МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ

УНИВЕРСИТЕТ (РИНХ)

Факультет **Компьютерных технологий и информационной безопасности**

Кафедра **Информационных технологий и программирования**

ДОПУСТИТЬ К ЗАЩИТЕ

Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_

к.э.н., доцент Е.В. Ефимова

« » \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**на тему:**

**«РАЗРАБОТКА ПЛАТФОРМЫ ДЛЯ КОРПОРАТИВНОГО ОБУЧЕНИЯ С ИСКУССТВЕННЫМ ИНТЕЛЛЕКТОМ»**

Выполнил А.Д.Горбоносова

студент группы ПРИ-342 \_\_\_\_\_\_\_\_\_\_

подпись

Направление

09.03.04 «Программная инженерия»

Направленность

09.03.04.01 «Системное и прикладное программное обеспечение»

Руководитель выпускной

квалификационной работы Е.В.Жилина

*к.э.н., доцент*  \_\_\_\_\_\_\_\_

подпись

Ростов-на-Дону

2025

ФГБОУ ВО «РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ

ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ (РИНХ)»

Факультет **Компьютерных технологий и информационной безопасности**

Кафедра **Информационных технологий и программирования**

«УТВЕРЖДАЮ»

Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_

доцент, к.э.н. Е.В. Ефимова

« » \_\_\_\_\_\_\_\_\_ 2025 г.

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы**

Обучающегося Горбоносовой Анны Дмитриевны группы ПРИ-342

1. Тема выпускной квалификационной работы: «Разработка платформы для корпоративного обучения с искусственным интеллектом».
2. Срок сдачи студентом законченной ВКР на кафедру «\_\_\_» \_\_\_\_\_2025 г.
3. Исходные данные для ВКР предоставлены ИП Миргородским Д.Ю., 344092, г. Ростов-на-Дону, ул. Стартовая, д. 16/1, кв. 113
4. Структура ВКР

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

3 РАСЧЕТНАЯ ЧАСТЬ

Дата выдачи задания «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись Ф. И. О.

Задание к исполнению принял \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись Ф.И.О. обучающегося

РЕФЕРАТ

Дипломная работа содержит 89 с., 47 рис., 13 табл., 39 источников, 0 прил.

LMS-ПЛАТФОРМА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, SPRING BOOT, FASTAPI, NEXT.JS, WHISPER, MISTRAL AI.

Выпускная квалификационная работа посвящена разработке корпоративной образовательной платформы с использованием технологий искусственного интеллекта.

Объектом исследования является алгоритм автоматизации работы с учебными материалами в корпоративной среде, охватывающий процессы их создания, обновления, распространения и контроля.

Предметом исследования выступает программный инструментарий, применяемый для реализации платформы: стек технологий, архитектурные решения, библиотеки и нейросетевые модели, обеспечивающие создание и управление обучающими материалами.

Целью работы является разработка масштабируемой LMS-платформы, интегрированной с внутренними процессами организации-заказчика — ОАО «ГлобалСофт».

Актуальность работы обусловлена необходимостью повышения эффективности управления обучающим контентом и сокращения затрат за счёт замены дорогостоящих лицензируемых решений собственной системой, адаптированной под корпоративную инфраструктуру.

Для реализации проекта использован современный стек технологий: Spring Boot (Kotlin), FastAPI (Python), Next.js (TypeScript), а также нейросетевые модели Whisper и Mistral AI. Разработка включала проектирование архитектуры, реализацию интерфейсов, интеграцию с LDAP и средствами хранения, организацию административной панели и систему ролей.

Экономическая оценка показала высокую эффективность внедрения: система достигает окупаемости менее чем за 10 месяцев. Анализ надёжности подтвердил достижение показателя 93,6% безотказной работы.

Результаты работы подтверждают целесообразность внедрения собственной LMS-платформы как конкурентоспособной альтернативы коммерческим решениям.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc199589812)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc199589813)

[1.1 Описание ИП «Миргородский Д.Ю.» и организации-заказчика 7](#_Toc199589814)

[1.2 Анализ существующих решений 8](#_Toc199589815)

[1.3 Выбор программного инструментария для разработки 11](#_Toc199589816)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 19](#_Toc199589817)

[2.1 Проектирование архитектуры системы 19](#_Toc199589818)

[2.2 Реализация системы и интеграция компонентов 36](#_Toc199589819)

[2.3 Тестирование и анализ корректности работы системы 64](#_Toc199589820)

[2.4 Организация и управление рабочим процессом 71](#_Toc199589821)

[3 РАСЧЁТНАЯ ЧАСТЬ 74](#_Toc199589822)

[3.1 Оценка экономической эффективности внедрения 74](#_Toc199589823)

[3.2 Расчёт показателей надёжности 76](#_Toc199589824)

[3.3 Анализ рисков 79](#_Toc199589825)

[ЗАКЛЮЧЕНИЕ 81](#_Toc199589826)

[СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ 83](#_Toc199589827)

# ВВЕДЕНИЕ

В современных условиях корпоративное обучение играет ключевую роль в повышении эффективности и конкурентоспособности организаций. Актуальность разработки корпоративной образовательной платформы с использованием искусственного интеллекта обусловлена необходимостью решения накопившихся проблем управления учебными материалами в компании-заказчике — ОАО «ГлобалСофт».

В компании за долгие годы сформировался обширный портфель программных продуктов с сопроводительной документацией и обучающими материалами, но рост ассортимента и разнообразие форматов хранения усложняют централизованное управление, обновление контента и оценку эффективности обучения. Платформа ориентирована как на конечных пользователей — сотрудников организаций-заказчиков, так и на специалистов, ответственных за создание и распространение учебных материалов, для которых она упрощает подготовку и актуализацию контента.

Существующие на рынке корпоративные LMS-решения часто не удовлетворяют специфическим требованиям компаний, при этом стоимость лицензирования может быть значительной, что ограничивает их применение. Разработка собственной платформы позволит обеспечить глубокую интеграцию с внутренними продуктами и бизнес-процессами, гибкую настройку функционала, высокий уровень безопасности и удобство обновления учебного контента.

Цель работы — разработка масштабируемой LMS-платформы с использованием искусственного интеллекта, автоматизирующей создание, обновление и распространение учебных материалов, а также обеспечивающей контроль и анализ результатов обучения.

Для достижения цели поставлены следующие задачи:

1. Анализ деятельности организации и требований к системе обучения, исследование существующих LMS-решений и выявление их недостатков;
2. Проектирование архитектуры и интерфейсов платформы, разработка и внедрение программного продукта с использованием современных технологий и методов искусственного интеллекта;
3. Проведение экономической оценки целесообразности внедрения, расчёт показателей надёжности, анализ рисков и разработка рекомендаций по обеспечению устойчивой работы системы.

Объектом исследования в данной работе является алгоритм автоматизации работы с учебными материалами, который охватывает поэтапные процессы создания, обновления, распространения и контроля учебного контента в корпоративной среде.

Предметом исследования выступает инструментарий разработки — набор программных технологий и библиотек, использованных для реализации данного алгоритма. В частности, серверная часть построена на Spring Boot с использованием Kotlin, для сервисов обработки данных применён FastAPI на Python, фронтенд разработан с помощью Next.js, а для автоматизации создания учебного контента интегрированы нейросетевые модели Whisper и Mistral AI. Кроме того, в работе применяются методы системного анализа, архитектурного проектирования с использованием UML, интеграции с корпоративными сервисами и тестирования программного продукта, что обеспечивает комплексный подход к реализации и оценке разрабатываемой платформы.

Первая глава включает описание деятельности организации-заказчика, анализ существующих решений и выбор программного инструментария для разработки, что обосновывает необходимость создания собственной платформы и определяет техническую базу проекта. Во второй главе рассмотрены проектирование, реализация и тестирование системы на контрольном примере, а также организация рабочего процесса с использованием канбан-доски. Третья глава посвящена экономической оценке эффективности внедрения, расчёту показателей надёжности системы и анализу рисков, связанных с её эксплуатацией.

Таким образом, выполнение данной работы направлено на создание инновационного инструмента для корпоративного обучения, способствующего повышению качества подготовки пользователей продуктов компании и оптимизации внутренних образовательных процессов.

Дипломная работа состоит из введения, трёх глав, заключения и списка использованных источников.

# 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

# Описание ИП «Миргородский Д.Ю.» и организации-заказчика

Материалы по организации-заказчику для дипломного исследования были собраны в ходе преддипломной практики в ИП «Миргородский Д.Ю.». Данная организация специализируется на предоставлении комплексных услуг в сфере IT-разработки. Основным направлением деятельности ИП является разработка компьютерного программного обеспечения, что соответствует коду ОКВЭД 62.01. Команда ИП успешно применяет аутсорсинговую модель разработки, обеспечивающую эффективное управление проектами различной сложности и масштаба.

В дальнейшем по тексту индивидуальный предприниматель «Миргородский Д.Ю.» будет кратко обозначен как ИП.

Команда ИП включает в себя менеджера, full-stack разработчиков, мобильных разработчиков, а также приглашённых специалистов в зависимости от специфики проекта.

Аутсорсинговая модель разработки программного обеспечения позволяет управлять составом команды с учетом потребностей текущих проектов, быстро подключать узкоспециализированных специалистов и эффективно реализовывать как краткосрочные, так и долгосрочные проекты.

ИП придерживается гибких методологий разработки, разбивая работу на спринты, что позволяет эффективно планировать задачи, контролировать прогресс разработки и своевременно получать обратную связь от заказчика. Для управления проектами используются различные инструменты, в зависимости от проекта и потребностей заказчика, в их числе Asana, Trello и Redmine, позволяющие сохранять постоянную связь между разработчиками и стейкхолдерами проекта.

Разработка типичного проекта включает в себя несколько этапов, на каждом из которых происходит тесное взаимодействие с заказчиком. На первом этапе формируются цели проекта, изучается предметная область. Затем проводится анализ требований, и на основе этого разрабатывается план, который определяет ключевые этапы разработки. Важным моментом является комплектование команды, которая будет работать над проектом, в зависимости от технических задач. Далее проект реализуется по спринтовой модели, что позволяет на каждом этапе проходить через проектирование, разработку, тестирование и выпуск MVP продукта. После завершения разрабатываемый продукт сопровождается и поддерживается до вывода из эксплуатации.

Опыт команды, наличие проверенных методик и подходов, а также гибкость в работе с заказчиками обеспечивают высокое качество создаваемых решений. Благодаря этому ИП успешно конкурирует на рынке аутсорсинговых услуг и демонстрирует устойчивый рост проектной активности.

Разработка настоящего проекта осуществлялась для компании ОАО «ГлобалСофт». Компания занимается разработкой программного обеспечения и имеет широкую клиентскую базу, охватывающую различные отрасли. Её решения ориентированы на решение узкоспециализированных задач в таких областях, как ТЭК, строительство, девелопмент, ЖКХ, торговля, склад, логистика и других. Компания разрабатывает и поддерживает отраслевые решения, которые соответствуют специфическим требованиям этих секторов.

Основная деятельность компании включает разработку, внедрение, установку и настройку программного обеспечения, а также обучение пользователей и его дальнейшее сопровождение. Она предлагает комплексные решения «под ключ», начиная с анализа и проектирования и заканчивая полным запуском системы в эксплуатацию.

Компания активно расширяет линейку своих продуктов, ориентируясь на современные тенденции рынка и потребности клиентов. Разработка данного проекта является частью стратегии расширения компании и направлена на улучшение доступности и удобства использования её решений.

# Анализ существующих решений

На текущий момент в компании «ГлобалСофт» накопилось значительное количество программных продуктов, охватывающих различные сферы деятельности. К каждому из решений прилагаются сопроводительные материалы — инструкции, руководства пользователя, обучающие видео и вспомогательная документация. Однако с течением времени и ростом продуктовой линейки управление этими материалами стало представлять всё большую сложность.

Каждое обновление программного обеспечения требует соответствующего обновления документации и своевременного донесения этих изменений до конечных пользователей — как действующих сотрудников компаний, которым предоставляются услуги, так и новых, проходящих обучение. В условиях, когда инструкции хранятся в различных форматах и расположены в разных источниках, таких как локальные хранилища, сетевые папки и видеоролики в мессенджерах, возникают следующие проблемы:

* трудно отследить, кто и когда получил доступ к материалам
* нет единого источника актуальной информации
* при увольнении сотрудников теряются цепочки коммуникации, а вместе с ними и история доступа к обучающим материалам
* обновления не доходят до всех пользователей своевременно, что приводит к ошибкам при работе с ПО
* невозможно объективно оценить, как продвигается обучение, какие разделы вызывают затруднения, и насколько эффективно осваиваются новые функции

Кроме того, отслеживание прогресса пользователей в обучении, сбор обратной связи и решение возникающих вопросов происходит вручную, через неструктурированные каналы, включая чаты, звонки, устные обсуждения, что не позволяет систематизировать информацию и эффективно её использовать для улучшения качества обучения.

В процессе выбора подходящей платформы для управления корпоративным обучением в компании «ГлобалСофт» были рассмотрены современные коммерческие LMS-решения, такие как iSpring Learn, Teachbase, Unicraft и Motivity. Эти системы ориентированы на корпоративный сегмент, поддерживают как облачную, так и коробочную установку, и предназначены в основном для обслуживания крупных компаний [1].

Однако при сравнении функциональных возможностей, гибкости настройки и совокупной стоимости владения было установлено, что ни одно из этих решений не в полной мере удовлетворяет требованиям компании. Основными ограничениями при использовании указанных платформ стала их ограниченная адаптация под специфические внутренние бизнес-процессы компании. Кроме того, большинство решений не позволяют реализовать глубокую кастомизацию ролевой модели и логики обучения, что критично для корпоративного использования. Существенным недостатком также является высокая стоимость лицензирования и сопровождения, особенно при длительном использовании и масштабировании системы.

В связи с этим было принято решение о разработке собственной обучающей платформы, которая с самого начала проектировалась с учётом внутренних требований компании, включая интеграцию с существующими ИТ-системами, масштабируемость и контроль над данными. Полные расчёты экономической эффективности данного подхода, включая анализ совокупной стоимости владения и срок окупаемости, приведены в разделе 3.1. Сравнительный анализ решений по ключевым характеристикам представлен в таблице 1.1.

Таблица 1. – Сравнительная характеристика существующих локальных решений по сравнению с собственным решением

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Характеристика | iSpring Learn | Teachbase | Unicraft | Motivity | Собственная платформа |
| Тип развертывания | Облако / Коробка | Облако / Коробка | Облако / Коробка | Облако / Коробка | Локально, в ИТ-инфраструктуре компании |
| Гибкость настройки | Ограниченная, в рамках UI и API [1] | Средняя, часть логики недоступна | Средняя, требует обращения к разработчику | Средняя, без расширяемой архитектуры | Максимальная, учитывает все внутренние процессы |
| Интеграция | Через API, ограниченно | Ограниченная интеграция | Частичная интеграция | Через стандартные коннекторы [1] | Полная интеграция с внутренними системами |
| Поддержка кастомной логики | Частичная | Частичная | Ограниченная | Ограниченная | Полная свобода в построении логики и интерфейсов |
| Контроль над данными | Вариативный (зависит от развертывания) | Вариативный | Вариативный | Вариативный | Полный контроль, хранение на собственных серверах |
| Стоимость | Высокая при масштабировании | Высокая | Умеренная | Наивысшая среди рассматриваемых | Минимальные эксплуатационные расходы |
| Масштабируемость | Ограничена лицензией | Ограничена лицензией | Ограничена лицензией | Ограничена лицензией | Гибкая, не требует дополнительных затрат |

Таким образом, в процессе выбора подходящего решения для управления обучением было принято решение разработать собственную систему, исходя из нескольких ключевых факторов. Коммерческие платформы, такие как iSpring Learn, Teachbase, Unicraft и Motivity, обладают широким функционалом и предлагают как облачные, так и коробочные установки, однако в контексте задач компании «ГлобалСофт» они оказались недостаточно гибкими. Их функциональность либо не учитывает особенности внутренних бизнес-процессов, либо требует значительных затрат на адаптацию. Кроме того, высокая стоимость лицензирования и сопровождения при длительном использовании и масштабировании делает эти решения экономически нецелесообразными.

В результате было принято стратегическое решение о создании собственной обучающей платформы, полностью интегрированной с корпоративной инфраструктурой и адаптированной под конкретные требования компании.

Собственное решение позволило бы создать экосистему, которая идеально интегрируется с уже существующими продуктами и процессами компании, без необходимости адаптации под внешний сервис. Кроме того, собственная система предоставляет полную гибкость в настройке функционала, что позволяет точно определить необходимые возможности и избежать лишних функций, которые могут создать дополнительную нагрузку на пользователей.

Также важно было учитывать аспекты безопасности и контроля данных. Собственная система гарантирует полный контроль над хранимой информацией, что особенно важно при работе с корпоративными данными и соблюдении требований безопасности.

Кроме того, такая система существенно упростит процесс создания обучающих материалов для сотрудников. Для этого им будет достаточно подготовить видеоинструкцию и откорректировать текстовые материалы, в то время как всю работу по распространению и актуализации контента система возьмёт на себя. Это позволит сократить затраты времени на обновление документации и повысить её доступность для всех пользователей, обеспечивая при этом более высокую степень прозрачности в процессе обучения.

В результате, собственная платформа для управления обучением и инструкциями представляется наиболее эффективным вариантом, так как она идеально подходит под внутренние процессы компании, предлагает максимальную гибкость и независимость, а также снижает риски, связанные с использованием внешних платформ.

# 

# Выбор программного инструментария для разработки

В процессе разработки проекта для ОАО «ГлобалСофт» была составлена майнд-карта, представленная на рисунке 1.1 и содержащая ключевые технологические направления системы: серверная часть или backend, клиентский интерфейс или frontend, база данных, а также модели машинного обучения. Такая декомпозиция позволяет последовательно спроектировать и реализовать функциональные компоненты системы. Отдельно стоит отметить наличие LDAP, используемого для авторизации и управления доступом. Его структура и поведение заданы корпоративными стандартами и предоставлены в тестовом варианте. Далее каждый из этих блоков будет подробно рассмотрен. Особое внимание будет уделено выбору между альтернативными технологиями для реализации каждого компонента — с учётом ограничений, предъявляемых корпоративной инфраструктурой, и требований к надёжности, расширяемости и удобству поддержки. Выбранные варианты отмечены жирным шрифтом.



Рисунок .1 – Майнд-карта инструментария разработки

Начнем с первого узла майнд-карты, который посвящён выбору технологий для серверной части. Этот узел делится на два подузла — один для технологий, отвечающих за пользовательскую часть бэкенда, и второй для связанной с нейросетями части. У каждого из подузлов собственные подузлы, содержимое которых будет описано далее. Более подробно backend-узел отражен на рисунке 1.2.

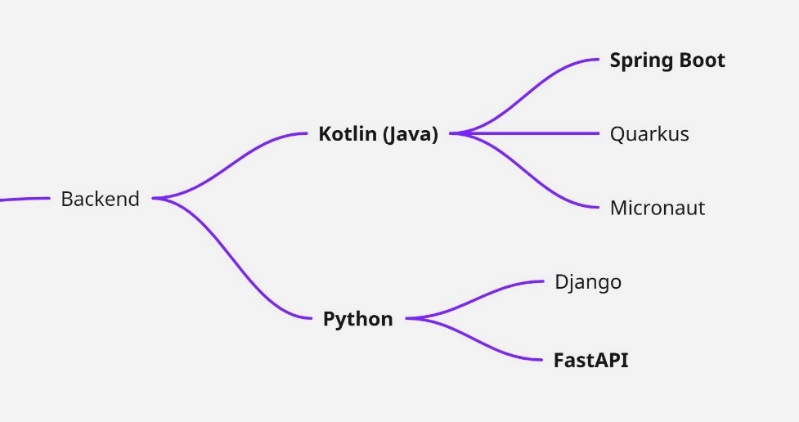


Рисунок 1.2 – Backend-узел майнд-карты инструментария

Для реализации функционала пользовательской части бэкенда было принято решение использовать Kotlin в сочетании с фреймворком Spring Boot. Альтернативные решения, такие как Quarkus и Micronaut, также были рассмотрены, но не выбраны по ряду причин, включая зрелость экосистемы и интеграцию с корпоративными стандартами. Сравним их в таблице 1.2.

Таблица 1.2 – Сравнительная характеристика веб-фреймворков на Kotlin (Java)

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Spring Boot | Quarkus | Micronaut |
| Основное назначение | Разработка полноценных веб-приложений и RESTful сервисов | Оптимизация для контейнеров и Kubernetes, микросервисы [2] | Разработка микросервисов с малым потреблением ресурсов [2] |
| Поддержка микросервисов | Хорошая поддержка, но не фокусируется исключительно на микросервисах | Отличная поддержка микросервисной архитектуры | Отличная поддержка, особенно для малых сервисов |
| Производительность | Средняя, может потреблять много памяти и времени на стартап | Высокая, оптимизирован для быстрого старта и низкого потребления ресурсов | Высокая, низкое потребление памяти и быстрый старт [2] |
| Интеграция с Kubernetes | Поддержка через Spring Cloud и Spring Boot Starter для Kubernetes | Встроенная поддержка Kubernetes и GraalVM для нативных приложений [2] | Поддержка Kubernetes, но требует дополнительных настроек |
| Обширность экосистемы | Огромная, много готовых решений и библиотек | Меньше, но активно развивается | Меньше, меньше библиотек и расширений |
| Поддержка сторонних библиотек и инструментов | Широкая поддержка, множество расширений | Меньше, но активно расширяется | Ограниченная поддержка сторонних библиотек [3] |
| Сообщество и документация | Очень большое сообщество, много материалов и примеров | Растущее сообщество, но меньше примеров и документации | Меньше материалов, сообщество только растёт |
| Гибкость | Высокая, можно настроить под любые задачи | Высокая, но ориентирован на облачные и контейнерные приложения [2] | Хорошая, но требует дополнительных настроек для сложных случаев [3] |

После анализа различных фреймворков для разработки серверной части было принято решение выбрать Spring Boot для реализации пользовательского бэкенда. Этот фреймворк является зрелым, с широким набором инструментов и решений, что обеспечивает высокую гибкость и возможности интеграции. Несмотря на существование более легковесных решений, таких как Quarkus и Micronaut, которые предлагают преимущества в плане производительности и быстрого старта, для текущих требований, где не ставится задача реализации микросервисной архитектуры, Spring Boot является оптимальным выбором. Он полностью удовлетворяет потребности проекта и гарантирует минимальные риски на всех этапах разработки.

Учитывая возможность масштабирования проекта в будущем, также была рассмотрена вероятность использование Kubernetes для оркестрации контейнеров.

Для реализации серверной части, работающей с нейросетями и машинным обучением, был выбран язык Python. Это решение обусловлено тем, что Python является наиболее подходящим языком для задач, связанных с искусственным интеллектом, благодаря богатой экосистеме библиотек и инструментов, специально разработанных для работы с нейронными сетями и анализом данных. Использование других языков программирования, таких как Java или C++, требует значительных усилий и дополнительной настройки для интеграции с нейросетевыми решениями, что делает Python более удобным и эффективным выбором для данного проекта.

Далее рассмотрим два популярных фреймворка для разработки серверной части на Python — FastAPI и Django. Сравнение представлено в таблице 1.3.

Таблица 1.3 – Сравнительная характеристика фреймворков на Python для разработки серверной части проекта

|  |  |  |
| --- | --- | --- |
| Критерий | FastAPI | Django |
| Основное назначение | Быстрая разработка API и асинхронных приложений | Полноценный фреймворк для создания веб-приложений с готовыми решениями |
| Производительность | Очень высокая, асинхронная обработка запросов, идеально подходит связанных с нейросетями задач [4] | Хорошая, но не оптимизирована для асинхронной работы в реальном времени |
| Поддержка асинхронности | Полная поддержка асинхронных запросов с использованием Python async/await | Ограниченная поддержка асинхронности через отдельные библиотеки [4] |
| Легкость интеграции с нейросетями | Отлично интегрируется с библиотеками для нейросетей, такими как TensorFlow, PyTorch, HuggingFace и другие [4] | Хорошо интегрируется с нейросетями, но требует больше усилий для настройки |

Таким образом, для реализации веб-интерфейса для нейросетевых решений, был выбран FastAPI, поскольку он поддерживает стандартную асинхронность и отлично интегрируется с нужными библиотеками, в отличие от Django, который является более универсальным фреймворком для разработки веб-приложений.

После выбора подходящего веб-фреймворка следующим этапом стало определение моделей, которые будут использоваться для решения ключевых задач — автоматической транскрибации видео и генерации тестовых заданий по тексту. Связанный с моделями узел майнд-карты разделяется на два подузла, каждый из которых посвящён отдельной задаче. В рамках каждой задачи рассматривались как локальные модели, так и API-решения, при этом предпочтение было отдано локальным моделям из-за их независимости от внешних сервисов, возможности работы в изолированной среде и потенциала для повторного использования в экосистеме компании. Более детально узел с выбором моделей отражён на рисунке 1.3.

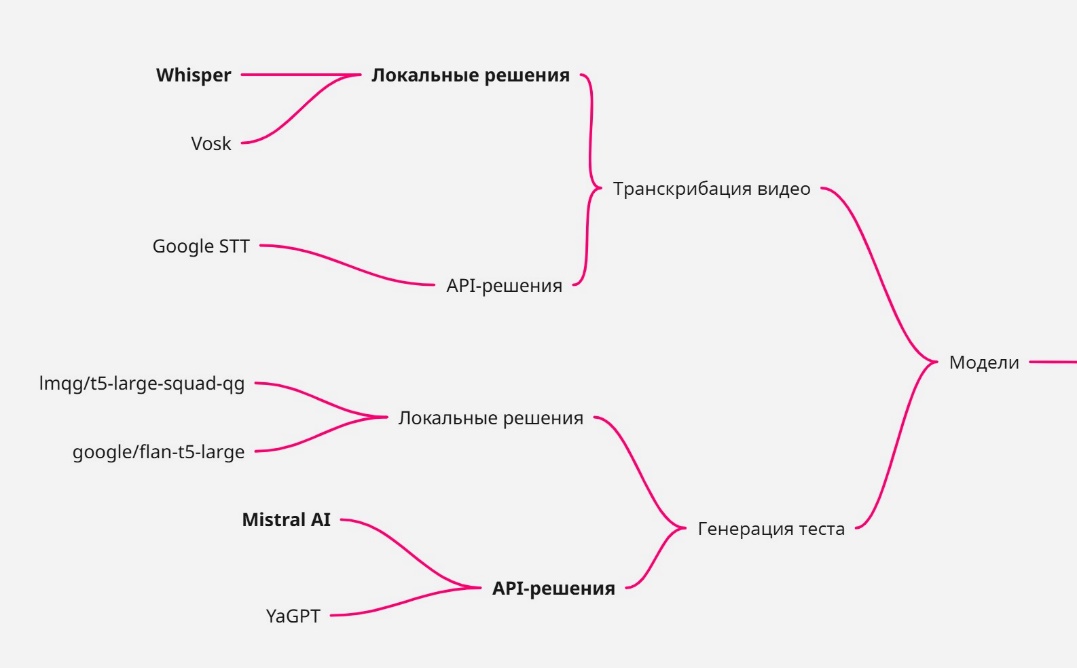


Рисунок 1.3 – Узел моделей на майнд-карте инструментария

Для решения задачи транскрибации видео были протестированы как локальные модели — Whisper и Vosk, так и облачное решение Google Speech-to-Text. На практике наилучшие результаты показала модель Whisper от OpenAI. Она обеспечила высокую точность распознавания речи, особенно в случае смешанного русско-английского аудио [5] и при работе с длительными записями. Помимо качества распознавания, Whisper отличается простой установкой, хорошей интеграцией с Python и минимальными требованиями к настройке, что делает её особенно удобной для встраивания в проект. В сравнении с ней Vosk показал значительно худшую производительность, оказался менее гибким и потребовал больше усилий при интеграции. Использование Google Speech-to-Text не рассматривалось как приоритетное решение, поскольку Whisper уже покрывал все требования без необходимости обращения к внешним сервисам.

Вторая ключевая задача — генерация тестов по транскрибированному тексту — также предполагала выбор между локальными моделями и облачными API. Первоначально были протестированы локальные решения, включая lmqg/t5-large-squad-qg и google/flan-t5-large. Однако результаты их работы оказались нестабильными: модели часто генерировали нерелевантные вопросы, допускали ошибки в структуре тестов, предоставляли повторяющиеся или неадекватные варианты ответов. Из-за низкой предсказуемости и необходимости дополнительной доработки этих моделей было принято решение в пользу использования внешних API-сервисов.

Для прототипирования хорошо себя зарекомендовал Mistral AI, предлагающий бесплатный доступ к своему API — это позволило быстро интегрировать генерацию тестов без дополнительных затрат. Однако для использования в продакшене потребовалась альтернатива, так как доступ к платным функциям сервиса в текущих условиях оказался невозможен. В результате было реализовано универсальное решение, позволяющее гибко подключать различные модели по API, с минимальными усилиями на конфигурацию. Рассматривался также YaGPT, однако отсутствие бесплатного тарифа сделало его непригодным для текущих задач.

Таким образом, оптимальной стратегией стало сочетание локальной модели Whisper для транскрибации и API-сервисов для генерации тестов, обеспечивающее баланс между качеством, удобством и возможностью масштабирования.

Перейдем к выбору технологий для клиентской части проекта. Этот узел майнд-карты делится на два подузла-фреймворка. Важным фактором при выборе было не только удобство и скорость написания кода, но и возможность создания отзывчивого и интерактивного интерфейса. В качестве вариантов были рассмотрены различные фреймворки, включая как более легковесные решения, так и полноценные инструменты для создания сложных приложений. Важно было учесть знакомство с технологией, чтобы избежать лишних затрат на обучение и ускорить процесс разработки. Подробно этот узел представлен на рисунке 1.4.

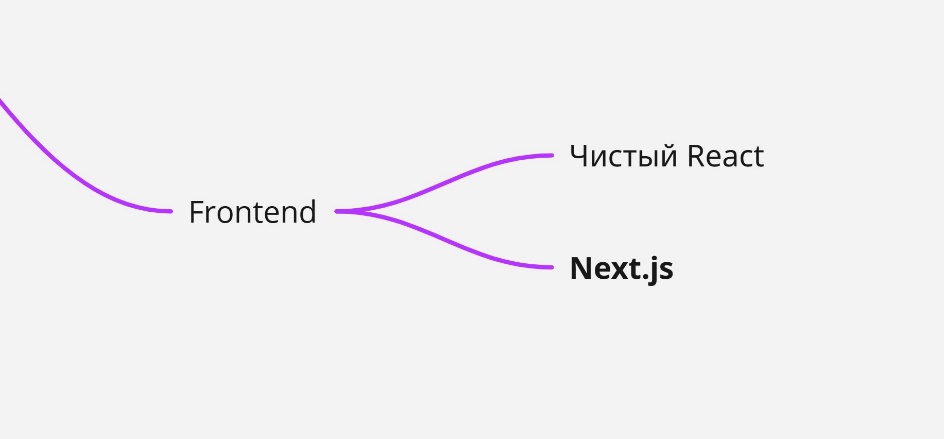


Рисунок 1.4 – Frontend-узел майнд-карты инструментария

В случае выбора технологий для фронтенда было принято решение отказаться от использования таких легковесных решений, как Bootstrap или чистого JavaScript с кастомными стилями, поскольку они не обеспечивают необходимую гибкость и функциональность для разработки полноценного веб-приложения со сложной бизнес-логикой. В качестве альтернативы было решено сравнить два решения: чистый React и Next.js, поскольку они предлагают разные подходы к разработке.

В отличие от чистого React, который представляет собой лишь библиотеку для создания пользовательских интерфейсов и требует сторонних инструментов для реализации роутинга и серверного рендеринга, Next.js предоставляет целый набор встроенных решений, что значительно упрощает разработку. Сравнение этих альтернатив представлено в таблице 1.4.

Таблица 1.4 – Сравнительная характеристика Next.js и чистого React

|  |  |  |
| --- | --- | --- |
| Критерий | Next.js | Чистый React |
| Основное назначение | Фреймворк для универсальных (full-stack) приложений с поддержкой серверного рендеринга (SSR) и статической генерации (SSG) [6] | Библиотека для построения пользовательских интерфейсов, требует дополнительных инструментов для роутинга и серверного рендеринга |
| Роутинг | Встроенная система маршрутизации, которая поддерживает динамические маршруты, статическую генерацию и SSR | Требует сторонних библиотек для роутинга (например, React Router) и конфигурации для SSR |
| Поддержка SSR (Server-Side Rendering) | Полная поддержка серверного рендеринга из коробки, что позволяет улучшить SEO и производительность при первоначальной загрузке страницы | Отсутствует поддержка SSR, требует использования сторонних решений, таких как Next.js или Gatsby, для реализации этой функции [6] |
| Генерация статического контента (SSG) | Встроенная поддержка статической генерации страниц на этапе сборки, что позволяет улучшить производительность и снизить нагрузку на сервер | Не поддерживает статическую генерацию страниц из коробки, требует дополнительных настроек или использования других фреймворков |
| Рендеринг на стороне клиента (CSR) | Поддерживает как рендеринг на стороне сервера, так и на клиенте (гибкость). Использует React для рендеринга на клиенте | Чисто клиентская библиотека, рендерит страницы только на стороне клиента |
| Поддержка API | Встроенная поддержка создания API-эндпоинтов, что позволяет легко интегрировать серверную логику с фронтендом в одном проекте | Не предоставляет готовых решений для API; для создания серверной части нужно использовать сторонние инструменты и фреймворки |
| Оптимизация производительности | Оптимизация с помощью автоматической разделённой загрузки (code splitting), сервера с рендерингом на стороне сервера и статической генерации | Требует самостоятельной настройки оптимизации, такие как code splitting и оптимизация изображений, с помощью сторонних решений |

Таким образом, после тщательного анализа, было принято решение выбрать Next.js. Это решение обусловлено преимуществами, которые он предоставляет, такими как встроенная поддержка роутинга, SSR, SSG, а также готовые решения для создания API и оптимизации производительности. Next.js оказался наиболее подходящим фреймворком для данного проекта, обеспечивая высокую производительность, удобство в разработке и гибкость в настройке различных аспектов рендеринга.

После выбора технологий для фронтенда и бэкенда следующим важным этапом стало принятие решения по выбору СУБД. Этот узел майнд-карты посвящён выбору системы управления базами данных, которая должна эффективно справляться с хранением как структурированных, так и неструктурированных данных — таких как текстовые материалы, документы, видео, изображения и другие типы информации. Основным критерием при выборе была способность системы устойчиво работать с большими объемами данных и поддерживать гибкую структуру данных, что является ключевым для дальнейшей масштабируемости и гибкости системы. Подробное описание данного узла представлено на рисунке 1.5.

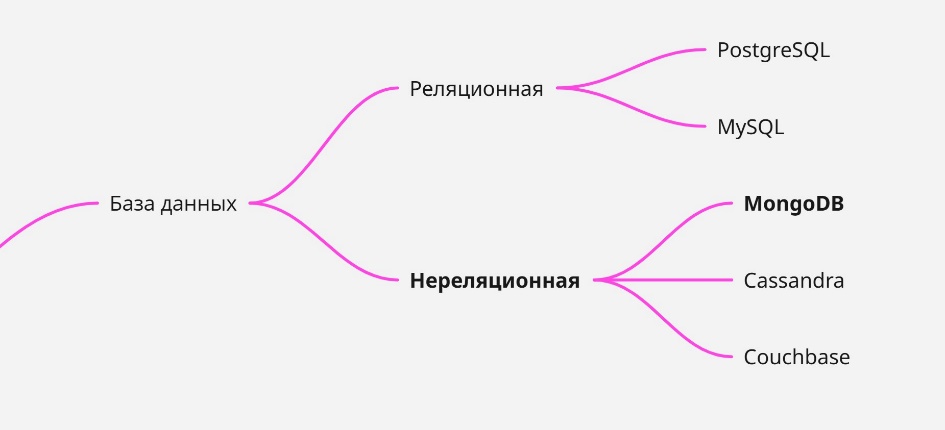


Рисунок 1.5 – Database-узел майнд-карты инструментария

Для выбора оптимального решения рассмотрим конкретные реляционные и нереляционные СУБД, сводное представление которых приведено в таблице 1.5.

Таблица 1.5 – Сравнительная таблица избранных реляционных и нереляционных СУБД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Параметр | PostgreSQL / MySQL | MongoDB | Cassandra | Couchbase |
| Тип хранения | Реляционный | Документно-ориентированный | Колонно-ориентированный [7] | Документно-ориентированный |
| Гибкость структуры данных | Низкая | Высокая | Средняя | Высокая |
| Масштабируемость | Вертикальная | Горизонтальная [7] | Отличная горизонтальная | Горизонтальная |
| Работа с большими объемами данных | Средняя | Хорошая | Отличная | Хорошая |
| Хранение мультимедийных материалов | Не предназначена напрямую | Подходит | Ограничено | Подходит |
| Поддержка транзакций | Полная поддержка | Частичная | Нет | Частичная |
| Простота интеграции | Высокая | Высокая | Средняя | Средняя |

С учётом специфики проекта — работы с материалами разных форматов, отсутствия требований к сложным транзакциям и акцентом на масштабируемость — было решено использовать документно-ориентированную нереляционную базу данных. Реляционные решения не подошли по нескольким причинам: они предполагают жёстко определённую схему и ориентированы на строго структурированные данные, в то время как обучающие материалы и нейросетевые результаты часто носят динамический, слабоформализованный характер [7].

В результате анализа MongoDB была выбрана в качестве основной СУБД. Её способность обрабатывать большие объёмы данных, удобство хранения документов и встроенные механизмы масштабирования позволяют эффективно реализовать задачи данного проекта без перегрузки инфраструктуры.

На основании проведённого анализа и серии практических испытаний был выбран стек, обеспечивающий оптимальный баланс между гибкостью, производительностью и удобством интеграции. В качестве пользовательской серверной части используется Kotlin с фреймворком Spring Boot, обеспечивающий строго типизированную, масштабируемую и легко расширяемую архитектуру. Для взаимодействия с нейросетевыми компонентами был выделен отдельный сервис, реализованный с использованием FastAPI, благодаря его поддержке асинхронного программирования и отличной интеграции с ML-библиотеками. Клиентская часть построена на Next.js — современном фреймворке на базе React, обеспечивающем высокую производительность, поддержку серверного рендеринга и удобную систему маршрутизации. В качестве базы данных выбрана MongoDB, так как она эффективно справляется с хранением разнородных данных, включая тексты, видео и документы, и не требует жёстко заданной схемы. Для задач распознавания речи используется модель Whisper, показавшая наилучшие результаты среди локальных решений. Генерация тестов реализована через API Mistral AI, благодаря его стабильной работе, простоте интеграции и доступности в тестовом режиме.

Итак, в первой главе был проведён анализ организации-заказчика и выбор программных инструментов для разработки. Материалы главы собраны по результатам преддипломной практики.

Сначала был выполнен детальный анализ деятельности базы преддипломной практики – ИП «Миргородский Д.Ю.» и организации-заказчика ОАО «ГлобалСофт», что позволило выявить ключевые проблемы и потребности заказчика, связанные с управлением обучающими материалами для широкой линейки продуктов. Обнаруженные ограничения существующих систем и неудовлетворённость готовыми решениями послужили основой для принятия решения о создании специализированного программного продукта.

Затем был проведён сравнительный анализ существующих программных решений, что обеспечило выбор оптимального инструментария для реализации проекта. В качестве технической базы были выбраны современные и эффективные технологии: Spring Boot на Kotlin для серверной части, FastAPI на Python, Next.js для фронтенда, а также нейросетевые инструменты для транскрибации и генерации тестов, включая Whisper и Mistral AI.

Таким образом, первая глава формирует прочную теоретическую и техническую основу, отвечающую требованиям заказчика и закладывающую фундамент для дальнейшей разработки системы.

# 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

# 2.1 Проектирование архитектуры системы

Для проектирования архитектуры системы были выбраны UML-диаграммы, предназначенные для визуализации ключевых элементов и их взаимодействия. Для этого начнем с определения, какие диаграммы нам потребуются. В общем случае UML-диаграммы делятся на два основных типа: структурные и поведенческие [8], каждый из которых имеет свои подкатегории. Эти диаграммы отражают статическую структуру системы и динамическое поведение соответственно. Выделим их в майнд-карте, представленной на рисунке 2.1, а более детально разберем выбор и особенности каждого из них дальше.



Рисунок 2.1 – Классификация UML-диаграмм

Для построения диаграмм был использован инструмент PlantUML, который позволяет преобразовывать текст в диаграммы. Этот инструмент удобен тем, что работает как в онлайн-режиме, так и в виде локальной утилиты, а также имеет плагины для популярных платформ, таких как Confluence. PlantUML — это отличный выбор для проектной документации, хотя стоит отметить, что не всегда он строго следует стандарту UML 2.0.

С использованием данного инструмента в рамках проектирования архитектуры системы был реализован ряд поведенческих диаграмм, целью которых является формализация взаимодействия пользователей с системой и отображение логики её функционирования. Поведенческие диаграммы позволяют описать, как система реагирует на внешние воздействия и как изменяется её поведение в процессе работы [8]. На рисунке 2.2 приведён фрагмент майнд-карты, в котором отражены типы выбранных поведенческих диаграмм: случаи использования, активности, состояния и взаимодействия.

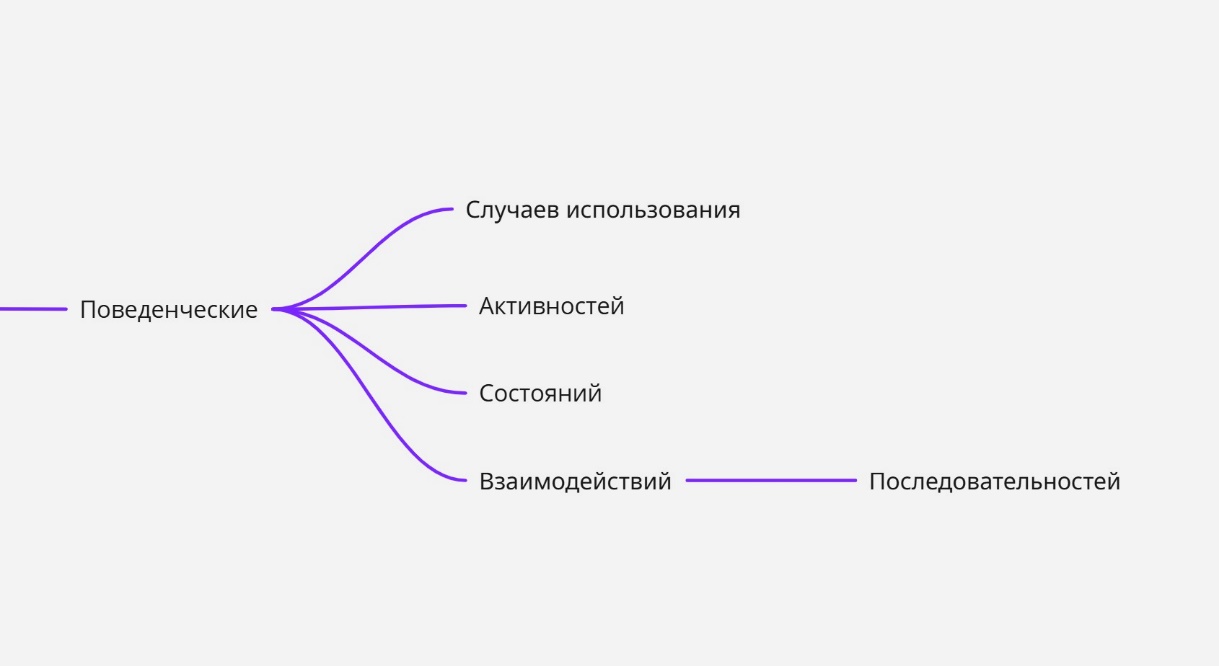


Рисунок 2.2 – Поведенческий узел майнд-карты UML-диаграмм

Рассмотрение поведенческих аспектов начнем с диаграмм случаев использования. Они позволяют описать функциональность системы с точки зрения взаимодействия с внешними акторами. В рамках системы были выделены два ключевых скоупа: «Курсы» и «Пользователи». Каждый из них охватывает свою группу функциональности и включает в себя набор соответствующих акторов и действий.

Первый скоуп — «Пользователи системы» — включает в себя действия, связанные с базовым взаимодействием пользователя с системой: вход в учётную запись, редактирование профиля и административное управление. На рисунке 2.3 представлена диаграмма вариантов использования, в которой все пользовательские роли сводятся к абстрактному базовому актору. Эта роль не реализуется напрямую в системе, но служит модельной основой для группировки общего функционала, доступного всем типам пользователей.



Рисунок 2.3 – Диаграмма вариантов использования для скоупа «Пользователи»

Конкретными реализациями базового актора являются пользователь, редактор и администратор. Пользователь получает доступ к базовым операциям, таким как вход и настройка профиля. Администратор, являясь расширением базового актора в плане своих прав в пользовательском скоупе, обладает системными правами: управление ролями и правами, конфигурациями, интеграциями и акторами. Редактор, показанный в этой диаграмме, находится на пересечении двух скоупов, и представлен здесь исключительно как субъект с правами базового пользователя — создание и редактирование курсов в этом контексте не рассматривается.

Второй скоуп — «Курсы» — охватывает действия, связанные с управлением содержанием и структурой курсов, а также с администрированием ролей внутри них. Диаграмма на рисунке 2.4 демонстрирует случаи использования, характерные для взаимодействия с курсами, и показывает, какие роли ответственны за выполнение соответствующих действий.

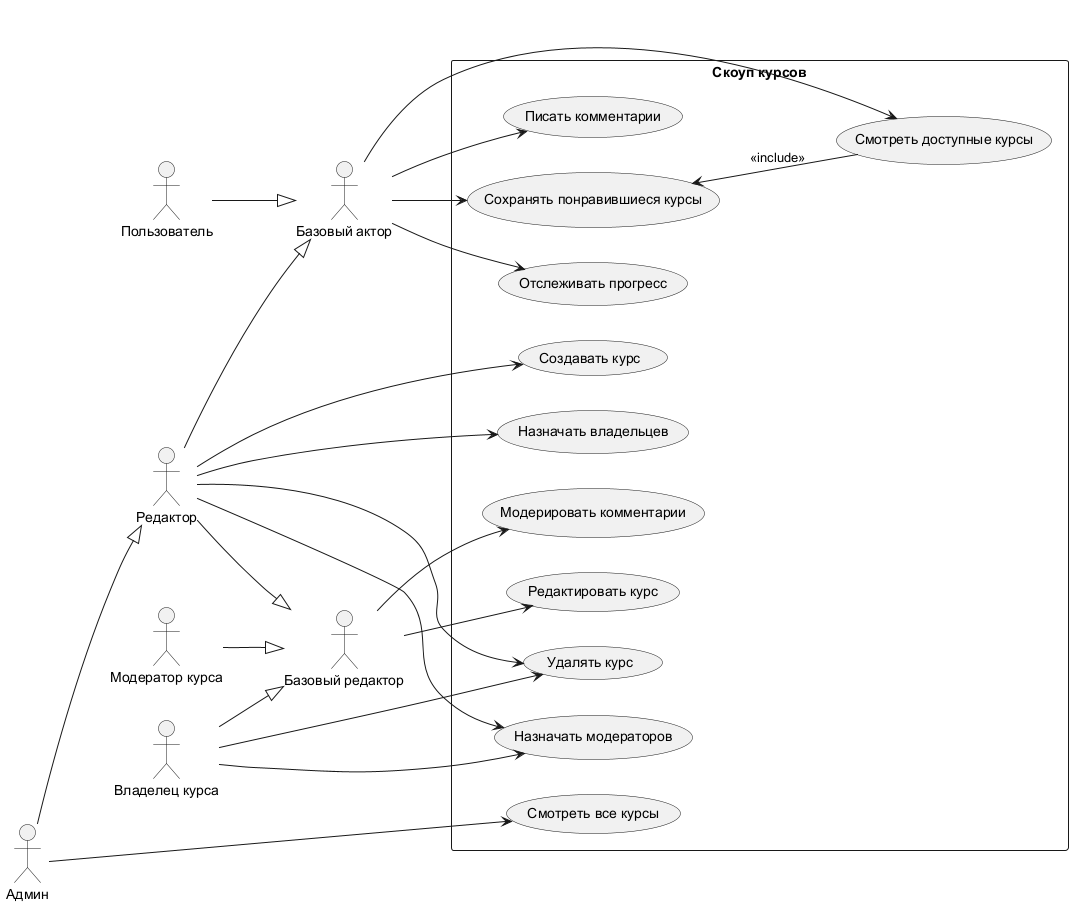


Рисунок 2.4 – Диграмма вариантов использования скоупа «Курсы»

В модели предусмотрена абстрактная роль базового редактора, которая агрегирует общие полномочия участников, взаимодействующих с курсами: редактирование курса и модерация комментариев. Хотя такая роль не реализуется в системе, она служит основой для дальнейшего наследования. Конкретные роли — редактор, владелец курса и модератор курса — реализуют постепенное снижение уровня доступа.

Редактор — наиболее полномочная роль: он может создавать и удалять курсы, а также назначать владельцев и модераторов. Ее права имеет и администратор системы, что показано наследованием. Владелец курса ограничен в правах: он не может создавать курсы, но может управлять назначением модераторов и удалением курса. Модератор обладает минимальным объёмом полномочий и отвечает исключительно за работу с комментариями и урезанное редактирование курса.

Все роли, наследуемые от базового актора в рамках скоупа «Курсы» — редактор, владелец курса и модератор — помимо своих основных обязанностей, также обладают функциональностью, доступной всем пользователям системы: они могут просматривать доступные курсы, сохранять понравившиеся, оставлять комментарии и отслеживать прогресс прохождения. В отличие от них, администратор обладает расширенными правами и может просматривать все курсы системы, включая недоступные для обычных пользователей.

На рисунке 2.5 представлена расширенная диаграмма вариантов использования, раскрывающая функциональность редактирования курсов, секций и лекций для ролей владельца и модератора курса. Эта диаграмма позволяет более детально зафиксировать, какие именно действия доступны каждой из этих ролей в рамках скоупа «Курсы».

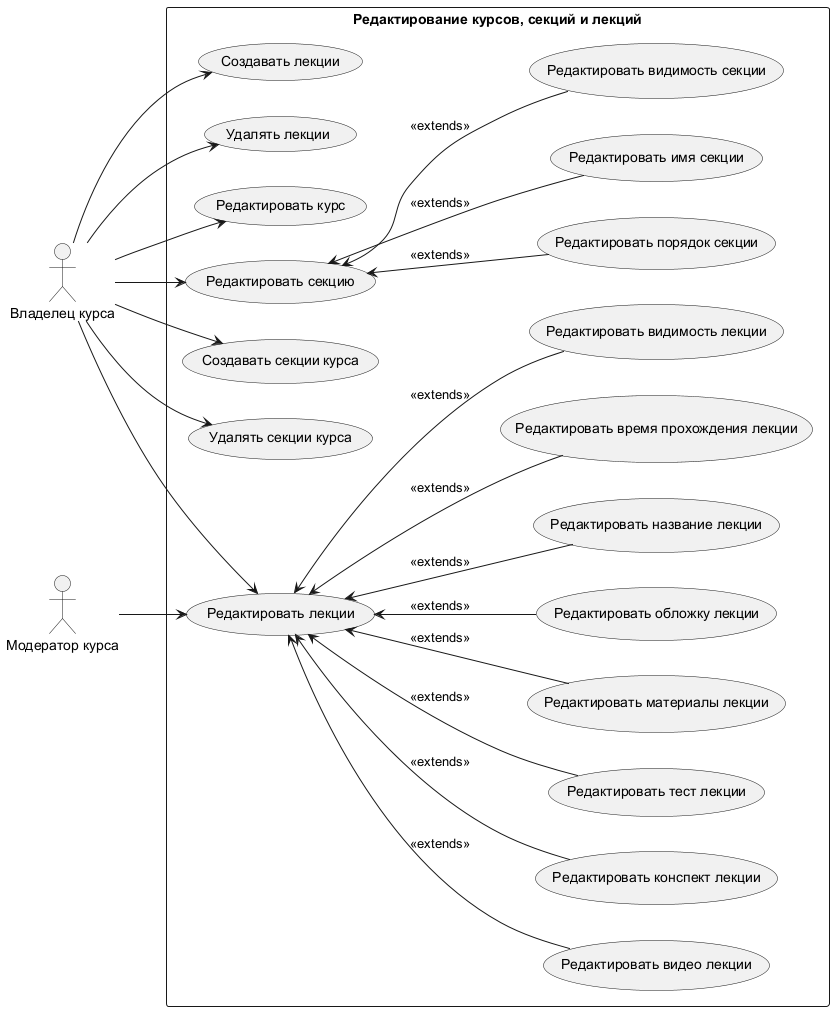


Рисунок 2.5 – Диаграмма вариантов использования редактирования курса, секций и лекций для владельца и модератора

Владелец курса обладает полным набором прав, связанных с модификацией структуры курса: он может редактировать сам курс, создавать и удалять секции, а также работать с лекциями. Возможности редактирования секций включают в себя изменение их названия, видимости и порядка — все эти действия представлены как расширения основного случая использования «Редактировать секцию». Аналогичным образом, действия по изменению лекций представлены как расширения для случая «Редактировать лекции». Владелец может редактировать обложку, название, длительность, видимость, видео, конспект, тест и материалы каждой лекции.

Модератор курса, в отличие от владельца, имеет ограниченные полномочия и может только редактировать лекции. Однако доступные ему действия по редактированию идентичны тем, что доступны владельцу — модератор может изменять все составляющие лекционного блока, но не может вмешиваться в структуру курса, например, создавать или удалять секции.

Диаграмма на рисунке 2.6 описывает процесс просмотра доступных курсов для различных ролей в системе. Все роли, унаследованные от базового редактора, могут просматривать курсы, к которым они были присоединены посредством создания курса или добавления роли другим пользователем. Это включает в себя как редакторов, владельцев курсов, так и модераторов.

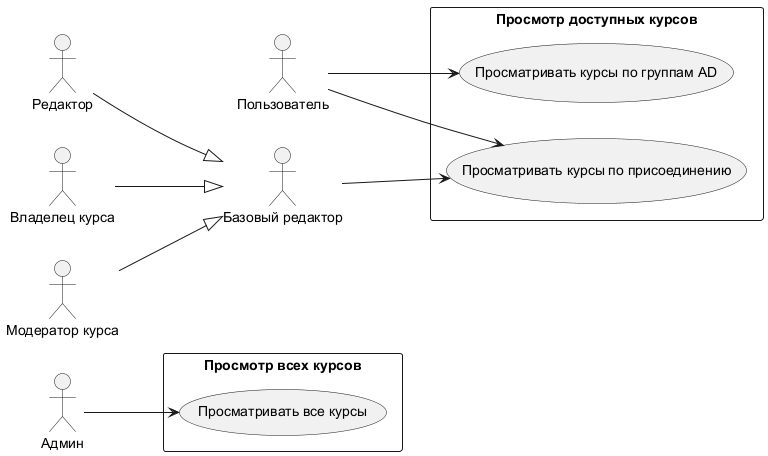


Рисунок 2.6 – Диаграмма вариантов использования для просмотра доступных курсов

Обычные пользователи имеют возможность просматривать курсы в рамках маппинга групп Active Directory на курсы или по принципу присоединения к курсу. В свою очередь, администраторы имеют расширенные полномочия и могут просматривать все курсы без ограничений.

После описания вариантов использования перейдем к более детальному описанию поведения системы с помощью диаграмм активности. Эти диаграммы позволяют более точно моделировать последовательность действий в процессе взаимодействия пользователей с системой и описывать изменения состояний объектов или компонентов системы в ответ на действия пользователей [9]. Диаграммы активности показывают, как различные шаги процесса взаимосвязаны и какую логику выполняет система в реальном времени.

Рисунок 2.7 демонстрирует диаграмму активности, отражающую процесс поиска курсов пользователем. Сначала пользователь вводит запрос для поиска, после чего система обрабатывает запрос и выводит результаты. Затем пользователь может фильтровать курсы по различным категориям, сложности и дате. После применения фильтров система обновляет список курсов. Этот процесс описывает типичный сценарий поиска и фильтрации курсов в системе.

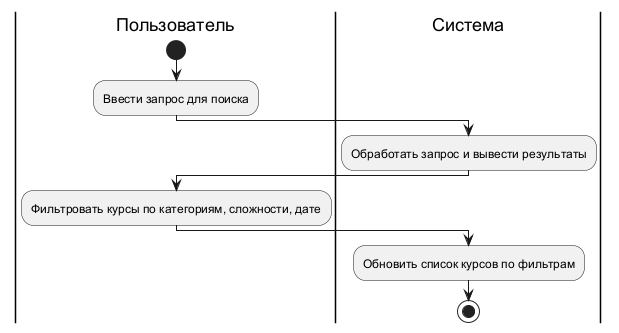


Рисунок 2.7 – Activity-диаграмма для поиска по курсам

Рисунок 2.8 представляет диаграмму активности для процесса интеграции сторонней системы. На начальном этапе базовый редактор отправляет запрос на выполнение интеграции. Система принимает запрос, подставляет секрет в скрипт и выполняет его. Сторонняя система, получив запрос с секретом, выполняет запрос и возвращает результат. Система, в свою очередь, обрабатывает полученные данные и преобразует их в читаемый вид. В конце базовый редактор получает результат выполнения интеграции.

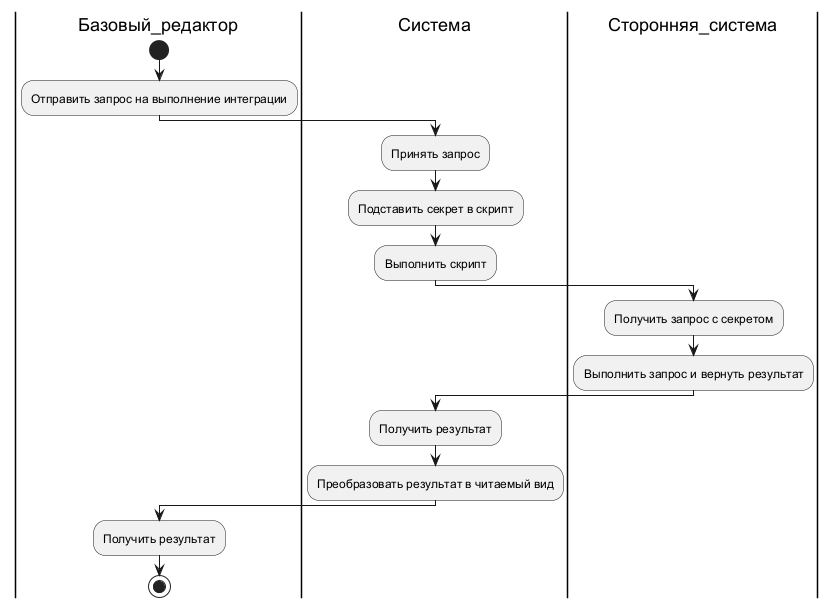


Рисунок 2.8 – Activity-диаграмма для выполнения интеграции

Рисунок 2.9 демонстрирует диаграмму активности, описывающую процесс версионирования курса. При создании курсу присваивается версия 0.1 – изначально курс не опубликован – после чего редактор может выполнять любые CRUD-операции над секциями, лекциями и материалами без ограничений. Если курс уже опубликован, при выполнении CRUD-операции редактором система автоматически повышает версию курса: мажорная версия повышается при изменениях в секциях, минорная — при изменениях в лекциях. После этого новая версия курса сохраняется.

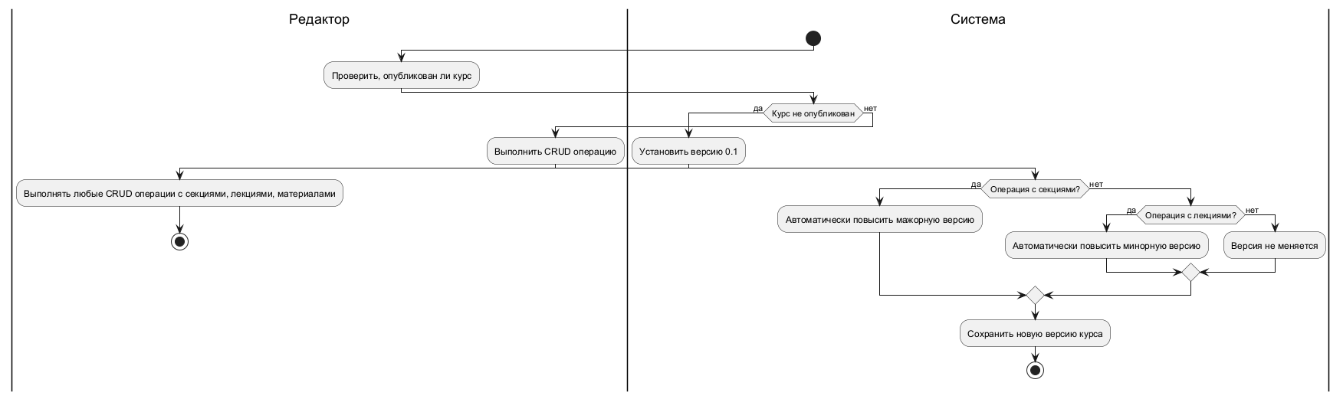


Рисунок 2.9 – Activity-диаграмма процесса версионирования

Рисунок 2.10 иллюстрирует процесс получения доступа к материалам курса. Пользователь запрашивает доступ, система проверяет, опубликован ли курс. Если курс не опубликован, доступ запрещен. Если курс опубликован, система проверяет наличие подписки пользователя на этот курс. В случае отсутствия подписки она создается для последней известной версии. При наличии подписки пользователь получает доступ только к версии курса, на которую он подписался. Кроме того, прогресс пользователя фиксируется исключительно в рамках версии курса, на которую он подписан.

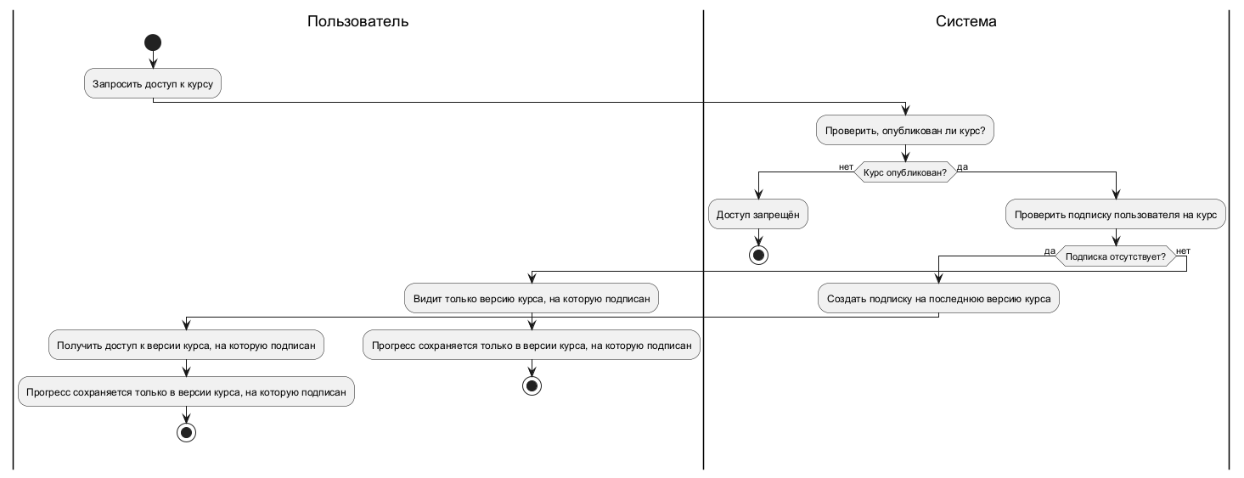


Рисунок 2.10 – Activity-диаграмма процесса доступа пользователя к курсу и отслеживания прогресса

Следующим элементом поведенческих диаграмм в проекте являются диаграммы состояний. Эти диаграммы используются для отображения различных состояний объектов или сущностей системы и переходов между ними, что позволяет визуализировать, как объект изменяет своё состояние в ответ на различные события или действия. Диаграммы состояний полезны для отображения жизненного цикла объекта, таких как курс, лекция или любой другой элемент системы, и помогают понять логику их функционирования.  
На рисунке 2.11 представлена диаграмма состояний для объекта «Курс» с точки зрения проходящего его пользователя. Изначально курс находится в состоянии «Не начат». После того как пользователь начинает обучение, он переходит в состояние «В процессе». Завершающим состоянием курса является «Завершён», когда пользователь полностью проходит курс.

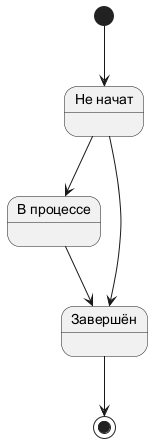


Рисунок 2.11 – Диаграмма состояний для курса

Рисунок 2.12 отображает более сложный процесс — создание лекции. Лекция проходит через несколько этапов, начиная от начала создания, загрузки видео, генерации конспекта и теста, и заканчивая добавлением деталей, таких как название и видимость. Каждый этап может приводить к различным вариантам завершения, включая создание черновика или окончательное создание лекции.

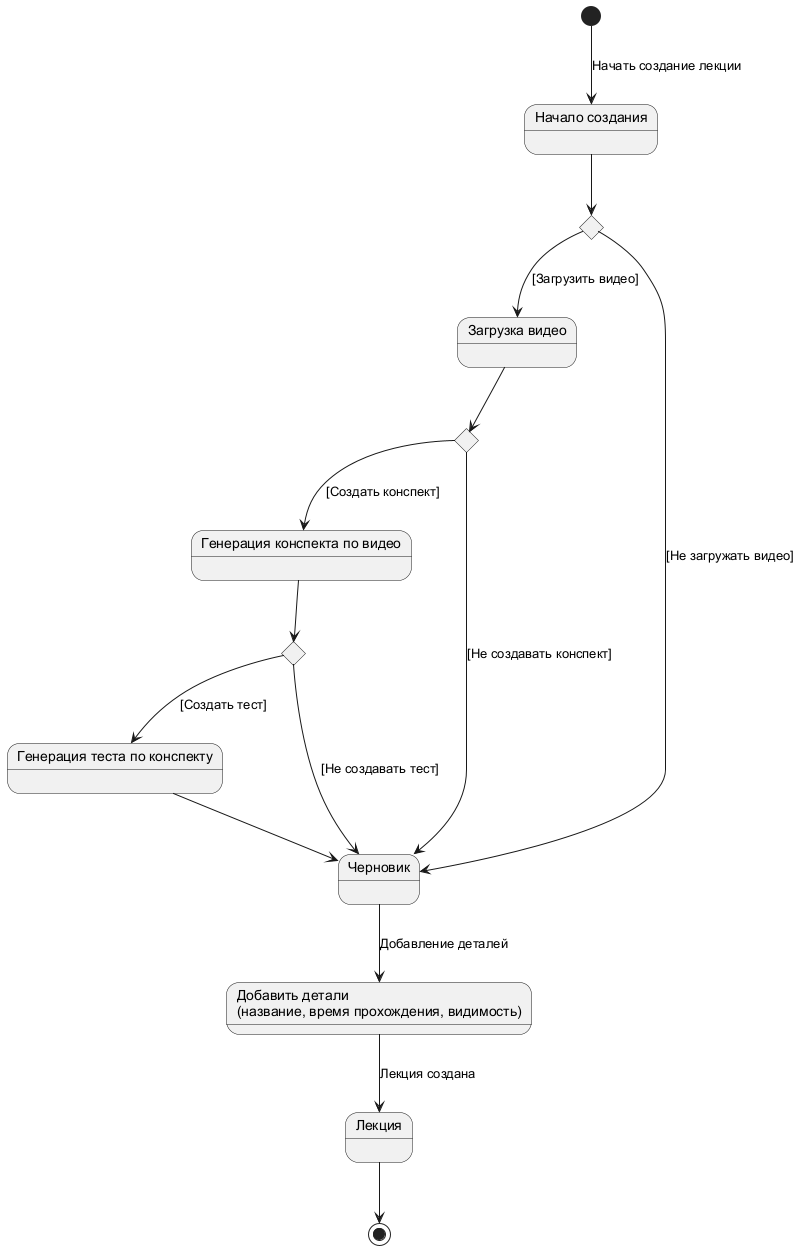


Рисунок 2.12 – Диаграмма состояний для создания лекции

Диаграмма, представленная на рисунке 2.13, иллюстрирует возможные состояния интеграции и переходы между ними в процессе жизненного цикла. Интеграция изначально создаётся в состоянии «Создана» и может быть либо сразу активирована, либо переведена во временно отключённое состояние, отправлена на редактирование или полностью удалена.

Активная интеграция может быть отключена, а отключённая — снова отредактирована или удалена. После редактирования интеграция может быть вновь активирована. Состояние «Удалена» является финальным, после которого дальнейшие переходы невозможны.

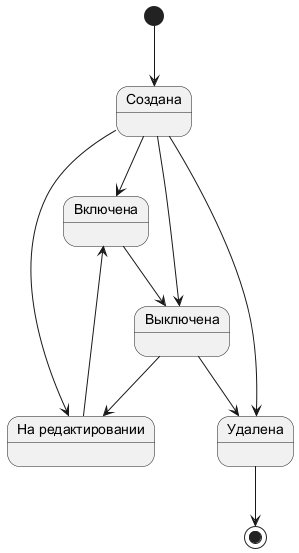


Рисунок 2.13 – Диаграмма состояний интеграции

Диаграмма, представленная на рисунке 2.14, описывает жизненный цикл объекта «Комментарий». После создания комментарий может быть отредактирован пользователем, скрыт базовым редактором – например, модератором курса – или удалён. Комментарий, находящийся в отредактированном состоянии, также может быть скрыт или удалён.

Состояния «Скрыт базовым редактором» и «Удалён» являются финальными и не предполагают дальнейших переходов: скрытый комментарий исключается из отображения, а удалённый полностью исключается из системы.

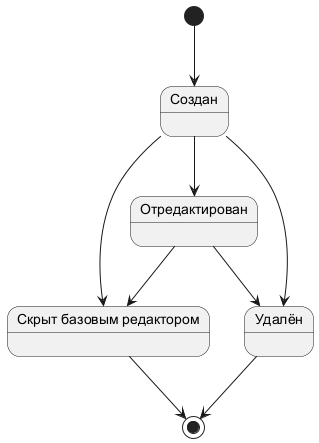


Рисунок 2.14 – Диаграмма состояний комментария к курсу

После завершения рассмотрения диаграмм состояний, мы переходим к последнему типу поведенческих диаграмм, выбранному для описания поведения компонентов системы, — диаграммам последовательностей. Эти диаграммы являются подтипом диаграмм взаимодействий и иллюстрируют обмен сообщениями между участниками в рамках одного конкретного сценария.

Но перед тем, как перейти к описанию диаграмм последовательностей, необходимо рассмотреть ещё один важный элемент архитектурного проектирования — диаграмму компонентов, которая относится к категории структурных диаграмм UML. В отличие от поведенческих диаграмм, описывающих, как работает система, структурные диаграммы фиксируют её архитектурное устройство, то есть из чего она состоит [8].

На рисунке 2.15 представлен узел майнд-карты UML-диаграмм, отражающая отобранные типы структурных диаграмм. В рамках проекта выбраны два типа: диаграммы компонентов и диаграммы классов. В данном разделе будет рассмотрена диаграмма компонентов, тогда как диаграммы классов будут описаны далее при переходе к логике данных и их связям.



Рисунок 2.15 – Узел структурных диаграмм на майнд-карте UML диграмм

На рисунке 2.16 представлена диаграмма компонентов, описывающая архитектуру системы на высоком уровне. Она отражает ключевые программные модули, их размещение в логических пакетах и зависимости между ними. Данная диаграмма является основой для понимания последующих диаграмм последовательностей, так как позволяет точно определить, какие компоненты участвуют во взаимодействии в каждом из сценариев.

Компонент Frontend — это клиентская часть системы, через которую пользователь взаимодействует с сервисами: инициирует создание лекций, авторизуется, запускает генерацию конспектов и тестов. Все операции выполняются через три ключевых внутренних сервиса. UserSphere реализует бизнес-логику, взаимодействует с базой данных MongoDB и обеспечивает аутентификацию через LDAP. SynopsisForge занимается обработкой видео и генерацией конспектов, а IntegrationsRunner отвечает за интеграции с внешними API, включая работу с моделью Mistral AI. Все три сервиса используют MongoDB как основное хранилище данных.

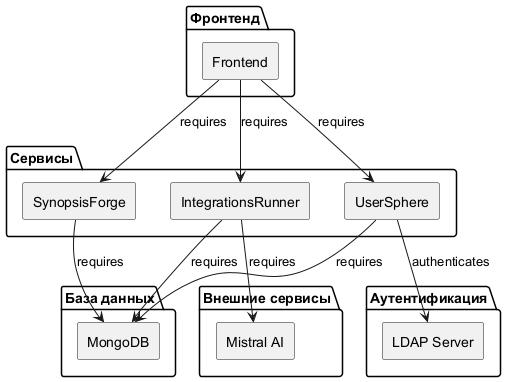


Рисунок 2.16 – Диаграмма компонентов

После рассмотрения компонентов системы сформировано общее представление о взаимодействующих модулях и их функциях, поэтому логично перейти к диаграммам последовательностей, которые иллюстрируют конкретные сценарии обмена сообщениями между этими компонентами.

На рисунке 2.17 изображён пошаговый процесс создания черновика лекции базовым редактором. Он инициирует создание лекции, загружает видео. Если видео успешно загружено, редактор может запросить генерацию вспомогательных материалов — конспекта и/или теста, которые создаются с использованием внешних компонентов – SynopsisForge и Mistral AI через IntegrationsRunner. После завершения всех необходимых этапов черновик сохраняется.

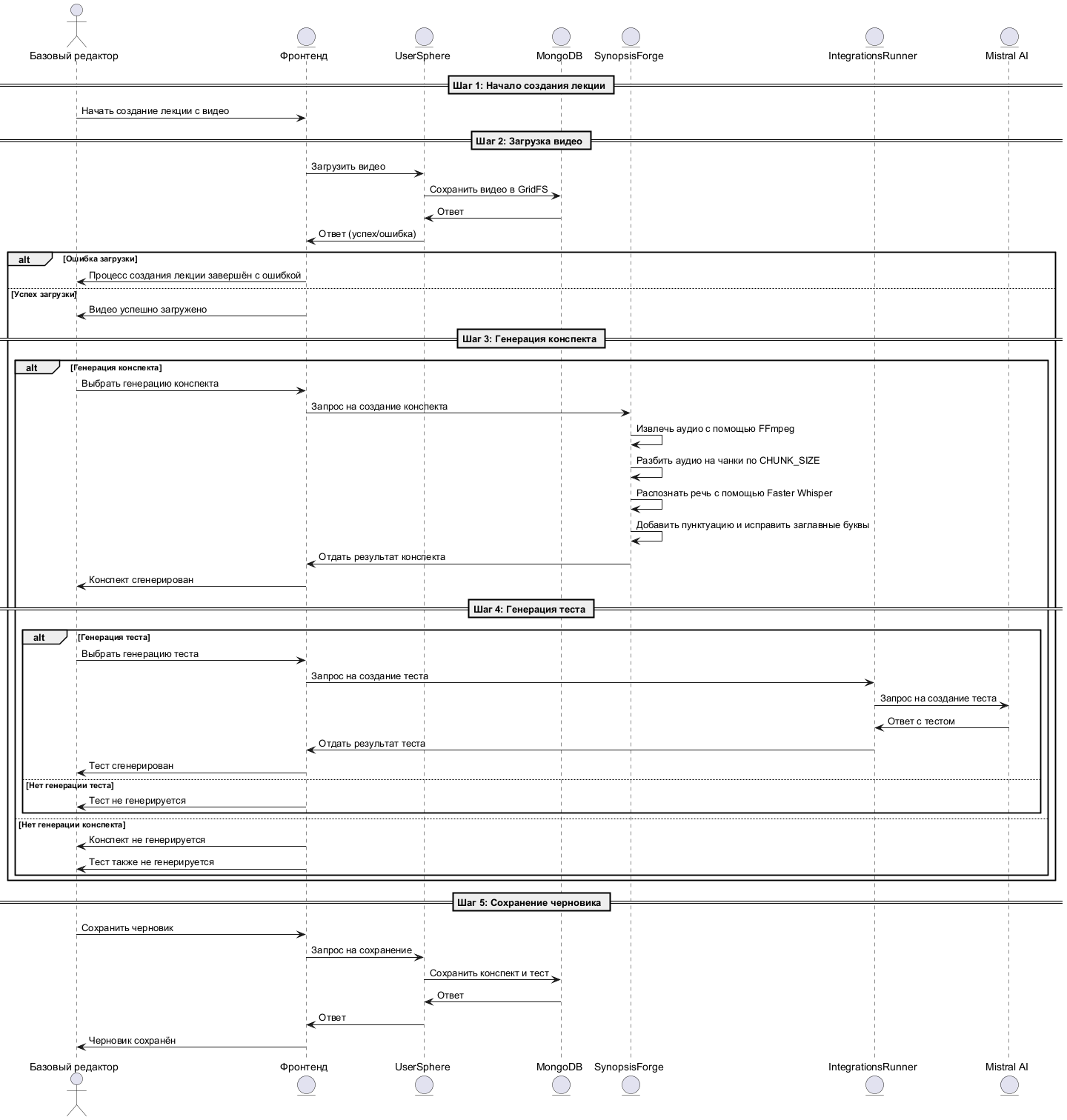


Рисунок 2.17 – Диаграмма последовательности для создания черновика лекции

На рисунке 2.18 показан процесс авторизации базового пользователя в системе. Пользователь вводит логин и пароль, которые передаются с фронтенда в UserSphere, далее запрос уходит в LDAP-сервер. В случае успешной авторизации проверяется активность учётной записи, после чего, при необходимости, происходит синхронизация данных с MongoDB.

Система также предусматривает fallback-механику: если LDAP временно недоступен, при включённой опции loginFallback возможна попытка локальной авторизации с использованием сохранённого хэша пароля в MongoDB.

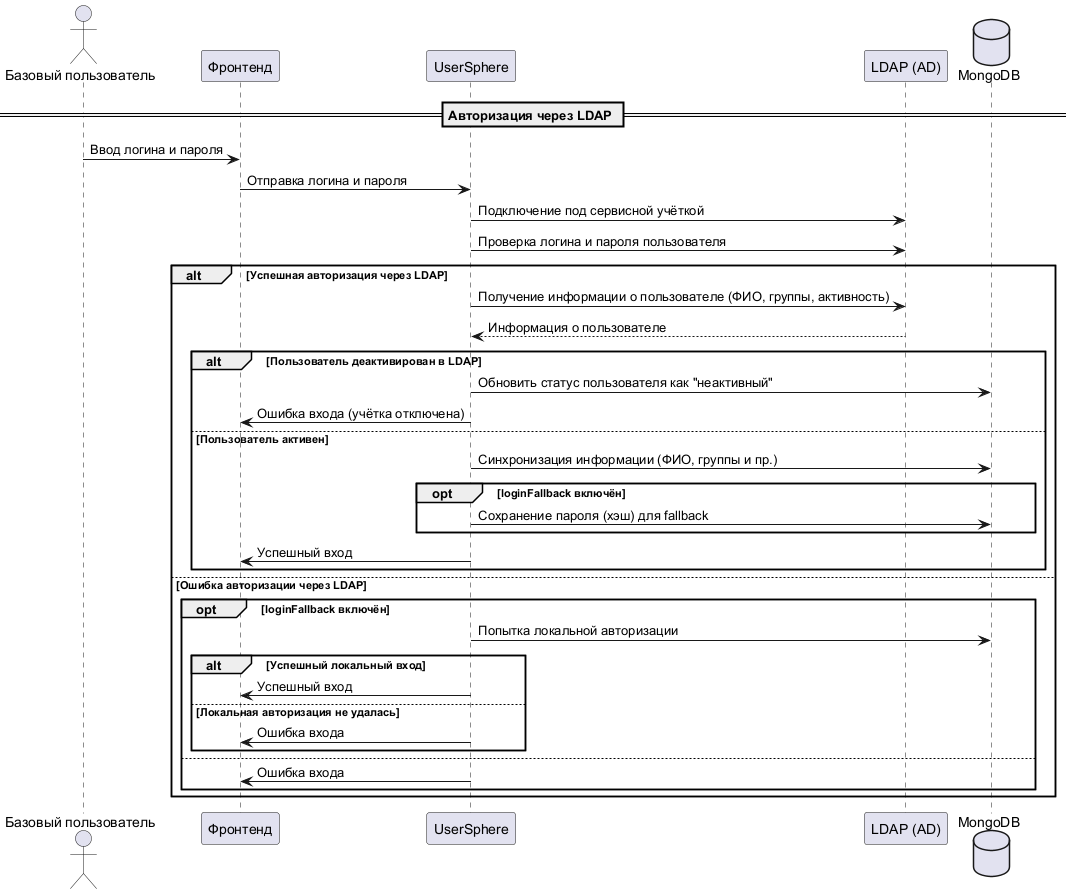


Рисунок 2.18 – Диаграмма последовательности для входа в систему

После анализа последовательностей взаимодействия становится возможным перейти к более структурированному представлению системы в виде диаграмм классов. Диаграммы классов позволяют отразить статическую архитектуру приложения, описав основные сущности, их свойства и связи между ними.

Общую диаграмму классов системы, охватывающую все ключевые сущности и связи, было решено разделить на две части для улучшения читаемости и удобства восприятия. Первая диаграмма — общесистемная — охватывает весь скоуп приложения, включая пользователей, роли, разрешения, настройки, загрузки и интеграции. Она даёт общее представление о ключевых сущностях, важных для функционирования системы и управления доступом. Вторая диаграмма сосредоточена исключительно на скоупе курсов — описывает структуру курсов, версий, секций, уроков, подписок и прогресса пользователя. Это позволяет детально рассмотреть предметную область функционала, связанного с курсами, и связи между его элементами.

Диаграмма классов, представленная на рисунке 2.19, отражает общесистемный скоуп и включает ключевые сущности, отвечающие за управление пользователями, их ролями, правами доступа, настройками, загрузками и интеграциями с внешними сервисами. Класс User содержит основные атрибуты пользователя, такие как уникальный идентификатор, дата создания, имя, логин, список ролей и состояние активности. Пользователи связаны с такими вспомогательными структурами, как UserEmail с адресом электронной почты и статусом верификации, UserSetting, определяющий настройки интерфейса, включая тему и язык, а также UserServices, которые предоставляют дополнительные функции, например, управление сессиями через JWT, LDAP-авторизацию и хранение паролей.

Роли пользователей описаны интерфейсом Role с идентификатором, именем, описанием и списком областей применения, таких как управление пользователями или курсами. Модель прав доступа Permission связывает роли с разрешениями, задавая, какие роли имеют доступ к определённым операциям.

Группа классов приватных настроек PrivateSetting включает типы настроек, их значения, условия видимости и дополнительные метаданные, что обеспечивает гибкую и многоуровневую конфигурацию системы. Сущность Upload хранит метаданные о файлах, загруженных пользователями, включая тип, размер, путь и время загрузки.

Особое место занимает пакет интеграций, в котором класс ApiIntegration описывает внешние API-сервисы, подключаемые к системе. Он содержит идентификатор, имя интеграции, секретный ключ, исполняемый скрипт, сведения о пользователе, дату создания и обновления, а также флаг активности. Это позволяет централизованно управлять внешними сервисами и расширять функционал системы.

Ассоциации между классами отражают связи, необходимые для реализации комплексного управления пользователями и их доступом, а также для конфигурирования системы через настройки и интеграции. Данная диаграмма даёт структурированное представление о базовых элементах и взаимодействиях в рамках общей архитектуры проекта.

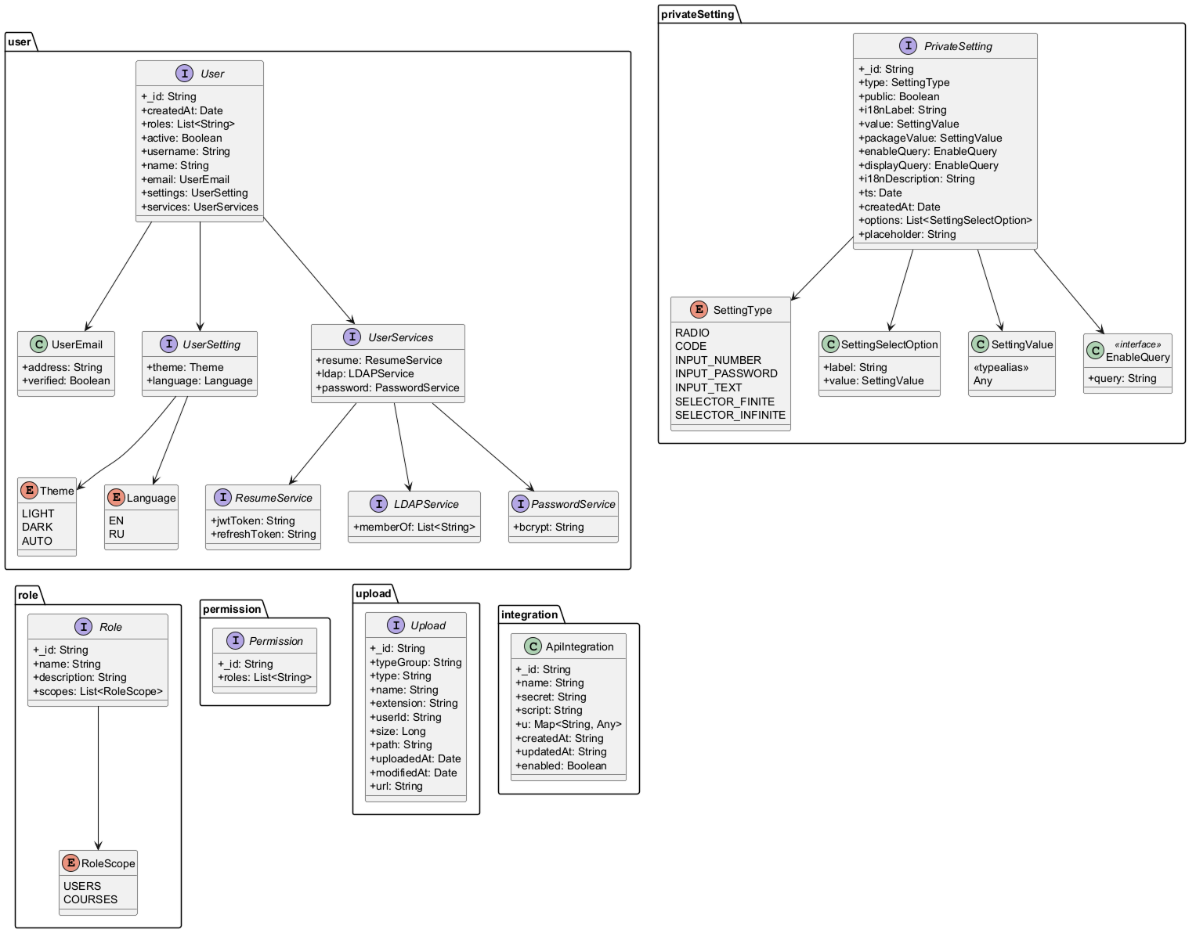


Рисунок .19 – Диаграмма классов общесистемного скоупа

Диаграмма классов, представленная на рисунке 2.20, описывает структуру и взаимосвязи компонентов, связанных с курсами. Основным элементом является класс Course с такими атрибутами, как идентификатор, название, описание, изображение, автор курса, даты создания и обновления, теги, статус публикации и список версий курса.

Каждая версия курса представлена классом CourseVersion, который содержит имя и список секций. Разделы курса представлены классом CourseVersionSection и включают идентификатор и список уроков. Лекции обозначены классом CourseVersionLesson, который содержит идентификатор, признак черновика, идентификаторы видео и опроса, а также информацию о наличии конспекта.

Подписка на курс описывается классом CourseSubscription, который связывает пользователя и курс, учитывая такие параметры, как избранное, статус подписки, роли пользователя, версию курса и прогресс. Прогресс пользователя моделируется классом CourseProgress, включающим список разделов с прогрессом, где каждый раздел представлен классом CourseProgressSection с идентификатором и списком лекций. Прогресс лекций отражается классом CourseProgressLesson, содержащим информацию о просмотре видео, изучении конспекта и прохождении опроса. Лекции описываются классом CourseLesson с такими полями, как идентификатор, заголовок, продолжительность, изображение, текст конспекта, текст опроса и даты создания и обновления. Для лекций реализована система тестов, которая представлена классом CourseLessonSurvey с идентификатором, списком вопросов, порогом прохождения и максимальным количеством попыток. Вопросы теста описываются классом CourseLessonSurveyQuestion, включающим идентификатор, заголовок и список вариантов ответов. Каждый вариант ответа представлен классом CourseLessonSurveyChoice с идентификатором, заголовком и признаком правильного ответа. Попытки пользователя отражены классом CourseLessonSurveyAttempt, который содержит идентификатор, ссылки на опрос и пользователя, дату завершения, набранный балл и список ответов. Каждый ответ описывается классом CourseLessonSurveyAnswer с идентификатором выбранного и правильного вариантов.

Ассоциации между классами отображают связи между курсом и его автором, тегами, версиями, разделами, лекциями, подписками, прогрессом, опросами, вопросами, вариантами ответов и попытками пользователя, что обеспечивает целостность структуры данных для управления системой курсов.

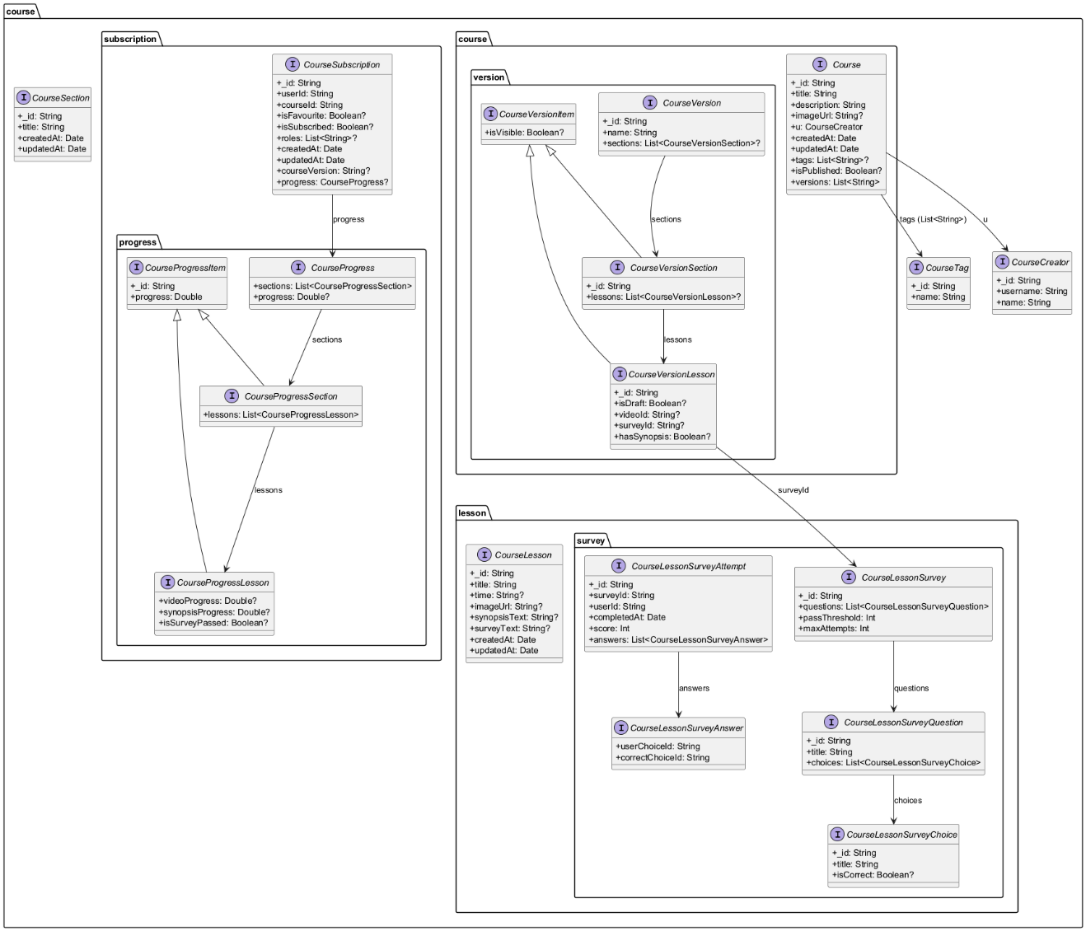


Рисунок 2.20 – Диаграмма классов системы управления курсами и учебным контентом

Таким образом, мы построили и последовательно рассмотрели все выбранные UML диаграммы, охватывающие как структурные аспекты системы, так и поведенческие. Это даёт нам чёткое представление о внутренней архитектуре системы, её компонентах, взаимодействиях между элементами и поведении на разных уровнях. Эти диаграммы позволяют более эффективно планировать и реализовывать систему, обеспечивая ясность в её проектировании и дальнейшем развитии.

# 2.2 Реализация системы и интеграция компонентов

В рамках реализации системы ключевую роль сыграла предварительная аналитическая работа, одним из результатов которой стала представленная на рисунке 2.21 майнд-карта. Она отражает логическую структуру проекта и формирует основу для последовательного изложения всех этапов разработки, начиная от исходного технического задания, предоставленного организацией, и заканчивая реализацией прикладных сервисов.

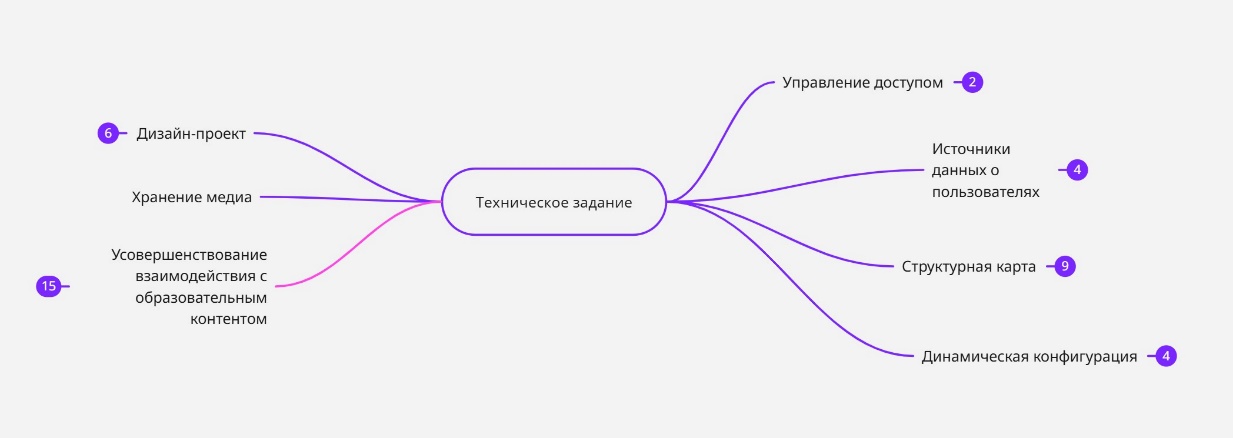


Рисунок 2.21 – Концептуальная карта основных направлений и решений, реализованных в ходе разработки

В центре карты — техническое задание, определяющее цели, требования и основные направления работы. От него расходятся ключевые блоки, каждый из которых раскрывает отдельный аспект реализации. На ранних этапах были разработаны дизайн-проект и структурная карта, определяющие логику пользовательских сценариев и архитектуру приложения. Далее — анализ и интеграция источников данных о пользователях, включая внешние системы авторизации. Существенное внимание было уделено организации хранения медиафайлов, обеспечению расширенного взаимодействия с образовательным контентом, а также внедрению динамической конфигурации, позволяющей изменять параметры системы без остановки её работы. Отдельный блок посвящён управлению доступом, в рамках которого реализована система ролей и прав, обеспечивающая разграничение функциональности в зависимости от типа пользователя и его статуса в системе.

Каждый из указанных узлов будет детально рассмотрен далее, с акцентом на принятые архитектурные решения, особенности интеграции и технические нюансы реализации.

Одним из ключевых этапов после утверждения архитектурных решений стало проектирование визуальной части платформы. Этот блок отражён в узле майнд-карты, представленной на рисунке 2.22, и охватывает основные аспекты дизайн-проекта: от общей концепции и логотипа до типографики и состояния интерфейсных компонентов. Естественно, что макеты страниц и модальные окна, разработанные в рамках этого этапа, напрямую соотносятся с пользовательскими интерфейсами, определёнными в структурной карте приложения, и в дальнейшем обеспечивают их целостную реализацию.

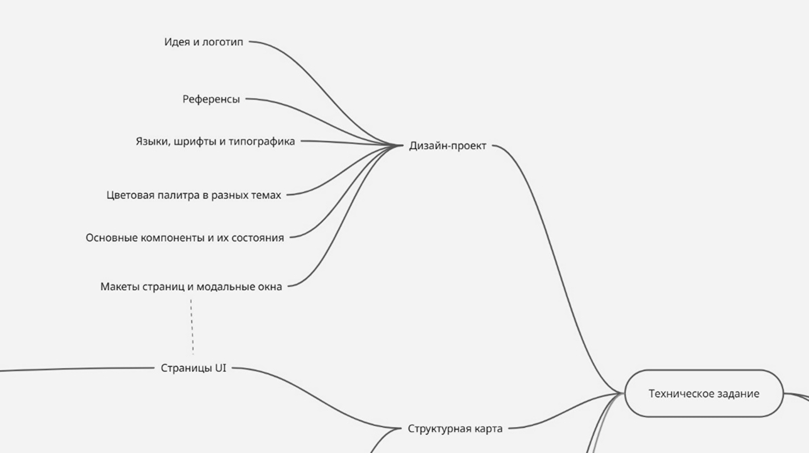


Рисунок 2.22 – Элементы дизайн-проекта и их связь со структурной картой пользовательских интерфейсов

Дизайн разрабатывался в Figma с упором на ключевые страницы веб-версии системы. Основная задача заключалась в формировании визуального стиля, соответствующего как требованиям организации-заказчика, так и современным стандартам в сфере цифровых образовательных продуктов. В основу легли стилистические гайды заказчика, на базе которых были определены компоненты интерфейса, их состояния, цветовая палитра, используемые шрифты и языки интерфейса.

Цветовая палитра интерфейса была сформирована на раннем этапе проектирования. Она включает основные акцентные и монохромные цвета, адаптирующиеся в зависимости от выбранной темы. Основные цвета приведены на рисунке 2.23.

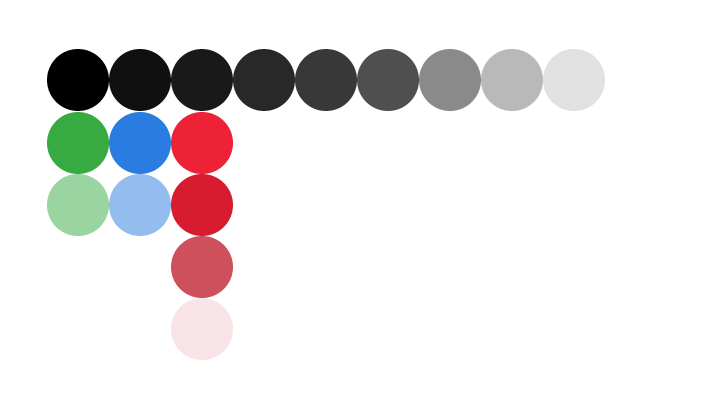


Рисунок 2.23 – Цветовая палитра интерфейса

Также была предусмотрена возможность масштабирования дизайн-системы, что критически важно для продуктов, ориентированных на широкий круг пользователей. Уже на данном этапе было решено, что система будет мультиязычной — с полной поддержкой русского и английского языков. Архитектура системы локализации изначально спроектирована с учётом масштабируемости: для добавления нового языка достаточно подключить один ещё файл с переводами.

Отдельное внимание при разработке было уделено формированию бренда создаваемой платформы. Её название — Flowledge — объединяет в себе понятия потока и знаний, подчёркивая ключевую идею проекта: непрерывное и поступательное обучение в рамках единой цифровой среды. Платформа ориентирована не только на конечных пользователей, но и на организации-клиенты, которым важно получать устойчивый, гибкий и современный инструмент для образовательного взаимодействия. Flowledge задумывалась как целостная экосистема, обеспечивающая долгосрочное развитие пользователя за счёт непрерывного доступа к знаниям, материалам и обучающим возможностям. Философия бренда воплощена в логотипе, визуально отражающем идеи движения, роста и устойчивости, а также в общем стиле интерфейса, выстроенного на принципах минимализма, функциональной ясности и открытости к пользователю. Логотип представлен на рисунке 2.24.

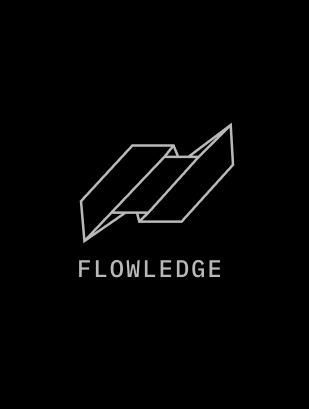


Рисунок 2.24 – Логотип платформы

Перед созданием макетов был собран ряд референсов — анализ аналогичных успешных образовательных систем, представленных на рынке. Это позволило избежать излишней сложности в дизайне и сосредоточиться на лучших практиках взаимодействия с пользователем.

Одной из особенностей платформы является наличие тем оформления. По умолчанию используется тёмная тема, которая наиболее органично вписывается в современный тренд цифровых продуктов, однако также предусмотрены светлая и системная темы, соответствующие настройкам операционной системы пользователя. Это обеспечивает максимальную доступность и комфорт при использовании платформы на разных устройствах и в различных условиях.

Все тексты в интерфейсе намеренно оформлены с использованием строчных букв — такой подход делает взаимодействие с системой менее формальным, создаёт более дружелюбную атмосферу и подчёркивает общий минималистичный подход в дизайне. Это решение согласуется с выбранной визуальной концепцией и усиливает ощущение простоты и лёгкости при работе с платформой.

Таким образом, визуальная и функциональная основа проекта была тщательно проработана ещё до начала непосредственной реализации, что позволило обеспечить целостность продукта и избежать существенных переделок на более поздних этапах разработки.

Следующим логичным этапом стало формирование структурных карт платформы. Были выделены два взаимосвязанных направления: карта пользовательских интерфейсных страниц и система API-маршрутов, детализированная отдельно для каждого сервиса. В процессе проектирования как интерфейсные страницы, так и маршруты API были классифицированы по типам доступа: безусловный доступ, доступ после аутентификации и доступ, зависящий от роли пользователя.

Безусловный доступ предоставляется исключительно для маршрутов, отвечающих за первичную авторизацию и обновление токенов. Все остальные действия требуют наличия действительного JWT-токена, передаваемого через cookie. Для маршрутов с ограничением по ролям дополнительно осуществляется проверка полномочий пользователя, таких как права администратора или редактора.

Аналогичным образом организован и доступ к интерфейсным страницам: в зависимости от наличия токена и прав, пользователю отображаются определённые разделы системы. Тем самым достигается согласованность логики доступа между серверной и клиентской частями, что чётко зафиксировано на рисунке 2.25, где представлены взаимосвязи между API-маршрутами, страницами интерфейса, типами доступа и ролевой моделью.

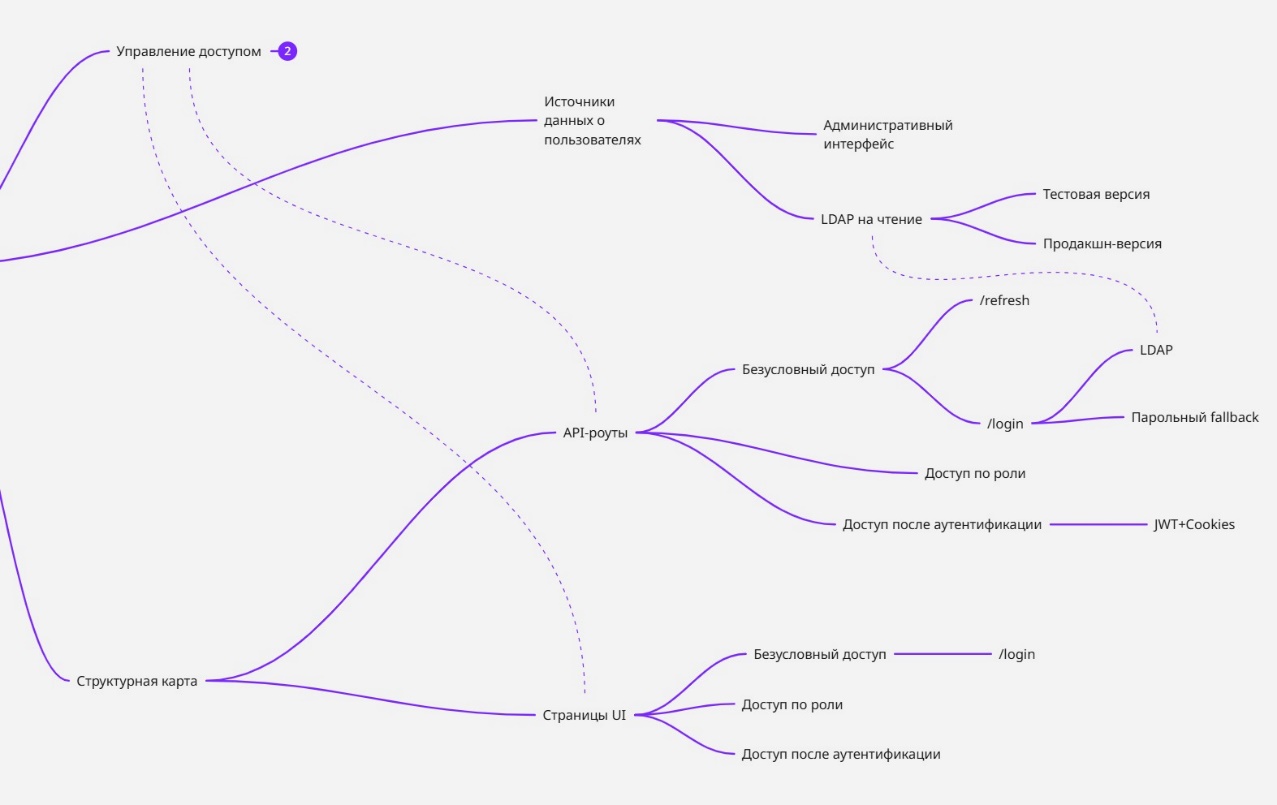


Рисунок 2.25 – Архитектура доступа: маршруты, интерфейс и пользовательские данные

Такой подход ставит вопрос происхождения данных о пользователях. В проекте предусмотрены два источника, также отмеченные на майнд-карте: LDAP-сервер с доступом на чтение – используемый как в тестовой, так и в продакшн-среде – и административный интерфейс, предназначенный для ручного управления записями пользователей. Эти источники дополняют друг друга и формируют единую систему, обеспечивающую аутентификацию и авторизацию на всех уровнях.

Структурная карта пользовательского бэкенда или User Sphere API, представленная на рисунке 2.26. В рамках схемы используется система обозначений для визуального разграничения уровней доступа: свободный доступ отмечается звёздочкой в зелёном круге, маршруты, требующие наличия определённой роли — флажком в красном круге, все остальные, предполагающие базовую аутентификацию, отображаются без маркировки. Такая система позволяет визуально сфокусироваться на ключевых аспектах разграничения доступа, не перегружая схему лишними деталями.

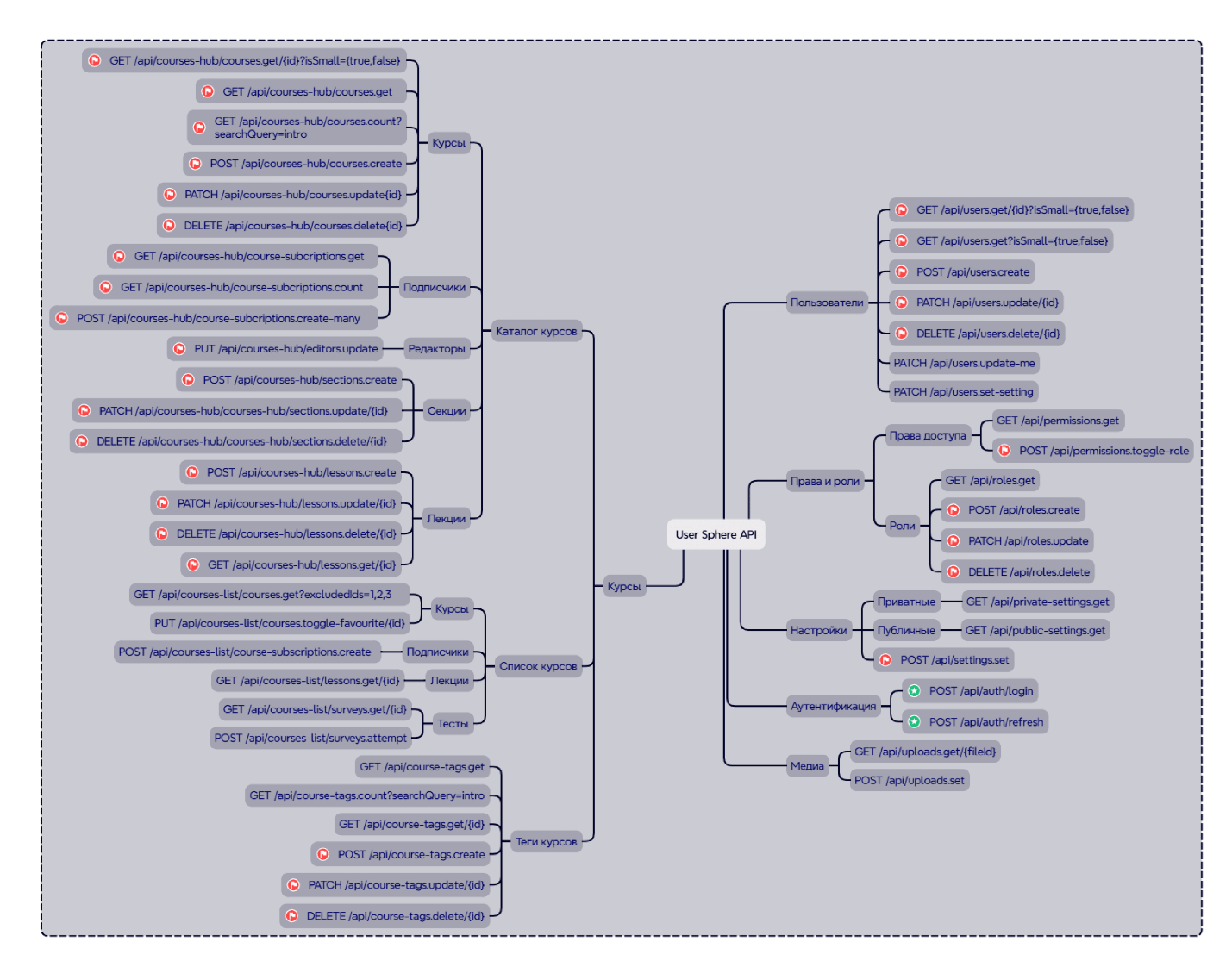


Рисунок 2.26 – Структурная карта User Sphere API

В данной карте описаны все маршруты, относящиеся к пользовательскому API. Она охватывает операции с пользовательскими профилями, включая как работу с собственными данными пользователя, например обновление информации о себе, так и административные действия — создание, удаление и обновление данных других пользователей. Отдельное внимание уделено управлению правами доступа и ролями, что включает получение перечня доступных прав, назначение ролей, создание и изменение моделей ролей. Система настроек организована с разделением на приватные и публичные настройки, что обеспечивает гибкое управление конфигурацией пользовательского окружения.

Аутентификация реализована через два маршрута — вход в систему и обновление токена. Эти маршруты единственные, для которых не требуется предварительная авторизация, что соответствует модели безусловного доступа.

Кроме того, карта содержит маршруты, отвечающие за работу с медиафайлами, включая загрузку и получение контента.

Значительную часть карты занимает система работы с курсами, включающая управление курсами, подписками, редакторами, секциями, лекциями и тестами. Структура организована таким образом, чтобы четко отделить административный блок — каталог курсов — от пользовательского блока — списка курсов, с соответствующим разграничением маршрутов и уровней доступа.

В завершение реализована поддержка системы тегов курсов, которая служит для категоризации и упрощения поиска образовательного контента.

После подробного рассмотрения пользовательского бэкенда, следует перейти к описанию двух других ключевых сервисов платформы — Synopsis Forge API и Integrations Runner API. Их структурные карты приведены на рисунке 2.27. В отличие от User Sphere, доступ к этим сервисам осуществляется исключительно по ролям, что отражает более строгие требования к безопасности и разграничению полномочий.

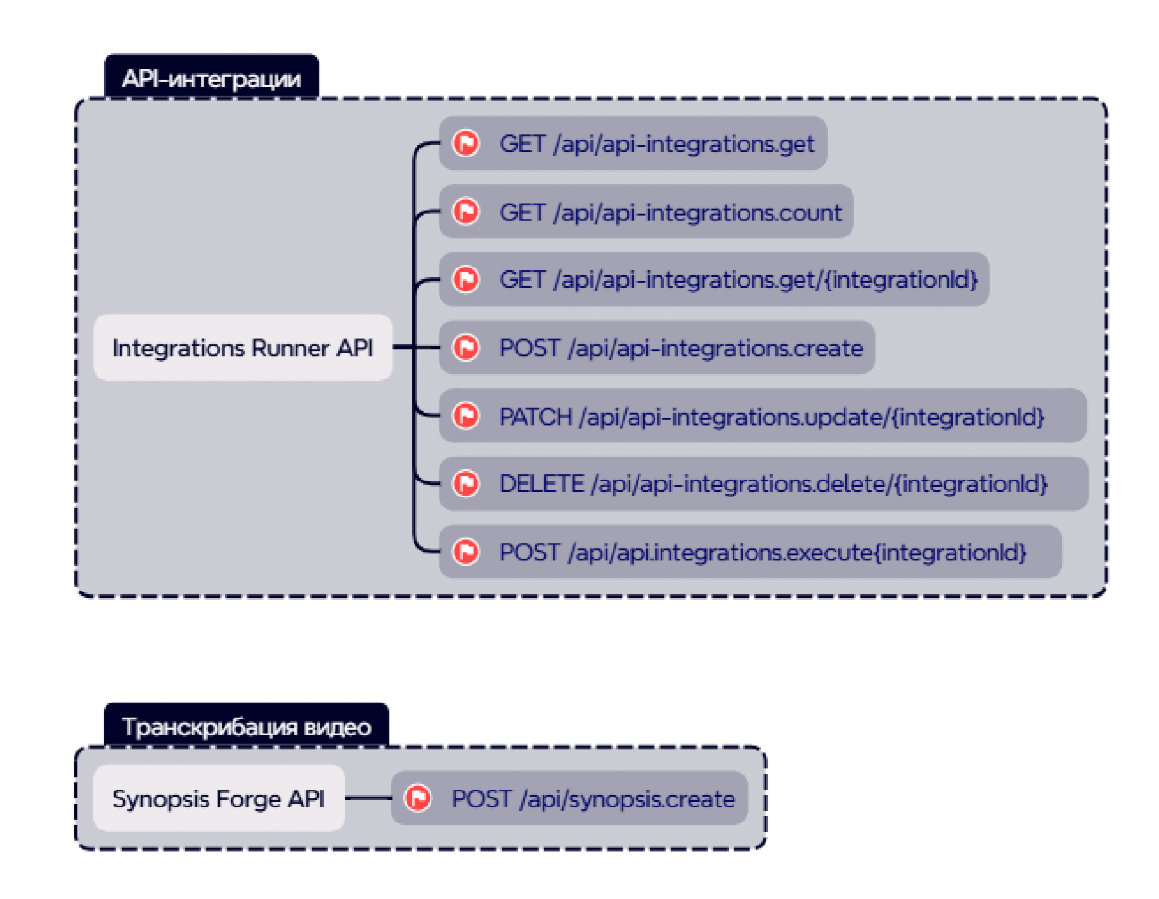


Рисунок 2.27 – Структурные карты API Synopsis Forge и Integrations Runner

Synopsis Forge API представлен одной основной операцией — созданием конспектов, что является специализированной задачей в рамках платформы. Integrations Runner API обеспечивает полный набор методов для управления внешними интеграциями: получение списка интеграций и их количества, просмотр конкретной интеграции, создание, обновление и удаление, а также выполнение интеграционных процессов. Все эти операции доступны только пользователям с соответствующими правами, что гарантирует контролируемое и безопасное взаимодействие с внешними системами.

После описания сервисов Synopsis Forge API и Integrations Runner API, логичным продолжением станет рассмотрение структурной карты фронтенда — пользовательского интерфейса системы. Здесь применяются те же обозначения, что и на карте бэкендов: доступ по роли, доступ после аутентификации и безусловный доступ. Дополнительно в интерфейсной карте используется иконка пользователя в оранжевом круге, которая указывает, что определённые страницы доступны только самому пользователю, например, его профиль или личные настройки. Она представлена на рисунке 2.28.

В структуре фронтенда отражены основные разделы: управление пользователями, права и роли, API-интеграции, приватные и публичные настройки, аутентификация, а также работа с курсами и тегами курсов. Каждый из этих разделов содержит соответствующие страницы и режимы работы, такие как просмотр, редактирование или создание.

Эти взаимосвязи и уровни доступа наглядно представлены на рисунке, который иллюстрирует структурную карту фронтенда с обозначениями прав доступа и ролей.

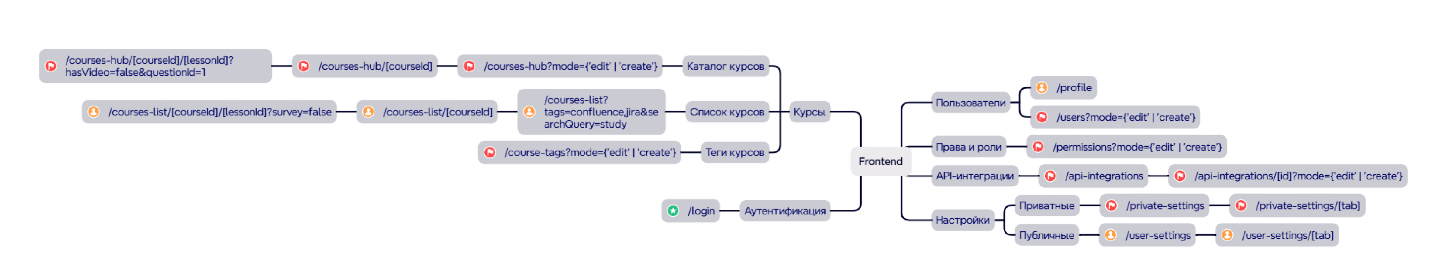


Рисунок 2.28 – Структурная карта фронтенда

Завершив рассмотрение всех структурных карт, включая карты API-маршрутов и страниц интерфейса, целесообразно перейти к следующему ключевому блоку — источникам данных о пользователях. Их роль в системе критически важна, поскольку именно на них опирается архитектура авторизации и распределения прав доступа. На рисунке 2.29, представляющем текущий узел майндкарты, показана связь этого блока с динамической конфигурацией и системой управления доступом, что подчёркивает его интеграционную значимость в рамках всей платформы.

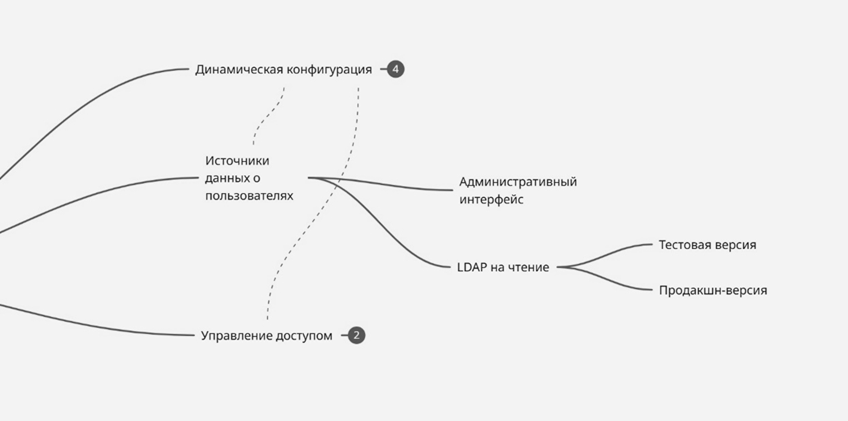


Рисунок 2.29 – Взаимосвязь управления доступом, динамической конфигурации и источников данных о пользователях

В рамках реализации системы аутентификации особое внимание было уделено интеграции с LDAP — широко применяемым протоколом доступа к информации о пользователях в корпоративных инфраструктурах [10]. Использование LDAP в подобных системах оправдано тем, что он позволяет централизованно управлять данными о сотрудниках, их принадлежностью к организациям, а также групповой структурой, отражающей ролевое распределение или функциональную специализацию внутри компании.

Для целей разработки и тестирования заказчиком был предоставлен отдельный тестовый LDAP-сервер с ограниченным доступом и заранее подготовленными данными, включая тестовые учётные записи, организации и группы. Доступ к этому серверу осуществлялся только в режиме чтения, что соответствует принятым практикам безопасности: система не должна изменять корпоративные данные, а лишь считывать их для нужд аутентификации и авторизации [10].

Информация о пользователях в LDAP представлена в стандартной иерархической форме. Каждая запись имеет уникальный DN или Distinguished Name, включающий организационную структуру [11], например: uid=testuser,ou=demo-org,dc=example,dc=org. Такая структура позволяет системе точно определять, к какой организации относится пользователь, и при необходимости — ограничивать доступ сотрудникам организаций, не входящих в список допустимых. Типичное содержимое записи пользователя включает в себя следующие атрибуты:

objectClass: inetOrgPerson

objectClass: organizationalPerson

objectClass: person

objectClass: top

cn: Test User

sn: User

uid: testuser

mail: testuser@example.org

userPassword: testpassword

Помимо индивидуальных атрибутов, каждая учётная запись может быть членом одной или нескольких LDAP-групп. Группы используются для логического объединения пользователей, как по организационному признаку, так и по функциональному [11] — например, сотрудники склада, менеджеры или составители курсов. Пример записи группы:

dn: cn=testgroup,dc=example,dc=org

objectClass: top

objectClass: groupOfNames

cn: testgroup

member: uid=testuser,dc=example,dc=org

При авторизации через LDAP система использует введённые пользователем учётные данные для проверки доступа. В случае успешной аутентификации из LDAP извлекаются основные данные о пользователе — ФИО, имя пользователя, адрес электронной почты, а также список групп, в которые он входит. Эти данные изначально формируются в корпоративной системе учёта — 1С, и оттуда автоматически синхронизируются в LDAP-репозиторий, однако доступ непосредственно к 1С отсутствует и не требуется: взаимодействие осуществляется только через LDAP-интерфейс. На основании полученной информации система определяет права доступа и наполняет пользовательский профиль. Регистрация пользователей в системе как отдельная процедура не предусмотрена: при первом успешном входе по LDAP данные автоматически сохраняются в базу, и с этого момента пользователь может полноценно работать с системой.

Тем не менее, в ряде случаев возникает необходимость в создании дополнительных тестовых учётных записей, не связанных с LDAP. Это может потребоваться, например, для разработки, тестирования отдельных модулей или временного предоставления доступа сторонним пользователям.

Также предусмотрен fallback-механизм — локальная аутентификация через заранее заданные параметры в базе данных. Суть заключается в том, что после успешного входа через LDAP система сохраняет хэш пароля в собственной базе. Это позволяет в дальнейшем производить аутентификацию пользователя даже в случае временной недоступности LDAP-сервера или сбоев в сетевом соединении. Таким образом, реализуется отказоустойчивый механизм входа с сохранением единых учётных данных, но без нарушения принципов безопасности.

Таким образом, LDAP в рамках архитектуры платформы выступает в качестве основного внешнего источника информации о пользователях. Из него извлекаются ключевые атрибуты, включая уникальные идентификаторы, имена, адреса электронной почты, принадлежность к организациям и группам, а также ролевые признаки, используемые для распределения прав доступа внутри системы. Эти данные синхронизируются с внутренним пользовательским репозиторием и становятся основой для механизмов авторизации, настройки профилей и назначения ролей.

На уровне реализации взаимодействие с LDAP в пользовательском бэкенде организовано с использованием Kotlin, Spring Boot и ряда специализированных библиотек — включая spring-ldap-core, spring-security-ldap и spring-boot-starter-security. Вся логика аутентификации и интеграции с LDAP целиком сосредоточена в одном сервисе — пользовательском бэкенде. Остальные сервисы, в том числе отвечающие за взаимодействие с нейросетями, не участвуют в процессе авторизации: они лишь валидируют JWT-токен, выданный пользователю. Обновление токена или повторная аутентификация также происходят исключительно через пользовательский сервис.

LDAP-интеграция реализована на основе стандартного механизма BindAuthenticator, инициализируемого через SecurityConfig [12, 13]. Конфигурация подключения гибкая и не зашита в код или application.properties — все параметры получаются динамически из централизованного сервиса настроек, что позволяет оперативно менять поведение системы без пересборки или повторного деплоя. Это особенно удобно в корпоративной среде, где могут использоваться разные LDAP-сервера – локальный, тестовый или продовый – с различными параметрами подключения.

@Configuration  
@EnableWebSecurity  
class SecurityConfig(  
 private val tokenService: TokenService,  
 private val ldapService: LDAPService  
) {

…  
 @Bean  
 fun ldapAuthenticationProvider(): LdapAuthenticationProvider {  
 val ldapUrl = ldapService.getUrl()  
 val adminDn = ldapService.getAdminDn()  
 val adminPassword = ldapService.getAdminPassword()  
 val baseUserDn = ldapService.getUserDn()  
 val userSearchFilter = ldapService.getUserSearchFilter()  
  
 val contextSource = DefaultSpringSecurityContextSource(ldapUrl)  
 contextSource.userDn = adminDn  
 contextSource.password = adminPassword  
 contextSource.afterPropertiesSet()  
  
 val authenticator = BindAuthenticator(contextSource)  
  
 authenticator.setUserSearch(  
 FilterBasedLdapUserSearch(  
 baseUserDn,  
 userSearchFilter,  
 contextSource  
 ).apply {  
 setReturningAttributes(arrayOf("uid", "mail", "cn", "sn", "memberOf"))  
 }  
 )  
  
 val provider = LdapAuthenticationProvider(authenticator)  
 provider.setUserDetailsContextMapper(CustomLdapUserDetailsMapper())  
  
 return provider  
 }  
  
 @Bean  
 fun authenticationManager(): AuthenticationManager {  
 val providers = mutableListOf<AuthenticationProvider>()  
  
 if (ldapService.isEnabled()) {  
 providers.add(ldapAuthenticationProvider())  
 }  
  
 return ProviderManager(providers, NoOpAuthenticationManager())  
 }  
  
 class NoOpAuthenticationManager : AuthenticationManager {  
 override fun authenticate(authentication: Authentication): Authentication? {  
 return null  
 }  
 }  
}

Настройки LDAP, включая адрес сервера, порт, DN и пароль сервисной учётной записи, базовый DN пользователей, фильтр поиска и другие параметры, запрашиваются из базы данных через сервис SettingService и кэшируются внутри приложения. Такой подход позволяет повторно использовать ранее загруженные параметры без необходимости повторных запросов, что сокращает время отклика и уменьшает нагрузку на сервис настроек. Кэширование данных в Spring реализуется с помощью встроенных механизмов, обеспечивая эффективный и быстрый доступ к конфигурации [14]. Например, при конфигурации URL подключения автоматически определяется протокол — ldap или ldaps — в зависимости от порта, а остальные параметры передаются напрямую в BindAuthenticator и соответствующие компоненты Spring Security.

@Service  
class LDAPService(private val settingService: SettingService) : StringParser {  
 companion object {  
 const val PREFIX = "ldap."  
 }  
  
 fun getUrl(): String {  
 val host = settingService.getSetting("${PREFIX}connection.host", String::class)  
 val port = settingService.getSetting("${PREFIX}connection.port", Int::class)  
  
 val protocol = if (port == 636) "ldaps" else "ldap"  
 return "$protocol://$host:$port"  
 }  
  
 fun isEnabled(): Boolean {  
 return settingService.getSetting("${PREFIX}enabled", Boolean::class)  
 }

…

}

Таким образом, LDAP-интеграция максимально адаптирована под корпоративные требования: при смене среды достаточно изменить параметры в настройках, чтобы подключиться к другому LDAP-серверу.

С точки зрения безопасности, система имеет доступ к LDAP только в режиме чтения, используя выделенную сервисную учётную запись. Это соответствует лучшим практикам и позволяет гарантировать, что никакие изменения в корпоративном каталоге не могут быть инициированы из приложения.

В тех случаях, когда проверка через LDAP не приводит к успешной аутентификации — например, если пользователь отсутствует в корпоративной структуре или подключение к серверу временно недоступно, — система инициирует резервную процедуру локальной проверки пароля.

На первом этапе выполняется поиск пользователя по логину в MongoDB. Если пользователь найден, из его модели извлекается ранее сохранённый хэш пароля, который находится в поле services.password.bcrypt. Хэш создаётся с использованием алгоритма BCrypt — криптографически устойчивой функции, специально разработанной для хранения паролей [15]. Он включает в себя встроенную соль и параметр сложности, что обеспечивает защиту от атак перебора и использования rainbow-таблиц.

Введённый пользователем пароль сравнивается с хэшем с помощью функции BCrypt.checkpw(...), которая безопасно выполняет проверку без раскрытия оригинального значения [15]. Если сравнение проходит успешно, пользователь считается аутентифицированным.

После успешной аутентификации создаются и возвращаются JWT-токен и refresh-токен, как и в случае LDAP-входа. При этом группы – которым соответствует атрибут memberOf – не присваиваются, так как они получаются только при LDAP-аутентификации.

fun passwordLogin(username: String, password: String): Mono<UserModel> {  
 return userRepository.findByUsername(username)  
 .switchIfEmpty(Mono.error(IllegalArgumentException("User not found")))  
 .flatMap **{** user **->** val hashedPassword = user?.services?.password?.bcrypt  
 if (hashedPassword != null && BCrypt.checkpw(password, hashedPassword)) {  
 Mono.just(user)  
 } else {  
 Mono.error(IllegalArgumentException("Invalid password"))  
 }  
 **}**}

Такой двухуровневый механизм позволяет системе оставаться доступной даже при недоступности внешнего LDAP-сервера, обеспечивая надёжную и безопасную локальную авторизацию для пользователей, имеющих пароль, зарегистрированный в системе.

На майндкарте, представленной ранее, эта логика отображена в контексте узла, отвечающего за способ доступа после аутентификации. Именно с этого уровня начинается работа с сессионными токенами и разграничение доступа на основе идентифицированного пользователя. В рамках этой модели применяется пара токенов — jwt-токен и refresh-токен, которые генерируются при входе в систему и сохраняются в виде HTTP-only cookies.

Все маршруты приложения по умолчанию защищены: доступ к ним возможен только при наличии валидного токена. Исключение составляют открытые точки входа, такие как /login и /refresh, предназначенные для аутентификации и обновления токенов. После успешного входа или подтверждения refresh-токена в базе данных производится обновление документа пользователя, в который сохраняются новые значения токенов.

@Configuration  
@EnableWebSecurity  
class SecurityConfig(  
 private val tokenService: TokenService,  
 private val ldapService: LDAPService  
) {  
  
 @Bean  
 fun securityFilterChain(http: HttpSecurity): SecurityFilterChain {  
 http  
 .csrf { it.disable() }  
 .sessionManagement { it.sessionCreationPolicy(SessionCreationPolicy.STATELESS) }  
 .securityContext { it.requireExplicitSave(false) }  
 .authorizeHttpRequests { requests ->  
 requests  
 .requestMatchers("/websocket").permitAll()  
 .requestMatchers(HttpMethod.OPTIONS, "/api/\*\*").permitAll()  
 .requestMatchers("/api/auth/\*\*").permitAll()  
 .requestMatchers("/api/\*\*").authenticated()  
 .anyRequest().authenticated()  
 }  
 .exceptionHandling { exceptions ->  
 exceptions.authenticationEntryPoint { \_: HttpServletRequest, response: HttpServletResponse, authException: AuthenticationException ->  
 response.status = HttpServletResponse.SC\_UNAUTHORIZED  
 response.writer.println("Unauthorized: ${authException.message}")  
 }  
 }  
 .addFilterBefore(JwtTokenFilter(tokenService), UsernamePasswordAuthenticationFilter::class.java)  
  
 return http.build()  
 }

…

}

Валидация токенов и извлечение информации из них происходит в фильтре JwtTokenFilter, добавленном в цепочку обработки запросов. Этот фильтр анализирует входящие HTTP-запросы, извлекает токен из cookies и, в случае его валидности, устанавливает текущего пользователя в контекст безопасности. При невалидном или просроченном токене фильтр немедленно возвращает ошибку авторизации, и запрос не проходит дальше по цепочке. Такой подход позволяет централизованно контролировать доступ к ресурсам приложения и минимизирует количество логики, связанной с безопасностью, в бизнес-слоях.

Аналогичная проверка выполняется и при установлении WebSocket-соединения. На этапе рукопожатия используется JwtHandshakeInterceptor, который извлекает токен из cookies и валидирует его через TokenService. Если токен действителен, идентификатор пользователя сохраняется в атрибутах сессии, и соединение считается авторизованным. В противном случае соединение не устанавливается.

Из токена извлекается идентификатор пользователя, что позволяет избежать необходимости передачи всей информации о пользователе с клиента. В современных API токены содержат минимальный набор информации для идентификации [16], их можно валидировать без постоянного обращения к базе данных, что повышает производительность и упрощает масштабирование. Полные данные о пользователе запрашиваются на сервере при необходимости, например, для определения его групп, прав или других параметров. Это упрощает клиентскую логику и делает взаимодействие более безопасным и централизованным.

Поскольку извлечённый идентификатор является ключом к определению полномочий пользователя, следующим логичным компонентом архитектуры становится система управления доступом. Она реализована через механизм прав и ролей, каждый из которых играет отдельную, но взаимосвязанную роль в разграничении функций и обязанностей. В майндкарте данный механизм представлен в виде узла «Управление доступом», разделённого на два подузла: «Роли» и «Права доступа». Эти структурные элементы зафиксированы на рисунке 2.30.

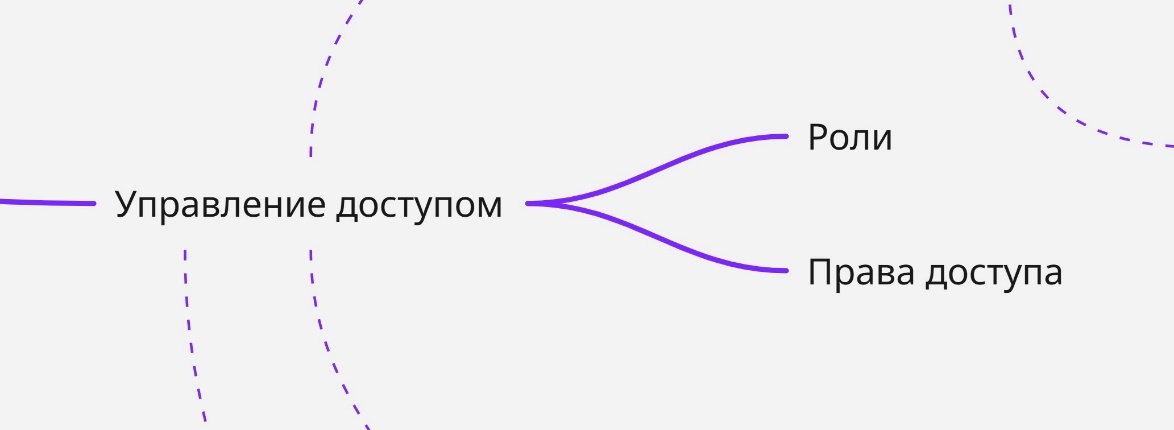


Рисунок 2.30 – Узел управления доступом

В рамках такой модели предусмотрена гибкость: поведение ролей может адаптироваться в процессе эксплуатации, тогда как набор прав остаётся фиксированным на уровне кода и конфигурации. Это обеспечивает надёжность системы при сохранении возможности адаптации — можно переопределить, какие роли обладают теми или иными правами, но невозможно в рантайме создавать или удалять сами типы прав.

Базовая роль в системе — это user. Она автоматически присваивается при первом входе через LDAP или при создании пользователя через административный интерфейс. Это происходит, если не указано иное явно. Все предопределённые роли создаются в момент инициализации приложения, и, если какая-либо из них уже присутствует в базе, она не будет пересоздана. Реализация этого механизма находится в классе PermissionsRegistry. При запуске метод initializeDefaultRolesAndPermissions() сначала проверяет наличие базовых ролей, таких как admin, user, editor, owner, moderator. Если в базе отсутствуют какие-то из них, они добавляются через репозиторий RoleRepository. Каждая роль описывает область применения или scope — например, RoleScope.USERS или RoleScope.COURSES, что позволяет разделять ответственность между разными типами ролей. Далее выполняется аналогичная проверка по правам доступа — метод addMissingPermissions() сверяет список предопределённых прав с теми, что уже есть в базе, и, при необходимости, добавляет недостающие. Каждый PermissionModel связывает конкретное право с набором ролей, которые этим правом обладают.

private fun addMissingRoles(): Mono<Void> {  
 val defaultRoles = listOf(  
 RoleModel(DefaultRoles.ADMIN.toLowerCase(), "Users", "admin role", listOf(RoleScope.USERS)),  
 RoleModel(DefaultRoles.USER.toLowerCase(), "Users", "user role", listOf(RoleScope.USERS)),  
 RoleModel(  
 DefaultRoles.EDITOR.toLowerCase(),  
 "Users",  
 "editor role",  
 listOf(RoleScope.USERS, RoleScope.COURSES)  
 ),  
 RoleModel(DefaultRoles.OWNER.toLowerCase(), "Courses", "owner role", listOf(RoleScope.COURSES)),  
 RoleModel(DefaultRoles.MODERATOR.toLowerCase(), "Courses", "moderator role", listOf(RoleScope.COURSES)),  
 )  
  
 return roleRepository.findAll()  
 .map { it.name }  
 .collectList()  
 .flatMap { existingRoleNames ->  
 val rolesToAdd = defaultRoles.filter { it.name !in existingRoleNames }  
 if (rolesToAdd.isNotEmpty()) {  
 roleRepository.saveAll(rolesToAdd).collectList().then()  
 } else {  
 Mono.empty()  
 }  
 }  
}

Среди прав доступа в системе предусмотрены как общие права, вроде возможности просматривать или редактировать настройки, так и более специализированные, например управление курсами, доступ к подписчикам или управление ролями других пользователей. Права типа view-private-settings, edit-private-settings, view-all-users, manage-integrations предоставляются только администраторам. В то время как такие права, как view-assigned-courses или view-subscribers, могут быть доступны и редакторам, и модераторам, и владельцам курсов, и даже обычным пользователям. Это позволяет точно контролировать, кто имеет доступ к тем или иным действиям в зависимости от текущего контекста.

Проверка наличия прав осуществляется как на клиентской, так и на серверной стороне. Клиент при инициализации получает список своих прав, определённых через назначенные роли, и на его основе строится пользовательский интерфейс. Например, скрываются или отображаются определённые действия, кнопки или разделы. Сервер, в свою очередь, не доверяет только клиентской информации и повторно проверяет наличие прав перед выполнением чувствительных операций. Это реализуется как в пользовательском бэкенде, так и в связанными с нейросетями бэкендами, если они обращаются к данным, доступ к которым должен быть ограничен.

Права и роли, как и источники данных о пользователях, тесно связаны с системой динамической конфигурации, которая раскрыта на рисунке 2.31 – фрагменте майндкарты с выделенным одноимённым узлом. Ранее было показано, как из узла динамической конфигурации расходятся связи к источникам данных о пользователе и системе управления доступом. Это подчёркивает роль конфигурации как центра принятия решений в приложении — от параметров подключения к внешним источникам до определения условий доступа и поведения клиентского интерфейса.

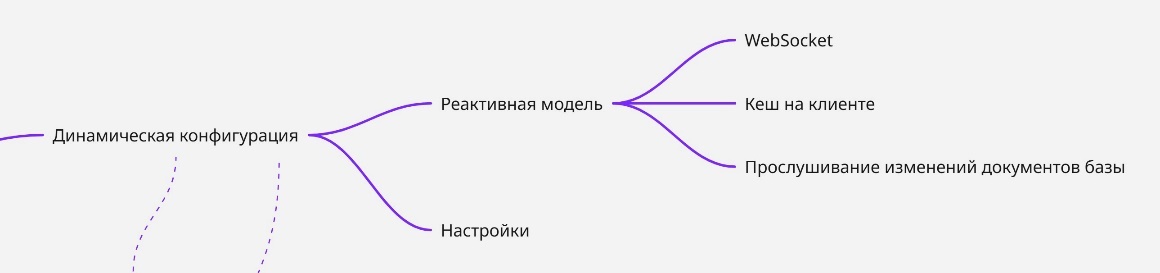


Рисунок 2.31 – Фрагмент общей майндкарты с раскрытием понятия динамической конфигурации

Динамическая конфигурация в системе реализована через настраиваемые параметры, которые могут быть изменены без перезапуска приложения. Это критично как для конфигурации LDAP, так и для управления доступом, где необходимо мгновенно реагировать на изменение ролей или прав. Чтобы такие изменения становились видимыми в клиенте без задержек, применяется реактивная модель — обновления отслеживаются и рассылаются в реальном времени.

Ключевым элементом, обеспечивающим такую реактивность, выступает архитектура хранения и обработки данных. Здесь важно перейти к рассмотрению того, как устроено само хранилище и каким образом реализована логика чтения, записи и подписки на изменения в нём.

Во всём приложении используется MongoDB — она выбрана в качестве основной и единственной базы данных. Это решение было принято на этапе проектирования. Вся работа с хранилищем реализована через spring-boot-starter-data-mongodb и spring-boot-starter-data-mongodb-reactive. Благодаря этому доступ к данным осуществляется в неблокирующем стиле, а вся архитектура остаётся согласованной и унифицированной.

Как и положено в типичном Spring Boot приложении, каждая сущность в базе описывается с помощью аннотации @Document [17], но в нашем случае модели не создаются с нуля в каждом модуле. Вместо этого они берутся из модуля coretypings, который представляет собой внутренний пакет, размещённый рядом с основным проектом в монорепозитории и публикуемый локально через ./gradlew publishToMavenLocal [18]. Включение модуля осуществляется через стандартную зависимость implementation("flowledge:coretypings:1.0.0").

Взаимодействие с MongoDB реализовано через реактивные репозитории, основанные на интерфейсе ReactiveMongoRepository. Такой подход позволяет выполнять операции с базой данных асинхронно, эффективно масштабируя приложение без блокировки потоков [19]. При этом Spring Data MongoDB автоматически генерирует реализацию методов репозитория на основе их сигнатур — достаточно корректно указать название метода, чтобы получить рабочий запрос к базе. Это значительно упрощает работу с данными и уменьшает объём шаблонного кода.

Для работы с настройками используется репозиторий PrivateSettingRepository, инкапсулированный в SettingService, который предоставляет типобезопасный доступ к значениям и реализует механизм кэширования для повышения производительности.

Однако при необходимости выполнения более сложных запросов, которые не могут быть корректно сгенерированы только по названию метода, требуется вручную реализовывать соответствующую логику. Это может включать как использование аннотаций @Query, так и реализацию кастомных методов репозитория через ReactiveMongoTemplate.

@Repository  
interface PrivateSettingRepository : ReactiveMongoRepository<PrivateSettingModel, String> {  
 @Query("{ '\_id': { \$regex: ?0 } }")  
 fun findByRegexId(regex: String): Flux<PrivateSettingModel>  
}

Еще один элемент системы для работы с настройками – сервис SettingsRegistry. Он отвечает за декларативную и централизованную инициализацию всех поддерживаемых системой настроек. Этот сервис вызывается при старте приложения и создаёт – или обновляет – документы настроек в базе. Такая организация позволяет безопасно и прозрачно управлять параметрами даже в условиях ограниченного доступа к продовой базе данных, когда запрещены ручные изменения в коллекции настроек. Все миграции или добавления новых параметров происходят исключительно через код, что гарантирует согласованность и поддержку со стороны клиента, а также возможность управления версионностью изменений.

Каждая настройка логически группируется — например, в user-default, ldap, file-upload, security, discover и другие. Группа может содержать одну или несколько вкладок, таких как connection, auth, user, group в случае настроек для LDAP. Это позволяет логически структурировать параметры и облегчает их отображение на клиентской стороне.

Формирование уникального идентификатора каждой настройки происходит на основе текущей группы, вкладки — если она есть — и названия самой настройки — например, ldap.connection.host или user-default.appearance.theme. Благодаря такой структуре достигается прозрачная иерархия и удобство интернационализации, ведь i18n-ключи автоматически формируются по тому же шаблону: ldap.connection.host.name, ldap.connection.host.description и так далее. Использование подобной унифицированной схемы ключей соответствует общепринятым практикам международной локализации, где структурированные ключи позволяют легко связывать тексты с соответствующими элементами интерфейса и упрощают поддержку нескольких языков без дублирования кода и данных [20]. Это значительно облегчает масштабирование приложения на новые регионы и поддерживает консистентность переводов в разных частях системы.

Создание настроек на лету в рантайме категорически запрещено. Все параметры конфигурируются и описываются в коде, поскольку без этого клиентское приложение не сможет корректно работать с настройками — не будет ни визуального интерфейса, ни правильных переводов, ни возможности валидировать типы или отображать select-опции. Таким образом, вся логика добавления новой настройки проходит исключительно через изменение исходного кода и последующую регенерацию конфигурации. Это ключевой аспект динамической конфигурации, который детально показан на соответствующем фрагменте общей майндкарты.

Во внутреннем устройстве проекта широко применяются мапперы, построенные на базе библиотеки MapStruct. Они обеспечивают удобное и безопасное преобразование данных между различными слоями приложения — например, из модели в DTO и обратно. DTO используются в контроллерах и при сетевом обмене данными, а контроллеры, в свою очередь, взаимодействуют с сервисами, такими как PrivateSettingService или LDAPService, которые непосредственно получают необходимые настройки из SettingService. Таким образом, архитектура системы настроек выстроена строго и прозрачно.

В пользовательском бэкенде реализована реактивная модель синхронизации данных между сервером и клиентом. Эта модель базируется на возможностях MongoDB, Spring WebSocket и локального кеша IndexedDB через LokiJS. Она позволяет обеспечивать мгновенное обновление настроек и прав доступа на клиенте при изменениях на сервере, что делает динамическую конфигурацию по-настоящему живой и актуальной. Эти процессы подробно раскрыты в разделе динамической конфигурации на майндкарте, демонстрируя, как технические решения обеспечивают высокую отзывчивость и согласованность системы.

На серверной стороне ключевым элементом является абстрактный класс AbstractMongoChangeStreamListener<T>. Этот класс предназначен для прослушивания изменений в MongoDB в режиме реального времени [21]. При инициализации в методе listenToChanges() происходит подписка на Change Stream конкретной коллекции. Для этого используется ReactiveMongoTemplate и фильтрация по операциям вставки, обновления, удаления и замены. Имя коллекции определяется реализацией метода getCollectionName(). Когда поступает событие изменения, метод handleChange() вызывается с информацией о типе операции – insert, update, delete, replace – и самим объектом данных.

abstract class AbstractMongoChangeStreamListener<T : Any> {  
  
 @Autowired  
 private lateinit var mongoTemplate: ReactiveMongoTemplate  
  
 private val logger: MainLogger = MainLogger(this.*javaClass*.*simpleName*)  
  
 abstract val entityClass: Class<T>  
  
 @PostConstruct  
 fun listenToChanges() {  
 val collectionName = getCollectionName()  
 logger.info("Listening to changes in collection: $collectionName")  
  
 val aggregation = Aggregation.newAggregation(  
 Aggregation.match(Criteria.where("operationType").`in`("insert", "update", "delete", "replace")),  
 Aggregation.match(Criteria.where("ns.coll").`is`(collectionName))  
 )  
  
 val changeStreamOptions = ChangeStreamOptions  
 .builder()  
 .filter(aggregation)  
 .build()  
  
 mongoTemplate.changeStream(changeStreamOptions, entityClass)  
 .doOnNext **{** change **->** val operationType = change.*operationType*  
 logger.info("Change detected: $change (operation: $operationType)")  
  
 change.*body*?.*let* **{** handleChange(operationType, **it**) **}  
 }** .doOnError **{** error **->** logger.error("Error processing changes: ${error.message}")  
 **}** .subscribe()  
 }  
  
 abstract fun handleChange(action: OperationType?, record: T)  
  
 abstract fun getCollectionName(): String  
}

Например, конкретная реализация CourseSubscriptionChangeListener слушает изменения коллекции courseSubscriptionModel. При обнаружении изменения она получает полные данные по подписке через сервис CourseSubscriptionService и формирует полезную нагрузку, которую отправляет через WebSocketNotificationService в топик course-subscriptions-changed. Этот сервис, используя SimpMessagingTemplate Spring, умеет отправлять сообщения на разные каналы.

@Component  
class CourseSubscriptionChangeListener(  
 private val webSocketService: WebSocketNotificationService,  
 private val subscriptionService: CourseSubscriptionService  
) : AbstractMongoChangeStreamListener<CourseSubscriptionModel>() {  
  
 override val entityClass = CourseSubscriptionModel::class.*java* override fun handleChange(action: OperationType?, record: CourseSubscriptionModel) {  
 subscriptionService.getCourseBySubscription(record).subscribe **{** result **->** val payload = *mapOf*("action" *to* action, "record" *to* result)  
 webSocketService.notifyLoggedInUsers("course-subscriptions-changed", payload)  
 **}** }  
  
 override fun getCollectionName(): String {  
 return "courseSubscriptionModel"  
 }  
}

В конфигурации MongoDB обязательно должен быть включён режим Replica Set или rs0, без чего Change Streams работать не будут [21].

На клиентской стороне реализован класс CachedCollection<T>, который инкапсулирует логику локального кеширования данных с помощью библиотеки LokiJS, которая в свою очередь хранит данные в IndexedDB браузера через адаптер LokiIndexedAdapter. При инициализации коллекция пытается загрузить данные из кеша, а если они отсутствуют или пусты, делает запрос к серверу по REST-роуту с названием типа коллекции, например, course-subscriptions.get, получая полные актуальные данные.

export class CachedCollection<T extends { \_id: string }, U = T> extends EventEmitter {

public collection: Collection<T>;

protected name: string;

public eventName: string;

protected updatedAt: Date = new Date(0);

private callbacks: Set<{ cb: CachedCollectionCallback<T>; regex?: string }> = new Set();

constructor(name: string) {

super();

this.name = name;

this.eventName = `${name}-changed`

this.collection = Application.addCollection<T>(name, { unique: ["\_id"] });

}

pushCallback(callback: CachedCollectionCallback<T>, regex?: string): void {

this.callbacks.add({ cb: callback, regex });

}

popCallback(callback: CachedCollectionCallback<T>): void {

for (const item of this.callbacks) {

if (item.cb === callback) {

this.callbacks.delete(item);

break;

}

}

}

processRecords(record: T[], usage: CallbackUsage) {

this.callbacks.forEach(({ cb, regex }) => {

cb(record, usage, regex);

});

this.emit(this.eventName, record);

}

async listen() {

await this.init();

}

async init() {

const cachedData = this.loadFromCache();

if (!cachedData || cachedData.length === 0) {

await this.loadFromServer();

}

return this.setupListener();

}

...

protected async handleRecordEvent(action: 'removed' | 'changed', record: any) {

const newRecord = this.handleReceived(record, action);

if (!this.hasId(newRecord)) {

console.warn("Received record without \_id. Skipping...", record);

return;

}

const { \_id } = newRecord;

if (action === 'removed') {

this.collection.findAndRemove({ \_id: { $eq: \_id } });

console.log(`Record removed: ${\_id}`);

} else {

const recordToUpdate = this.collection.findOne({ \_id: { $eq: \_id } });

if (recordToUpdate) {

const updatedRecord = { ...recordToUpdate, ...newRecord, \_id: recordToUpdate.\_id };

this.collection.update(updatedRecord);

console.log(`Record updated: ${\_id}`);

} else {

console.log(`Insert new record with id: ${\_id}`);

this.collection.insert(newRecord);

}

}

this.processRecords([record], CallbackUsage.ONE);

}

async setupListener() {

if (!WebSocketClient.isConnected) {

console.warn("WebSocket is not connected. Waiting for connection...");

try {

await WebSocketClient.waitForConnection();

console.log("WebSocket connected, proceeding with subscription...");

} catch (error) {

console.error("WebSocket connection failed or timed out", error);

return;

}

}

const channel = `/topic/${this.eventName}`;

WebSocketClient.subscribeToChannel(channel, async (message) => {

try {

const { action, record } = JSON.parse(message.body);

console.log("Received WebSocket message:", action, record);

await this.handleRecordEvent(action, record);

} catch (error) {

console.error("Error processing WebSocket message:", error);

}

});

console.log(`Subscribed to WebSocket channel: ${channel}`);

}

...

async loadFromServer() {

try {

const response = await userApiClient.get<T[]>(`${this.name}.get`);

const startTime = new Date();

const lastTime = this.updatedAt;

response.forEach((record) => {

const newRecord = this.handleLoadFromServer(record);

if (!this.hasId(newRecord)) {

return;

}

const { \_id } = newRecord;

this.collection.findAndRemove({ \_id: { $eq: \_id } });

this.collection.insert(newRecord);

if (this.hasUpdatedAt(newRecord) && newRecord.\_updatedAt! > this.updatedAt) {

this.updatedAt = newRecord.\_updatedAt!;

}

});

this.processRecords(response, CallbackUsage.MANY);

this.updatedAt = this.updatedAt.getTime() === lastTime.getTime() ? startTime : this.updatedAt;

} catch (error) {

console.error("Ошибка при загрузке данных с сервера:", error);

}

}

...

}

После начальной загрузки коллекция подписывается на WebSocket-канал с именем, соответствующим шаблону ${name}-changed. Для подключения используется отдельный WebSocket клиент, реализованный на библиотеке STOMP.js, который управляет жизненным циклом соединения, подписками и обработкой сообщений. STOMP.js управляет жизненным циклом соединения, подписками и обработкой сообщений, упрощая двунаправленный обмен данными и обеспечивая надежное восстановление соединения при необходимости [22]. При получении сообщения с изменением коллекции класс CachedCollection обновляет или удаляет записи в локальной базе LokiJS, что автоматически отражается в пользовательском интерфейсе. Все подписчики на коллекцию получают уведомления через встроенную систему событий EventEmitter.

class WebSocketClient {

private client: Client;

public isConnected: boolean = false;

private connectPromise: Promise<void>;

private resolveConnect: () => void = () => { };

constructor(brokerURL: string) {

this.client = new Client({

brokerURL,

onConnect: this.onConnect,

onDisconnect: this.onDisconnect,

onStompError: this.onStompError,

});

this.connectPromise = new Promise((resolve) => {

this.resolveConnect = resolve;

});

this.client.activate();

}

private onConnect = () => {

this.isConnected = true;

console.log("WebSocket connected");

this.resolveConnect();

};

private onDisconnect = () => {

this.isConnected = false;

console.log("WebSocket disconnected");

};

private onStompError = (frame: any) => {

console.error("STOMP error: ", frame);

};

public activate() {

this.client.activate();

}

public deactivate() {

if (this.client.connected) {

this.client.deactivate();

}

}

public subscribeToChannel(channel: string, callback: (message: IMessage) => void) {

if (this.isConnected) {

this.client.subscribe(channel, (message: IMessage) => {

console.log("Received message on channel:", channel);

console.log(message.body);

callback(message);

});

} else {

console.error("WebSocket is not connected. Unable to subscribe.");

}

}

public unsubscribeFromChannel(channel: string) {

if (this.isConnected) {

this.client.unsubscribe(channel);

console.log(`Unsubscribed from channel: ${channel}`);

} else {

console.error("WebSocket is not connected. Unable to unsubscribe.");

}

}

public getConnectionStatus() {

return this.isConnected;

}

public waitForConnection() {

return this.connectPromise;

}

}

Таким образом, архитектура работает так: когда данные в MongoDB изменяются, слушатель AbstractMongoChangeStreamListener перехватывает событие и через WebSocket оповещает клиента. Клиентская часть в свою очередь быстро обновляет локальный кеш, обеспечивая реактивное и максимально оперативное обновление интерфейса без избыточных запросов. Хранение данных в IndexedDB обеспечивает их долговременную доступность и быструю загрузку при повторных открытиях приложения.

Кроме того, в проекте используется несколько видов WebSocket-каналов и событий: публичные, доступные всем слушателям; защищённые, предназначенные только для авторизованных пользователей; а также персональные, адресованные конкретным пользователям. Такая организация позволяет реализовывать сложные сценарии уведомлений и синхронизации с учётом требований безопасности.

Переходя к вопросу хранения мультимедийных данных, стоит отметить, что система загрузки и хранения файлов построена на базе GridFS — встроенного механизма MongoDB, предназначенного для работы с файлами, размер которых превышает лимит BSON-документа [23]. Это решение отлично подходит для надёжного и удобного хранения внутренних пользовательских материалов, включая документы, изображения и презентации, и глубоко интегрировано с общей архитектурой платформы.

Вся логика работы с файлами инкапсулирована в сервисе UploadService. При загрузке файла через эндпоинт /uploads.set происходит несколько шагов. Файл, представленный в виде объекта MultipartFile, передаётся в GridFS через метод store класса GridFsTemplate. Этот вызов сохраняет файл и возвращает его идентификатор, который преобразуется в строку и используется далее как ключ. Параллельно с этим создаётся вспомогательная сущность UploadModel, содержащая метаинформацию о файле: имя, размер, MIME-тип, дата загрузки и идентификатор пользователя, осуществившего загрузку. Эта информация сохраняется в отдельной MongoDB-коллекции и может использоваться для последующего аудита, отображения в пользовательском интерфейсе или фильтрации.

fun getFile(fileId: String): Resource {  
 val gridFSFile: GridFSFile = gridFsTemplate.findOne(  
 org.springframework.data.mongodb.core.query.Query.query(  
 Criteria.where("\_id").`is`(ObjectId(fileId))  
 )  
 )  
 ?: throw FileNotFoundException("File not found with id: $fileId")  
  
 return gridFsTemplate.getResource(gridFSFile)  
}

Выгрузка файла осуществляется через эндпоинт /uploads.get/{fileId}. При обращении к нему происходит извлечение GridFSFile по его идентификатору, далее — получение самого файла в виде ресурса и определение его типа содержимого. Если файл не найден, возвращается статус 404. При успешной загрузке файл передаётся в потоковом виде с правильным MIME-типом и заголовками, позволяющими браузеру отобразить или скачать файл.

Текущая архитектура позволяет в будущем отказаться от монолитного хранения всех файлов в MongoDB и перейти на использование облачных провайдеров. Например, можно интегрировать Amazon S3, Google Cloud Storage или Yandex Object Storage. Это может быть реализовано через абстракцию в UploadService, где логика сохранения и извлечения будет делегироваться соответствующему провайдеру. Таким образом, GridFS может использоваться в качестве локального или fallback хранилища, в то время как основное хранение файлов будет происходить в распределённой облачной среде. Такая архитектура обеспечит масштабируемость, лучшую отказоустойчивость и потенциальное снижение нагрузки на MongoDB.

Важно отметить, что при переходе на облачное хранение потребуется сохранить обратную совместимость с уже загруженными файлами. Для этого может быть реализована проверка источника хранения в UploadModel и соответствующая маршрутизация запроса — либо к GridFS, либо к облачному API.

Прогресс загрузки файла показывается на клиенте, поскольку это особенно важно для крупных файлов — например, видео — когда пользователю необходимо понимать, насколько продвинулась операция и не произошёл ли сбой. GridFS и стандартный механизм загрузки через MultipartFile в Spring не предоставляют встроенного поточного API для передачи состояния загрузки в процессе выполнения. Вместо этого используется XMLHttpRequest с подпиской на событие upload.onprogress.

Фронтенд-реализация выглядит следующим образом: при вызове функции uploadVideo, создаётся экземпляр XMLHttpRequest, который настраивается на отправку POST-запроса на эндпоинт uploads.set. Файл упаковывается в FormData, после чего инициализируется загрузка. Событие xhr.upload.onprogress срабатывает каждый раз, когда браузер получает обновление о текущем объёме переданных данных. Оно содержит два значения: loaded — сколько байт уже загружено, и total — общий размер файла. На основе этих данных рассчитывается процент выполнения: (loaded / total) \* 100. Далее значение округляется и передаётся в UI.

Если происходит ошибка – например, потеря соединения – в обработчике xhr.onerror прогресс сбрасывается, и управление передаётся в блок обработки ошибки. При успешном завершении запроса xhr.onload проверяет статус ответа, и, если он успешен, результат распарсивается и возвращается вызывающей стороне [24].

Благодаря такому подходу прогресс загрузки становится доступным в режиме реального времени и может отображаться пользователю через прогресс-бар или текстовый индикатор. Это позволяет создать отзывчивый и современный пользовательский интерфейс, не перегружая сервер дополнительными запросами на получение статуса.

  const uploadVideo = async (file: File): Promise<UploadedVideo> => {

    return new Promise((resolve, reject) => {

        const xhr = new XMLHttpRequest();

        xhr.open('POST', `${uploadsApiPrefix}.set`, true);

        xhr.withCredentials = true;

        const formData = new FormData();

        formData.append('file', file);

        xhr.upload.onprogress = (event) => {

            if (!event.lengthComputable) {

                return;

            }

            const progress = (event.loaded / event.total) \* 100;

            if (progress === 100) {

                return;

            }

            setProgress(Math.round(progress) || 0);

        };

        xhr.onerror = (error: any) => {

            setProgress(0);

            reject(new Error(error.message));

        };

        xhr.onload = async () => {

            if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {

                const result = JSON.parse(xhr.responseText);

                resolve(result);

            } else {

                reject(new Error(xhr.responseText));

            }

        };

        xhr.send(formData);

    });

};

Переходя к последнему ключевому блоку общей майндкарты — усовершенствованию взаимодействия с образовательным контентом, отметим, что он делится на две основные составляющие: аудиторию и механизмы. В аудиторию входят создатели и потребители контента — создателям не нужно вручную писать конспекты и тесты, а потребителям эти материалы помогают лучше усваивать информацию и проверять свои знания. Подробнее структура и связи этого блока показаны на рисунке 2.32 с конкретизацией узла общей майндкарты «Усовершенствование взаимодействия с образовательным контентом».

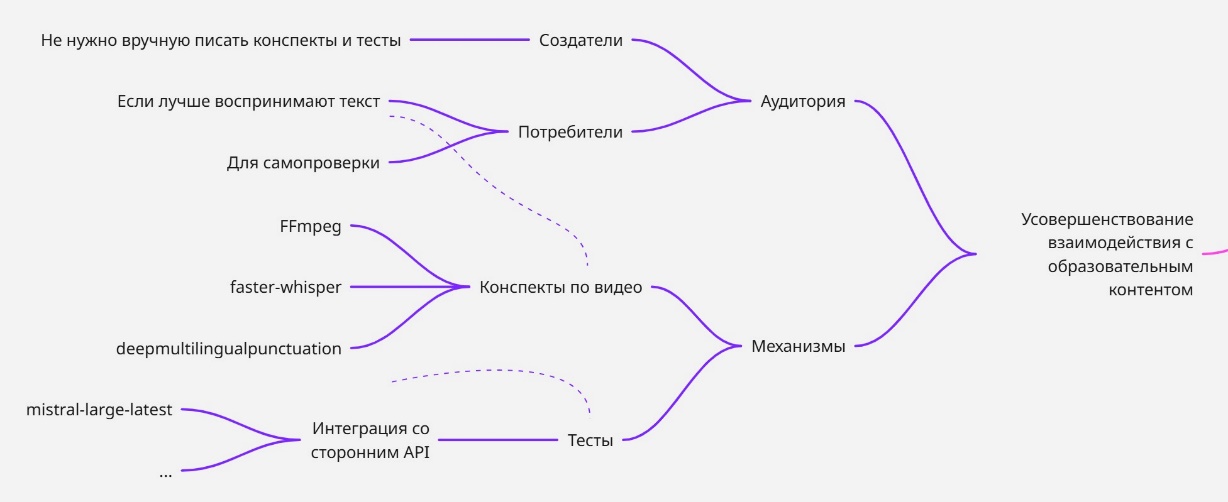


Рисунок 2.32 – Конкретизация узла «Усовершенствование взаимодействия с образовательным контентом» общей майндкарты

В частности, сервис для взаимодействия с нейросетями реализован на базе FastAPI с модульной архитектурой. Он использует MongoDB и GridFS для хранения видеоматериалов и интегрирует несколько моделей машинного обучения для распознавания речи, расстановки пунктуации и стилистической постобработки текста. Видеофайлы загружаются пользователем через пользовательский бэкенд, сохраняются в MongoDB с помощью GridFS и получают уникальный fileId, используемый для доступа к ним. FastAPI-приложение, отвечающее за транскрипцию, обращается к базе через асинхронный клиент motor и AsyncIOMotorGridFSBucket, извлекая файлы напрямую из GridFS.

При запросе на транскрипцию сначала видеофайл по fileId временно сохраняется на диск в папке uploads. Далее с помощью FFmpeg — мощного инструмента для обработки мультимедиа, широко используемого для конвертации аудио- и видеоформатов [25] — из видео извлекается звуковая дорожка, которая конвертируется в формат WAV с параметрами, оптимальными для речевых моделей: частота дискретизации 16 кГц и монофонический канал.

Поскольку крупные файлы невозможно эффективно обрабатывать сразу целиком, аудио разбивается на фрагменты продолжительностью 30 секунд. Для этого используется библиотека pydub, удобный Python-инструмент для работы с аудио, позволяющий легко манипулировать звуковыми файлами и сегментировать их [26]. Каждый фрагмент сохраняется как отдельный WAV-файл.

Транскрипция каждого аудиофрагмента выполняется параллельно с использованием ThreadPoolExecutor, который упрощает организацию многопоточной обработки задач в Python и позволяет эффективно использовать ресурсы процессора [27]. В качестве модели распознавания речи используется faster-whisper — высокопроизводительный CPU-ориентированный форк модели Whisper от OpenAI. Whisper — это универсальная нейросетевая модель, обученная на многомасштабном корпусе аудио и текста, способная распознавать речь с высокой точностью даже в неблагоприятных акустических условиях [28]. Faster-версия сохраняет точность, но оптимизирована для ускоренной инференции на центральных процессорах, что критично для масштабируемых систем без использования GPU.

Результаты транскрипции из каждого чанка объединяются в один длинный текст. На этом этапе текст ещё не содержит корректной пунктуации и разделения на предложения.

Для приведения текста к читаемому виду применяется модель deepmultilingualpunctuation, которая восстанавливает знаки препинания и начальные заглавные буквы, используя современные методы многоязыковой обработки текста [29]. После этого производится дополнительная стилистическая обработка, в том числе ручная капитализация первых букв предложений и форматирование с переносами строк.

В конце обработки временные файлы удаляются, и пользователю возвращается готовый текстовый конспект с восстановленной пунктуацией, пригодный для индексации, отображения в интерфейсе и поиска по контенту. Этот механизм является важной частью узла майндкарты «Усовершенствование взаимодействия с образовательным контентом», подробно представленного на соответствующем рисунке.

В продолжение работы с образовательным контентом, в данном модуле также реализован сервис интеграций с внешними API, предназначенный для подключения нейросетевых моделей, таких как Mistral AI, через настраиваемую систему исполнения скриптов. Основная задача этого сервиса — обеспечить универсальный интерфейс для подключения различных провайдеров генеративного ИИ и выполнить скрипт, описывающий порядок работы с выбранным API, передав в него параметры и получив результат.

Mistral AI предоставляет RESTful API, позволяющее обращаться к современным языковым моделям, включая mistral-small, mistral-medium и mistral-large-latest. Для подключения используется стандартная схема авторизации через API-ключ, который передаётся в заголовке HTTP-запроса как Authorization: Bearer <token>. Запросы направляются на конечную точку https://api.mistral.ai/v1/chat/completions. В теле запроса указываются параметры генерации текста: список сообщений, передаваемых в поле messages, идентификатор модели в поле model, а также параметр temperature, определяющий степень креативности и вариативности ответа [30].

Каждая интеграция хранится в MongoDB и содержит уникальный идентификатор, название, API-ключ, исходный Python-скрипт, а также информацию о пользователе, создавшем интеграцию, и дату создания. Скрипт, связанный с интеграцией, представляет собой шаблонный код, предназначенный для выполнения в песочнице и обращения к стороннему API, например, Mistral. Он получает ключ доступа и данные в формате JSON, выполняет вызов к модели mistral-large-latest и форматирует ответ в соответствии с заданным шаблоном — в данном случае, генерация вопросов и ответов на основе переданного текста.

Сервис написан на FastAPI и предоставляет ряд REST-эндпоинтов, включая создание, обновление, удаление и получение интеграций, а также специальный маршрут, через который запускается исполнение соответствующего скрипта. При вызове этого маршрута клиент передаёт integration\_id и контекст в виде JSON-структуры – например, количество нужных вопросов и сам текст, по которому нужно сгенерировать тест, после чего происходит следующее: из базы извлекается нужная интеграция, валидируется наличие ключа и скрипта, затем скрипт выполняется в отдельном процессе с передачей аргументов через командную строку. В качестве механизма исполнения используется стандартный subprocess.Popen, обеспечивающий изоляцию выполнения и контроль за stdout и stderr. В случае успешного завершения исполнения скрипта результат возвращается в HTTP-ответе; если в процессе возникает ошибка, она логируется и клиент получает соответствующее исключение с описанием проблемы.

Следует понимать, что текущая реализация является скорее прототипом, чем полноценным продакшн-решением, поскольку она не обеспечивает полноценной защиты от произвольного кода, который может быть выполнен в пользовательском скрипте. Для безопасного выполнения недоверенного кода широко применяется концепция песочницы – механизма изоляции, который создает контролируемую среду, ограничивающую доступ исполняемого кода к системным ресурсам, таким как файловая система, сеть и память. Это позволяет минимизировать риск выполнения вредоносных операций и защищает основную систему от потенциального ущерба [31].

В контексте Python существует несколько решений, направленных на изоляцию среды выполнения. Например, библиотека RestrictedPython предоставляет ограниченное подмножество языка и позволяет запускать скрипты в контролируемом окружении, однако она не гарантирует полного предотвращения всех угроз и не является полноценной песочницей [32]. Более серьезный уровень изоляции достигается с помощью Pyodide – реализации интерпретатора CPython на базе WebAssembly, которая запускает код в изолированной среде браузера или Node.js, что обеспечивает высокий уровень безопасности за счет ограничения доступа к системным ресурсам [33]. Также следует отметить sandbox-режим интерпретатора PyPy, который запускает недоверенный код в отдельном процессе с ограниченными правами, что обеспечивает изоляцию на уровне операционной системы.

Таким образом, для усиления безопасности и контроля над исполнением сторонних скриптов в будущем проекте целесообразно рассмотреть использование подобных специализированных инструментов, обеспечивающих реальную изоляцию среды выполнения.

Таким образом, в ходе разработки мы подробно рассмотрели ключевые компоненты и архитектурные решения проекта, строго следуя структуре общей майндкарты. Были раскрыты основные аспекты динамической конфигурации, источников данных о пользователях, управления доступом и ролями, а также механизмы хранения медиафайлов. Особое внимание уделено усовершенствованию взаимодействия с образовательным контентом, что значительно повышает удобство и качество пользовательского опыта. Всё это органично вписано в дизайн-проект и техническое задание.

# 

# 2.3 Тестирование и анализ корректности работы системы

В ходе проектирования, рассмотренного в разделе 2.1, были разработаны диаграммы вариантов использования, отражающие ключевые действия, доступные различным категориям пользователей в системе. Эти диаграммы позволили формализовать функциональные ожидания и сформировать основу для последующей реализации архитектурных решений. Были выделены два основных контекста взаимодействия — пользовательский скоуп и скоуп курсов, а также определены роли: пользователь, редактор, администратор, владелец курса и модератор. Эти роли определяют уровень доступа и набор допустимых операций в рамках системы.

Проектирование вариантов использования позволило определить важнейшие пользовательские действия: вход в систему, редактирование профиля, просмотр курсов, отслеживание прогресса, взаимодействие с образовательным контентом, создание курсов, управление ролями и настройками, а также выполнение административных задач.

Разработка велась в строгом соответствии с данными диаграммами. В результате реализованный функционал полностью покрывает спроектированные сценарии. Вместе с тем система изначально была спроектирована как расширяемая: в будущем могут быть добавлены новые роли или изменены права существующих, что приведёт к изменению и расширению текущих вариантов использования. Таким образом, представленные диаграммы выступают в роли исходного проектного ориентира, а не жёсткого ограничения.

В рамках ручного тестирования для иллюстрации соответствия между проектной моделью и реализованным функционалом приведён контрольный пример работы системы.

В рамках тестовой среды LDAP-сервер развернут в Docker-контейнере для упрощения настройки и изоляции инфраструктуры. Для управления данными используется веб-интерфейс phpLDAPadmin, позволяющий просматривать и редактировать записи в каталоге. На рисунке 2.33 приведены скриншоты из административной панели, где отображён тестовый пользователь с именем testuser. Также прилагается результат выполнения команды ldapsearch, демонстрирующий, что данный пользователь состоит в группе testgroup, что дает атрибут memberOf.

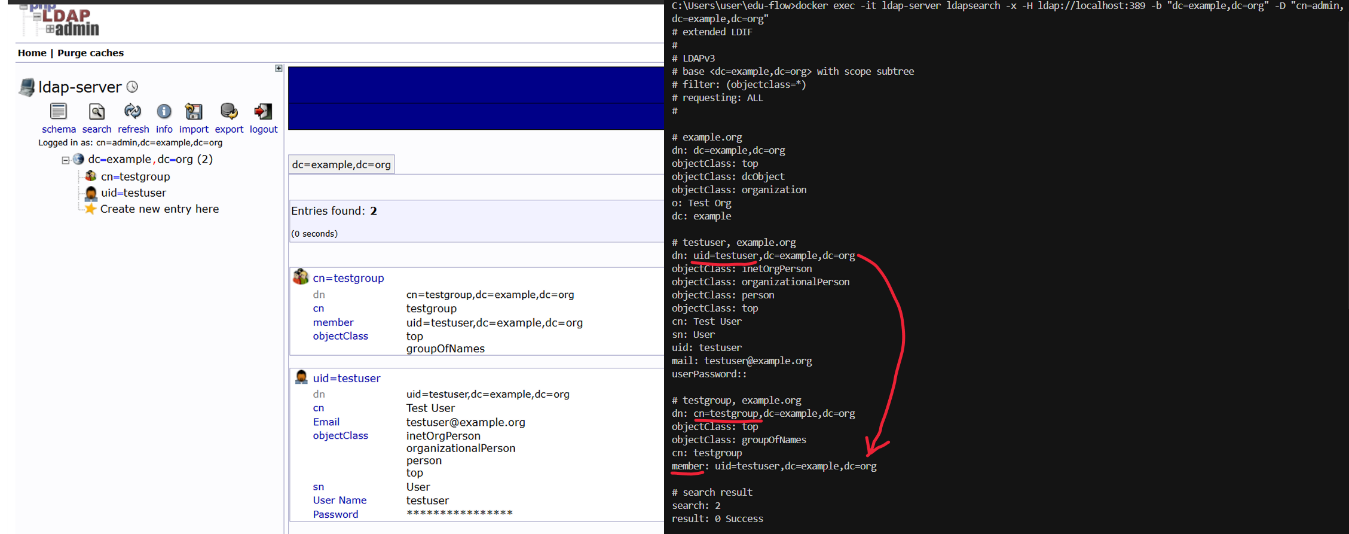


Рисунок 2.33 – Тестовый пользователь LDAP и его принадлежность к группе testgroup в phpLDAPadmin и через ldapsearch

На рисунке 2.34 представлена административная панель, в которой заданы настройки по умолчанию для новых пользователей. Изображённая конфигурация подтверждает, что при первичной инициализации аккаунта устанавливаются русский язык интерфейса и тёмная тема.

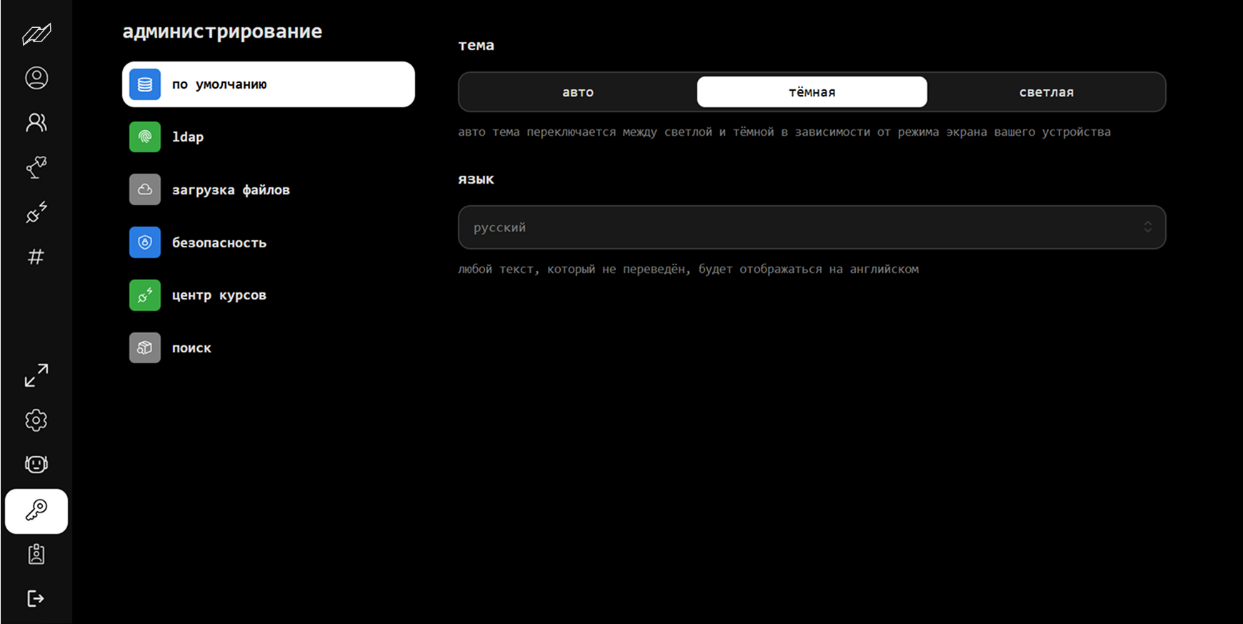


Рисунок 2.34 – Настройки пользователя по умолчанию в административной панели

На рисунке 2.35 отображается настройка сопоставления LDAP-групп с доступными курсами: в частности, видно, что пользователям из группы testgroup автоматически предоставляется доступ к курсам с тегом презентации и к курсу «Как пользоваться Microsoft Outlook». Эти конфигурации подтверждают корректность функционирования механизма связывания внешней аутентификации с логикой персонализации внутри системы.



Рисунок 2.35 – Сопоставление LDAP-группы testgroup с доступными курсами

На рисунке 2.36 представлена вкладка каталога курсов, доступная пользователю после авторизации. Интерфейс вкладки предоставляет возможности для поиска по названию курса, а также фильтрации по различным критериям, включая теги, предполагаемую продолжительность изучения и категорию отображаемых курсов, таких как доступные, избранные и те, подписчиком которых пользователь является. На момент отображения без применения дополнительных фильтров и поисковых запросов в каталоге отображается три курса: «Создание презентаций в Figma», «Искусство деловой презентации» и «Как пользоваться Microsoft Outlook». Первые два курса снабжены тегами, соответствующими настройкам, заданным в административной панели в привязке к группе testgroup, а третий курс полностью совпадает с названием, указанным в сопоставлении LDAP-группы и назначенных курсов.

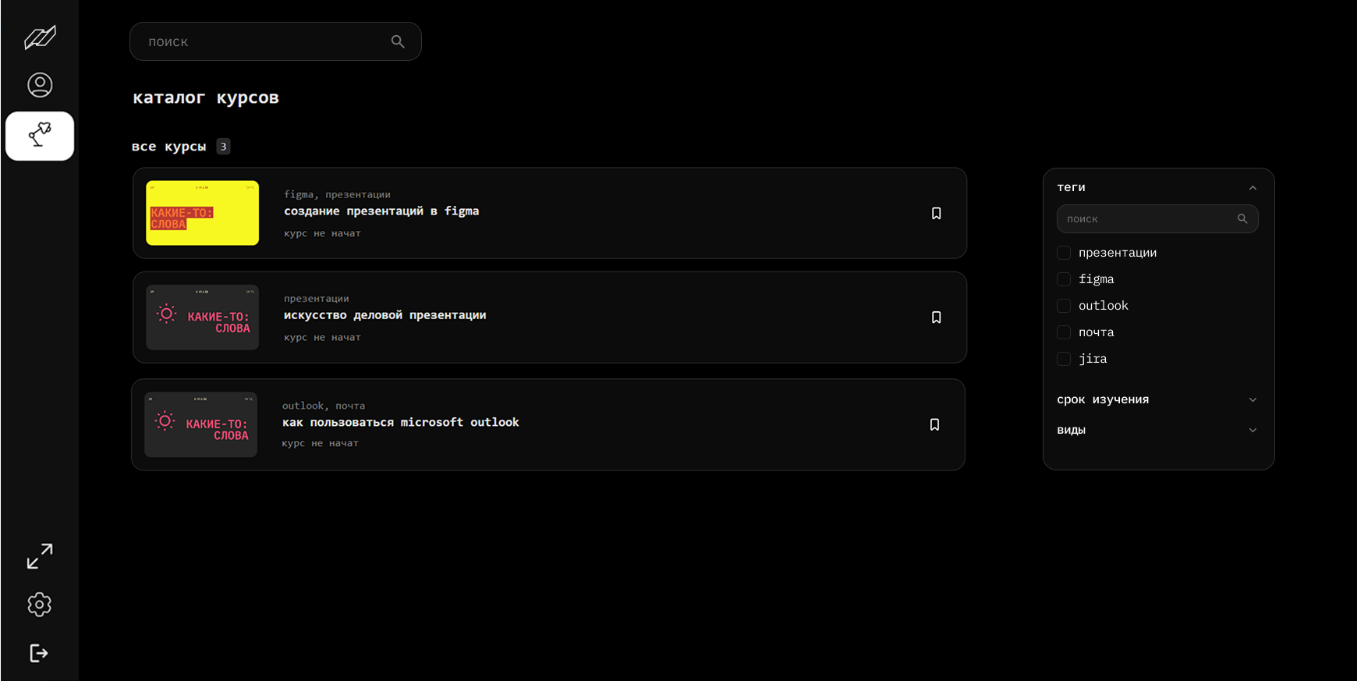


Рисунок 2.36 – Вкладка каталога курсов с поиском и фильтрацией

На рисунке 2.37 представлена страница курса в начальном состоянии до начала его прохождения. В верхней части интерфейса отображается краткое описание курса с возможностью перехода на вкладку «О курсе» для изучения расширенного описания и просмотра тегов, определяющих тематику материала. Прямо со страницы курса пользователь может добавить его в избранное. Ниже представлено содержание курса, включающее секции с лекциями. В рамках демонстрации отображаются две секции: «Первый раздел» и «Второй раздел». В составе первой секции размещена лекция «Работа со смыслами». При переходе к ней начинается процесс прохождения курса, в ходе которого пользователь сможет просматривать видео, читать конспекты и проходить тесты.

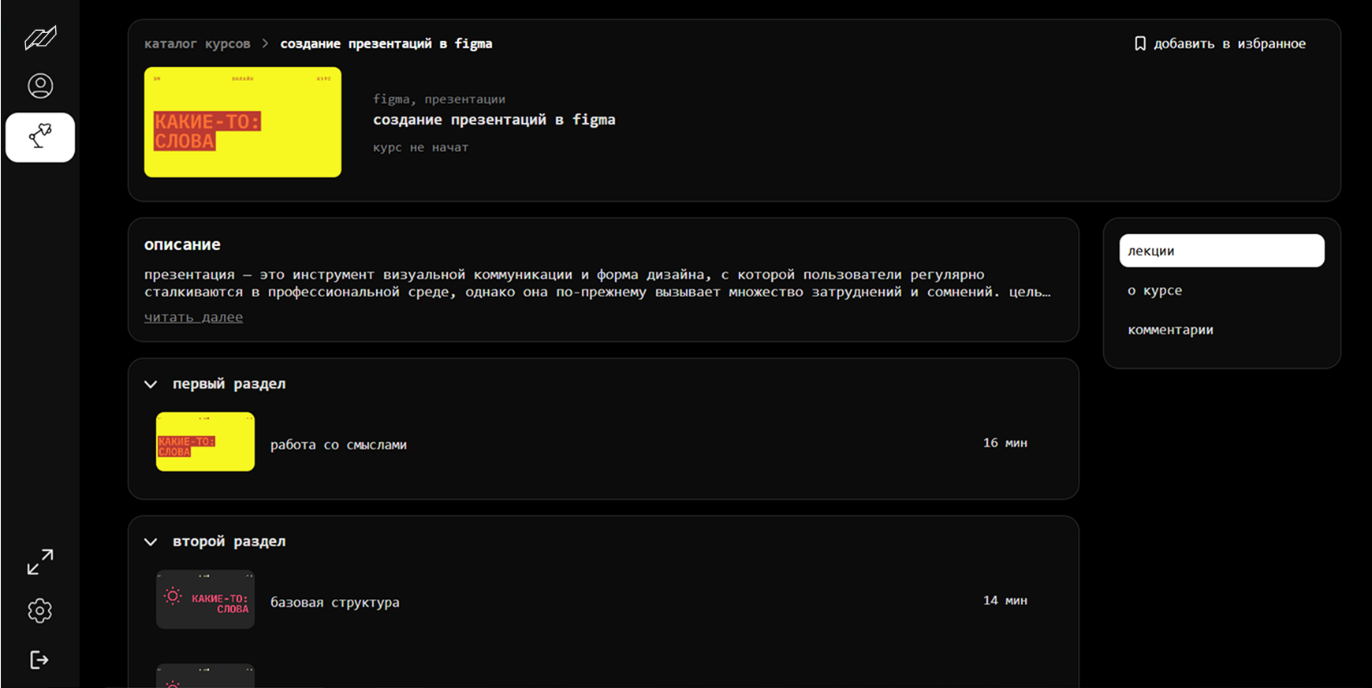


Рисунок 2.37 – Страница курса в начальном состоянии с описанием и содержанием

На рисунке 2.38 представлена страница прохождения лекции «Базовая структура», относящейся ко второму разделу курса. В данный момент пользователем открыт видеоматериал, и процесс просмотра уже начат, о чём свидетельствует отображаемый прогресс в 1%. Интерфейс предоставляет возможность перейти к следующему учебному элементу — сгенерированному текстовому конспекту, а также ознакомиться с полным текстовым сопровождением текущей лекции.

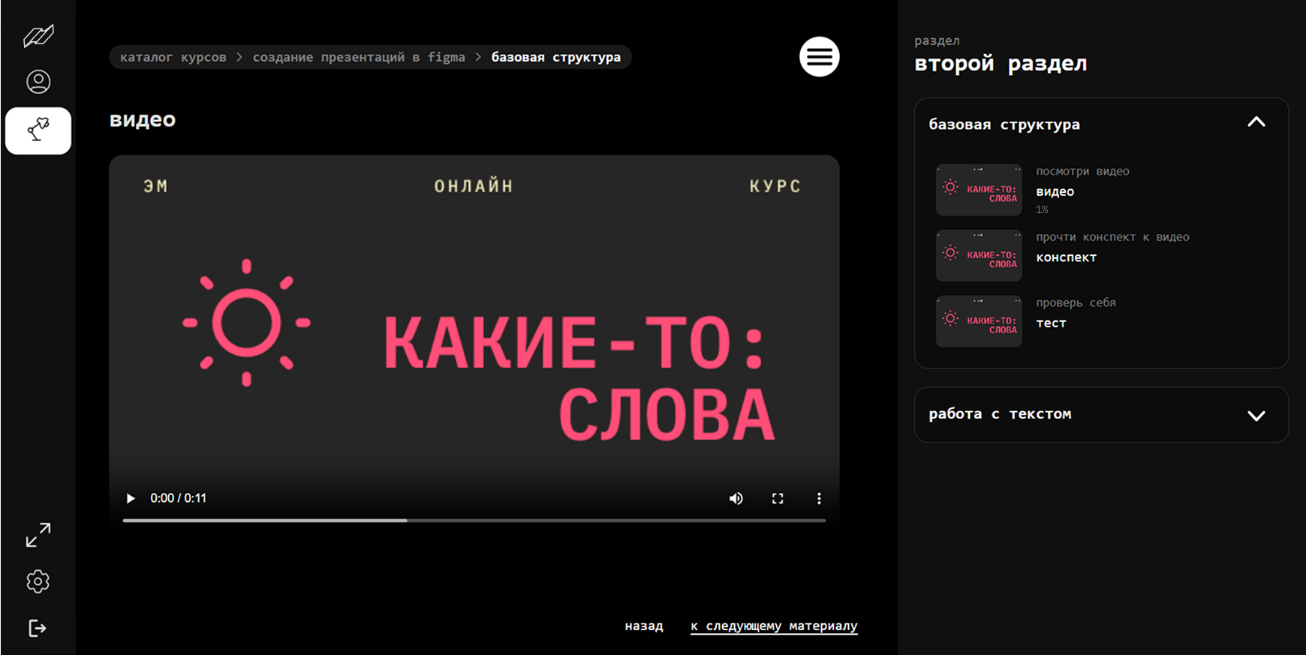


Рисунок 2.38 – Страница прохождения лекции с видео, текстом и прогрессом обучения

На рисунке 2.39 представлен первый вопрос теста по лекции «Базовая структура», расположенной во втором разделе курса. На изображении показан выбранный пользователем правильный вариант ответа.



Рисунок 2.39 – Первый вопрос теста по лекции «Базовая структура» с выбранным правильным ответом

На рисунке 2.40 представлена административная панель управления пользователями. В рамках демонстрации отображён процесс назначения роли редактора пользователю с именем testuser. В интерфейсе выбран соответствующий пользователь, после чего из выпадающего списка ролей выбирается значение «Редактор». Завершением процедуры служит сохранение изменений, после чего пользователь получает дополнительные права в системе, соответствующие назначенной роли.

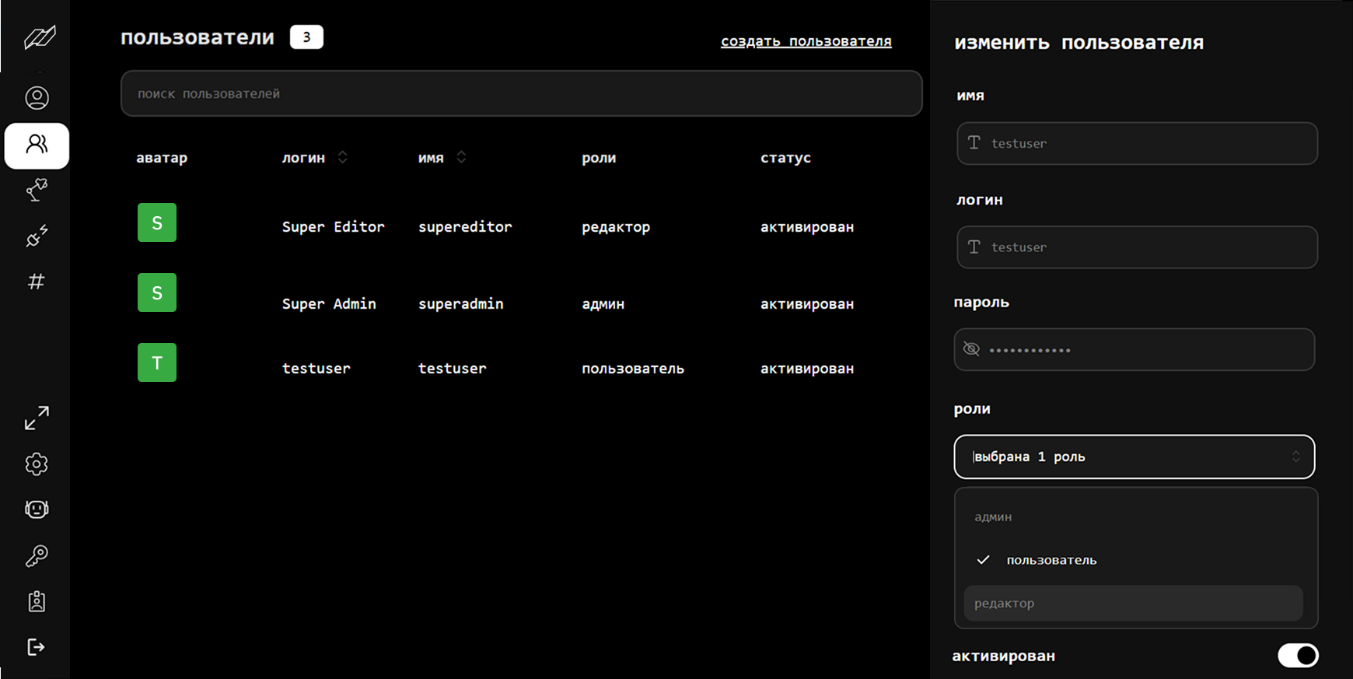


Рисунок 2.40 – Назначение роли редактора пользователю testuser через административную панель

После назначения роли редактора пользователь testuser получает возможность создавать собственные курсы. Он может добавлять в них секции и лекции, управлять доступом, назначать владельцев и модераторов, приглашать других пользователей. При создании лекций редактору доступны функции автоматической генерации конспектов и тестов, их последующего редактирования, управления видимостью отдельных учебных блоков, а также публикации курса.

На рисунке 2.41 представлена последовательность шагов, иллюстрирующая процесс создания лекции редактором курса. В начальном окне отображается страница управления курсом, в которой доступны функции добавления новых лекций и разделов, управления редакторами, добавления подписчиков, а также просмотра статистики. В верхней части интерфейса отображается статус курса — «не опубликован», что означает его доступность исключительно для редакторов. Указанная версия курса — 0.1, то есть исходная, самая первая. Добавленный раздел в текущей конфигурации также помечен как невидимый.

Далее демонстрируется этап добавления лекции, включающий окно загрузки видеоматериала, где выбрано видео с названием top\_level.mp4, и установлены флаги автоматической генерации конспекта и теста. На следующем этапе показан результат генерации конспекта — пользователь имеет возможность просмотреть его и внести правки, например, отредактировать текст в формате Markdown – что и было сделано, жирным и курсивом выделили текст в начале первого абзаца.

Завершающим элементом является редактор теста: из пяти автоматически сгенерированных вопросов пользователь сохранил два, удалив остальные. Таким образом, редактор получает полный контроль над содержанием лекции и может настраивать её структуру в соответствии с поставленными образовательными задачами.

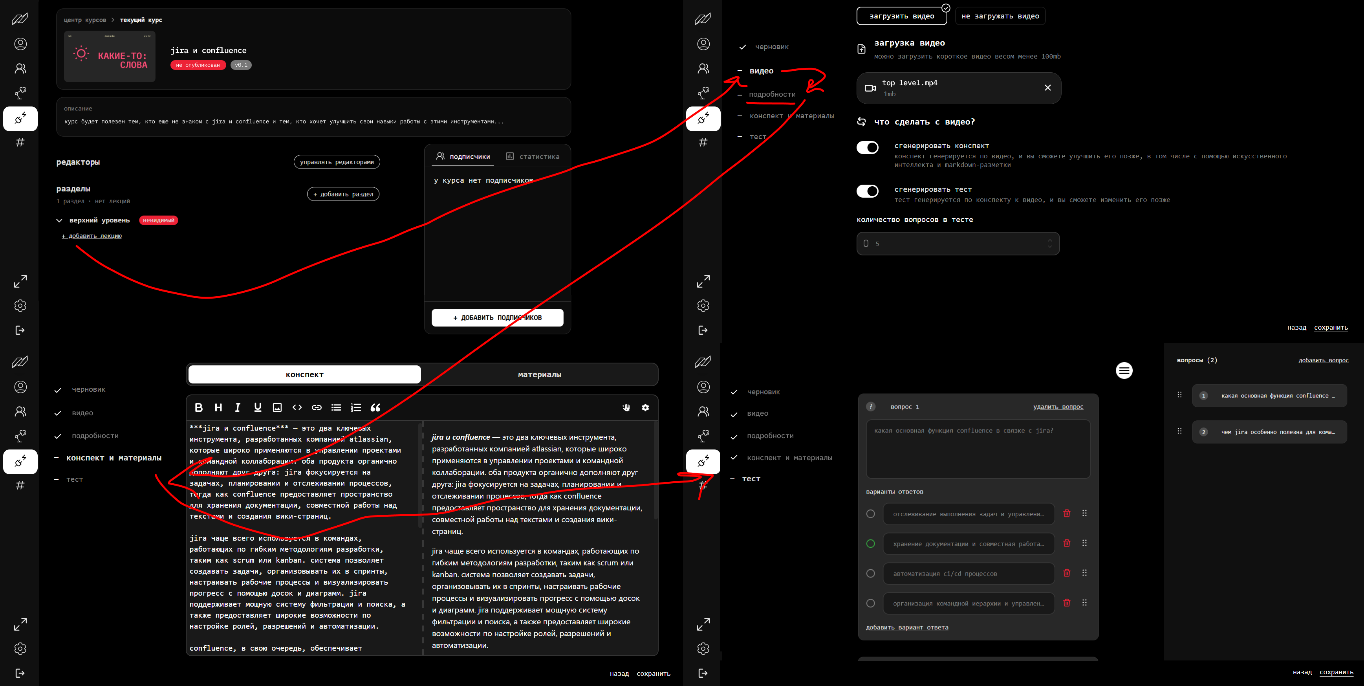


Рисунок 2.41 – Этапы создания лекции редактором: от добавления до редактирования учебных блоков

Таким образом, контрольный пример демонстрирует, что система в полной мере реализует ключевые сценарии, отражённые в диаграммах вариантов использования. Хотя в рамках данного документа представлен лишь один пример, он убедительно иллюстрирует соответствие реализованного функционала изначально спроектированной архитектуре.

# 2.4 Организация и управление рабочим процессом

В ходе работы над проектом применялась канбан-доска, созданная с использованием инструмента Trello. Канбан — это методология визуального управления рабочими процессами, позволяющая отслеживать ход выполнения задач в реальном времени, выявлять узкие места и эффективно планировать ресурсы [34]. Она помогает визуализировать рабочий процесс, ограничивать количество задач в работе, управлять потоком задач и обеспечивать постоянное улучшение процессов. Trello был выбран благодаря своей гибкости и простоте использования. Он предоставляет удобные механизмы конфигурации доски с возможностью добавления сроков выполнения, меток приоритета, комментариев, вложений и прочей вспомогательной информации.

Для визуализации хода работы применялась концепция инициатив — укрупнённых единиц управления, каждая из которых охватывает значимый и логически завершённый этап проекта. Инициатива в данном случае соответствует целостному блоку проектной деятельности, как правило, выравненному по смыслу с одним из разделов пояснительной записки. Это позволило не только структурировать доску по содержательным элементам проекта, но и обеспечить соответствие между визуальным представлением работы и архитектурой конечного решения.

Следует отметить, что в рамках классических agile-подходов, таких как Scrum и Kanban, рабочий процесс организуется через иерархию задач: от инициатив к эпику, от эпика к пользовательским историям или техническим задачам. Scrum ориентирован на итеративную командную работу с фиксированными спринтами и чётко определёнными ролями [34], тогда как Kanban более гибок и фокусируется на непрерывном управлении потоком задач [34]. Однако в данном проекте Trello-доска использовалась преимущественно для стратегического планирования, поэтому на ней отражались только инициативы верхнего уровня. Эпики и подзадачи реализовывались и документировались в рамках внутренней разработки, но на самой доске не фиксировались. Это обусловлено тем, что структура доски должна была оставаться компактной, обозримой и соответствующей этапам дипломной работы, не распыляя внимание на операционные детали.

На доске представлены инициативы, охватывающие изучение базы преддипломной практики и деятельности организации-заказчика, анализ существующих LMS-решений, выбор технологического стека, проектирование архитектуры программной системы, реализацию и интеграцию компонентов, разработку административного интерфейса и взаимодействие с внешними сервисами, тестирование системы, а также оценку надёжности и экономической эффективности внедрения.

Фрагмент канбан-доски, иллюстрирующий распределение инициатив, представлен на рисунке 2.42.



Рисунок 2.42 – Канбан-доска в Trello с инициативами проекта

Рабочий период охватывал временной интервал с 15 февраля по 1 июня, в течение которого было реализовано восемь двухнедельных спринтов. Несмотря на то, что классическая канбан-модель ориентирована на непрерывное выполнение задач без жёсткой итеративности, в данном случае применялся смешанный подход, включающий элементы scrum. Каждые две недели подводились промежуточные итоги, фиксировались достигнутые результаты, проводился обзор прогресса и формировался план работ на следующий период. Такой подход обеспечивал баланс между гибкостью и регулярностью контроля, способствуя эффективному управлению проектом в целом.

Таким образом, канбан-доска в рамках данного проекта выполняла роль инструмента высокого уровня планирования и контроля, обеспечивая прозрачность хода работы, отслеживание ключевых этапов и сохранение логической целостности между планом разработки и документационной структурой.

Итак, во второй главе были поэтапно рассмотрены ключевые этапы проектирования и реализации программной системы, а также проведена проверка её работоспособности на контрольном примере.

На первом этапе была выполнена проработка архитектуры разрабатываемой системы с использованием языка моделирования UML. В рамках проектирования были построены как структурные, так и поведенческие диаграммы, отражающие ключевые компоненты системы и их взаимодействие. Для документирования использовался инструмент PlantUML. Среди поведенческих диаграмм были выбраны варианты, наиболее релевантные целям проекта: диаграммы вариантов использования, активностей, состояний и последовательностей. В качестве структурных моделей применялись диаграммы классов и компонентов.

Далее, в соответствии с техническим заданием от организации, были выделены и реализованы ключевые этапы разработки. В первую очередь был подготовлен дизайн-проект, включающий формулирование концепции, подбор визуальных и лингвистических параметров интерфейса, разработку логотипа, цветовой палитры, шрифтов, типографики и интерфейсных компонентов, а также макетирование пользовательских страниц. На следующем этапе была составлена структурная карта приложения, охватывающая API-маршруты и клиентские представления.

Особое внимание было уделено интеграции источников данных о пользователях, включая корпоративный каталог LDAP и административную панель. В рамках демонстрации было организовано хранение медиафайлов с использованием GridFS в MongoDB. Далее была реализована динамическая конфигурация, основанная на настройках, прослушивании изменений в базе данных и механизме реактивного взаимодействия, включая использование WebSocket и клиентского кеша. Также была внедрена система разграничения доступа на основе ролей и прав. Завершающим этапом реализации стало усовершенствование взаимодействия с образовательным контентом с использованием генеративных нейросетей: автоматическое создание конспектов и тестов, интеграция внешних API.

Следующий раздел главы был посвящён контрольному примеру работы системы. На основе тестовых данных проведено поэтапное взаимодействие с разработанным решением. Сопоставление полученных результатов с заранее определёнными вариантами использования подтвердило, что все функциональные требования, изложенные в техническом задании и архитектурном проекте, были выполнены полностью.

Дополнительно, организация и управление рабочим процессом с использованием канбан-доски, описанные в заключительном разделе главы, обеспечили эффективный контроль и прозрачность выполнения задач в течение всего периода разработки. Применение данного инструмента позволило своевременно выявлять узкие места, корректировать планы и поддерживать системный подход к реализации проекта.

# 3 РАСЧЁТНАЯ ЧАСТЬ

# Оценка экономической эффективности внедрения

Экономическая эффективность внедрения внутренней обучающей платформы оценена в сравнении – через сопоставление совокупных затрат на создание и сопровождение собственного решения со стоимостью использования коробочных решений с аналогичной функциональностью. В качестве альтернативных вариантов были рассмотрены коммерческие LMS-системы iSpring Learn, Teachbase, Unicraft и Motivity, имеющие возможность коробочной установки и ориентированные на корпоративный сегмент с числом пользователей от одной тысячи.

Оценка основана на методике расчёта совокупной стоимости владения или Total Cost of Ownership, предложенной аналитической компанией Gartner в 1987 году и ставшей отраслевым стандартом в сфере информационных технологий [35]. Методика TCO учитывает как прямые, так и косвенные расходы, включая затраты на приобретение, внедрение, сопровождение, обучение персонала, обновления и другие составляющие, влияющие на стоимость владения системой в течение её жизненного цикла. Применение такого подхода позволяет получить целостное представление о финансовых последствиях внедрения собственной платформы по сравнению с коммерческими альтернативами.

Расчёты произведены на период в тридцать шесть месяцев. Для каждого варианта сопоставляется совокупная стоимость владения за весь указанный срок. Текущие тарифы решений представлены в таблице 3.1.

Таблица 3. – Сравнение стоимостей использования LMS-решений (1000 пользователей, 36 месяцев)

|  |  |  |  |
| --- | --- | --- | --- |
| Решение | Стоимость на одного пользователя в месяц, ₽ | Месячная стоимость, ₽ | За 36 месяцев, ₽ |
| iSpring | 103 | 103000 | 3708000 |
| Teachbase | 161 | 161000 | 5796000 |
| Unicraft | 100 | 100000 | 3600000 |
| Motivity | 196 | 196000 | 7056000 |

В отличие от этих решений, собственная разработка требует стартовых вложений на этапах проектирования и реализации. При этом расчётная стоимость одного месяца разработки составляет 100000 рублей. Эта сумма отражает фактическую стоимость услуг подрядчика и включает участие аналитика, дизайнера и разработчика, распределённых между несколькими проектами. Расходы на поддержку составляют 60000 рублей в месяц и начинаются с месяца внедрения. Поддержка осуществляется внутренней технической службой, ресурсы которой задействованы также в обслуживании других информационных систем организации. Стоимость хостинга учитывается косвенно, так как серверы используются в составе корпоративной инфраструктуры.

Суммарные затраты на разработку составили 375000 рублей. Поддержка в течение оставшихся тридцати двух месяцев обошлась в 1920000 рублей, а месяц внедрения — в 60000. Таким образом, общая стоимость проекта за весь период составила 2355000 рублей. Распределение затрат по месяцам представлено в таблице 3.2.

Таблица 3. – Затраты на разработку и поддержку собственного решения

|  |  |  |  |
| --- | --- | --- | --- |
| Месяц | Разработка, ₽ | Поддержка, ₽ | Итого, ₽ |
| Февраль | 75000 | 0 | 75000 |
| Март | 100000 | 0 | 100000 |
| Апрель | 100000 | 0 | 100000 |
| Май | 100000 | 0 | 100000 |
| Июнь | 0 | 60000 | 60000 |
| Июль–декабрь третьего года | 0 | 60000 ежемесячно | 1920000 |
| Итого | 375000 | 1980000 | 2355000 |

По данным Аткинсона и соавторов, период окупаемости является простой и широко применяемой моделью оценки возврата инвестиций, отражающей время, необходимое для возмещения первоначальных затрат за счёт экономии или доходов, генерируемых проектом [36].

Далее рассчитывается разница между затратами на готовые решения и затратами на собственную платформу. Экономия по каждому варианту рассчитывается как разность между затратами на коробочное решение за 36 месяцев и стоимостью собственной разработки и поддержки за тот же срок. Среднемесячная экономия после внедрения рассчитывается как эта разность, делённая на тридцать два месяца эксплуатации.

Для определения срока окупаемости применяется простая модель возврата вложений. Показатель Payback Period — срок окупаемости инвестиций — рассчитывается по формуле [37, 38]:

Где под суммарными инвестициями понимаются расходы на разработку и внедрение – в данном случае 375000 рублей, а среднемесячная экономия вычисляется на основе сопоставления с альтернативными решениями.

Таблица 3. – Экономия и срок окупаемости собственного решения по сравнению с конкурентами

|  |  |  |  |
| --- | --- | --- | --- |
| Решение | Экономия за 3 года, ₽ | Среднемесячная экономия, ₽ | Срок окупаемости, месяцев |
| iSpring | 1353000 | 42952 | 8,73 |
| Teachbase | 3441000 | 109238 | 3,43 |
| Unicraft | 1245000 | 39524 | 9,49 |
| Motivity | 4701000 | 149238 | 2,51 |

Из таблицы 3.3 видно, что несмотря на сравнительно невысокие инвестиции, разработка собственной платформы позволяет достичь окупаемости уже в первые месяцы эксплуатации. Наиболее быстрая окупаемость достигается при сравнении с наиболее дорогостоящими конкурентами — Teachbase и Motivity. Даже по сравнению с наиболее дешёвой альтернативой, Unicraft, проект окупается менее чем за десять месяцев.

Следует также отметить, что расчёты произведены на текущую численность пользователей — 1000 сотрудников. Однако в случае увеличения числа пользователей, стоимость готовых решений будет расти пропорционально числу лицензий. В то же время собственная платформа обладает высокой масштабируемостью без существенного увеличения эксплуатационных расходов, что дополнительно усиливает её экономическую привлекательность на перспективу.

Таким образом, сравнительный анализ демонстрирует явную экономическую эффективность внедрения собственного решения в условиях крупной корпоративной инфраструктуры. Учитывая возможность масштабирования без линейного роста затрат, внутренняя LMS-платформа оказывается наиболее целесообразной инвестицией на горизонте в три года.

# 

# Расчёт показателей надёжности

Поскольку система находится в стадии разработки и внедрения, реальные эксплуатационные данные, такие как частота отказов и время восстановления, отсутствуют. Поэтому оценка надёжности системы будет проведена на основе статистической модели Коркорэна, которая применяется для прогнозирования вероятности безотказной работы программного обеспечения с учётом количества выявленных и исправленных ошибок разных типов [39].

Модель относится к статистическим методам оценки надёжности и не учитывает время тестирования, а опирается исключительно на результаты испытаний, в которых обнаружено и устранено определённое количество ошибок. В модели используются априорные вероятности обнаружения ошибок для каждого типа, что позволяет рассчитать вероятность безотказного функционирования системы на момент оценки [39].

Вероятность безотказного выполнения программы R в модели Коркорэна вычисляется по формуле:

Где N — общее число запусков программы, — число безотказных запусков, K — количество типов ошибок, а ​ — вероятность обнаружения ошибки i-го типа при тестировании. Если N=0, то вероятность R=0. Для оценки вероятностей ​ используются данные о типичных ошибках, характерных для прикладного программного обеспечения, и их вероятностях возникновения. Они приведены в таблице 3.4.

Таблица 3.4 – Вероятности возникновения типичных ошибок в прикладном программном обеспечении

|  |  |  |
| --- | --- | --- |
| № | Тип ошибки | Вероятность появления |
| 1 | Ошибки вычислений | 0,09 |
| 2 | Логические ошибки | 0,26 |
| 3 | Ошибки ввода-вывода | 0,16 |
| 4 | Ошибки манипулирования данными | 0,18 |
| 5 | Ошибки сопряжения | 0,17 |
| 6 | Ошибки определения данных | 0,08 |
| 7 | Ошибки в БД | 0,06 |

Переходя от теоретического описания к практическому применению, следует отметить, что метод Коркорэна основан на поэтапной оценке надёжности с постепенным повышением уровня доверия [39]. На первом этапе рассматривается предварительный уровень в 70%, который позволяет получить общую картину готовности системы при минимуме предположений. Второй этап предполагает достижение уровня надёжности 80% и требует более строгих исходных данных, включая частичное покрытие тестами. На заключительном этапе оценивается надёжность на уровне 95%, что соответствует высокой степени уверенности в безотказной работе системы. Такой поэтапный подход позволяет постепенно уточнять параметры надёжности и переходить от предварительной оценки к наиболее точной и обоснованной.

Исходя из этого, приступим к практическому расчёту надёжности разработанной системы для ОАО «ГлобалСофт». Следуя методологии Коркорэна, поочерёдно увеличивая количество успешных безотказных запусков и уточняя параметры вероятностей выявления ошибок, определим фактические показатели надёжности на каждом этапе.

На первом этапе, соответствующем уровню надёжности в 70%, критерии тестирования менее строгие, поэтому количество безотказных запусков относительно невелико. Количество безотказных запусков по типам ошибок приведено в таблице 3.5.

Таблица 3.5 – Количество безотказных запусков по типам ошибок на этапе 70% надёжности

|  |  |  |
| --- | --- | --- |
| Тип ошибки |  | (безотказных запусков) |
| Ошибки вычислений | 0.09 | 64 |
| Логические ошибки | 0.26 | 65 |
| Ошибки ввода-вывода | 0.16 | 70 |
| Ошибки манипулирования данными | 0.18 | 68 |
| Ошибки сопряжения | 0.17 | 72 |
| Ошибки определения данных | 0.08 | 73 |
| Ошибки в БД | 0.06 | 75 |

Для расчёта вероятности безотказной работы системы на первом этапе используем выражение, учитывающее доли различных типов ошибок и соответствующие вероятности их невыявления. Подставляя значения в формулу, получаем:

R = 0.09 \* 0.64 + 0.26 \* 0.65 + 0.16 \* 0.70 + 0.18 \* 0.68 + 0.17 \* 0.72 + 0.08 \* 0.73 + 0.06 \* 0.75 = 0.6868

Таким образом, на первом этапе расчёта вероятность безотказной работы составляет 0.6868, или 68.7%, что близко к целевому уровню надёжности в 70% и соответствует предварительной оценке готовности системы.

Для повышения уровня надёжности до 80% требуется увеличить количество успешных безотказных запусков. Рассмотрим, как изменяются показатели системы при новых значениях , представленных в таблице 3.6.

Таблица 3.6 – Количество безотказных запусков по типам ошибок на этапе 80% надёжности

|  |  |  |
| --- | --- | --- |
| Тип ошибки |  | (безотказных запусков) |
| Ошибки вычислений | 0.09 | 75 |
| Логические ошибки | 0.26 | 80 |
| Ошибки ввода-вывода | 0.16 | 83 |
| Ошибки манипулирования данными | 0.18 | 82 |
| Ошибки сопряжения | 0.17 | 85 |
| Ошибки определения данных | 0.08 | 86 |
| Ошибки в БД | 0.06 | 88 |

Для оценки надёжности системы на втором этапе подставим актуальные значения в расчётную формулу, учитывающую доли типовых ошибок и вероятности их невыявления:

R = 0.09 \* 0.75 + 0.26 \* 0.80 + 0.16 \* 0.83 + 0.18 \* 0.82 + 0.17 \* 0.85 + 0.08 \* 0.86 + 0.06 \* 0.88 = 0.8219

После выполнения вычислений получаем значение надёжности 0.8219, что эквивалентно 82.2%. Это соответствует промежуточному уровню, при котором система демонстрирует уверенное приближение к целевому показателю в 80%, что позволяет перейти к следующему этапу оценки.

Далее рассмотрим третий этап, при котором достигается высокий уровень надёжности — 95%. На этом этапе качество тестирования значительно улучшилось, и количество безотказных запусков существенно возросло, что отражено в таблице 3.7.

Таблица 3.7 – Количество безотказных запусков по типам ошибок на этапе 95% надёжности

|  |  |  |
| --- | --- | --- |
| Тип ошибки |  | (безотказных запусков) |
| Ошибки вычислений | 0.09 | 90 |
| Логические ошибки | 0.26 | 93 |
| Ошибки ввода-вывода | 0.16 | 92 |
| Ошибки манипулирования данными | 0.18 | 94 |
| Ошибки сопряжения | 0.17 | 95 |
| Ошибки определения данных | 0.08 | 96 |
| Ошибки в БД | 0.06 | 97 |

Для оценки надёжности системы на завершающем, третьем этапе расчётов, используем обновлённые значения вероятностей невыявления ошибок и их удельных весов:

R = 0.09 \* 0.90 + 0.26 \* 0.93 + 0.16 \* 0.92 + 0.18 \* 0.94 + 0.17 \* 0.95 + 0.08 \* 0.96 + 0.06 \* 0.97 = 0.9357

После выполнения вычислений итоговое значение надёжности составляет 0.9357, или 93.6%. Этот результат близок к заданному уровню в 95%, что указывает на высокую степень отказоустойчивости системы. Для точного достижения целевого значения можно либо дополнительно повысить количество безотказных запусков, либо уточнить параметры модели, в том числе вероятности обнаружения ошибок.

Таким образом, в результате расчётов, выполненных по методу Коркорэна, определены значения вероятности безотказной работы системы на трёх этапах. На первом этапе надёжность составила 68.7%, на втором — 82.2%, на третьем — 93.6%. Эти данные подтверждают, что с увеличением объёма тестирования и уточнением параметров модель демонстрирует стабильный рост надёжности. Достигнутый уровень в 93.6% близок к целевому и позволяет рассматривать систему как практически готовую к эксплуатации.

# 

# Анализ рисков

На этапе внедрения и эксплуатации внутренней обучающей платформы возможны различные риски, способные повлиять на стабильность работы системы, её безопасность и эффективность использования. В таблице 3.8 представлен обзор ключевых категорий рисков и их предварительная оценка.

Таблица 3.8 – Анализ рисков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория риска | Возможный риск | Вероятность | Влияние | Комментарий / меры управления |
| Технические | Сбой в работе платформы из-за ошибки в коде | Средняя | Высокое | Регулярное тестирование, автоматизация CI/CD |
|  | Несовместимость с LDAP или внутренней сетью | Низкая | Среднее | Прототипирование, ручное тестирование интеграции |
| Организационные | Недостаток ресурсов на поддержку | Средняя | Среднее | Назначение ответственного, контроль нагрузки |
|  | Отсутствие мотивации у сотрудников использовать систему | Средняя | Высокое | Обучение, вовлечение пользователей, геймификация |
| Информационная безопасность | Утечка персональных данных при ошибочной настройке | Низкая | Высокое | Ограничение доступа, аудит, шифрование |
|  | Взлом или атака на систему | Низкая | Высокое | Хостинг во внутренней сети, защита на уровне сервера |
| Функциональные | Недостаточная адаптация под нужды подразделений | Средняя | Среднее | Гибкая архитектура, сбор обратной связи |
|  | Отсутствие нужного функционала | Средняя | Среднее | Постепенное масштабирование, приоритеты по итерациям |

Наибольшую угрозу представляют технические и организационные риски, особенно связанные с эксплуатацией и адаптацией платформы под реальные условия. Тем не менее, все риски считаются управляемыми при условии соблюдения предусмотренных мер контроля. В целях повышения надёжности и оперативного реагирования на возможные сбои рекомендуется внедрить систему мониторинга ключевых компонентов платформы с автоматической отправкой уведомлений при обнаружении критических сбоев или отклонений в работе. Это позволит своевременно выявлять и устранять неисправности, минимизируя простой и снижая потенциальные потери.

Итак, в третьей главе была выполнена комплексная оценка экономической и технической целесообразности внедрения собственной LMS-платформы в условиях корпоративной инфраструктуры.

На первом этапе был проведён сравнительный анализ совокупной стоимости владения четырёх популярных коммерческих решений — iSpring, Teachbase, Unicraft и Motivity. Экономическая оценка, представленная в разделе 3.1, подтвердила, что разработка собственной платформы позволяет достичь окупаемости менее чем за десять месяцев. Особенно значимым это преимущество становится при сравнении с наиболее дорогими системами — Teachbase и Motivity. Даже по сравнению с наиболее доступным вариантом, Unicraft, собственная разработка оказывается экономически выгодной уже в первый год эксплуатации. Отдельно подчёркнута высокая масштабируемость собственной системы, позволяющая обслуживать большее число пользователей без существенного роста затрат, в отличие от лицензионных моделей.

В продолжение была выполнена расчётная оценка надёжности платформы по методу Коркорэна. На трёх последовательно усложняющихся этапах были получены значения вероятности безотказной работы: 68.7%, 82.2% и 93.6%. Эти результаты демонстрируют стабильное улучшение качества системы по мере увеличения объёма тестирования и позволяют заключить, что текущая реализация близка к промышленному уровню надёжности.

Дополнительно рассмотрены риски, связанные с внедрением и эксплуатацией платформы. Основные угрозы — технические и организационные — оценены как управляемые. Для повышения устойчивости рекомендовано внедрение системы мониторинга с автоматическим оповещением об отказах, что обеспечит оперативное реагирование и поддержание высокого уровня доступности.

Таким образом, проведённый анализ подтверждает, что собственная LMS-платформа является обоснованным и выгодным решением как с экономической, так и с технической точки зрения. С учётом расчётов и перспектив масштабирования проект отвечает требованиям корпоративной инфраструктуры и обеспечивает долгосрочную эффективность.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была поставлена и решена задача разработки корпоративной образовательной платформы с использованием искусственного интеллекта, адаптированной под внутренние процессы организации-заказчика — ОАО «ГлобалСофт». Разработка охватывала весь цикл проектирования — от анализа исходных условий и выбора технических средств до реализации, тестирования и экономической оценки эффективности внедрения.

В первой главе проведён детальный анализ деятельности предприятия-базы преддипломной практики и организации-заказчика, что позволило выявить ключевые проблемы в управлении обучающими материалами и ограничения существующих решений. Анализ популярных LMS-систем — iSpring, Teachbase, Unicraft и Motivity — подтвердил недостаточную адаптируемость и высокую стоимость лицензий, что послужило основанием для выбора стратегии создания собственной платформы. Для реализации системы был выбран современный стек технологий: серверная часть на Spring Boot с использованием Kotlin, FastAPI на Python для сервисов обработки, фронтенд на Next.js, а также нейросетевые инструменты Whisper и Mistral AI для автоматизации создания учебного контента. Это обеспечило фундамент для создания гибкой и масштабируемой системы, отвечающей техническим и бизнес-требованиям заказчика.

Вторая глава описывает этапы проектирования и разработки платформы. Были построены структурные и поведенческие модели разрабатываемой платформы с использованием UML, что позволило чётко определить архитектуру системы и взаимодействие её компонентов. Создан дизайн-проект, включающий визуальные и интерфейсные решения, а также сформирована структурная карта приложения с описанием API-маршрутов и клиентских представлений. Особое внимание уделялось интеграции с корпоративным каталогом LDAP и организации административной панели, хранению медиафайлов через GridFS, динамической конфигурации и реактивному взаимодействию с использованием WebSocket. Реализована система разграничения доступа на основе ролей, а также интегрированы возможности генеративных нейросетей для автоматизации создания конспектов и тестов. Контрольный пример подтвердил полноту и корректность реализации, соответствие системы техническому заданию. Кроме того, важным инструментом организации и управления рабочим процессом на всех этапах реализации проекта выступила канбан-доска, созданная с использованием сервиса Trello. Она обеспечила прозрачность выполнения задач, позволила своевременно выявлять узкие места и эффективно планировать последующие действия, что способствовало системному и последовательному развитию проекта.

Третья глава была посвящена экономической и технической оценке проекта. Проведён сравнительный анализ совокупной стоимости владения четырьмя популярными коммерческими LMS — iSpring, Teachbase, Unicraft и Motivity. Результаты показали, что собственная платформа достигает окупаемости менее чем за десять месяцев, что особенно выгодно в сравнении с более дорогостоящими решениями Teachbase и Motivity. Высокая масштабируемость собственной разработки позволяет существенно снизить рост эксплуатационных затрат при увеличении числа пользователей, что делает проект привлекательным для крупной корпоративной среды. Расчёт надёжности по методу Коркорэна продемонстрировал устойчивый рост вероятности безотказной работы с 68,7% на начальном этапе до 93,6% на заключительном, приближая систему к промышленному уровню качества. Анализ рисков выявил управляемые технические и организационные угрозы, для минимизации которых рекомендовано внедрение системы мониторинга с автоматическими уведомлениями, обеспечивающей своевременное реагирование и поддержание высокой доступности платформы.

Таким образом, комплексный анализ подтверждает экономическую и техническую целесообразность внедрения собственной LMS-платформы, разработанной с учётом специфики корпоративной инфраструктуры и требований организации. Реализованное решение обладает высокой надёжностью, масштабируемостью и функциональностью, что обеспечивает его перспективность для долгосрочного использования и дальнейшего развития цифровой образовательной среды компании.

# СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

1. Маврицкая Е. Подборка: 10 лучших LMS в России по мнению пользователей // LmsList. URL: https://lmslist.ru/podborka-10-luchshih-lms-v-rossii-po-mneniju-polzovatelej/ (дата обращения: 26.04.2025).
2. Farith Jose Heras García. Spring Boot, Quarkus, or Micronaut? // DZone. URL: https://dzone.com/articles/spring-boot-quarkus-or-micronaut (Дата обращения: 26.04.2025).
3. Nikhil Soman Sahu. Spring Boot vs. Quarkus vs. Micronaut: The Best Java Frameworks for Modern Development // Dev.to. URL: https://dev.to/nikhilxd/spring-boot-vs-quarkus-vs-micronaut-the-best-java-frameworks-for-modern-development-5ccd (Дата обращения: 26.04.2025).
4. Evgenia Verbina. Which Is the Best Python Web Framework: Django, Flask, or FastAPI? // JetBrains. URL: https://blog.jetbrains.com/pycharm/2025/02/django-flask-fastapi/ (Дата обращения: 26.04.2025).
5. sanitar1988. Сравнение Vosk и Whisper // Habr. URL: https://habr.com/ru/articles/814057/ (Дата обращения: 26.04.2025).
6. Salman Ravoof. Next.js vs React? It’s a Partnership, Not a Competition // Kinsta. URL: https://kinsta.com/blog/nextjs-vs-react/ (Дата обращения: 03.05.2025).
7. System Properties Comparison Apache Cassandra vs. Couchbase vs. MongoDB // DB-Engines. URL: https://db-engines.com/en/system/Apache+Cassandra%3BCouchbase%3BMongoDB (Дата обращения: 03.05.2025).
8. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006.
9. Литвинов Ю. Лекция 3: Моделирование, UML // Computer Science Center. URL: https://compscicenter.ru/courses/Software\_Design/2022-spring/classes/9039/ (Дата обращения: 24.11.2023).
10. achekalin. Понимание LDAP-протокола, иерархии данных и компонентов записей // Habr. URL: https://habr.com/ru/articles/538662/ (дата обращения: 26.04.2025).
11. Муратов И. Объединённая служба каталога (LDAP) // ALT Linux. URL: https://docs.altlinux.org/ru-RU/archive/2.4/html-single/master/alt-docs-master/ch06s11.html (дата обращения: 26.04.2025).
12. Authenticating a User with LDAP // Spring.io. URL: https://spring.io/guides/gs/authenticating-ldap (дата обращения: 26.04.2025).
13. Securing a Web Application // Spring.io. URL: https://spring.io/guides/gs/securing-web (дата обращения: 26.04.2025).
14. Caching Data with Spring // Spring.io. URL: https://spring.io/guides/gs/caching (дата обращения: 26.04.2025).
15. Class BCrypt // Spring Security. URL: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html (дата обращения: 26.04.2025).
16. Ioram Gordadze. Spring Security With JWT for REST API // Toptal. URL: https://www.toptal.com/spring/spring-security-tutorial (дата обращения: 26.04.2025).
17. Accessing Data with MongoDB // Spring.io. URL: https://spring.io/guides/gs/accessing-data-mongodb (дата обращения: 26.04.2025).
18. The Maven Publish Plugin // Gradle. URL: https://docs.gradle.org/current/userguide/publishing\_maven.html (дата обращения: 26.04.2025).
19. Baeldung. Spring Data Reactive Repositories with MongoDB // Baeldung. URL: https://www.baeldung.com/spring-data-mongodb-reactive (дата обращения: 26.04.2025).
20. Mohammad Ashour. A Guide to React Localization with i18next // Phrase. URL: https://phrase.com/blog/posts/localizing-react-apps-with-i18next/ (дата обращения: 26.04.2025).
21. Marek Chodak. Change Data Capture with MongoDB Change Streams // Medium. URL: https://medium.com/@marekchodak/change-data-capture-with-mongodb-change-streams-539a02cf401d (дата обращения: 26.04.2025).
22. STOMP Over WebSocket // jmesnil.net. URL: https://jmesnil.net/stomp-websocket/doc/ (дата обращения: 26.04.2025).
23. Baeldung. GridFS in Spring Data MongoDB // Baeldung. URL: https://www.baeldung.com/spring-data-mongodb-gridfs (дата обращения: 26.04.2025).
24. XMLHttpRequestUpload: load event // MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequestUpload/load\_event (дата обращения: 31.05.2025).
25. ffmpeg Documentation // FFmpeg. URL: https://ffmpeg.org/ffmpeg.html?utm\_source=chatgpt.com (дата обращения: 31.05.2025).
26. Working with wav files in Python using Pydub // GeeksforGeeks. URL: https://www.geeksforgeeks.org/working-with-wav-files-in-python-using-pydub/ (дата обращения: 31.05.2025).
27. How To Use ThreadPoolExecutor in Python 3 // DigitalOcean. URL: https://www.digitalocean.com/community/tutorials/how-to-use-threadpoolexecutor-in-python-3 (дата обращения: 31.05.2025).
28. averkij. Распознавание речи, генерация субтитров и изучение языков при помощи Whisper // Habr. URL: https://habr.com/ru/companies/ods/articles/692246/ (дата обращения: 26.04.2025).
29. Fullstop Punctuation Multilang Large // Dataloop. URL: https://dataloop.ai/library/model/oliverguhr\_fullstop-punctuation-multilang-large (дата обращения: 31.05.2025).
30. Mistral AI. Mistral AI API Documentation // Mistral AI. URL: https://docs.mistral.ai/api/ (дата обращения: 31.05.2025).
31. Healey A. Running Untrusted Python Code // HealeyCodes. 27 июля 2023 г. URL: https://healeycodes.com/running-untrusted-python-code (дата обращения: 31.05.2025).
32. RestrictedPython — A restricted execution environment for Python // ReadTheDocs. URL: https://restrictedpython.readthedocs.io/en/latest/ (дата обращения: 31.05.2025).
33. Pyodide — Python for the browser and Node.js using WebAssembly // Pyodide. URL: https://pyodide.org (дата обращения: 31.05.2025).
34. Rehkopf M. Kanban vs. Scrum: which agile are you? Uncover the key considerations when choosing between scrum or kanban, and what to do if you can’t decide // Atlassian. URL: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (дата обращения: 31.05.2025).
35. Gartner Research. Unlocking Value: Principles of Total Cost Ownership // Gartner. URL: https://www.gartner.com/en/documents/ (дата обращения: 29.05.2025).
36. Аткинсон Э.А., Банкер Р.Д., Каплан Р.С., Юнг М.С. Управленческий учёт. — СПб.: ООО «Диалектика», 2019.
37. Ремнева И., Можарова Т., Аксенова О. Срок окупаемости проекта // Fintablo. URL: https://fintablo.ru/business/srok-okupaemosti-proekta (дата обращения: 26.04.2025).
38. Окупаемость бизнеса: способы расчета срока окупаемости проекта // Franshiza.ru. URL: https://franshiza.ru/article/read/okupaemost\_biznesa/ (дата обращения: 26.04.2025).
39. Василенко Н. В., Макаров В. А. Модели оценки надежности программного обеспечения // Вестник НовГУ. 2004. №28. URL: https://cyberleninka.ru/article/n/modeli-otsenki-nadezhnosti-programmnogo-obespecheniya (дата обращения: 29.05.2025).