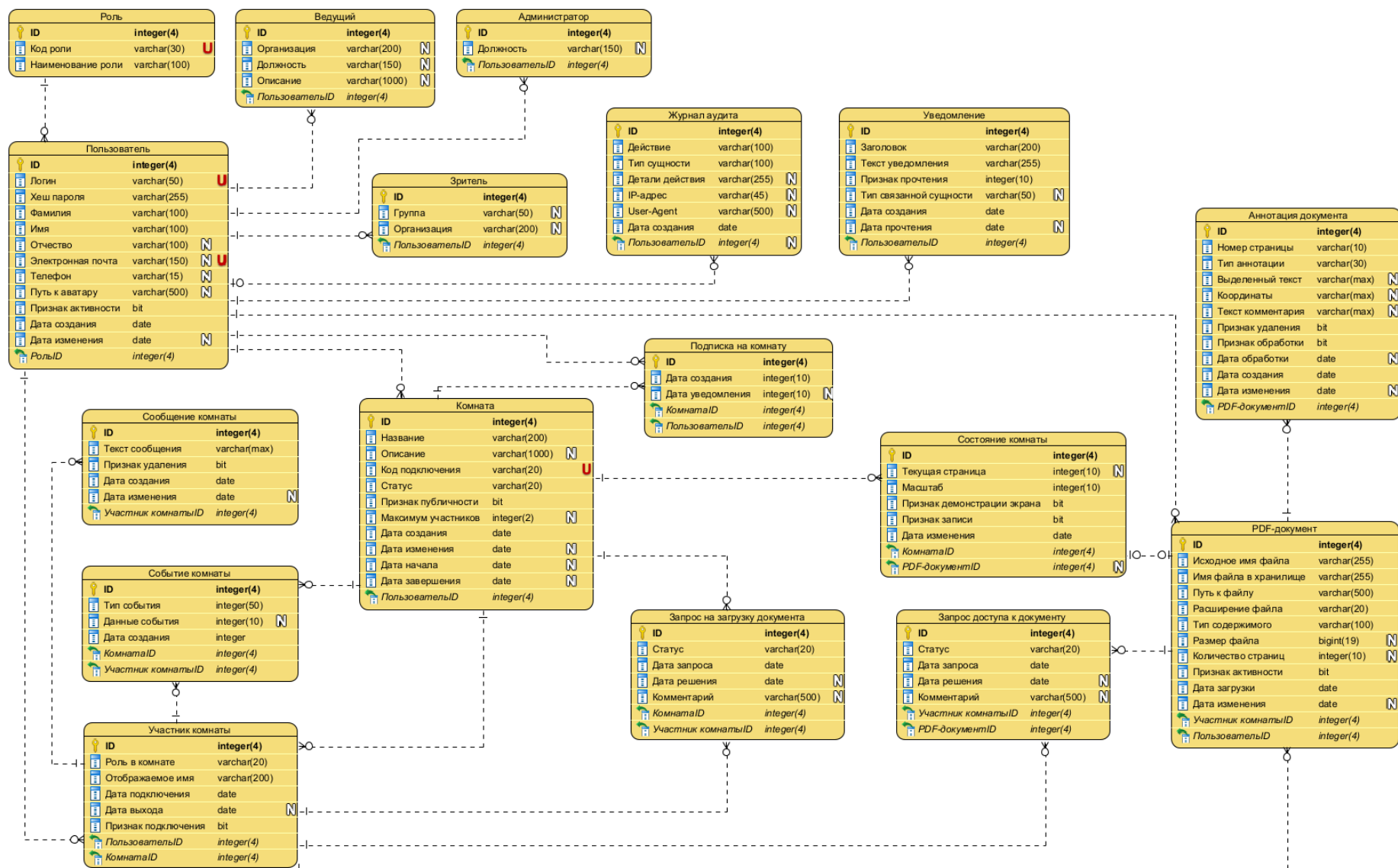


Рисунок Б.7 – ER-диаграмма логической модели данных

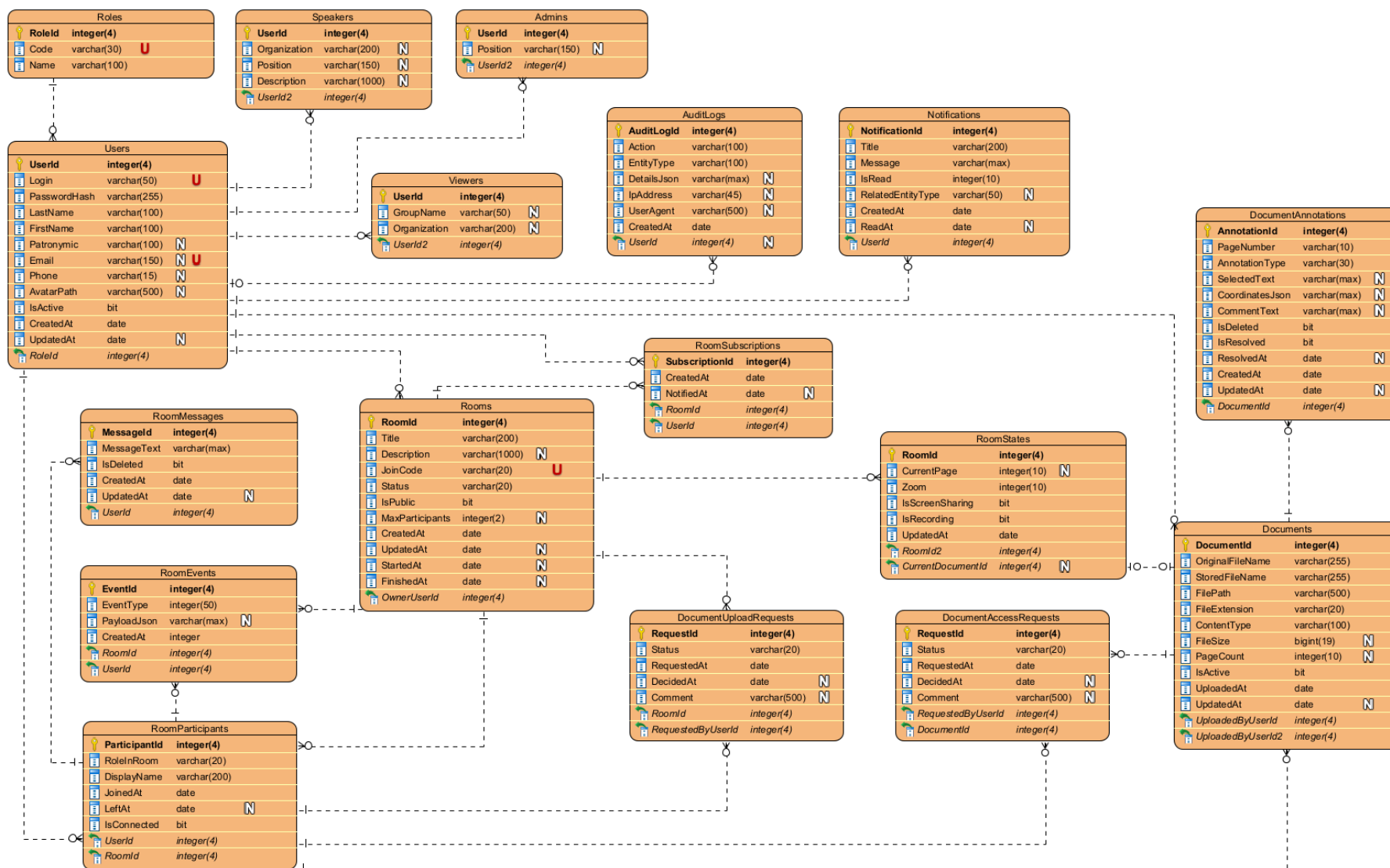


Рисунок Б.8 – ER-диаграмма физической модели данных

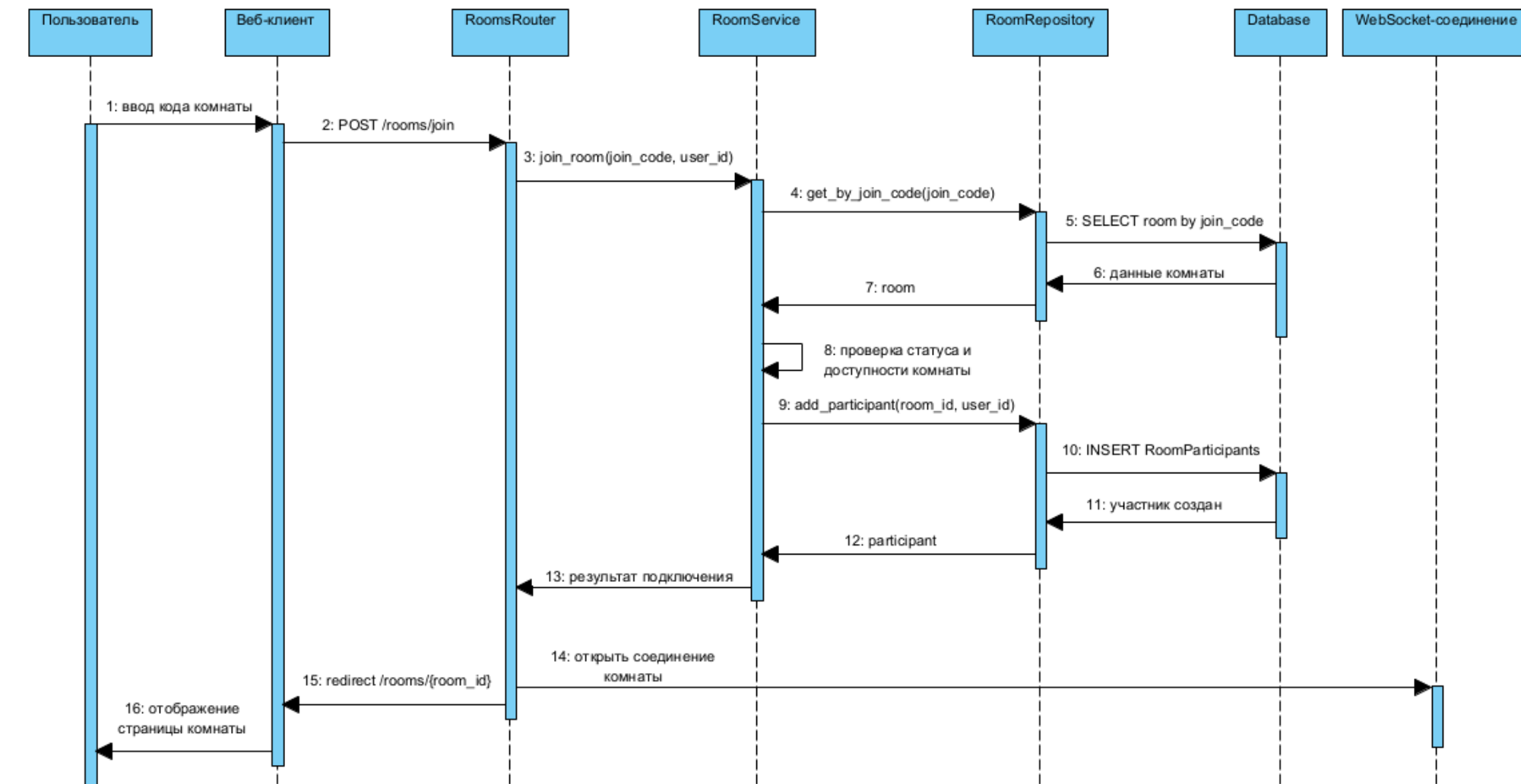


Рисунок Б.9 – Диаграмма последовательности подключения пользователя к комнате

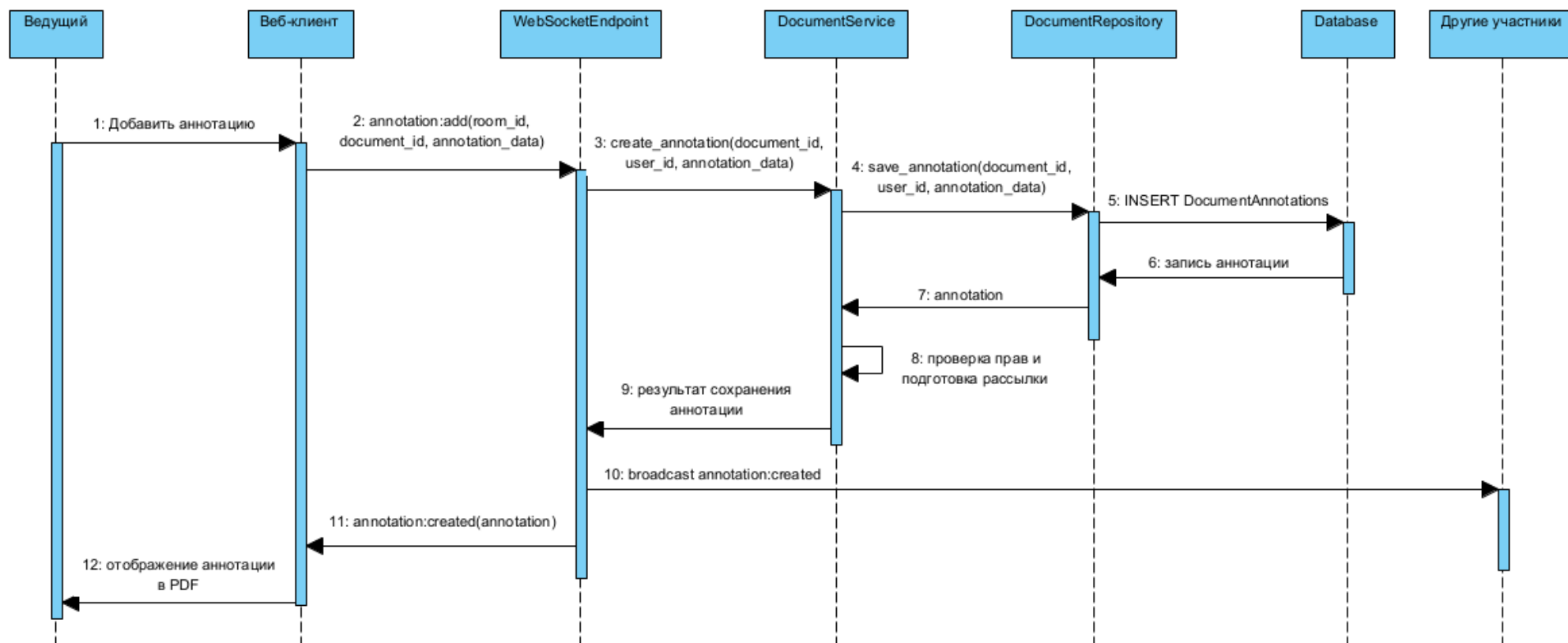


Рисунок Б.10 – Диаграмма последовательности загрузки PDF-документа в комнате

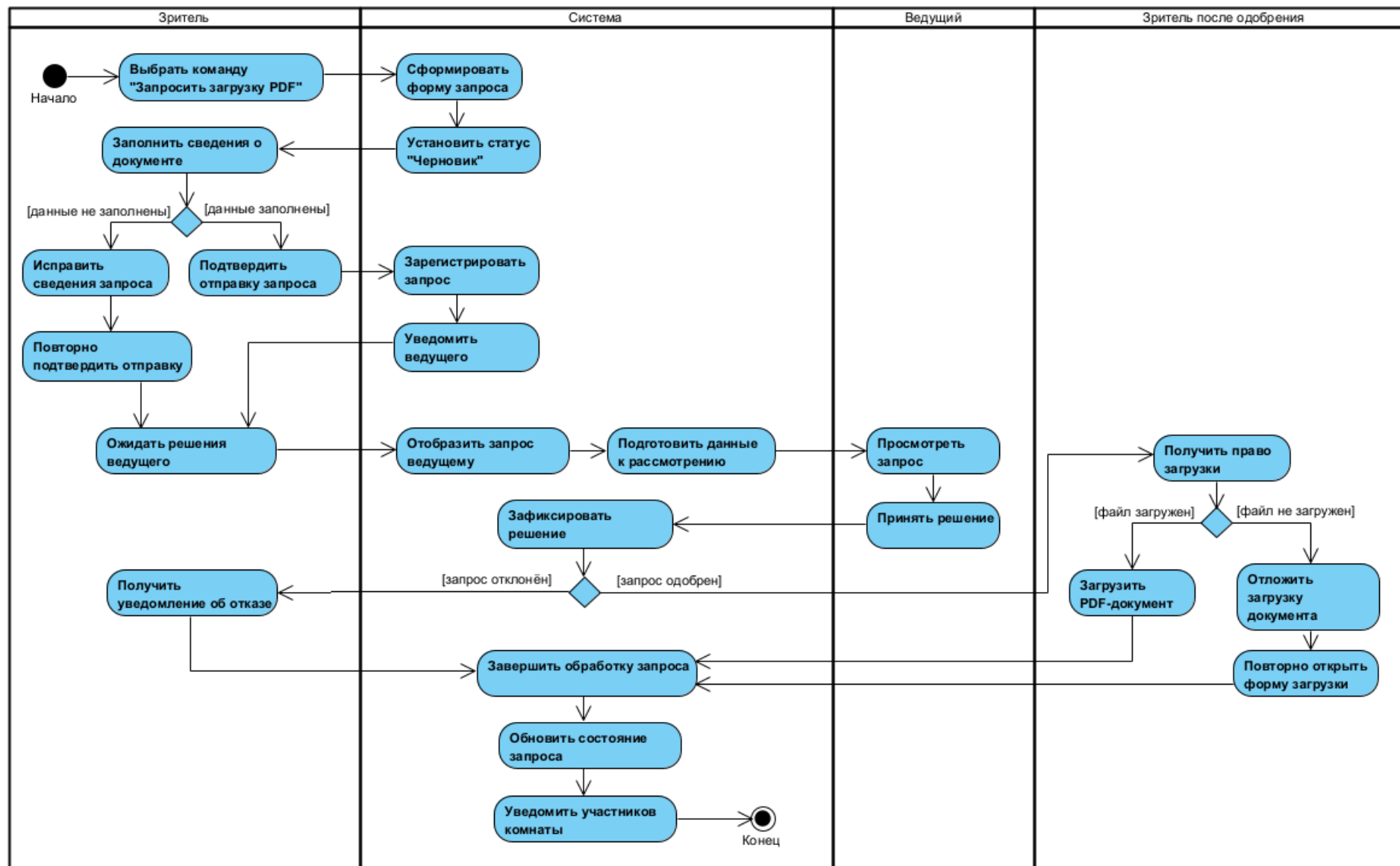


Рисунок Б.11 – Диаграмма активности процесса работы с PDF-документом

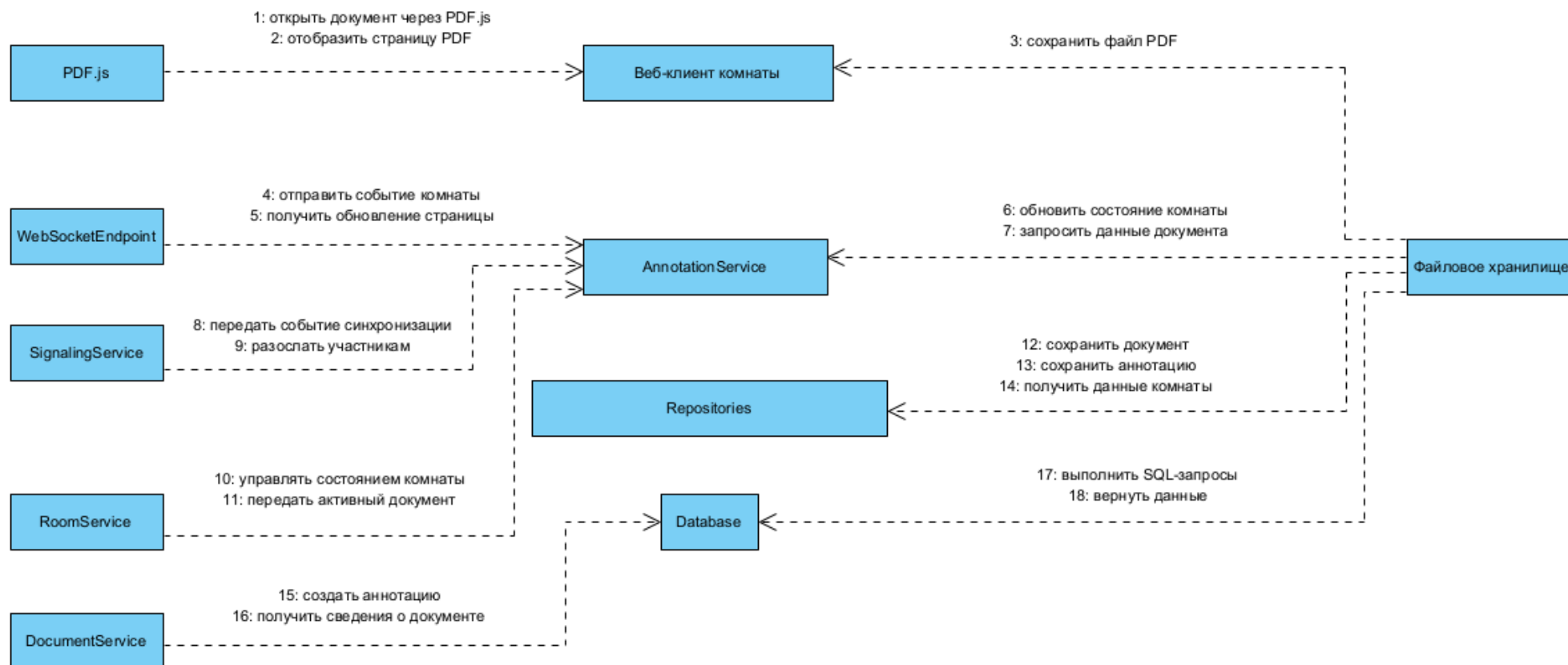


Рисунок Б.12 – Диаграмма кооперации участников при взаимодействии с PDF-документом

Крупные схемы и диаграммы приложения Б используются как графическое дополнение к разделам проектирования и реализации информационной системы.

Приложение В
(рекомендуемое)
Результаты тестирования

В приложении приведены фактические результаты тестирования разработанного веб-приложения. Перечень рисунков соответствует столбцам «Ссылка на результат» в таблицах раздела 4: рисунки В.1–В.7 относятся к тестированию методом черного ящика, а рисунок В.8 содержит результат выполнения автоматизированных модульных тестов и проверки независимых маршрутов алгоритма `upload_document`.

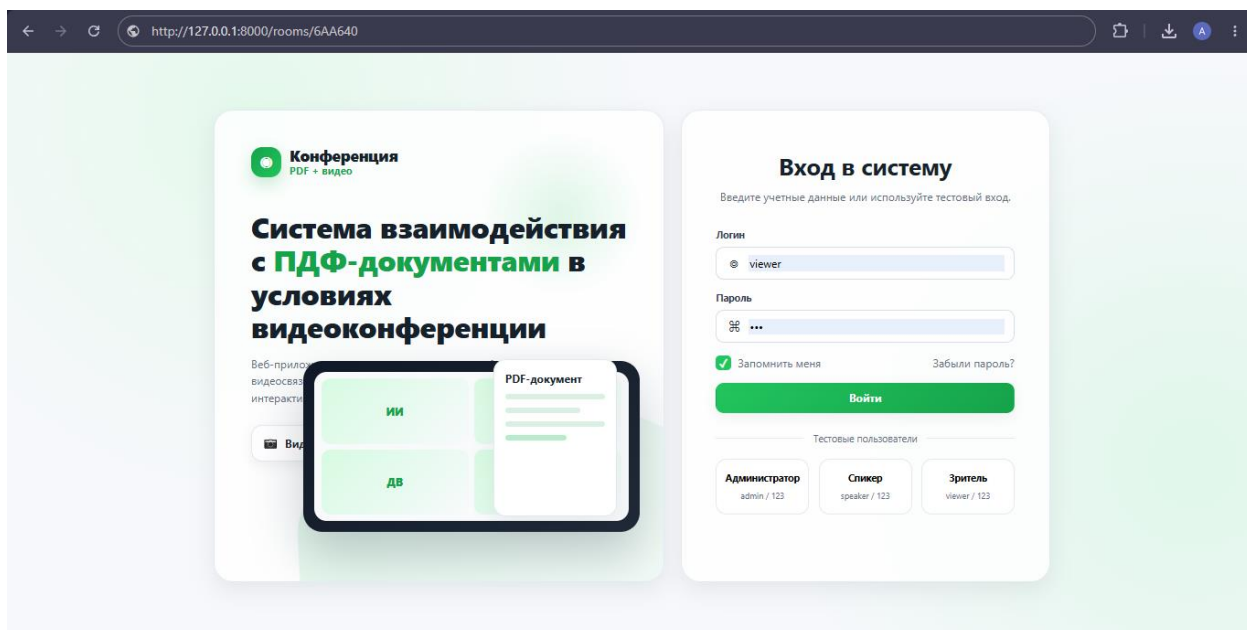


Рисунок В.1 – Результат проверки неавторизованного пользователя

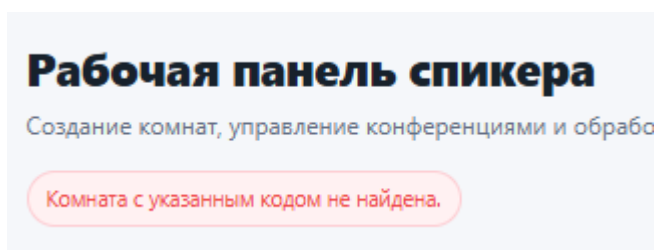


Рисунок В.2 – Результат проверки ввода кода несуществующей комнаты

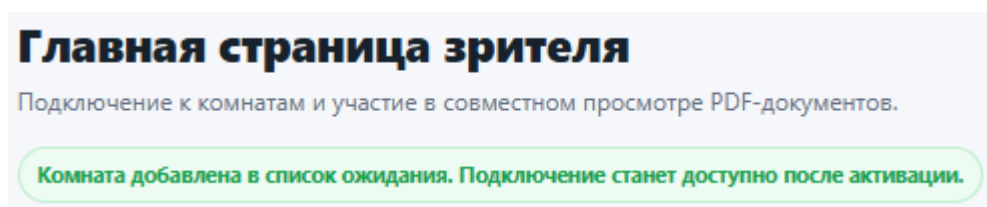


Рисунок В.3 – Результат проверки обращения к комнате со статусом PLANNED или FINISHED

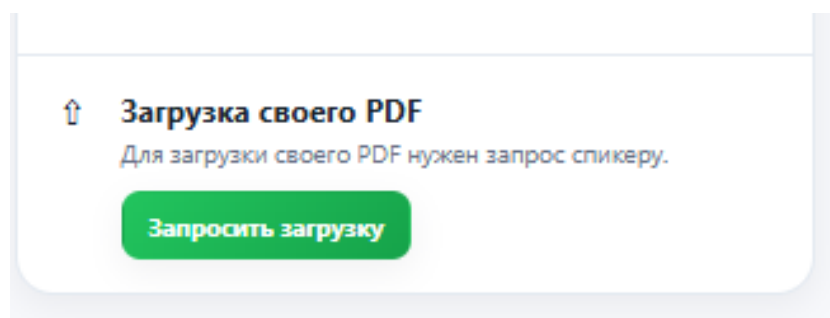


Рисунок В.4 – Результат проверки отсутствия разрешения на загрузку PDF у зрителя

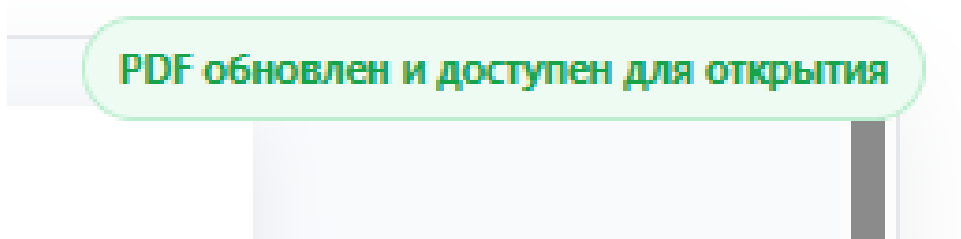


Рисунок В.5 – Результат загрузки корректного PDF-файла ведущим



Рисунок В.6 – Результат загрузки корректного PDF-файла зрителем с подтвержденным запросом

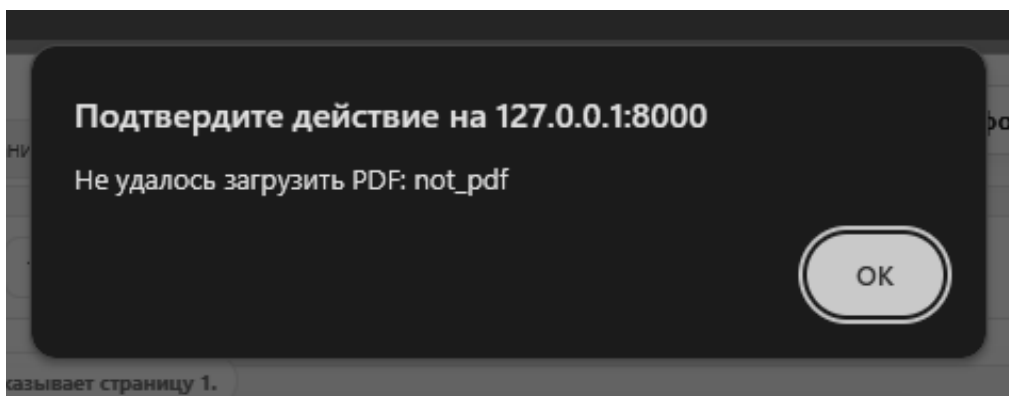


Рисунок В.7 – Результат отклонения файла с неверным расширением или сигнатурой

```

C:\Windows\System32\cmd.exe
plugins: anyio-4.13.0, asyncio-1.4.0
asyncio: mode=Mode.STRICT, debug=False, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=functio
n
collected 21 items

test_appendix_v_results.py::test_v01_black_box_unauthorized_user_is_redirected_to_login PASSED [ 4%]
test_appendix_v_results.py::test_v02_black_box_unknown_room_code_returns_room_not_found PASSED [ 9%]
test_appendix_v_results.py::test_v03_black_box_planned_or_finished_room_is_not_opened PASSED [ 14%]
test_appendix_v_results.py::test_v04_black_box_viewer_without_upload_permission_gets_request_required PASSED [ 19%]
test_appendix_v_results.py::test_v05_black_box_speaker_uploads_valid_pdf PASSED [ 23%]
test_appendix_v_results.py::test_v06_black_box_approved_viewer_uploads_valid_pdf PASSED [ 28%]
test_appendix_v_results.py::test_v07_black_box_wrong_extension_or_signature_is_rejected PASSED [ 33%]
test_appendix_v_results.py::test_v08_unit_resolve_room_user_returns_user_from_room_token PASSED [ 38%]
test_appendix_v_results.py::test_v09_unit_can_upload_document_checks_admin_owner_and_approved_viewer PASSED [ 42%]
test_appendix_v_results.py::test_v10_unit_upload_document_success_creates_document_and_event PASSED [ 47%]
test_appendix_v_results.py::test_v11_unit_upload_document_without_pdf_extension_returns_not_pdf PASSED [ 52%]
test_appendix_v_results.py::test_v12_unit_set_page_for_viewer_returns_forbidden PASSED [ 57%]
test_appendix_v_results.py::test_v13_unit_create_annotation_question_by_viewer PASSED [ 61%]
test_appendix_v_results.py::test_v14_unit_resolve_question_by_speaker PASSED [ 66%]
test_appendix_v_results.py::test_v15_upload_document_route_without_authorization_returns_unauthorized PASSED [ 71%]
test_appendix_v_results.py::test_v16_upload_document_route_without_room_returns_room_not_active PASSED [ 76%]
test_appendix_v_results.py::test_v17_upload_document_route_inactive_room_returns_room_not_active PASSED [ 80%]
test_appendix_v_results.py::test_v18_upload_document_route_without_permission_returns_upload_permission_required PASSED [ 85%]
test_appendix_v_results.py::test_v19_upload_document_route_wrong_extension_returns_not_pdf PASSED [ 90%]
test_appendix_v_results.py::test_v20_upload_document_route_wrong_pdf_signature_returns_not_pdf PASSED [ 95%]
test_appendix_v_results.py::test_v21_upload_document_route_success_returns_document_payload PASSED [100%]

===== warnings summary =====
..\..\Users\plave\AppData\Local\Python\pythoncore-3.14-64\lib\site-packages\fastapi\routing.py:233: 17 warnings

```

Рисунок В.8 – Результат выполнения автоматизированных тестов функций и маршрута `upload_document`

Приложение В предназначено только для размещения результатов, на которые есть ссылки в таблицах раздела тестирования. Дополнительные скриншоты без ссылок из таблиц в приложение не добавляются.