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 программирования"
на тему

Проектирование и разработка программной системы
информационной системы «Университет»

Выполнил: *Грачев Д. А.*

Принял: *доц. кафедры ИСПИ*
Верешинин В.В.

Владимир, 2022

Календарные сроки выполнения

1. Согласование тем курсовых проектов, выдача и оформление заданий.
2. Анализ предметной области, изучение структуры и функций программной системы.
3. Моделирование структуры объектов предметной области и их взаимодействия на концептуальном уровне (уровне предметной области). Диаграмма прецедентов и классов (дата представления итогов 1 этапа: 07.10.22).
4. Динамическое моделирование программной системы. Формирование диаграмм состояний, последовательностей (сотрудничества), видов деятельности (дата представления итогов 2 этапа: 04.11.22).
5. Реализация программной системы, составление тестов. Сохранение результатов работы в репозиторий проекта. Демонстрация работоспособности системы (дата представления итогов 3 этапа: 02.12.22).
6. Составление пояснительной записки и оформление графического материала.
7. Сдача курсовых работ и их проверка. Защита.

Дата выдачи задания "23" сентября 2022 г.

Задание получил

Руководитель проектирования: Вершинин В.В.

Ход выполнения

Дата										
Процент выполнения										
Подпись руководителя										

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
"Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых"

Кафедра "Информационные системы и программная инженерия"

УТВЕРЖДАЮ:

Зав. кафедрой

ЗАДАНИЕ

на курсовой проект (работу)

по дисциплине Технологии программирования

Студент 3 курса института ИТиР

группы ПРИ-120 тов. Братев Даниил Алексеевич

Тема проекта Университет

Срок сдачи законченного проекта "16" декабря 2022 г.

Исходные данные

1. Методология проектирования и разработки RUP
2. Язык моделирования UML
3. Платформа разработки ASP.NET
4. Функции системы: различные типы пользователей; рейтинговые списки; курсы с тестами и оценками; лента новостей; справочник по специальностям
5.

Разработать следующие вопросы

1. Выполнить исследование и анализ предметной области
2. Разработать прототип ПС
3. Выполнить моделирование работы ПС
4. Разработать схему
5. Реализовать ПС с использованием выбранных средств и технологий
6. Оформить отчёт по работе в виде ПЗ согласно требованиям

Конструктивно разработать (вычертить)

1. Диаграмма прецедентов
2. Диаграмма состояний
3.

Рекомендуемая литература

1. Липаев В.В. Проектирование программных систем. М.: Высш. шк, 1990.
2. Буч Г. Объектно-ориентированное проектирование / Пер. с анг. Конкорд, 1996.
3. Эндрю Троелсен, Филипп Джепикс Язык программирования C# 6.0 и платформа .NET 4.6 / М.: Вильямс, 2016. - 1440 с.
4. Джозеф Албахари, Бен Албахари C# 6.0. Справочник. Полное описание языка / М.: Вильямс, 2018. – 1040 с.
5. Шмуллер Дж. Освой самостоятельно UML. 2-е изд. М.: изд. Дом «Вильямс», 2002
6. Калберстон Р., Браун К., Кобб Г. Быстрое тестирование М.: изд. Дом «Вильямс», 2002
7. Кармайкл Э., Хейвуд Д. Быстрая и качественная разработка программного обеспечения. М.: изд. дом «Вильямс», 2003

АННОТАЦИЯ

В данном курсовом проекте производилось проектирование и разработка программной системы «Университет». 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.

Реализованная система содержит 9 сущностей, представляющих данные о пользователях системы, с возможностью присваивания различных ролей, о курсах, содержащих тесты, и о прочей информации, с которой может взаимодействовать пользователь. В рамках выполнения курсового проекта был реализован функционал личного кабинета и подсистемы курсов.

In this course project, the design and development of the software system "University" was carried out. 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, the implementation of a software system containing the server part of the system and the user interface.

The implemented system contains 9 entities that represent data about users of the system, with the possibility of assigning different roles, about courses containing tests, and about other information that the user can interact with. As part of the implementation of the course project, the functionality of the personal account and the course subsystem was implemented.

Курсовой проект представлен на 36 страницах, рисунков – 22, таблиц – 3, использованных источников – 7, приложений – 5.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В ходе выполнения курсового проекта требуется спроектировать и разработать программную систему «Университет» для автоматизации взаимодействия университета с пользователями.

Цель работы: изучение процесса разработки сложных систем на примере программной системы, позволяющей проводить онлайн тестирование студентов, получать обратную связь от пользователей и регистрировать различных пользователей в системе под разными ролями

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
3. Разработать серверную часть приложения, которая будет являться посредником между интерфейсом пользователя и хранилищем данных

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

В университете, обучающем студентов и осуществляющем набор абитуриентов, важно иметь программную систему, обеспечивающую:

- автоматизацию рейтинговых списков абитуриентов
- сдачу лабораторных работ в электронной форме и контрольных работ в онлайн формате с автоматическим указанием баллов
- новостную ленту с возможностью написания статей любым пользователем программной системы

1.2. Требования в системе

Функциональные требования:

- Зарегистрировать аккаунт
- Войти в аккаунт
- Изменить данные о пользователе
- Пройти тест
- Выйти из аккаунта
- Присвоить пользователю роль
- Оставить заявку на обратную связь
- Отображение информации о ближайшем дне открытых дверей

Нефункциональные требования:

- Реализация серверной части на .NET с использованием технологии ASP.NET
- Взаимодействие интерфейса и сервера с помощью API
- Реализация интерфейса с помощью библиотеки React.js
- Использование СУБД MySQL
- Использование Entity Framework для работы с данными

1.3. Роли в разрабатываемой системы

1.3.1. Администратор – сотрудник университета, который администрирует работу программной системы, подтверждает некоторые действия пользователей

1.3.2. Пользователь – зарегистрированный участник системы, который может полностью взаимодействовать с подсистемой новостей и подавать документ на поступление в университет

1.3.3. Студент – зарегистрированный участник системы, который владеет всеми возможностями обычного пользователя, так же может взаимодействовать с подсистемой курсов

1.3.4. Преподаватель – зарегистрированный участник системы, который обладает всеми возможностями обычного пользователя, так же может взаимодействовать с подсистемой курсов

1.4. Словарь предметной области

1.4.1. *Курс* – часть программной системы, с помощью которой «Преподаватель» может создавать задания для «Студента», такие как контрольные работы

1.4.2. *Новостная лента* – часть программной системы, обеспечивающая возможность написания новостей любым пользователем, так же любой прочитавший новость может поставить оценку прочитанному материалу и оставить комментарий

1.4.3. *Рейтинговые списки* – часть программной системы, обеспечивающая автоматизацию работы с абитуриентами

1.4.4. *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

1.4.5. *Студент* – роль пользователя в системе, являющегося студентом в университете, может выполнять тесты и лабораторные работы с курса

1.4.6. *Преподаватель* – роль пользователя в системе, являющегося преподавателем в университете, может создавать и редактировать курсы

1.4.7. *Тест* – часть курса, задание, в ходе которого студенты должны отвечать на поставленные вопросы с данными вариантами ответов

1.4.8. *Группа* – выделенная часть студентов, объединённая одной специальностью

1.4.9. *Обращение* – заполненная пользователем форма обратной связи

1.5. 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

1.6.Начальная оценка и выделение сущностей

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

2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА СИСТЕМЫ

2.1.Концептуальный уровень

2.1.1. Диаграмма прецендентов

На данной диаграмме отображены все прецеденты 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 и прикрепить студентов. 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).

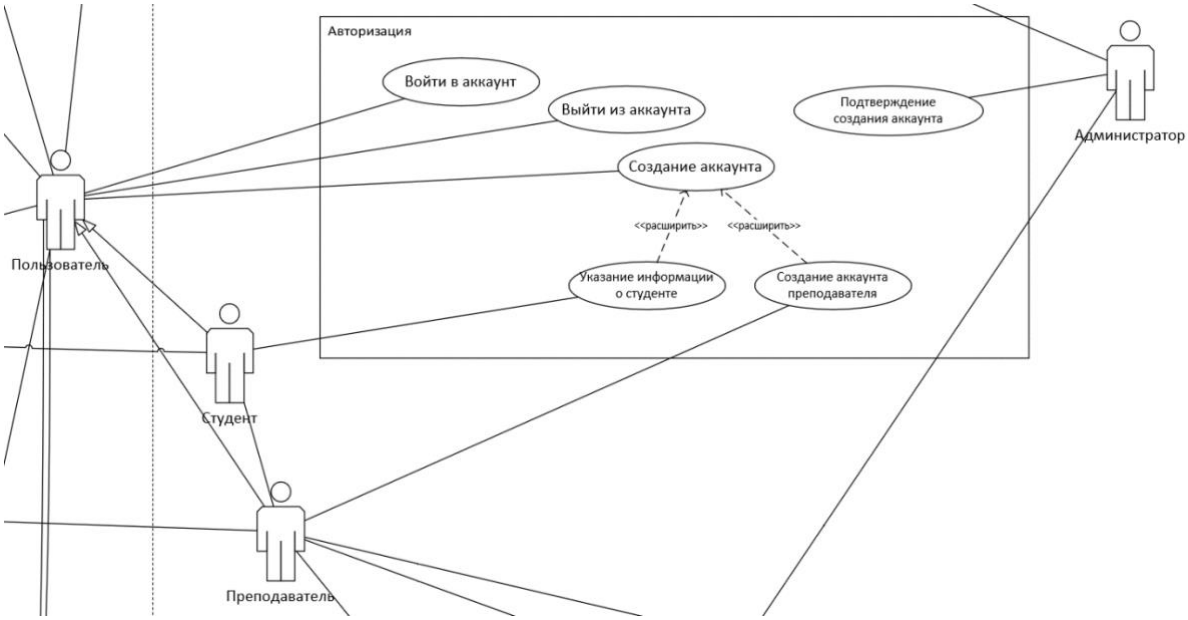


Рисунок 1. Диаграмма прецендентов - Подсистема "Авторизация"

Полная версия диаграммы прецендентов представлена в приложении В.

2.1.2. 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 пустая строка, 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

2.1.3. 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 диаграммы классов, показывающая отношения между объектами «Пользователь» и объектами подсистемы «Новости»(рис. 2).

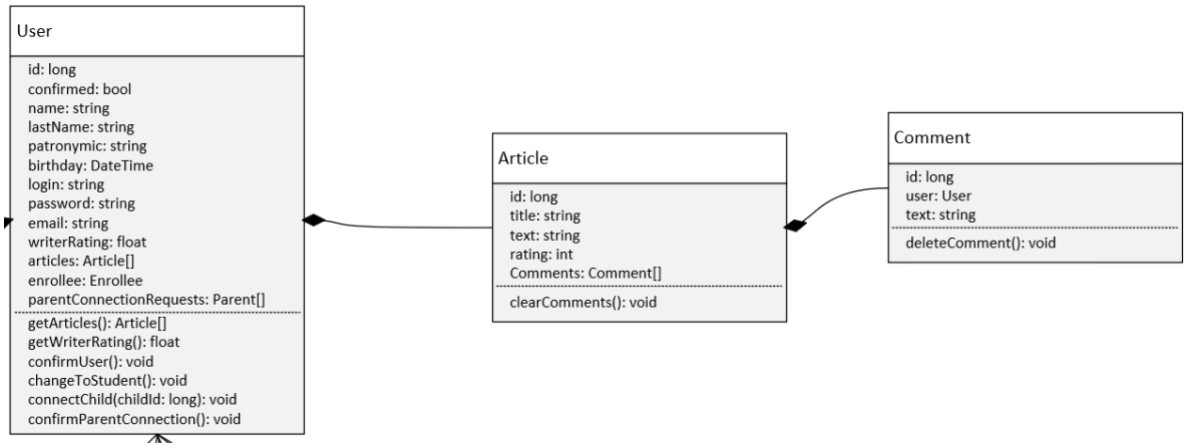


Рисунок 2. Диаграмма классов - Отношение пользователя с подсистемой "Новости"

Полная версия диаграммы классов представлена в приложении Г.

2.1.4. 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. 3). На ней представлены различные состояния доступности теста в зависимости от действий пользователей разных ролей.

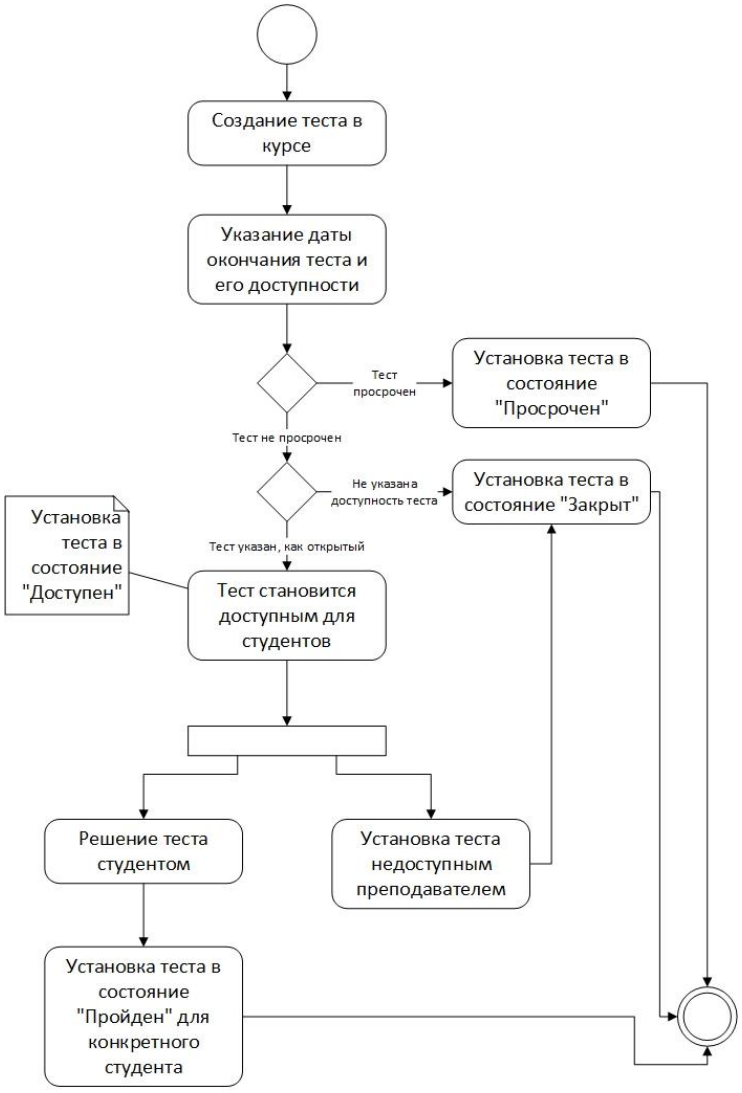


Рисунок 3. Диаграмма состояний - Тест

В приложении Д также представлена диаграмма состояния объекта «Абитуриент»

2.1.5. 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 подтверждения результатов экзамена у абитуриента, попадания его в рейтинговые списки и последующее зачисление(рис. 4)

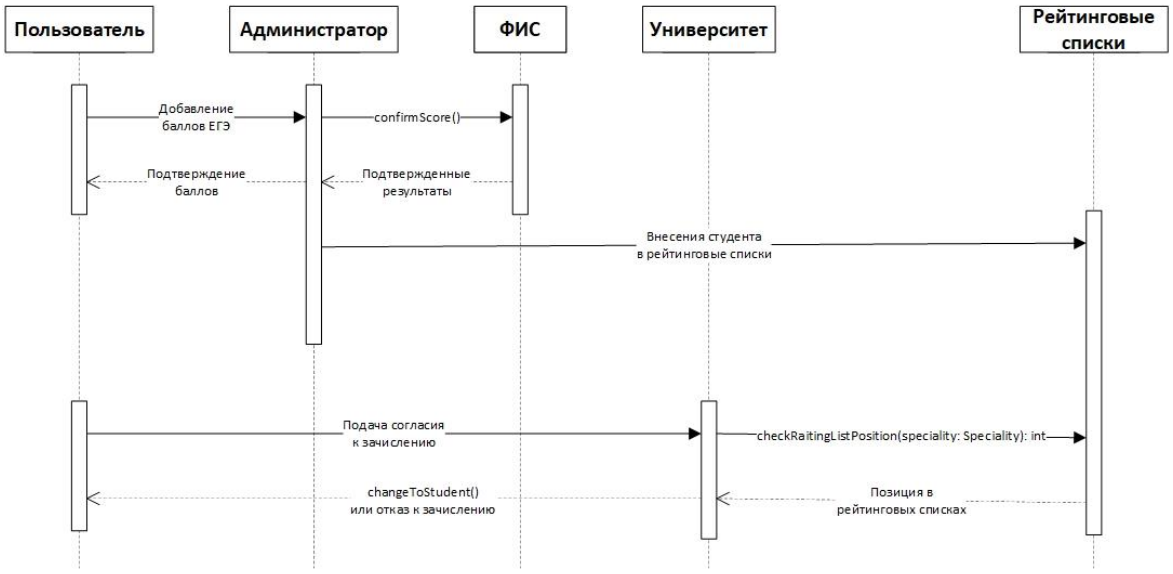


Рисунок 4. Диаграмма последовательностей - Абитуриент и рейтинговые списки

2.1.6. Диаграмма видов деятельности

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

На представленном ниже рисунке показана диаграмма видов деятельности для возможных действий пользователя(рис. 5)

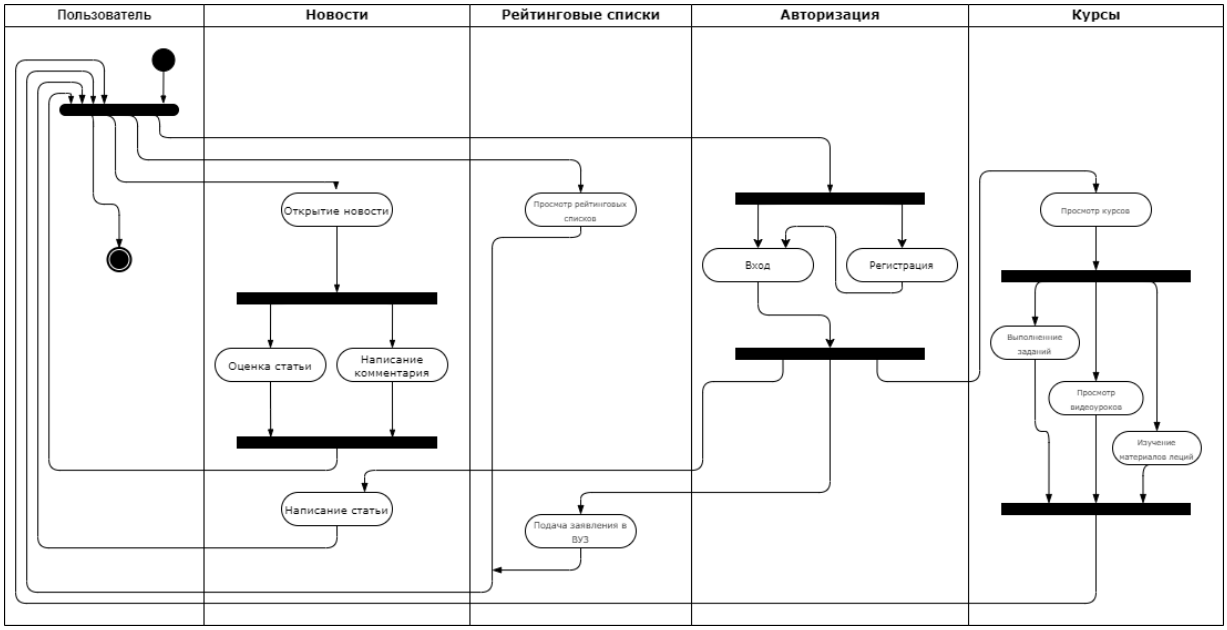


Рисунок 5. Диаграмма видов деятельности

2.1.7. Общие принципы организации системы

Разрабатываемая система представляет собой веб-приложение, построенное на базе архитектуры ASP.NET CORE 6 по шаблону «веб-API».

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

Каждый контроллер относится к своему классу модели и содержит необходимые методы, которые реализуют бизнес-функции и в результате своей работы возвращают данные для отправки их на интерфейс.

Для реализации моделей и применения подхода Model First использован Entity Framework.

2.2. База данных

В качестве системы управления базой данных была выбрана MySQL – свободная реляционная система управления базами данных, разрабатываемая и поддерживаемая корпорацией Oracle.

На представленном ниже рисунке показана логическая схема базы данных (рис. 6). 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.

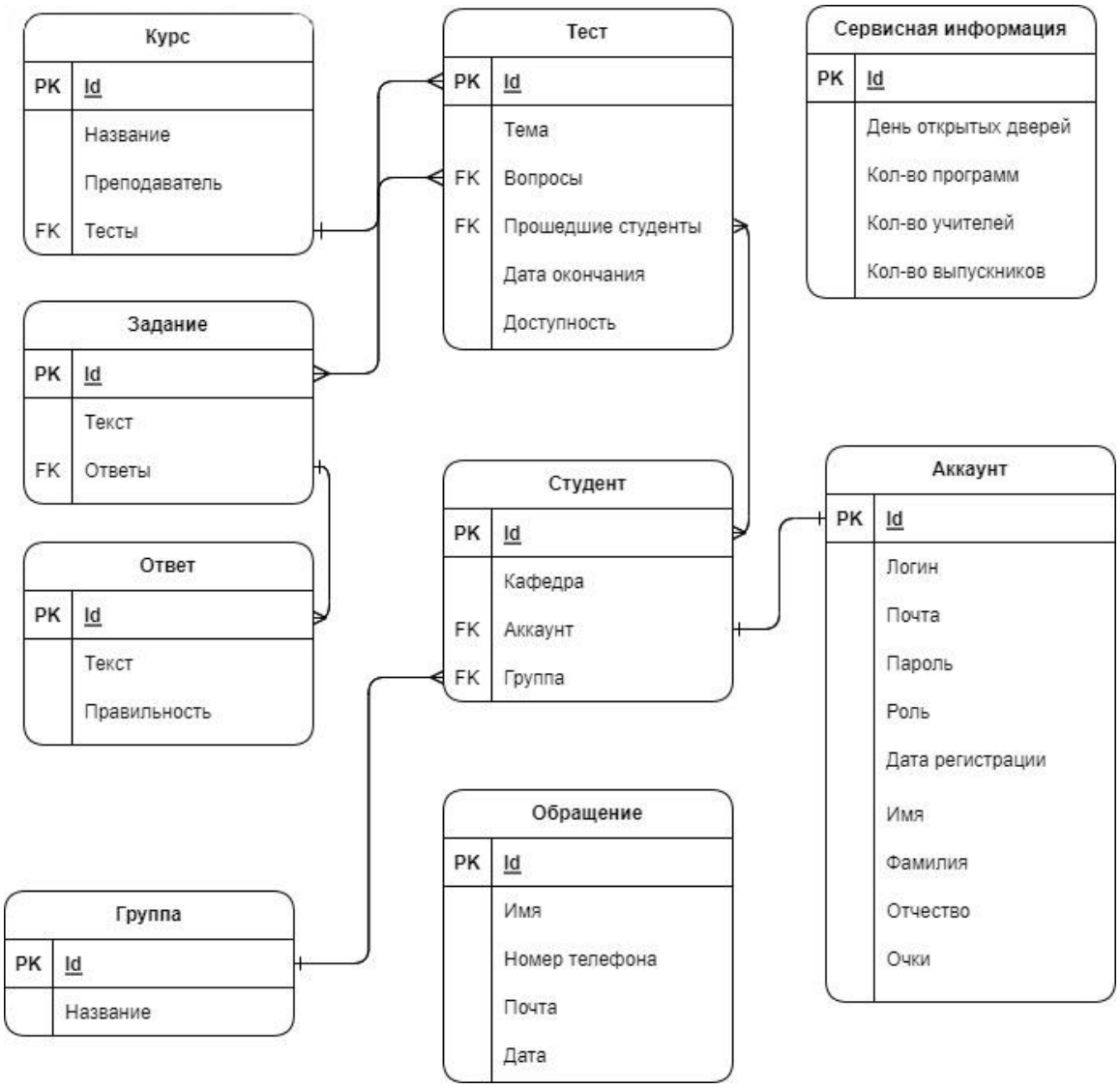


Рисунок 6. Логическая схема базы данных

2.3.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

2.3.1. Реализация маршрутизации

С помощью подключения дополнительной библиотеки “react-router-dom” можно настроить маршрутизацию в приложении(рис. 7)

```
<Router>
  <ScrollToTop/>
  <Routes>
    <Route path={PATHS.MAIN} element={<Layout/>}>
      <Route index element={<Main/>}/>
      <Route path={PATHS.LOGIN} element={<Login/>}/>
      <Route path={PATHS.PROFILE+"/:login"} element={<Profile/>}/>
      <Route path={PATHS.COURSE+"/:id"} element={<Course/>}/>
      <Route path={PATHS.TESTS+"/:testid"} element={<Test/>}/>
      <Route path="*" element={<NotFound/>}/>
    </Route>
  </Routes>
</Router>
```

Рисунок 7. Маршруты приложения

Также для удобства разработки пути были вынесены в перечисляемый тип enum(рис. 8)

```
enum PATHS {  8+ usages  Daniil *
  MAIN = "/",  2 usages  Daniil
  OPEN_DOORS = "/open-doors",  1 usage  Daniil
  NEWS = "/news",  2 usages  Daniil
  TEST = "/mainpagetest",  1 usage  Daniil
  WORLD_IT = "/world-it",  1 usage  Daniil
  PROGRAMS = "/programs",  2 usages  Daniil
  EVENTS = "/events",  1 usage  Daniil
  CAREER = "/career",  1 usage  Daniil
  LOGIN = "/login",  2 usages  Daniil
  COURSE = "/course",  2 usages  Daniil
  TESTS = "/test",  2 usages  Daniil
  PROFILE = "/user",  2 usages  Daniil
  SCORE_INFO = "/score-info",  1 usage  new *
}
```

Рисунок 8. Перечисление маршрутов приложения

2.3.2. Главная страница приложения

Вверху главной страницы приложения мы можем увидеть три составные части(рис. 9). В самом верху располагается навигационная панель приложения, при нажатии на элементы которой будет происходить переход на другие страницы(рис. 9-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. 9-б). Внизу рисунка отображается информация о университете, в данной области можно увидеть количество программ, преподавателей и выпускников в университете(рис. 9-в). Немного ниже этого так же можно увидеть информацию о ближайшем дне открытых дверей.

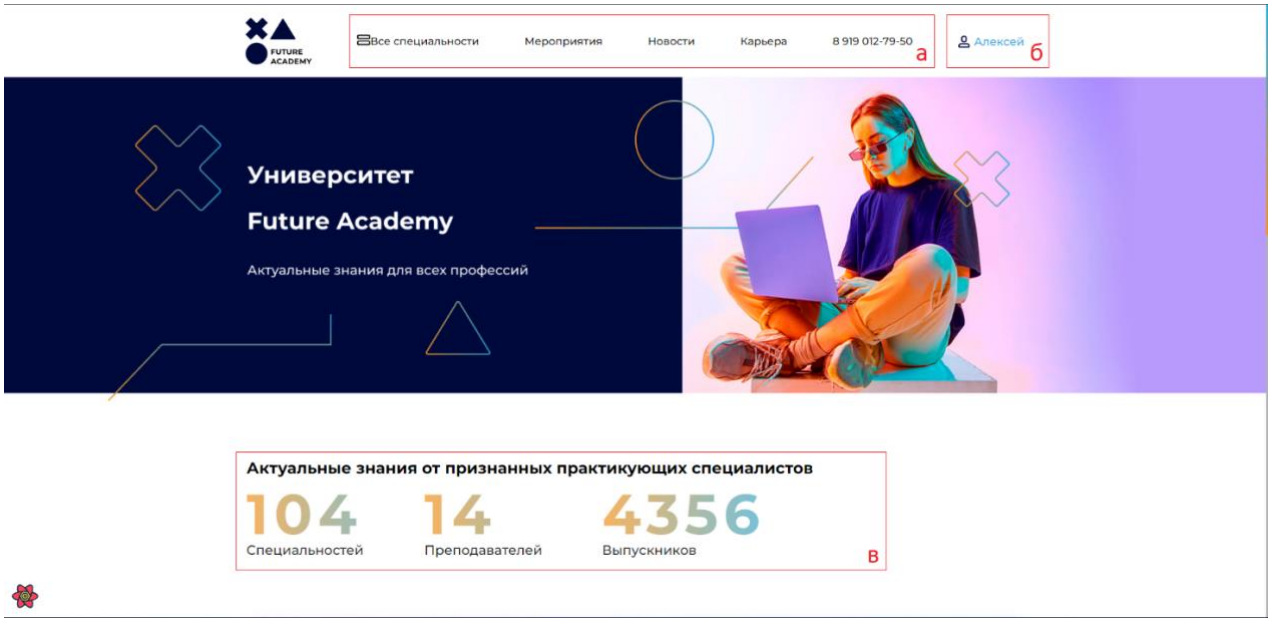


Рисунок 9. Главная страница приложения(а – навигация, б – метка пользователя, в – информация о университете)

2.3.3. Профиль пользователя

При открытии профиля пользователя мы увидим разделение на два блока: левый и правый(рис. 10). 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. 10-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. 10-б).

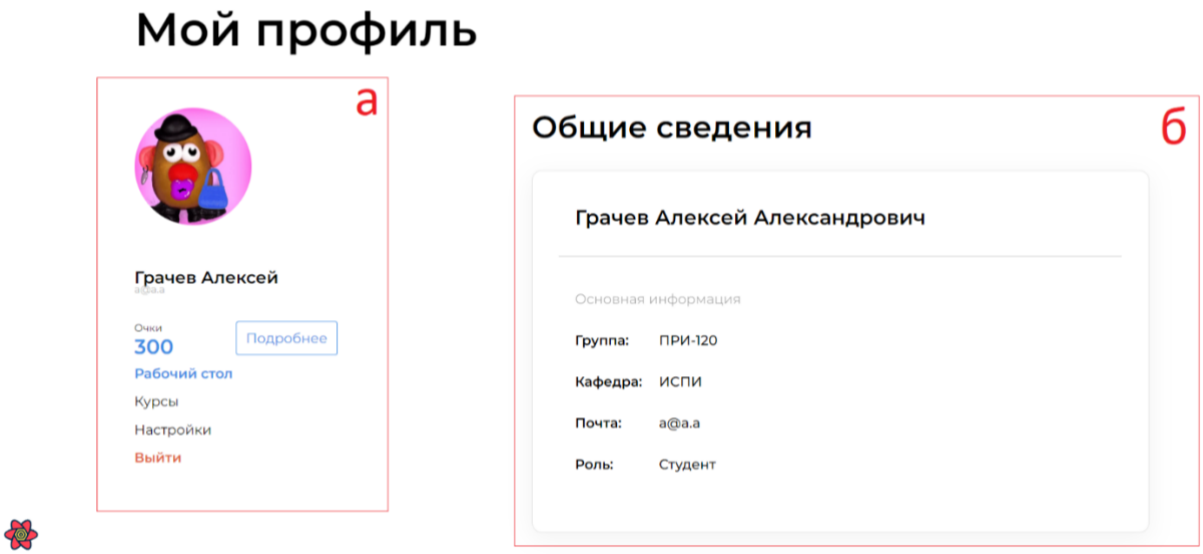


Рисунок 10. Профиль пользователя(а – краткая информация, б – вкладки)

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.

2.3.4. 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. 11).

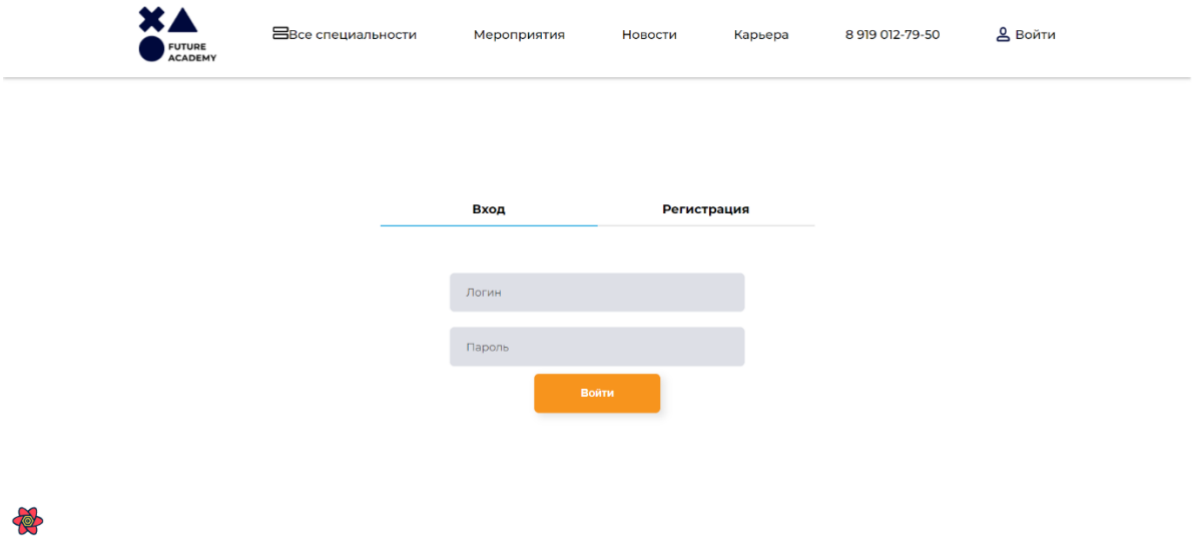


Рисунок 11. Страница входа/регистрации

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 по алгоритму SHA1.

В целях ознакомления на рисунке ниже представлен хешированный вид паролей, хранящихся в базе данных(рис. 12)

id	login	password
1	chase	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
2	student1	9d4e1e23bd5b727046a9e3b4b7db57bd8d6ee684
4	user1	9d4e1e23bd5b727046a9e3b4b7db57bd8d6ee684
NULL	NULL	NULL

Рисунок 12. хешированные пароли

2.3.5. 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. 13).

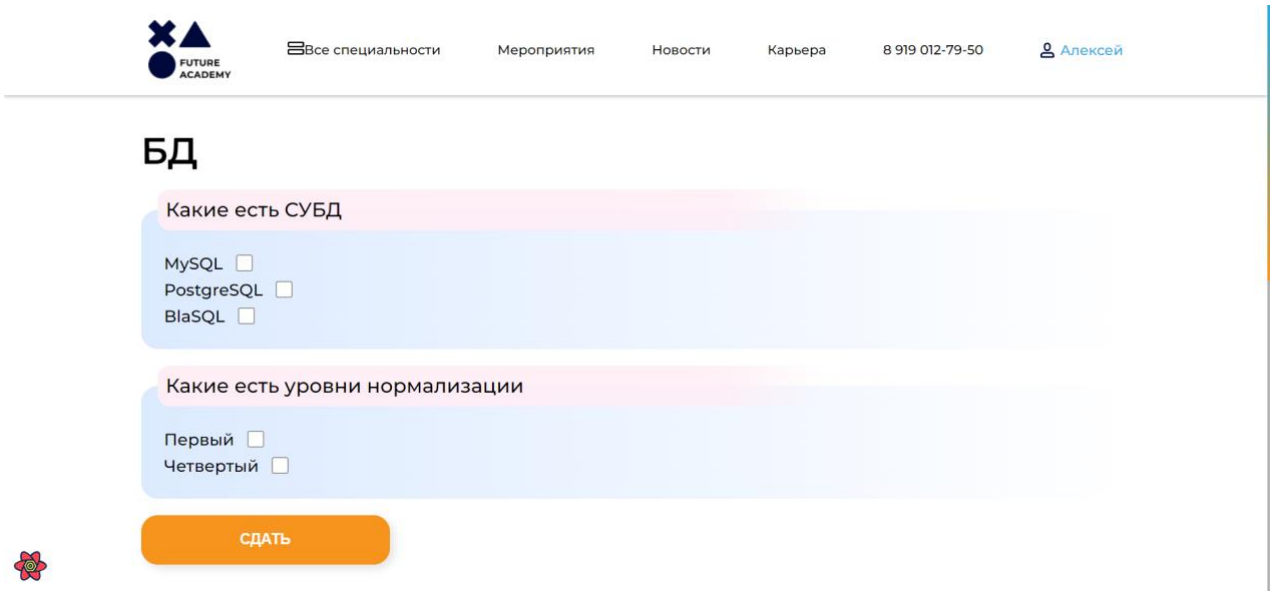


Рисунок 13. Страница теста

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 нужным выбрать, 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. 14)

```
export default function serialize(form) {  2 usages  Daniil *
  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;
}
```

Рисунок 14. Сериалайзер формы

Про оценивание тестов будет описано в пункте *2.4.3. Оценка теста*

2.3.6. 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. 15). 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 *2.4.2. Реализация конечных точек API*). В данном примере, после получения данных, они записываются в “стейт” компонента.

```
React.useEffect( effect: () => {
  axios.get( url: `/api/admin/usersWithoutRole`).then(({data}) => setUsers(data));
}, deps: [])
```

Рисунок 15. GET-запрос с помощью axios

2.3.7. Панель администратора

В панели администратора присутствует единственный интерактивный элемент – это таблица со списком новых пользователей, в которой имеется информация о логине и дате регистрации, также имеется выпадающее меню с возможностью выбрать роль для нового пользователя и кнопкой «Присвоить»(рис. 16).

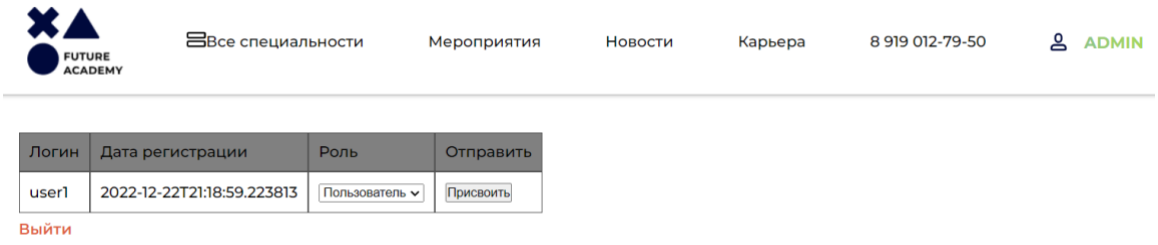


Рисунок 16. Панель администратора

2.4.Серверная часть

Для реализации серверной части была выбрана технологии .NET – это модульная платформа для разработки программного обеспечения с открытым

исходным кодом, выпущенная компанией Microsoft. Выбран шаблон ASP.NET Core web-api, который позволяет быстро и удобно настроить API своего приложения.

2.4.1. Модели сущностей в системе и подключение к базе данных

В системе шестнадцать моделей данных, 9 из которых относятся к базе данных, остальные же описывают данные получаемые и отправляемые в интерфейс пользователя(рис. 17)

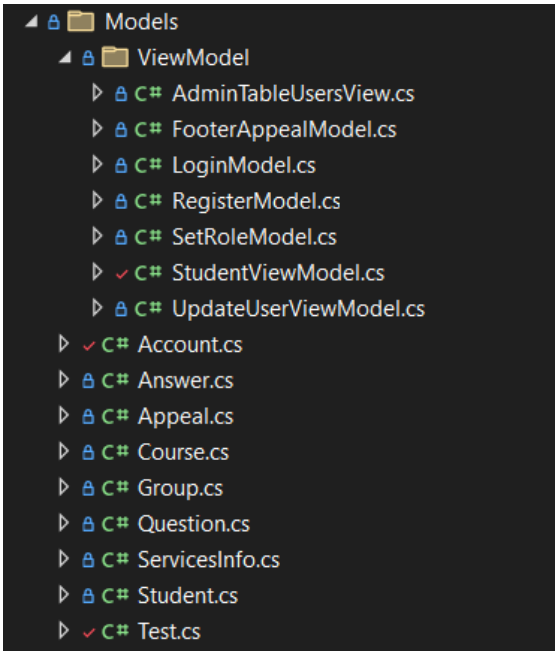


Рисунок 17. Модели в системе

Те из показанных моделей, которые относятся к базе данных, указаны в контексте взаимодействия с базой данных(рис. 18), к которой происходит подключение с помощью строки подключения, указанной в файле настроек приложения(рис. 19), с использованием этих данных в файле, являющимся входной точкой в программу происходит подключение к базе данных и взаимодействие с ней(рис. 20)

```
Ссылок: 18
public class UniversityContext : DbContext
{
    Ссылок: 0
    public UniversityContext(DbContextOptions<UniversityContext> options) : base(options) { }
    Ссылок: 1
    public DbSet<ServicesInfo> ServicesInfo { get; set; }
    Ссылок: 11
    public DbSet<Account> Accounts { get; set; }
    Ссылок: 1
    public DbSet<Appeal> Appeals { get; set; }
    Ссылок: 1
    public DbSet<Group> Groups { get; set; }
    Ссылок: 5
    public DbSet<Student> Students { get; set; }
    Ссылок: 0
    public DbSet<Answer> Answers { get; set; }
    Ссылок: 0
    public DbSet<Question> Questions { get; set; }
    Ссылок: 2
    public DbSet<Test> Tests { get; set; }
    Ссылок: 4
    public DbSet<Course> Courses { get; set; }
}
```

Рисунок 18. Контекст базы данных

```
"ConnectionStrings": {
  "defaultConnection": "server=localhost;user=root;password=pass;database=future_university;"
}
```

Рисунок 19. Строка подключения к базе данных

```
builder.Services.AddDbContext<UniversityContext>(
    opt => opt.UseMySql(
        builder.Configuration.GetConnectionString("defaultConnection"),
        new MySqlServerVersion(new Version(8, 0, 31)))
    );
```

Рисунок 20. Подключение к базе данных

Далее с помощью Entity Framework мы можем добавить миграцию 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.

2.4.2. Реализация конечных точек API

В данном пункте будет описана обработка GET-запроса “отправленного” из пункта *2.3.6. 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. 21-а) находят нужный метод обработки запроса, у которого так же может быть указан маршрут(относительно маршрута контроллера)(рис. 21-б).

```
[ApiController]
[Route("api/admin")]  а
Ссылок: 1
public class AdminController : ControllerBase
{
    private readonly UniversityContext _context;

    Ссылок: 0
    public AdminController(UniversityContext context)
    {
        _context = context;
    }

    [HttpGet("usersWithoutRole")]  б
    Ссылок: 0
    public List<AdminTableUsersView> getAllUsersWithoutRole()
    {
        var users = _context.Accounts
            .Where(u => u.role == null)
            .Take(15)
            .ToList();                          в

        List<AdminTableUsersView> adminViewUsers = new List<AdminTableUsersView>();

        foreach (Account user in users) adminViewUsers.Add(
            new AdminTableUsersView { login = user.login, registerDate = user.registerDate }
            );

        return adminViewUsers;
    }
```

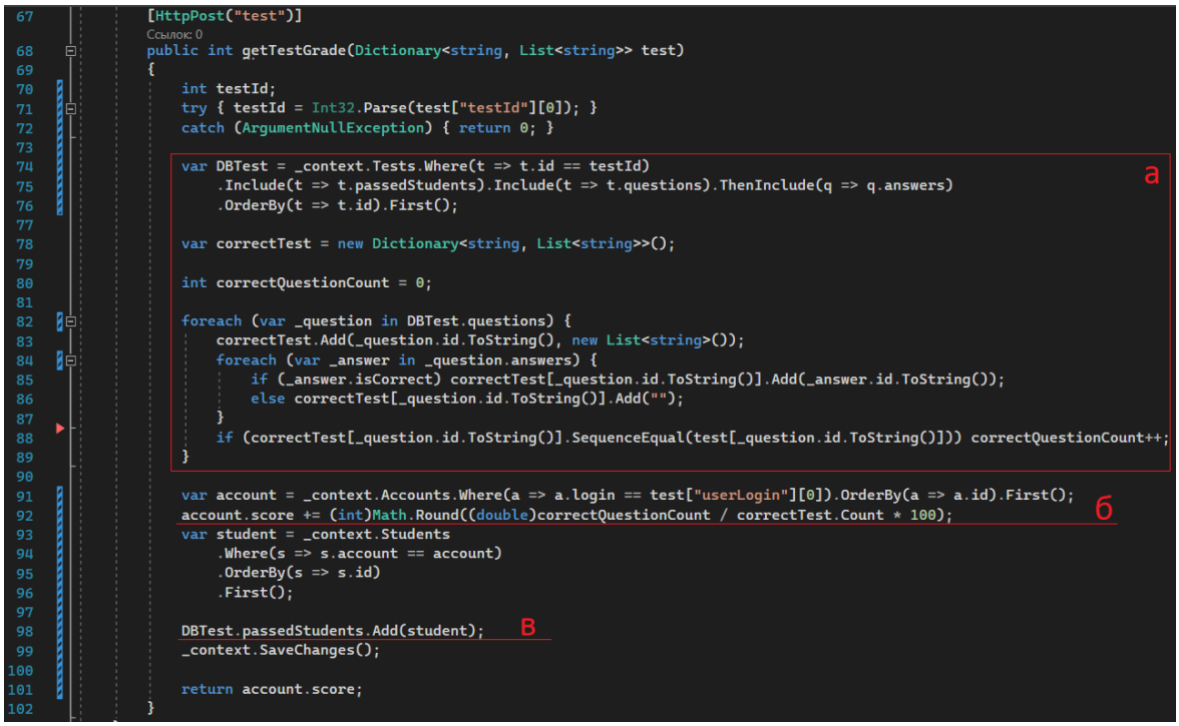
Рисунок 21. API-контроллер (а – маршрут контроллера, б – маршрут метода, в – взаимодействие с базой данных)

Когда “сервер” определил метод, который должен обработать запрос, он обращается к базе данных, чтобы взять у нее запрашиваемые данные, как в это примере(рис. 21-в).

2.4.3. Оценка теста

После действий, описанных в пункте *2.3.5. 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(рис. 22-а). После, из этого числа высчитывается балл по стобалльной системе, который прибавляется к очкам пользователя и позже возвращается клиенту(рис. 22-б). Также в этом методе происходит добавление студента в список сдавших тест, чтобы он не мог пройти тест дважды(рис. 22-в)




```
[HttpPost("test")]
Ссылок: 0
public int getTestGrade(Dictionary<string, List<string>> test)
{
    int testId;
    try { testId = Int32.Parse(test["testId"][0]); }
    catch (ArgumentNullException) { return 0; }

    var DBTest = _context.Tests.Where(t => t.id == testId)
        .Include(t => t.passedStudents).Include(t => t.questions).ThenInclude(q => q.answers)
        .OrderBy(t => t.id).First();

    var correctTest = new Dictionary<string, List<string>>();

    int correctQuestionCount = 0;

    foreach (var _question in DBTest.questions) {
        correctTest.Add(_question.id.ToString(), new List<string>());
        foreach (var _answer in _question.answers) {
            if (_answer.isCorrect) correctTest[_question.id.ToString()].Add(_answer.id.ToString());
            else correctTest[_question.id.ToString()].Add("");
        }
        if (correctTest[_question.id.ToString()].SequenceEqual(test[_question.id.ToString()])) correctQuestionCount++;
    }

    var account = _context.Accounts.Where(a => a.login == test["userLogin"][0]).OrderBy(a => a.id).First();
    account.score += (int)Math.Round((double)correctQuestionCount / correctTest.Count * 100);
    var student = _context.Students
        .Where(s => s.account == account)
        .OrderBy(s => s.id)
        .First();

    DBTest.passedStudents.Add(student);
    _context.SaveChanges();

    return account.score;
}
```

Рисунок 22. Метод обработки теста

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, ASP.NET Core – создание веб приложений, Entity Framework – взаимодействие с базами данных, MySQL – 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.

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]: Oracle. – Режим доступа: https://dev.mysql.com/doc/
5. React Documentation [Электронный ресурс]: Meta. – Режим доступа: https://ru.reactjs.org/docs/getting-started.html
6. ASP.NET Core Documentation [Электронный ресурс]: Microsoft – Режим доступа: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0
7. Entity Framework Documentation [Электронный ресурс]: Microsoft – Режим доступа: https://learn.microsoft.com/en-us/ef/

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД КОНТРОЛЛЕРА КУРСОВ И МОДЕЛИ КУРСА

```
using future_academy.Contexts;
using future_academy.Models;
using future_academy.Models.ViewModel;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace future_academy.Controllers
{
    [ApiController]
    [Route("api/courses")]
    public class CoursesController : ControllerBase
    {
        private readonly UniversityContext _context;

        public CoursesController(UniversityContext context)
        {
            _context = context;
        }

        [HttpGet]
        public List<Course> getCourse() {
            return _context.Courses
                .Include(c => c.tests)
                .ThenInclude(t => t.questions)
                .ThenInclude(q => q.answers)
                .ToList();
        }

        [HttpGet("{id}")]
        public Course getCourseById(int id)
        {
            var cookieLogin = HttpContext.Request.Cookies["login"];
            if (cookieLogin == null) {
                return _context.Courses
                    .Where(c => c.id == id)
                    .Include(c => c.tests.Where(t => t.endDate > DateTime.Now &&
t.isAvailable))
                    .OrderBy(c => c.id)
                    .First();
            }
            else {
                var student = _context.Students.Where(s => s.account.login ==
cookieLogin).OrderBy(s => s.id).First();
                return _context.Courses
                    .Where(c => c.id == id)
                    .Include(c => c.tests.Where(t => t.endDate > DateTime.Now &&
t.isAvailable && !t.passedStudents.Contains(student)))
                    .OrderBy(c => c.id)
                    .First();
            }
        }

        [HttpGet("toprofilepage")]
        public List<Course> getCourseToProfile() {
            return _context.Courses.ToList();
        }
```

```
        [HttpGet("testbyid/{id}")]
        public Test getTestById(int id)
        {
            return _context.Tests
                .Where(t => t.id == id)
                .Include(t => t.questions)
                .ThenInclude(t => t.answers)
                .OrderBy(c => c.id)
                .First();
        }

        [HttpPost("test")]
        public int getTestGrade(Dictionary<string, List<string>> test)
        {
            int testId;
            try { testId = Int32.Parse(test["testId"][0]); }
            catch (ArgumentNullException) { return 0; }

            var DBTest = _context.Tests.Where(t => t.id == testId)
                .Include(t => t.passedStudents).Include(t =>
t.questions).ThenInclude(q => q.answers)
                .OrderBy(t => t.id).First();

            var correctTest = new Dictionary<string, List<string>>();

            int correctQuestionCount = 0;

            foreach (var _question in DBTest.questions) {
                correctTest.Add(_question.id.ToString(), new List<string>());
                foreach (var _answer in _question.answers) {
                    if (_answer.isCorrect)
correctTest[_question.id.ToString()].Add(_answer.id.ToString());
                    else correctTest[_question.id.ToString()].Add("");
                }
                if
(correctTest[_question.id.ToString()].SequenceEqual(test[_question.id.ToString()]))
correctQuestionCount++;
            }

            var account = _context.Accounts.Where(a => a.login ==
test["userLogin"][0]).OrderBy(a => a.id).First();
            account.score += (int)Math.Round((double)correctQuestionCount /
correctTest.Count * 100);
            var student = _context.Students
                .Where(s => s.account == account)
                .OrderBy(s => s.id)
                .First();

            DBTest.passedStudents.Add(student);
            _context.SaveChanges();

            return account.score;
        }
    }
}

namespace future_academy.Models
    public class Course {
        public int id { get; set; }
        public string title { get; set; }
        public ICollection<Test> tests { get; set; }
        public string teacher { get; set; }
    }
}
```

ПРИЛОЖЕНИЕ Б

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

КОМПОНЕНТ «TEST»:

```
import React from 'react';
import style from "./Test.module.scss"
import {useNavigate, useParams} from "react-router-dom";
import {useTestById} from "../../queries/Courses/coursesQueries";
import axios from "axios";
import serializer from "../../util/form-serialize"
import {useAppSelector} from "../../redux/hooks";

const Test = () => {
  const {testid} = useParams();
  const {data: test} = useTestById(testid as string);
  const user = useAppSelector(state => state.user);
  const navigate = useNavigate();

  function onsubmit(e: any) {
    e.preventDefault();
    const result = {testId:[testid], userLogin:[user.login], ...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${item.id + item.question}`}>
            <legend>{item.question}</legend>
            {item.answers.map((answer) =>
              <label>
                {answer.text}
                <input type="checkbox" name={`${item.id}`}
id={`question${item.id}_${answer.id}`} value={answer.id}/>
              </label>
            )}
          </fieldset>
        )}
        <input type="submit" value="Сдать"/>
      </form>
    </div>
  );
};

export default Test;
```

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»:

```
import axios from 'axios';
import React from 'react';
import {SubmitHandler, useForm} from 'react-hook-form';
import {useNavigate} from 'react-router-dom';
import style from "./Login.module.scss"
import {useAppDispatch} from "../../redux/hooks";
import {setUser, dropUser} from '../../redux/userSlice';
import crypto from "crypto-js";

type FormRegisterValueType = {
  regLogin: string;
  regEmail: string;
  regPassword: string;
  regName: string;
  regSurname: string;
  regPatronomic: string;
}

type FormLoginValueType = {
  logLogin: string;
  logPassword: string;
}


const Login = () => {

  const dispatch = useAppDispatch();
  const navigate = useNavigate();

  const {
    register: loginRegister,
    handleSubmit: loginHandleSubmit,
    reset: loginReset,
    setError: loginSetError
  } = useForm<FormLoginValueType>();

  const onLoginSubmit: SubmitHandler<FormLoginValueType> = async (data) => {
    const accountRole = await axios.post("/api/account/login", {
      logLogin: data.logLogin,
      logPassword: crypto.SHA1(data.logPassword).toString()
    }).then(({data}) => data);
    if (accountRole === "err") dispatch(dropUser())
```

```
    else {
      localStorage.setItem("login", data.logLogin);
      document.cookie = `role=${accountRole};`
      dispatch(setUser({login: data.logLogin, role: accountRole}));
      loginReset();
      navigate("/");
    }
  }

  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 {
      await axios.post("/api/account/register", {
        regLogin: data.regLogin,
        regEmail: data.regEmail,
        regPassword: crypto.SHA1(data.regPassword).toString(),
        regName: data.regName,
        regSurname: data.regSurname,
        regPatronomic: data.regPatronomic,
      });
      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("logLogin", {required: true})}
```

```
                placeholder={"Логин"}
                autoComplete={"off"}/>
          <input type="password"
                className={style.input}
                {...loginRegister("logPassword", {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("regPatronomic", {required: true})}
                placeholder={"Отчество"}
                autoComplete={"off"}/>
          <button type="submit" className={style.submit}>Зарегистрироваться</button>
        </form>
      </div>
    </div>
  );
};

export default Login;
```