СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время информационные технологии занимают большое место в жизни человека. Современного человека окутывает различный информационный прогресс: цифровизация, новые профессии и форматы обучения, практики подготовки кадров, дуальное обучение и наставничество, полезные ресурсы, базы знаний и тематические онлайн-библиотеки.

Информационные технологии играют ключевую роль в обеспечении современном образовании, так как в современном мире получение знаний и навыков является важной необходимостью в жизни человека. Чем качественнее образование может получить человек, тем выше его навыки в трудовой сфере, в частности, и в повседневной жизни, в целом.

Раньше человек мог получить стандартный набор знаний и использовать его на протяжении всей жизни. Однако в современном мире появилась необходимость в непрерывном обучении. Человеку приходится находиться в режиме постоянного поиска новых знаний, новой информации. В некоторых случаях речь идёт даже о полной смене профессии, и в этом случае сменить профессию можно с помощью онлайн курсов от различных учебных центров. Для этого необходимо обратиться в учебный центр или институт, в котором проводятся курсы по повышению квалификации. Такие курсы могут пройти представители любых специальностей. При этом современные технологии позволяют обучаться как дистанционно, так и очно с использованием гибкого графика обучения, то есть, данный аспект позволяет человеку получать знания в любом месте и в любое удобное время для конкретного обучающегося.

В различных образовательных учреждениях используется интеграция различных технологий в обучающие программы для повышения качества обучающихся школьников или студентов. А также ведётся учёт о деятельности сотрудников организации.

Кроме того, современные технологии в сфере образования стали давать большой толчок для самообразования. Самообразование является

важнейшей составляющей непрерывной системы образования при минимизации трудовых затрат учителями, преподавателями. В современном мире человеку стало необходимо самостоятельно находить полезную информацию и применять её на практике. Но для самообразования необходимо иметь мотивацию для продолжения обучения, закрепления полученных навыков, приобретения опыта.

Старая система образования не сталкивалась с проблемами, которые стали актуальными в наше время. Одна из проблем — это постоянное обновление информации, то есть информационный прогресс заставляет человека оперативно находить пути решения тех или иных задач, при этом не всегда человек психологически способен быстро адаптироваться или приспособиться к меняющемуся миру. И для снижения стресса в условиях постоянно меняющегося мира человеку помогают информационные ресурсы.

В современном мире учителя и преподаватели не являются единственным источником информации, но являются достоверными носителями информации. Такая тенденция наблюдается во всём мире. В современном мире обучающийся может получить знания с помощью интернета. В интернете есть огромное количество учебников, лекций, видеокурсов по различным предметам, по решению различных задач и ситуаций. Следовательно, меняется и роль педагога. Теперь основной важнейшей ролью преподавателя будет являться руководство над образовательным процессом. Преподаватель должен помогать, направлять и заинтересовывать обучающихся.

Человек каждый день сталкивается с огромным количеством информации, так что заинтересовать его в получении каких-то определённых знаний достаточно трудно. Ранние методы обучения, которые практиковались раньше, сейчас малоэффективны. Преподавателю нужно применять новые технологии и использовать различные каналы информации. Из-за увеличения объёма информации, с которым сталкивается человек, особую важность получает навык критического мышления. Именно этому должен учить педагог. Учащийся образовательного учреждения, должен

привыкнуть к тому, что любую информацию следует перепроверять. Необходимо научить учащегося образовательного учреждения работать с информацией. Ученику нужно основывать своё мнение на фактах и логических выводах. Нужно уметь фильтровать лишнюю информацию.

Современные возможности образования постоянно расширяются за счёт развития цифровых технологий. Их используют практически во всех сферах жизнедеятельности человека.

Количество людей, которые хотят получить новые знания, непрерывно растёт. Кроме того, люди с ограниченными возможностями теперь не испытывают трудностей при получении образования.

Целью дипломного проекта является разработка универсальной платформы обучения, которая позволить создавать различные текстовые и статейные курсы бесплатного или платного формата.

Задачи дипломного проекта:

1) исследование предметной области;

2) анализ требований;

3) планирование разработки программного продукта;

4) программирование программного продукта;

5) тестирование и отладка программного продукта;

6) создание текстовой пользовательской документации;

7) создание технической документации.

Таким образом, универсальная платформа обучения представляет собой перспективное решение для удовлетворения разнообразных запросов пользователей к цифровому образованию, предоставляя им возможность гибкого и персонализированного подхода к обучению в условиях растущего спроса на онлайн-образование и удалённые образовательные ресурсы.

1. ОБЩАЯ ЧАСТЬ

1.1 Исследование предметной области

В современном обществе наблюдается значительный рост спроса на знания и навыки, что закономерно привело к увеличению зависимости людей от возможностей дополнительного образования, предоставляемого через глобальную сеть Интернет. Одним из ключевых инструментов в образовании, является веб-приложения. Онлайн-курсы, вебинары, цифровые библиотеки и интерактивные тренажеры стали не просто дополнением, а часто основным или предпочтительным источником обучения для широких слоев населения, стремящихся к профессиональному развитию и личностному росту. Параллельно с этим, традиционные образовательные учреждения - школы, колледжи, университеты - активно и стремительно внедряют разнообразные цифровые технологии, стремясь модернизировать и повысить эффективность учебного процесса, сделать его более гибким, интерактивным и отвечающим вызовам цифровой эпохи. Этот двусторонний тренд - массовый запрос на онлайн-образование и цифровая трансформация очных учреждений - формирует новую образовательную реальность.

1.2 Анализ требований

Для создания веб-приложения, которое удовлетворяло бы основные потребности всех заинтересованных сторон, необходимо детально разобраться с ключевыми требованиями платформы. Требования могут основываться в зависимости от специфики образовательного учреждения и особенностей использования существующих платформ.

Анализ требований станет фундаментом для создания полноценного и эффективного веб-приложения, которое упростит работу всех пользователей и станет надёжным инструментом в образовательном процессе:

1) функциональные требования;

2) нефункциональные требования;

3) требования к аппаратному и программному обеспечению;

4) требования к документации.

Данные этапы позволят лучше понять требования для разработки продукта и выявить критически важные элементы.

Функциональными требованиями веб-приложения будут являться:

1) безопасная передача данных и работа, т.е. веб-приложение обязано использовать защищенное подключение к веб-приложению в виде HTTPS.

2) разделение пользователей на роли, т.е. администратор и учащийся;

3) отдельная админ меню в веб-приложения;

4) возможность создания платных и бесплатных курсов;

5) возможность редактирования и удаления курса;

6) возможность создание страниц курсов с помощью «Markdown»;

7) возможность редактирования и удаления страниц курсов;

8) добавления курсов в список избранные;

9) автоматическое добавления курсов в список истории курсов.

Данные требования являются основными, на момент проектирования и разработки в программном продукте может быть больше или меньше функций, но данный перечень является основным, и при разработки стоит придерживаться данному перечню функциональных требований.

Основными нефункциональными требованиями будут являться доступность по стоимости для заказчика (потенциального владельца) этого приложения. Также разработка не должна превышать по времени 3 месяцев.

Дополнительные требования:

1) безопасность - переход и использование веб-приложения должно являться безопасным;

2) производительность - обязательное требование, которое должна быть выполнено, так как пользователь не должен долго ожидать загрузку страниц, также требуется использовать современные технологии в виде хранения данных, подгрузки данных, кэширование, cookie файлы и многое другое, что позволит работать веб-приложению отзывчиво и не создавая дискомфорт при

использовании;

3) масштабируемость - требование для того, чтобы использовать веб-приложение на различных устройствах, т.е. стационарный компьютер, смартфон, планшет. Интерфейс должен быть адаптирован для любых устройств.

Аппаратная часть сервера является одним из важных критериев при разработке и эксплуатирование программного продукта. Аппаратной частью, будет являться сервер, где будет работать наш программный продукт. Нужно описать самый главный элемент сервера, это процессор.

Процессор будет использоваться процессор не самый дорогой модели, но и не самый дешёвый, так как при обработке данных, есть вероятность того, что в любой момент может произойти множественные одновременные запросы, что может спровоцировать нагрузку на сервер. Рекомендуется использовать процессоры не дешевле Intel Core i5 или AMD Ryzen 5, а также не позднее чем поколение Intel Core i5 11400 или AMD Ryzen 5500 OEM.

Для использования и хранения временной памяти лучше всего будет использовать не менее 16 ГБ, так как требуется постоянное взаимодействие между данными. Для быстрого взаимодействия между программами, для быстрого обращения к базе данных, загрузке данных с помощью кэша пользователя.

Видеокарту можно использоваться абсолютно любую не менее 2 ГБ и не позднее 2020 года выпуска. В основном сервер не будет обрабатывать изображение большого разрешения, поэтому использовать среднее или большое по мощности оборудование не имеет смысла.

Хранение постоянной памяти лучше использовать не меньше 512 ГБ, так как данного объема хватит для базы данных, дополнительных программ для работы веб-приложения. Для быстрой скорости данных нужно использовать накопители формата SSD или NVMe SSD, но всё-таки рекомендуется использовать обыкновенные HDD диски с вращением шпинделя не менее 7200 об/мин и кэш хранения 512 МБ. Данный тип хранения позволит в 90% случаях поломок извлечь данные на исправленный накопитель, это позволит

продолжить хранить и использовать ранее хранимую информацию.

Использование высокоскоростного интернета, со скоростью не менее 100Мб/с для быстрой передачи данных.

Также стоит учесть и программную часть, так как для работы системы требуется заранее установленные программы, драйвера, библиотеки. Из основных требований будет являться обновлённая операционная система, т.е. последняя версия обновления во избежание дыр, уязвимостей и критических ошибок. Также должно быть включено логирование действий, брандмауэр системы, или другие встроенные утилиты для безопасности соединения и работы сервера. Также рекомендуется иметь драйвера с последними версиями, для стабильной работы аппаратной части сервера. Рекомендуется использовать дополнительные программы безопасности в виде антивирусов и специальных сетевых SmartScreen. Рекомендуется иметь последнюю версию СУБД, чтобы исключить системные дыры, уязвимости и критические ошибок внутри базы данных.

Дополнительно требуется составить техническую документацию для администраторов, для избежание собственного сервера и управления веб-приложением, что позволит самостоятельно настроить, обслуживать и работать с веб-приложение.

2. СПЕЦИАЛЬНАЯ ЧАСТЬ

2.1. Проектирование программного продукта

Для разработки веб-приложения, требуется разработать различные диаграммы для ясности цели и задач при разработке программного продукта.

Для создания диаграмм будем использовать Microsoft Visio Pro 2021. Для разработки вайрфреймов будем использовать программу Adobe Illustrator. После создание вайрфреймов будем создавать макет по вайрфрейму с помощью Adobe XD.

Для начала будем разрабатывать диаграммы, для начала требуется разработать диаграмму прецедентов. Диаграмма прецедентов сможет показать какие пользователи смогут пользоваться веб-приложением и что они смогут в нём делать.

После определение какие пользователи смогут пользоваться веб-приложением, можно определить какие действия будут делать пользователя в веб-приложение для достижения своей цели. В этом нам поможет диаграмма сценариев использование, т.е. диаграмма деятельности.

Диаграмма последовательности покажет, состояние и работу веб-приложение при обращении пользователю. Диаграмма покажет состояние веб-приложения и её действия.

Ранее перечисленные диаграммы больше относятся к пользователю, так как диаграммы описывают действия пользователя или состояние веб-приложения при взаимодействии пользователя. Поэтому требуется разработать технические диаграммы, для веб-приложения, которые позволят определить какие технологии и ресурсы будет использовать программный продукт. Во-первых, требуется разработать диаграмму технологического стека, для понятия, какие технические решения будут использоваться и для чего. Во-вторых, требуется разработать ER-диаграмму для создания базы данных, так как она будет являться основным местом для хранения данных в веб-приложении. В-третьих, требуется разработать диаграмму последовательности, для лучшего

понимания, как будет выполняться логика программы, при запросе пользователя к веб-приложению.

Все вышеперечисленные диаграммы очень важны при разработке. После разработки диаграмм, требуется разработать вайрфреймы, а потом макеты по подобию вайрфреймов.

Для начала создадим диаграмму прецедентов, на такой диаграмме обычно изображают два основных элемента. Первый, это пользователи, их обычно называют «актёрами», которые взаимодействуют с системой. Второй элемент, это действия, которые эти пользователи выполняют. Например: «просматривать список», «открыть курс» или «добавить курс». Диаграмма прецедентов изображена на рисунке 1.

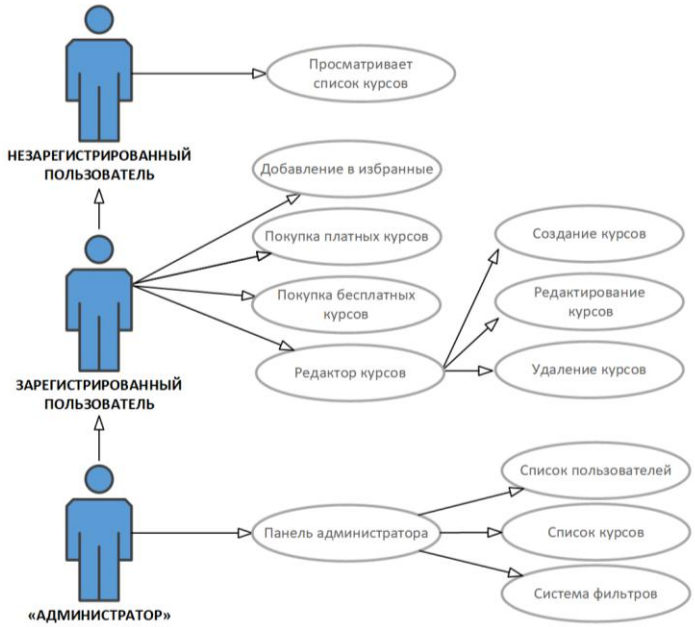


Рисунок 1 - Диаграмма прецедентов

Визуально диаграмма выглядит просто. Пользователи обозначаются фигурками, похожими на человечков. Действия изображаются в виде овалов с названиями внутри, а линии соединяют пользователей с теми действиями, которые они могут выполнять.

Диаграмма деятельности, часто сравнивают с дорожной картой: стрелки указывают направление движения от одного шага к другому, начальная точка отмечается кружком, а завершение процесса обозначается другим кружком.

Это помогает увидеть не только общий порядок действий, но и возможные «ответвления». Диаграммы деятельности изображены на рисунке 2.

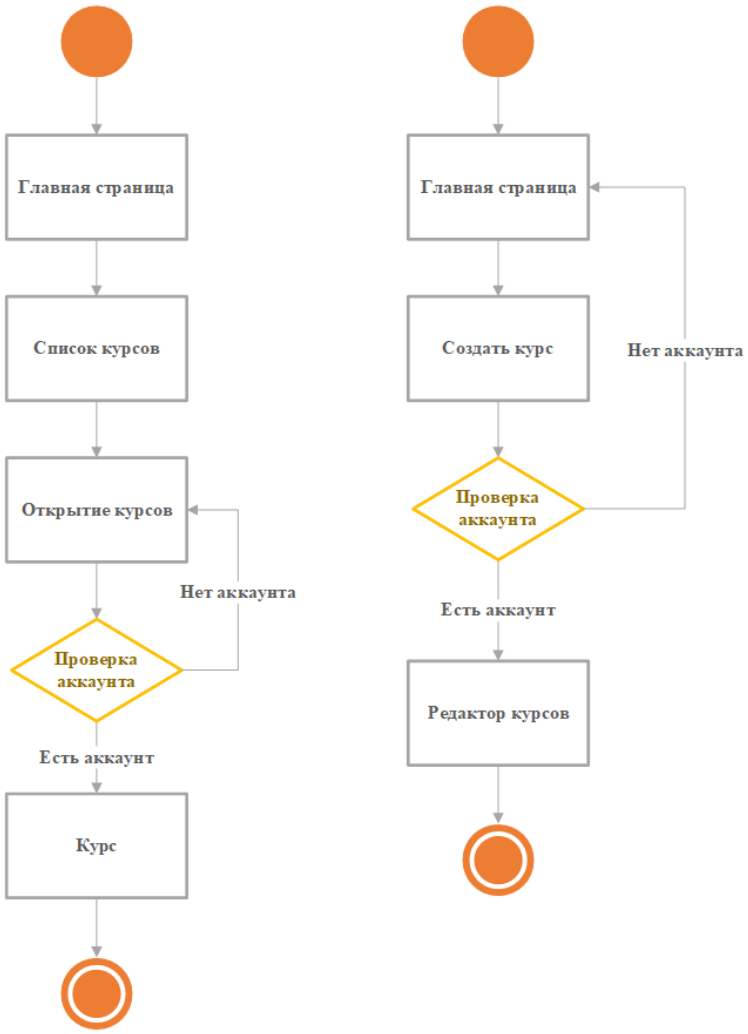


Рисунок 2 - Диаграммы деятельности

Визуально диаграмма выглядит просто. Диаграмма показывает последовательность действий, решения, которые необходимо принимать на разных этапах, и то, как эти этапы связаны между собой.

Диаграмма технологического стека, показывают какие визуальные инструменты существуют и помогают проектировать, объяснять и документировать, как устроены веб-приложения или сайты. Перед созданием технологического стека, нужно определиться, какие технологии, библиотеки или фреймворки будут использоваться в программном проекте.

Основной веб-приложения будет являться HTML5, CSS3, JavaScript, TypeScript и React.

HTML - это язык гипертекстовой разметки, появился в 1986г. используемый для создание скелета веб-приложения, который поможет организовать используемые проекты в программном продукте. Однако в программном продукте будет использоваться HTML пятого поколение, т.е. HTML5. HTML5 появился в 2014 году и значительно расширил возможности веб-ресурсов, добавив семантические элементы, т.е. теги, которые строго и чётко описывают свой смысл содержимого, одними из таких является «header», «main», «footer» и т.д. Также в HTML5 добавилась новая возможность работы с мультимедиа, так как ранее работа с мультимедийностью в HTML было не очень удобно.

CSS - это язык стилей для оформление страниц, появился в 1994г. позволив создать революционные возможности верстки, включая Flexbox и Grid системы, плавные анимации а адаптивный дизайн через медиа-запросы. CSS позволил создавать визуально привлекательные веб-приложения. CSS3 поколения появился 2005г. и на данный момент постоянно обновляется и развивается в своём направлении.

JavaScript - это язык интерпретируемого языка программирования появился в 1995г. для выполнения и обработки запросов пользователя в веб-приложении. JavaScript работал только в браузере, однако в 2015 году язык интерпретируемого языка получил официальный статус «Язык программирование», так как в 2015г. появилась версия ES6, которая добавила в JavaScript принцип объектно-ориентированного программирования, а также добавились константы, которые использовались как переменные. С этого момента JavaScript получил возможность работать на мобильных устройствах с использованием React Native, а также на персональных компьютерах без использования браузера с использованием фреймворка Electron.

TypeScript – это язык программирования, который является надстройкой над JavaScript и добавляет строгую типизацию данных. Это позволяет лучше контролировать структуру и поведения кода, обнаружение ошибок на этапе разработки. TypeScript был разработан в 2012г. компанией Microsoft как

ответ на растущую сложность приложений на JavaScript.

React - это JavaScript-библиотека для разработки пользовательских интерфейсов, однако React получил возможность TypeScript-библиотека. React работает с помощью Node.js позволяет создавать интерактивные и быстрые веб-приложения, а также разрабатывать мобильные приложение с появлением в 2015г. React Native. Основные преимущества React, это использование виртуального DOM в котором постоянно изменяются данные, но при этом не меняя настоящий DOM браузера. Изменения настоящего DOM происходит, только при изменения корневого строения объектов DOM. Также React позволяет использовать «компонентный подход», то есть использовать используемые объекты повторно.

Для программного продукта «Универсальная платформа образования» будет в основном использоваться TypeScript, это позволит минимизировать неправильную работу с типами данными. Также для работы React рекомендуется использоваться Vite для быстрой сборки клиентской части.

Vite - это современный инструмент сборки клиентской части. Vite появился в 2020г. и работает вместе с Node.js и основной задачей Vite является создание максимальной производительности и удобства клиентской части. Vite позволяет выполнять быструю сборку приравнивая их к одному целому, то есть Vite собирает весь как одно целое приложение, что позволяет намного быстрее выполнять сборку и разработка проектов, так как имеется возможность горячей перезагрузки, что позволяет видеть изменения без новой компиляции, т.е. сборки проекта.

Это будет основными элементами для программного продукта. Однако, пользователя будут иметь возможность создание статей и создавать с помощью HTML и CSS является неудобным. Поэтому в программном продукте, для написания курсов будет использоваться Markdown.

Markdown - это язык текстовой разметки и форматирования с помощью простого синтаксиса. Появился в 2004г. в основном использовался в README файлах для разработчиков, как пояснительная записка, что находится

в данном проекте и как оно работает. Однако Markdown получил большую возможность создавать удобные и простые для чтения и редактирования текста, а также легко конвертировать его в HTML.

В программном продукте требуется создать специальную область с инструментами для пользователя, чтобы легко создавать статьи, без каких-либо трудностей и знаний синтаксиса.

Серверной частью, а именно обработка запросов клиентской части для программного продукта будет являться занимается сервер. За сервер отвечает ASP.NET API.

ASP.NET API - это фреймворк для создание веб-сервера и веб-сервисов, который представляет данные и функциональность через HTTP. ASP.NET API построен на базе платформы разработки веб-приложений ASP.NET от компании Microsoft. ASP.NET был разработан 2002г. одновременно с платформой .NET Framework, по этому платформа построена и работает на .NET по принципу MVC. MVC (Model-View-Controller) является архитектурным шаблоном используемый тремя частями: модель или же объект, представление или же отображение, контроллер управления. Система работает на языке программирование C#. Язык программирования C# является языком объектно-ориентированный программирование (ООП). Язык программирования был разработан корпорацией Microsoft в 2001г. на базе языка программирования C. Объектно-ориентированное программирование (ООП) является подходом, когда программа рассматривается как набор объектов, взаимодействие друг друга. Основными аспектами ООП является инкапсуляция, наследование, полиморфизм, абстракция. Инкапсуляция является представлением только интерфейса для взаимодействия с ним, при этом другие части проекта является отдельными, тем самым делая разработку более безопаснее для других компонентов и модулей программного продукта. Наследование является создание новых классов на базе уже существующих классов, это позволяет способствует повторному использованию кода и создания иерархии классов.

Полиморфизм является обработкой различных классов с использованием одного интерфейсом. Абстракция позволяет скрыть сложные детали реализации и оставить только важные характеристики объекта.

Для хранения данных в программном продукте будет использоваться СУБД PostgreSQL.

PostgreSQL - это объектно-реляционная система управления базы данных (СУБД) с открытым исходным кодом. Позволяет создавать, изменять и удалять записи с помощью языка запросов SQL. В PostgreSQL используется модифицированный SQL язык запросов PL/pgSQL. Основной задачей PostgreSQL является хранить данные и файлы (курсы) внутри таблиц в бинарном виде. СУБД PostgreSQL способна принимать большое количество запросов и данных, а также сохранять и хранить информацию в безопасности при любых условиях.

Все используемые технологи для программного продукта изображены на рисунке 3.

Рисунок 3 - Используемых технологии для программного продукта

На диаграмме технологического стека изображен порядок и взаимодействую технологий между друг другом в программном продукте, диаграмма технологического стека изображена на рисунке 4.

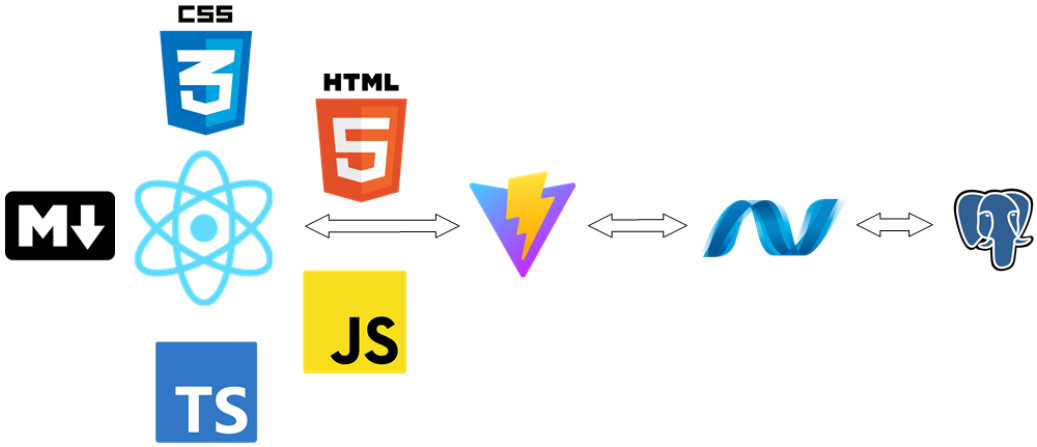


Рисунок 4 - Диаграмма технологического стека

На диаграмме технологического стека описывается, что HTML5 CSS3, Markdown, TypeScript и JavaScript являются одним целым с React. От React идёт стрелка к Vite для сборки клиентской части. React способен собирать клиентскую часть самостоятельно, однако Vite позволяет ещё сильнее ускорить процесс сборки и работу React. От Vite стрелка уходит к ASP.NET API, который будет являться сервером для обработки и выполнение запросов от клиентской части React и Vite. При обработке ASP.NET API будет обращаться к PostgreSQL, где будут добавляться, храниться, изменяться и удаляться данные программного продукта. Диаграмма технологического стека превращает сложные технические детали в наглядную схему, где видно, как данные «протекают» от интерфейса к базе данных и обратно.

ER-диаграмма помогает проектировать базу данных, показывая, какие данные хранятся и как они связаны и взаимодействую между собой в базе данных. ER-диаграмма изображена на рисунке 5.

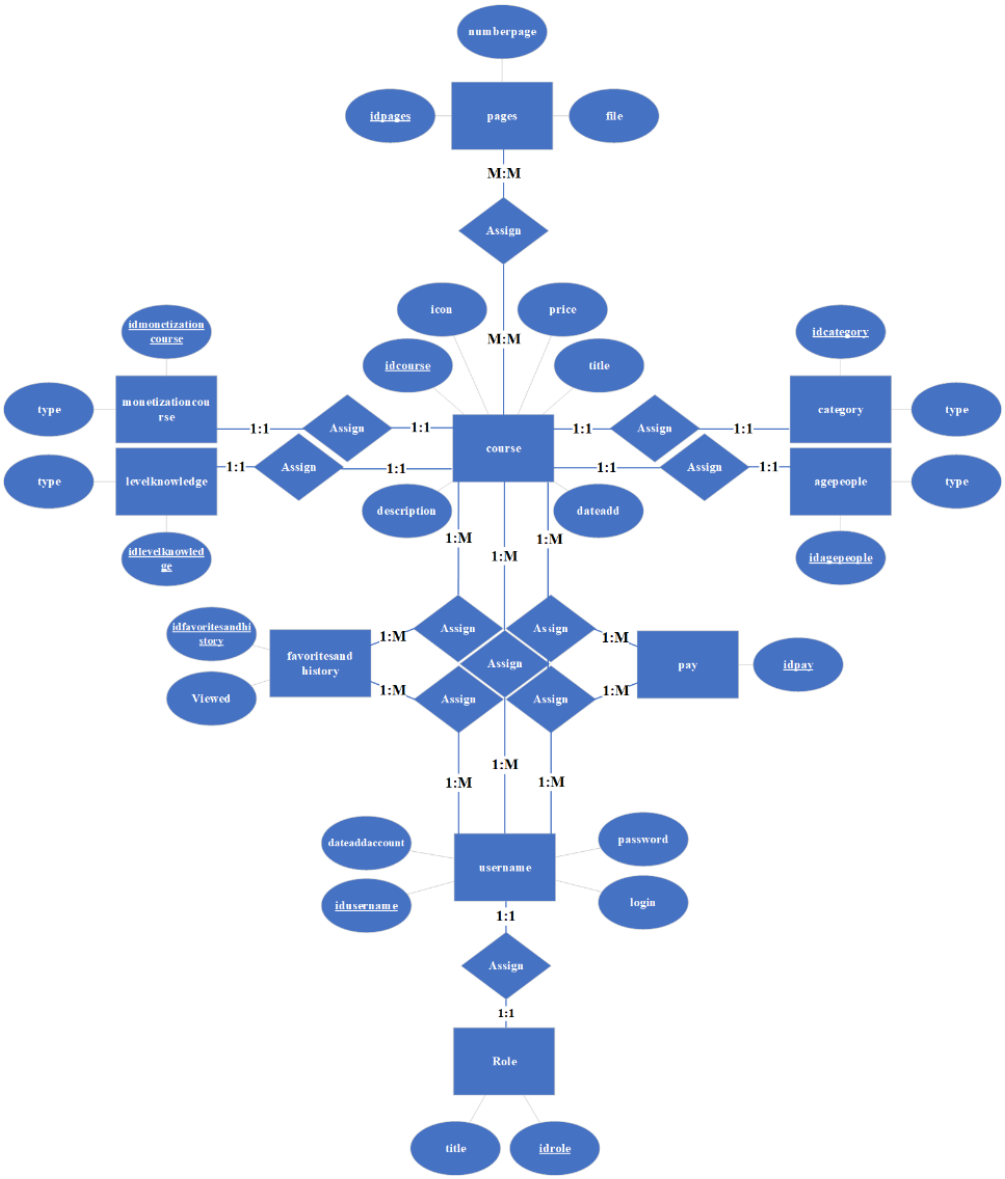


Рисунок 5 - ER-диаграмма

Слово ERD расшифровывается как Entity-Relationship Diagram, что переводится как «диаграмма сущностей и связей».

В ER-диаграмме сущности обозначаются в виде квадратов, а атрибуты в виде овалов. Также в виде ромбиков указывается связь между сущностями. На узлах, соединяющих сущности указывается как устроена связь сущностей.

Диаграмма последовательности, главная особенность диаграммы последовательности, это чёткое отражение последовательности событий во времени. Чем ниже расположено действие на схеме, тем позже оно происходит. Диаграмма последовательности изображена на рисунке 6.

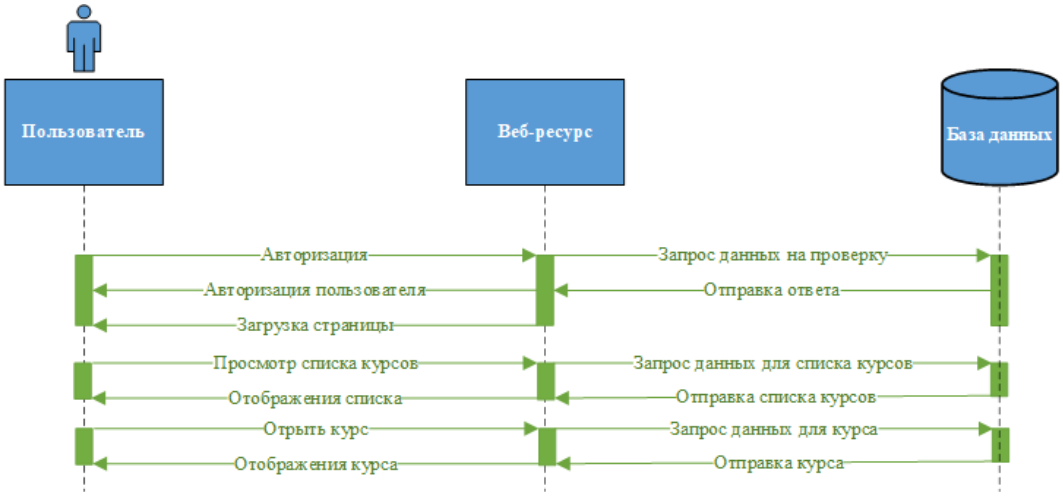


Рисунок 6 - Диаграмма последовательности

Диаграмма последовательности превращает абстрактные идеи в конкретную «дорожную карту», где видно, кто за что отвечает и в какой момент. Диаграмму последовательности часто используют, чтобы спланировать логику работы программы, или аналитики, чтобы согласовать сценарии взаимодействия между пользователем и системой.

Все вышеперечисленные диаграммы являются очень важными для разработки программного продукта. После диаграмм, можно разработать с вайрфреймы и макеты программного продукта.

Вайрфреймы позволяют создать скелеты программного продукта, обозначить схематически упрощенное представление интерфейса, где и какие элементы объектов будут находиться в программном продукте.

Самым первым вайрфреймом будет домашняя страница, на страницы будут указаны элементы описание программного продукта. Через главный экран, у пользователя будет возможность авторизации или регистрации. Вайрфрейм домашней страницы изображено на рисунке 7.

Рисунок 7 - Вайрфрейм (домашняя страница)

Вторым вайрфреймом будет являться вкладка список курсов, в данном месте будут показываться все существующие курсы в системе, изображено на рисунке 8.

Рисунок 8 - Вайрфрейм (курсы)

Также можно разработать заранее вайрфреймы для создания курса. Будет создано три этапа создания. Первый этап является обозначение курса, вторым этапом является написания страниц курса, третьим этапом является установка свойств курсу, такие как стоимость, категория и другое. Вайрфреймы изображены на рисунка 9-11.

Рисунок 9 - Вайрфрейм (первый этап создания курса)

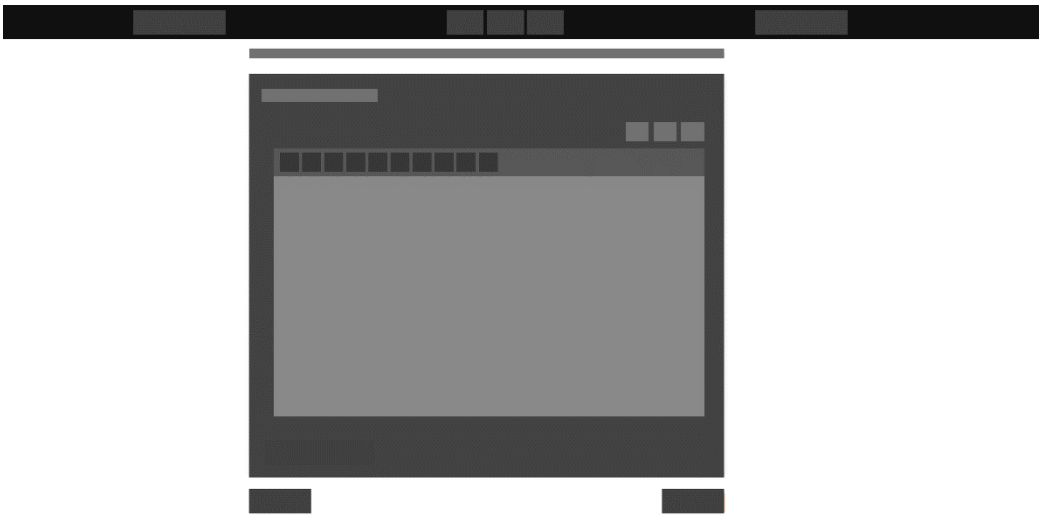


Рисунок 10 - Вайрфрейм (второй этап создания курса)

Рисунок 11 - Вайрфрейм (третий этап создания курса)

Рисунки 7-11 являются основными и хоть они сделаны, они могут отличаться от макетов и конечного результата.

Макеты являются детализированными и является визуальным представлением будущего интерфейса программного продукта. Макет выполняет несколько ключевых функций в процессе разработки. Прежде всего, он позволяет наглядно представить, как будет выглядеть конечный продукт. Макеты изображены на рисунках 12.

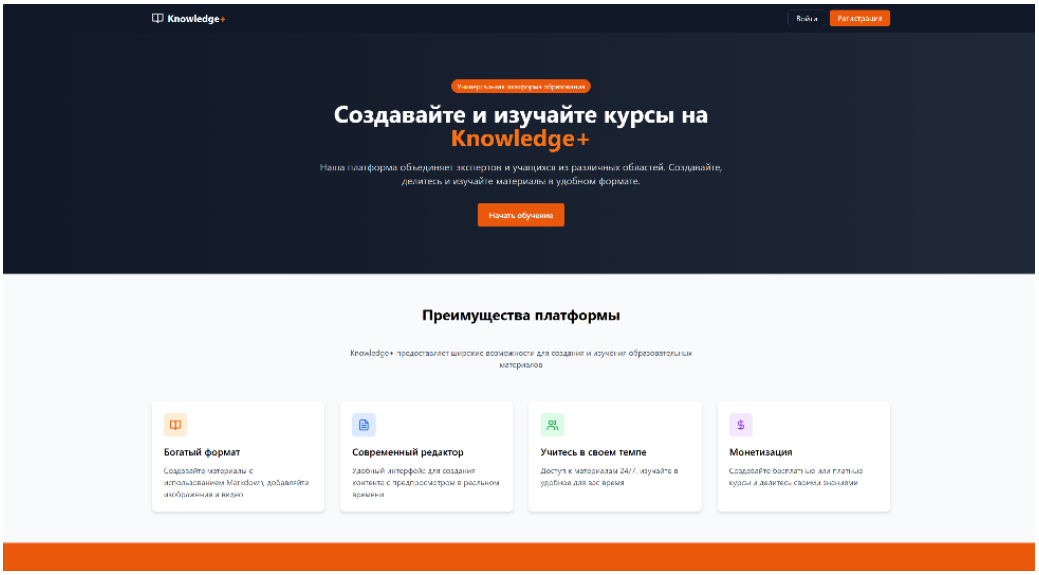


Рисунок 12 - Макет (главный экран)

На рисунке 13 изображен макет экрана курсов, где отображаются доступные курсы бесплатного и платного формата.

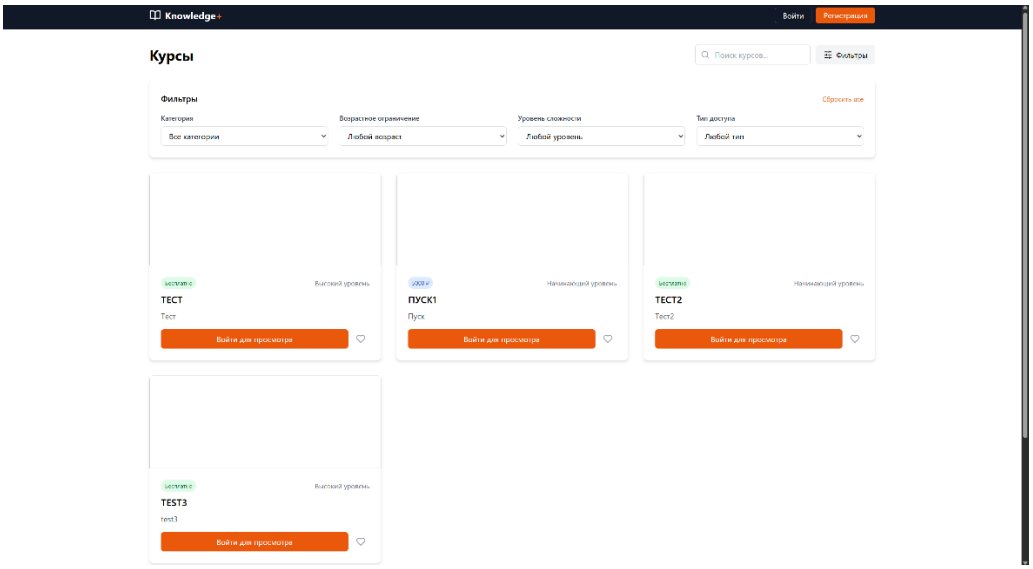


Рисунок 13 - Макет (экран с курсами)

На рисунке 14-16 изображен макет поэтапного создания курса, где пользователи создают свои курсы.

Первый этап основные сведения о курсе, рисунок 14.

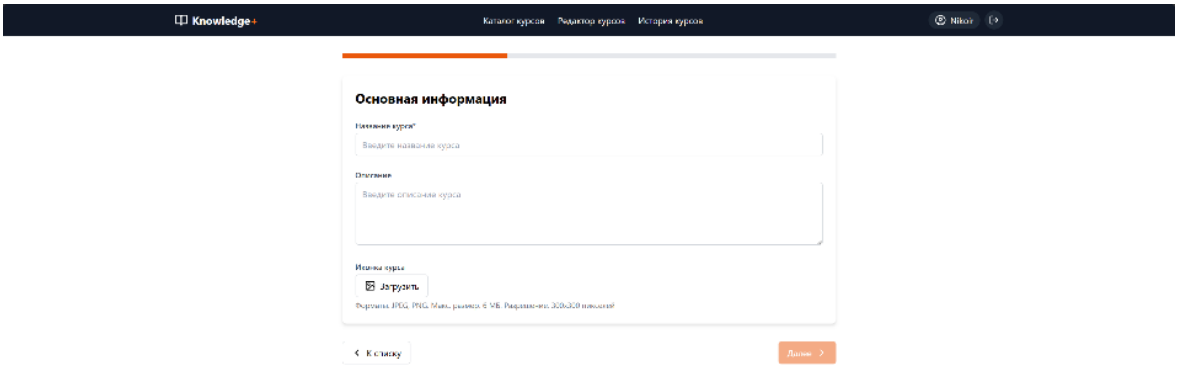


Рисунок 14 - Макет (первый этап создания курса)

Второй этап написание курса с помощью Markdown о курсе, рисунок 15.

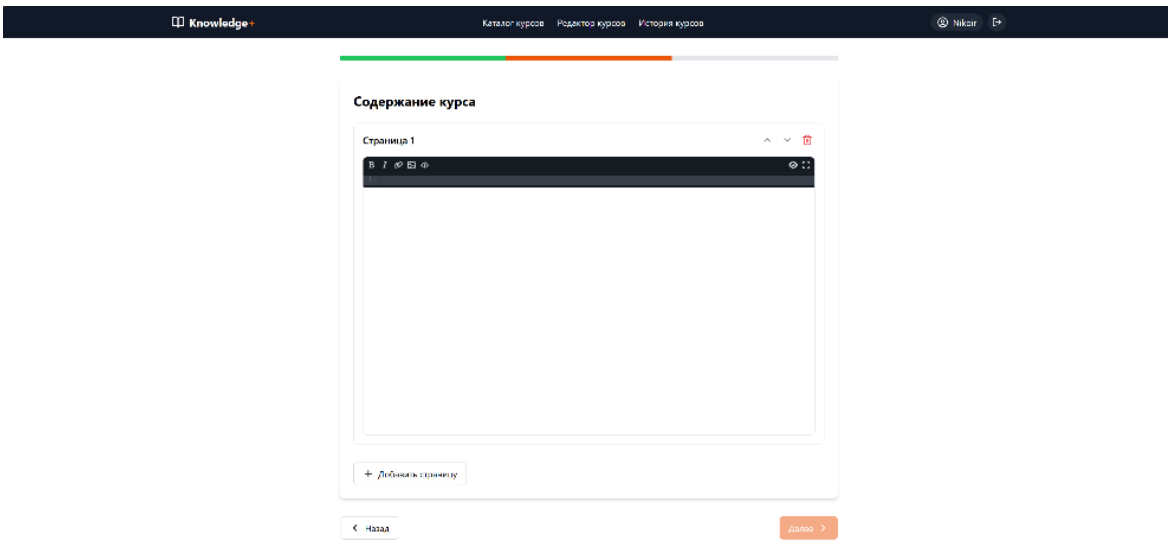


Рисунок 15 - Макет (второй этап создания курса)

Третьи этап свойства курса, благодаря которому курс можно найти с помощью каталога курсов, поисковика и фильтров, рисунок 16.

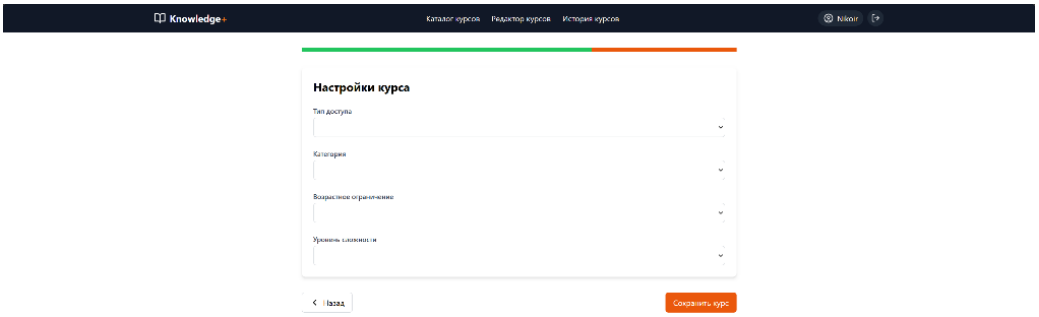


Рисунок 16 - Макет (третий этап создания курса)

Макеты реализовываются по подобию вайрфреймов, но они всё равно могут не являться конечным вариантом, однако макеты создают уже приближенный вариант программного продукта.

2.2 Программирование программного продукта

Разработка веб-платформы состоит из нескольких ключевых компонентов: реляционная база данных, серверная часть, клиентская часть.

Реляционная база данных будет разрабатываться через ER-диаграмму. Для начало требуется разработать логическую модель, позже создать физическую схему базы данных. Логическая модель выглядит как таблицы, в которых расписаны «Статусы ключей», «Название атрибутов», «Тип данных» и «Примечание». Логическая модель изображена в таблицах 1-10.

Таблица 1 - «Role»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idrole	Serial	Идентификатор
-	title	Text	Название роли

Таблица 2 - «Username»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idusername	BigSerial	Идентификатор
-	login	VarChar(50)	Логин
-	password	Text	Пароль
-	dateaddaccount	Date	Дата создания
References	Idrole	Integer	Внешний ключ

Таблица 3 - «MonetizationCourse»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idmonetizationcourse	Serial	Идентификатор
-	type	VarChar(25)	Статус стоимости

Таблица 4 - «LevelKnowledge»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idlevelknowledge	Serial	Идентификатор
-	type	VarChar(25)	Уровень курса

Таблица 5 - «Category»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idcategory	Serial	Идентификатор
-	type	VarChar(25)	Категория курса

Таблица 6 - «AgePeople»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idagepeople	Serial	Идентификатор
-	type	VarChar(3)	Возраст курса

Таблица 7 - «Course»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idcourse	BigSerial	Идентификатор
-	title	VarChar(100)	Название курса
-	description	Text	Описание курса
-	icon	Bytea	Иконка курса
-	dateadd	Date	Дата создания
References	idusername	BigInt	Внешний ключ
References	idmonetizationcourse	Integer	Внешний ключ
References	idlevelknowledge	Integer	Внешний ключ
References	idcategory	Integer	Внешний ключ
References	idagepeople	Integer	Внешний ключ

Таблица 8 - «Pages»

Статус ключа	Название колонки	Тип данных	Примечание
Primary key	idpages	BigSerial	Идентификатор
-	numberpage	Int	Номер страницы

Продолжение таблицы 8

-	file	Bytea	Страница курса
References	idcourse	BigInt	Внешний ключ

Таблица 9 - «FavoritesAndHistory»

Статус ключа	**Название колонки**	**Тип данных**	**Примечание**
Primary key	idfavoritesandhistory	Serial	Идентификатор
-	viewed	Int	Номер страницы
References	idcourse	BigInt	Внешний ключ
References	idusername	BigInt	Внешний ключ

Таблица 10 - «Pay»

Статус ключа	**Название колонки**	**Тип данных**	**Примечание**
Primary key	idpay	BigSerial	Идентификатор
References	idcourse	BigInt	Внешний ключ
References	idusername	BigInt	Внешний ключ

В логической модели у нас используется два вида ключа, такие как «Primary key» - главный ключ, и «References» - внешний ключ. Главный ключ указывается в основном только один раз в таблицы, его задача обозначить главный атрибут таблицы, который будет считать ведущим для таблицы. В основном главный ключ устанавливается на атрибутах с уникальной нумерацией «id» записей. Благодаря «id» таблицы можно связывать друг с другом, тем самым получается внешний ключ. Внешний ключ является фиксированным, и в поле с таким атрибутом можно указывать только то, что содержится в первичном ключе внешний таблицы.

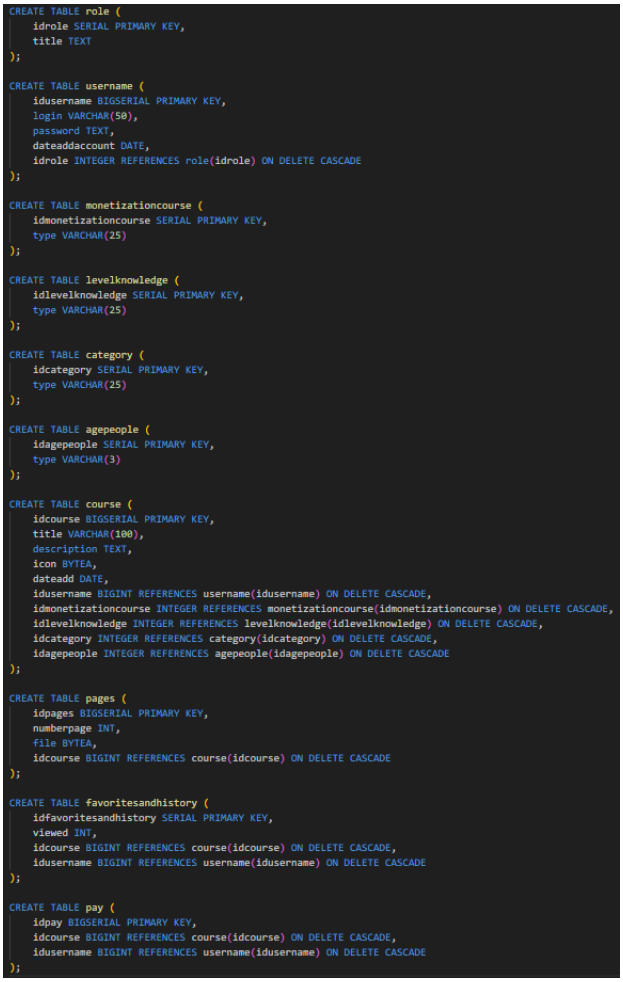
Для атрибутов с главным ключом в логической модели используются типы данных «Serial» - автоинкремент типа данных integer, и «BigSerial» - автоинкремент типа данных biginteger. Автоинкремент в основном устанавливается на главный ключ и позволит создавать уникальную нумерацию, которая не может повторяться.

Также используется типы данных категории текста «VarChar» и «Text», а также числовой «Integer» и «BigInt». Основная задача текстового это ввод

любых символов и значений. Отличие между «VarChar» и «Text» является то, что «VarChar» имеет свойство быть ограниченным по количеству введенных символов, ограничение указывается в скобках после слова «VarChar». Тип данных «Text» в свою очередь не имеет каких-либо ограничений, и тем самым имеет значение max.

Дополнительно, реляционная база данных программного продукта имеет возможность хранить файлы в бинарном формате типа данных «Bytea», с его помощью файл при загрузки конвертируется в бинарное значение, т.е. цифровое значение. «Bytea» имеет свойство хранения без размерного объема файла, однако такой способ вызывает сильную нагрузку на базу данных при поиске больших файлов, редактирование и использование их, так как файл нужно собрать в целый элемент и проверить на совпадение.

Логическая схема базы данных создана. После этого этапа, можно перейти к физической модели базы данных. Физическая модель основывается на создание SQL запроса для базы данных. На рисунке 17 указывается SQL запрос для создания базы данных.



```
CREATE TABLE role (
    idrole SERIAL PRIMARY KEY,
    title TEXT
);

CREATE TABLE username (
    idusername BIGSERIAL PRIMARY KEY,
    login VARCHAR(50),
    password TEXT,
    dateaddaccount DATE,
    idrole INTEGER REFERENCES role(idrole) ON DELETE CASCADE
);

CREATE TABLE monetizationcourse (
    idmonetizationcourse SERIAL PRIMARY KEY,
    type VARCHAR(25)
);

CREATE TABLE levelknowledge (
    idlevelknowledge SERIAL PRIMARY KEY,
    type VARCHAR(25)
);

CREATE TABLE category (
    idcategory SERIAL PRIMARY KEY,
    type VARCHAR(25)
);

CREATE TABLE agepeople (
    idagepeople SERIAL PRIMARY KEY,
    type VARCHAR(3)
);

CREATE TABLE course (
    idcourse BIGSERIAL PRIMARY KEY,
    title VARCHAR(100),
    description TEXT,
    icon BYTEA,
    dateadd DATE,
    idusername BIGINT REFERENCES username(idusername) ON DELETE CASCADE,
    idmonetizationcourse INTEGER REFERENCES monetizationcourse(idmonetizationcourse) ON DELETE CASCADE,
    idlevelknowledge INTEGER REFERENCES levelknowledge(idlevelknowledge) ON DELETE CASCADE,
    idcategory INTEGER REFERENCES category(idcategory) ON DELETE CASCADE,
    idagepeople INTEGER REFERENCES agepeople(idagepeople) ON DELETE CASCADE
);

CREATE TABLE pages (
    idpages BIGSERIAL PRIMARY KEY,
    numberpage INT,
    file BYTEA,
    idcourse BIGINT REFERENCES course(idcourse) ON DELETE CASCADE
);

CREATE TABLE favoritesandhistory (
    idfavoritesandhistory SERIAL PRIMARY KEY,
    viewed INT,
    idcourse BIGINT REFERENCES course(idcourse) ON DELETE CASCADE,
    idusername BIGINT REFERENCES username(idusername) ON DELETE CASCADE
);

CREATE TABLE pay (
    idpay BIGSERIAL PRIMARY KEY,
    idcourse BIGINT REFERENCES course(idcourse) ON DELETE CASCADE,
    idusername BIGINT REFERENCES username(idusername) ON DELETE CASCADE
);
```

Рисунок 17 - Физическая модель базы данных

В физической модели базы данных используется SQL в формате PL/pgSQL для PostgreSQL. Запрос на создание таблицы в базе данных основывается на написании «CREATE TABLE (название таблицы)» и открытие скобок. В скобках пишется порядок атрибутов и свойства-параметры атрибутов. Например, в первом атрибуте, будет указываться как уникальный идентификатор записи. Указывается название, а также свойство «SERIAL», которое обозначает, что данный атрибут имеет статус автоинкремента и имеет тип INTEGER, счётчик начинается с 1. Дополнительно в основном к атрибуту, у которого установлено свойство «SERIAL» добавляется «PRIMARY KEY», что означает главный ключ атрибута.

Также в запросе SQL в некоторых атрибутах можно встретить «REFERENCES», это обозначается, что к атрибуту установлен внешний ключ, дальше указывается какая таблица и из какого атрибута брать данные.

Дополнительно указывается «ON DELETE CASCADE», это означает, что запись, которая будет удалена из основной таблицы будет удалено значение из внешних таблиц, для избежание ошибок.

Для того, чтобы выполнить запрос SQL мы создадим базу данных в СУБД PostgreSQL с помощью менеджера управления СУБД pgAdmin4. После запуска pgAdmin4 и ввода пароля администратора, мы создадим с помощью модального окна базу данных. Через несколько секунд, после создание базы данных, она отобразится в списке. После этого можно создать окно запроса и ввести SQL запрос, где после pgAdmin4 сообщит нам о статусе запроса.

После успешного создания, можно вызвать ERD-диаграмму внутри СУБД, так мы сможем определить создалась наша база данных или нет. После того, как ERD-диаграмма схожа с ERD-диаграммой в базе данных, будет считаться, что создана физическая модель, на рисунке 18 изображена физическая ERD-диаграмма.

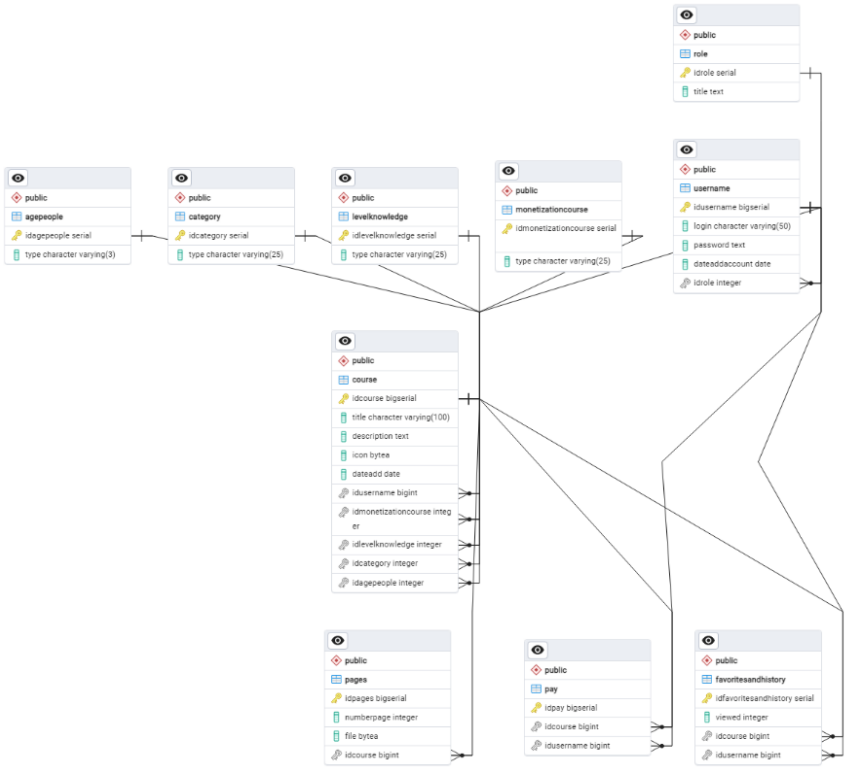


Рисунок 18 - Физическая ERD-диаграмма

Основные взаимодействия с базой данной выполнены, поэтому можно приступить к созданию клиентской части.

Для создание серверной части, требуется создать проект ASP.NET API. При создании проекта, установим, что основную платформу будем использовать .NET 9.0 с типом проверки подлинности «нет». Дополнительно укажем, что будем использовать сервер с HTTPS. Установим галочку на параметре «Включить поддержку OpenAPI», данный параметр позволит нам использовать «Swagger», через который мы сможем проверить работу API и соединение с PostgreSQL. Swagger может дать возможность на тестирование действий API, так как через Swagger можно создавать, получать, изменять и удалять данные в базе данных. Также убедимся, что стоит галочка на «Использовать контроллеры», данная функция очень является обязательной, так как позволяет производить разделение управлением запросов. Контролеры отвечают за обработкой и выполнение запросов пользователя. Остальные параметры можно оставить по умолчанию, изображение параметров создания API на рисунке 19.

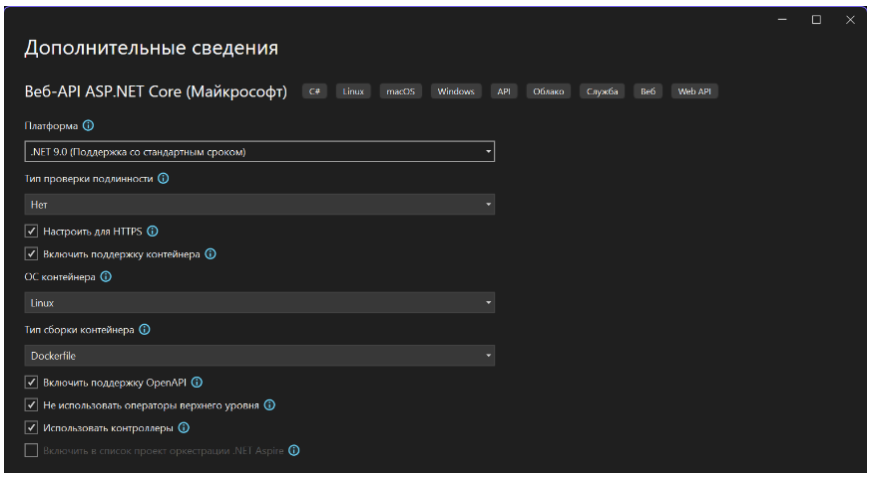


Рисунок 19 - Параметры создания ASP.NET API

Для начала нужно установить дополнительные и основные пакеты из «NuGet». При создании проекта уже были автоматически скачены пакеты для работы ASP.NET API, но нам нужно установить дополнительные пакеты для работы нашего программного продукта. Самым основным пакетом, будет являться «Microsoft.EntityFrameworkCore» для работы с Microsoft EntityFramework. Установим пакет «Swashbuckle.AspNetCore» для создания Swagger страницы, а также COM чтобы была возможность создать передачу данными

между сервером ASP.NET API и клиентской части на Vite + Reacte. Установим пакет «Npgsql.EntityFrameworkCore.PostgreSQL» для создания подключения к базе данных PostgreSQL, а также для взаимодействия с СУБД. Также установим пакет «Microsoft.EntityFrameworkCore.Design» для создания и работа с представлением данных. Дополнительно, для расширенной и лучшей работы JSON пакетов в проекте используется пакет «Microsoft.AspNetCore.Mvc.NewtonsoftJson». На рисунке 20 изображены установленные пакеты NuGet для ASP.NET API.

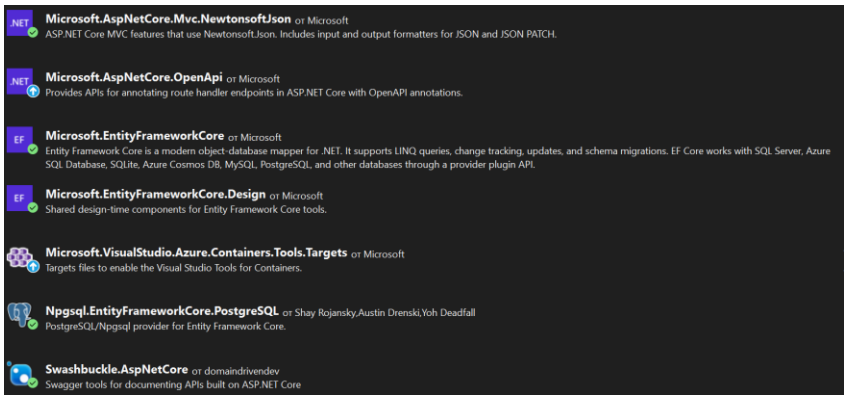


Рисунок 20 - Установленные пакеты NuGet

Для начала требуется указать подключение к базе данных PostgreSQL, подключение рекомендуется создать в файле «appsettings.json», чтобы избежать перехвата подключения к базе данных, строка подключения к базе данных указана на рисунке 21.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",

  "ConnectionStrings": {
    "DefaultConnection": "Host=localhost;Port=5432;Database=odbDiplomProject;Username=postgres;Password=root"
  }
}
```

Рисунок 21 - Строка подключения к базе данных

Начнём создавать модели для ASP.NET API, модели являются что-то на подобии таблиц и атрибутов как в PostgreSQL, но они будут использоваться как буфер данных из базы данных, которые будут хранить данные заранее. Модели создаются с помощью классов, класс должен иметь схожее имя, что

и название таблицы в базе данных. Изображения кода классов, а также каталог моделей указана на рисунках 22-24.

```
// Role.cs
namespace DiplomApi.Models;

public class Role
{
    public int Idrole { get; set; }
    public string Title { get; set; }
}

// Username.cs
namespace DiplomApi.Models;

public class Username
{
    public long Idusername { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
    public DateTime Dateaddaccount { get; set; }
    public int Idrole { get; set; }
    public Role Role { get; set; }
}

// MonetizationCourse.cs
namespace DiplomApi.Models;

public class MonetizationCourse
{
    public int Idmonetizationcourse { get; set; }
    public string Type { get; set; }
}

// LevelKnowledge.cs
namespace DiplomApi.Models;

public class LevelKnowledge
{
    public int Idlevelknowledge { get; set; }
    public string Type { get; set; }
}

// Category.cs
namespace DiplomApi.Models;

public class Category
{
    public int Idcategory { get; set; }
    public string Type { get; set; }
}

// AgePeople.cs
namespace DiplomApi.Models;

public class AgePeople
{
    public int Idagepeople { get; set; }
    public string Type { get; set; }
}
```

Рисунок 22 - Код классов моделей №1

```
// Course.cs
namespace DiplomApi.Models;

public class Course
{
    public long Idcourse { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public byte[] Icon { get; set; }
    public Datetime Dateadd { get; set; }

    public long Idusername { get; set; }
    public int Idmonetizationcourse { get; set; }
    public int Idlevelknowledge { get; set; }
    public int Idcategory { get; set; }
    public int Idagepeople { get; set; }

    public Username User { get; set; }
    public MonetizationCourse Monetization { get; set; }
    public LevelKnowledge Level { get; set; }
    public Category Category { get; set; }
    public AgePeople Age { get; set; }
}

// Pages.cs
namespace DiplomApi.Models;

public class Pages
{
    public long Idpages { get; set; }
    public int Numberpage { get; set; }
    public byte[] File { get; set; }

    public long Idcourse { get; set; }
    public Course Course { get; set; }
}

// FavoritesAndHistory.cs
namespace DiplomApi.Models;

public class FavoritesAndHistory
{
    public int Idfavoritesandhistory { get; set; }
    public int Viewed { get; set; }

    public long Idcourse { get; set; }
    public long Idusername { get; set; }

    public Course Course { get; set; }
    public Username User { get; set; }
}

// Pay.cs
namespace DiplomApi.Models;

public class Pay
{
    public long Idpay { get; set; }

    public long Idcourse { get; set; }
    public long Idusername { get; set; }

    public Course Course { get; set; }
    public Username User { get; set; }
}
```

Рисунок 23 - Код классов моделей №2

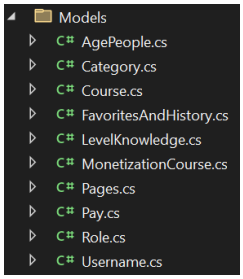


Рисунок 24 - Каталог моделей

После создания моделей требуется создать контекст базы данных, который будет иметь понимание, как работать ASP.NET API с базой данных в PostgreSQL. В контексте указывается метод подключения к базе данных, который будет использовать строку подключения, которая была указана ранее в «appsettings.json». В контексте указываются таблицы из базы данных, а также модели, к которым будут привязаны таблицы. Также в контексте указывается логика получения и взаимодействие с данными из таблиц базы данных. Изображения кода контекста указывается на рисунках 25.

```
public class AppDbContext : DbContext
{
    Ссылок: 0
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

    Ссылок: 0
    public DbSet<Role> Roles { get; set; }
    Ссылок: 0
    public DbSet<Username> Usernames { get; set; }
    Ссылок: 0
    public DbSet<MonetizationCourse> MonetizationCourses { get; set; }
    Ссылок: 0
    public DbSet<LevelKnowledge> LevelKnowledges { get; set; }
    Ссылок: 0
    public DbSet<Category> Categories { get; set; }
    Ссылок: 0
    public DbSet<AgePeople> AgePeoples { get; set; }
    Ссылок: 0
    public DbSet<Course> Courses { get; set; }
    Ссылок: 0
    public DbSet<Pages> Pages { get; set; }
    Ссылок: 0
    public DbSet<FavoritesAndHistory> FavoritesAndHistories { get; set; }
    Ссылок: 0
    public DbSet<Pay> Pays { get; set; }

    Ссылок: 0
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Username>()
            .HasOne(u => u.Role)
            .WithMany()
            .HasForeignKey(u => u.Idrole)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Course>()
            .HasOne(c => c.User)
            .WithMany()
            .HasForeignKey(c => c.Idusername)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Pages>()
            .HasOne(p => p.Course)
            .WithMany()
            .HasForeignKey(p => p.Idcourse)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<FavoritesAndHistory>()
            .HasOne(f => f.Course)
            .WithMany()
            .HasForeignKey(f => f.Idcourse)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<FavoritesAndHistory>()
            .HasOne(f => f.User)
            .WithMany()
            .HasForeignKey(f => f.Idusername)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Pay>()
            .HasOne(p => p.Course)
            .WithMany()
            .HasForeignKey(p => p.Idcourse)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Pay>()
            .HasOne(p => p.User)
            .WithMany()
            .HasForeignKey(p => p.Idusername)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Рисунок 25 - Контекст базы данных

После того, как будут созданы все модели и настроена логика работы с базой данной PostgreSQL, можно заняться настройкой самого сервера с помощью класса «Programs.cs». Для удобства в первую очередь добавим возможность взаимодействие с нашим ASP.NET API с помощью Swagger.

Swagger, это набор инструментов для разработки, тестирования API и документирования. Swagger может помочь при разработке, создав тестовые контроллеры и тем самым проверив взаимодействия API с СУБД, на рисунке 26 изображен код тестового контроллера для таблицы «AgePeople».

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Backend_DiplomProject.Models;
using Backend_DiplomProject;

namespace DiplomApi.Controllers;

[Route("api/[controller]")]
[ApiController]

public class AgePeopleController : ControllerBase
{
    private readonly AppDbContext _context;


    public AgePeopleController(AppDbContext context) => _context = context;

    [HttpGet]

    public async Task<ActionResult<IEnumerable<AgePeople>>> GetAll()
        => await _context.AgePeoples.ToListAsync();

    [HttpGet("{id}")]

    public async Task<ActionResult<AgePeople>> GetById(int id)
    {
        var item = await _context.AgePeoples.FindAsync(id);
        return item == null ? NotFound() : item;
    }

    [HttpPost]

    public async Task<ActionResult<AgePeople>> Create(AgePeople item)
    {
        _context.AgePeoples.Add(item);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetById), new { id = item.Idagepeople }, item);
    }

    [HttpPut("{id}")]

    public async Task<IActionResult> Update(int id, AgePeople item)
    {
        if (id != item.Idagepeople) return BadRequest();

        _context.Entry(item).State = EntityState.Modified;
        await _context.SaveChangesAsync();
        return NoContent();
    }

    [HttpDelete("{id}")]

    public async Task<IActionResult> Delete(int id)
    {
        var item = await _context.AgePeoples.FindAsync(id);
        if (item == null) return NotFound();

        _context.AgePeoples.Remove(item);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
```

Рисунок 26 - Тестовый контроллер «AgePeople»

После создания контроллера, нужно правильно настроить Swagger для работы с HTTPS. Swagger в основном работает с помощью протокола HTTP для

избежание проблем при передаче и получение данных. Однако для безопасности требуется подготовить ASP.NET API к использованию HTTPS и Swagger для этого является отличным инструментом тестирования, однако есть проблема в том, что пакет данных при отправлении и получение используется в формате JSON. При отправке данных с помощью сетевого протокола HTTPS пакет данных кодируется и при получении пакета его нужно раскодировать. Кодирование помогает избежать перехвата данных третьими лицами, но создает дополнительные нагрузки на сервер, так как при больших запросах системе нужно выполнить расшифровку, а после выполнить обработку. Для проверки логики ASP.NET API нужно настроить Swagger под работу HTTPS. Для настройки Swagger HTTPS нужно указать в Program.cs запуск протокола HTTPS, на рисунке 27 указан запуск Swagger и интерфейс Swagger, а также указано, что сервер должен запускаться по протоколу HTTPS.

```
app.UseSwagger();
app.UseSwaggerUI();
app.UseHttpsRedirection();
```

Рисунок 27 - Параметры запуска Swagger и ASP.NET API

Дополнительно нужно указать в конфигурации запуска алгоритм работы в «launchSettings.js» как запускать компиляцию проекта, на рисунке 28 изображен конфиг запуска сервера.

Рисунок 28 - Параметр запуска конфигурации сервера

После запуска сервера мы можем открыть веб-страницу API, при нажатии на ссылку из консоли сервера, на рисунке 29 изображена консоль сервера ASP.NET API, на рисунке 30 изображен интерфейс Swagger.

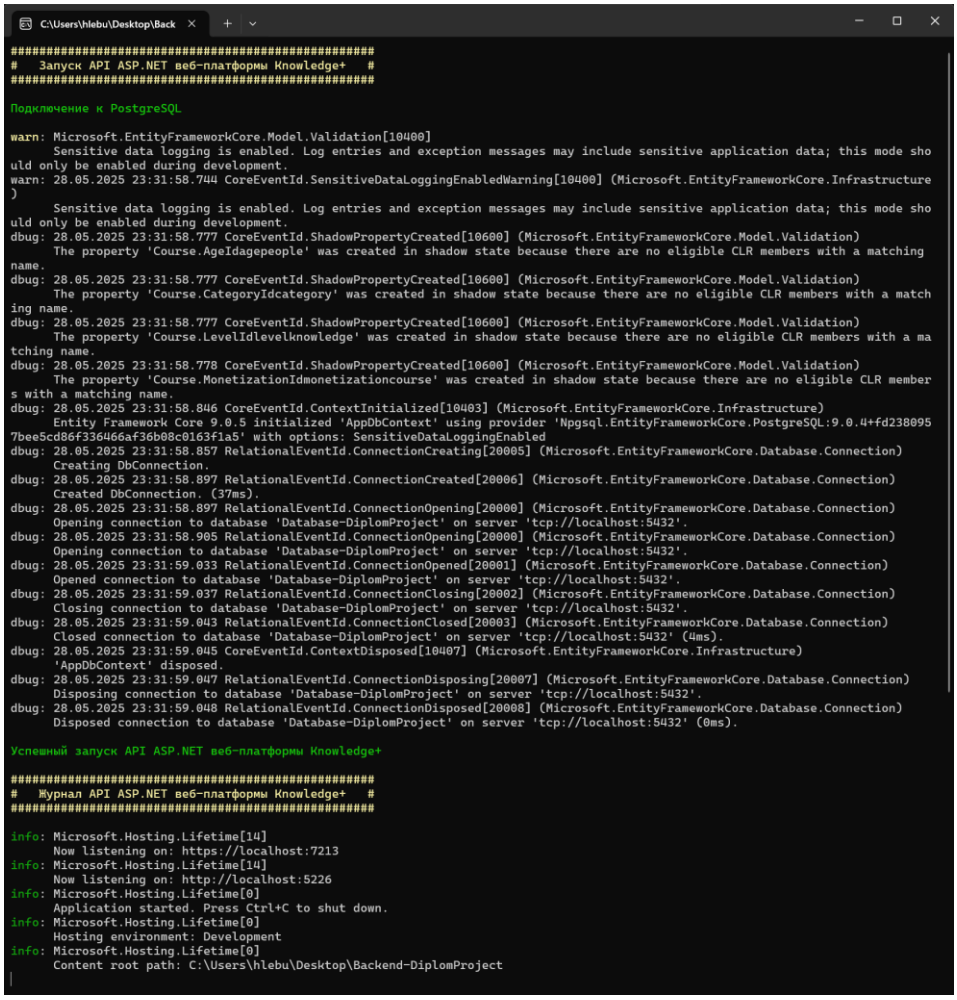


Рисунок 29 - Консоль сервера ASP.NET API

Рисунок 30 - Интерфейс Swagger

Чтобы проверить вывод данных, требуется использовать запрос «GET» чтобы добавить данные в таблицу нужно использовать запрос «PULL». Для

редактирования записи нужно использовать запрос «PUT», чтобы выполнить редактирование определённой записи нужно создать поиск по уникальному идентификатору, тем самым находя запись, можно выполнить редактирование без перезаписи. Если нужно удалить определённую запись, то выполним запрос «DELETE», с данным запросом нужно выполнить аналогичные действия, что и при редактировании, найти запись по уникальному идентификатору и выполнить удаление. После настройки и тестирования, мы создам систему обмена данными между сервером и клиентской частью. Для этого нам поможет система «CORS».

CORS, это система совместного использования ресурсов разных источников. Система CORS позволяет веб-страницам получать доступ к объектам сторонних интернет-ресурсов, в нашем случае получение данных из клиентской части. Аналогичная система будет установлена в клиентской части для получения данных от серверной части. Данная система позволит передавать данные по HTTP или HTTPS протоколу, но в нашем случае так как сервер и клиентская часть находится на одном устройстве, это позволит нам реализовать передачу через обычное локальное соединение с использованием сетевого порта.

Для передачи данными с помощью ASP.NET API кроме контроллеров и моделей, нужно ещё создать DOT.

DOT, это динамический параметр метода контроллера. Получает принимать из тела запроса, в методе контроллера без необходимости привязки к модели. Использование DOT в ASP.NET API позволяет упростить разработку, так как не нужно реализовывать пользовательскую привязку модели для обработки запросов с определёнными данными, в частности JSON. Технология DOT может адаптироваться к растущим требованиям пользователей, тем самым упрощаю нагрузку на серверную часть.

Сервер ASP.NET API для «Универсальной платформы образовании» готов. Дальнейшие обновления и изменения в сервер будут происходить по мере необходимости разработки клиентской части.

Для создание клиентской части, требуется установить «Node.js», с официального сайта для нужной платформы, на рисунке 31 изображена конфигурация «Node.js».

Рисунок 31 - Установка конфигурации Node.js

После установки можно создавать проект «Vite + React» с использованием TypeScript. На рисунке 32 изображен интерфейс создания проекта.

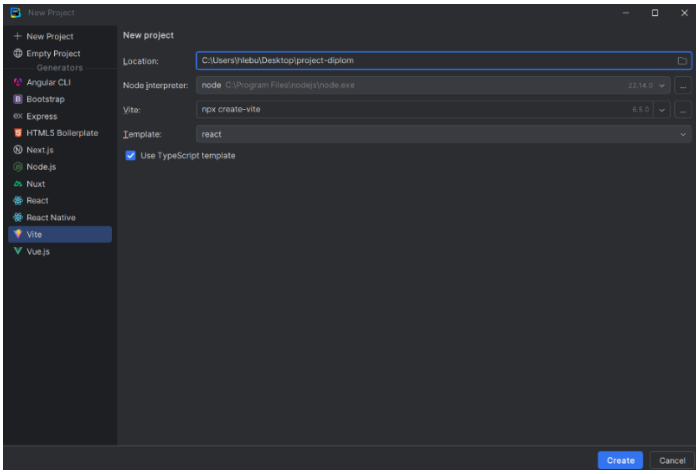


Рисунок 32 - Создание проекта «Vite + React»

После создания у нас будет пустой проект, по умолчанию не всегда может быть установлен компилятор, чтобы его создать в верхнем части нужно нажать по кнопки «Current File» и нажмём на «Edit Configuration». В открытом окне нужно сверху выбрать «+» и выбрать там использование

«npm». После выбора будут параметры компилятора, где нужно указать режим компиляции «dev», на рисунке 33 изображено окно настройки компилятора npm.

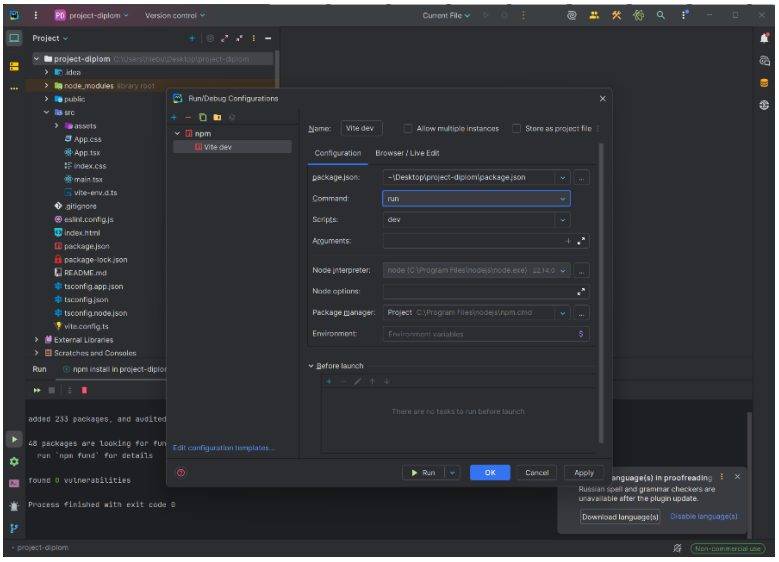


Рисунок 33 - Настройка компилятора npm

После настройки нажмём на «Apply» и «OK» можно запустить проект, после запуска в консоли появится процесс сборки, и если всё хорошо, появится поле с локальным портом, по которому можно увидеть собранный проект. На рисунке 34-35 изображена сборка и итоговой вариант сборки проекта.

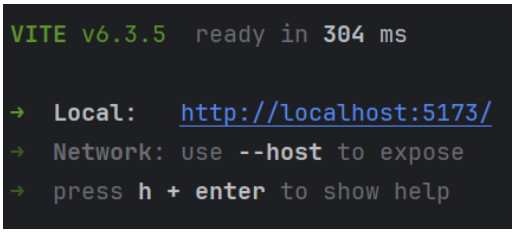


Рисунок 34 - Консоль сборки

Рисунок 35 - Итоговый вариант сборки проекта

В клиентской части разрабатывается в основном интерфейс клиента, так как это очень обильно долгий этап, так как при разработке клиентской части нужно возвращаться к серверной части и дорабатывать, так как клиентская часть принимает и показывает запросы пользователя, а сервер должен правильно получать и обрабатывать запросы пользователя от клиентской части. Создание клиентской части реализовывается с помощью HTML, CSS, Markdown, а также framework Tailwind CSS для упрощения разработки веб-приложения и правильной работы Vite с CSS.

Для начала требуется в проект добавить конфигурацию подключения к серверной части с помощью «CORS». Для этого мы создадим в корневом проекте файл с названием «vite.config.ts» и укажем туда конфигурации всего проекта и в том числе взаимодействия с ASP.NET API, на рисунке 36 указан код конфигурации проекта и подключения.

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

no usages
export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': '/src', // Алиас для удобных импортов
    },
  },
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:5226',
        // target: 'https://localhost:7213',
        changeOrigin: true,
        secure: false,
        rewrite: (path : string ) : string  => path.replace( searchValue: /^\/api/,  replaceValue: '/api'),
        headers: {
          Connection: 'Keep-Alive'
        }
      },
    },
    port: 5173,
    open: false,
  },
  build: {
    outDir: 'dist',
    sourcemap: true,
  },
  optimizeDeps: {
    exclude: ['lucide-react'],
    include: ['react-router-dom'],
  },
});
```

Рисунок 36 - код конфигурации проекта и подключения

В коде конфигурации мы указываем использование плагина react, где

упрощаем понимание alias, для упрощения написания ссылок, теперь, когда в коде указывается «@», это означает, что это путь к папке «src». В конфигурации server указывается proxy для работы, дополнительно указывается файл конфигурации работы сервера на клиентской части api, указывается target для подключения к ASP.NET API. В конфигурации также указывается port клиентская части по умолчанию. Всё остальное используется по умолчанию как без использования конфигурации в проекте.

Мы указали файл api, где будет приниматься и отправляться запросы, так что нам нужно оформить файл api, на рисунке 37 указан код файла api.

```
import axios from 'axios';

// Создание api
Show usages
const api : AxiosInstance = axios.create({
  baseURL: '/api',
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
});

// Вывод ошибок от API
api.interceptors.response.use(
    response : AxiosResponse<any, any> => response,
    error : any => {
        // Логирование деталей ошибки
        console.error( ...data: 'API ошибка:', {
            message: error.message,
            status: error.response?.status,
            data: error.response?.data,
            url: error.config.url
        });

        // Обработка ошибки 401
        if (error.response?.status === 401) {
            localStorage.removeItem( key: 'token');
            localStorage.removeItem( key: 'user');
            window.location.href = '/login';
        }

        return Promise.reject(error);
    }
);

// Перехватчик для добавления токена к запросам
api.interceptors.request.use(config : InternalAxiosRequestConfig<any> => {
    const token : string | null = localStorage.getItem( key: 'token');
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
});
```

Рисунок 37 - код файла api

В файле api указывается подключение библиотеки axios для работы с API. Создаем константу api и загружаем в axios основные параметры axios, уста-

новим по умолчание таймер на 10000 тикитов = 10 секунд для запросов. Константа нам поможет использовать данный вариант axios без риска перезаписи, так как константа запрещает перезапись данных, если она не является основным.

Создаем логирование всех ошибок API от ASP.NET API. Дополнительно создадим обработку ошибки 401 при получении от ASP.NET API, то система удаляет данные пользователя и его токен из локальной базы, где пользователь перенаправляется на страницу авторизации. Для этого используется метод «api.interceptors.response.use».

Создадим перехватчик для добавления токена к запросам JWT. При создании запроса на отправку JWT ищет JWT-токен в локальной базе по ключу token. Если пользователь авторизован, то система найдет его токен, если не авторизован, то возвращается пустота и запрос невозможен. Для этого используется метод «api.interceptors.request.use».

После настройки vite-config и api можно начать разработку модулей клиентской части. В основном для разработки клиентской части используется формат файлов tsx, данный формат является форматом проекта библиотеке React, в ней же указывается логические элементы веб-платформы с использованием TypeScript и графический пользователь интерфейс с использованием HTML и CSS, однако стили в CSS с использованием технологии Vite является не полностью функциональными или ошибочными, для избежание данной проблемы используется библиотека стилей Tailwind с использованием конфигурацией PostCSS. Основные библиотеки позволят упростить написание стилей CSS и позволит CSS работать полностью правильно и корректно в проекте Vite. Также при создании модуля курсов, с разделением на редактирование и просмотра курсов используется текстового форматирования Markdown, что позволяет использовать формат разработки и формат просмотра, где не отображается код. При разработки каждого модуля, которому требуется работа сервера ASP.NET API, требуется создавать контроллеры и DOT при необходимости.

2.3 Тестирование и отладка программного продукта

После разработки, требуется провести различные тестирования на наличия ошибок, конфликтов, опечаток, забытие строчек кода и даже неправильное написание комментариев в коде.

Ошибки могут быть настолько скрыты, что они могут появляться не часто. Ошибки могут возникать не часто или раз в какое-то нажатие или в определенный момент времени и действия клиентской части.

Для лучшего тестирование, требуется использовать различные сценарии и различные виды тестирования. На данный момент, разработчики очень часто делают тестирование автоматическими с помощью Unit-Test.

При написании сценария и условия на языке программирование, можно запустить тесты и выявить различные проблемы. Но автоматические тесты могут оказаться не всегда точными из-за своей прямолинейности. В большинстве случаев автоматические тесты не находят 70% ошибок.

Для проверки нашей программы будем использовать в основном ручное тестирование. Составим сценарии тестирования, для более лучше понимая, что нужно протестировать во избежание критических ошибок. Сценарии тестирования представлены в таблице 11.

Таблица 11 - «Сценарий тестирования»

Название тестирования	**Процесс тестирования**	**Ожидаемый результат**	**Фактический результат**
Проверка валидации типов данных в атрибутах PostgreSQL	Создание данных с соответствием и не соответствием установленным типам данных	Данные с верными типами данных будут созданы и не будут созданы с не верными данными	Успешно
Проверка связей между сущностями PostgreSQL	Проверка ключей и связей между сущностями	Все нужные атрибуты имеют связи между сущностями	Успешно
Запуск сервера ASP.NET API	Выполненная компиляция	Проект выполни компиляцию без ошибок	Успешно

Продолжение таблицы 11

Название тестирования	Процесс тестирования	Ожидаемый результат	Фактический результат
Подключение Swagger к ASP.NET API	Запуск сервера и открытие веб-платформы Swagger	Открытие Swagger при запуске сервера по локальной сети	Успешно
Создание данных с помощью Swagger	Отправка данных с помощью Swagger через JSON	Правильная передача данных и занесение в PostgreSQL	Успешно
Изменение определенной записи с помощью Swagger	Найти запись с помощью идентификатора и изменить содержание	Правильная передача данных и изменение в PostgreSQL	Успешно
Удаление определенной записи с помощью Swagger	Найти запись с помощью идентификатора и удалить содержимое	Правильная передача данных и удаление в PostgreSQL	Успешно
Вывод всех данных с помощью Swagger	Найти все записи из таблицы	Правильная передача данных и получение из PostgreSQL	Успешно
Вывод определенной записи с помощью Swagger	Найти определенную запись из таблицы	Правильная передача данных и получение из PostgreSQL	Успешно
Запуск клиентской части с использованием технологии Vite	Запуск клиентской части и открытие веб-платформы Vite	Открытие Vite проекта при запуске клиентской части по локальной сети	Успешно
Запуск клиента с домашним интерфейсом	Запуск клиентской части с интерфейсом	Открытие Vite проекта без проблем	Успешно
Подключение к серверной части	Создание локального подключения	Информация логирования в консоли	Провалено
Регистрации пользователя в клиентскую часть	Запуск клиентской части, произвести регистрацию пользователя	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Авторизация пользователя в клиентскую часть	Запуск клиентской части, произвести авторизацию пользователя	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно

Продолжение таблицы 11

Название тестирования	Процесс тестирования	Ожидаемый результат	Фактический результат
Выход из системы после авторизации	Запуск клиентской части, произвести авторизацию пользователя	Обновления axios и логика работы клиентской части	Успешно
Авторизация в систему после выхода	После выхода из системы, нужно произвести вход в систему	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Провалено
Изменение пароля авторизованного пользователя	После авторизации, пользователь может изменить пароль	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Провалено
Поисковая навигация по курсам	Введение запроса на поиск курсов	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Провалено
Получение данных для фильтров курсов	При нажатии на фильтр, должен отображаться список	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Работа фильтров курсов	Выбор определенной категории запросов	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Провалено
Создание курсов	Нажать на кнопку и создать курс	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Изменение курсов	Нажатием на кнопку изменить курс	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Удаление курсов	Нажатием на кнопку удалить курс	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно

Продолжение таблицы 11

Название тестирования	Процесс тестирования	Ожидаемый результат	Фактический результат
Создание дополнительный страниц для курса	Нажатием на кнопку добавить страницу для курса	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Сохранение записей предыдущей страницы курса	Нажатием на кнопку предыдущая страница для курса	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Изменение дополнительной страниц для курса	Нажатием на кнопку изменить курс и выбрать страницу	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Сохранение страницы при создании курса	Нажатием на кнопку сохранить курс с дополнительными страницами	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Провалено
Просмотр бесплатных курсов	Нажатием на кнопку открыть бесплатный курс	Формирование правильной запроса и отправка на сервер с ожидающим ответа	Успешно
Просмотр платных курсов	Нажатием на кнопку открыть платный курс и произвести оплату	Формирование запроса и отправка на сервер с ожидающим ответа	Провалено
Создание записи истории курсов	Нажатием на кнопку открыть курс, и система автоматически добавит его в историю курсов	Формирование запроса и отправка на сервер с ожидающим ответа	Провалено
Создание записи избранный курс	Нажатием на кнопку добавить курс в избранный, и система автоматически добавит его в список избранные курсы	Формирование запроса и отправка на сервер с ожидающим ответа	Провалено

В момент проверки различных сценариев было выявлено, что девять тестов из тридцати одного теста не получили положительного заключения. По данным тестам, можно сделать вывод, что 66% программы работает правильно, однако 34% работает не правильна или вообще не работает.

Основная задача, сделать программный продукт рабочим на 100%, требуется выполнить отладку программного продукта и произвести повторное тестирование.

Первый не прошедший тест был связан с проблемой подключением к серверу из-за того, что не было правильного настроенного COM соединения сервера и клиентской части. После изменения данных для подключения клиентской части к серверу, а клиентской части к серверу, системы стали сообщаться друг с другом.

Второй не прошедший тест был связан с проблемой повторной авторизацией после выхода из системы, из-за чего, как только пользователь выходил из системы, клиентская часть не отправляла информацию о том, что пользователь вышел из системы на сервер. Сервер при получении данных авторизации считал, что старые данные, найденные в базе данных, являются актуальными и сверял на совпадение с введенными новыми данными. Для исправления требуется изменить логику сервера и клиентской части, чтобы при выходе из системы клиентская часть, отправляла запрос об этом серверу, чтобы сервер обновлял найденные данные. Теперь при авторизации, сервер автоматические обнуляет свой буфер, чтобы в будущем можно было авторизоваться. Проблема создавала ошибку не только для нынешнего пользователя, но и для других пользователей, так как сервер считал, что пользователь один и тот же. Дополнительно выяснилось, что в клиентской части неправильно указано подключение к данному контроллеру.

Третий не прошедший тест был связан с проблемой из-за не правильной работы контроллера, который не содержал в себе команда PUT для изменения данных, а также не правильное получение данных и логику работу всего контроллера.

Четвертый не прошедший тест был связан из-за не правильной логики в контроллере при поиске в базе данных.

Пятый не прошедший тест был связан из-за не правильной логики контроллера при сортировке данных с помощью LINQ.

Шестой не прошедший тест был связан из-за не правильной работы контроллера и формирование файла Markdown в формат бинарного кодирования для базы данных.

Седьмой не прошедший тест был связан из-за того, что контроллер неправильно сохранял данные в атрибуты в таблице многим ко многим.

Восьмой не прошедший тест был связан с тем, что клиентская часть не отправляла данные об открытие курсов серверу, тем самым контроллер не отправлял данные в таблицу базы данных.

Девятый не прошедший тест был связан из-за того, что контроллер не создавал запись в таблицу базы данных.

После исправления проблем, было произведено повторное тестирование программного продукта, и система прошла полностью все тесты, тем самым подтвердив свою 100% работоспособность. Тестирование проходило с помощью использование черного ящика, белого ящика и серого ящика.

Черный ящик - это тестирование функциональности приложения, не зная внутренний код программного продукта. В основном используется только входные и выходные данные, а также поведение системы.

Белый ящик - это тестирование функциональности приложения, а также имея доступ к внутренней структуре программы, но не имеет доступ к пользовательскому интерфейсу. В основном используется для проверки серверной части.

Серый ящик - это тестирование определенного модуля программного продукта. То есть, тестировщик может для начала проверить модуль интерфейса, а после проверить серверную часть данного модуля. В основном рассматривается проверка как частичная проверка определенного модуля клиентской или серверной части.

2.4 Документирование

Раздел "Документирование" включает в себя комплекс документов, направленных на обеспечение информации о настройке программного продукта для технического персонала, а также комплекс документов, направленный на обеспечение информации об использование программного продукта для пользователя.

Документация для технического персонала используется для того, чтобы технический персонал имел полное понимание архитектуры, использование технологий, различные зависимости и интерфейсах. Документация для технического персонала позволяет глубоко понять логику работы программного продукта. Сервер построен на базе PostgreSQL 17 и ASP.NET API на базе NET 9. Сервер и клиентская часть соединены по умолчанию по локальному соединению. Сервер использует порт 5226 для обычного соединения, и порт 7213 для защищенного соединения. Клиентская часть использует порт 5173. Для авторизации в систему PostgreSQL используется пароль по умолчанию «root».

Документация для пользователей программного продукта используется для того, чтобы пользователь имел понимание как взаимодействовать ему с этим веб-приложением. При открытии веб-приложения, пользователь увидит домашнюю страницу, на рисунке 38 изображена домашняя страница.

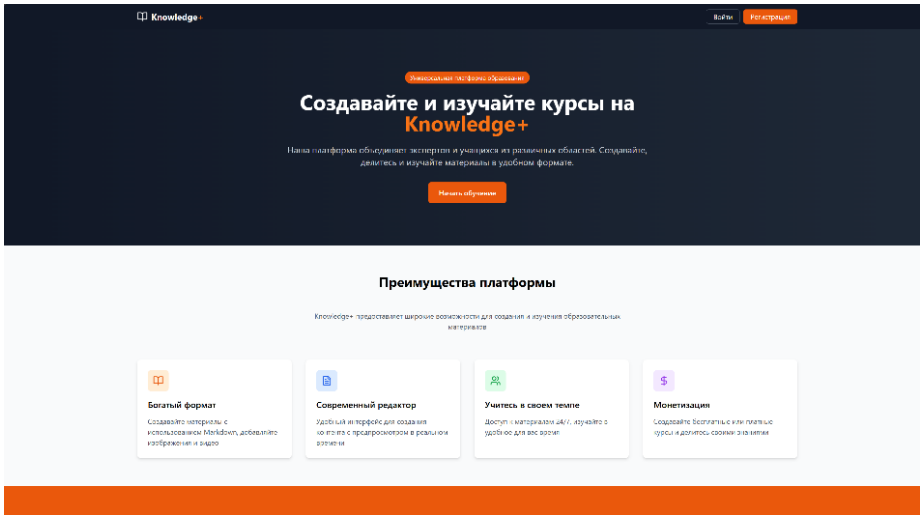


Рисунок 38 - Домашняя страница

Через главную страницу можно авторизоваться с помощью кнопки «Войти» или если нужно зарегистрироваться новому пользователю, то это можно сделать с помощью кнопки «Регистрация». Дополнительно, пользователи на этой страницы могут спуститься вниз и открыть страницы из маленьких глоссариев для пользователей, на рисунке 39 изображены глоссарий для пользователей.

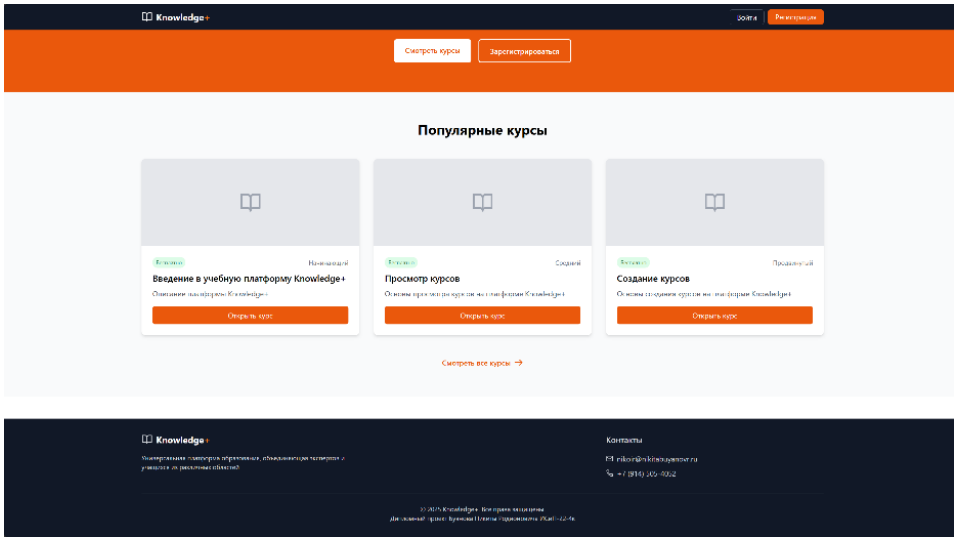


Рисунок 39 - Дополнительные глоссарии для пользователей

Глоссарии построены на базе обычных курсов, но они находятся только на домашней странице. Пользователь имеет право просматривать список курсов, но для того, чтобы открыть курс, требуется выполнить авторизацию в систему, на рисунке 40 изображено окно авторизации в систему.

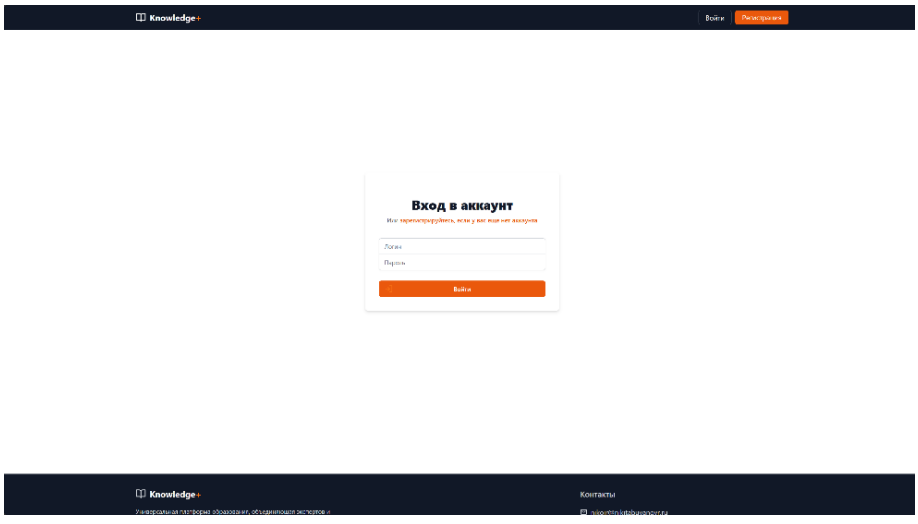


Рисунок 40 - Авторизация пользователя

В окне авторизации пользователь обязан ввести свою электронную почту и пароль. Если пользователь не имеет профиль в «Универсальной платформе образования Knowledge+», то он может произвести регистрацию, для этого можно в окне войти выбрать раздел «Регистрироваться», на рисунке 41 изображено окно регистрации в систему.

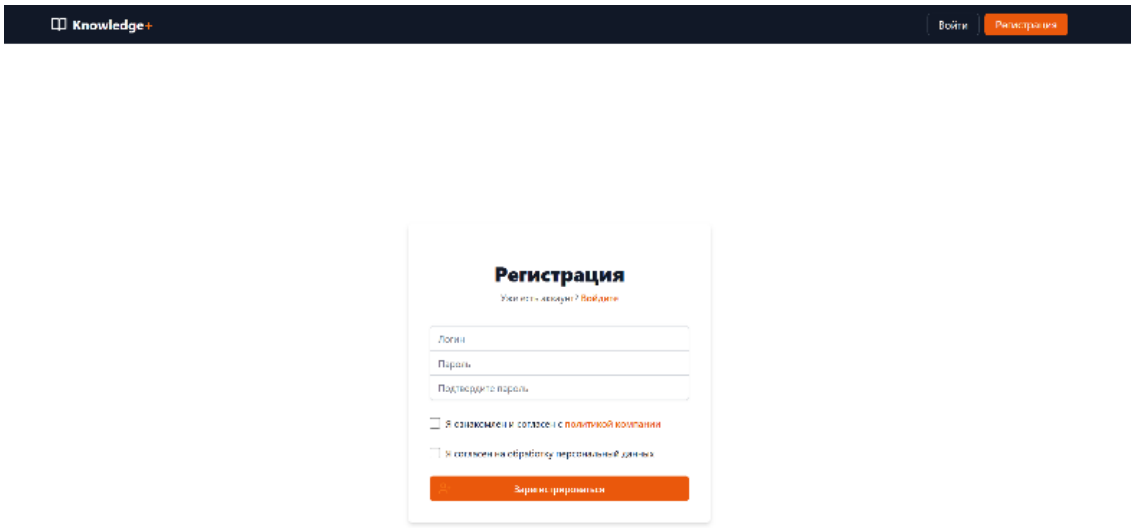


Рисунок 41 - Регистрация пользователя

В окне регистрации пользователь обязан ввести электронную почту, ввести пароль и повторить пароль, чтобы избежать ошибку для пользователя. Также при регистрации, пользователь обязан ознакомиться с политикой компании, где указаны все права пользователя и обязательства платформы перед пользователем. Также дополнительно при регистрации, пользователь обязан установить галочку, что пользователь согласен на обработку персональных данных, так как электронная почта является персональными данными. Как только пользователь ввёл электронную почту, будет произведена проверка на совпадение. После Регистрации, пользователя перебросит на домашнюю страницу, однако у него появится дополнительные кнопки в положении «Header». Авторизированный пользователь изображен на рисунке 42.

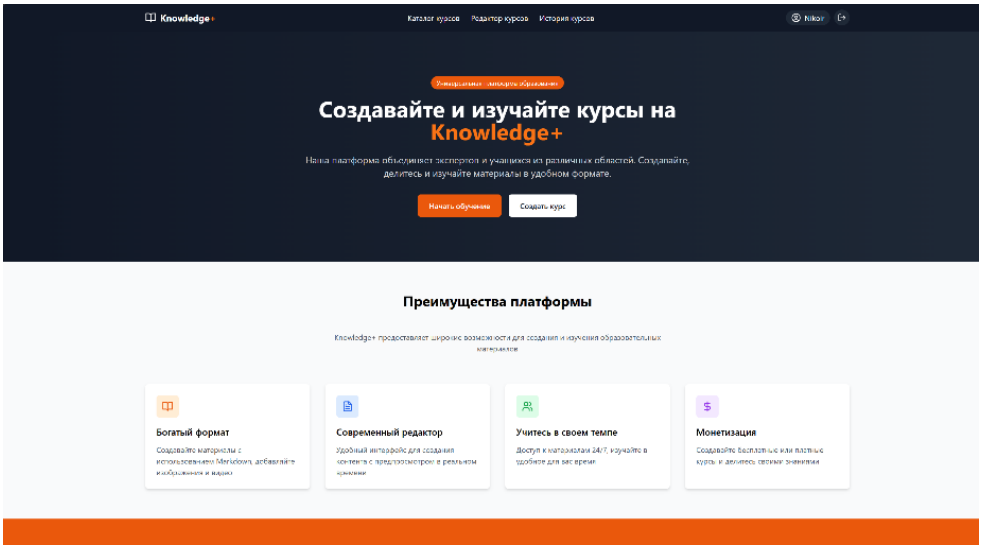


Рисунок 42 - Авторизированный пользователь

Для авторизированного пользователя появилась дополнительная кнопка «Создать курс», а также в верхней части веб-приложения появились дополнительные вкладки «Каталог курсов», «Редактор курсов», «История курсов».

Каталог курсов, это вкладка, которая отображает все существующие курсы в системе, где пользователь может открыть. Курсы можно отыскать с помощью поисковика или фильтров. Фильтры делятся на тему курса, возраст пользователя, уровень знаний для пользователя, а также указывается как формат курса (бесплатный/платный). При выборе платный, пользователь может установить от какой до какой стоимости показывать курсы в каталоге. При открытии платного курса, пользователь обязан выполнить оплату с помощью банковской карты, после этого платный курс будет доступен пользователю для открытия. Бесплатный курс открыт для авторизованных пользователей, так как система при открытие любых курсов будет добавлять их в список историй открытых курсов, и для этого требуется авторизацию пользователя, на рисунке 43 изображен раздел каталог курсов.

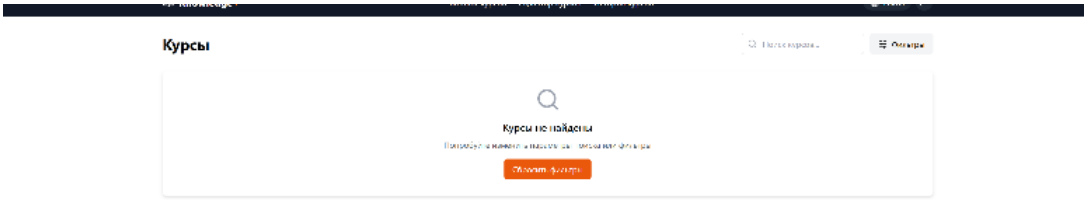


Рисунок 43 - Каталог курсов

В разделе «История курсов» можно посмотреть ранее открытые курсы пользователем, а также курсы, которые были добавлены в избранные, так как в данной вкладке содержатся курсы, которые пользователь добавил в избранные, на рисунке 44 изображена вкладка история курсов, на рисунке 45 изображена вкладка избранные.

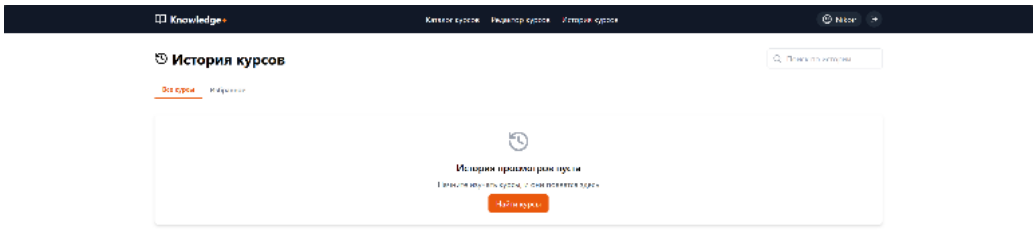


Рисунок 44 - Каталог истории курсов

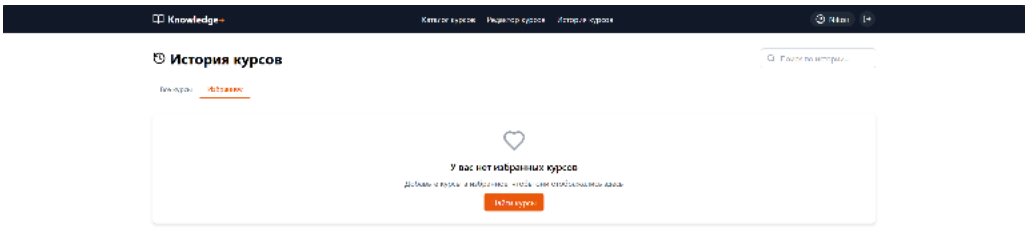


Рисунок 45 - Каталог избранных курсов

Любой авторизованный пользователь имеет полное право создать свой собственный курс с помощью конструктора в разделе «Редактор курсов». В

окне «Мои курсы» указывается список курсов, который были разработаны данным пользователем, в данной вкладки, пользователь может создать новый курс или отредактировать свой предыдущий курс. На рисунке 46 изображена вкладка курсов пользователя.

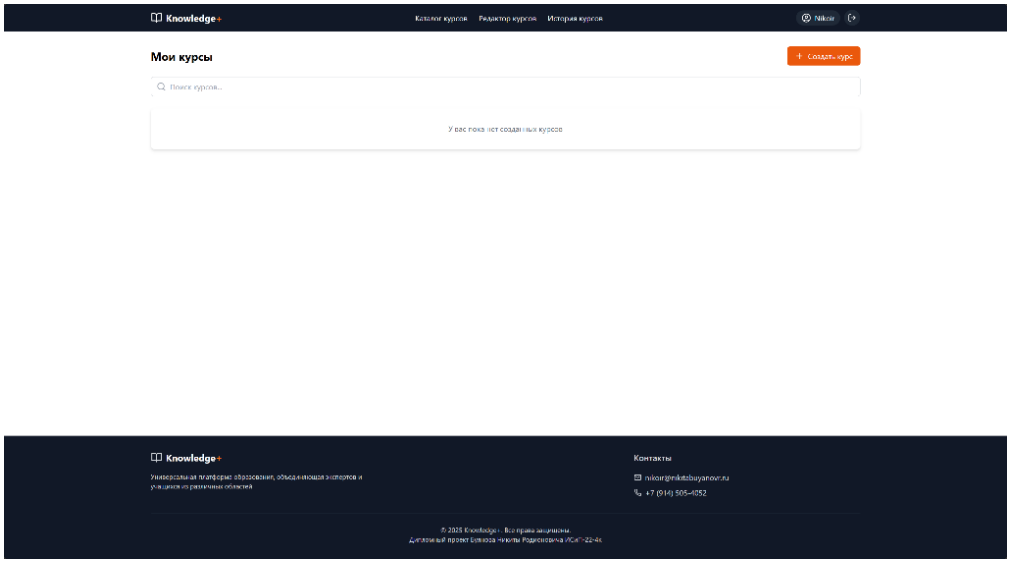


Рисунок 46 - Каталог редактора курсов

Создание курсов делится на три этапа. Первый этап является установки обозначения курса, т.е. его названия, после чего можно указать описание курса, а также указать изображение в виде иконки курса. На рисунке 47 изображен первый этап создания курса.

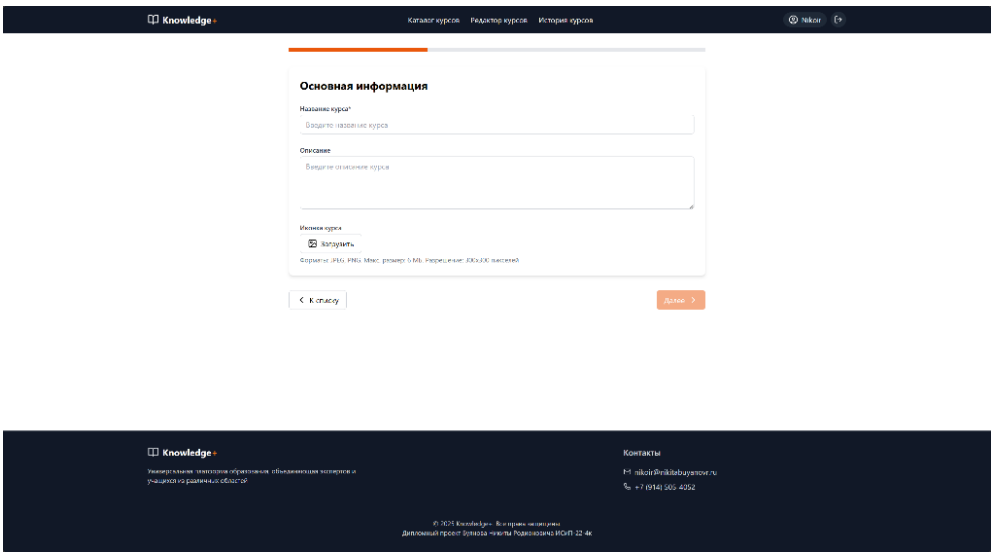


Рисунок 47 - Первый этап создания курсов

Второй этап, это написание курсов. Пользователь может писать страницы

с помощью Markdown и специального пользовательского редактора. Пользователю предоставляется панель инструментов, что позволит пользователям создавать красивые курсы без использования языка Markdown, однако пользователь может сам указываться в текст синтаксис. Чтобы пользователь мог использовать панель инструментов, ему требуется выделять определенный текст, и выбрать что нужно сделать из панели инструментов, после этого система автоматически форматирует его в заданный формат. Если пользователь хочет добавить изображение, то на панели инструментов нужно нажать кнопку добавить изображение, где откроется окно для загрузки изображения. Если пользователь хочет посмотреть, как выглядит страница курса в готовом виде, то ему необходимо нажать на глазик в правой части панели инструментов, чтобы открыть окно показа. На рисунке 48 изображена пустая страница курса, на рисунке 49 изображен пример написание страницы.

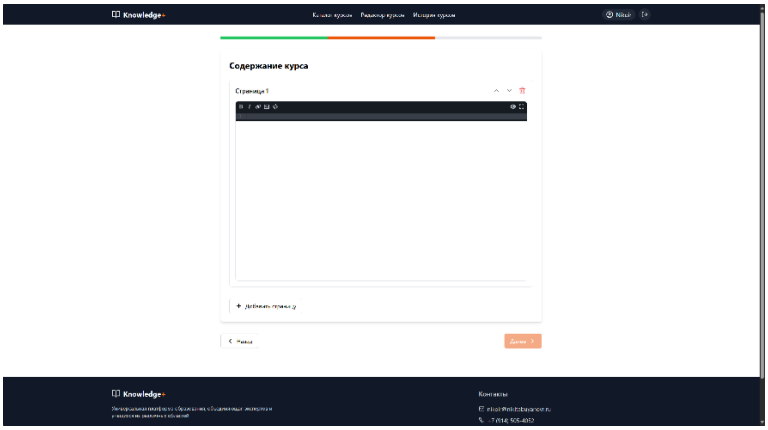


Рисунок 48 - Пример написание страницы

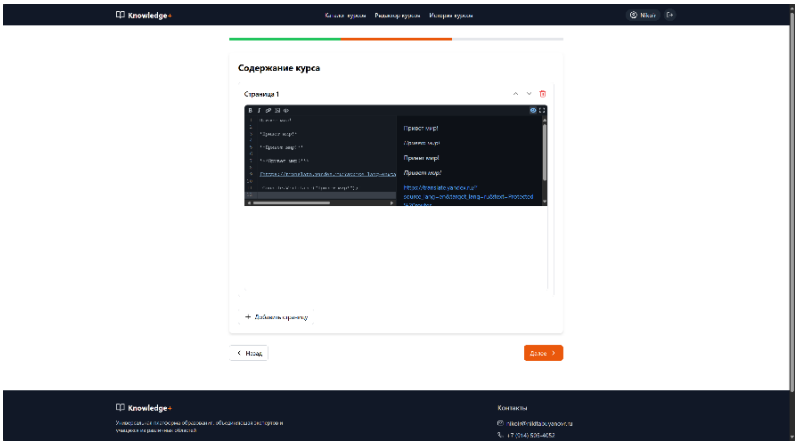


Рисунок 49 - Пример написание страницы

Если для пользователя панель курса становится не удобная, то в правом углу есть режим открыть на весь экран, для удобства, на рисунке 50 и 51 изображены примеры написание страницы на весь экран.

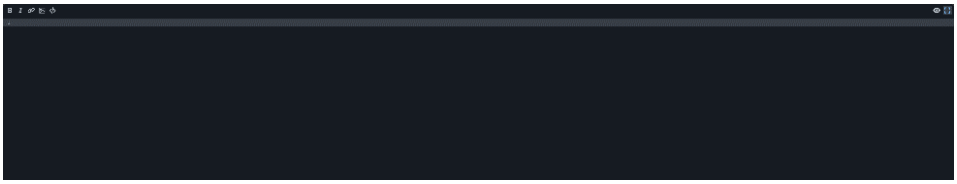


Рисунок 50 - Написание страницы в полном экране (пустое поле)

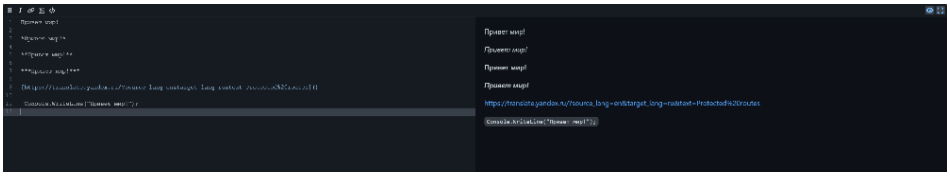


Рисунок 51 - Написание страницы в полном экране (заполненное поле)

Если пользователь хочет создать дополнительную страницу для курса, то ему нужно нажать на кнопку «Добавить страницу». После создания, пользователь может изменить последовательность страницы, и может удалить страницу, однако для курса требуется как минимум одна страница, тем самым система не позволит удалить полностью все страницы во время создания или редактирования курса. На рисунке 52 изображен пример написание курса с использованием второй страницы.

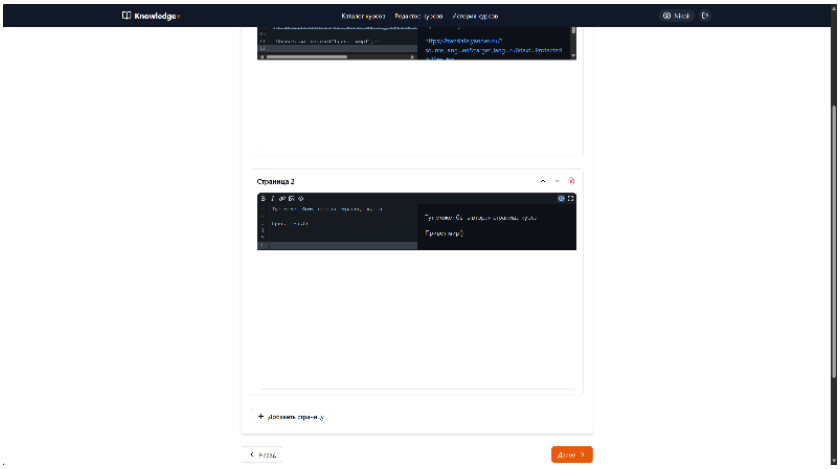


Рисунок 52 - Пример написание на второй странице для курса

Третий этап, это установка параметров курса при создании. Пользователь устанавливает открытость курса (бесплатный/платный), если курс является

платным, то пользователь устанавливает стоимость курса от 1000 до 20000 рублей. Пользователь устанавливает к какой категории относится его курс, а также устанавливает для каких возрастов подойдет данный курс, а также какой формат сложности является курс (начинающий/средний/эксперт), на рисунке 53 изображен пример третьего этапа создания курса.

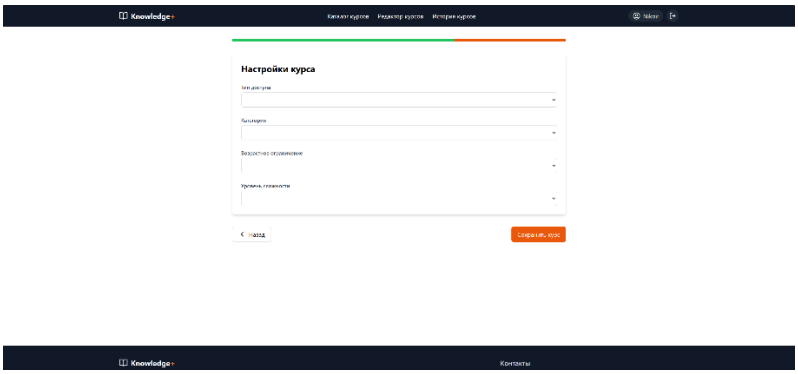


Рисунок 53 - Пример написание на третьей странице для курса

После создания курса, пользователь должен нажать кнопку «Сохранить курс», если всё будет хорошо, то сервер ответ о том, что курс создан и пользователя перенаправит во вкладку «Мои курсы».

2.5 Эксплуатация и сопровождение программного продукта

Данный пункт является описанием как развернуть «Универсальную платформу образования Knowledge+» и планы развития программного продукта.

Для эксплуатации и развертки программного продукта, требуется установить СУБД PostgreSQL с официального сайта, изображено на рисунке 54. Лучше использовать версию PostgreSQL 17, для точной совместимости с сервером.

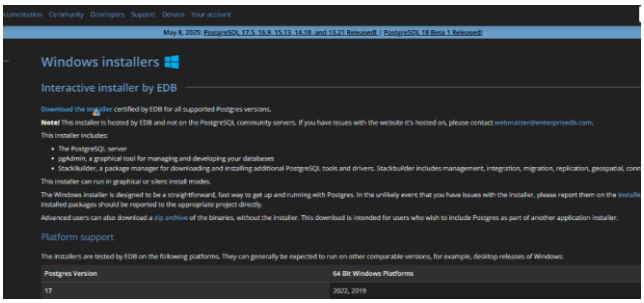


Рисунок 54 - Пример написание на третьей странице для курса

После скачивания, нужно произвести обычную установку, система установки СУБД устанавливает все нужные зависимости заранее, поэтому при установке, просто требуется всё время указать «Next» и «OK», однако при установки программного продукта, требуется установить пароль администратора для сервера. Если ничего не ввести, то пароля для сервера не будет, однако в серверной части устанавливается соединение к серверу, где указывается пароль для подключения к серверу СУБД. Так что требуется изменить в параметры подключения к серверу СУБД в серверной части. Однако, если не хочется изменять параметры подключения, то лучше всего установить пароль «root», что является негласным правилом программистов, как установка пароля по умолчанию. На рисунке 55 указано изображение, где изображено окно, где указывается пароль.

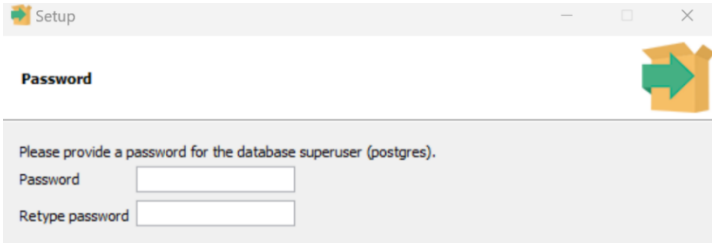


Рисунок 55 - Окно ввода пароля для сервера СУБД

После установки СУБД PostgreSQL, чтобы воспользоваться будем использовать установленный менеджер управления pgAdmin 4, который устанавливается при установке PostgreSQL, на рисунке 56 указана загрузка pgAdmin 4.

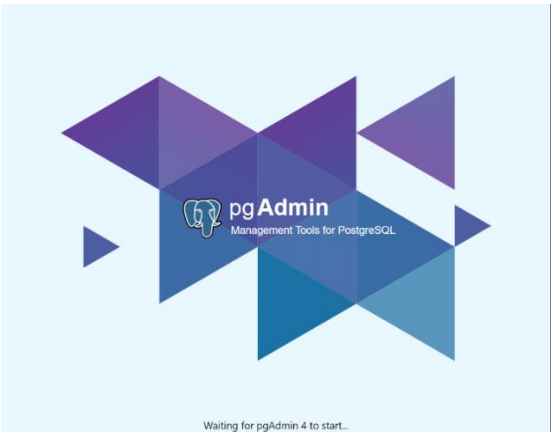


Рисунок 56 - Окно загрузки pgAdmin 4

После запуска и авторизации в систему, требуется создать базу данных под названием «Database-DiplomProject». Также как и с паролем, в параметрах подключения серверу к СУБД указывается название базы данных, так что при изменение названии, требуется изменить параметры подключения.

После создания базы данных, требуется выполнить запрос SQL для базы данных, который указан в разделе «Программирование программного продукта». После этого СУБД PostgreSQL будет работать правильно.

Для установки сервера ещё проще, требуется просто распаковать репозиторий и запустить его через Microsoft Visual Studio 2022, Docker или командной строки с правами администратора. Самое главное, требуется установить .NET 9 и SDK, которые можно установить с помощью командной строки от имени администратора, на рисунке 57 указана проверка установленных версий SDK и NET.

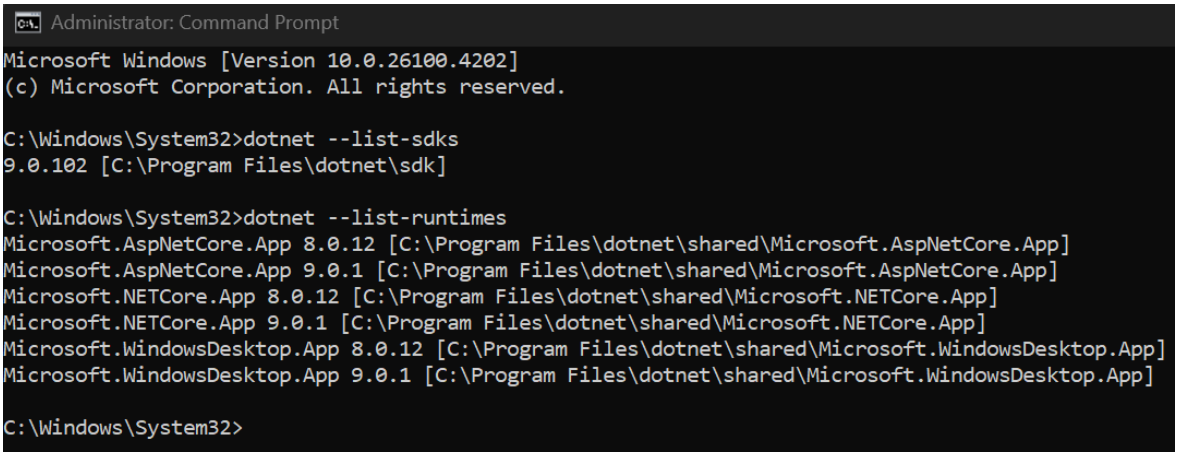


Рисунок 57 - Проверка установленных версий SDK и Net

Если на сервере нет определённой версии Net или установленных SDK, то нужно установить их с официального сайта. В основном для работы сервера требуется использовать «Microsoft.NETCore.App 9.0.1» и «Microsoft.AspNetCore.App 9.0.1», после установки снова проверим через консоль. Если все зависимости установлены, до запускаем через вышеуказанных способов.

Для установки клиентской части можно использовать WebStorm, NGINX, Apacher или Docker с правильными установленными контейнерами. Также

можно использовать командную строку с правами администратора. Для правильной работы клиентской части, требуется заранее запустить сервер. Сервер использует порты 5226 и 7213, а клиентская часть использует порт 5173, так что нужно убедиться, что эти порты не заняты в системе. Если при установке изменений не было, то всё должно запуститься и работать корректно.

Если следовать вышеуказанной инструкции, то программный продукт «Уникальная платформа образования» должно работать нормально и правильно.

Программный продукт после разработки всегда требуется обновлениям и модернизации, в будущем веб-приложение Knowledge+ может получить мобильную или компьютерную версию. В дальнейшем, можно добавить интерактивные проверки, игры, уроки или даже преобразовать в учебную программу для образовательных организаций бюджетного и коммерческого формата, тем самым предоставляя свои разработки организациям.

3. ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ ПРИ РАБОТЕ НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Работа на персональном компьютере является важной частью современной профессиональной деятельности, однако требует соблюдения правил охраны труда и техники безопасности для предотвращения негативного влияния на здоровье. Организация рабочего места играет ключевую роль в обеспечении комфортной работы. Рабочий стол должен быть достаточно просторным, чтобы на нем можно было разместить монитор, клавиатуру, мышь и другие необходимые предметы. Монитор должен находиться на расстоянии 50–70 см от глаз, а верхний край экрана должен быть на уровне глаз или чуть ниже. Клавиатура и мышь должны располагаться так, чтобы руки находились в естественном положении без напряжения. Стул должен быть регулируемым с поддержкой поясницы, а ноги должны стоять на полу или на подставке, образуя прямой угол в коленях.

Освещение рабочего места должно быть достаточным, но без бликов на экране монитора. Рекомендуется использовать рассеянный свет и избегать прямого попадания солнечных лучей на экран. При необходимости можно использовать настольную лампу с регулируемой яркостью. Режим работы должен включать регулярные перерывы каждые 45–60 минут на 5–10 минут, во время которых рекомендуется выполнять упражнения для глаз, шеи и спины. Каждые 20–30 минут следует отводить взгляд от экрана и смотреть вдаль на 20–30 секунд. Также важно регулярно выполнять упражнения для улучшения кровообращения и снятия напряжения в мышцах.

Гигиена и эргономика рабочего места также имеют большое значение. Необходимо следить за чистотой рабочего пространства и регулярно протирать монитор, клавиатуру и мышь. Использование эргономичных аксессуаров, таких как подставки для запястья и подставки для монитора, помогает снизить нагрузку на мышцы и суставы.

Электробезопасность является важным аспектом работы с компьютером.

Необходимо убедиться, что все кабели и розетки находятся в исправном состоянии, и избегать перегрузки электрических сетей. В случае возникновения неисправностей следует немедленно отключить оборудование и обратиться к специалисту.

Психологический комфорт также играет важную роль в обеспечении эффективной работы. Создание комфортной атмосферы на рабочем месте, отсутствие шума и других отвлекающих факторов способствуют повышению производительности. Важно соблюдать баланс между работой и отдыхом, чтобы избежать переутомления.

Все сотрудники должны проходить регулярный инструктаж по охране труда и технике безопасности при работе на компьютере. Необходимо ознакомиться с инструкциями по эксплуатации оборудования и строго соблюдать их. Соблюдение этих правил помогает минимизировать риски для здоровья, повысить производительность труда и создать комфортные условия для работы на персональном компьютере. Забота о своем здоровье является залогом успешной и долгосрочной профессиональной деятельности.

ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта была разработана веб-платформа «Универсальная платформа образования» под названием «Knowledge+», призвана на создание самообразовательной платформы для создания и просмотра учебных курсов. Основной целью разработки было создание удобного и интуитивно понятным интерфейсом и составляющими пользователя, способном удовлетворить пользователя.

Веб-платформа «Knowledge+» разработана с использованием современных технологий, одним из технологий является современная и реляционная СУБД PostgreSQL. Для сервера использовалась современная платформа ASP.NET API с использованием языка программирования C# и LINQ. Использование современных технологий при создании клиентской части платформы, основными технологиями является Vite, для быстрой компиляции проекта и библиотеку React, для быстрой и удобной разработки пользовательского интерфейса с использованием HTML, CSS, Markdown, TypeScript, JavaScript, а также для правильной работы CSS в проекте с Vite использовалась технология Tailwind в связки PostCSS.

Функциональная платформа включает в себя создание бесплатных и платных онлайн курсов любой темы. Просмотр бесплатных или платных курсов для авторизованных пользователей.

Разработка и тестирование платформы показала, что учебная платформа «Knowledge+» успешно справляется с задачами, связанная с самостоятельным образованием.

Реализация проекта подтвердила важность системного подхода к проектированию образовательных платформ, включая продуманное распределение ролей пользователей и их прав доступа. Разработка детальных схем прецедентов и последовательностей действий позволила точно определить ключевые элементы взаимодействия, что обеспечивает защиту данных и эффективную работу всех компонентов системы.

В ходе выполнения дипломного проекта, была исследована и собрана онлайн статистика из различных онлайн форумах и научных статей об образовании. Была собрана информация о существующих образовательных программах и платформах.

В ходе проектирования и реализации платформы были учтены основные тенденции в области цифрового образования, такие как потребность в гибкости, доступности учебного процесса и персонализированном подходе к обучению. Благодаря этому платформа универсальная платформа образования «Knowledge+» может стать эффективной веб-платформой будущего для использована повышения знаний и возможностей людей.

В будущем платформа имеет потенциал для дальнейшего развития и может, что позволит интегрировать дополнительные функции, отвечающие новым требованиям образовательного процесса.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ 2.105-79. Общие требования к текстовым документам. - М.: Издательство стандартов. 1979. - 36 с.

2. ГОСТ 2.304-81. Шрифты. - М.: Издательство стандартов. 1981. - 57 с.

3. Зыков Н.В. единые требования к оформлению курсового и дипломного проекта (работы): методические указания для студентов очного и заочного обучения всех специальностей – 4-е изд., испр. и доп. /Н.В. Зыков. – Чита: ЗабГК, 2020 – 53с.

4. Беликова С. А. Основы HTML и CSS: проектирование и дизайн веб-сайтов : учебное пособие по курсу «Web-разработка» / С. А. Беликова, А. Н. Беликов ; Южный федеральный университет. – Ростов-на-Дону ; Таган-рог : Южный федеральный университет, 2020. – 176 с.

5. Брылёва А. А. Программные средства создания интернет-приложений : учебное пособие / А. А. Брылёва. – Минск : РИПО, 2019. – 381 с.

6. Вагин Д. В. Современные технологии разработки веб-приложений : учебное пособие / Д. В. Вагин, Р. В. Петров ; Новосибирский государственный технический университет. – Новосибирск : Новосибирский государственный технический университет, 2019. – 52 с.

7. Волкова Т. И. Введение в программирование : учебное пособие / Т. И. Волкова. – Москва ; Берлин : Директ-Медиа, 2018. – 139 с.

8. Гудов А. М. Администрирование систем управления базами данных : учебное пособие / А. М. Гудов, И. Ю. Степанов ; Кемеровский государственный университет. – Кемерово : Кемеровский государственный университет, 2022. – 167 с.

9. Долженко А. И. Разработка и сопровождение программных систем : технологии Microsoft.NET для разработки приложений : практикум / А. И. Долженко, С. А. Глушенко ; Ростовский государственный экономический университет (РИНХ). – Ростов-на-Дону : Издательско-полиграфический комплекс РГЭУ (РИНХ), 2019. – 140 с.

10. Долженко А. И. Технология Micrisoft ADO.Net и платформа Entity Framwork : учебное пособие / А. И. Долженко, С. А. Глушенко ; Ростовский государственный экономический университет (РИНХ). – Ростов-на-Дону : Издательско-полиграфический комплекс РГЭУ (РИНХ), 2021. – 191 с.

11. Лисяк В. В. Моделирование информационных систем : учебное пособие / В. В. Лисяк, Н. К. Лисяк. – Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2018. – 89 с.

12. Маркин А. В. SQL-программирование в Ред База Данных : учебное пособие / А. В. Маркин. – Москва : б.и., 2023. – Часть 1. – 420 с.

13. Маркин А. В. SQL-программирование в Ред База Данных. : учебное пособие / А. В. Маркин. – Москва : б.и., 2023. – Часть 2. – 377 с.

14. Марухленко А. Л. Разработка защищённых интерфейсов Web-приложений : учебное пособие / А. Л. Марухленко, Л. О. Марухленко, М. А. Ефремов. – Москва ; Берлин : Директ-Медиа, 2021. – 175 с.

15. Моргунов А. В. Управление Веб-технологиями, сервисами и контентом : учебное пособие / А. В. Моргунов. – Новосибирск : Сибирский государственный университет телекоммуникаций и информатики, 2021. – 88 с.

16. Нагаева И. А. Основы web-дизайна. Методика проектирования : учебное пособие / И. А. Нагаева, А. Б. Фролов, И. А. Кузнецов. – Москва ; Берлин : Директ-Медиа, 2021. – 236 с.

17. Онопенко Г. А. Базы данных : учебное пособие / Г. А. Онопенко, Н. А. Вихорь ; Томский государственный архитектурно-строительный университет. – Томск : Томский государственный архитектур-но-строительный университет (ТГАСУ), 2019. – 104 с.

18. Сидорова Н. П. Базы данных : практикум по проектированию реляционных баз данных : учебное пособие / Н. П. Сидорова ; Технологический университет, Институт техники и цифровых технологий, Факультет инфокоммуникационных систем и технологий. – Москва ; Берлин : Директ-Медиа, 2020. – 93 с.

19. Солодушкин С. И. Web и HTML : учебное пособие / С. И.

Солодушкин, И. Ф. Юманова ; науч. ред. В. Г. Пименов ; Уральский федеральный университет им. первого Президента России Б. Н. Ельцина. – Екатеринбург : Издательство Уральского университета, 2018. – 131 с.

20. Червенчук И. В. Моделирование объектно ориентированных систем с помощью UML : учебное пособие / И. В. Червенчук, А. С. Грицай ; Омский государственный технический университет. – Омск : Омский государственный технический университет (ОмГТУ), 2019. – 108 с.

ПРИЛОЖЕНИЕ 1 Код программы

Приложения 1

Код программы

продолжение Приложения 1

окончание Приложения 1