



Geekbrains

Проектирование и разработка серверной части мобильного приложения для быстрой проверки знаний обучающихся в образовательной среде

Программа: Разработка.
Специализация: Программист JAVA
Панферов Антон Сергеевич

Лениногорск
2024

Содержание

Введение	3
Глава 1. Основы разработки мобильных приложений.....	7
1.1 Особенности разработки мобильных приложений.....	7
1.2 Этапы разработки мобильных приложений.....	13
1.3 Существующие сервисы и приложения по проверки знаний обучающихся.....	17
1.4 Преимущества разрабатываемого нами мобильного приложения..	19
Глава 2. Разработка серверной части мобильного приложения.....	21
2.1 Обоснование выбранных инструментов разработки.....	21
2.2. UX/UI – дизайн.....	24
2.3 Проектирование базы данных.....	26
2.4 API приложения.....	31
2.5 Основные модули приложения.....	33
2.6 Модульное и интеграционное тестирование.....	39
2.7 Сквозное тестирование.....	40
2.8 Разработка предложений по улучшению и расширению функционала приложения.....	50
Заключение.....	52
Список используемой литературы.....	53

Введение

Цель дипломной работы:

- спроектировать и разработать серверную часть мобильного приложения для проверки знаний обучающихся в образовательной среде.

Основные задачи:

- Изучить литературу, касающуюся темы исследования;
- Рассмотреть особенности разработки серверной мобильных приложений;
- Изучить существующие приложения по данной направленности, обосновать их достоинства и недостатки;
- составить план разработки мобильного приложения, продумать основной функционал и базу данных;
- спроектировать и создать макет приложения, делая акцент на backend-разработке;
- разработать предложения по дальнейшему расширению функциональности приложения

Инструменты, используемые в разработке:

- основной дистрибутив
- postgresSQLgit
- html
- Docker

Обоснование выбранных инструментов и указание их версий будет рассмотрено в разделе 2.1.

Актуальность и значимость. Образовательная деятельность является неотъемлемой частью нашей жизни. Каждый современный человек учился в школе, получал высшее или среднее специальное образование. Но если раньше однажды полученных знаний хватало на всю оставшуюся жизнь, то в XXI веке приходится постоянно переобучаться и получать новые знания. И такая тенденция сохранится ближайшее время.

По данным некоторых исследований источник информация удваивается каждый год, растут и прикладные знания, какую бы отрасль деятельности мы не взяли: медицина, инженерия, торговля, психология и т.д. Это диктует необходимость постоянно переучиваться - повышать свою квалификацию. Отсюда и возрастает роль учителя в образовании.

Учитель, как и ученики сталкивается с колоссальным объёмом информации: приходится тратить много времени на подготовку к уроку, общение с обучающимися, проверку их знаний и навыков. Помимо этого, необходимо ещё вести методическую документацию, что ещё отрывает на себя ресурс времени. Понятно, что современный урок длительностью в 45 минут (или пара в 90 минут) покрыть эти потребности не может в силу своей быстротечности. И педагогу, преподавателю всегда приходится расставлять приоритеты в своей деятельности, но даже если он делает это, то времени всё равно не хватает, и приходится чем-то жертвовать: либо давать материал на самостоятельное изучение, либо сократить проверку, либо вообще ограничиться самоконтролем учащихся друг друга и т.д. Поэтому есть потребность автоматизировать процесс обучения, который будет полезен не только учителю, но и ученикам.

Попытки такой автоматизации уже имеются. Это различные веб-сервисы для школ и колледжей, приложения по тренировке знаний, веб-сайты, онлайн-школы и т.д. Все они направлены, чтобы оптимизировать всё то, что поддаётся оптимизации в образовательном процессе. Однако все эти сервисы лишены одного существенного свойства: они не могут обеспечить оптимизацию времени самого урока либо совсем, либо в определённой части его части.

Что именно оптимизируют популярные на сегодняшний 2024 год сервисы, будет показано далее. Можно с уверенностью сказать, что на сегодняшний день сам урок остаётся малообитаемым островом для информационных технологий. Поэтому мы задались целью - разработать инструмент, который позволил оптимизировать сам урок, позволил бы педагогу высвободить время для более важных вещей; а ученику позволил бы с интересом и увлечением проходить обучение.

Если взять классическую структуру урока, не вдаваясь в специфику, то он разделён на следующие этапы (источник):

- Организация - включает приветствие и контроль посещаемости;
- Актуализация - включает проверку домашнего задания;
- Подача нового материала;
- закрепление нового материала;
- Проверка усвоенного;
- Рефлексия.

Понятно, что в зависимости от специфики дисциплины, конкретного урока, уровня подготовки обучающихся, те или иные этапы могут удлиняться, укорачиваться, либо вообще исчезать из урока. Однако каждый этап можно оптимизировать по-своему. Но мы решили остановиться на проверке знаний.

Проверка знаний один из важных этапов образовательного процесса. Все проверки можно разделить на две большие группы:

- 1) проверка знаний под контролем учителя - то есть такая проверка, при которой обучающийся находится под присмотром учителя, чтобы он не смог списать готовый ответ. Такая проверка всегда проходит на уроке.
- 2) проверка знаний без присутствия учителя - такая проверка проходит вне урока.

Мы запланировали создать приложение, которое бы автоматизировало бы процесс проверки знаний обучающихся именно на уроке. Помимо этого, важного для нашей проектной работы, разделения отметим, что также существуют устные и письменные проверки. Устные проверки мы пока затрагивать не будем - возможно эта интересная идея для расширения функционала. Остановимся на письменных проверках: они также в свою очередь бывают такими, где сама по себе форма письменной проверки важна - например, решение математической задачи, или сочинение. Такие проверки также отпадают, так как идея их автоматизации - также для расширения функционала, но не основа для текущей работы.

Таким образом мы остановим свой взгляд именно на письменных проверках во время урока, для которых не важна форма записи. Сюда входят: **тесты**, прежде всего, с разными вариациями вопросов, а также **вопросы открытого типа** - где нужно вписать слово, словосочетание, пример и пр.. Именно на оптимизацию этой проверки будет направлен функционал нашего приложения.

Вот несколько аргументов, почему в основе функционала нашего приложения был взят именно такой вид проверки:

1. Этот вид проверки преобладает в образовательной среде, начиная от младшей школы и заканчивая курсами переподготовки и повышения квалификации. 2. Он позволяет быстро оценить и довольно широко оценить уровень знаний и умений обучающихся.

- Такая проверка готовится быстро;
- Относительно быстро проходит и сам процесс проверки;
- Относительно быстро и удобно проходит подсчёт результатов.

Какими же недостатками обладает текущий алгоритм проверки с точки зрения использования?!

1) Несмотря на то, что этот алгоритм проверки проводится относительно быстро, тем не менее преподавателю приходится тратить на него время.

- Время уходит на то, чтобы подготовить материалы проверки: найти такие материалы либо приготовить их самостоятельно;
- Время уходит на то, чтобы подготовить материальный носитель - распечатать материалы проверки/тестирования или сделать копии;
- Время уходит на то, чтобы раздать эти материалы обучающимся;
- Время уходит на то, чтобы собрать бланки с ответами;
- Время уходит на то, чтобы обработать ответы в соответствии с ключом. А если несколько вариантов, то, соответственно, с несколькими ключами;
- Время уходит на то, чтобы упорядочить эти материалы для хранения, чтобы относительно быстро иметь к ним доступ в последующем;

2) Материальные носители, а это бумага - ресурс довольно исчерпаемый, при многократном использовании - изнашивается, занимает место на полке. Как правило, подготовить для каждого обучающегося материалы - это проблема учителя, а не организации. Соответственно - учитель должен найти, распечатать эти материалы, размножить их на количество обучающихся, а при износе - продублировать материалы.

3) Частое использование одних и тех же проверочных заданий приводит к утечке ответов, то есть обучающиеся из параллельных групп/классов узнают ответы на задания у тех, кто уже прошёл ответ. Предотвратить списывание при традиционной проверке и заставить ученика выполнять задания собственным умом – ещё одна задача. Дело в том, что порядок заданий и ответов на них остаётся неизменным, вариантов проверочных работ как правило - два, иногда - 3, крайне редко - 4. Поэтому ученик может просто списать последовательность ответов на вопросы.

4) Как правило, преподаватель самостоятельно не разрабатывает варианты проверочных работ, а берёт их в готовом виде из открытых источников, чтобы сэкономить время. Но в открытых источниках также есть и ответы, которые легко обнаруживаются учениками. Поэтому приходится постоянно вести контроль за ходом выполнения проверочной работы, чтобы исключить списывания и уложиться в отведённое для теста время.

5) Поскольку подсчёт баллов за правильные и неправильные ответы происходит вручную, что приводит не только к тратам времени, как было сказано в пункте №1, но и к вероятности появления ошибок - то есть в оценке обучающихся возрастает субъективный фактор.

6) Статистика проверок и оценок обучающихся также ведётся в ручном режиме, что вызывает неудобства учителя, он может не усмотреть детали прогресса учеников, сделать акцент на определённые темы, которые вызвали трудности.

Это те недостатки, которые мы видим при самом общем рассмотрении проблематики. Но их достаточно, чтобы искать способы оптимизировать этот процесс.

Представим теперь ситуацию, что существует мобильное приложение, которое позволяет: а) выполнять проверку знаний обучающихся по любым дисциплинам и направлениям; б) преподаватель может как сам создавать свои вопросы, так и пользоваться готовой базой проверочных работ; в) порядок вопросов и ответов на них будут меняться в зависимости от количества учеников и количества вариантов; г) по каждому ученику ведётся статистика прохождения проверочных работ - учитель может видеть статистику как отдельного ученика, так и всего класса и общего количества учеников, что облегчает постановку итоговых оценок, а также работы с теми темами, которые вызывают затруднения; д) у приложения существует удобный интерфейс, который позволяет легко ориентироваться в функционале: для учителя - это важность - легко добавлять проверочные работы, видеть прогресс прохождения, статистику; для ученика - это удобный функционал для прохождения проверочных работ.

Созданию такого приложения и посвящена наша работа. О недостатках ныне существующих сервисов с подобным функционалом мы поговорим ниже.

Глава 1. Основы разработки мобильных приложений

1.1 Особенности разработки мобильных приложений

На сегодняшний день мировой рынок мобильных приложений стремительно идёт вверх. Так в статье [###] сайта mindbox.ru говорится, что среднее количество скачиваний на пользователя России составляет - .

Получается, пользователь проводит в смартфоне очень много времени, что существенно увеличивает шансы на востребованность нашего приложения. Тем более Россия уже не один год проводит политику импортозамещения, что повышает востребованность в отечественных IT-разработках.

Таким образом, наша данное направление является очень перспективным для реализации нашего проекта.

Выбор вида мобильного приложения

Чтобы оправдать собственные ожидания и экономно расходовать имеющиеся ресурсы при создании мобильного приложения, необходимо понимать, какой вид приложения мы хотим создать.

Существуют следующие виды приложений:

1. Нативные (от англ. native - родственный) - приложения, которые разработаны под определённую платформу (родную). Для Android-приложений - это приложения, созданные в программе Android Studio, и написанные на языках java и kotlin (последний одобрила корпорация Google в 2019 году). Главными достоинствами нативных приложений являются 1) доступ ко все аппаратным возможностям устройства (камера, геолокация, датчики движения и т.д.) и 2) потенциально меньшее количество багов, 3) производительность, 4) лучшая кастомизация, 5) сбор и анализ пользовательских данных. Недостатком нативного приложения будут: длительность разработки, дороговизна.

2. Веб-приложения - приложения, которые будут открываться браузером смартфона. По сути это веб-сайты, которые выглядят как мобильные приложения. Плюсы данных приложений: быстрая скорость разработки, совместимость с любой ОС, отсутствует согласованность с магазином. Недостатки: зависимость

от интернет-трафика, низкая кастомизация, нет доступа к аппаратной части устройства, сложнее поиск приложений в интернете.

3. Гибридные приложения - приложения, которые сочетают в себе как функционал нативного приложения, так и использование веб-ресурсов. Выглядят и используются как нативные приложения, используют компонент для открытия веб-ресурсов (WebView).

Преимущества:

- широкий функционал и высокая степень кастомизации;
- можно создать приложение, которое будет работать с несколькими платформами;
- удешевляют и ускоряют разработку MVP или несложного готового продукта для заказчиков;
- являются срединным решением между функционалом и производительностью нативного приложения и дешевизной веб-приложения.

Недостатки:

- слишком сложные приложения лучше делать нативными, как и приложения с громоздкими визуальными решениями вроде игр;
- разработка потребует больше времени и усилий, чтобы гибридное приложение выглядело и ощущалось как нативное;
- магазины приложений отклоняют недостаточно хорошо работающие гибридные приложения и важно соблюдать стандарты качества.

Таким образом, мы сделали обзор видов приложений и остановимся пока на нативном приложении.

Выбор операционной системы.

Так как мы пишем серверную часть мобильного приложения, для нас не имеет значение, какая операционная система будет для мобильного приложения.

Особенности разработки мобильных приложений

Разработка мобильных приложений - это динамичный процесс, требующий постоянного обучения и адаптации к новым технологиям и трендам. Успешная разработка требует подробного изучения платформы, тщательного планирования, тестирования и внимания к пользовательскому опыту.

Мобильные имеют ограниченные ресурсы, такие как процессорная мощность, оперативная память и батарея. Очень неприятно, если приложение будет потреблять слишком много заряда батареи, или оперативной памяти будет не хватать. Тут нужно соблюсти баланс между быстродействием приложения и ёмкостью ресурсной системы и батареи.

Приложение необходимо разрабатывать таким образом, чтобы пользовательский интерфейс был удобным и приятным, креативность и стандартизация пользовательского интерфейса также должна быть в балансе. При разработке UI также необходимо учитывать психологию человека.

Нужно обеспечить надежность и безопасность приложения, чтобы защитить персональные данные пользователя от несанкционированного доступа. И чем меньше приложение собирает персональные данные, тем лучше.

Приложения должны соответствовать определенным правилам и требованиям магазинов (Google Play, RuStore, AppGallery, AppStore), чтобы быть опубликованными. Мобильные приложения могут быть спроектированы так, чтобы при необходимости работать в оффлайн-режиме.

Мобильные устройства имеют широкий набор аппаратных возможностей, таких как камера, геолокация, акселерометр и датчики отпечатков пальцев. Можно использовать эти возможности для создания более интерактивного и функционального приложения.

Мобильные платформы и устройства постоянно обновляются, поэтому необходимо обеспечивать поддержку своих приложений, выпуская регулярные обновления и исправления ошибок (по стандарту не реже одного раза в год). Ещё более подробно с особенностями и требованиями к разработке мобильных приложений можно почитать.

Таким образом, здесь представлены самые общие особенности разработки мобильных приложений без привязки к операционной системе. Их необходимо учитывать при создании нашего приложения.

Мы провели краткий обзор по особенностям и обосновали выбор направления разработки приложения для проверки знаний обучающихся. Но так как мы разрабатываем серверную часть, то для нас не имеет значения операционная система мобильных приложений.

1.2. Этапы разработки мобильных приложений

Чтобы разработка нашего приложения была эффективнее, необходимо видеть собственно разработку приложения, то есть написание функционала и логики программы в общем процессе жизненного цикла приложения. Иначе есть шанс создать никому ненужный сервис. Не будем претендовать на детальность каждого из этапов жизненного цикла приложения, но сделаем краткий обзор.

Примечание: далее по тексту "разработка" - это собственно написание программного кода; "создание" - полное создание приложения от появления идеи до получения обратной связи по приложению от пользователей (включает весь жизненный цикл приложения).

Создание приложения включает в себя несколько шагов: 1. Проработка идеи, 2. UX/UI, 3. Разработка, 4. Аналитика приложения, 5. Продвижение, 6. Развитие. Давайте кратко рассмотрим некоторые шаги.

Шаг 1. Проработка идеи. Анализ рынка.*

Поскольку у нас нет возможности обратиться в специализированную компанию по получению данных по нашему сегменту, то будем исходить из следующих аспектов:

- * Наш сегмент - образовательный.
- * Сейчас цифровизация образования интенсивно развивается: появляется большое количество онлайн-школ, мобильных приложений для подготовки к экзаменам; есть площадки по полноценному обучению. Обзор основных приложений мы изучим в следующей главе.

Основными потребителями (пользователями) нашего приложения будут учителя и ученики. Сайт mayaksbor.ru публикует такие данные на 2022-2023 у.г.:

- * Школы: - 17,7 миллиона школьников, - более 1 миллиона учителей,
- * ВУЗы: - 4,1 миллиона студентов, - 0,2 миллиона преподавателей,
- * СПО (по данным сайта rsr-online.ru на 2021-2022 уч.г.):
 - 3,4 миллиона студентов. - 0,3 миллиона преподавателей.

По численности преподавателей СПО найти актуальных данных не удалось, поэтому оценка численности посчитана следующим образом. Средний колледж в 600 студентов имеет в штате около 60 преподавателей общих и специальных дисциплин, не включая мастеров. Отсюда взято и соотношение 1:10 - на одного преподавателя приходится 10 студентов. Получается $3,4 \text{ миллиона} / 10 =$ примерно 0,34 миллиона. Округлили в меньшую сторону, чтобы уменьшить ошибку реальных оценок численности.

Если предположить, что приложением не будет пользоваться большая часть учеников и учителей младших классов, а это примерно ****35%**** от всех учителей школ и школьников, то:

- $(17,7 \text{ учеников} + 1 \text{ учителей}) - 35\% = 12,1 \text{ миллиона человек от школы.}$

Считаем общее количество обучающихся и преподавателей:

- $12,1 \text{ со школы} + (4,1 \text{ студентов ВУЗов} + 3,4 \text{ студентов СПО}) + (0,2 \text{ преп-лей ВУЗов} + 0,3 \text{ преп-лей СПО}) = \mathbf{20,1 \text{ млн.}}$ пользователей*

Предположим, что 1/5 часть не будет пользоваться приложением в силу инерции и возраста.

- $20,1 \text{ млн.} - 20\% = \mathbf{16,07 \text{ млн.}}$ ***

Ещё 20% процентов не будут пользоваться в силу того, что это приложение им не понравится:

- $16,07 - 20\% = \mathbf{12,86 \text{ млн.}}$ ***

Исходя приведённых цифр мы имеем количество потенциальных пользователей: 12 млн. человек.

12 млн.пользователей приложения - есть над чем работать, не правда ли?

Мы считаем, что на сегодняшний день в России сложилась благоприятная среда, чтобы внедрить такого рода приложения. *Во-первых*, цифровизация образования идёт давно, но полным ходом она пошла с 2020 года.

Во-вторых, политика государства в современных реалиях направлена на политику импортозамещения и поддержку IT-сферы.

В первую очередь это работники образования - а именно учителя и преподаватели, через них - ученики, их родители; пользователи, которым через приложение будет удобно проверить какие-либо знания у клиентов.

Но главное - это то, что приложением должны пользоваться две стороны - проверяющий и проверяемый, причём проверяющий - активный пользователь, а проверяемый - вынужденный, пассивный. Поэтому: маркетинг приложения должен быть нацелен в первую очередь на проверяющих - на учителей и преподавателей.

Мы исходим из того, что наше приложение будет бесплатным до этапа зрелости. Потом возможно рассмотреть внедрение платного функционала. На этапах внедрения и роста приложением можно будет пользоваться бесплатно, но с учётом показанной рекламы. Места функционала, в которых будет размещена реклама - будут рассмотрены на этапе описания функционала. Отметим, что оплачивать наше приложение будут заинтересованы активные пользователи (учителя и преподаватели).

Шаг 2. UX/UI. Этот шаг включает в себя дизайнерские решения приложения, но так как мы пишем серверную часть, то мы не будем останавливаться на этом этапе, отметим, что исходя из задачи нашего диплома - учебной, а не коммерческой, мы проработаем дизайн, который покроет нашу учебную задачу.

Шаг 3. Разработка

Включает в себя собственно разработку приложения - основная задача нашего дипломного проекта. Более подробно о разработке речь пойдёт (в главах 1.5, 2.3 - 2.5, 3.1 - 3.6). Здесь же вкратце поясним, какие задачи решаются на данном этапе

1. Выбор и обоснование технологического стека - какие программы будут использоваться для разработки, на каких языках программирования, с помощью какой СУБД будет реализована база данных. Очень важно выбрать корректную технологию, так как одна технология подходит лучше для решения одних задач, а другая - для других. Выбор стека технологий будет обоснован (в главе)

2. Проработка архитектуры. Существует два основных типа архитектур у которых разнонаправленные и однонаправленные потоки данных. Для небольших команд и проектов лучше использовать разнонаправленные архитектуры (MVVM, MVC, MVP и др.), так как они обычно проще для понимания и реализации. В то же время, для больших проектов лучше подойдут однонаправленные архитектуры (Clean Swift, MVI и др.), которые обеспечивают более строгое разделение обязанностей и упрощают управление сложными экранами.

3. Проработка компонентов приложения. Очень полезный шаг, и если его правильно продумать, то это позволит,

во-первых, сразу писать читабельный и чистый код в соответствии с принципами SOLID.

во-вторых, проработка сервисов - поможет в дальнейшем расширять приложение, с лёгкостью и быстротой добавлять новые функции, что крайне важно, когда приложение уже вышло в продакшн.

Описание архитектуры нашего приложения, его основных модулей будет дано (

Шаг 4. Анализ работы приложения

Этот этап включает в себя тестирование и отладку приложения. Важный этап, ибо если пользователь сталкивается с тем, что приложение неожиданно закрывается и не работает должным образом, это может привести к тому, что он перейдет к конкурентам. Поэтому необходимо интегрировать в ваше приложение сторонний сервис для мониторинга и отслеживания падений приложения.

Итак, мы определили, каких этапов необходимо придерживаться, чтобы цель создания нашего мобильного приложения была успешно достигнута. Также мы оценили целевую аудиторию, её численность; рассмотрели этапы создания пользовательского интерфейса, разработки, тестирования и анализа использования приложения потенциальными потребителями. В следующих главах мы остановимся подробнее на разработке приложения.

1.3 Существующие сервисы и приложения по проверки знаний обучающихся

В этой главе мы рассмотрим самые популярные образовательные приложения по оценке знаний обучающихся; выделим их основные достоинства и недостатки. Мы будем анализировать как веб-приложения, так и нативные мобильные приложения. И те, и другие могут применяться в образовании. Перечислим приложения, которые нам удалось найти в интернете (поисковая система "Яндекс" - далее для краткости: Яндекс - без кавычек) и на площадках Google Play и RuStore.

Веб-приложения

По поисковому запросу в Яндексе "ресурсы для проверки знаний студентов" были показаны результаты статей в стиле "топ образовательных ресурсов". Воспользуемся такими источниками:

1. [Лучшие сервисы для создания тестов](<https://webinar.ru/blog/online-test/> "Ссылка на webinar.ru") от 29.11.2022;
2. [ТОП-18 отечественных сервисов для помощи студенту](<https://vc.ru/services/438953-top-18-otechestvennyh-servisov-dlya-pomoshchi-studentu> "Ссылка на vc.ru") от 08.06.2022;
3. [Ресурсы для проведения Интернет- тестирования при оценке качества знаний студентов](<https://nsportal.ru/shkola/obshchepedagogicheskie-tehnologii/library/2021/12/12/resursy-dlya-provedeniya-internet> "Ссылка на nsportal.ru") от 12.12.2022;
4. [7 лучших сервисов для создания тестов и опросов](<https://www.eduneo.ru/7test/> "Ссылка на eduneo.ru") - дата публикации не указана.

Они приводят следующие ресурсы: Google Forms, Madtest, Kahoot, Socrative,* LearningApps, Quizizz, Quizlet, Online Test Pad, Classmarker, Мастер-Тест, Конструктор тестов, Anketolog, Let's test, 1С.Образование и другие...

Веб-ресурсы

Общие достоинства веб-ресурсов:

- позволяют легко добавлять и создавать тесты. Ввод текста теста или проверочного задания с клавиатуры легче и проще, чем через телефон. Само создание теста учителем будет проще, чем через телефон.
- они кросс-платформенные, их можно использовать на любом устройстве;

Общие недостатки:

- невозможность использовать веб-ресурс без отключения проектора при учениках;
- так как тестирование проходит в классах, где нет компьютеров, то через телефон некорректно отображаются элементы.

Мобильные приложения

Мы провели поиск мобильных приложений для тестирования знаний по нескольким поисковым запросам в яндексе и google-play и обнаружили, что приложений для проверки знаний не существует в том, виде как мы это представляем. Есть приложения, которые мы уже прорабатывали: quizizz, socrative. Их достоинства и недостатки мы уже перечисляли.

Приложения для проверки знаний, которые удавалось найти, - это приложения по отдельным предметам. Они направлены не на проверку знаний со стороны учителя, а на самопроверку, а точнее - на самоподготовку. Это полноценные тренажёры, которые помогают лучше узнать предмет и подготовиться к сдаче экзаменов.

1.4 Преимущества разрабатываемого нами мобильного приложения.

Если говорить о тех, достоинствах, которое мы хотим воплотить в нашем мобильном приложении, то большая часть из них, конечно, уже присутствует в различных веб-приложениях.

Первое. Приложение будет полностью бесплатным. Небольшая монетизация будет происходить за счёт рекламы. Рекламу мы планируем разместить в трёх местах:

- Когда учитель создал тест, перед загрузкой на сервер, в базу приложения.
- Когда ученик завершил прохождение теста перед результатами: чтобы посмотреть собственные результаты, необходимо просмотреть рекламу.
- Когда учитель захотел посмотреть статистику по своим ученикам: необходимо также просмотреть рекламу.

Второе. Существует простая авторизация, минимальные сведения - это номер телефона, фамилия и имя.

Третье. Лёгкий и удобный конструктор тестов. Тесты должны создаваться легко с телефона, удобно редактироваться.

Четвёртое. Большая база тестовых заданий, классифицированных по предметам, разделам и темам. Лёгкий поиск теста по темам и разделам.

Пятое. Возможность использовать мультимедиа при создании заданий: добавлять видео, рисунки, использовать камеру и микрофон.

Шестое. Геймификация процесса прохождения тестов. Стимулирование использования, прогресс достижений.

Мы осознаём, что очень хорошо использовать это приложение на android и IOS, так как это самые распространённые ОС телефона. Также было бы не плохо совместить мобильное приложение с веб-приложением, так как веб-приложение удобно демонстрировать аудитории, а также на компьютере удобнее

конструировать тесты. Мы учитываем эти факторы, и в дальнейшем мы расширим наше приложение. А на сегодняшний день мы считаем, что мобильная разработка должна покрыть минимальные потребности.

Глава 2. Разработка серверной части мобильного приложения

2.1 Обоснование выбранных инструментов разработки

Внимание: с проектом серверной части мобильного приложения вы можете ознакомиться по ссылке: <https://github.com/antonipan/cognition.git>.

ВАЖНОЕ ЗАМЕЧАНИЕ! Хотя мы в теоретической части и говорили о создании серверной части мобильного приложения, мы сначала напишем сырой проект сервера для веб-приложения. Веб-приложение – это не наша самоцель, это лишь платформа для перехода к мобильной версии приложения. Когда мы говорим о мобильном приложении, то подразумеваем, что в конечный наш продукт будет использоваться на мобильных телефонах, и под серверную часть будет написана соответствующая мобильная оболочка.

Для разработки приложения нам понадобится следующее программное обеспечение и инструменты разработки.

1. Бизнес-логика приложения будет создаваться на языке JAVA с использованием фреймворка SPRING версии SPRING BOOT 3.23.

- java – один из наиболее популярных языков программирования, эффективный для написания серверной части приложений. Язык имеет широкие возможности, массу встроенных библиотек, а также большую поддержку сообщества разработчиков в интернете.
- SPRING FRAMEWORK – один из популярных фреймворков для JAVA, имеет множество встроенных пакетов для создания веб-приложений, позволяет автоматизировать разработку, отладку, тестирование и работу приложения.

2. Создавать код приложения мы будем в интеллектуальной среде intelliJ IDEA - наиболее лучшая среда разработки, позволяет легко писать код и тестировать его, удобный функционал для рефакторинга.

3. Для java будем использовать пакет jdk.17 дистрибьютера libertica. Мы берём 17-ую версию пакета, так как он является долгосрочно поддерживаемой, в нём

присутствует весь необходимый для разработки арсенал библиотек, актуальный на 2024 год.

4. Графический дизайн будет разработан на языке HTML с использованием фреймворка Thymeleaf, который позволяет создавать динамические веб-страницы. Она позволяет разрабатывать компоненты графического интерфейса с указанием их кода, обладает широкими возможностями.

5. СУБД приложения мы решили выбрать PostgreSQL версии 16.2. Это одна из мощных, легко осваиваемых и доступных СУБД. Так как наш проект предполагает взаимодействие нескольких таблиц, то данная СУБД – оптимальный вариант. Также мы предполагаем, что будем хранить в базе json – файлы.

6. База данных будет развёрнута в Docker-контейнере через приложение Docker Desktop, графический интерфейс которого позволяет легко управлять контейнерами. Для управления контейнерами используется Docker 25.0.2. Для управления совокупностью контейнеров будет использован

5. Покрытие тестами будет с помощью библиотеки Junit и Mockito из библиотеки Spring Framework. Одна из популярных библиотек для написания тестов.

6. Для тестирования работы приложения и отслеживания изменений в базе данных мы использовали десктопные программы:

- Postman 10.24.3 – для тестирования REST-запросов к нашему приложению.
- Debeaver 24.0.0 – просмотра состояния базы данных.

7. При разработке используются следующие зависимости:

- implementation 'org.springframework.boot:spring-boot-starter-web' – основная библиотека для написания веб-приложения.
- implementation 'org.springframework.boot:spring-boot-starter-thymeleaf' – библиотека для создания динамических веб-страниц.
- developmentOnly 'org.springframework.boot:spring-boot-devtools' – инструменты разработчика.

Модули для подключения и работы с базой данных.

- implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
- runtimeOnly 'org.postgresql:postgresql'

Модули для тестирования кода:

- testImplementation 'org.springframework.boot:spring-boot-starter-test'
- testImplementation group: 'org.mockito', name: 'mockito-core', version: '5.8.0'

Модуль для управления безопасностью приложения:

- implementation group: 'org.springframework.boot', name: 'spring-boot-starter-security', version: '3.2.2' – классы и методы написаны, но не используются пока что в нашей версии приложения.

Модули для отслеживания основных показателей приложения.

- implementation group: 'org.springframework.boot', name: 'spring-boot-starter-actuator', version: '3.2.2'
- implementation group: 'io.micrometer', name: 'micrometer-registry-prometheus', version: '1.12.3'

Итак, давайте опишем практическую составляющую нашего проекта.

2.2. UX/UI - дизайн

Рабочее название нашего приложения “cognition”. Это название будет часто встречаться в коде и в структуре базы данных.

Так как наша цель – создать серверную часть приложения, мы сделали дизайн максимально упрощённым, чтобы была возможность посмотреть работу функционала и примерно понимать в каком направлении двигаться.

Далее, чтобы было понятно, в приложении существуют роли:

- учитель – далее это тот, кто обучает, роль с ограниченными правами.
- Ученик – далее это то, кто обучается, роль с ограниченными правами.
- Администратор – далее это тот, кто контролирует работу приложения изнутри, максимальные права доступа.

Фронтэнд приложения представляет из себя набор html-страниц, распределённых по нескольким пакетам приложения:

Пакет “admin” включает в себя страницы:

- registration – представляет из себя форму заполнения логина, пароля и роли.

Примечание: поле «role» (роль) – должно быть представлено не окном для ввода текста; роль должна выбираться из выпадающего списка. Пользователь не должен вводить роль свободно, а выбирать из существующих ролей.

- authentication – представляет из себя форму для ввода логина и пароля.
- roles - страница для просмотра ролей и их ID, должна быть доступна только для администратора.

В пакете static.api представлена страница приветствия “welcome” – общая страница для всех пользователей, с которой есть кнопки регистрации и авторизации.

Пакет “questionnaire” включает в себя страницы:

- all-questionnaires – страница, отображающая все методики в системе, должна быть доступна учителям и администраторам.

- my-questionnaires – страница, на которую есть выход из личного кабинета учителя или ученика, показывает методики, которые имеются в распоряжении: у учителя – это методики, которые он создал и добавил; у ученика – это методики, которые он прошёл или должен пройти.

- questionnaireProfile

Пакет “student” включает в себя страницы:

- all-students – показывает список всех учеников, доступных в приложении
- my-students – показывает список учеников, связанных с учителем, доступен из личного кабинета учителя.
- studentProfile – кабинет профиля ученика, где он видит свою информацию, может искать учеников, отправлять заявку учителю, просматривать пройденные и не пройденные методики

Пакет “subject” включает в себя страницы:

- all-subjects – показывает список предметов, доступных в системе.
- my-subjects – список предметов, видный из профиля учителя, должен быть связан непосредственно с этим учителем.

Пакет “teacher” включает в себя страницы:

- all-teachers – список всех учителей, присутствующих в системе.
- my-teachers – список учителей, видных через кабинет профиля ученика.
- teacherProfile – страница профиля учителя, через которую он видит свою информацию и осуществляет свой функционал.

ВАЖНОЕ ЗАМЕЧАНИЕ: пользовательский интерфейс не проработан до конца, то есть до того момента, когда им можно пользоваться без сторонних приложений и без обращения к разработке. Так как наша задача не состояла в том, чтобы сделать широкий и всеобъемлющий интерфейс, мы его сделали только в том объёме, который необходим для решения наших задач разработки, тестирования и отладки.

2.3 Проектирование базы данных

Для проектирования базы данных используется СУБД PostgreSQL 16.2. Мы создали базу данных благодаря фреймворку Spring Data Jpa, поэтому специальные запросы написаны не были, а создание таблиц происходило через создание сущностей. В нашем приложении существуют такие сущности:

- questionnaire – методика приложения (в смысле учебная методика для проверки знаний)
- user – пользователь приложения
- teacher – пользователь приложения в роли учителя
- student – пользователь приложения в роли ученика
- subject – предмет (в смысле – учебная дисциплина).
- role – роль пользователя в приложении.

Ниже на рис.1 представлена диаграмма базы данных нашего приложения. Давайте опишем основные таблицы и связи между ними.

Таблица “users” служит для хранения сущности “User”:

- id – идентификатор пользователя, является первичным ключом,
- username – имя пользователя, поле является уникальным, чтобы идентифицировать пользователя в системе.
- password – пароль пользователя, хранится в том виде, в каком пользователь его передал через форму приложения. **НУЖНО ДОРАБОТАТЬ:** при рефакторинге и подготовке пароль необходимо кодировать в целях безопасности.
- role_id – роль пользователя, внешний ключ, указывающий на таблицу “roles”.

Таблица “roles” хранит список ролей.

- На неё ссылается таблица “users” по соотношению один-ко многим: одна роль может быть у многих юзеров, но каждый юзер имеет только одну роль.
- id – идентификатор роли, первичный ключ;
- name – имя роли, уникальное поле.

Таблица “teachers” хранит данные о пользователях роли «Учитель».

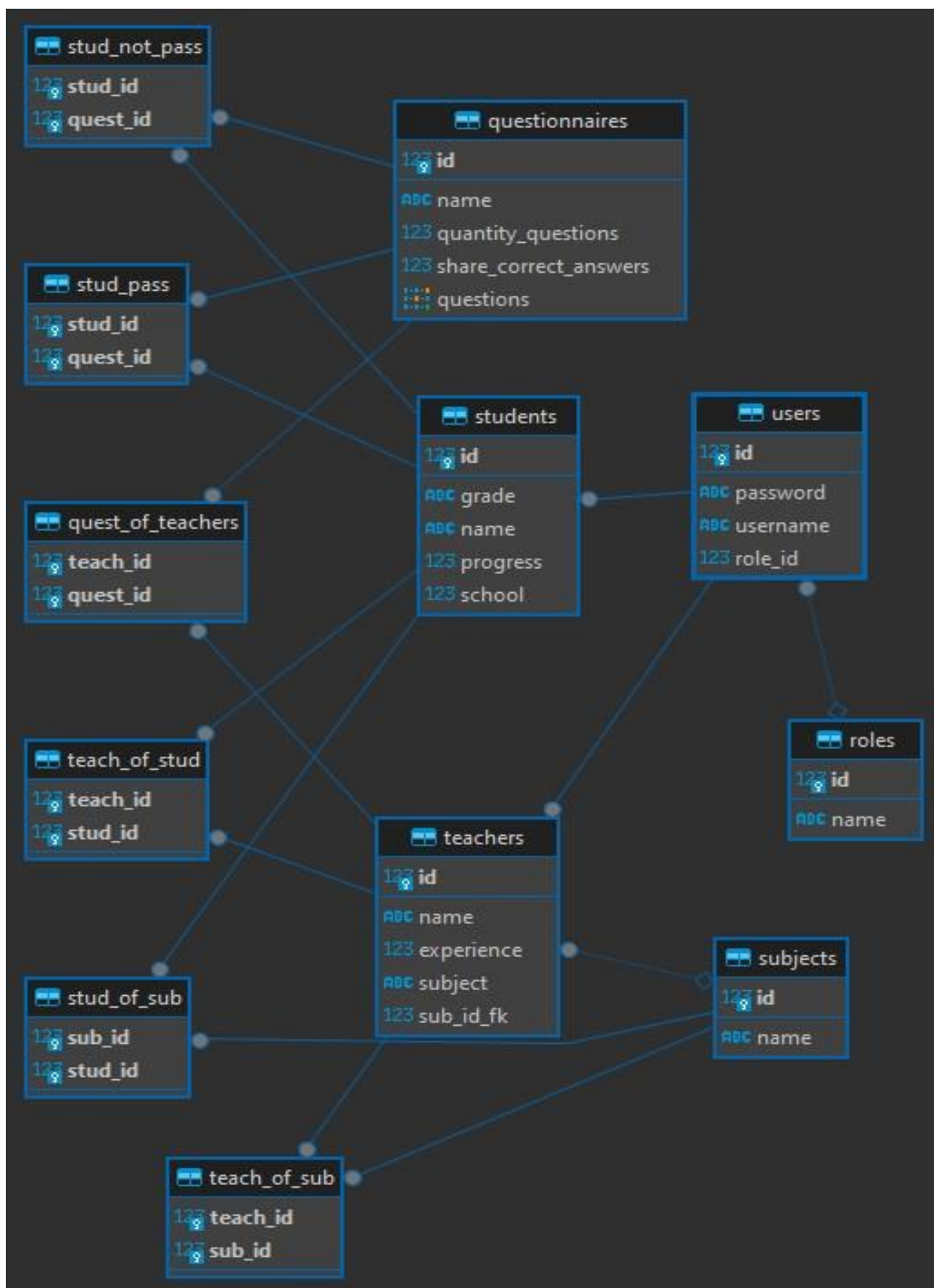


Рис.1. Диаграмма базы данных приложения “cognition”

- таблица связана с таблицей “users” соотношением один-к-одному: один пользователь может быть одним учителем, один учитель может быть одним пользователем.
- Сущность “teacher” является наследником “user”;
- id – внешний ключ, ссылается на id пользователя таблицы “users”;
- name – имя учителя, дублирует имя пользователя с ролью «учитель»: но данное поле можно поменять, при этом имя учителя не поменяется.
- experience – опыт учителя, измеряется в целых числах.
- Subject – основной предмет, который учитель ведёт;
- subject_id_fk – внешний ключ, ссылается на таблицу “subjects” по соотношению многие-ко-многим: один учитель может вести множество предметов, один предмет ведёт множество учителей.

Таблица “students” хранит данные о пользователях роли «Ученик». Ученик является наследником пользователя. При создании пользователя с ролью «ученик», ученик создаётся автоматически.

- id – внешний ключ, ссылается на таблицу “users” по принципу один-к-одному: один студент может быть только одним пользователем, один пользователь может быть только одним студентом.
- grade – класс, в который входит ученик.
- name – имя ученика, дублирует имя пользователя
- progress – средний балл, показывающий успеваемость ученика.
- school – номер школы, в которую входит ученик. **НУЖНО ДОРАБОТАТЬ:** необходимо будет создать ещё одну сущность ‘school’ – для хранения данных о школах.

Таблица “subjects” хранит данные о предметах.

- id – идентификатор предмета,
- name – название предмета, уникальное поле.

Таблица “questionnaires” хранит данные о сущности «Методика».

- id – идентификатор методики, первичный ключ.

- name – имя или название методики.
- quantity_questions – количество вопросов
- share_correct_answer – количество правильных ответов, которые необходимы для того, чтобы статус методик у студента был обозначен как пройденный.
- questions – вопросы методики. **НУЖНО ДОРАБОТАТЬ:** сейчас это поле представлено массивом строк, но в реальности необходимо создать конвертер, который сущность «Вопрос» преобразовывал в удобочитаемый формат базы данных.

Таблица **“teach_of_sub”** хранит данные о связи таблиц “teachers” и “subjects”.

- Связь таблиц – многие-ко-многим, так как один учитель может вести несколько предметов, один предмет может вести множество учителей.
- teach_id – внешний ключ, указывающий на первичный ключ учителя.
- sub_id – внешний ключ, указывающий на первичный ключ предмета.

Таблица **“stud_of_sub”** хранит данные о связях сущностей «Ученик» и «Предмет».

- связь многие-ко-многим, так как один студент может проходить множество предметов, один предмет может изучать множество студентов.
- sub_id – внешний ключ, указывающий на первичный ключ предмета.
- stud_id – внешний ключ, указывающий на первичный ключ студента.

Таблица **“teach_of_stud”** хранит данные о связях сущностей «Учитель» и «Ученик».

- связь многие-ко-многим, так как у одного учителя может быть множество учеников, и у одного ученика может быть множество учителей.
- teach_id – внешний ключ, указывающий на первичный ключ учителя.
- stud_id – внешний ключ, указывающий на первичный ключ студента.

Таблица **“quest_of_teachers”** хранит данные о связях сущностей «Методика» и «Учитель».

- связь многие-ко-многим, так как один учитель может пользоваться множеством методик, и одна методика может быть в распоряжении множества учителей.
- quest_id – внешний ключ, указывающий на первичный ключ методики.
- teach_id – внешний ключ, указывающий на первичный ключ учителя.

Таблица “stud_of_noPass” хранит данные о связях сущностей «Ученик» и «Методика».. Важное отличие – методики есть в очереди у студента, но они не пройдены.

- связь многие-ко-многим, так как у одного ученика может быть множество непройденных методик, так и одну не пройденную методику может хранить у себя множество студентов.
- quest_id – внешний ключ, указывающий на первичный ключ методики.
- stud_id – внешний ключ, указывающий на первичный ключ студента.

Таблица “stud_of_pass” хранит данные о связях сущностей «Ученик» и «Методика». Важное отличие – методики есть в очереди у студента, и они - пройдены.

- связь многие-ко-многим, так как у одного ученика может быть множество пройденных методик, так и одну пройденную методику может хранить у себя множество студентов.
- quest_id – внешний ключ, указывающий на первичный ключ методики.
- stud_id – внешний ключ, указывающий на первичный ключ студента.

Таким образом мы имеем 12 таблиц базы данных “cognitiondb”: 6 таблиц хранят данные о сущностях, 6 – о связях между сущностями.

2.4 API приложения

API нашего приложения включает в себя следующие контрольные точки:

Общие конечные точки:

- GET /welcome – страница приветствия.
- GET /reg - регистрация, получает страницу регистрации.
- POST /reg – отправляет на сервер через форму регистрации данные: имя, пароль и роль.
- GET /auth – показывает страницу с авторизацией.
- POST /auth -- отправляет запрос для подтверждения авторизации пользователя: имя и пароль.

Контрольные точки для сущности “Role”:

- GET /role – отображает список ролей
- POST /role – сохраняет роль в базу данных.

Контрольные точки для сущности “Student”

- GET /students/{studentId} – отображает страницу с профилем студента по его ID.
- PUT students/{studentId} - обновляет информацию о студенте по его ID.
- GET /students/{studentId} /my-teachers – получает список учителей студента.
- POST /students/{studentId} /all-teach/findByName – находит список учителей по имени.
- GET /students/{studentId}/all-quest/{questId} – просмотр студентом/учеником его методик.
- POST /students/{studentId} /all-quest/{questId} – прохождение методики учеником по ID методики.

Контрольные точки для сущности “Teacher”:

- GET /teachers/{teacherId} – показывает профиль учителя
- PUT /teachers/{teacherId} – обновляет учителя
- DELETE /teachers/{teacherId} – удаляет учителя
- GET /teachers/{teacherId}/subjects – показывает список всех предметов
- POST /teachers/{teacherId}/subjects – добавляет новый предмет в список
- PUT /teachers/{teacherId}/subjects/{subjectId} – добавляет предмет в профиль учителя
- GET /teachers/{teacherId}/my-subjects – отображает список всех предметов данного учителя.
- DELETE /teachers/{teacherId}/my-subjects/{subjectId} – удаляет выбранный по ID предмет из списка предметов данного учителя.

- GET /teachers/{teacherId}/my-quest – показывает список методик, добавленных в коллекцию учителя
- POST /teachers/{teacherId}/my-quest – создаёт новую методику и добавляет её в общую базу данных.
- PUT /teachers/{teacherId}/my-quest/{questId} – обновляет методику, добавленную в коллекцию данного учителя
- DELETE /teachers/{teacherId}/my-quest – удаляет данную методику из списка учителя.
- GET /teachers/{teacherId}/all-quest – показывает все методики из базы данных.
- GET /teachers/{teacherId}/all-quest/{questId} – просмотр методики учителем по её ID.
- PUT /teachers/{teacherId}/all-quest/{questId} – добавляет методику по её ID в коллекцию учителя по его ID.
- GET /teachers/{teacherId}/all-quest/find/{questName} – ищет методики по их названию
- GET /teachers/{teacherId}/my-students – показывает список учеников в коллекции учителя по его ID.
- GET /teachers/{teacherId}/my-students/{studentId} – позволяет просматривать профиль ученика по его ID.
- POST /teachers/{teacherId} /my-quest/{questId}/{studentId} – отправляет ученику методику для прохождения. Методика и ученик определяются по их ID.
- DELETE /teachers/{teacherId} /my-students/{studentId} – удаляет ученика по его ID из коллекции учителя.
- GET /teachers/{teacherId}/all-students – показывает список всех учеников из базы данных.
- POST /teachers/{teacherId}/all-students/{name} – добавляет ученика по его ID в коллекцию учителя.

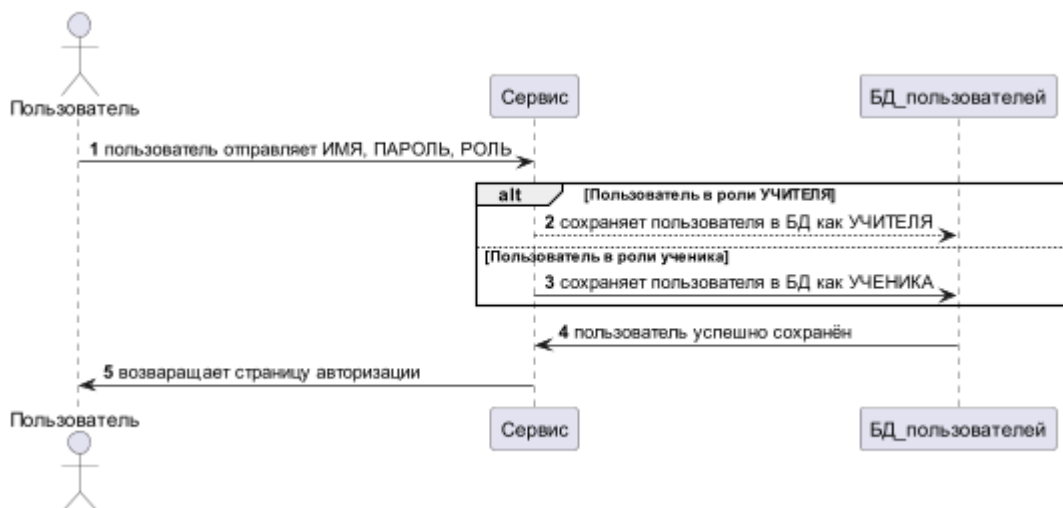
Таким образом, в своём приложении мы имеем 36 точек взаимодействия с сервером. Из них: 18 GET-запросов, 9 POST –запросов, 5 PUT-запросов, 4 DELETE-запроса.

2.5 Реализованный функционал приложения

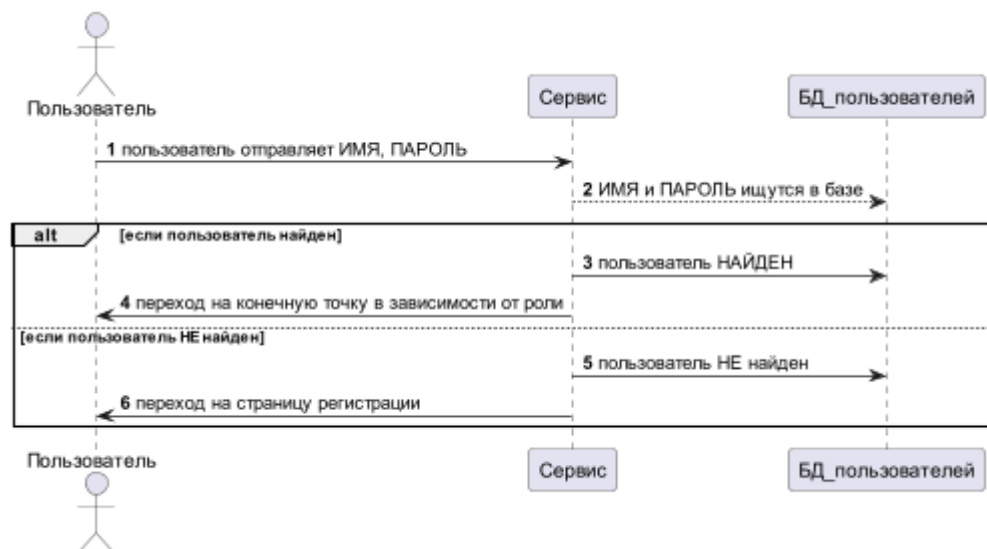
Основной функционал нашего приложения представлен use-case-диаграммами. Он наглядно показывает, какие функции реализованы в приложении, а также из каких модулей оно состоит.

Роль «Пользователь»

Ниже представлена процедура регистрации пользователя:

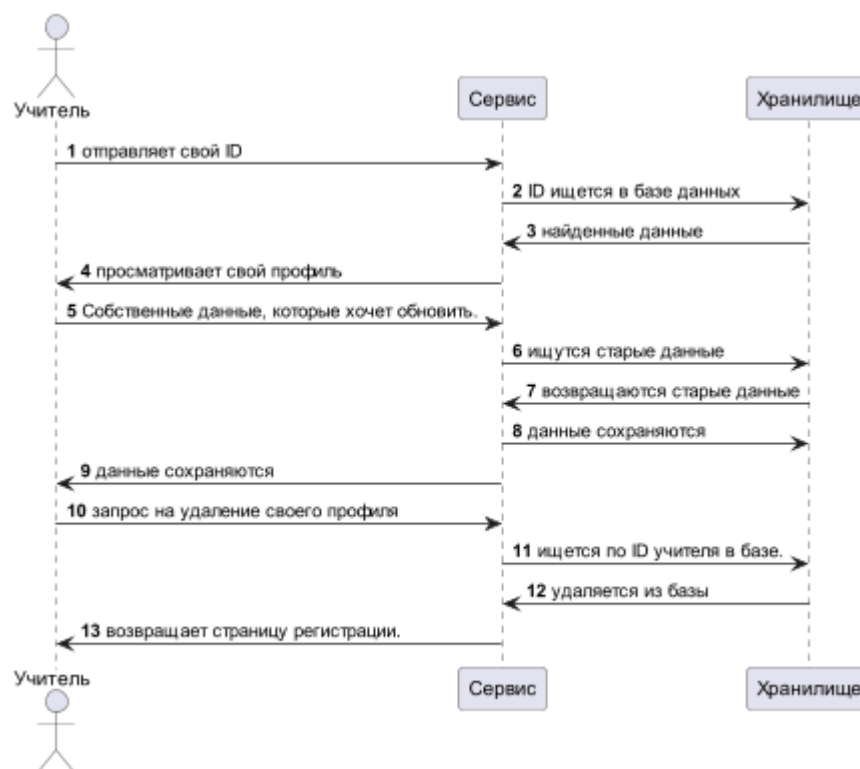


Ниже представлена процедура авторизации пользователя:

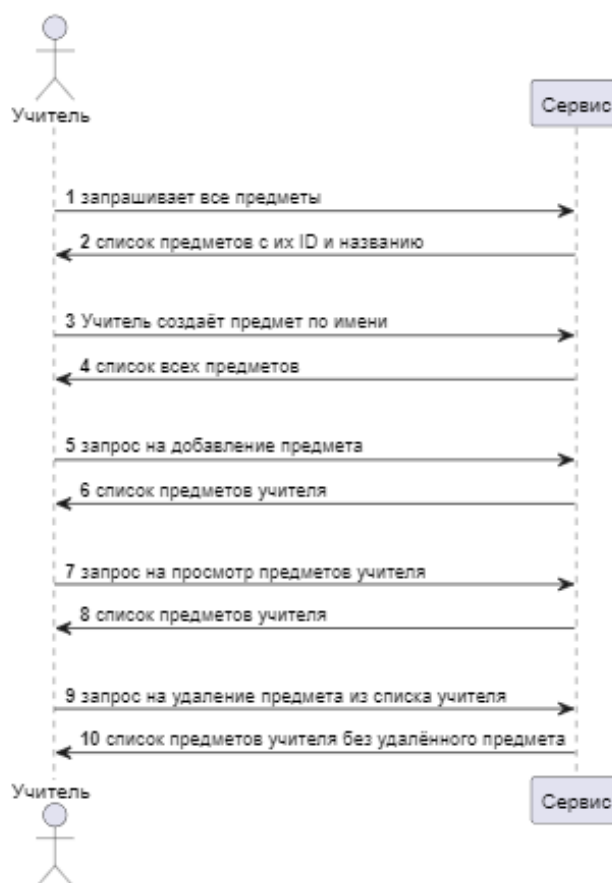


Роль «Учитель»

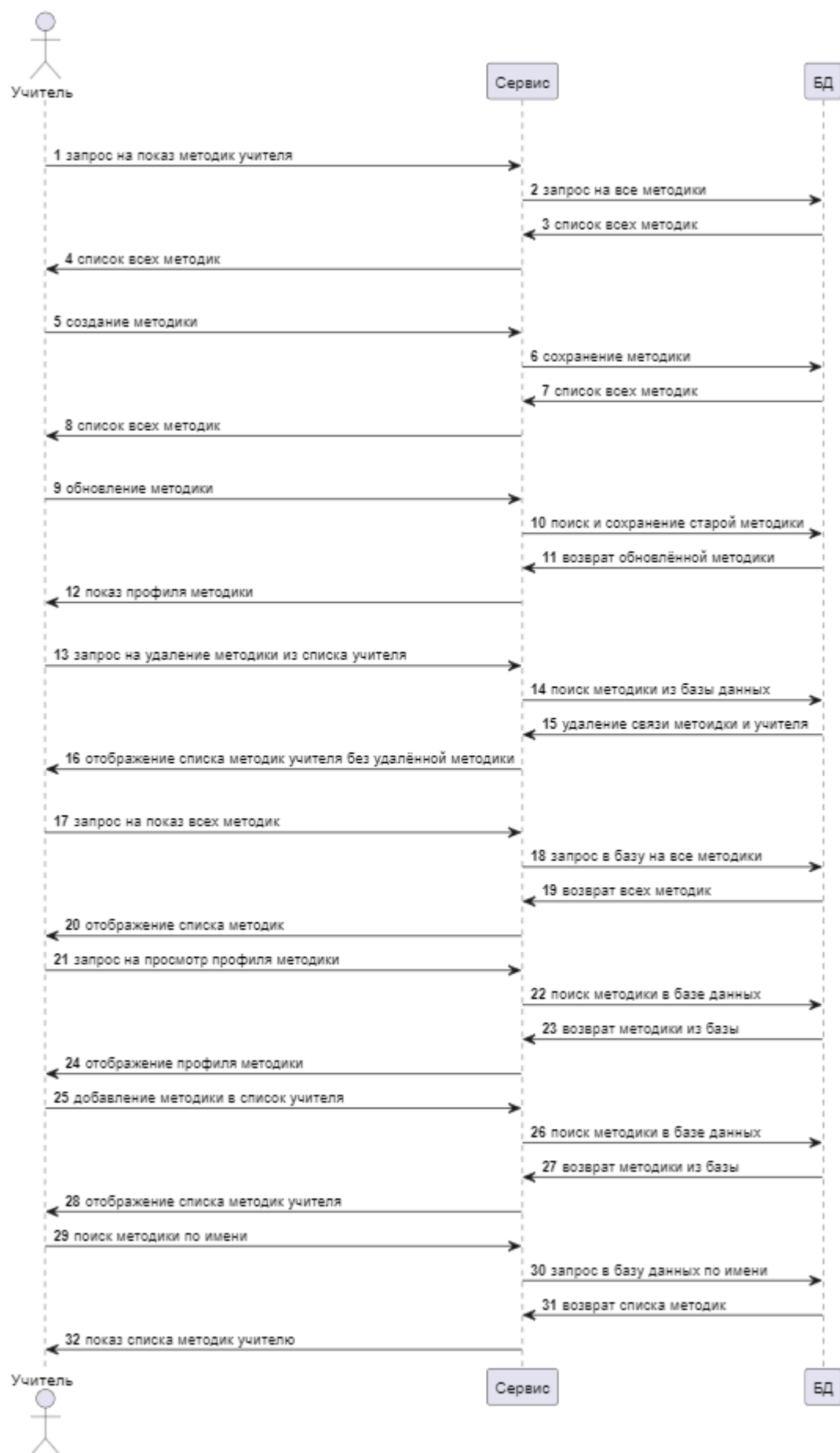
Ниже представлена процедура работы учителя со своим профилем:



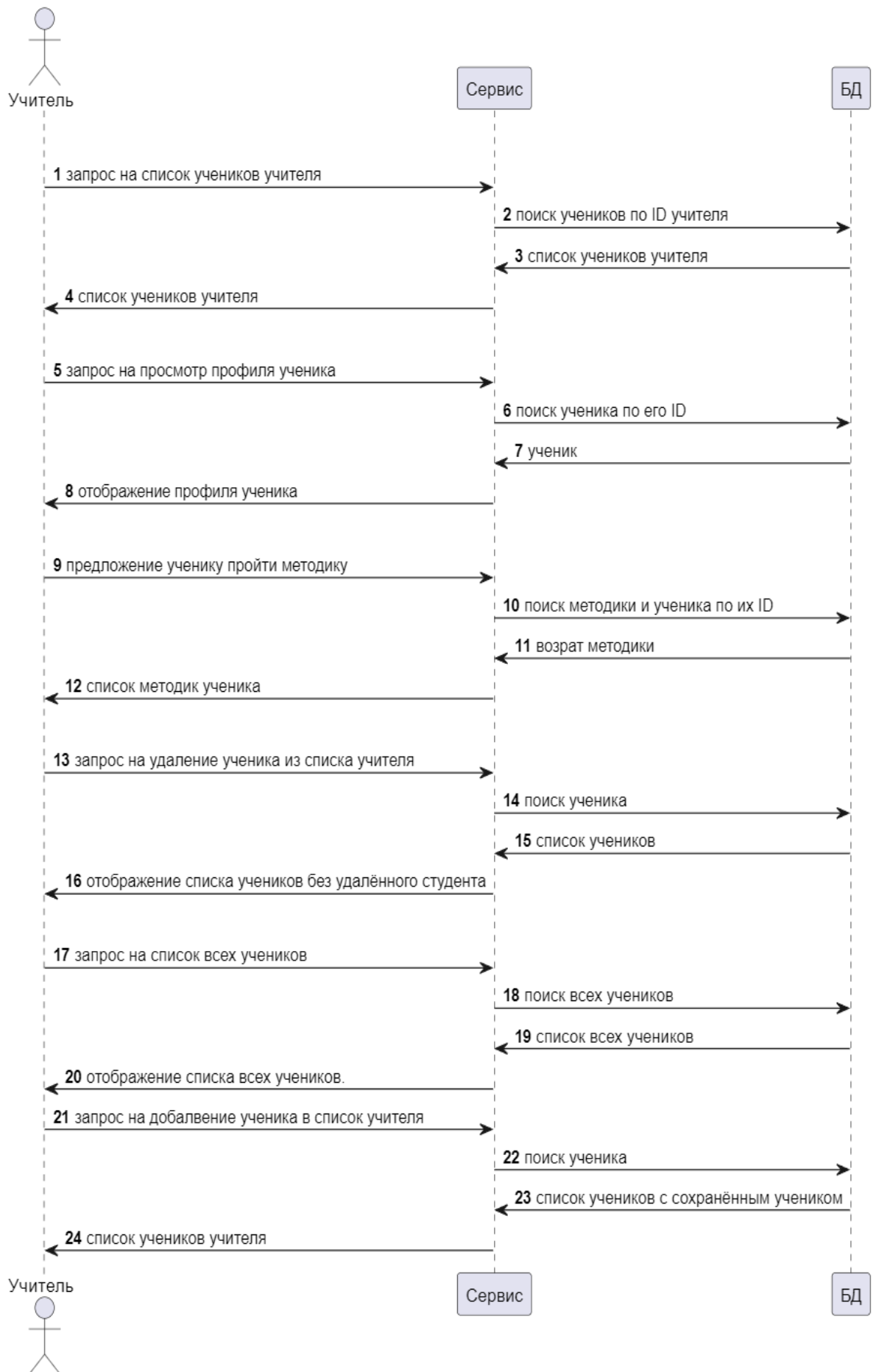
Ниже представлена процедура работы учителя с учебными предметами (слой базы данных опущен специально, но он подразумевается по умолчанию:



Ниже представлена процедура работы учителя с методиками:

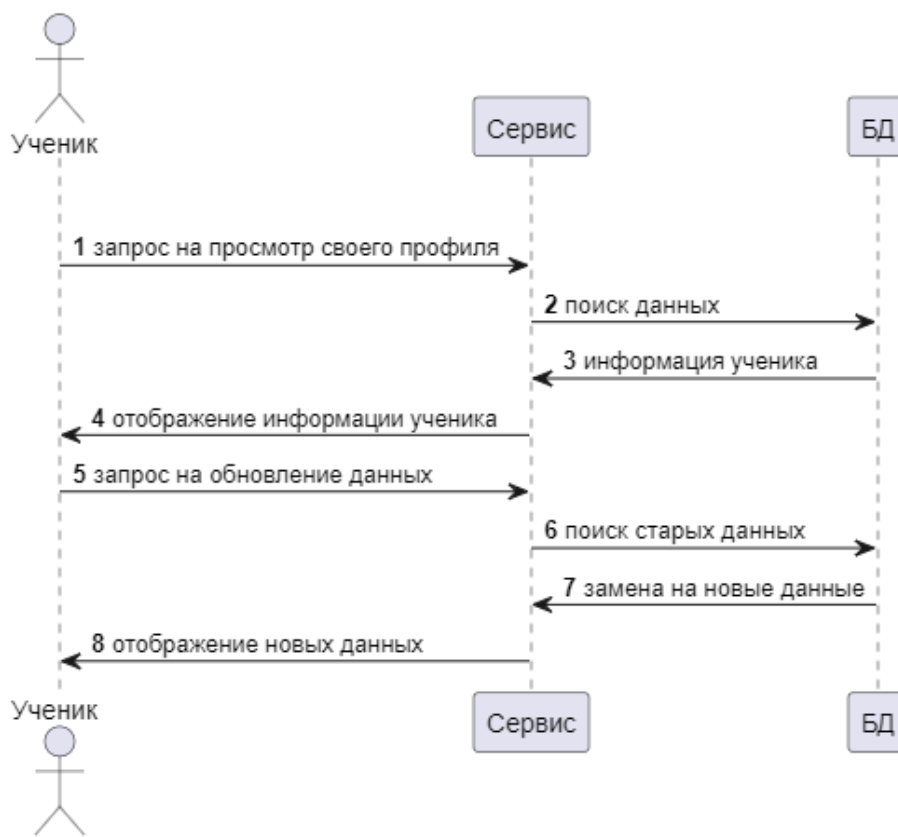


Ниже представлена процедура работы учителя с учениками:

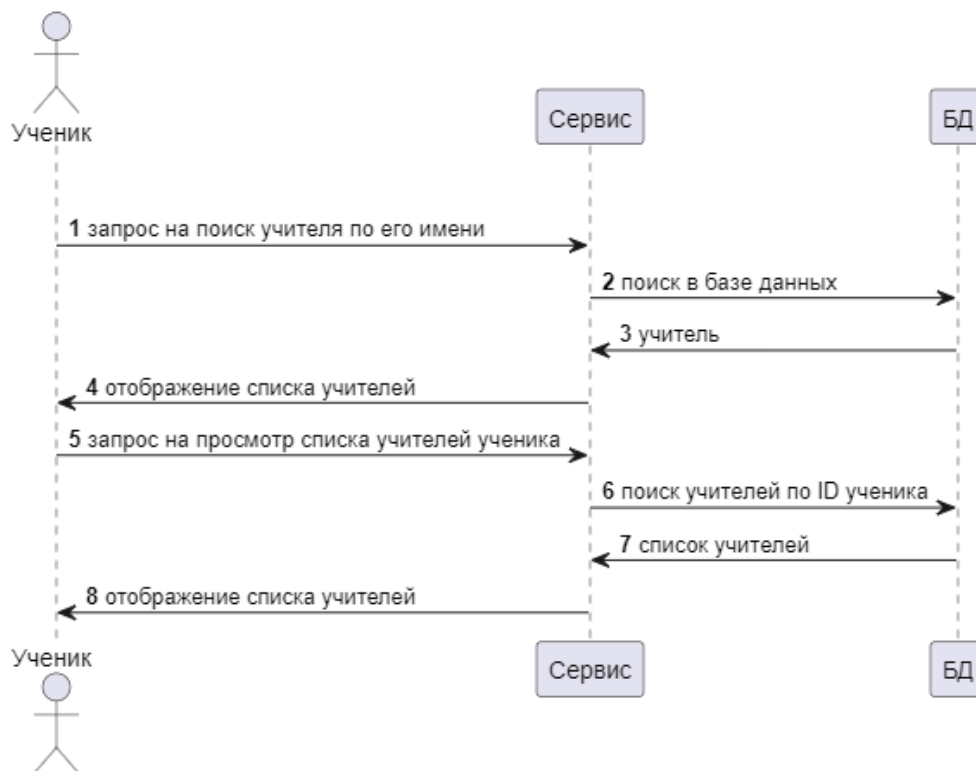


Роль «Ученик»

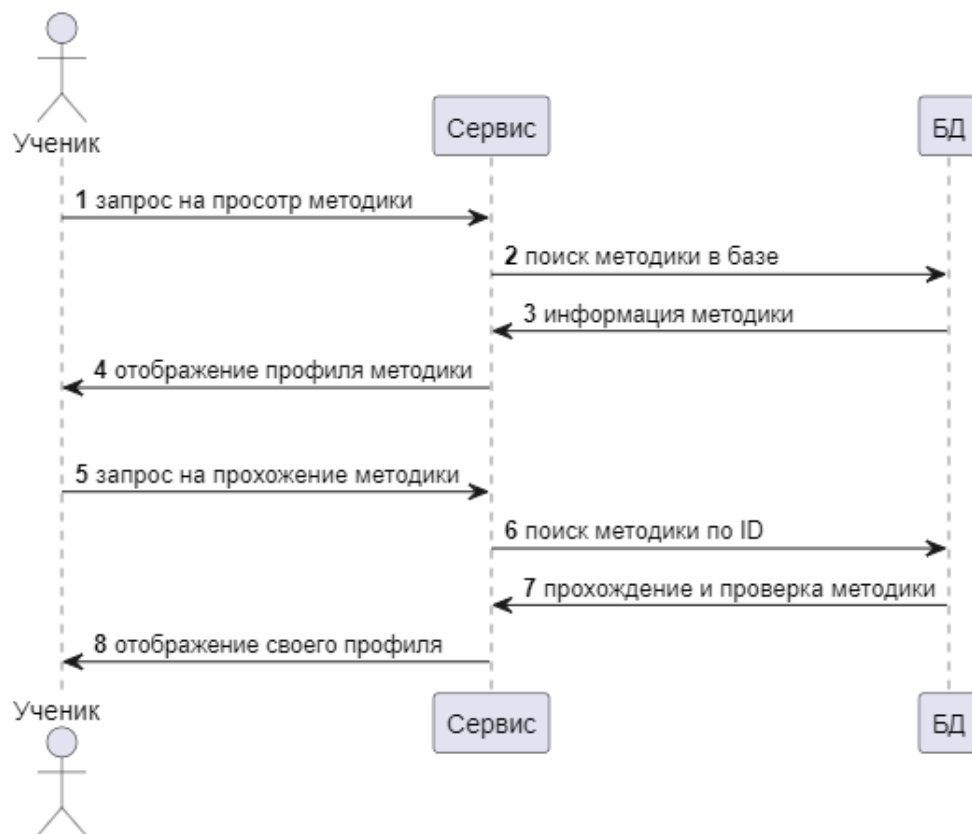
Ниже представлена процедура работы ученика с профилем:



Ниже представлена процедура работы ученика с учителями:



Ниже представлена процедура работы ученика с методиками:



Таким образом, в нашем проекте пользователь может пребывать в трёх ролях: как пользователь, как учитель, как ученик.

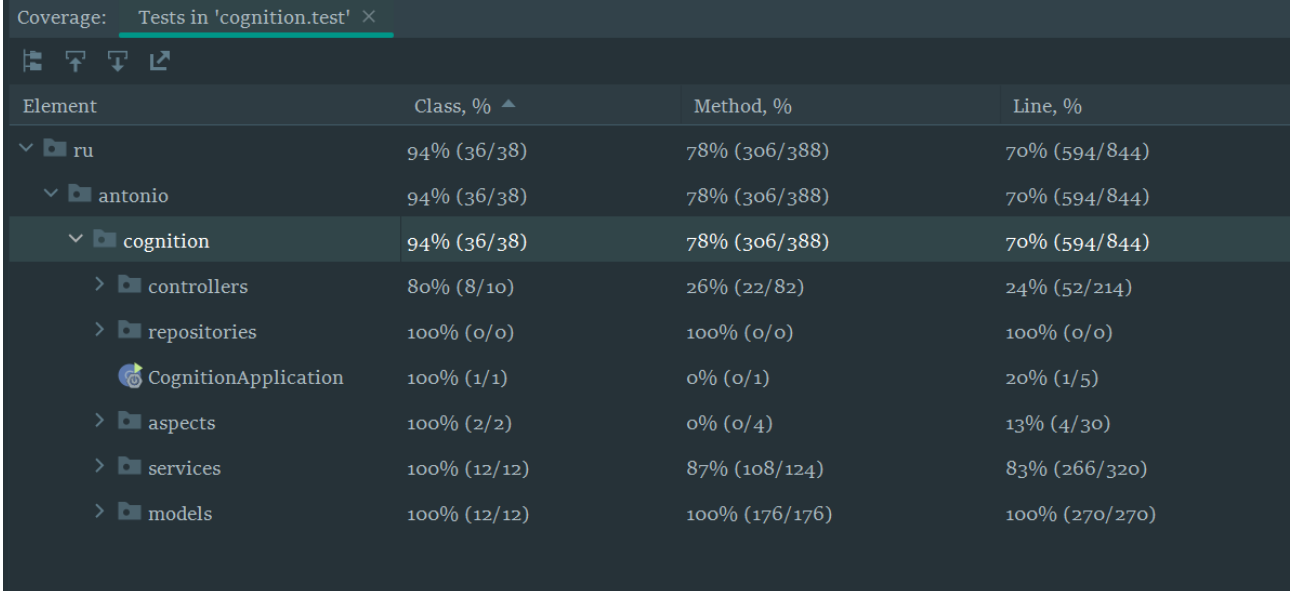
Как пользователь он может регистрироваться и авторизовываться. Необходимо также добавить роль «Администратор» и реализовать этот функционал через Spring Security. Самым большим функционалом пользователь приложения обладает в роли «Учителя». Он может создавать и добавлять предметы и методики, может просматривать их в базе данных, иметь и редактировать свои списки методик и предметов, править методики, просматривать их профили, смотреть список студентов, редактировать собственный список студентов, отправлять студенту методики для прохождения. Пользователь в роли «Ученик» может просматривать и проходить методики, просматривать и редактировать свой профиль, просматривать список учителей и искать их по имени.

В дальнейшем планируется расширение функционала, о чём мы подробно распишем в следующей главе.

2.6 Модульное и интеграционное тестирование

Тестирование приложения – неотъемлемый компонент любой разработки. Самая общая классификация предполагает такие виды тестирования как: модульное, интеграционное, сквозное.

Мы покрыли наше приложение тестами, используя библиотеки из фреймворка Spring Boot Test. По тестам у нас происходит следующая картина:



Coverage: Tests in 'cognition.test' ×			
Element	Class, % ▲	Method, %	Line, %
▼ ru	94% (36/38)	78% (306/388)	70% (594/844)
▼ antonio	94% (36/38)	78% (306/388)	70% (594/844)
▼ cognition	94% (36/38)	78% (306/388)	70% (594/844)
> controllers	80% (8/10)	26% (22/82)	24% (52/214)
> repositories	100% (0/0)	100% (0/0)	100% (0/0)
CognitionApplication	100% (1/1)	0% (0/1)	20% (1/5)
> aspects	100% (2/2)	0% (0/4)	13% (4/30)
> services	100% (12/12)	87% (108/124)	83% (266/320)
> models	100% (12/12)	100% (176/176)	100% (270/270)

Рис.2.6-1. Таблица покрытия кода тестами, распределённая по пакетам приложения.

Из данного тестирования видно, что, во-первых, все написанные тесты были пройдены успешно. Это значит ожидаемое поведение методов совпало с действительным. Во-вторых, мы покрыли свой код на уровне классов на 94%, на уровне методов – на 78%, на уровне строк – 70%, что является неплохим, но недостаточным результатом. Из таблицы видно, что самым мало покрытым пакетом является пакет контроллеров. Это связано с тем, что из-за недостатка времени и нехватки знаний мы не смогли написать тесты для контроллеров с аннотацией `@Controller`, использующих «модель» в качестве аргумента метода.

Помимо этого, мы не стали тестировать служебный пакет «aspects», так как он содержит всего 2 метода и одну аннотацию.

На данном этапе разработки объём покрытия кода тестами нас удовлетворяет. В дальнейшем необходимо учесть недочёты тестирования и довести его до допустимого для продакшена уровня.

2.7 Сквозное тестирование

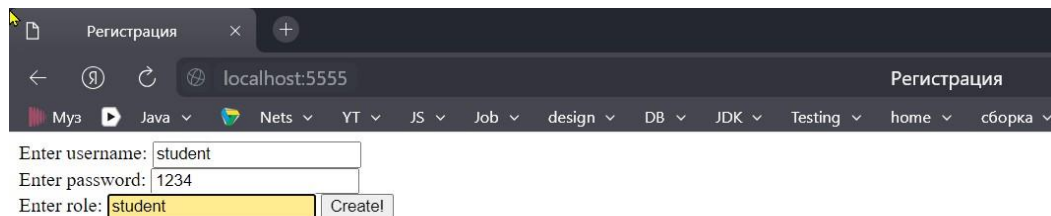
Давайте теперь проведём сквозное тестирование и посмотрим, как работает наше приложение. Сквозное тестирование предполагает проверку функционала приложения по пользовательскому сценарию. Сразу оговоримся, что мы не будем проверять весь функционал, а посмотрим работу по следующему сценарию, на каждом этапе которого мы должны получить определённый ожидаемый результат. Сценарий представлен ниже в таблице:

№	Поведение, которое будем проверять	Ожидаемые результаты
1	Регистрация пользователя. Зарегистрируем пользователя как ученика	Мы должны ввести данные: имя, пароль и роль. Эти данные должны сохраниться в базе данных в таблице пользователь и имя должно продублироваться в таблице ученик, где должен появиться внешний ключ на таблицу пользователей. В таблице пользователей должен появиться внешний ключ на таблицу ролей. Нас должно перебросить на страницу авторизации.
2	Авторизация. Отправляем корректные имя и пароль.	Нас должно перебросить на страницу с профилем ученика, на ней мы будем видеть свои данные. Так как ученик – новый, таблицы с методиками должны быть пустыми, так как только учитель может отправить ученику методику на прохождение.

3	Авторизуемся учителем, который есть в базе данных и создадим новую методику и попробуем её отправить студенту. Проверим существует ли методика, перейдём на список всех методик	После авторизации нас должно перебросить в профиль учителя. Методика должна создаваться и появляться в базе данных, она должна присутствовать в списке всех методик.
4	Добавим методику в список методик учителя	В списке методик должна появиться методика, а базе данных связь между этой методикой и учителем.
5	Отправим ученику методику для прохождения от имени учителя, находящегося в базе данных (такие учителя есть! В процессе разработки они уже были добавлены для отладки работы приложения)	У учителя статус ответа должен быть – 200, а у ученика в не пройденных методиках должна появиться методика, которую нужно пройти. А в базе данных образоваться связь между непройденной методикой и студентом.
6	От ученика заполним ответы методики, корректные и не корректные и отправим их на проверку.	При некорректном заполнении методики, должно быть брошено исключение – вы не прошли методику, при корректном заполнении нас перебрасывает в профиль студента, и мы должны увидеть, что методика из списка не пройденных удалась, а в списке пройденных появилась. Мы также должны увидеть уничтожение связи в базе данных между студентом и методикой из таблицы непройденных методик, и занесении этой связи в пройденные методики.

Проведём наше сквозное тестирование по вышеуказанному сценарию, результаты представим в виде скриншотов работы приложения ниже.

Кейс 1. Процедура регистрации студента. Заходим в браузер «Яндекс», перед нами появляется форма регистрации, значит наш get-запрос отработал корректно. Зполняем форму регистрации студента: имя – student, пароль – 1234, роль – student.



Регистрация

localhost:5555

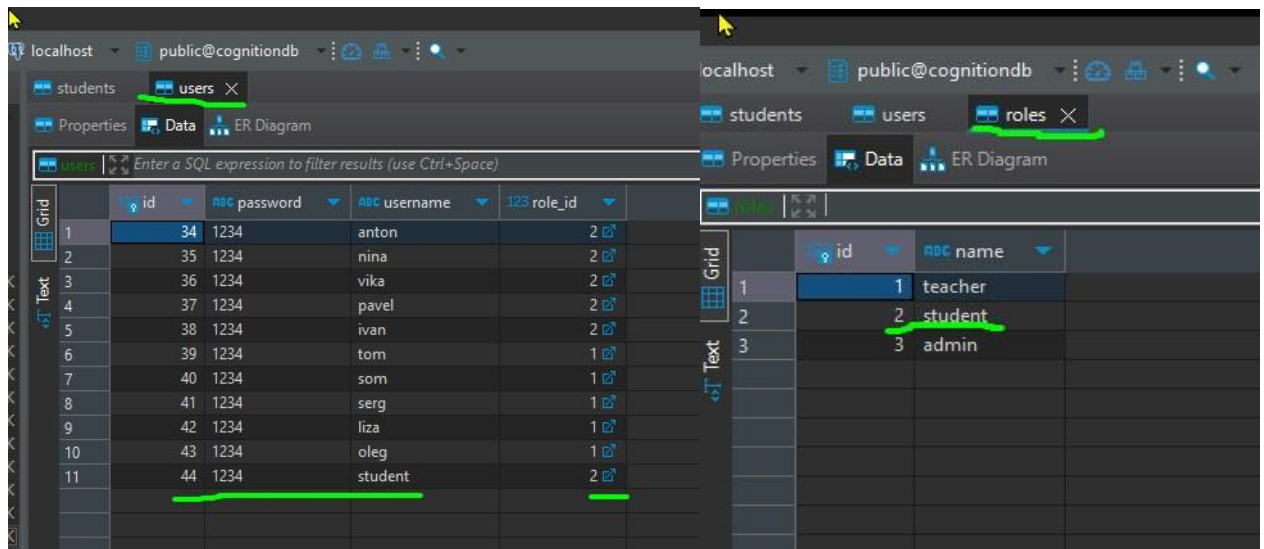
Регистрация

Enter username: student

Enter password: 1234

Enter role: student Create!

Здесь и далее будем просматривать базу данных через программу “Debeaver”. Видим на слайде слева, что наш пользователь успешно создан в таблице “users”. Все необходимые изменения на слайдах мы будем подчёркивать. Видим, что пользователю присвоен **ID – 44**, а также есть ссылка на таблицу ролей, где role_id = 2. Справа представлена таблица, под номером 2 роль “student”.

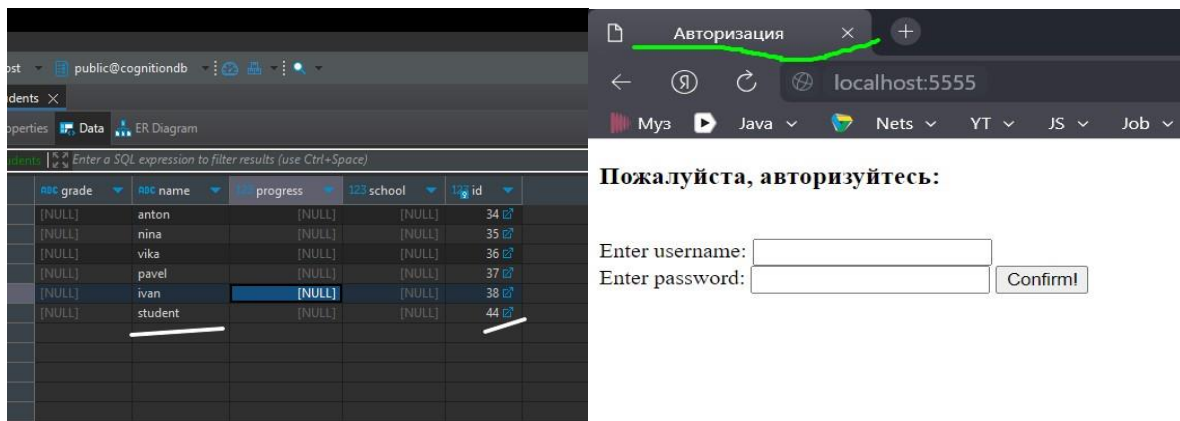


id	password	username	role_id
34	1234	anton	2
35	1234	nina	2
36	1234	vika	2
37	1234	pavel	2
38	1234	ivan	2
39	1234	tom	1
40	1234	som	1
41	1234	serg	1
42	1234	liza	1
43	1234	oleg	1
44	1234	student	2

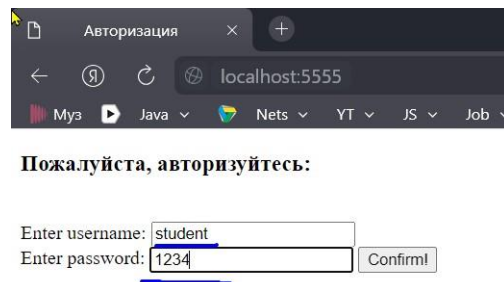
id	name
1	teacher
2	student
3	admin

Заходим в таблицу “students” и видим, что наш студент записался в неё. Имя “student” также продублировано и в эту таблицу. Справа видим, что браузер нас автоматически перекинул на вкладку «Авторизация».

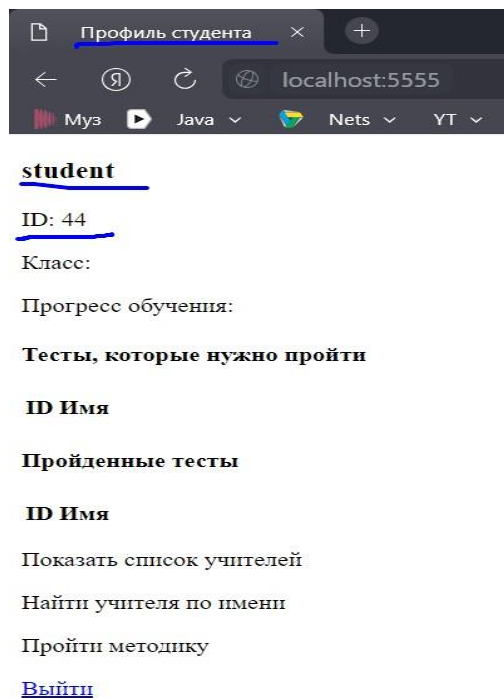
Результат: кейс выполнен успешно.



Кейс 2. Заполняем форму авторизации именем и паролем только что зарегистрированного студента.



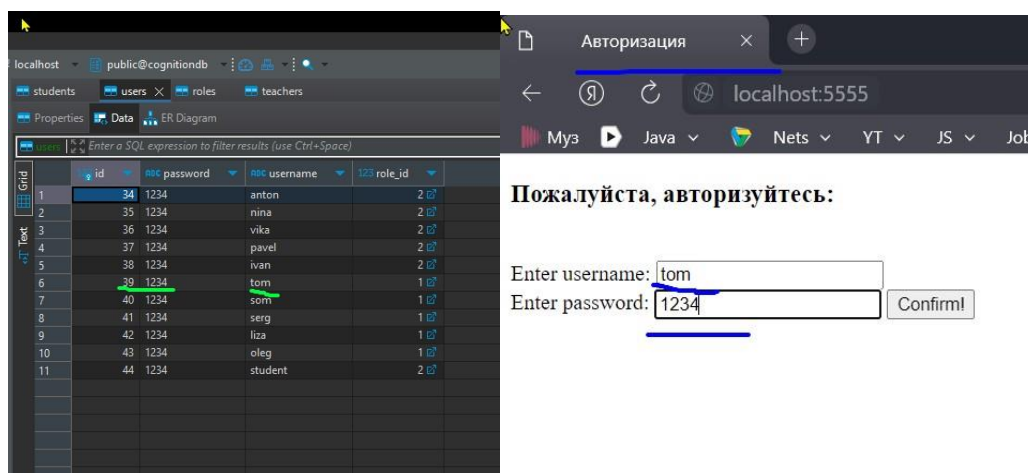
Нас благополучно перебрасывает на вкладку с профилем студента, где отображается его имя и ID.



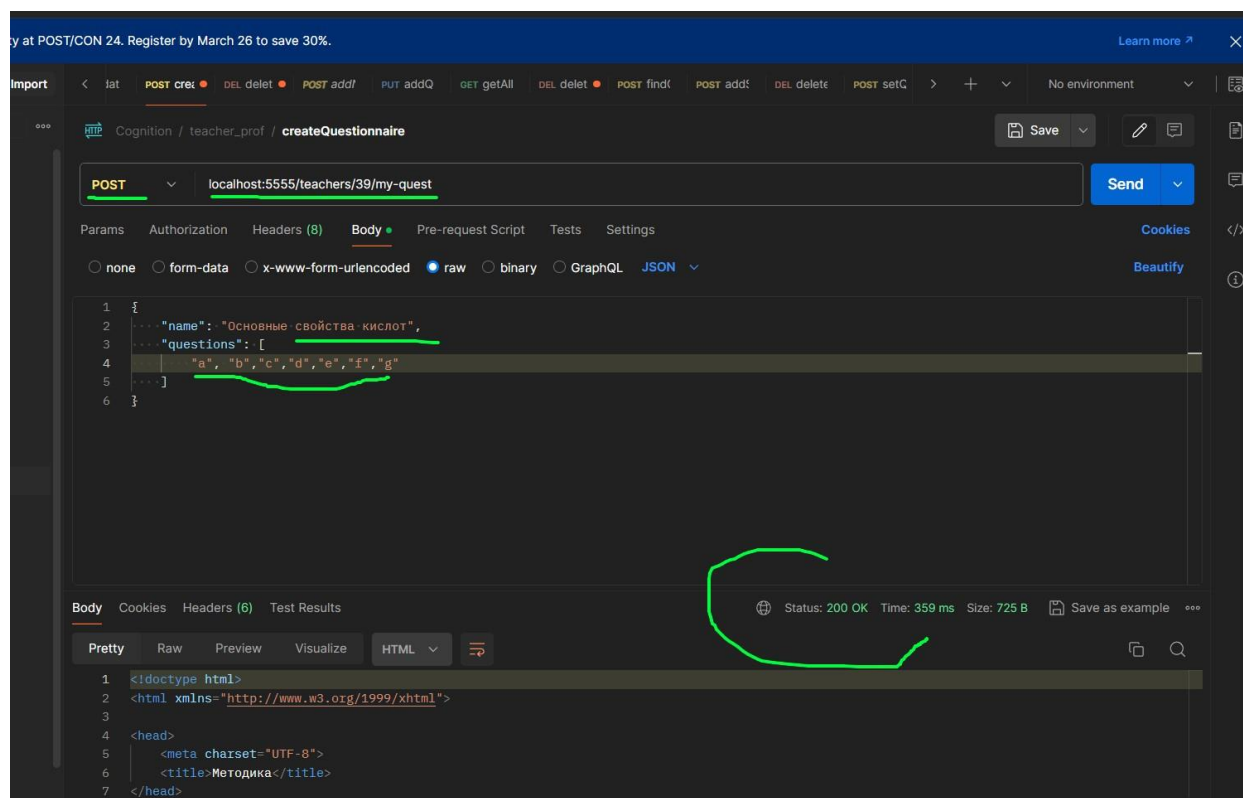
Обратите внимание, что таблицы – «Тесты, которые нужно пройти», и «Пройденные тесты» - пока пусты, так как не один учитель ещё не предложил ученику пройти методику.

Результат: кейс успешно выполнен.

Кейс 3. Пройдём авторизацию в качестве учителя по имени “tom”, с паролем – 1234, и его ID в базе данных – 39.



Авторизация успешно пройдена. Так как мы не доработали форму для заполнения методики, будем запросы писать через приложение Postman. Итак, отправим POST-запрос на создание методики. В адресе запроса пропишем id учителя – это 39. В теле запроса отправляем данные в json-формате: имя методики «основные свойства кислот», вопросы в массиве “a”, “b”, “c”, “d”, “e”, “f”, “g”. Отлично, сервер нам ответил кодом – 200.



Обратите внимание, что вопросы сделаны в формате массива строк. Это связано с тем, что мы настроились на самую логику приложения, а не на операции с вопросами. В дальнейшем, мы разработаем этот функционал более широко, чтобы можно было создавать вопросы в формате «вопрос».

Итак, наша методика появилась в базе данных (рисунок ниже) с id – 5.

The screenshot shows a database interface on the left and a browser window on the right. The database table 'questionnaires' has columns: id, name, quantity_questions, share_correct_answers, and questions. Row 5 is highlighted, showing id=5, name='Основные свойства кислот', quantity_questions=7, and share_correct_answers=3.5. The browser window shows the page 'Анкеты, тесты, опросники' with a list of questionnaires. The 5th item is 'Основные свойства кислот' with a href to '/questionnaires/5'. A green bracket connects the id=5 in the database to the href in the browser.

ID	Название
1	Кино
2	биология
3	история в биографиях
4	история на письмах
5	Основные свойства кислот

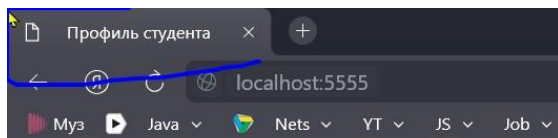
Результат: кейс успешно пройден.

Кейс 4. Добавим вновь созданную методику в коллекцию учителя, от имени которого мы действуем.

The screenshot shows a REST client interface. The request is a PUT to 'localhost:5555/teachers/39/all-question/5'. The response status is 200 OK. The response body is HTML, showing a table with the questionnaire details. A green bracket connects the id=5 in the URL to the href in the response body.

```

1 <!doctype html>
2 <html lang="ru">
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Анкеты, тесты, опросники</title>
7   <link rel="stylesheet" href="style.css" />
8 </head>
9
10 <body>
11   <h2>
12     <th>tom</th>
13   </h2>
14   <p>Ниже представлены ваши тесты, методики для проверки знаний учащихся</p>
15   <table>
16     <tr>
17       <th>ID</th>
18       <th>Название</th>
19       <th>Автор</th>
20     </tr>
21
22     <tr>
23       <th>5</th>
24       <th><a href="/questionnaires/5">Основные свойства кислот</a></th>
25     </tr>
26   </table>
  
```

student

ID: 44

Класс:

Прогресс обучения:

Тесты, которые нужно пройти

ID Имя
5 Основные свойства кислот

Пройденные тесты

ID Имя

Показать список учителей

Найти учителя по имени

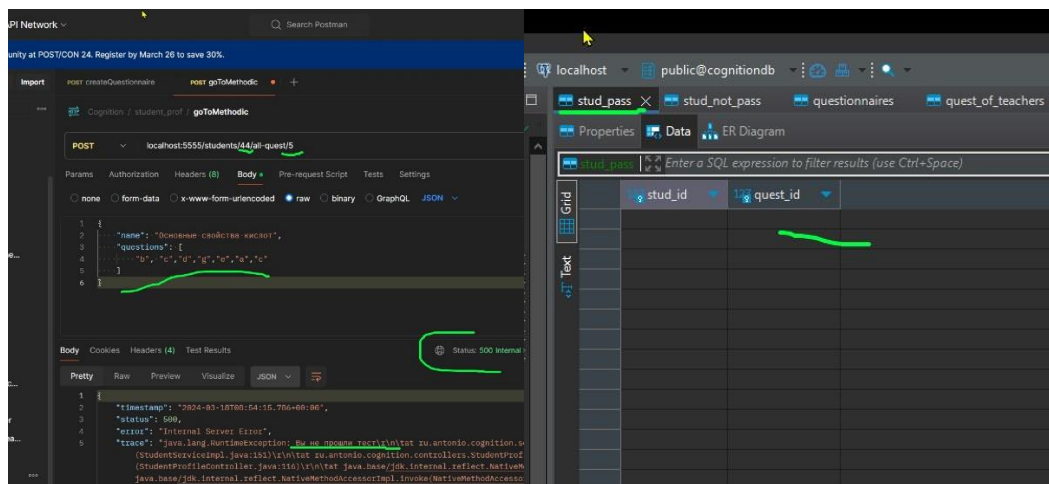
Пройти методику

[Выйти](#)

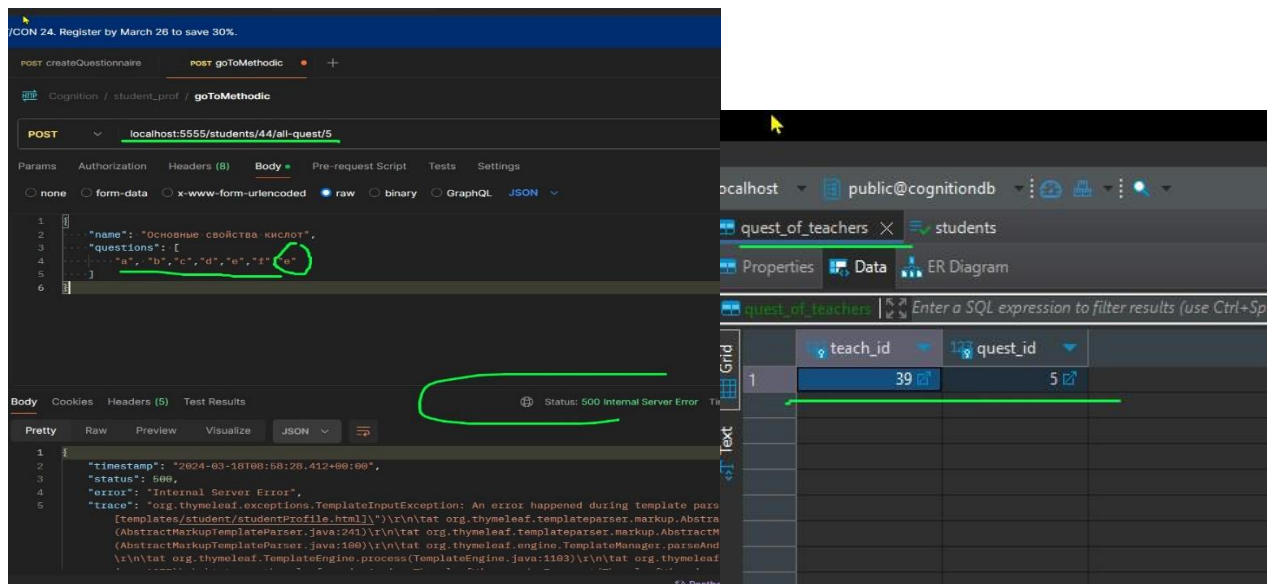
	stud_id	quest_id
1	36	3
2	36	1
3	35	1
4	44	5

Результат: кейс пройден успешно.

Кейс 6. От имени студента попробуем пройти данную методику. Специально введём некорректные ответы. Обратите внимание, что логика приложения работает на сравнение строк массивов вопросов, один из которых взят из базы данных, другой – отправляет студент. Их длина должна совпадать, так как это должно заполняться через форму. Также, в массиве ответов студента мы специально оставили один правильный ответ – “е” – пятый элемент.

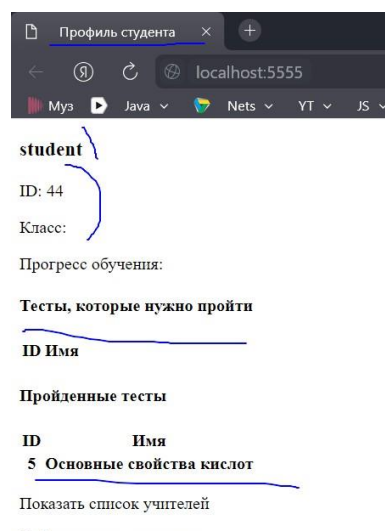


Но тем не менее, мы получили ошибку, что методика не пройдена: «вы не прошли тест». Давайте попробуем отправить корректные ответы.



И мы получаем ошибку сервера, но эта ошибка связана с отображением формы Thymeleaf, но не с логикой нашего приложения. Идём в базу данных (рис выше справа) и видим, что в таблице студент и пройденные методики образовалась связь между ними. Обратите внимание на рисунке выше слева мы специальность 7 элемент массива оставили неправильным, но методика всё равно стала пройденной. Это связано с тем, что у каждой методики есть поле «количество правильных ответов». Это поле должно устанавливаться учителем, но мы этот функционал пока не реализовали, потому сделали по умолчанию, что это поле будет равняться половине от количества вопросов в методики. Здесь 7 элементов = 7 вопросов, значит количество баллов для прохождения теста = 3.5. По умолчанию: каждый вопрос имеет вес в 1 балл. В дальнейшем будет разработан функционал, чтобы для каждого вопроса был свой вес баллов.

Посмотрим профиль студента и увидим, что у студента методика из таблицы «Тесты, которые нужно пройти» переместилась в таблицу «Пройденные тесты».



Результат: кейс успешно пройден, но есть проблемы с отображением успешного и неуспешного прохождения методики.

Таким образом, мы провели сквозное тестирование и показали работу нашего функционала, не обращая внимания на фронтенд-сторону проекта. В результате мы выявили некоторые недостатки работы системы, которые необходимо улучшить, нюансы и тонкости, которые нужно поправить и доработать.

2.8 Разработка предложений по улучшению и расширению функционала приложения

В ходе работы над приложением выявились аспекты, которые необходимо улучшить.

Первое. Проработать модуль регистрации и авторизации с помощью библиотеки Spring Security, в котором сообразно ролям будет подключение к тем или иным конечным точкам. Помимо, этого в целях безопасности необходимо кодировать пароль пользователя по крипто-ключу. Такие алгоритмы есть и Spring Security позволяет работать с этим функционалом. Пока что приложение является не безопасным.

Второе. Необходимо проработать отображение формы, более детально поработать с Thymeleav, чтобы можно было тестировать и пользоваться приложением, не прибегая к использованию приложения Postman. Необходимо разработать форму профиля, чтобы можно было из неё отправлять данные для изменения профиля; форму профиля методики, чтобы из неё можно было как создавать, так и проходить методики.

Третье. Необходимо улучшить работу с методиками, с сохранением вопросов. Изначально предполагалось, что за вопросы внутри методики будет отвечать отдельная сущность – Question, а за ответы внутри этой сущности – сущность Answer. Это бы позволило более гибко настраивать вопросы, приводить их разные типы: вопрос с одним ответом, со множеством ответов, последовательность ответов, соответствие, открытый вопрос и так далее. Сложности возникают с форматом хранения вопрос. Создавать для них отдельную таблицу нецелесообразно, ведь когда объёмы методик разрастутся, то обращение к базе данных с вопросами увеличится по времени. Пока нам не хватает знаний, чтобы доработать этот функционал.

Четвёртое. Необходимо проработать профиль администратора: добавить просмотр, добавление и удаление других пользователей, просмотр методик и предметов и т.д.

Пятое. Провести рефакторинг кода в соответствии с принципами SOLID, так как в коде часто нарушаются эти принципы.

Шестое. Провести модульное тестирование пакет контроллеров и пакет репозиториев.

Седьмое. Добавить следующие новые функции: учёт статистика прохождения тестов, количество учеников, методик и учителей. Для учителя добавить функционал авторства и просмотр статистики прохождения учениками тестов, распределив их по классам. Добавить сущности «Школа», чтобы можно было искать учеников и студентов по школе.

Восьмое. Добавить логирование для кода, обработать основные исключения, которые возникают в процессе функционирования приложения. Подключить для отслеживания статистики использования приложения метрики – Prometheus.

Девятое. Разбить своё приложение на микросервисы.

Это те основные шаги, которые необходимо сделать по улучшению приложения.

Заключение

Итак, мы теоретическую и практическую часть разработки серверной части мобильного приложения по проверке знаний обучающихся. В теоретической части мы обосновали, почему это приложение должно быть именно мобильным. Был проведён обзор основных приложений, существующих на рынке, были указаны их достоинства и недостатки. Также в теоретической части мы выяснили основные этапы разработки серверной части мобильного приложения.

В практической части мы описали небольшой дизайн нашего приложения, как вспомогательный инструмент для разработке, спроектировали базу данных, описали основное API, указали основные модули и их функционал. Также провели модульное, интеграционное тестирование, чтобы показать эффективность работы методов. Провели сквозное тестирование, чтобы продемонстрировать работу приложения с позиции конечного пользователя. Выявили и описали неработаннные стороны приложения, которые необходимо улучшить. Был составлен чек-лист по улучшению.

Считаем с поставленной целью мы справились: приложение для проверки знаний обучающихся было разработано, однако пока что приложение не готово пойти в продакшн из-за некоторых недоработок, которые необходимо исправить.

Список используемой литературы

1. Виды мобильных приложений: <https://appmaster.io/ru/blog/otliche-nativnyh-mobilnyh-prilozhenij-ot-vseh-ostalnyh>
2. Документация базы данных PostgreSQL: <https://www.postgresql.org/docs/>.
3. Использование библиотек под мобильную разработку: <https://apptractor.ru/info/articles/sovremennaya-android-razrabotka-v-2023-godu.html>
4. Количество пользователей мобильных устройств: <https://mindbox.ru/journal/experts/issledovanie-mobilnogo-trafika/>
5. Особенности разработки мобильных приложений: <https://livetyping.com/ru/blog/osobennosti-sozdaniya-android-prilozhenij>
6. Этапы создания мобильного приложения: <https://habr.com/ru/articles/726516/>
7. Серия статей сайтов:
 - 7.1. <https://www.baeldung.com/spring-boot>.
 - 7.2. <https://www.javaguides.net/>,
 - 7.3. <https://stackoverflow.com/>