*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
" Технологии разработки мобильных приложений "***

***на тему***

*Разработка мобильной игры кликера с возможностью покупки улучшений*

*Выполнил: Грачев Д. А.*

*Принял: ассистент кафедры*

*ИСПИ Петрова А.И.*

*Владимир, 2023*

**Аннотация**

В данном курсовом проекте производилось проектирование и разработка мобильной игры кликера с возможностью покупки улучшений. Проект состоит из 3 этапов, включающих в себя описание предметной области, моделирование структуры объектов предметной области и их взаимодействия на концептуальном уровне, реализацию программной системы, содержащую серверную часть системы и интерфейс пользователя.

Разработка мобильного приложения для курсового проекта велась на языке программирования JavaScript с использованием библиотеки React-Native. Для реализации авторизации и хранения данных использовалась платформа Firebase, в которой были подключены компоненты «Authentication» и «Cloud Firestore».

This course project involved the design and development of a mobile clicker game with the ability to purchase improvements. The project consists of 3 stages, including a description of the subject area, modeling the structure of objects in the subject area and their interaction at the conceptual level, implementation of a software system containing the server part of the system and the user interface.

The development of a mobile application for the course project was carried out in the JavaScript programming language using the React-Native library. To implement authorization and data storage, the Firebase platform was used, in which the “Authentication” and “Cloud Firestore” components were connected.

Курсовой проект представлен на 55 страницах, рисунков – 39, таблиц – 3, использованных источников – 7, приложений – 6.

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc137107920)

[1 Описание предметной области 5](#_Toc137107921)

[1.1. Описание предметной области 5](#_Toc137107922)

[1.2. Требования в системе 5](#_Toc137107923)

[1.3. Стек технологий 6](#_Toc137107924)

[1.4. Обзор аналогов 6](#_Toc137107925)

[1.5. Роли в разрабатываемой системы 7](#_Toc137107926)

[1.6. Словарь предметной области 7](#_Toc137107927)

[1.7. Сценарий взаимодействия пользователя с системой 8](#_Toc137107928)

[1.8. Начальная оценка и выделение сущностей 9](#_Toc137107929)

[1.9. Вывод 9](#_Toc137107930)

[2. проектирование системы 10](#_Toc137107931)

[2.1. Диаграмма прецендентов 10](#_Toc137107932)

[2.2. Спецификации прецендентов 10](#_Toc137107933)

[2.3. Диаграмма классов 13](#_Toc137107934)

[2.4. Диаграмма состояний 16](#_Toc137107935)

[2.5. Диаграмма последовательностей 17](#_Toc137107936)

[2.6. Диаграмма BPMN 19](#_Toc137107937)

[2.7. Диаграмма зон ответственности 19](#_Toc137107938)

[2.8. Диаграмма развертывания 20](#_Toc137107939)

[2.9. Схема навигации приложения 21](#_Toc137107940)

[2.10. Диаграмма взаимодействия 22](#_Toc137107941)

[2.11. Общие принципы организации системы 23](#_Toc137107942)

[3 Разработка системы 24](#_Toc137107943)

[3.1. База данных 24](#_Toc137107944)

[3.2. Интерфейс 25](#_Toc137107945)

[3.3. Серверная часть 33](#_Toc137107946)

[4 Тестирование 38](#_Toc137107947)

[4.1. Цель тестирования 38](#_Toc137107950)

[4.2. Методика испытаний 38](#_Toc137107951)

[4.3. Алгоритм тестирования 38](#_Toc137107952)

[4.4. План тестирования 38](#_Toc137107953)

[4.5. Спецификации системы 39](#_Toc137107954)

[4.6. Процесс тестирования 39](#_Toc137107955)

[4.7. Вывод 39](#_Toc137107956)

[заключение 41](#_Toc137107957)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](#_Toc137107958)

[ПРИЛОЖЕНИЕ А 43](#_Toc137107959)

[ПРиЛОЖЕНИЕ Б 46](#_Toc137107960)

[Приложение В 50](#_Toc137107961)

[пРИЛОЖЕНИЕ г 51](#_Toc137107962)

[пРИЛОЖЕНИЕ Д 52](#_Toc137107963)

# ВВЕДЕНИЕ

В ходе выполнения курсового проекта требуется спроектировать и разработать мобильную игру кликер с возможностью покупки улучшений.

Цель работы: изучение процесса разработки мобильных приложений на примере мобильной игры кликера с возможностью покупки улучшений.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать хранилище данных, в котором будет находиться информация о различных сущностях системы
2. Разработать интерфейс, с которым будет взаимодействовать пользователь

# Описание предметной области

* 1. Описание предметной области

В сфере развлечений для программистов необходимо создать мобильную игру-кликер, предназначенную для развлечения и отдыха. Данная игра обеспечивает:

* Возможность кратковременного ежедневного входа в игру без надобности постоянного контроля за происходящим в ней
* Возможность покупать себе помощников
* Авторизацию через Google
* Сохранение данных между устройствами и между перезапусками приложения
  1. Требования в системе

Функциональные требования:

* Войти в аккаунт
* Выйти из аккаунта
* Сохранить прогресс
* Загрузить прогресс
* Покупка помощников

Нефункциональные требования:

* Все возможности приложения должны быть доступны максимум в два нажатия
* Звуковое сопровождение игровых действий
* Использование Firebase в качестве хранилища данных
  1. Стек технологий
     1. JavaScript (ECMAScript2023)
     2. React-Native
     3. React
     4. MobX
     5. Metro
     6. Firebase
  2. Обзор аналогов

Система сайтов Владимирского государственного университета является аналогом разрабатываемой мной системы, но имеет некоторые недостатки:

* Различные сервисы и компоненты распределены по разным сайтам с отличающимися логинами и паролями, что усложняет навигацию и доступ к данным: существует 6 различных сайтов(сайт с личным кабинетов абитуриента, сайт с информацией о приемной комиссии, сайт с информацией об университете, сайт дистанционного образования кафедры, сайт дистанционного образования университета, сайт с личным кабинетом студента)
* Интерфейс многих сайтов из системы выглядит устаревшим и со сложным дизайном пользовательского опыта – требуется достаточно много действий для получения ожидаемого результата, многие блоки находятся не в интуитивно понятных местах
* Большая часть работ оценивается в ручном режиме, что замедляет скорость работы как студента, так и преподавателя
  1. Роли в разрабатываемой системы
     1. Администратор – сотрудник университета, который администрирует работу программной системы, назначает роли пользователям
     2. Пользователь – зарегистрированный участник системы, который может полностью взаимодействовать с подсистемой новостей и подавать документ на поступление в университет
     3. Студент – зарегистрированный участник системы, который владеет всеми возможностями обычного пользователя, так же может взаимодействовать с подсистемой курсов, в виде прохождения тестов
     4. Преподаватель – зарегистрированный участник системы, который обладает всеми возможностями обычного пользователя, так же может взаимодействовать с подсистемой курсов, в виде создания тестов и редактирования информации по ним
  2. Словарь предметной области
     1. *Курс* – часть программной системы, с помощью которой «Преподаватель» может создавать задания для «Студента», такие как контрольные работы
     2. *Новостная лента* – часть программной системы, обеспечивающая возможность написания новостей любым пользователем, так же любой прочитавший новость может поставить оценку прочитанному материалу и оставить комментарий
     3. *Рейтинговые списки* – часть программной системы, обеспечивающая автоматизацию работы с абитуриентами
     4. *Личный кабинет* – часть программной системы, внутри которой пользователь может увидеть данные о своем аккаунте, такие как количество очков, курсы, если это аккаунт студента или преподавателя
     5. *Студент* – роль пользователя в системе, являющегося студентом в университете, может выполнять тесты и лабораторные работы с курса
     6. *Преподаватель* – роль пользователя в системе, являющегося преподавателем в университете, может создавать и редактировать курсы
     7. *Тест* – часть курса, задание, в ходе которого студенты должны отвечать на поставленные вопросы с данными вариантами ответов
     8. *Группа* – выделенная часть студентов, объединённая одной специальностью
     9. *Обращение* – заполненная пользователем форма обратной связи
  3. Сценарий взаимодействия пользователя с системой

Взаимодействие с системой осуществляется посредством интернет-браузера.

Каждый пользователь имеет одну из нескольких ролей, которые определяют, какими возможностями он обладает. Можно выделить следующие роли и их возможности:

* Пользователь
  + Отредактировать данные аккаунта
  + Написать новость
  + Читать новости
  + Оставлять комментарии под новостями
  + Подать нужные документы для поступления
  + Отслеживать позиции в рейтинговых списках
* Студент
  + Проходить тесты
  + Отслеживать свой прогресс по курсам университета
* Преподаватель
  + Создавать курсы
  + Удалять курсы
  + Создавать тесты в курсах
  + Создавать лабораторные работы
  + Проверять и оценивать работы студентов
  + Добавлять студентов в курс
* Администратор
  + Присваивать роли новым пользователям
  + Удалять курсы

Также, каждый зарегистрированный пользователь может авторизоваться и аутентифицироваться в системе и выйти из своего аккаунта.

Незарегистрированный пользователь может просматривать новостную ленту университета, справочную информацию по специальностям, а так же зарегистрироваться

* 1. Начальная оценка и выделение сущностей

Для предметной области «Университет» были выделены следующие сущности: аккаунт, студент, группа, курс, тест, вопрос, ответ, обращение, сервисная информация

* 1. Вывод

Определена предметная область и цель разработки. Выявлены функциональные и нефункциональные требования, а также составлен стек технологий, на основе которых будет разрабатываться система

# проектирование системы

* 1. Диаграмма прецендентов

На данной диаграмме отображены все преценденты и ее актеры. Каждый прецендент имеет своего инициатора в виде актера. Некоторые отношения входят в отношение включения, например, чтобы создать курс преподавателю необходимо добавить к нему тесты. Актеры могут входить в отношения обобщения, например, актер «Студент» может выполнять все функции «Пользователя», но также имеет свои собственные функции.

На представленном ниже рисунке показана часть диаграммы прецендентов, показывающая подсистему «Авторизация» (рис. 1).

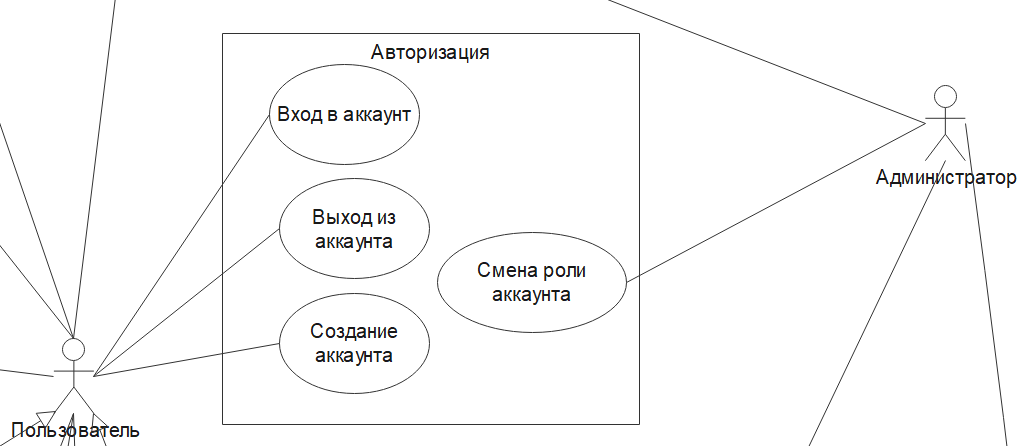


Рисунок 1. Диаграмма прецендентов - Подсистема "Авторизация"

Полная версия диаграммы прецендентов представлена в приложении В.

* 1. Спецификации прецендентов

Таблица 1. Спецификация прецендента "Создание аккаунта"

|  |  |
| --- | --- |
| Раздела спецификации прецендента | Описание |
| Название | Создание аккаунта |
| Актер | Не зарегистрированный пользователь |
| Предусловия | Пользователь заходит на сайт, нажимает кнопку войти и переходит на вкладку регистрации |
| Основной поток | Пользователь заполняет форму регистрации, состоящую из шести полей: логин, пароль, электронная почта, имя, фамилия, отчество. При нажатии на кнопку «Зарегистрироваться» из браузера отправляется два запроса на наличие в системе пользователя с таким логином и такой почтой, если оба запроса возвращают отрицательный ответ, значит такого пользователя нет в системе, значит можно создать, отправляется еще один запрос на создание пользователя, ему присваивается дата регистрации равная сегодняшней дате и времени, а также в поле роли устанавливается “user”, после этого новый пользователь может войти в аккаунт и пользоваться веб-приложением |
| Альтернативный поток | В случае, если в системе уже оказался пользователь с таким логином или почтой, незарегистрированному пользователю будет предложено сменить логин или почту, чтобы пройти регистрацию |
| Постусловие | После регистрации нового пользователя его данные, такие как логин и дата регистрации отправляются в панель администратора, чтобы тот мог присвоить аккаунту нужную роль, после чего в профиле пользователя будет отображаться новая информация в зависимости от роли |

Таблица 2. Спецификация прецендента "Написание статьи"

|  |  |
| --- | --- |
| Раздел спецификации прецендента | Описание |
| Название | Написание статьи |
| Актер | Пользователь |
| Предусловие | Пользователь вошел в свой аккаунт, перешел на страницу с новостями и нажал кнопку «Написать статью» |
| Основной поток | Пользователь заполняет форму, состоящую из пяти полей: Название статьи, фон статьи, заглавное изображение, текст статьи, дополнительные файл. При нажатии на кнопку «Опубликовать», происходит валидация данных формы и отправка статьи |
| Альтернативный поток | В случае возникновения какой-либо ошибки во время валидации формы или при попытке ее отправить, система уведомляет о ней пользователя, с просьбой исправить проблемы(если это возможно) |
| Постусловие | После отправки, статья попадает на рассмотрение администратора, который проверяет статью, после чего либо опубликовывает ее, либо направляет сообщение пользователю с просьбой исправить недочеты |

Таблица 3. Спецификация прецендента "Решение теста с курса"

|  |  |
| --- | --- |
| Раздел спецификации прецендента | Описание |
| Название | Решение теста с курса |
| Актер | Студент |
| Предусловие | Преподаватель во время создания курса, либо позже добавил студента в курс, чтобы тот мог видеть задания в нем. Студент, войдя в свой аккаунт и перейдя в профиле во вкладку «Курсы» открывает нужный курс и открывает нужный тест |
| Основной поток | Перед студентом высвечивается тема теста, а также вопросы по тесту с вариантами ответов в виде чекбоксов, после того как студент выберет ответы, которые считает нужными во всех вопросах, он нажимает кнопку «Сдать», далее отправляется запрос на проверку теста |
| Альтернативный поток | В случае закрытия пользователем страницы с тестом, все данные об ответах сбрасываются и студенту придется проходить тест заново |
| Постусловие | После отправки запроса на проверку теста, высчитываются баллы студента по стобалльной системе, а также этот студент добавляется в список сдавших данный тест |

* 1. Диаграмма классов

Диаграммы классов могут быть двух уровней. Первый – концептуальный; используется для изображения модели данных будущей системы. Второй – уровень реализации приближен к структуре классов готовой системы. На нем изображаются классы системы, которые тем или иным образом должны реализовывать функциональность системы, прецеденты, модель данных и работу с данными.

На представленном ниже рисунке показана часть концептуальной диаграммы классов, показывающая отношения между объектами «Аккаунт» и объектами подсистемы «Новости»(рис. 2).

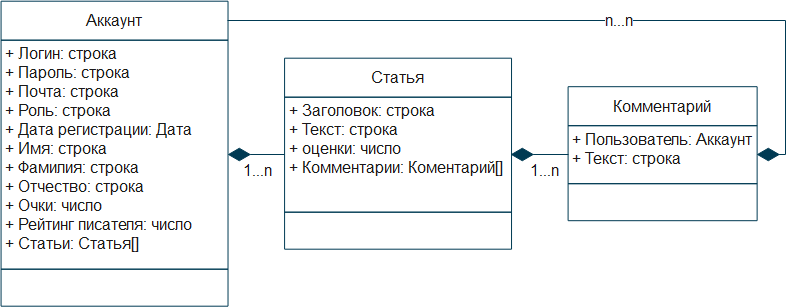


Рисунок 2. Диаграмма классов - Отношение пользователя с подсистемой "Новости"

Полные версии концептуальной диаграммы классов и диаграммы классов уровня реализации представлены в приложении Г.

Сущности программной системы были спроектированы с помощью подхода code-first, который означает, что сначала реализуются классы в системе, после чего на их основе создаются сущности в базе данных. Данная система была реализована с помощью технологии Hibernate.

Система взаимодействия с клиентской частью построена на разделении серверной части приложения на 3 уровня: уровень обработки запросов(контроллеры), уровень бизнес-логики(сервисы) и уровень доступа к данным(репозитории)

На рисунке 3 представлены классы уровня доступа к данным.

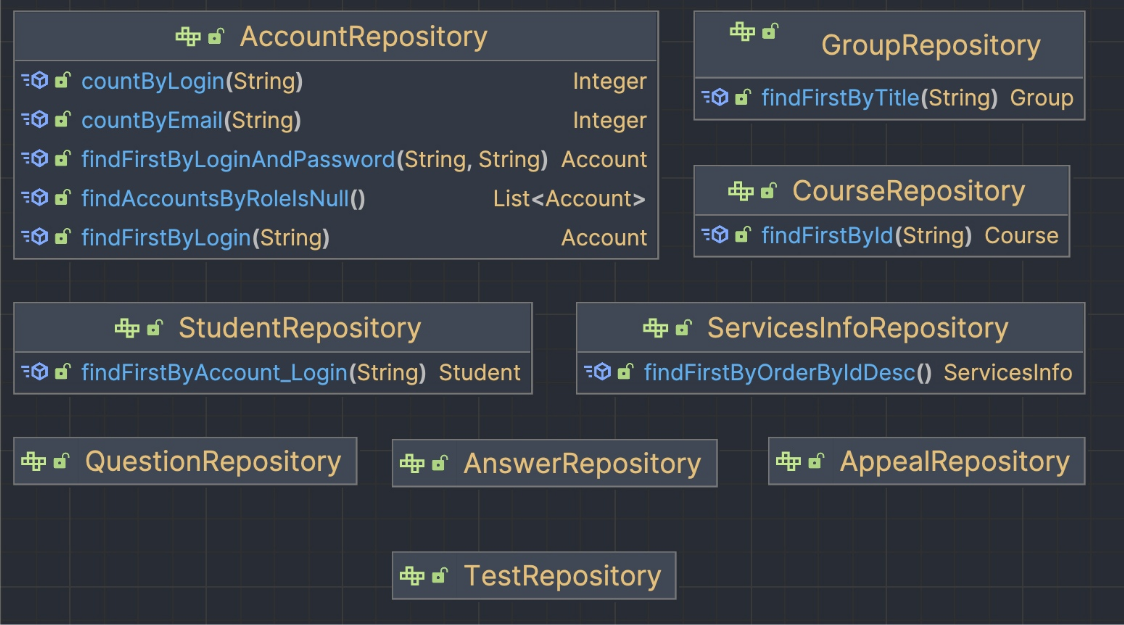


Рисунок 3. Классы уровня доступа к данным

AccountRepository – обеспечивает доступ к персональным данным пользователя в системе

GroupRepository – обеспечивает доступ к информации о группах в университете

CourseRepository – обеспечивает доступ к курсам в системе

StudentRepository – обеспечивает доступ к информации о студентах в системе

ServicesInfoRepository – обеспечивает доступ к сервисной информации системы, такой как дата дня открытых дверей, количество программ обучения, количество преподавателей, количество выпускников

QuestionRepository – обеспечивает доступ к вопросам тестов в системе

AnswerRepository – обеспечивает доступ к ответам к вопросам в системе

AppealRepository – обеспечивает доступ к обращениям пользователей системы

TestRepository – обеспечивает доступ к тестам внутри курсов в системе

На рисунке 4 представлены классы уровня бизнес-логики, которые предназначены для инкапсуляции бизнес-логики внутри себя и могут изменяться независимо друг от друга и от других уровней, что дает возможность разделения кода на зоны ответственности.

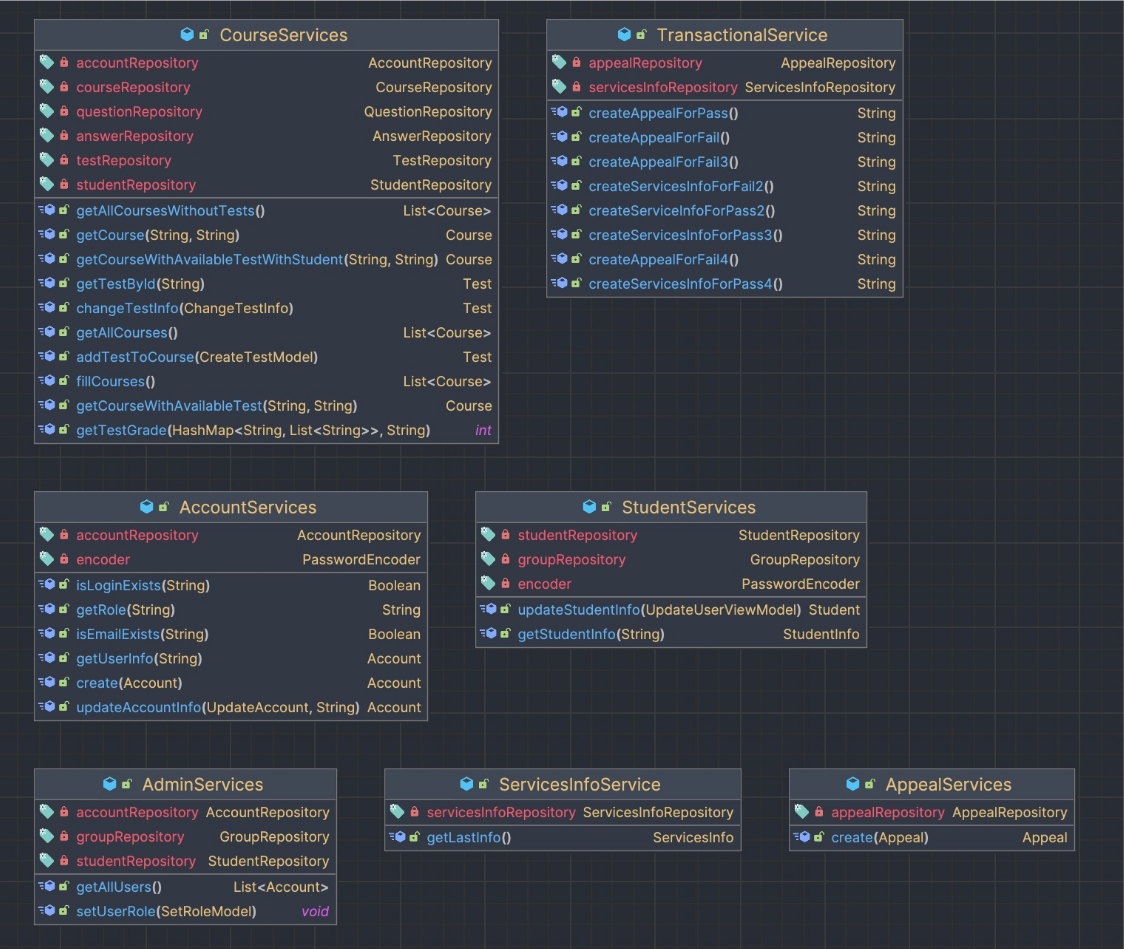


Рисунок 4. Классы уровня бизнес-логики

На рисунке 5 представлены классы уровня взаимодействия с клиентом, этот уровень отвечает за принятие запросов от пользователя и перенаправление управления к уровню бизнес-логики.

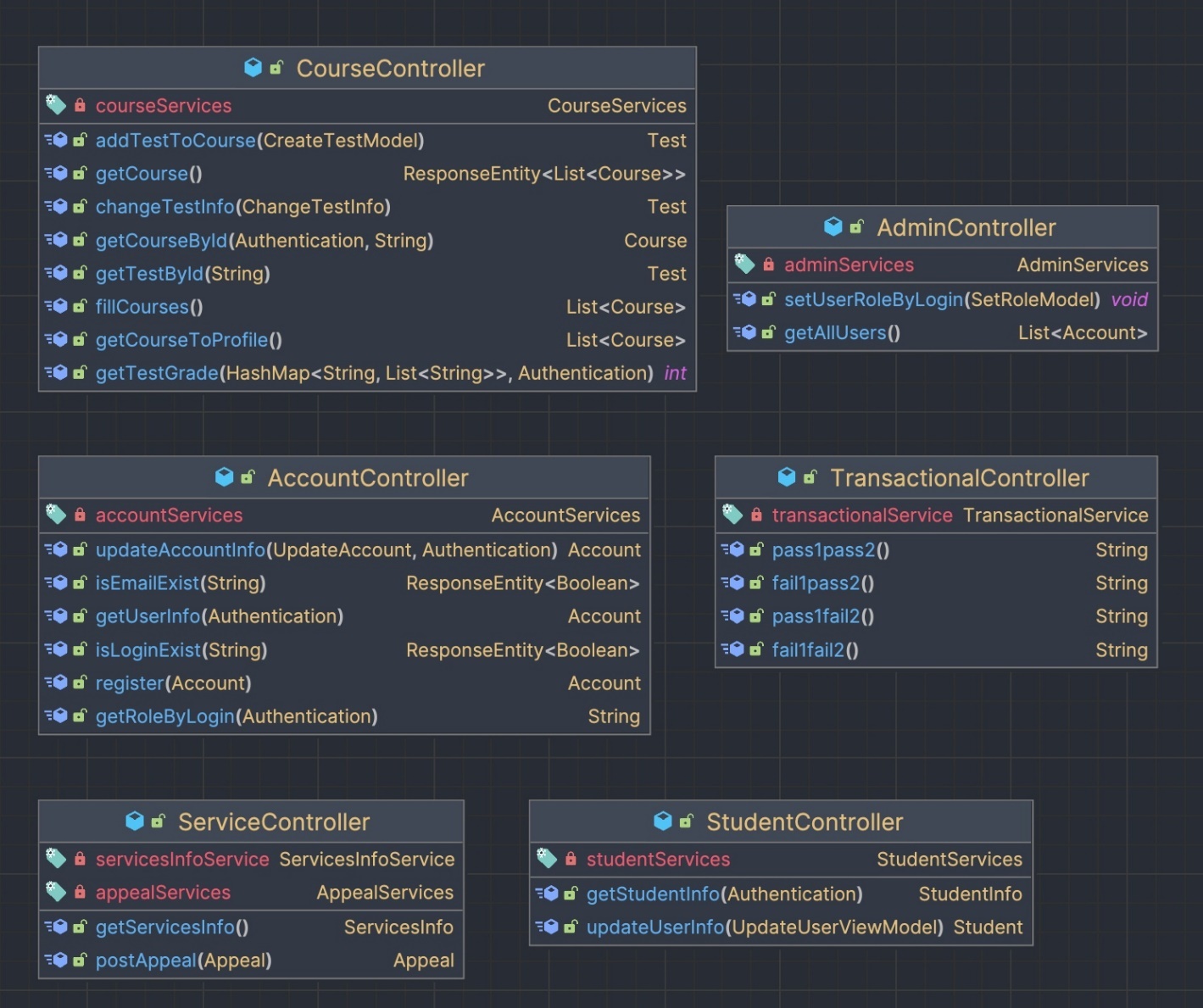


Рисунок 5. Классы уровня взаимодействия с клиентом

* 1. Диаграмма состояний

Диаграмма состояний нужна для изображения состояний, в котором может находится бизнес-объект системы.

На представленном ниже рисунке показана диаграмма состояния объекта «Тест» (рис. 6). На ней представлены различные состояния доступности теста в зависимости от действий пользователей разных ролей.

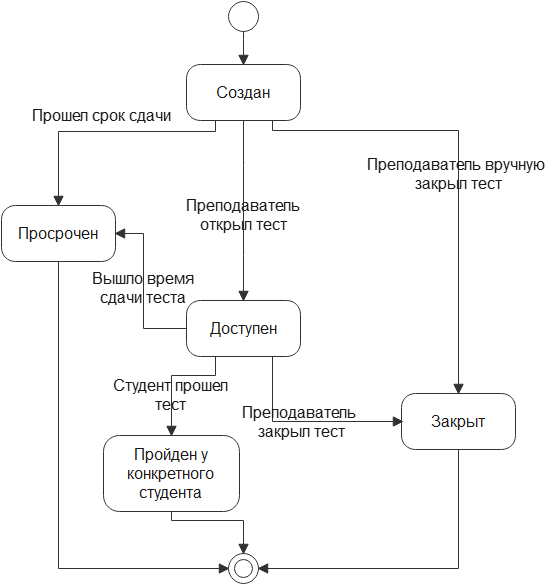


Рисунок 6. Диаграмма состояний - Тест

* 1. Диаграмма последовательностей

На диаграмме последовательностей система изображается в динамике. На ней изображается взаимодействие объектов во времени, время жизни и активности.

На представленном ниже рисунке показана диаграмма последовательностей процесса авторизации (рис. 7).

Так же в приложении Д представлена диаграмма последовательностей подачи документов абитуриентом. Для нее была сформирована диаграмма коммуникаций, которая показывает взаимодействие компонентов системы в процессе подачи заявления абитуриентом. (рис. 8)

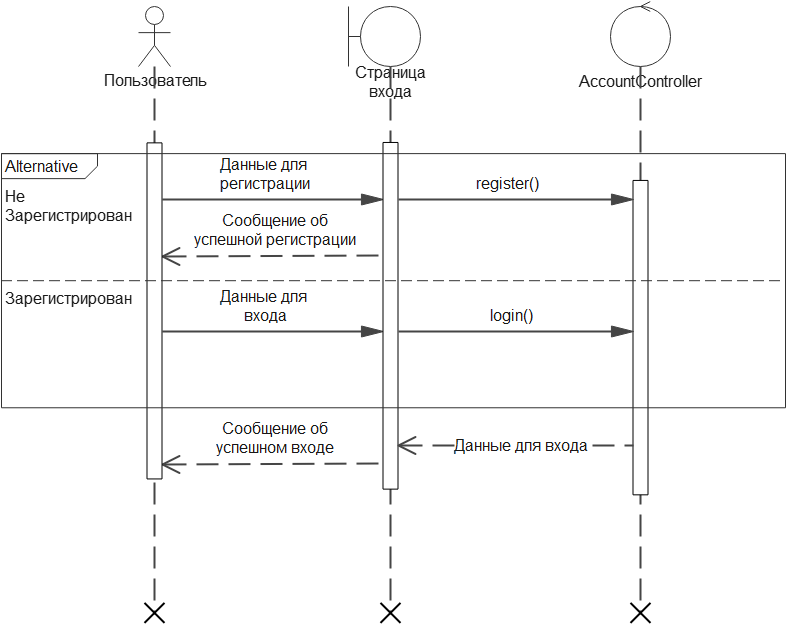


Рисунок 7. Диаграмма последовательностей – Авторизация

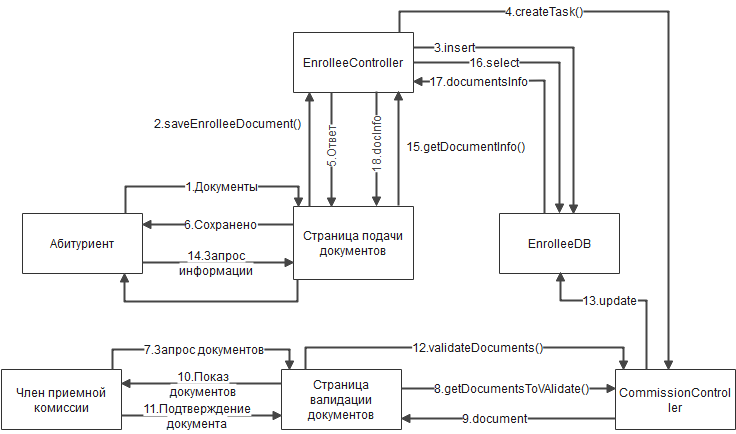


Рисунок 8. Диаграмма коммуникаций

* 1. Диаграмма BPMN

На диаграмме BPMN показывается то, как поток управления переходит от одной деятельности к другой.

На представленном ниже рисунке показана диаграмма BPMN процесса подачи документов в приемную комиссию (рис. 9).

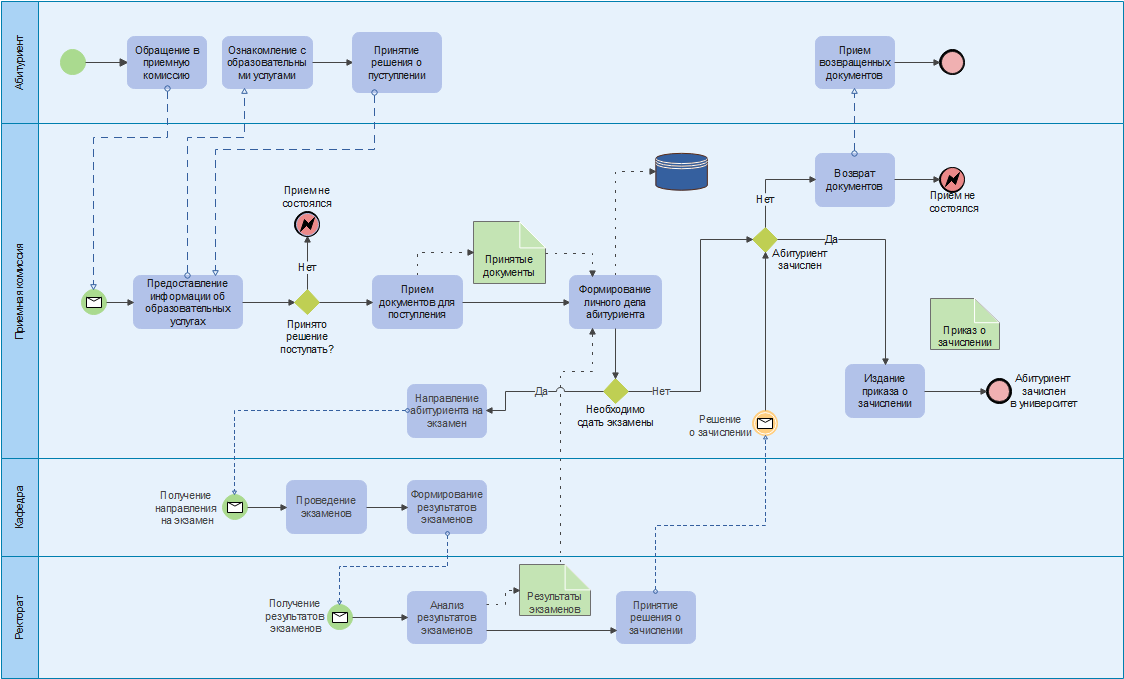


Рисунок 9. Диаграмма BPMN

* 1. Диаграмма зон ответственности

Диаграмма зон ответственности показывает то, какие действия находятся под чьей ответственностью и схожа с диаграммой прецендентов.

На представленном ниже рисунке показана диаграмма зон ответственности для процесса создания курсов и прохождение тестов в нем студентами (рис. 10)

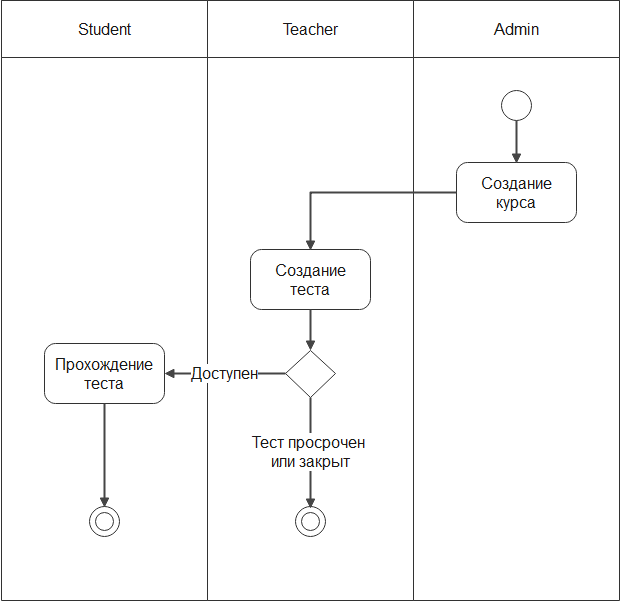


Рисунок 10. Диаграмма зон ответственности

* 1. Диаграмма развертывания

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их.

Приложение разворачивается на нескольких отдельных серверах, которые взаимодействуют между собой посредством распределительного сервера.

Диаграмма развертывания представлена на рисунке 11.

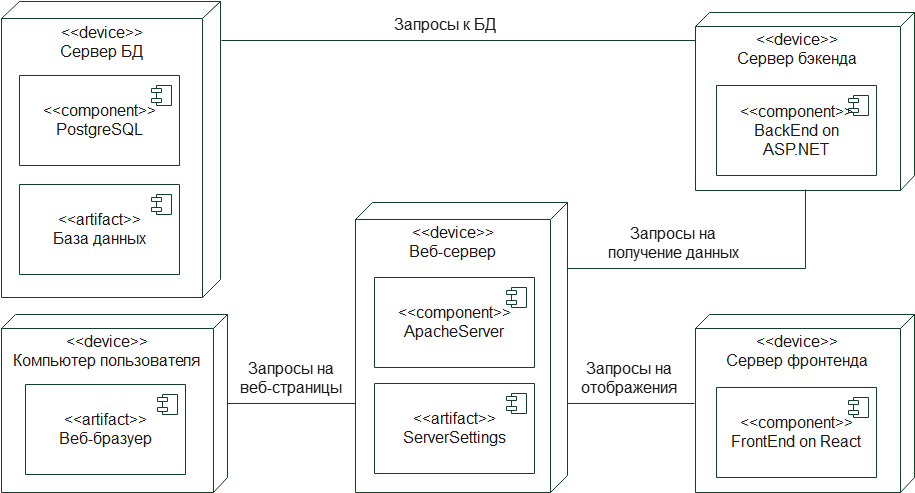


Рисунок 11. Диаграмма развертывания

* 1. Схема навигации приложения

На схеме навигации описываются возможные пути перехода между страницами в приложении.

В данном случае схема навигации получается довольно простая в виду того, что в приложении содержится зафиксированная навигационная панель с помощью, которой можно попасть с любой страницы на практически любую другую страницу. (рис. 12)



Рисунок 12. Схема навигации

* 1. Диаграмма взаимодействия

Диаграмма взаимодействия — это диаграмма, на которой представлено взаимодействие, состоящее из множества объектов и отношений между ними, включая и сообщения, которыми они обмениваются.

На рисунке 13 представлена диаграмма взаимодействий процесса приемной комиссии от лица студента включая действия системы.

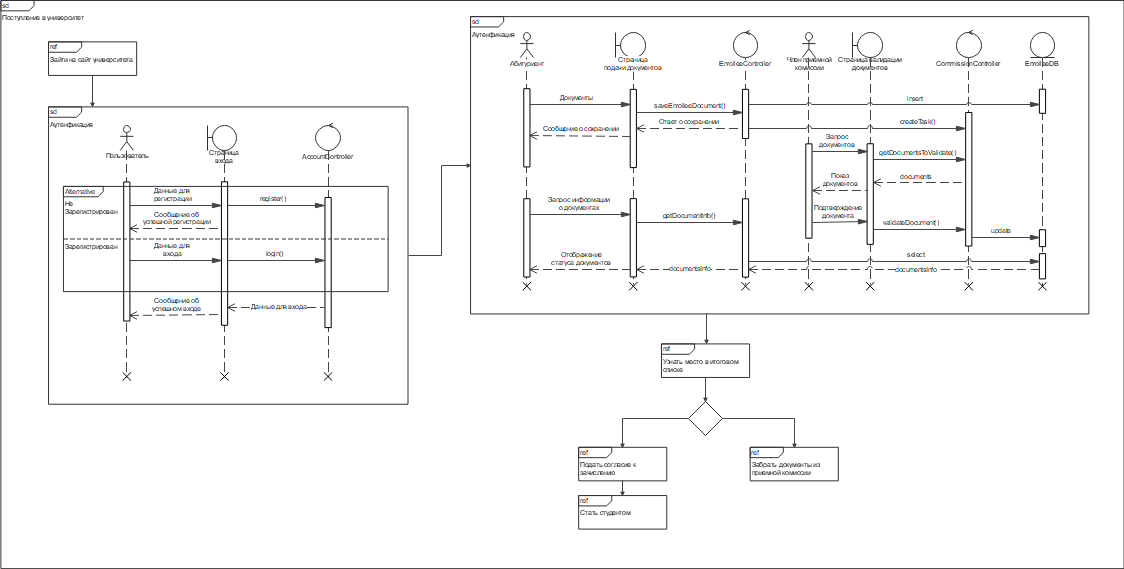


Рисунок 13. Диаграмма взаимодействий

* 1. IDEF0

IDEF0 — методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов.

На рисунках 14 и 15 представлены IDEF0 диаграммы описывающие бизнес-процесс «Деятельность университета» в целом и «Выполнение заданий» точечно.

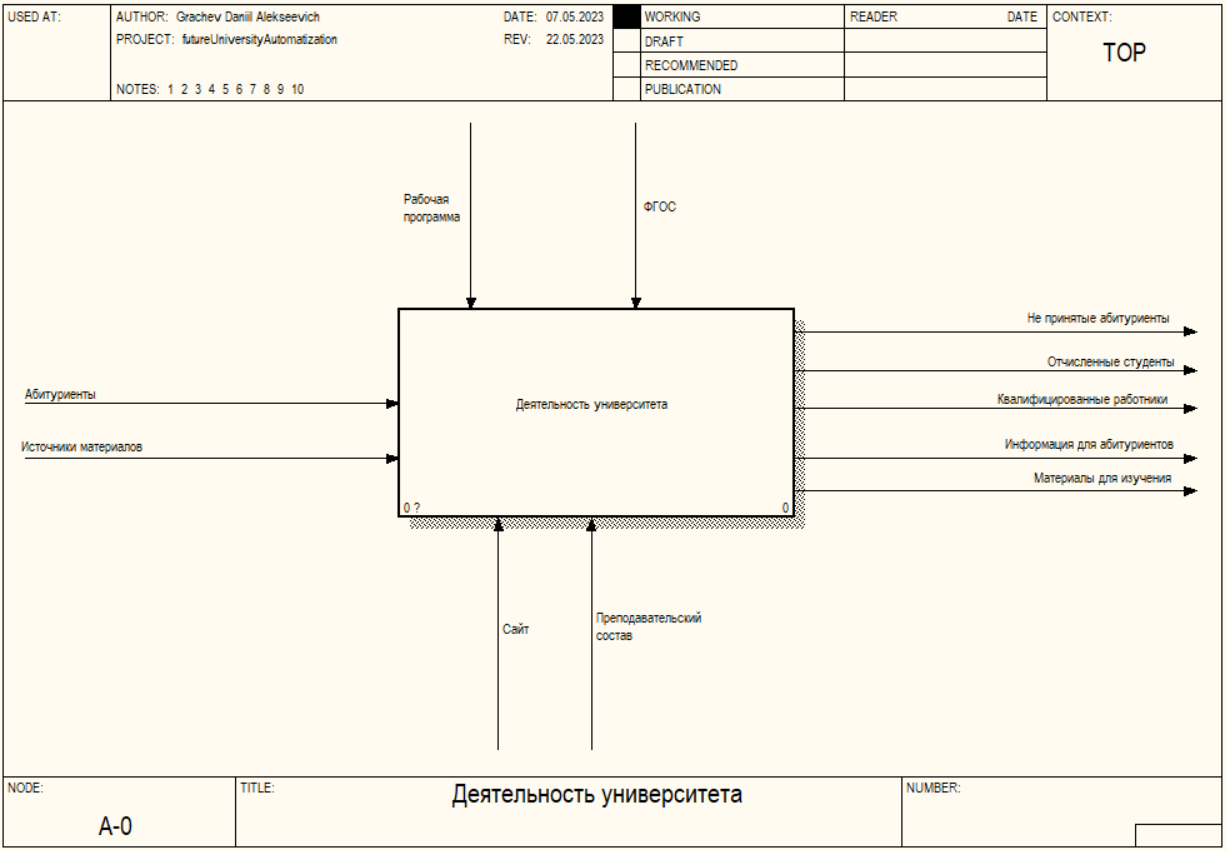


Рисунок 14. IDEF0 - уровень 0

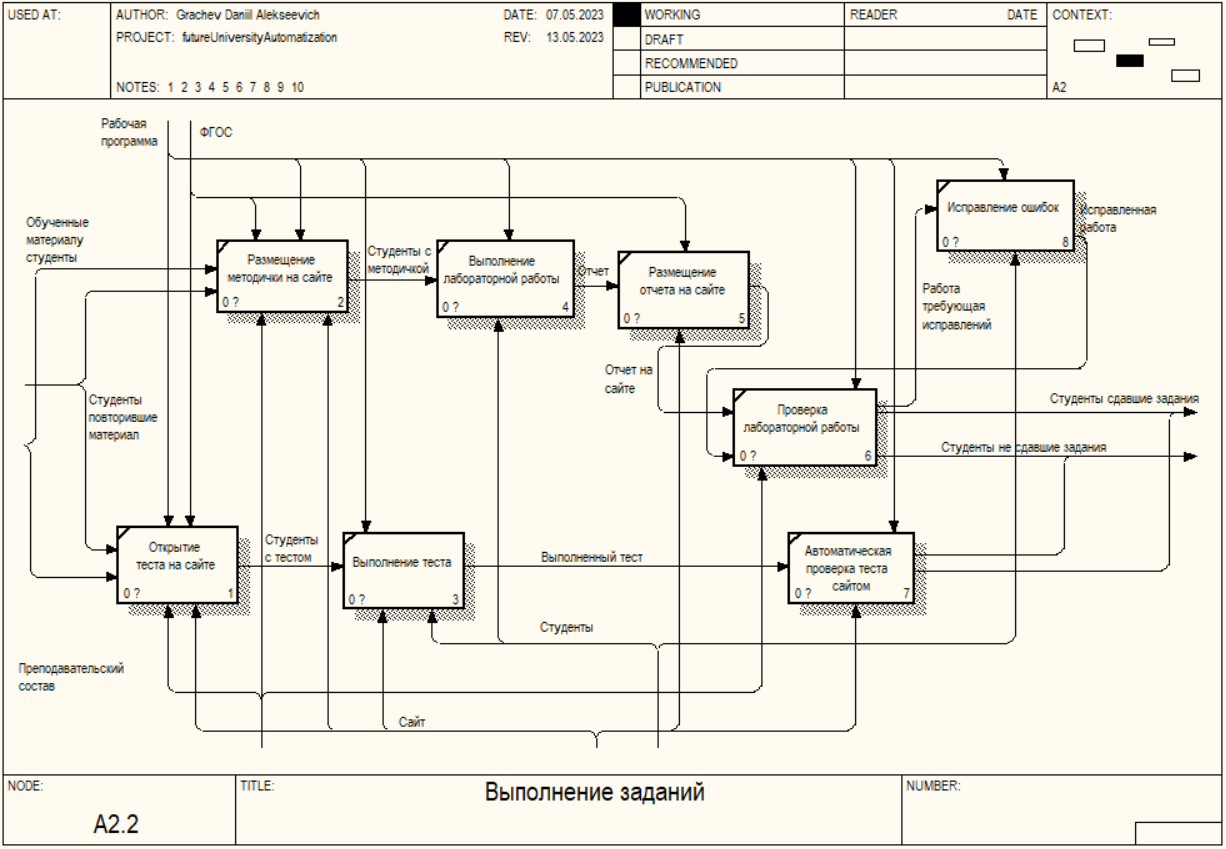


Рисунок 15. IDEF0 - уровень 2

* 1. Общие принципы организации системы

Разрабатываемая система представляет собой веб-приложение, построенное на базе архитектуры SpringBoot по шаблону «веб-API».

Как и в любом приложении, реализующее шаблон web-API, программная система «Университет» содержит четыре компонента: модели данных, контроллеры, сервисы и репозитории для общения с интерфейсом по средством HTTP-запросов.

Каждый контроллер относится к своему классу модели и содержит необходимые методы, которые принимают запросы пользователей и перенаправляют их для обработки в другие классы и в результате своей работы возвращают данные для отправки их на интерфейс.

Каждый репозиторий относится к своей сущности в базе данных и взаимодействуя только с ней способен возвращать данные из нее или изменять хранящуюся информацию.

Каждый сервис относится к своему контроллеру и берет на себя выполнение всей бизнес-логики и взаимодействие с репозиториями.

Для реализации моделей и применения подхода Model First использован Hibernate.

# Разработка системы

* 1. База данных

В качестве системы управления базой данных была выбрана PostrgeSQL – свободная реляционная система управления базами данных, разрабатываемая и поддерживаемая силами сообщества.

На представленном ниже рисунке показана логическая схема базы данных (рис. 16). В ней присутствуют две взаимосвязанный подсистемы «Авторизации», которая хранит все данные о аккаунте и студенте, и «Курсы», которая хранит в себе данные о курсах, присутствующих в системе, также на схеме видно, что присутствуют две не связанные ни с чем таблицы: таблица «Обращения», которая хранит обращения пользователей для обратной связи, и таблица «Сервисная информация», которая хранит информацию о днях открытых дверей, количестве программ в университете, количестве преподавателей и количестве выпускников.



Рисунок 16. Логическая схема базы данных

* 1. Интерфейс

Так как для реализации проекта был выбран паттерн WEB-API – появилось разделение системы на две составные части: само API, так называемый “сервер”, о нем мы поговорим позже, и представление – интерфейс, с которым взаимодействует пользователь.

Для реализации пользовательского интерфейса была выбрана библиотека React.js – javascript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов, которая разрабатывается и поддерживается запрещенной в России организацией. Данная библиотека позволяет разрабатывать одностраничные веб-приложения без перезагрузок, с помощью подмены данных на странице

* + 1. Реализация маршрутизации

С помощью подключения дополнительной библиотеки “react-router-dom” можно настроить маршрутизацию в приложении (рис. 17)



Рисунок 17. Маршруты приложения

Также для удобства разработки пути были вынесены в перечисляемый тип enum(рис. 18)

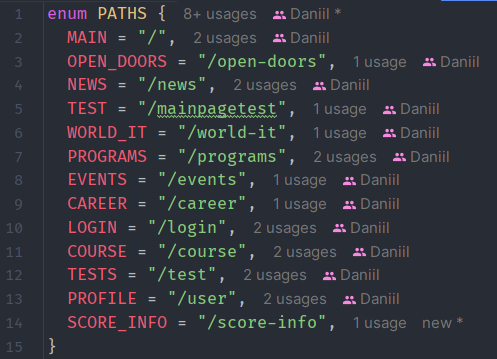


Рисунок 18. Перечисление маршрутов приложения

* + 1. Главная страница приложения

Вверху главной страницы приложения мы можем увидеть три составные части (рис. 19). В самом верху располагается навигационная панель приложения, при нажатии на элементы которой будет происходить переход на другие страницы (рис. 19-а). На востоке от него находится метка пользователя, на данном рисунке видно, что пользователь вошел в свой аккаунт, так как отображается его имя и соответствующий роли цвет, если пользователь еще не вошел в свой аккаунт или не зарегистрировался, в этом месте будет отображаться надпись «Войти», по нажатию на которую произойдет переход на страницу регистрации и входа (рис. 19-б). Внизу рисунка отображается информация о университете, в данной области можно увидеть количество программ, преподавателей и выпускников в университете (рис. 19-в). Немного ниже этого так же можно увидеть информацию о ближайшем дне открытых дверей.

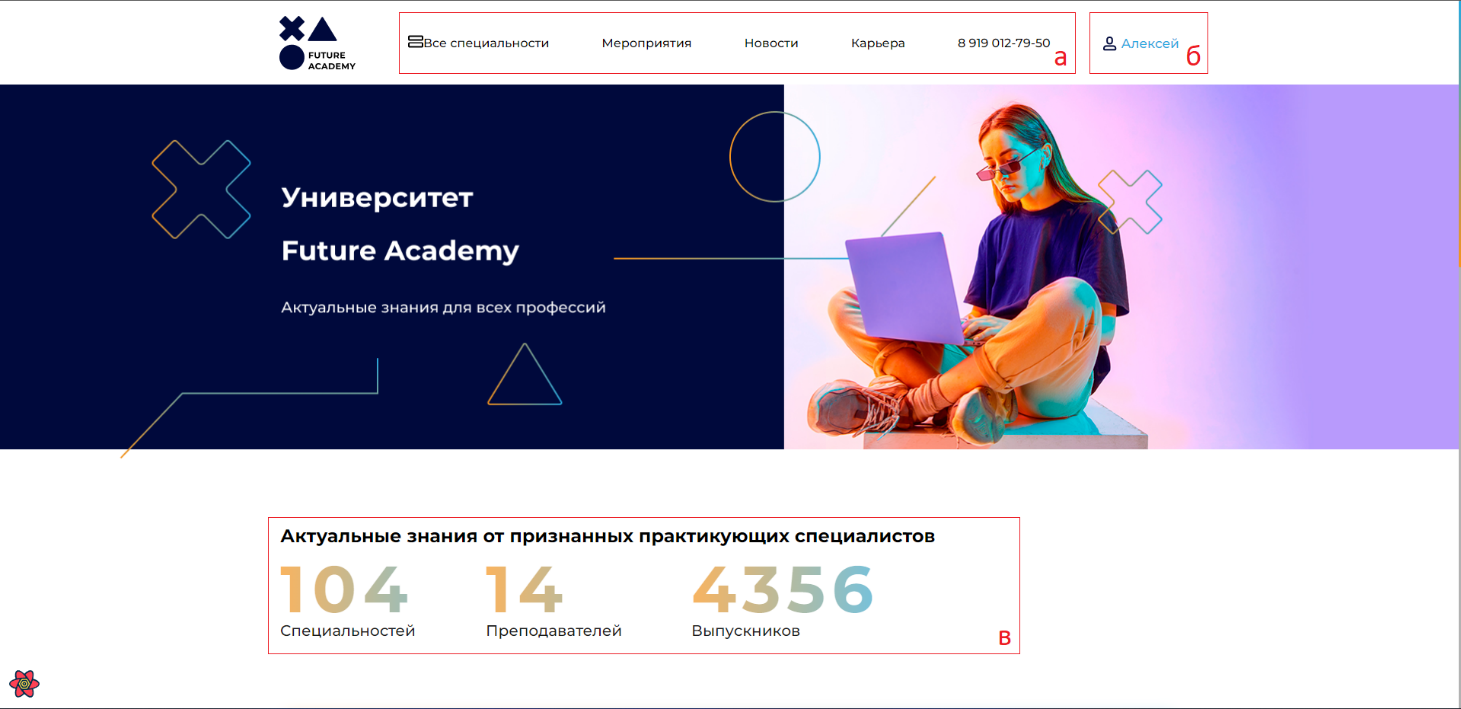


Рисунок 19. Главная страница приложения (а – навигация, б – метка пользователя, в – информация о университете)

* + 1. Профиль пользователя

При открытии профиля пользователя мы увидим разделение на два блока: левый и правый (рис. 20). В левом блоке можно увидеть аватар пользователя, краткую информацию о его имени, почте и очках, также переключатель вкладок, который будет отображаться по-разному в зависимости от роли пользователя, а также от того находится он на странице своего аккаунта или нет (рис. 20-а). В правой части профиля можно заметить открытую вкладку «Общие сведения», в которой будет указана более подробная информация об аккаунте пользователя, данная вкладка открывается по стандарту и будет единственной доступной при открытии чужого профиля (рис. 20-б).

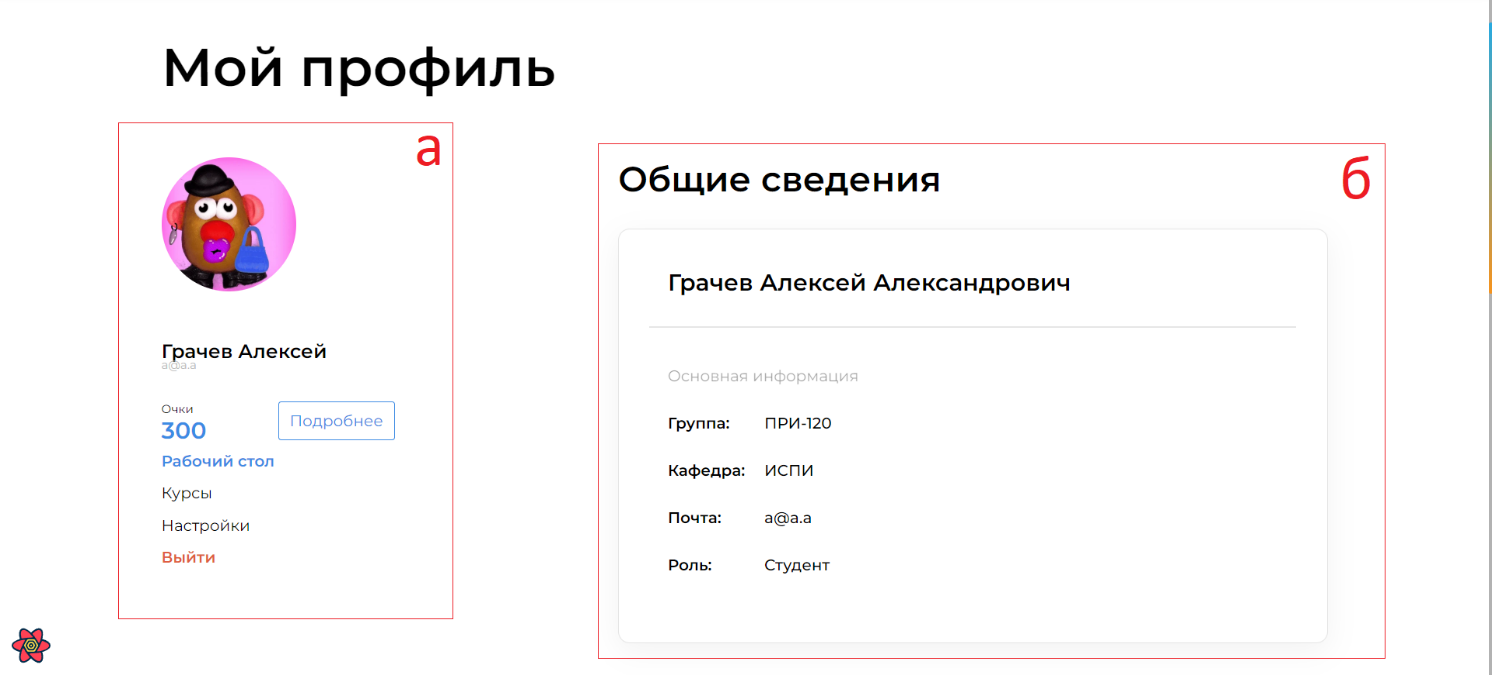


Рисунок 20. Профиль пользователя (а – краткая информация, б – вкладки)

Так же на данной странице можно перейти к курсам студента и настройкам аккаунта.

Информация о пользователе в приложении является “реактивной”, что обозначает ее обновлении на странице при изменении данных на сервере, например, если пользователь обновит данные о себе, с помощью вкладки настройки, то он увидит изменения в информации без перезагрузки страницы. Данный эффект достигается с помощью библиотеки “react-query”, которая раз в указанный период сверяет данные в браузере с данными на сервере и, в случае их отличия, подгружает новые данные, которые мгновенно показываются пользователю.

* + 1. Регистрация, авторизация и выход из аккаунта

Перейдя на страницу входа и регистрации, если уже не вошли в аккаунт, мы можем увидеть две вкладки с формами для заполнения: вход и регистрацию (рис. 21).

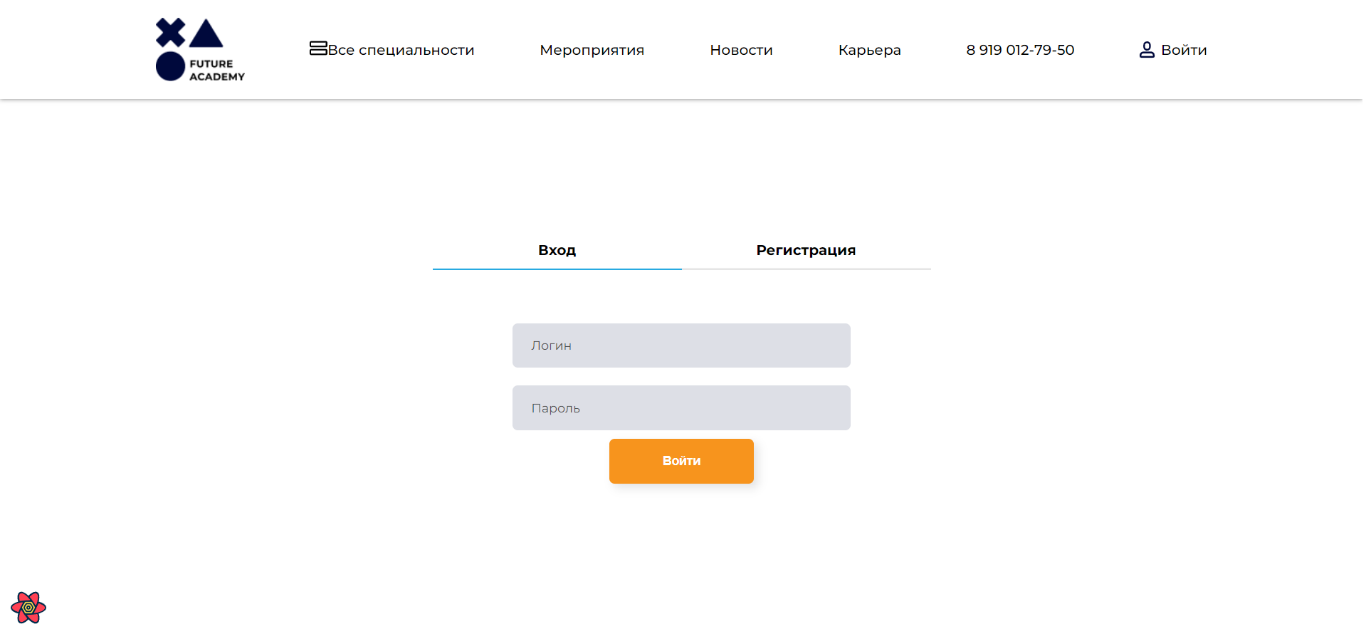


Рисунок 21. Страница входа/регистрации

Данные формы были реализованы с помощью библиотеки “react-hook-form”, которая позволяет быстро и удобно делать формы и отправлять данные с них. Когда пользователь заполнил форму, происходит валидация данных в ней(подробное описание процесса регистрации описано в *таблице 1. Спецификация прецендента «Создание аккаунта»*). В целях безопасности данных пользователя при сохранении паролей происходит их хеширование.

В целях ознакомления на рисунке ниже представлен хешированный вид паролей, хранящихся в базе данных (рис. 22)

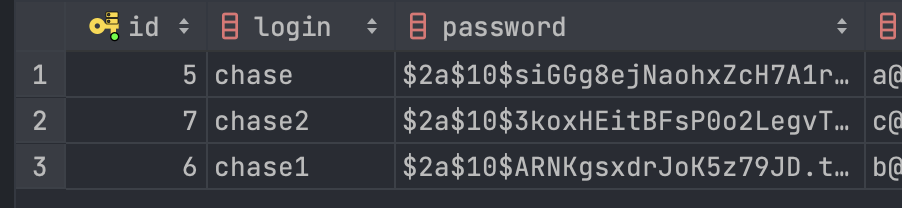


Рисунок 22. хешированные пароли

* + 1. Тесты и их сериализация

Зайдя под аккаунтом студента, мы можем пройти доступные нам тесты (тесты могут быть не доступны по нескольким причинам: преподаватель явно установил у теста статус «Закрыт», у теста истек срок сдачи, вы уже прошли данный тест), они будут находиться в курсе, которых находится во вкладке «Курсы» в профиле (рис. 23).

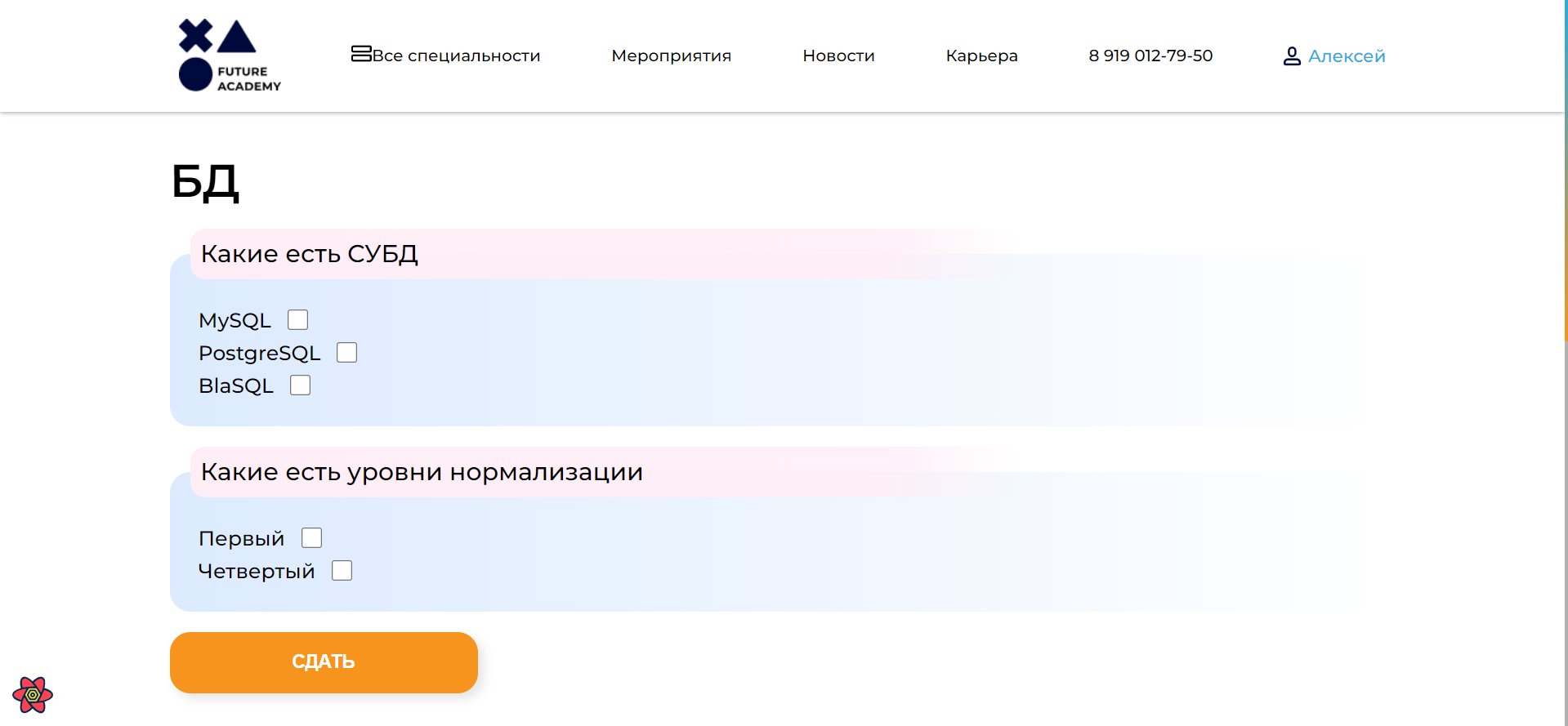


Рисунок 23. Страница теста

На странице теста будут представлены все вопросы по этому тесту, а вверху страницы будет записана тема теста. После того, как студент выберет все ответы, которые считает правильными, он может нажать кнопку сдать (более подробная информации представлена *в таблице 3. Спецификация прецендента «Решение теста с курса»*). После этого происходит обработка данных формы на стороне клиента: данные из формы преобразуются в читабельный для сервера вид с помощью сериалайзера(рис. 24)



Рисунок 24. Сериалайзер формы

Про оценивание тестов будет описано в пункте *3.3.3. Оценка теста*

В случае если мы зайдем под аккаунтом с ролью преподавателя – получим возможность создавать тесты в курсах с набором вопросов, редактировать созданные ранее тесты указывая дату автоматического закрытия тестов и их доступность студентам. На рисунке 25 представлен интерфейс, который увидит преподаватель в момент создания тестов. В нем есть возможность добавить вопросы, а также ответы к ним.

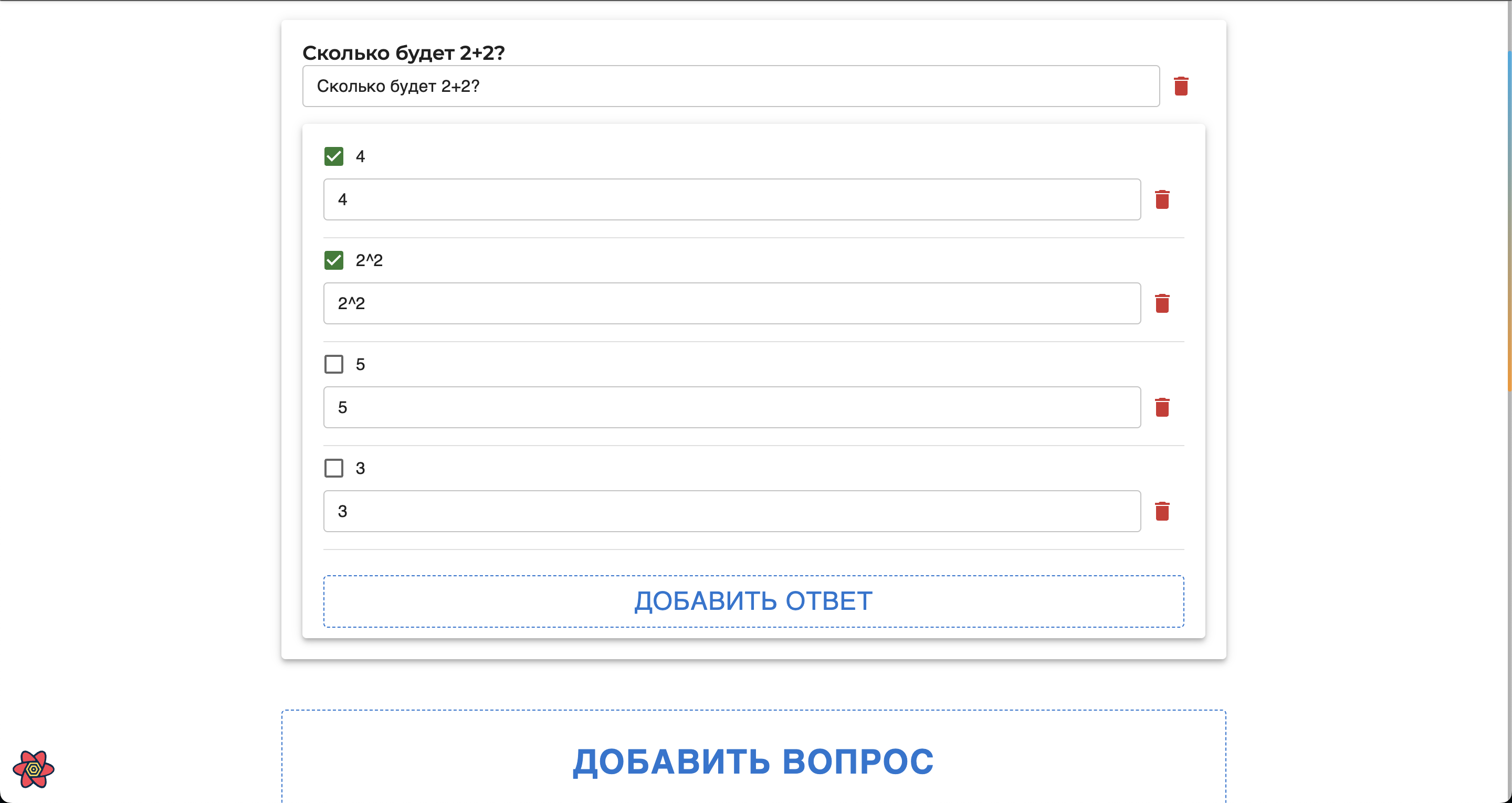


Рисунок 25. Интерфейс создания теста

После завершения работы над вопросами преподаватель может нажать на кнопку «СОХРАНИТЬ», после чего появится дополнительное диалоговое окно, в котором можно указать название теста, которое будет отображаться в курсе, дату окончания, после которой тест будет недоступен для прохождения студентами, и ручная доступность теста. (рис. 26). Так же, такое же окно будет появляться при изменении информации о уже существующем тесте.

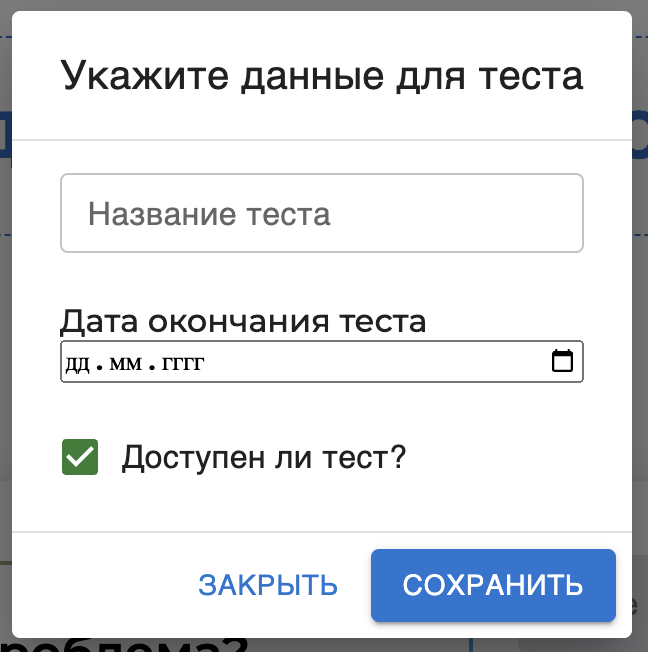


Рисунок 26. Интерфейс настройки теста

* + 1. Использование API

Так как приложение использует шаблон WEB-API – интерфейс общается с сервером посредством HTTP-запросов по адресу, указанному в контроллере. На уровне клиента для отправки запросов, не требующих обновления данных, как для информации о пользователях, используется библиотека axios, которая позволяет легко обратиться к API и получить ответ (рис. 27). Метод get, отвечающий за отправку HTTP-GET запроса, принимает в себя обязательный параметр “url”, который является путем до конечной точки в контроллере, который и обрабатывает этот запрос (обработка со стороны “сервера” и контроллера описана в пункте *3.3.2. Реализация конечных точек API*). В данном примере, после получения данных, они записываются в “стейт” компонента.

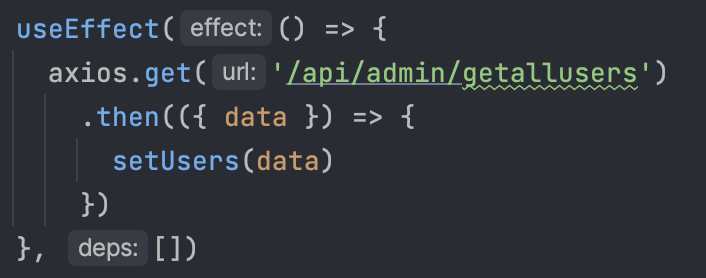


Рисунок 27. GET-запрос с помощью axios

* + 1. Панель администратора

В панели администратора присутствует единственный интерактивный элемент – это таблица со списком пользователей, в которой имеется вся информация об аккаунтах, за исключением паролей. Поле «Роль» является редактируемым для возможности сменить роль отдельного пользователя, в случае присваивания роли студента появится дополнительное всплывающее окно, в котором нужно будет указать группу студента и кафедру. Таблица является сортируемой и фильтруемой по всем столбцам (рис. 28).

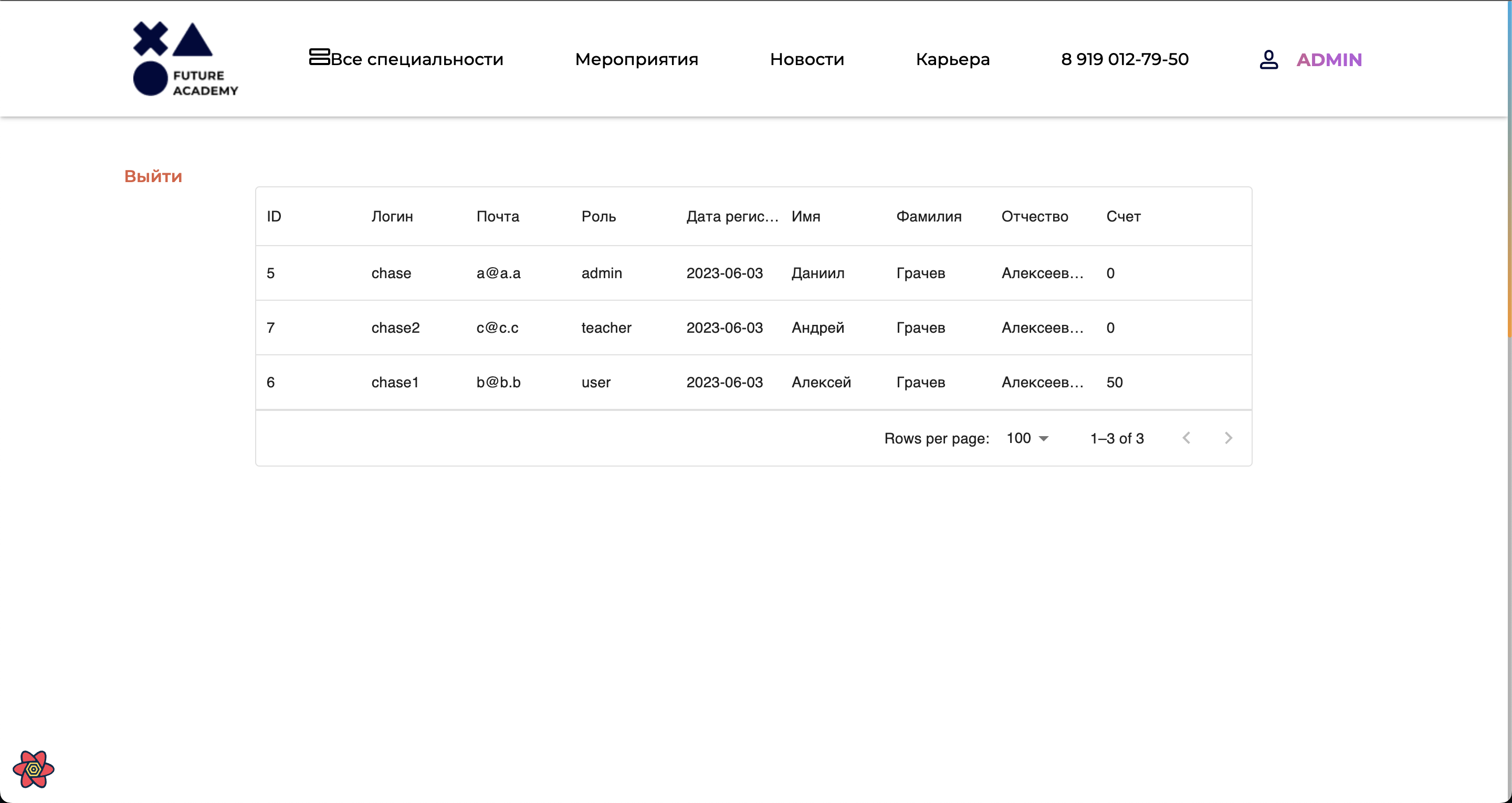


Рисунок 28. Панель администратора

* 1. Серверная часть

Для реализации серверной части была выбрана технология SpringBoot – универсальный фреймворк с открытым исходным кодом для Java-платформы, выпущенная компанией Pivotal Software.

* + 1. Модели сущностей в системе и подключение к базе данных

В системе девятнадцать моделей данных, девять из которых относятся к базе данных, остальные же описывают данные получаемые и отправляемые в интерфейс пользователя (рис. 29)

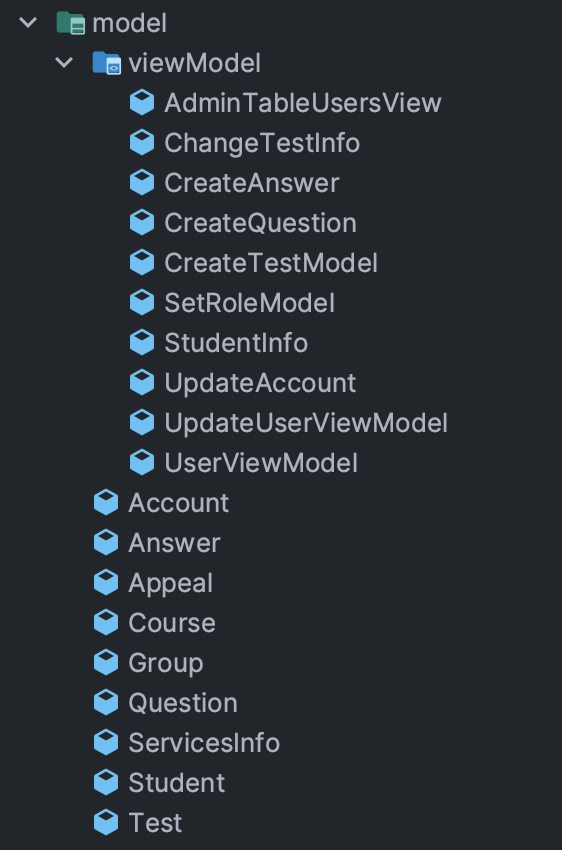


Рисунок 29. Модели в системе

Те из показанных моделей, которые относятся к базе данных, помечены аннотациями Entity и Table с указанием названия таблицы (рис. 30), к которой происходит подключение с помощью строки подключения, указанной в файле настроек приложения (рис. 31).

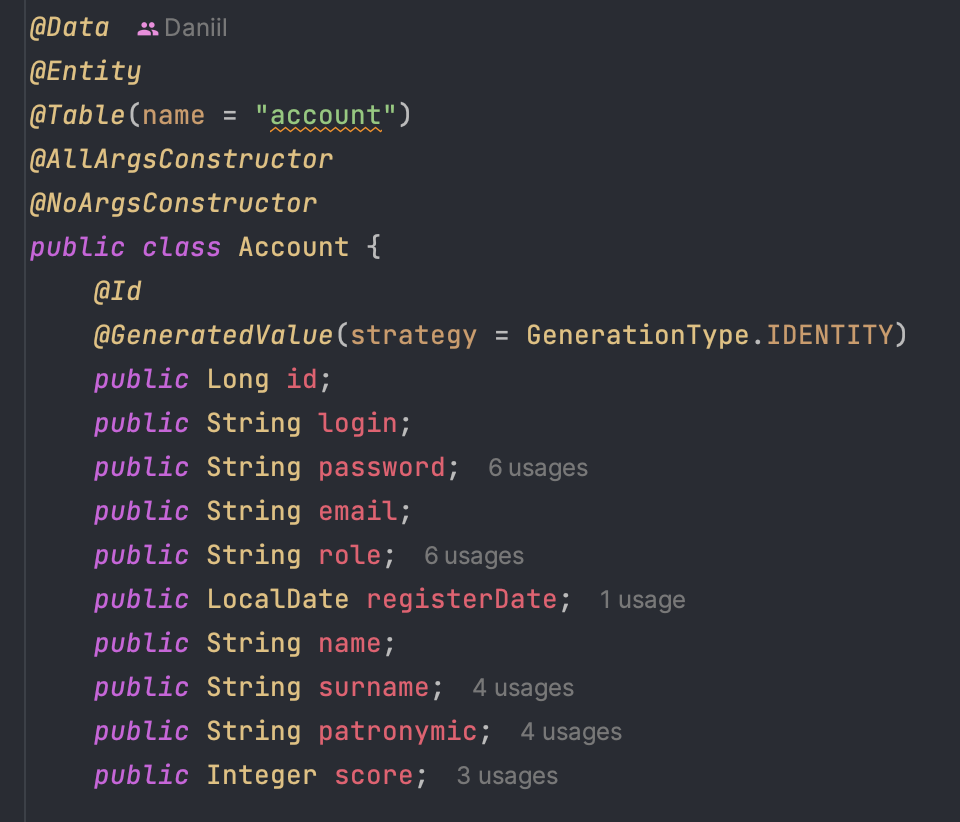


Рисунок 30. Аннотированная модель

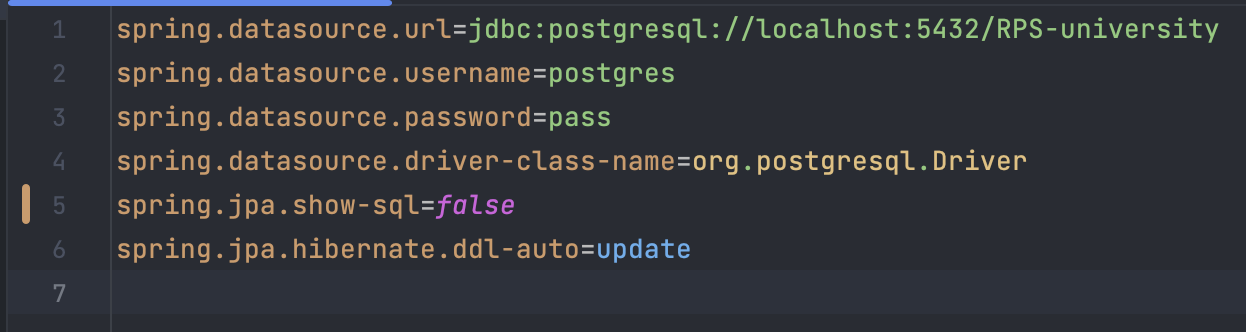


Рисунок 31. Строка подключения к базе данных

Далее с помощью Hibernate мы можем установить значение для его действий при запуске приложения, в данном случае используется «update», что означает, что база данных при каждом запуске будет обновляться, сверяясь с моделями данных, и обновить состояние базы данных, использовав подход “Model First”, что позволяет нам, как разработчикам, сосредоточиться на реализации бизнес-логики приложения, а не взаимодействии с СУБД.

* + 1. Реализация конечных точек API

В данном пункте будет описана обработка GET-запроса “отправленного” из пункта *3.2.6. Использование API.*

После отправки запроса от клиента на относительный адрес, прокси, настроенный в проекте, перенаправляет запрос на адрес, на котором запущен “сервер” и далее, с помощью API-контроллеров с указанными маршрутами(рис. 32) находят нужный метод обработки запроса, у которого так же может быть указан маршрут (относительно маршрута контроллера).



Рисунок 32. API-контроллер администратора

Когда “сервер” определил метод, который должен обработать запрос, он обращается к методам сервисного слоя для обработки бизнес-логики (рис. 33)

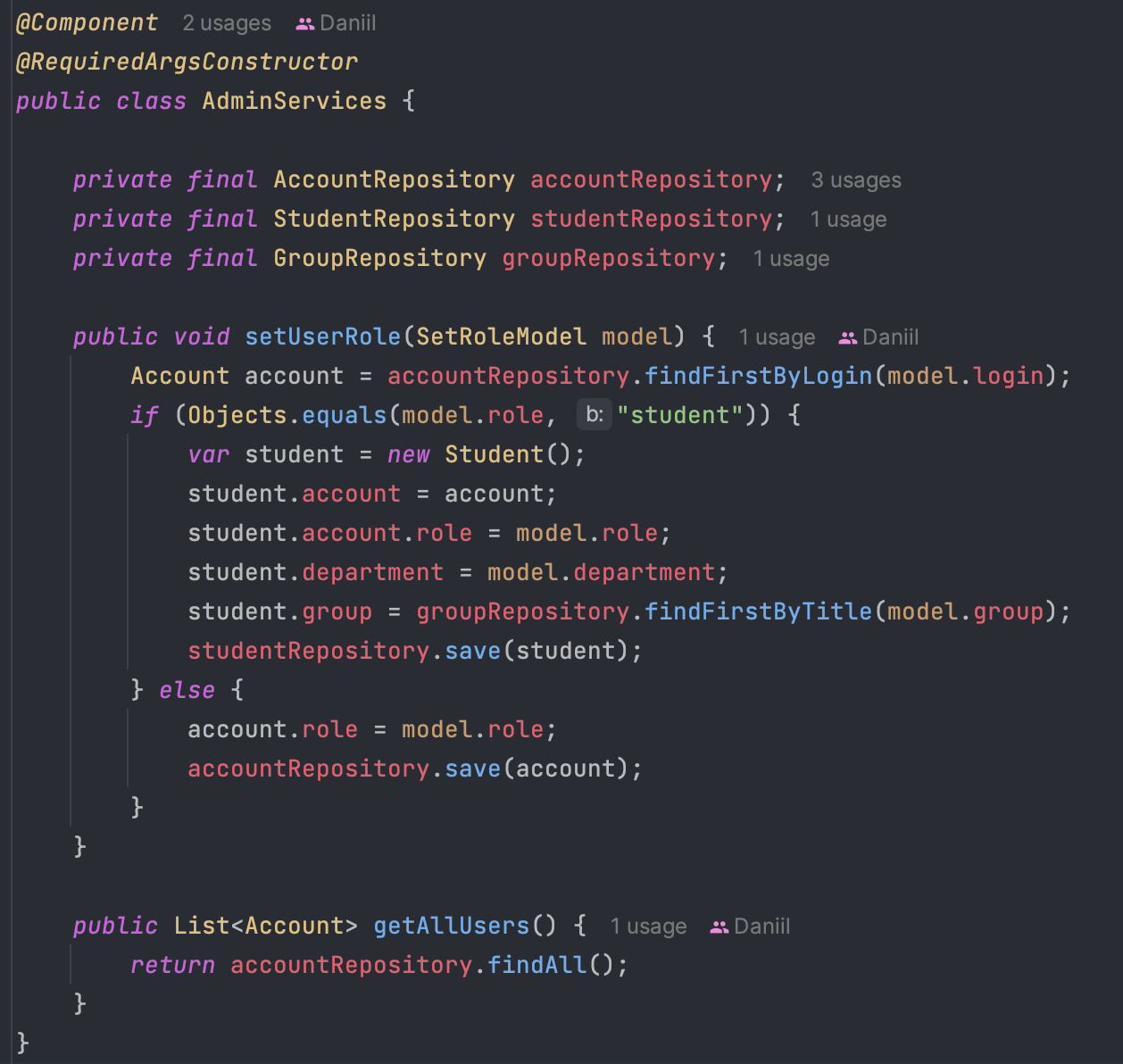


Рисунок 33.Сервисный слой

Далее сервисный слой обращается к базе данных посредством JpaRepository, который дает разработчикам удобный формат взаимодействия с базой данных с помощью названия методов, без написания SQL кода (рис. 34)

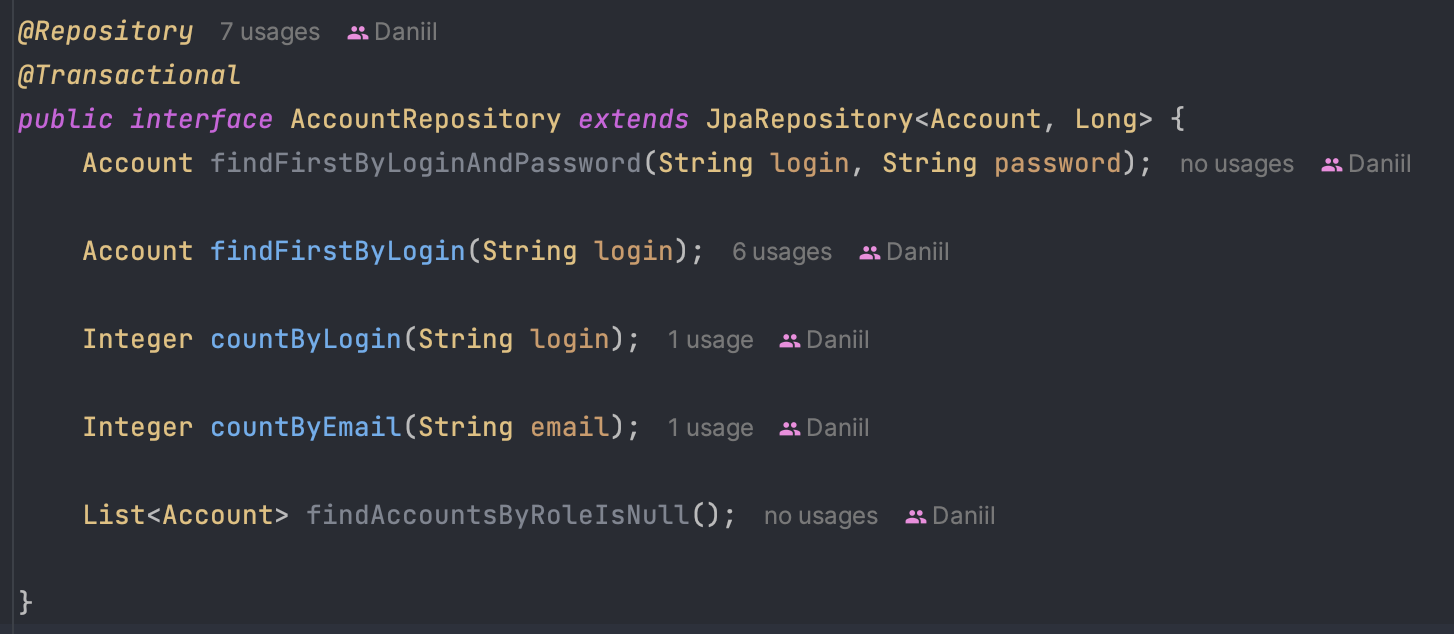


Рисунок 34. Репозиторий аккаунта

* + 1. Оценка теста

После действий, описанных в пункте *3.2.5. Тесты и их сериализация* от пользователя “серверу” приходят данные в формате словаря, где ключом является идентификатор вопроса, а значением массив строк с выбранными вариантами ответа(если вариант выбран, в строке записан его идентификатор, если не выбран, то на его месте остается пустая строка).

Для проверки правильности ответов студента из базы данных берется этот тест и так же преобразуется в словарь, после чего сравниваются правильные ответы с ответами от пользователя и в случае совпадения увеличивается счетчик правильных ответов у пользователя. После, из этого числа высчитывается балл по стобалльной системе, который прибавляется к очкам пользователя и позже возвращается клиенту. Также в этом методе происходит добавление студента в список сдавших тест, чтобы он не мог пройти тест дважды (рис. 35)



Рисунок 35. Метод обработки теста

# Тестирование

2. 1. Цель тестирования

Целью нагрузочного тестирования является проверка производительности системы при реальных нагрузках в виде отправки большого количества запросов, что позволяет определить производительности системы и найти узкие места.

* 1. Методика испытаний

Методика тестирования будет заключаться в постепенном увеличении нагрузки на сервер до околопредельных значений и момента появления явных ограничений, к которым может относиться: нехватка оперативной памяти на локальной машине, ограничение пропускной способности сети, нехватка вычислительной мощности процессора.

* 1. Алгоритм тестирования

Целевым значением одновременных пользователей в системе является 800 человек, максимальное количество 2000 одновременных подключений. Изначальное количество пользователей для тестирования будет установлено в количестве 200. На каждой итерации количество пользователей будет увеличивать до достижения целевого количества – 2000 пользователей.

* 1. План тестирования
     1. Пользователь авторизуется
     2. Получает всю информацию для своего аккаунта (этот запрос автоматически выполняется веб-приложением для возможности отображения личной информации пользователя)
     3. Получает все вопросы для теста
     4. Отправляет обращение
  2. Спецификации системы
     1. CPU: Apple M1, 4 х 3.2Ггц, 4 х 2.1Ггц
     2. ОЗУ: 8 Гб
  3. Процесс тестирования

Все тесты проводились по 5 минут

* + 1. 200 пользователей

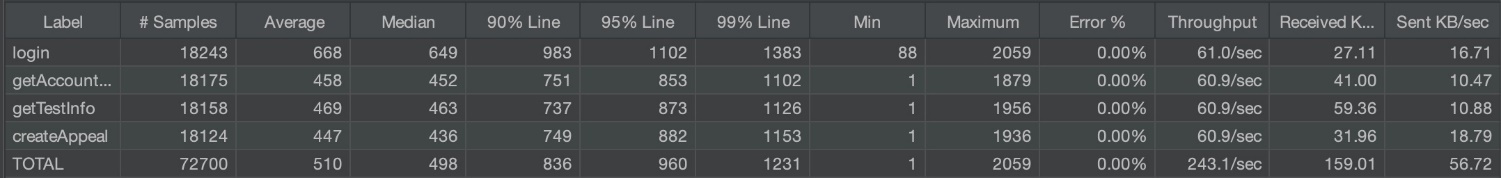


Рисунок 36. Тест с 200 пользователями

* + 1. 400 пользователей

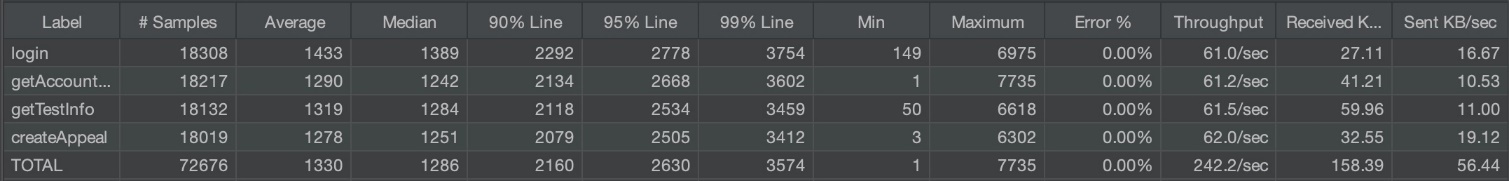


Рисунок 37. Тест с 400 пользователями

* + 1. 800 пользователей

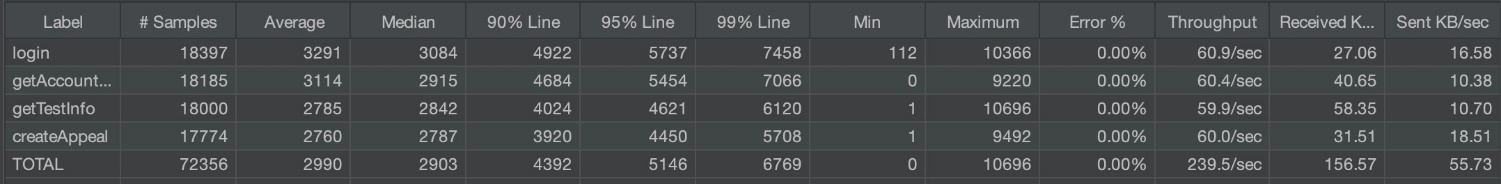


Рисунок 38. Тест с 800 пользователями

* + 1. 2000 пользователей

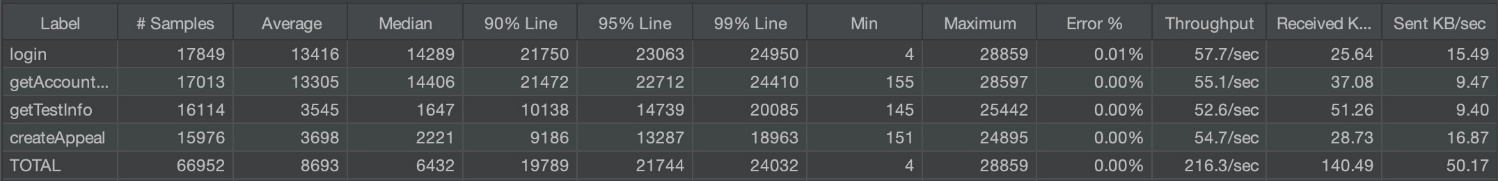


Рисунок 39. Тест с 2000 пользователями

* 1. Вывод

По данным тестам можем заметить, что средняя задержка ответа сервера увеличивается с увеличением количества одновременных пользователей. Так до 800 пользователей система отвечает в среднем за 3 секунды, что является допустимым значением, но при увеличении количества одновременных подключений до 2000 задержка увеличивается до 13 секунд, что уже является большой задержкой, но все же не критичной при пиковых нагрузках, так как предполагается, что такая нагрузка на сервер будет только в единичные моменты, либо при попытке атаки. Также можно заметить, что даже при условии большой задержки сервер обработал практически все запросы, провалив лишь одну сотую процента из семнадцати тысяч, что является примерно ста семьюдесятью восьми запросам.

# заключение

В ходе выполнения курсового проекта была спроектирована и разработана часть программной системы «Университет» для автоматизации взаимодействия университета с пользователями.

В процессе разработки данного курсового проекта были освоены такие технологии как React – построение пользовательского интерфейса, SpringBoot – создание веб приложений, Hiberante – взаимодействие с базами данных, PostgreSQL – система управления базами данных.

Было разработано клиент-серверное приложение, в котором было налажено сообщение с помощью HTTP-запросов. На клиенте для этого использовался fetch API и обертки над ним в виде библиотек axios и React Query. Для разработки графического интерфейса была применена библиотека ui-компонентов MaterialUI.

Также был изучен процесс разработки сложных систем на примере системы, позволяющей проводить онлайн тестирование студентов, получение заявок на обратную связь от пользователей и регистрацию с различными ролями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Липаев В.В. Проектирование программных систем. М.: Высш. шк, 1990.
2. Буч Г. Объектно-ориентированное проектирование / Пер. с анг. Конкорд, 1996.
3. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980
4. MySQL Documentation [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/>
5. React Documentation [Электронный ресурс]: Meta. – Режим доступа: <https://ru.reactjs.org/docs/getting-started.html>
6. Spring Boot Documentation [Электронный ресурс]: Pivotal Software – Режим доступа: <https://spring.io/projects/spring-boot>
7. Hibernate Documentation [Электронный ресурс]: Red Hat – Режим доступа: <https://hibernate.org/>

# ПРИЛОЖЕНИЕ А

**ИСХОДНЫЙ КОД КОНТРОЛЛЕРА КУРСОВ И МОДЕЛИ КУРСА**

@Component

@RequiredArgsConstructor

public class CourseServices {

private final AccountRepository accountRepository;

private final StudentRepository studentRepository;

private final CourseRepository courseRepository;

private final TestRepository testRepository;

private final QuestionRepository questionRepository;

private final AnswerRepository answerRepository;

public List<Course> getAllCourses() {

return courseRepository.findAll();

}

public List<Course> getAllCoursesWithoutTests() {

return courseRepository.findAll().stream().peek(course -> course.tests = null).toList();

}

public Course getCourse(String id, String login) {

var user = accountRepository.findFirstByLogin(login);

if (Objects.equals(user.role, "teacher")) {

return courseRepository.findFirstById(id);

} else {

Student student = studentRepository.findFirstByAccount\_Login(login);

Course course = courseRepository.findFirstById(id);

course.tests = course.tests

.stream()

.filter(test -> test.endDate.isAfter(LocalDate.now())

&& test.isAvailable

&& test.passedStudents.stream().noneMatch(s -> Objects.equals(s.id, student.id))

).toList();

return course;

}

}

public Course getCourseWithAvailableTest(String id, String login) {

Course course = courseRepository.findFirstById(id);

course.tests = course.tests.stream().filter(test -> test.endDate.isAfter(LocalDate.now()) && test.isAvailable).toList();

return course;

}

public Course getCourseWithAvailableTestWithStudent(String id, String login) {

Student student = studentRepository.findFirstByAccount\_Login(login);

Course course = courseRepository.findFirstById(id);

course.tests = course.tests

.stream()

.filter(test -> test.endDate.isAfter(LocalDate.now())

&& test.isAvailable

&& test.passedStudents.stream().noneMatch(s -> Objects.equals(s.id, student.id))

).toList();

return course;

}

public Test getTestById(String id) {

return testRepository.findById(Long.valueOf(id)).get();

}

public int getTestGrade(HashMap<String, List<String>> test, String login) {

var testId = test.get("testId").get(0);

var DBTest = testRepository.findById(Long.valueOf(testId)).get();

var student = studentRepository.findFirstByAccount\_Login(login);

var correctTest = new HashMap<String, List<String>>();

int correctQuestionCount = 0;

for (var question : DBTest.questions) {

correctTest.put(question.id.toString(), new ArrayList<>());

for (var answer : question.answers) {

if (answer.isCorrect) {

correctTest.get(question.id.toString()).add(answer.id.toString());

} else {

correctTest.get(question.id.toString()).add("");

}

}

if (correctTest.get(question.id.toString()).equals(test.get(question.id.toString()))) {

correctQuestionCount++;

}

}

student.account.score += Math.round((float) correctQuestionCount / correctTest.size() \* 100);

DBTest.passedStudents.add(student);

studentRepository.save(student);

testRepository.save(DBTest);

return student.account.score;

}

public Test addTestToCourse(CreateTestModel model) {

var course = courseRepository.findById(model.courseId).get();

var test = new Test(

course,

model.testTitle,

new ArrayList<>(),

new ArrayList<>(),

model.endDate,

model.isAvailable

);

var questions = new ArrayList<Question>();

var answers = new ArrayList<Answer>();

for (var question : model.questions) {

var newQuestion = new Question(

test,

question.question,

new ArrayList<>()

);

questions.add(newQuestion);

for (var answer : question.answers) {

answers.add(new Answer(

newQuestion,

answer.text,

answer.isCorrect

));

}

}

var newTest = testRepository.save(test);

questionRepository.saveAll(questions);

answerRepository.saveAll(answers);

return testRepository.findById(newTest.id).get();

}

public Test changeTestInfo(ChangeTestInfo model) {

var test = testRepository.findById(model.testId).get();

test.theme = model.theme;

test.endDate = model.endDate;

test.isAvailable = model.isAvailable;

return testRepository.save(test);

}

}

*@Data  
@Entity  
@Table*(name = "course")  
*@AllArgsConstructor  
@NoArgsConstructor  
public class* Course {  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *public* Long id;  
 *public* String title;  
 *public* String teacher;  
 *@OneToMany*(mappedBy = "course")  
 *public* List<Test> tests;  
  
 *public* Course(String title, String teacher, List<Test> tests) {  
 *this*.title = title;  
 *this*.teacher = teacher;  
 *this*.tests = tests;  
 }  
}

# ПРиЛОЖЕНИЕ Б

**ИСХОДНЫЙ КОД КОМПОНЕНТА «TEST», СЕРИАЛАЙЗЕРА И КОМПОНЕНТА «LOGIN»**

**КОМПОНЕНТ «TEST»:**

const Test = () => {

const { testid } = useParams()

const { data: test } = useTestById(testid as string)

const navigate = useNavigate()

function onsubmit (e: any) {

e.preventDefault()

const result = {

testId: [testid],

...serializer(e.target)

}

axios

.post('/api/courses/test', result)

.then(r => r.data)

navigate('/')

}

return (

<div className={style.wrapper}>

<h1 className={style.title\_\_title}>{test?.theme}</h1>

<form onSubmit={onsubmit}>

{test?.questions.map((item) =>

<fieldset key={`question${item.id}${item.question}`}>

<legend>{item.question}</legend>

{item.answers.map((answer) =>

<label key={answer.id}>

{answer.text}

<input type="checkbox" name={`${item.id}`} id={`question${item.id}\_${answer.id}`} value={answer.id}/>

</label>

)}

</fieldset>

)}

<input type="submit" value="Сдать"/>

</form>

</div>

)

}

**СЕРИАЛАЙЗЕР:**

export default function serialize(form) {

if (!form || form.nodeName !== "FORM") {

return;

}

let result = {};

for (let i = 0; i < form.elements.length - 1; i++) {

if (form.elements[i].name === "") continue

result[form.elements[i].name] = []

}

for (let i = 0; i < form.elements.length - 1; i++) {

if (form.elements[i].name === "") {

continue;

}

switch (form.elements[i].type) {

case 'checkbox':

case 'radio':

if (form.elements[i].checked) {

result[form.elements[i].name].push(encodeURIComponent(form.elements[i].value))

} else {

result[form.elements[i].name].push("")

}

break;

}

}

return result;

}

**КОМПОНЕНТЫ «LOGIN»:**

ype FormRegisterValueType = {

regLogin: string

regEmail: string

regPassword: string

regName: string

regSurname: string

regPatronymic: string

}

type FormLoginValueType = {

username: string

password: string

}

const Login = () => {

const navigate = useNavigate()

const {

register: loginRegister,

handleSubmit: loginHandleSubmit,

reset: loginReset

} = useForm<FormLoginValueType>()

const onLoginSubmit: SubmitHandler<FormLoginValueType> = async (data) => {

const formData = new FormData()

formData.append('username', data.username)

formData.append('password', data.password)

axios.post('/api/account/login', formData)

.then(() => {

loginReset()

fetchAccountData()

navigate('/')

})

.catch(() => {

UserStore.clear()

})

}

const {

register: registerRegister,

handleSubmit: registerHandleSubmit,

reset: registerReset,

setError: registerSetError

} = useForm<FormRegisterValueType>()

const onRegisterSubmit: SubmitHandler<FormRegisterValueType> = async (data) => {

const usersWithLogin = await axios.get('api/account/existslogin', { params: { login: data.regLogin } }).then(({ data }) => data)

const usersWithEmail = await axios.get('api/account/existsemail', { params: { email: data.regEmail } }).then(({ data }) => data)

if (!(usersWithEmail) || !usersWithLogin) {

registerSetError('regLogin', {

type: 'custom',

message: 'userLogin is using'

})

} else {

console.log(data)

await axios.post('/api/account/register', {

login: data.regLogin,

email: data.regEmail,

password: data.regPassword,

name: data.regName,

surname: data.regSurname,

patronymic: data.regPatronymic

})

registerReset()

navigate('/')

}

}

return (

<div className={style.wrapper}>

<div className={style.radioButtons}>

<input type="radio" name="logReg" id={style.login} defaultChecked={true}/>

<label htmlFor={style.login}

id={style.loginLabel}

onClick={() => {

registerReset()

loginReset()

}}>Вход

</label>

<input type="radio" name="logReg" id={style.register}/>

<label htmlFor={style.register}

id={style.registerLabel}

onClick={() => {

registerReset()

loginReset()

}}>Регистрация

</label>

</div>

<div className={style.loginForm}>

<form onSubmit={loginHandleSubmit(onLoginSubmit)}>

<input type="text"

className={style.input}

{...loginRegister('username', { required: true })}

placeholder={'Логин'}

autoComplete={'off'}/>

<input type="password"

className={style.input}

{...loginRegister('password', { required: true })}

placeholder={'Пароль'}

autoComplete={'off'}/>

<button type="submit" className={style.submit}>Войти</button>

</form>

</div>

<div className={style.registerForm}>

<form onSubmit={registerHandleSubmit(onRegisterSubmit)}>

<input type="text"

className={style.input}

{...registerRegister('regLogin', { required: true })}

placeholder={'Логин'}

autoComplete={'off'}/>

<input type="email"

className={style.input}

{...registerRegister('regEmail', { required: true })}

placeholder={'Почта'}

autoComplete={'off'}/>

<input type="password"

className={style.input}

{...registerRegister('regPassword', { required: true })}

placeholder={'Пароль'}

autoComplete={'off'}/>

<input type="text"

className={style.input}

{...registerRegister('regName', { required: true })}

placeholder={'Имя'}

autoComplete={'off'}/>

<input type="text"

className={style.input}

{...registerRegister('regSurname', { required: true })}

placeholder={'Фамилия'}

autoComplete={'off'}/>

<input type="text"

className={style.input}

{...registerRegister('regPatronymic', { required: true })}

placeholder={'Отчество'}

autoComplete={'off'}/>

<button type="submit" className={style.submit}>Зарегистрироваться</button>

</form>

</div>

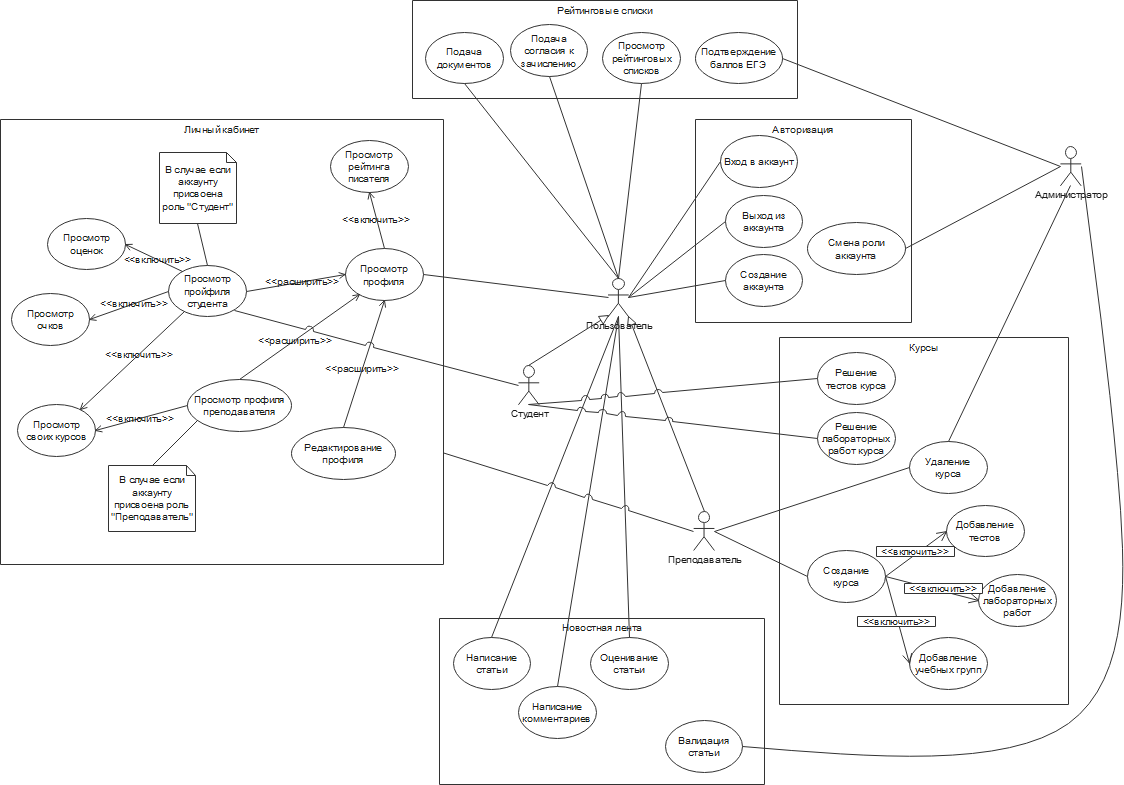
</div>

)

}

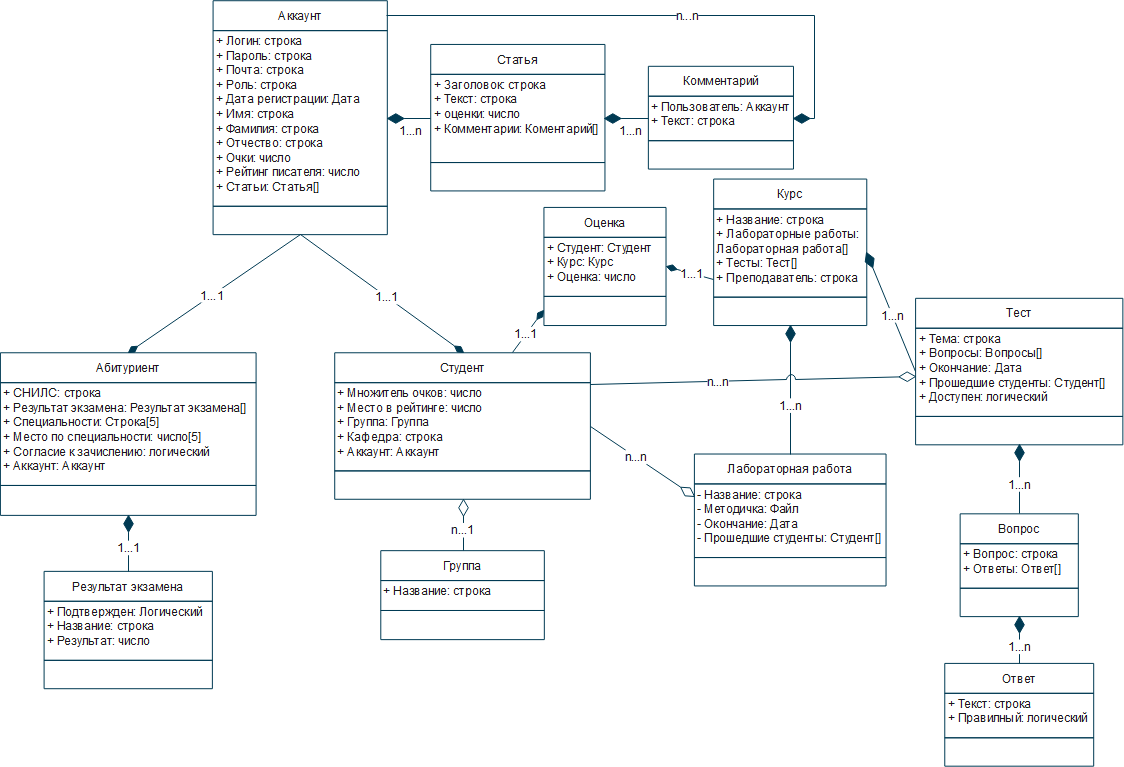
# Приложение В

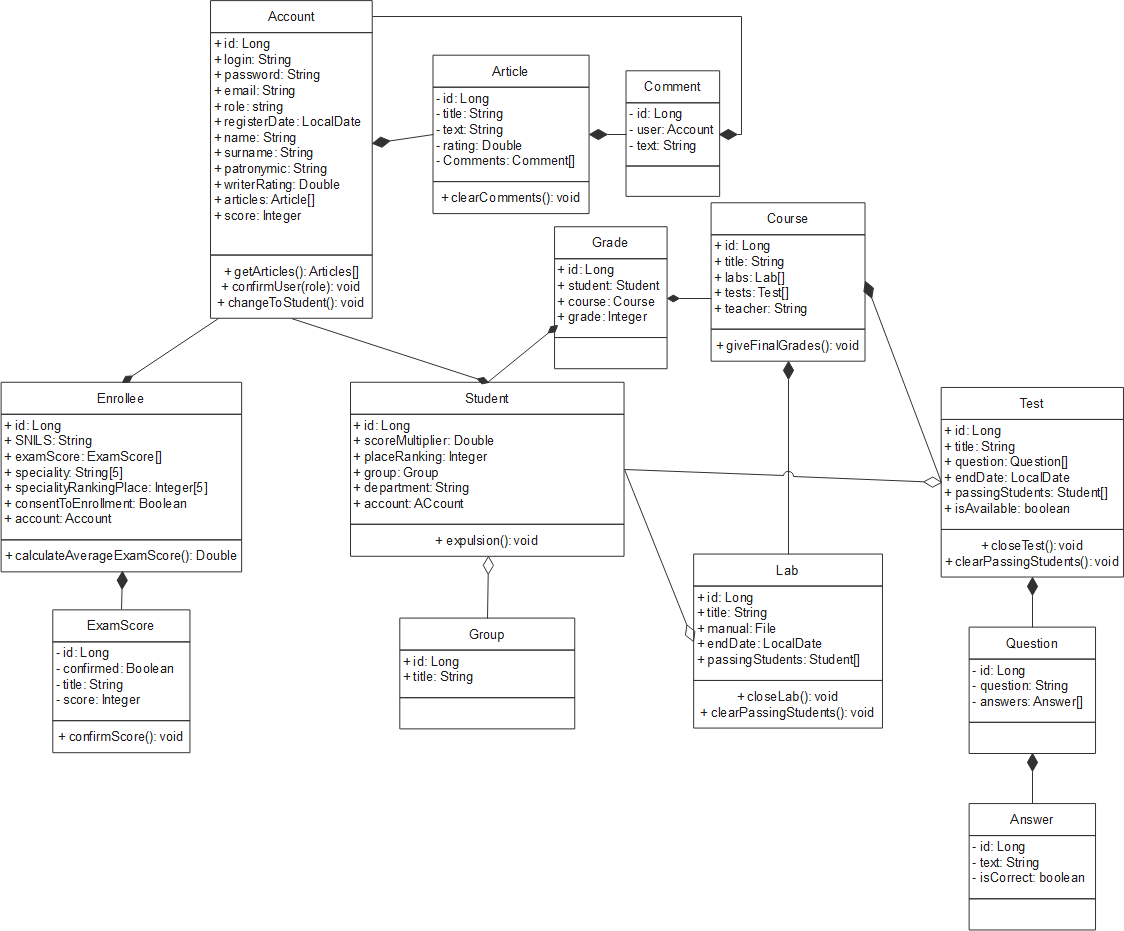
**ПОЛНАЯ ДИАГРАММА ПРЕЦЕНДЕНТОВ**



# пРИЛОЖЕНИЕ г

**КОНЦЕПТУАЛЬНАЯ ДИАГРАММА КЛАССОВ И ДИАГРАММА КЛАССОВ УРОВНЯ РЕАЛИЗАЦИИ**





# пРИЛОЖЕНИЕ Д

**ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТЕЙ ПОДАЧИ ДОКУМЕНТОВ АБИТУРИЕНТОМ**

