ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

1 СИСТЕМЫ ПЕРСОНАЛЬНОГО ПЛАНИРОВАНИЯ И КОНТРОЛЯ ПРОЦЕССОВ ОБУЧЕНИЯ

1.1 Методы персонального планирования и контроля обучения

Методы планирования обучения необходимо использовать, чтобы выровнять все аспекты обучения так, чтобы они соответствовали друг другу. График должен быть соразмерен времени, установленному для изучения той или иной темы, и все его ресурсы должны использоваться оптимальным образом. Учитывая изменчивый характер процесса обучения и, иногда, его масштаб, его сложно спланировать, но это необходимо сделать, потому что без какого-либо плана вероятность успешного завершения обучения очень мала.

График состоит из всех действий, включённых в процесс обучения в течение заранее определённого периода времени. График обучения помогает расставить приоритеты в работе над проектом и завершить ее упорядоченным образом. Это также помогает в надлежащем распределении доступных ресурсов, главным из которых является время. Управление временем и корректировка в рамках обучения возможны только при наличии надлежащего графика.

Планирование обучения можно сравнить с планированием хода выполнения проекта, так как обучение это проект определённого человека. Планирование проекта обучения, как правило, включает в себя различные методы, краткое описание которых приводится ниже.

Первыми методами планирования проектов являются методы математического анализа: метод критического пути и метод оценки и анализа программ. Метод критического пути (*CPM*) и метод оценки и анализа программ (*PERT*) являются двумя наиболее часто используемыми методами для руководителей проектов. Эти методы используются для расчёта времени выполнения проекта. В данном случае руководителем проекта выступает человек, который хочет получить новые знания и умения.

Древовидная диаграмма каждого проекта обучения имеет критический путь. Метод критического пути оценивает максимальное и минимальное время, необходимое для завершения проекта. *CPM* также помогает определять критические задачи, которые должны быть включены в проект.

PERT – это способ запланировать поток задач в проекте и оценить общее время, необходимое для его выполнения. Этот метод помогает понять, как каждая задача зависит от другой. Чтобы запланировать проект с использованием *PERT*, необходимо определить виды деятельности, упорядочить их и определить основные этапы.

Следующим методом планирования проектов является моделирование. В данном методе ожидаемая продолжительность проекта рассчитывается с использованием другого набора задач в симуляции. Расписание создаётся на основе предположений, поэтому его можно использовать, даже если область действия изменилась или задачи недостаточно ясны, что очень актуально при обучении.

Далее необходимо выделить список задач. Он является одним из самых простых методов планирования проектов обучения. Для его реализации необходимо в электронной таблице или текстовом редакторе привести список всех возможных задач, связанных с проектом. Этот метод является простым и самым популярным из всех методов. Это очень полезно при реализации небольших проектов. Но для больших проектов с многочисленными аспектами рассмотреть список задач не представляется возможным.

Также одним из самых популярных методов планирования проектов является диаграмма Ганта. Диаграмма Ганта представляет собой метод визуализации, используемый в управлении проектами. Он используется менеджерами проектов большую часть времени, чтобы получить представление о среднем времени, необходимом для завершения проекта. В данном случае менеджером проекта является обучающийся. График Ганта – это столбчатая диаграмма, которая представляет ключевые действия в последовательности слева и в зависимости от времени. Каждая задача представлена полосой, которая отражает начало и конец действия в проекте обучения, то есть его продолжительность. Данный метод позволяет ограничить время на изучение конкретных областей знаний, вследствие чего обучающийся всегда находится в жёстких рамках и не занимается искусственным увеличением времени выполнения проекта обучения.

И заключительным популярным методом планирования проектов является календарь. Большинство календарей написаны для персональных нужд работников проекта, что отлично подходит для персонального обучения. Календарь показывает временную шкалу для всего проекта. Основным преимуществом является то, что он может подвергаться изменениям, поскольку он является разделяемым.

Также как и планирование обучения, контроль знаний и умений является важным элементом процесса обучения. Результативность процесса обучения во многом зависит от тщательности разработки методики контроля знаний. Контроль знаний необходим при всякой системе обучения и любой организации учебного процесса. Это средство управления учебной деятельностью обучающихся. Но для того, чтобы, наряду с функцией проверки, реализовались и функции обучения, необходимо создать определённые условия, важнейшее из которых – объективность проверки знаний.

Объективность проверки знаний предполагает корректную постановку контрольных вопросов, вследствие чего появляется однозначная возможность отличить правильный ответ от неправильного. Кроме того, желательно, чтобы форма проверки знаний позволяла легко выявить результаты.

Существует несколько традиционных форм контроля знаний и умений:

– устный или письменный опрос;
– карточки;
– краткая самостоятельная работа;
– практическая или лабораторная работа;
– тестовые задания.

Устная проверка знаний подразумевает наличие человека, проводящего контроль. Устная проверка может быть в форме фронтальной беседы, когда проверяющий задаёт вопросы всем учащимся. При этом происходит непосредственный контакт этого человека с людьми. При опросе кого-либо из обучающихся все остальные должны внимательно следить за ответом, поправляя и дополняя его. Устная фронтальная проверка не позволяет установить всю глубину усвоенных понятий, но зато в течение короткого времени проверяющий человек уточняет, насколько люди усвоили основные представления об изучаемом материале или объекте, умеют ли обобщать и систематизировать знания, устанавливать связи.

Работе с карточками придаётся особое значение, так как такая проверка знаний даёт возможность дифференцированно подойти к обучающимся, проверить знания большого количества людей.

Карточки, которые предлагаются учащимся, могут быть очень разными по содержанию, объёму, оформлению. Кроме того, следует сделать карточки для сложного, среднего и простого уровней, что позволяет использовать «зону ближайшего развития» каждого человека.

Если же речь идёт о самообучении, то карточками могут быть различного вида объекты, содержащие вопрос по определённой теме и скрытый ответ на эту тему. Обучающийся сам составляет их и использует для контроля своих знаний. В данном случае этот метод, кроме функции контроля, выполняет функцию запоминания, так как повторение информации через определённые промежутки времени – это самый естественный и эффективный способ запомнить материал.

Письменная проверка знаний – распространённая форма контроля знаний и умений обучающихся. Она представляет собой перечень вопросов, на которые обучающиеся должны дать незамедлительные и краткие ответы. Время на каждый ответ строго регламентировано и достаточно мало, поэтому сформулированные вопросы должны быть чёткими и требовать однозначных, не требующих долгого размышления, ответов. Именно краткость ответов отличает его от остальных форм контроля. С помощью письменной проверки можно проверить ограниченную область знаний обучающихся: буквенные обозначения, названия единиц, определения, формулировки, связь между величинами, формулировки научных фактов. Именно эти знания могут быть проверены в быстрых и кратких ответах. Письменная проверка не позволяет проверить умения, которыми овладели люди при изучении той или иной темы. Таким образом, быстрота проведения письменной проверки является одновременно как его достоинством, так и недостатком, т.к. ограничивает область проверяемых знаний. Однако эта форма контроля снимает часть нагрузки с остальных форм, а также может быть с успехом применена в сочетании с ними.

При кратковременной самостоятельной работе обучающимся также задаётся некоторое количество вопросов, на которые предлагается дать свои обоснованные ответы. В качестве заданий могут выступать теоретические вопросы на проверку усвоенных знаний; задачи, на проверку умения выполнить

расчёты по заданию; задания по моделированию (воспроизведению) конкретных ситуаций, соответствующих технологическим понятиям. В самостоятельной работе могут быть охвачены все виды деятельности кроме создания понятий, т.к. это требует большего количества времени. При этой форме контроля обучающиеся обдумывают план своих действий, формулируют и записывают свои мысли и решения. Понятно, что кратковременная самостоятельная работа требует гораздо больше времени, чем предыдущие формы контроля, и количество вопросов может быть не более трёх, а иногда самостоятельная работа состоит и из одного задания.

Практическая или лабораторная работа – достаточно необычная форма контроля, она требует от обучающихся не только наличия знаний, но ещё и умений применять эти знания в новых ситуациях, сообразительности. Лабораторная работа активизирует познавательную деятельность людей. Практическую лабораторную работу целесообразно комбинировать с такими формами контроля, как письменная проверка или тест. Такая комбинация может достаточно полно охватить знания и умения обучающихся при минимальных затратах времени, а также снять при этом трудность длинных письменных высказываний.

В тестовых работах предлагается несколько, обычно четыре или пять, вариантов ответов на вопрос, из которых надо выбрать правильный. Эта форма контроля тоже имеет свои преимущества, неслучайно это одна из наиболее распространённых форм контроля. Обучающиеся не теряют времени на формулировку ответов и их запись, что позволяет охватить большее количество материала за то же время.

Несмотря на все очевидные достоинства, тестовые задания имеют ряд недостатков. Главный из них – это трудность формулирования вариантов ответов на вопросы при их составлении. Если ответы подобраны без достаточного логического обоснования, большинство обучающихся очень легко выбирают требуемый ответ, исходя не из имеющихся у них знаний, а только лишь из простейших логических умозаключений и жизненного опыта. Поэтому бывает трудно или даже невозможно составить удачный тест без теоретической подготовки. Следует также отметить, что тестовые задания дают возможность проверить ограниченную область знаний обучающихся, оставляя в стороне деятельность по созданию объектов труда, воспроизведению конкретных действий, соответствующих практическим навыкам и т.п. По результатам выполнения тестов нельзя проверить умения обучающихся решать комбинированные задачи, а также способности построения логически связанного ответа в устной форме.

Итак, эффективность планирования обучения, контроля знаний и умений во многом зависит от умения правильно организовать обучение и грамотно выбрать ту или иную форму проведения контроля. И для правильной организации планирования и контроля обучения существует большое количество разработанных программных решений.

1.2 Сравнительный анализ существующих программных решений для планирования и контроля процессов обучения

Для всех вышеперечисленных методов планирования проектов существует многочисленное и разнообразное программное обеспечение. Одни программы являются бесплатными, другие коммерческими, третьи предоставляют пробный период. Все эти программы отличаются и пользователь может выбирать то, что ему нравится больше, исходя из собственных предпочтений.

Первым рассмотренным приложением для планирования персональными проектами является *Any.DO*. Программа *Any.DO* – один из самых популярных планировщиков среди пользователей *Android* и *iOS*. Приложение отличается удобным и простым интерфейсом и может синхронизироваться с несколькими устройствами. Для добавления заданий в приложении можно пользоваться голосовым набором – при этом включается интеллектуальный ввод, позволяющий выбрать запись из готовых вариантов.

Преимущества планировщика *Any.DO*:

– ввод текста с возможностью прикрепления к нему видеофайлов, изображений или фото – возможность, отсутствующая у большинства похожих утилит;

– простое добавление пользовательских списков из главного меню;

– удобное переключение между режимами;

– работа с разными видами данных – в том числе списками дел, покупок.

Приложение работает не только на смартфонах, но и в браузере. Интеграция с операционной системой позволяет ему выдавать сообщения прямо в строке состояния. Среди других плюсов стоит отметить многопользовательскую работу, защиту информации с помощью кода и геотеги. Базовая версия приложения бесплатна, но функциональность можно расширить, используя платную версию.

Следующим приложением для планирования является *Todoist*. Программа *Todoist* содержит целый комплекс полезных функций, позволяющих контролировать выполнение любых задач. С её помощью можно вести статистику, составлять различные списки, синхронизировать несколько устройств, на которых установлен планировщик.

Список возможностей приложения включает:

– добавление задач прямо с рабочего стола мобильного устройства – для этого достаточно использовать виджеты;

– настройку напоминаний;

– запись комментариев и родительских задач;

– контроль выполнения задач в статистике профиля.

Главные преимущества приложения - целый ряд доступных пользователю цветовых схем, автоматическое перемещение новых задач в список входящих, удобные фильтры и метки, позволяющие отсортировать задания. Ещё один важный плюс – поддержка более чем десяти платформ и приложений, в которые интегрируется *Todoist* – среди них *MacOS*, *Windows*, *iOS*, *Android*, *браузеры Firefox* и *Chrome*.

Далее была рассмотрена программа *GTasks*. Планировщик *GTasks* отличается удобством использования и возможностью синхронизации с несколькими устройствами с помощью облачных сервисов. Для этого пользователю придётся авторизоваться в программе через аккаунт *Google*. Причём, несмотря на то, что приложение выпущено самой *Google*, пользоваться им могут также владельцы *iOS*.

Основные особенности программы:

– синхронизация с *Google* Календарем, из которого можно импортировать данные;

– работа в локальном режиме, если все сведения хранятся на устройстве;

– стильный дизайн и удобный интерфейс;

– голосовой ввод заданий.

Преимущества утилиты состоят в мультиплатформенности и удобном отображении информации. Списки, которые пользователь добавляет в *GTasks*, легко добавляются, скрываются и удаляются. При подключении платной версии становится доступным ещё и резервное копирование, смена оформления и блокировка данных ключом.

После этого был рассмотрен визуальный инструмент планирования *Sectograph*. *Sectograph* – это практичный визуальный помощник, выдающий список дел в виде циферблата часов. Схема *Sectograph* позволяет не только узнать, какие дела запланированы на день, но и увидеть, сколько осталось до их предполагаемого начала или завершения. События подгружаются из *Google*-календаря, с которым синхронизирован планировщик.

Функциональность приложения включает:

– добавление ежедневных дел;

– оригинальный таймер поездок и длительности авиаперелётов;

– возможность планировать приёмы лекарств и пищи;

– отслеживание времени, потраченного на тренировку;

– совместную работу с системой Android Wear на смарт-часах и фитнес-браслетах.

Преимуществами утилиты является встроенный в неё функциональный таймер, обратный отсчёт и контроль хронометража различных действий. Эти особенности будут полезными для спортсменов, путешественников и деловых людей. При этом планировщик также может синхронизироваться с компьютерами *Apple*, *Windows* и телефонами на базе *Android* и *iOS*.

Последним популярным кандидатом является приложение *Trello*. Приложение *Trello* представляет собой продукт, созданный по тому же принципу, что и программное обеспечение для управления крупными проектами. С его помощью можно создавать задачи для коллектива, семьи и даже интернет-магазина.

Возможности *Trello*:

– создание списков задач для индивидуального и общего использования;

– приглашение в группу коллег, членов семьи и друзей;

– назначение заданий другим пользователям;

– ответы на комментарии;

– привязка карточек к координатам на карте;

– визуализация задач.

Программа поддерживается различными операционными системами – *Microsoft Windows*, *macOS*, *iOS* и *Android*. Среди её плюсов – возможность следить за проектами, простой и удобный интерфейс, загрузка файлов и настройка сроков завершения проекта. При этом планировщик не требует платы за использование.

Сравнительная десятибальная характеристика приложений для персонального планирования проектами обучения и разработанного приложения, приведена в таблице 1.1.

Таблица 1.1 – Характеристика приложений для планирования проектов

Параметры	*Any.DO*	*Todoist*	*GTasks*	*Sectograph*	*Trello*	Разработанное приложение
Доступность	6	10	8	10	6	10
Понятный интерфейс	5	7	6	9	7	9
Поддержка устройств	10	10	6	6	10	10
Вложения	7	0	0	0	10	8
Простота	4	9	6	8	7	10
Функционал	9	4	5	5	8	6
Визуализация	4	4	4	9	7	8
Итоги	45	44	35	47	55	61

Из результатов видно что приложения существенно отличаются друг от друга. Большинство функций приложений являются бесплатными, однако в некоторых приложениях часть функций заблокирована либо присутствует реклама. Современные приложения практически все имеют неплохой графический интерфейс, однако большое количество функционала делает его более сложным, чем он должен быть.

Тяжело совместить большое количество функционала и простоту использования приложения. Поэтому пользователь должен сам проанализировать и решить что ему подходит больше.

Часть приложений не имеет браузерной версии и доступна только в мобильных магазинах приложений, что сильно ограничивает их применение. Многие приложения предоставляют большой функционал, но не содержат вложений для задач, что является большой проблемой для эффективной организации проекта.

Разработанное приложения является узконаправленным для управления персональными проектами обучения, поэтому не содержит большого

количества функционала, однако имеет все необходимые функции для реализации эффективного процесса обучения. Это позволяет ему занять свою нишу и предоставить пользователю более простой и линейный пользовательский интерфейс при том, что организация проектов в приложении имеет древовидную структуру.

Вместе с приложениями для планирования и управления проектами обучения часто используются и приложения для контроля этого обучения, так как невозможно правильно оценить процесс обучения без методов контроля.

Первой популярной программой для контроля обучения является *Study Smarter*. В этом приложении можно создавать обучающие карточки и учебные заметки в очень быстрые сроки. Также оно даёт доступ к общим учебным материалам и учебникам от ведущих издателей. Кроме того в этом приложении можно настроить свой индивидуальный учебный план.

Из особенностей приложения можно выделить:

– возможность создавать карточки для запоминания;
– функционал по созданию и прохождению тестов;
– возможность сохранять и просматривать конспекты лекций;
– создание учебных заметок и пособий;
– проектирование своего плана обучения;
– возможность устанавливать таймеры для напоминания об обучении;
– функционал создания учебных групп пользователей;
– возможность загружать и создавать учебники.

Из минусов приложения необходимо выделить поддержку приложения только на мобильных устройствах. А также приложение не имеет поддержки русского языка.

Следующее приложение для обучения является более специализированным, так как предназначено для контроля обучения программистов. Это приложение называется *Solo Learn*. Оно позволяет людям, которые хотят обучиться программированию сделать это проще и веселее. Приложение поддерживается всеми платформами, переведено на русский язык и также имеет привлекательный и простой интерфейс пользователя.

Это приложение сконцентрировано на практических навыках, так как пользуясь им человек начинает писать первый код сразу же после начала обучения. Также приложение имеет очень хорошо организованные подзадачи, что позволяет пользователю концентрироваться на небольших частях обучения.

Приложение содержит ряд курсов, в конце которых пользователь получает сертификат о прохождении и готовый проект в резюме. Эти курсы направлены как на профессионалов, которые хотят изучить новые технологии, так и на студентов, которым необходимо, например, сдать экзамен.

Из особенностей приложения необходимо выделить следующее:

– направленность на практику;
– чёткое направление обучения – программирование;
– современный и удобный пользовательский интерфейс;
– кроссплатформенность;
– коммуникации с людьми в процессе обучения;

– своя песочница для работы с кодом;

– интерактивность;

– функционал достижений.

Минусы приложения следуют из его плюсов. Оно является очень узконаправленным и подходит только для обучения связанного с программированием. Однако явных минусов приложение не имеет.

Следующим приложением является *Khan Academy*. Это приложение позволяет получить доступ к большой базе знаний из любой точки мира. Это бесплатный сервис, работающий на благотворительности и не требующий покупки каких-либо услуг.

В данном приложении можно заниматься как самообучением, так и обучением с учителем. Для этого учителю необходимо зарегистрировать свой класс и добавить в него учеников. Кроме учителя и ученика в приложении также есть роль родителя. Родитель может смотреть результаты обучения и просматривать программу обучения.

Из особенностей приложения можно выделить:

– приложения является полностью бесплатным;

– содержит большую базу знаний по многим школьным направлениям;

– поддерживается на любых устройствах;

– имеет поддержку русского языка и переведено на множество других языков;

– удобный и простой пользовательский интерфейс, который будет понятен людям любого возраста.

Минусом приложения является его направленность только на школьные дисциплины, однако благодаря этому всё, что связано с этими дисциплинами, имеет отличную организацию.

Следующие два приложения реализуют концепцию ментальных карт. Ментальная карта или диаграмма связей – техника структуризации концепций с использованием графической записи в виде диаграммы. Популяризована британским психологом *Tony Buzan*.

Ментальная карта реализуется в виде древовидной схемы, на которой изображены слова, идеи, задачи или другие понятия, связанные ветвями, отходящими от центрального понятия или идеи.

Ментальные карты могут использоваться как:

– инструменты управления знаниями;

– при обучении или самообучении;

– для записи результатов мозгового штурма.

Приложения *Mindomo* и *Mind Meister* имеют очень широкий спектр применения. Их можно использовать для преподавания и обучения, для работы, для личных целей и т.д. Создавая ассоциативные карты, обучающиеся изучают информацию, определяют важные моменты и решают, как они связаны с уже имеющимися данными. Этот процесс отлично развивает критическое мышление.

Благодаря ментальным картам легче понять, как понятия соотносятся друг с другом и научиться находить новые связи и закономерности. Также легче

проводить собственные исследования и структурировать эссе. Карту можно трансформировать в реальный план действий и приступить к его исполнению.

Кроме обучения, ассоциативные карты можно интегрировать в рабочий процесс, чтобы провести мозговой штурм, стратегическое планирование и использовать нестандартное мышление, а также перейти на качественно новый уровень решения проблем.

Эти два приложения имеют много похожих особенностей и практически не отличаются, однако всё же есть пару моментов, которые необходимо выделить:

– *Mindomo* дороже своего конкурента, однако имеет больший рейтинг удовлетворённости пользователей;

– *Mindomo* интегрируется с большим количеством приложений и поддерживает большее количество языков;

– *Mindomo* имеет более широкий набор функций.

Результаты сравнения приложений для контроля процесса обучения представлены в таблице 1.2.

Таблица 1.2 – Характеристика приложений для контроля обучения

Параметры	*Study Smarter*	*Solo Learn*	*Khan Academy*	*Mindomo*	*Mind Meister*	Разработанное приложение
Доступность	6	6	10	6	7	10
Понятный интерфейс	8	10	10	7	7	9
Поддержка устройств	6	10	10	10	10	10
Интеграция	0	4	4	10	6	0
Простота	8	8	8	6	5	10
Функционал	6	7	6	8	7	3
Визуализация	6	7	5	10	10	8
Итоги	40	52	53	57	52	50

Из таблицы видно, что приложение *Mindomo* является очень хорошим вариантом для контроля процессов обучения. Разработанное приложение предоставляет средний, по меркам похожих приложений, функционал для контроля процессов обучения.

1.3 Требования к программному комплексу

После исследования рынка приложений и доступного функционала была утверждена следующая тема дипломной работы: программный комплекс для персонального планирования и контроля процесса обучения. Был составлен график выполнения дипломной работы, состоящий из следующих шагов:

– изучение требований, предъявляемых к безопасной и комфортной работе инженеров-программистов. Знакомство со спецификой этапов жизненного цикла программного продукта;

– анализ методов планирования проектов и контроля результатов обучения;

– изучение методов тестирования и классификации тестов;

– сравнительный анализ существующих программных решений для персонального планирования в том числе для обучения;

– сравнительный анализ существующих программных решений для контроля результатов обучения;

– проектирование структуры программного комплекса, базы данных для хранения информации, формирование пользовательских правил (ролевых политик) для доступа к ресурсам и функциям системы, графического интерфейса;

– разработка системы авторизации и аутентификации пользователей различных групп;

– формирование информационной базы для выполнения дипломной работы и тестирования предполагаемых к созданию программных продуктов;

– создание программного продукта в соответствии с темой дипломной работы.

Приложение должно поддерживаться всеми современными видами устройств: настольными компьютерами, ноутбуками, мобильными телефонами.

Также приложение должно предоставлять функционал аутентификации и авторизации пользователей для того, что бы каждый пользователь мог получать доступ к своим проектам и ограничить доступ другим пользователям.

Акцент приложения следует сделать на визуальной составляющей и удобстве использования, так как это самая главная характеристика такого рода приложений. Для этого приложение должно иметь только самый необходимый функционал, который должен располагаться в различных местах программы.

Организация проектов должна иметь древовидную структуру, так как такого рода организация избегает ограничения линейного представления процессов работы над проектами обучения:

– невозможность динамичной декомпозиции задач проекта;

– невозможность сделать точную иерархию задач;

– невозможность наглядного представления большого объёма данных.

Приложение должно позволять пользователю добавлять и удобно просматривать вложения различного рода, например ссылки или таблицы. Это позволит включать в удобную древовидную структуру проектов дополнительную информацию, представленную различными типами данных.

Кроме того, приложение должно иметь функционал контроля обучения и общения пользователей по определённым темам.

Приложение должно иметь понятный и удобный пользовательский интерфейс, который не будет отвлекать пользователя и одинаково удобно отображаться на различного рода устройствах.

2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА

2.1 Технологические решения, применённые для создания программного комплекса

В качестве языка для создания серверного приложения выступает *C#*. Он относится к семье языков с *C*-подобным синтаксисом, из них его синтаксис наиболее близок к *C++* и *Java*. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов, делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, как механизм управления данными, и многое другое, что должен иметь любой современный язык программирования.

Переняв многое от своих предшественников – языков *C++*, *Delphi*, Модула, *Smalltalk* и, в особенности, *Java* – *C#*, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, *C#* в отличие от *C++* не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов).

Язык *C#* работает в рамках платформы *.Net Framework*. *.NET Framework* – программная платформа, выпущенная компанией *Microsoft* в 2002 году. Основой платформы является общеязыковая среда исполнения *Common Language Runtime* (*CLR*), которая подходит для различных языков программирования. Функциональные возможности *CLR* доступны в любых языках программирования, использующих эту среду. В настоящее время *.NET Framework* развивается в виде *.NET*.

Фреймворком для разработки серверной части приложения является *ASP.NET*. *ASP.NET* (*Active Server Pages* для *.NET*) – платформа разработки веб-приложений, в состав которой входят: веб-сервисы, программная инфраструктура, модель программирования, от компании Майкрософт.

В частности используется *ASP.NET Web API*. *Web API* представляет способ построения приложения *ASP.NET*, который специально предназначен для работы в стиле *REST* (*Representation State Transfer*).

REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Другими словами, *REST* – это набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать. *REST* представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры.

В качестве библиотек для серверного приложения используются следующие пакеты:

– *Entity Framework Core*;

– *MongoDB Driver*;
– *Automapper*;
– *Authentication JWT Bearer*;
– *SignalR;*
– *xUnit*.

Entity Framework Core (*EF Core*) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании *Microsoft* для доступа к данным. *EF Core* является *ORM*-инструментом (*object-relational mapping* – отображения данных на реальные объекты). То есть *EF Core* позволяет работать с базами данных без написания *SQL* запросов, представляя собой более высокий уровень абстракции: *EF Core* позволяет абстрагироваться от самой базы данных и её таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне программист оперирует таблицами, индексами, первичными и внешними ключами, то на концептуальном уровне, который предлагает *Entity Framework*, происходит работа с объектами.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, через *EF Core*, можно работать с любой СУБД, если для неё имеется нужный провайдер.

По умолчанию, на данный момент, *Microsoft* предоставляет ряд встроенных провайдеров: для работы с *MS SQL Server*, *SQLite*, *PostgreSQL*. Также имеются провайдеры от сторонних поставщиков, например, для *MySQL*.

В данном случае используется СУБД *MS SQL Server*.

SQL Server является одной из наиболее популярных систем управления базами данных в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

SQL Server долгое время был исключительно системой управления базами данных для *Windows*, однако начиная с версии 16 эта система доступна и на *Linux*.

SQL Server характеризуется такими особенностями как:

– производительность. *SQL Server* работает очень быстро;

– надёжность и безопасность. *SQL Server* предоставляет шифрование данных;

– простота. С данной СУБД относительно легко работать и вести её администрирование.

Для организации баз данных *MS SQL Server* использует реляционную модель. Эта модель баз данных была разработана в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Но в некоторых ситуациях, реляционные базы данных не самое эффективное решения для хранения данных и их управления. В таких случаях используют нереляционные базы данных, одной из которых является *MongoDB*.

MongoDB – документоориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров *NoSQL*-систем, использует *JSON*-подобные документы

и схему базы данных. Написана на языке *C++*. Чаще всего ременяется в веб-разработке.

СУБД *MongoDB* масштабируется горизонтально, используя технику сегментирования объектов баз данных – распределение их частей по различным узлам кластера. Администратор выбирает ключ сегментирования, который определяет, по какому критерию данные будут разнесены по узлам (в зависимости от значений хэша ключа сегментирования). Благодаря тому, что каждый узел кластера может принимать запросы, обеспечивается балансировка нагрузки. Именно это позволяет нереляционным базам данных обеспечивать эффективную работу в высоконагруженных приложениях.

В качестве *ORM* для этой СУБД на платформе *.Net* выступает *MongoDB Driver*. Эта библиотека, как и любая другая, может быть установлена при помощи одной команды пакетного менеджера *.Net*. Она также абстрагирует пользователя от прямого манипулирования объектами базы данных и предоставляет удобный программный интерфейс для этого.

Практически всегда сущности в базе данных не совпадают с тем, что необходимо возвращать клиенту приложения, запросившему определённые данные. Из-за этого присутствует необходимость в преобразовании сущностей базы данных в сущности, необходимые клиенту приложения.

Такого рода преобразование удобно делать при помощи библиотеки *AutoMapper*. Она позволяет проецировать одну модель на другую, что позволяет сократить объёмы кода и упростить программу. Возможные применения данной библиотеки:

– преобразование иерархической модели объектов в плоскую;

– получение проекций объектов;

– получение объекта сущности базы данных из объекта передачи данных;

– условное преобразование объектов;

– преобразование объектов базы данных в сериализуемые объекты для передачи через какую-либо коммуникационную среду.

В нереляционной базе данных хранится аутентификационная информация пользователей. Для удобства работы с авторизацией и аутентификацией посредством токенов была установлена библиотека *Authentication JWT Bearer.* Благодаря ей легко настроить конфигурацию аутентификации на сервере и использовать встроенный механизм аутентификации пользователей.

Сама аутентификация происходит при помощи *JWT* токена. *JSON Web Token* (*JWT*) – это открытый стандарт для создания токенов доступа, основанный на формате *JSON*. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

Токен *JWT* состоит из трех частей: заголовка, полезной нагрузки и подписи или данных шифрования. Первые два элемента – это JSON объекты определённой структуры. Третий элемент вычисляется на основании первых и зависит от выбранного алгоритма. Токены могут быть перекодированы в компактное представление (*JWS/JWE Compact Serialization*): к заголовку и

полезной нагрузке применяется алгоритм кодирования *Base64-URL*, после чего добавляется подпись и все три элемента разделяются точками.

Заключительной библиотекой в серверном приложении является *SignalR*. *SignalR* представляет библиотеку от компании *Microsoft*, которая предназначена для создания приложений, работающих в режиме реального времени. Чаще всего её используют вместе с *ASP.NET Core*, однако она может быть использована в любого рода клиент-серверных приложений. *SignalR* использует двунаправленную связь для обмена сообщениями между клиентом и сервером, благодаря чему сервер может отправлять в режиме реального времени всем клиентам некоторые данные.

Для обмена сообщениями между клиентом и сервером *SignalR* использует ряд механизмов: *WebSockets*, *Server-Side Events*, *Long Polling*. Исходя из возможностей клиента и сервера инфраструктура *SignalR* выбирает наилучший механизм для взаимодействия. В частности, наиболее оптимальным является *WebSockets*, соответственно если и клиент, и сервер позволяют использовать этот механизм, то взаимодействие идёт через *WebSockets*.

В качестве клиента *SignalR* могут выступать:

– приложение на *JavaScript*, запущенное на *Node.js*;

– приложение на *JavaScript*, которое работает в рамках браузеров *Google Chrome* (в том числе на *Android*), *Microsoft Edge*, *Mozilla Firefox*, *Opera*, *Safari* (в том числе на *iOS*), *Internet Explorer*;

– приложение на *.NET*. Это может быть десктопное приложение *WPF*, *Windows Forms*, мобильное приложение *Xamarin*;

– приложение на языке *Java*.

Для тестирования серверного приложения используется *xUnit*. *xUnit.net* – это фреймворк тестирования для платформы *.NET*. Наиболее популярный фреймворк для работы именно с *.NET Core* и *ASP.NET Core*, в более поздних версиях просто *.Net* и *ASP.NET*.

Юнит-тесты позволяют быстро и автоматически протестировать отдельные компоненты приложения независимо от остальной его части. Не всегда юнит-тесты могут покрыть весь код приложения, но тем не менее они позволяют существенно уменьшить количество ошибок уже на этапе разработки.

В качестве клиентской части приложения выступает браузерное веб-приложение, написанное на фреймворке *Angular*. *Angular* – это фреймворк от компании *Google* для создания клиентских приложений. Прежде всего он нацелен на разработку *SPA*-приложений, то есть одностраничных приложений. В этом плане *Angular* является наследником другого фреймворка *AngularJS*. В то же время *Angular* это не новая версия *AngularJS*, а принципиально новый фреймворк.

Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Одной из ключевых особенностей *Angular* является то, что он использует в качестве языка программирования *TypeScript*, что позволяет использовать

строгую типизацию данных, интерфейсы и многие другие возможности объектно-ориентированных языков программирования.

Но *Angular* приложение не ограничено языком *TypeScript*. При желании разработчик может писать приложения на *Angular* с помощью таких языков как *Dart* или *JavaScript*. Однако *TypeScript* все же является основным.

Angular использует менеджер пакетов *NodeJS*. Данный менеджер пакетов позволяет устанавливать любые библиотеки при помощи одной команды. Таким образом можно установить клиентскую часть *SignalR* для взаимодействия с серверным приложением в реальном времени.

Все вышеперечисленные библиотеки и фреймворки являются современными и самыми актуальными для разработки клиент-серверных приложений. Кроме обширного функционала они предоставляют высокую степень абстракции и безопасности при написании кода.

2.2 Логическая и физическая модели баз данных

В разработанном программном комплексе было решено использовать две СУБД: нереляционную *MongoDB* и реляционную *MS SQL Server*. Данное решение было принято исходя из возможности улучшения масштабирования приложения. Благодаря тому, что аутентификационные данные пользователей и остальные данные приложения хранятся в разных базах данных и обрабатываются разными сервисами, существует возможность рационального распределения ресурсов приложения.

Аутентификационная база данных является нереляционной и реализована при помощи технологии *MongoDB*. В этой базе данных вся информация хранится в виде коллекций документов. В данном случае документом является сущность пользователя с его полями. Представление сущности пользователя изображено на рисунке 2.1.

Рисунок 2.1 – Сущность пользователя

Коллекция пользователей хранит документы о пользователях, которые содержат следующие поля:

– *_id* – строковый идентификатор пользователя;

– *Email* – почта пользователя;
– *PasswordHash* – хеш пароля пользователя;
– *Roles* – массив чисел, представляющих роли пользователя.

Ресурсная база данных приложения (база данных для хранения основной информации приложения) имеет реляционную структуру. Она реализована с помощью *MS SQL Server*. Для указания связи между пользователем и сущностью в ресурсной базе данных используется поле *accountId*. Это строковое поле и оно совпадает с идентификатором пользователя в аутентификационной базе данных пользователей.

Например, это поле используется для указания связи между пользователями и их профилями. Каждый пользователь имеет свой профиль, который возвращается ему при обращении к базе данных, используя данный пользовательский идентификатор.

Также данное идентификационное поле используется для указания связей между пользователями и их областями проектов и между пользователями и их вложениями, так как у каждого пользователя свои наборы этих сущностей. Остальные сущности базы данных не зависят от идентификатора пользователя, однако имеют связи с зависящими сущностями. Это хорошо с точки зрения хранения информации, однако немного замедляет получение данных из базы.

Сущности базы данных и их связи изображены на рисунке 2.2.

Рисунок 2.2 – Связи сущностей базы данных

В результате проектирования базы данных были получены следующие сущности:
– «Профиль пользователя»;
– «Область проектов»;
– «Проект»;
– «Стадия»;

– «Вложение в стадии»;
– «Вложение»;
– «Тип вложения»;
– «Контакт»;
– «Карточка проекта».

В сущности «Профиль пользователя» определены следующие атрибуты: «Идентификатор пользователя», «Имя профиля», «Описание профиля», «Иконка профиля» и «Изображение профиля». В качестве уникального идентификатора служит «Идентификатор профиля». Более подробное описание каждого атрибута приведено в таблице 2.1.

Таблица 2.1 – Описание атрибутов сущности «Профиль пользователя»

Атрибуты	Описание домена	Тип данных
Идентификатор профиля	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный
Идентификатор пользователя	Содержит идентификатор пользователя, связанного с этим профилем	Строка
Имя профиля	Содержит имя профиля пользователя	Строка
Описание профиля	Содержит описание профиля пользователя	Строка
Иконка профиля	Содержит небольшое изображение, представляющее иконку профиля пользователя	Массив байт
Изображение профиля	Содержит изображение профиля пользователя	Массив байт

В сущности «Область проектов» определены следующие атрибуты: «Название области проектов», «Идентификатор пользователя» и «Иконка области проектов». В качестве уникального идентификатора служит «Идентификатор области проектов». Более подробное описание каждого атрибута приведено в таблице 2.2.

Таблица 2.2 – Описание атрибутов сущности «Область проектов»

Атрибуты	Описание домена	Тип данных
Идентификатор области проектов	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный

Продолжение таблицы 2.2

Атрибуты	Описание домена	Тип данных
Название области проектов	Содержит наименование области проектов	Строка
Идентификатор пользователя	Содержит идентификатор пользователя, связанного с этой областью проектов	Строка
Иконка области проектов	Содержит изображение области проектов	Массив байт

В сущности «Проект» определены следующие атрибуты: «Идентификатор области проектов», «Название проекта» и «Описание проекта». В качестве уникального идентификатора служит «Идентификатор проекта». Более подробное описание каждого атрибута приведено в таблице 2.3.

Таблица 2.3 – Описание атрибутов сущности «Проект»

Атрибуты	Описание домена	Тип данных
Идентификатор проекта	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный
Идентификатор области проектов	Содержит идентификатор области проектов. Является внешним ключом	Целочисленный
Название проекта	Содержит название проекта	Строка
Описание проекта	Содержит описание проекта	Строка

В сущности «Стадия» определены следующие атрибуты: «Идентификатор проекта», «Название стадии», «Описание стадии», «Дата создания», «Последний срок сдачи» и «Прогресс выполнения». В качестве уникального идентификатора служит «Идентификатор стадии». Более подробное описание каждого атрибута приведено в таблице 2.4.

Таблица 2.4 – Описание атрибутов сущности «Стадия»

Атрибуты	Описание домена	Тип данных
Идентификатор стадии	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный

Продолжение таблицы 2.4

Атрибуты	Описание домена	Тип данных
Идентификатор проекта	Содержит идентификатор области проектов. Является внешним ключом	Целочисленный
Название стадии	Содержит название стадии	Строка
Описание стадии	Содержит описание стадии	Строка
Дата создания	Содержит дату создания стадии	Дата
Последний срок сдачи	Содержит дату ожидаемого завершения проекта	Дата
Прогресс выполнения	Содержит число от нуля до ста, обозначающее прогресс выполнения проекта	Целочисленный

В сущности «Вложение» определены следующие атрибуты: «Идентификатор пользователя», «Название вложения», «Идентификатор типа вложения» и «Данные вложения». В качестве уникального идентификатора служит «Идентификатор вложения». Более подробное описание каждого атрибута приведено в таблице 2.5.

Таблица 2.5 – Описание атрибутов сущности «Вложение»

Атрибуты	Описание домена	Тип данных
Идентификатор вложения	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный
Идентификатор пользователя	Содержит идентификатор пользователя, связанного с этим вложением	Целочисленный
Название вложения	Содержит название вложения	Строка
Идентификатор типа вложения	Содержит идентификатор типа вложения. Является внешним ключом	Целочисленный
Данные	Содержит данные пользовательского вложения	Строка

В сущности «Вложение в стадии» определены следующие атрибуты: «Идентификатор стадии», «Идентификатор вложения». В качестве уникального идентификатора служит «Идентификатор вложения в стадии». Более подробное описание каждого атрибута приведено в таблице 2.6.

Таблица 2.6 – Описание атрибутов сущности «Вложение в стадии»

Атрибуты	Описание домена	Тип данных
Идентификатор вложения в стадии	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный
Идентификатор стадии	Содержит идентификатор стадии. Является внешним ключом	Целочисленный
Идентификатор вложения	Содержит идентификатор стадии. Является внешним ключом	Целочисленный

В сущности «Тип вложения» определён аттрибут «Название типа вложения». В качестве уникального идентификатора служит «Идентификатор типа вложения». Более подробное описание каждого атрибута приведено в таблице 2.7.

Таблица 2.7 – Описание атрибутов сущности «Тип вложения»

Атрибуты	Описание домена	Тип данных
Идентификатор типа вложения	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный
Название типа вложения	Содержит наименование типа вложения	Строка

В сущности «Контакт» определены аттрибуты «Идентификатор первого профиля», «Идентификатор второго профиля» и «Утверждён». В качестве уникального идентификатора служит «Идентификатор контакта». Более подробное описание каждого атрибута приведено в таблице 2.8.

Таблица 2.8 – Описание атрибутов сущности «Контакт»

Атрибуты	Описание домена	Тип данных
Идентификатор контакта	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный

Продолжение таблицы 2.8

Атрибуты	Описание домена	Тип данных
Идентификатор первого профиля	Содержит идентификатор пользователя, который подал заявку на добавление в контакты. Является внешним ключом	Целочисленный
Идентификатор второго профиля	Содержит идентификатор пользователя, который должен утвердить заявку в контакты. Является внешним ключом	Целочисленный
Утверждён	Содержит информацию о том, утверждён ли пользовательский контакт	Двоичный

В сущности «Карточка проекта» определены аттрибуты «Вопрос», «Ответ» и «Идентификатор проекта». В качестве уникального идентификатора служит «Идентификатор карточки проекта». Более подробное описание каждого атрибута приведено в таблице 2.9.

Таблица 2.9 – Описание атрибутов сущности «Карточка проекта»

Атрибуты	Описание домена	Тип данных
Идентификатор карточки проекта	Уникальный инкрементируемый идентификатор. Является первичным ключом	Целочисленный
Вопрос	Содержит вопрос данной карточки	Строка
Ответ	Содержит ответ данной карточки	Строка
Идентификатор проекта	Содержит идентификатор проекта, который содержит данную карточку. Является внешним ключом	Целочисленный

Все таблицы базы данных были приведены в третью нормальную форму для более эффективного хранения и изменения пользовательской информации.

2.3 Программная архитектура приложения и использованные шаблоны проектирования

Разработанный программный комплекс имеет микросервисную архитектуру. Микросервисная архитектура – распространенный подход к разработке программного обеспечения, когда приложение разбивается на небольшие автономные компоненты(микросервисы) с четко определенными интерфейсами. Эта архитектура часто ставится в противопоставление с монолитной архитектурой приложений.

Монолиты – это приложения, построенные как единое целое, где вся логика по обработке запросов помещается внутрь одного процесса. Разумеется, монолиты могут иметь модульную структуру – содержать отдельные классы, функции, области имён. Но связи между этими модулями настолько сильны, что изменение каждого из них неизбежно отражается на работе приложения в целом. Ниже приведены преимущества микросервисной архитектуры относительно монолитной.

Первым неоспоримым преимуществом такой архитектуры является простота развертывания приложения. Можно развертывать только изменяющиеся микросервисы, независимо от остальной системы, что позволяет производить обновления чаще и быстрее.

Следующим преимуществом является оптимальность масштабирования. Можно расширять только те сервисы, которые в этом нуждаются, то есть сервисы с наименьшей производительностью, оставляя работать остальные части системы на менее мощном оборудовании.

Далее необходимо отметить устойчивость к сбоям. Отказ одного сервиса не приводит к остановке системы в целом. Когда же ошибка исправлена, необходимое изменение можно развернуть только для соответствующего сервиса – вместо повторного развертывания всего приложения. Правда, для этого еще на этапе проектирования микросервисов потребуется тщательно продумать связи между ними для достижения максимальной независимости друг от друга, а также заложить возможность корректного оповещения пользователя о временной недоступности определенного сервиса без ущерба для всей системы.

Также микросервисная архитектура позволяет применять более широкий спектр технологий для приложения. Можно подбирать различные наборы технологий, оптимальные для решения задач, стоящих перед отдельными сервисами.

Для микросервисной архитектуры нет необходимости в большой команде разработчиков. При разработке микросервисов команды принято закреплять за конкретными бизнес-задачами (и сервисами, соответственно). Такие команды, как правило, показывают большую эффективность, а управлять ими легче.

Микросервисы легко заменять при необходимости, в отличие от модулей монолитов.

И заключительным преимуществом микросервисов является независимость моделей данных. Каждый микросервис, как правило, использует

собственное хранилище данных – поэтому изменение модели данных в одном сервисе не влияет на работу остальных.

Разработанное приложение имеет клиентский браузерный интерфейс, написанный на веб фреймворке *Angular*, который взаимодействует с тремя сервисами.

Первый сервис предназначен для авторизации и аутентификации пользователей. Пользовательские данные хранятся в нереляционной базе данных *MongoDB*.

Второй сервис является ресурсным сервисом. Этот сервис отвечает за хранение данных о проектах пользователей, их обработку и поставку клиенту. Этот сервис хранит данные в реляционной базе данных *MSSQL Server*.

Третий же сервис отвечает за взаимодействие пользователей в реальном времени и реализует групповые пользовательские чаты.

Графическое представление архитектуры приложения изображено на рисунке 2.3.

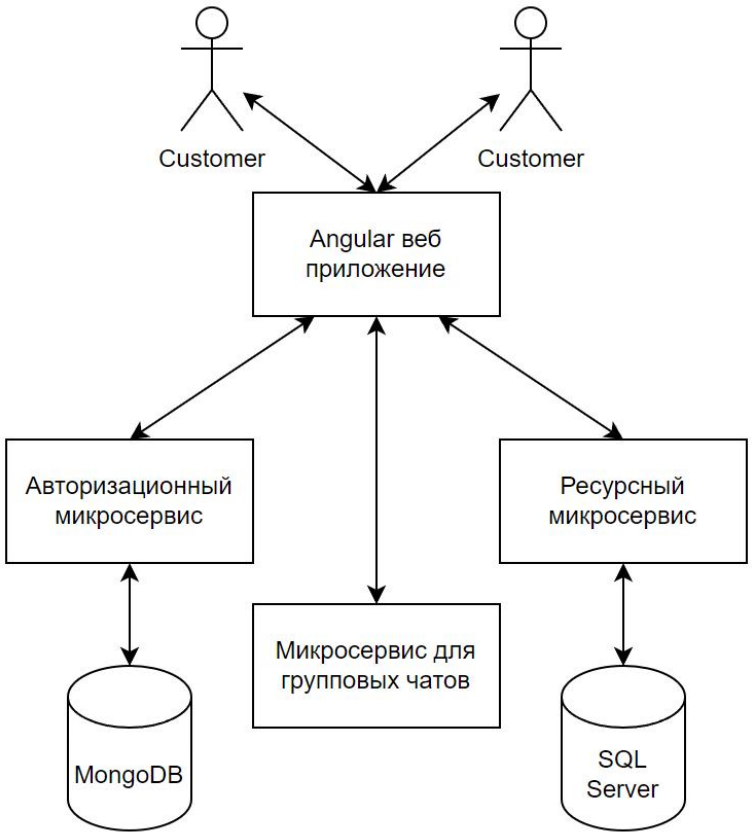

Рисунок 2.3 – Архитектура приложения

Авторизационный микросервис не взаимодействует с ресурсным микросервисом напрямую. Для того, что бы ресурсный микросервис знал какой пользователь к нему обращается, в приложении происходит постоянный обмен авторизационным токеном, хранящим основную пользовательскую информацию, полученную авторизационным сервисом из его базы данных. Всю остальную информацию ресурсный микросервис берёт из своей базы данных.

3 СТРУКТУРА ПРОГРАММНОГО КОМПЛЕКСА

3.1 Функциональные возможности и последовательность обработки информации программным комплексом

Взаимодействие приложения с пользователем начинается с браузера. Пользователь переходит по адресу веб-приложения и попадает на страницу аутентификации и авторизации.

Система аутентификации пользователей представляет собой отдельный микросервис, который хранит данные о пользователях и предоставляет токены доступа к приложению.

Токены имеют тип *JWT*. Они создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

В качестве алгоритма шифрования используется *HmacSha256*. *HMACSHA256* – это тип хэш-алгоритма с ключом, созданный на основе хэш-функции *SHA*-256 и используемый в качестве кода аутентификации сообщений на основе хэша (*HMAC*).

Благодаря такой технологии авторизации и аутентификации пользователей микросервисы становятся менее зависимыми друг от друга и реализуется подход проектирования сервера без сохранения состояния пользователя.

После аутентификации происходит перенаправление пользователя на основную страницу приложения. На данной странице присутствует панель навигации, которая позволяет переходить к различным частям приложения. Одной из таких частей является чат.

Когда пользователь переходит на страницу чата происходит установка связи с микросервисом, отвечающим за межпользовательское взаимодействие. Данный микросервис позволяет пользователям создавать динамические групповые чаты, где они могут общаться по интересующим их темам в реальном времени.

Основным микросервисом серверной части приложения является ресурсный сервис. Он предоставляет непосредственно основной функционал приложения:

– получение данных из базы данных;

– обработка и поставка данных;

– взаимодействие с пользовательским веб клиентом при помощи *REST*;

– обработка ошибок.

Все реализованные микросервисы написаны при помощи технологии *ASP.NET* 6. Это версия фреймворка работает с платформой *.NET* шестой версии и языком *C#* десятой версии. Эти технологии являются одними из самых эффективных и современных, что позволяет быстро разрабатывать высоконагруженные веб приложения.

Для доступа к базе данных *MongoDB* авторизационный сервис использует *MongoDB* драйвер для языка *C#*. Этот драйвер является оболочкой над стандартными командами *MongoDB*. При помощи него можно быстро и

безошибочно манипулировать коллекциями и документами в нереляционной базе данных. Основным классом для взаимодействия с этой базой данных является *BsonDocument*. По сути этот класс является представлением *JSON* документа в двоичном формате, с которым взаимодействует *MongoDB*.

Над драйвером был реализован шаблон репозиторий для разделения ответственности и более удобного тестирования. Этот репозиторий манипулирует моделью приложения, представляющей собой пользователя.

Кроме модели базы данных сервис также содержит модели, при помощи которых он взаимодействует с клиентским приложением. Это сделано для скрытия от клиента информации о хранении сущности пользователя в базе данных. Такими моделями являются модель токена приложения и модель авторизационной формы пользователя. Модель токена приложения авторизационный сервер возвращает при входе пользователя в систему. Модель формы авторизации сервис получает, когда к нему обращается клиентское приложение с запросом на авторизацию или регистрацию пользователя.

Ресурсный сервис взаимодействует с реляционной базой данных *SQL Server* при помощи *ORM* технологии *Entity Framework*. Ресурсный сервис содержит все модели сущностей базы данных и контекст базы данных для обращения к ней. Над контекстом реализованы шаблоны репозиторий и *UnitOfWork*. Это сделано для удобного тестирования сервиса и разделения ответственности его классов.

Данный сервис манипулирует основными сущностями приложения, такими как: проекты, профили и так далее. Функционал этого сервиса включает:

– управление пользовательскими проектами, вложениями и карточками;

– управление профилем пользователя;

– управление пользовательскими контактами и статистикой.

Также ресурсный сервер содержит информацию о миграциях базы данных приложения. Это необходимо для контроля изменения сущностей базы данных и их отношений.

При помощи инструмента *AutoMapper* приложение понятно преобразует сущности базы данных в сущности, которые сервис принимает от клиента и посылает ему. Для работы этого инструмента были разработаны и внедрены профили преобразования сущностей.

Вся бизнес логика микросервисов выделена в специальные классы, называемые службами. Эти службы обращаются к репозиториям, получая необходимые данные, обрабатывает их и отдают контроллерам для возвращения клиенту.

Кроме того ресурсный микросервис содержит классы-инструменты, которые содержат бизнес логику, не связанную с непосредственным получением и обработкой данных. Эти инструменты нужны для выделения сложной бизнес логики в отдельные классы и их применения в службах.

Микросервисы взаимодействуют с клиентским приложением при помощи протокола *HTTP* и архитектуры *REST*. Для реализации непосредственного взаимодействия с клиентом используются классы контроллеры. Они принимают запросы пользователей, обращаются к службам приложения и

возвращают необходимые данные обратно клиенту. Если служба завершила выполнение метода с ошибкой, контроллер решает, что необходимо вернуть клиенту. Чаще всего это краткая информация об ошибке или конкретное указание почему запрос завершился с ошибкой.

Микросервис групповых чатов использует *SignalR* для взаимодействия пользователей с сервером в реальном времени. В большинстве случаев это значит, что клиент приложения будет использовать протокол *WebSockets*, однако, если данный протокол не доступен, будут использоваться менее эффективные подходы.

Клиентская часть представляет собой браузерное приложение, написанное на *Angular*. Этот фреймворк имеет чёткую архитектуру и использует типизируемый язык, что позволяет делать меньше ошибок в ходе разработки.

Структурными единицами веб приложения являются:

– файлы конфигурации;

– компоненты;

– сервисы;

– модели.

В файлах конфигурации содержится вся информация о клиентском приложении: адреса микросервисов, зависимости, версия и т.д.

Компоненты приложения – это классы, которые содержат ссылки на код гипертекстовой разметки, стили для этой разметки и реализуют логику взаимодействия с пользователем. Они являются частями веб приложения и часто их взаимодействие между собой представляет графовую структуру.

Эти компоненты обращаются к сервисам или службам. Эти службы содержат бизнес логику веб приложения: код отправки запросов к серверу, код валидации данных, код обработки данных и так далее.

Эти службы манипулируют моделями. Модели – это классы, хранящие информацию о сущностях приложения. Эти модели по своим аттрибутам аналогичны моделям, которые сервер отправляет клиенту.

Разработанное приложение содержит две модели для авторизации пользователя: модель авторизационной формы и модель токена. Также приложение содержит семь моделей для реализации основного функционала приложения: вложение, область проектов, профиль, проект, стадия, карточка и статистика пользователя.

И для реализации чата также были разработаны две модели: группа и сообщение. Эти модели используются для межпользовательского группового взаимодействия.

На каждую из этих моделей присутствует служба, которая получает и отправляет эти модели при взаимодействии с сервером.

Службы используются в компонентах приложения. Можно выделить несколько компонентов верхнего уровня приложения. Корневым компонентом является сам компонент приложения *app*. В него входят компоненты, представляющие основные модули веб приложения.

Модули веб приложения взаимодействуют друг с другом при помощи специальных привязок данных и событий. Приложение является

одностраничным приложением. Весь контент приложения формируется при помощи *JavaScript*.

3.2 Структура программного комплекса

Структуру всего программного комплекса можно разделить на четыре составляющие:

– структура авторизационного сервиса;
– структура ресурсного сервиса;
– структура сервиса групповых чатов;
– структура клиентского браузерного приложения.

Структура авторизационного сервиса включает следующие части:

– точка входа в приложение;
– контроллеры;
– данные;
– модели;
– службы;
– модели для взаимодействия с клиентом.

Точкой входа в приложение является класс *Program*. Этот класс конструирует приложение при запуске, используя класс, выполняющий конфигурацию всего приложения – *Startup*. В классе *Startup* настраиваются все зависимости микросервиса, в том числе происходит:

– добавление возможности использования контроллеров;
– настройка политики кросс-доменных запросов;
– добавление сервисов в механизм внедрения зависимостей;
– настройка контекста данных микросервиса;
– настройка механизма аутентификации приложения.

Контроллеры сервиса находятся в папке *Controllers*. В авторизационном микросервисе находится только один контроллер, отвечающий за регистрацию, аутентификацию и авторизацию пользователей – *AuthContoller*. Данный контроллер содержит следующие конечные точки:

– *POST api/auth/signIn* – конечная точка, принимающая данные авторизационной формы пользователя и генерирующая токен доступа к приложению в случае успеха авторизации;

– *POST api/auth/signUp* – конечная точка, принимающая данные авторизационной формы пользователя, производящая регистрацию пользователя и, в случае успеха, генерирующая токен доступа пользователя к приложению;

Данные авторизационного микросервиса находятся в папке *Data*. Данная папка разделена на две части: интерфейсы и реализации шаблона репозиторий. В части интерфейсов находится один интерфейс, определяющий протокол взаимодействия программиста с базой данных – *IAccountRepository*. Данный интерфейс определяет три метода:

– *GetByIdAsync* – метод, получающий модель аккаунта пользователя по его идентификатору;

– *GetByEmailAsync* – метод, получающий модель аккаунта пользователя по его электронной почте;

– *CreateAsync* – метод, создающий нового пользователя.

Реализация этого интерфейса – *MongoAccountRepository*. Данная реализация взаимодействует с базой данных *MongoDB* посредством *MongoDB Driver* и инкапсулирует данную конкретную логику взаимодействия.

Модели приложения представлены одним классом – *Account*. Данный класс отображает сущность из базы данных в коде и содержит те же поля, что и записи пользователей в базе данных.

Авторизационный микросервис содержит две службы:

– служба авторизации;

– служба хеширования паролей.

Служба авторизации связывает между собой репозиторий и контроллер. Кроме двух операций контроллера, данная служба реализует функционал генерации токена доступа.

Служба хеширования паролей содержит несколько операций: хеширование пароля, проверку валидности хеша и верификацию пароля пользователя.

Заключительной частью авторизационного сервиса являются модели для взаимодействия с клиентом. Эти модели располагаются в папке *UIModels*. Данная папка содержит два класса:

– *Token* – токен доступа;

– *AuthData* – данные авторизационной формы пользователя.

Ресурсный сервис имеет похожую структуру. Дополнительными частями данного микросервиса являются:

– исключения;

– профили для трансформации данных;

– миграции базы данных;

– инструменты.

Ресурсный сервис содержит семь контроллеров. Контроллеры также располагаются в папке *Controllers*.

AttachmentsContoller отвечает за управление пользовательскими вложениями и содержит следующие конечные точки:

– *GET api/attachments/stage/{stageId}* – возвращает все вложения для определённой стадии проекта;

– *GET api/attachments* – возвращает все вложения пользователя;

– *POST api/attachments/stage/{stageId}* – добавляет вложение в определённую стадию;

– *POST api/attachments* – создаёт новое вложение для пользователя;

– *PUT api/attachments/{attachId}* – изменяет вложение пользователя;

– *DELETE api/attachments/stage/{stageId}* – удаляет вложение из стадии;

– *DELETE api/attachments/{id}* – удаляет пользовательское вложение.

CardsController отвечает за управление проектными карточками и содержит следующие конечные точки:

– *GET api/cards/{projectId}* – возвращает все карточки для определённого проекта;

– *POST api/cards/{projectId}* – создаёт новую карточку для проекта;

– *PUT api/cards/{cardId}* – изменяет проектную карточку;

– *DELETE api/cards/{id}* – удаляет проектную карточку.

FieldsController отвечает за управление областями проектов и содержит следующие конечные точки:

– *GET api/fields* – возвращает все области проектов пользователя;

– *POST api/fields* – создаёт новую область проектов;

– *PUT api/fields* – изменяет область проектов;

– *DELETE api/fields/{id}* – удаляет область проектов.

ProfilesController отвечает за управление профилем пользователя и содержит следующие конечные точки:

– *GET api/profiles* – возвращает профиль пользователя;

– *PUT api/profiles* – изменяет профиль пользователя;

– *GET api/profiles/contacts/accepted* – возвращает утверждённые контакты пользователя;

– *GET api/profiles/contacts/requested* – возвращает запрошенные контакты пользователя;

– *GET api/profiles/contacts/search* – выполняет поиск пользователей;

– *GET api/profiles/contacts/profile/{id}* – возвращает профиль другого пользователя;

– *DELETE api/profiles/contacts/{id}* – удаляет другого пользователя из списка контактов;

– *DELETE api/profiles/contacts/requests/{id}* – отменяет приглашение в список контактов определённого пользователя;

– *POST api/profiles/contacts/requests/accept* – принимает приглашение в список контактов определённого пользователя;

– *POST api/profiles/contacts/requests/send* – отправляет приглашение в список контактов определённому пользователю;

ProjectsController отвечает за управление проектами пользователя и содержит следующие конечные точки:

– *GET api/projects/{fieldId}* – возвращает все проекты пользователя для определённой области проектов;

– *GET api/projects/single/{projectId}* – возвращает определённый проект пользователя;

– *POST api/projects/{fieldId}* – создаёт новый проект для определённой области проектов;

– *PUT api/projects/{projectId}*– изменяет проект;

– *DELETE api/projects/{id}* – удаляет проект.

StagesController отвечает за управление стадиями проекта пользователя и содержит следующие конечные точки:

– *GET api/stages/{projectId}* – возвращает стадии проекта пользователя для определённого проекта;

– *POST api/stages/{projectId}* – создаёт новую стадию для определённого проекта пользователя;

– *PUT api/stages/{stageId}* – изменяет стадию проекта;

– *DELETE api/stages/{id}* – удаляет стадию проекта.

StatisticController отвечает за получение статистики пользователя и содержит следующие конечные точки:

– *GET api/statistic* – возвращает статистику профиля пользователя.

Данные приложения содержатся в папке *Data*. Она также разделена на две части: интерфейсы и реализации.

В состав интерфейсов входят следующие классы:

– *IAccountAttachmentsRepository* – интерфейс репозитория для управления вложениями;

– *ICardsRepository* – интерфейс репозитория для управления проектными карточками;

– *IFieldsRepository* – интерфейс репозитория для управления областями проектов;

– *IProfilesRepository* – интерфейс репозитория для управления пользовательскими профилями;

– *IProjectsRepository* – интерфейс репозитория для управления проектами;

– *IStagesAttachmentsRepository* – интерфейс репозитория для управления вложениями в стадиях проектов;

– *IStagesRepository* – интерфейс репозитория для управления стадиями проекта;

– *IUnitOfWork* – интерфейс, реализующий шаблон *UnitOfWork*, объединяющий все репозитории в один класс, что способствует удобному использованию репозиториев в сервисах приложения и тестировании;

Для каждого интерфейса репозитория существует реализация, использующая контекст базы данных *MS SQL Server* для определения всех операций. Данный контекст базы данных является классом, предоставляющимся технологией *EntityFramework*. Данный контекст называется *AppDbContext* и также находится в папке *Data*.

В данном микросервисе используется отдельное исключение, которое при выбросе свидетельствует о нарушении хода выполнения операции. Оно принимает строку об ошибке, которая после будет возвращена клиенту из метода контроллера, в котором произошла ошибка.

При помощи библиотеки *AutoMapper* был реализован механизм преобразования сущностей базы данных в модели, передаваемые клиенту. Данных механизм выражается посредством глобального определения профилей преобразования.

Профили преобразования сущностей находятся в папке *MapperProfiles* и включают:

– *AttachmentProfile* – профиль, выполняющий преобразование объекта базы данных *Attachment* в объект для клиентского взаимодействия *AttachmentUI*;

– *CardProfile*– профиль, выполняющий преобразование объекта базы данных *Card* в объект для клиентского взаимодействия *CardUI*;

– *FieldMapperProfile*– профиль, выполняющий преобразование объекта базы данных *Field* в объект для клиентского взаимодействия *FieldUI*;

– *ProfileMapperProfile*– профиль, выполняющий преобразование объекта базы данных *Profile* в объект для клиентского взаимодействия *ProfileUI*;

– *ProjectMapperProfile*– профиль, выполняющий преобразование объекта базы данных *Project* в объект для клиентского взаимодействия *ProjectUI*;

– *StageMapperProfile*– профиль, выполняющий преобразование объекта базы данных *Stage* в объект для клиентского взаимодействия *StageUI*;

Кроме данных моделей, которые трансформируются и передаются клиенту, приложение также включает модели, которые используются только для реализации логики приложения. Это такие связующие модели, как:

– *AccountAttachment* – модель, хранящаяся в базе данных и связывающая сущность пользователя и его вложения;

– *AttachmentType* – модель, хранящаяся в базе данных и обозначающая тип вложения;

– *Contact* – модель, хранящаяся в базе данных и связывающая пользователей, которые добавили друг друга в контакты;

– *StageAttachment* –модель, хранящаяся в базе данных и связывающая стадии проектов с их вложениями;

Для управления изменением базы данных приложения в ходе разработки используется механизм миграций. *EntityFramework* предоставляет инструменты командной строки для удобного управления изменением базы данных. При использовании данных команд создаются специальные классы миграций, которые хранят информацию что было изменено в базе данных и как это изменение в случае ошибки откатить. Данные миграции хранятся в папке *Migrations*.

Ресурсный сервис содержит следующие службы:

– *AttachmentsService* – служба, содержащая логику управления пользовательскими вложениями в базе данных и их преобразования в сущности, передаваемые клиенту;

– *CardsService* – служба, содержащая логику управления проектными карточками в базе данных и их преобразования в сущности, передаваемые клиенту;

– *FieldsService* – служба, содержащая логику управления областями проектов в базе данных и их преобразования в сущности, передаваемые клиенту;

– *ProfilesService* – служба, содержащая логику управления профилями пользователей в базе данных и их преобразования в сущности, передаваемые клиенту;

– *ProjectsService* – служба, содержащая логику управления проектами в базе данных и их преобразования в сущности, передаваемые клиенту;

– *StagesService* – служба, содержащая логику управления стадиями проектов в базе данных и их преобразования в сущности, передаваемые клиенту;

– *StatisticService* – служба, содержащая логику формирования пользовательской статистики;

Кроме служб ресурсный сервис определяет следующие инструменты:

– *ColorEvaluator* – инструмент для определения цвета стадии проекта или самого проекта в зависимости от его выполнения. Данный инструмент является реализацией графического акцентирования внимания пользователя на определённые проекты и стадии.

– *PictureConverter* – инструмент для работы с пользовательскими изображениями. Содержит логику по созданию иконок из изображений, обрезке и масштабированию изображений;

– *UserAccessValidator* – инструмент, реализующий валидацию пользовательских операций. Проверяет имеет ли пользователь права на изменение той или иной сущности в приложении.

Последним микросервисом приложения является микросервис групповых чатов. Данный микросервис отличается от предыдущих тем, что:

– он не имеет контроллеров для взаимодействия клиента с сервером по *HTTP*;

– он не имеет базы данных для хранения информации о пользовательских группах, так как группы являются динамическими;

– он имеет класс, представляющий собой *SignalR Hub*. Данный класс является абстракцией над технологиями взаимодействия клиента и сервера в реальном времени.

Класс *ChatHub* хранит информацию о текущих группах пользователей, определяет методы подключения, отключения и отправки сообщений пользователей.

Точка входа этого микросервиса не содержит конфигурацию базы данных и контроллеров приложения, однако конфигурирует интерфейс взаимодействия клиента с сервером в реальном времени.

Клиентом для всех этих микросервисов является браузерное *Angular* приложение. Данное приложение сделано при помощи современных технологий и подходов, таких как: асинхронный *JS*, *adaptive design* и так далее.

Angular приложение содержит множество файлов конфигураций для различных его аспектов: настройка языка программирования, настройка зависимостей приложения и самого приложения.

Точкой входа в приложение является файл *main.ts*. Данный класс определяет запускаемый модуль приложения и позволяет делать условное развёртывание в зависимости от параметров среды.

Весь основной код приложения содержится в папке *src*. Данная папка содержит в себе следующие директории:

– *app* – содержит компоненты приложения, их стили и *html*-шаблоны;

– *assets* – содержит статические компоненты приложения: картинки и шрифты;

– *environments* – содержит классы, хранящие информацию о различных окружениях и свойствах приложения в данных окружениях;

– *global-styles* – содержит глобальные стили приложения;

– *models* – содержит модели, используемые в приложении;

– *services* – содержит службы приложения.

Модели приложения подразделяются на части в зависимости от микросервиса, для которого они используются.

Модели для взаимодействия с авторизационным микросервисом:

– *Auth* – модель, хранящая информацию об авторизационной форме пользователя;

– *Token* – модель для хранения токена доступа приложения.

Модели для взаимодействия с микросервисом групповых чатов:

– *Group* – модель для хранения информации о многопользовательских чатах приложения;

– *Message* – модель для хранения информации о сообщениях пользователей;

Модели для взаимодействия с ресурсным микросервисом:

– *Attachment* – модель, хранящая информацию о вложениях;

– *Card*– модель, хранящая информацию о проектных карточках;

– *Field*– модель, хранящая информацию об областях проектов;

– *Profile*– модель, хранящая информацию о профилях пользователей;

– *Project*– модель, хранящая информацию о проектах;

– *Stage*– модель, хранящая информацию о стадиях проектов;

– *Statistic*– модель, хранящая статистику профиля;

Браузерное приложение имеет множество служб для получения данных с сервера. Кроме того оно имеет службы для навигации по страницам и создания модальных окон.

В службы приложения входят:

– *attachments* – служба для работы с пользовательскими вложениями;

– *auth* – служба для работы с аутентификацией и авторизацией пользователя;

– *cards* – служба для работы с проектными карточками;

– *chat* – служба для работы с групповыми пользовательскими чатами;

– *field* – служба для работы с областями проектов пользователя;

– *modal* – служба для работы с модальными окнами приложения;

– *profile* – служба для работы с профилем пользователя;

– *project* – служба для работы с проектами пользователя;

– *routing* – служба для работы с навигацией приложения;

– *stage* – служба для работы со стадиями проектов;

– *statistic* – служба для работы со статистикой пользовательского профиля;

Компоненты браузерного приложения разделены на части:

– *auth* – компонент для авторизации и регистрации пользователей;

– *footer* – компонент для реализации нижней части страницы;

– *header* – компонент для реализации верхней части страницы, основную часть которой занимает компонент навигации;

– *main* – компонент приложения, в котором содержится вся основная логика взаимодействия с пользователем;

– *nav* – компонент навигации в приложении.

Компонент *main* включает основные компоненты приложения:

– *chat* – компонент, содержащий *html*-шаблон и стили для пользовательских групповых чатов в приложении. Он разделён на более мелкие компоненты, отвечающие за отображение текущих групповых чатов и отображение текущего чата пользователя, если пользователь в нём состоит;

– *home* – компонент, отвечающий за отображение главной страницы приложения;

– *profile* – компонент, содержащий всю логику управления пользовательским профилем и его отображения. Разделён на компоненты, содержащие контакты пользователя, редактирование профиля пользователя и отображение статистики пользователя;

– *projects-manager* – самый объёмный компонент приложения, содержащий всю логику взаимодействия с пользовательскими проектами, областями проектов, их стадиями, вложениями и карточками.

Такая структура приложения делает его расширяемым, модифицируемым и относительно простым для понимания для других разработчиков.

3.3 Настройка и администрирование программного комплекса

Благодаря микросервисной архитектуре настройка любой части приложения может быть выполнена отдельно, в зависимости от необходимых параметров производительности или безопасности системы.

Каждый микросервис приложения имеет собственные файлы конфигурации. Так как все микросервисы написаны при помощи единого фреймворка, то структура и расположение файлов конфигурации одинакова для всех.

В каждом микросервисе есть папка *Properties*. В ней находится файл с конфигурацией запуска приложения. Примерами опций запуска являются:

– необходим ли запуск браузера по адресу приложения;

– использовать ли сообщения о ходе работы приложения;

– адрес приложения.

Кроме конфигурации запуска приложения есть конфигурация выполнения приложения. Данная конфигурация находится в корневом каталоге в файле *appsettings.json*. В данном файле можно настроить все аспекты выполнения приложения, например:

– логирование;

– параметры аутентификации, такие как поставщик и клиент сервиса, время жизни токена доступа и опционально секретный ключ для генерации токена. Однако секретный ключ лучше хранить в переменных окружения для большей безопасности;

– строки подключения к базам данных.

В браузерном приложении файлов конфигураций больше, так как используется более широкий спектр различных технологий. Примерами таких файлов конфигураций являются:

– *angular.json* – данный файл содержит конфигурацию *Angular* приложения. В нём можно указать основные параметры: версия, схемы, пути и т.д.

– *package.json* – в этом файле хранится конфигурация всех зависимостей приложения;

– *tsconfig.json* – в этом файле содержится конфигурация языка программирования *TypeScript* для этого приложения.

Такого рода приложения принято запускать отдельно друг от друга, однако отдельные машины для каждого сервиса выделять неэффективно. Поэтому для запуска приложения чаще всего используется контейнеризация.

Самой популярной технологией контейнеризации является *Docker*. *Docker* – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть развёрнут на любой *Linux*-системе с поддержкой контрольных групп в ядре, а также предоставляет набор команд для управления этими контейнерами.

Для контейнеризации микросервисов достаточно скачать официальный образ контейнера от *microsoft* и установить туда необходимый микросервис при помощи простой команды. Далее необходимо установить все зависимости проекта и настроить порты приложения. После этого контейнеры приложения можно распределить по виртуальным машинам, серверам или облакам.

Самой эффективной и безопасной опцией для развёртывания приложения является облако. Лучшим вариантом облака для микросервисов от *microsoft* и не только, является облако *Azure*. Большим плюсом является то, что оно имеет встроенную поддержку платформы *.Net* и полную совместимость с ней.

У данного облачного провайдера есть сервис «Контейнеры приложений Azure», позволяющий в пару кликов развернуть свой контейнер с любым приложением, написанным на любом языке и технологиях.

Кроме того, облачный провайдер сам может позаботиться о резервных копиях базы данных и общей безопасности приложений, что позволяет программистам сконцентрироваться на функционале разрабатываемого приложения.

Большинство настроек в разработанных микросервисах являются оптимальными, а изменение остальных не имеет никаких сложностей. Это достигается благодаря использованию современных подходов разработки и архитектуры приложения.

4 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС И ВЕРИФИКАЦИЯ ПРОГРАММНОГО КОМПЛЕКСА

4.1 Графическая реализация функций пользователей

При переходе по ссылке приложения пользователю предлагается войти в систему или зарегистрироваться. Окно авторизации и регистрации изображено на рисунке 4.1.

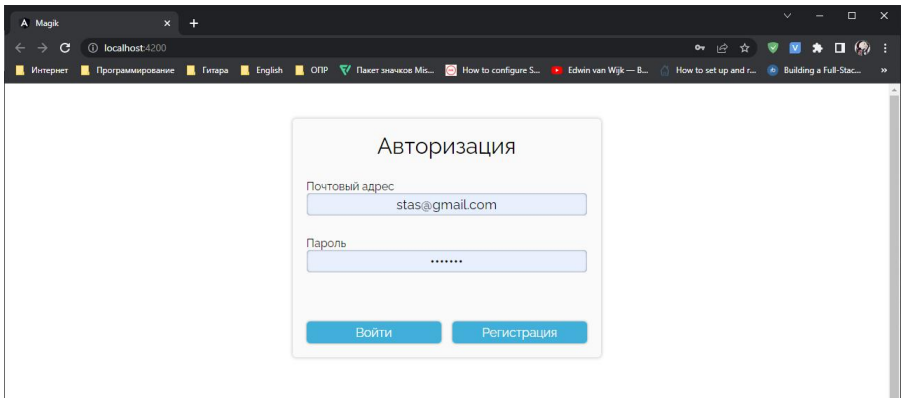

Рисунок 4.1 – Внешний вид окна авторизации и регистрации

После авторизации или регистрации пользователь переходит на главную страницу приложения. На данной странице располагается вводная информация для пользователя сайта. При использовании компьютера пользователь видит сверху панель навигации приложения. При использовании телефона панель навигации располагается снизу приложения. Внешний вид панели навигации и главной страницы изображён на рисунке 4.2.

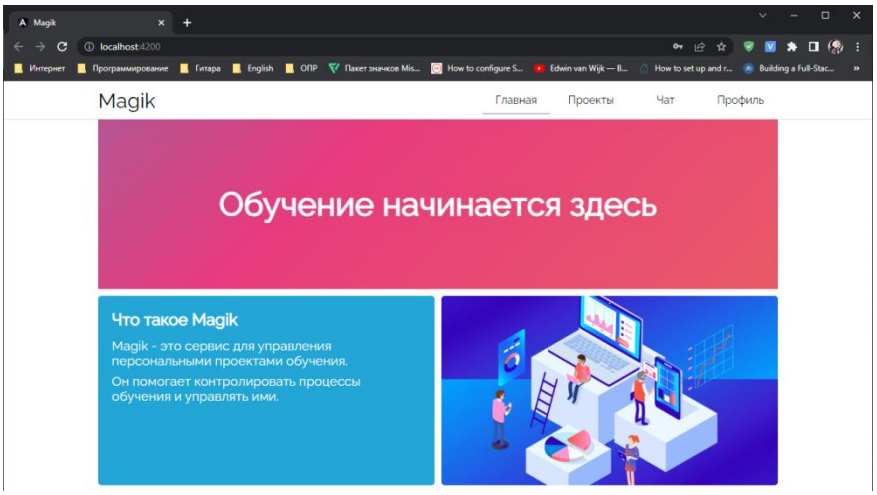

Рисунок 4.2 – Внешний вид панели навигации и главной страницы

Далее пользователь может перейти на одну из вкладок приложения. Одним из вариантов перехода является вкладка профиля пользователя.

На вкладке профиля пользователя располагается пользовательская информация и функционал редактирования профиля, просмотра статистики профиля, управления контактами пользователя и кнопка выхода из системы.

Внешний вид окна профиля пользователя изображён на рисунке 4.3.

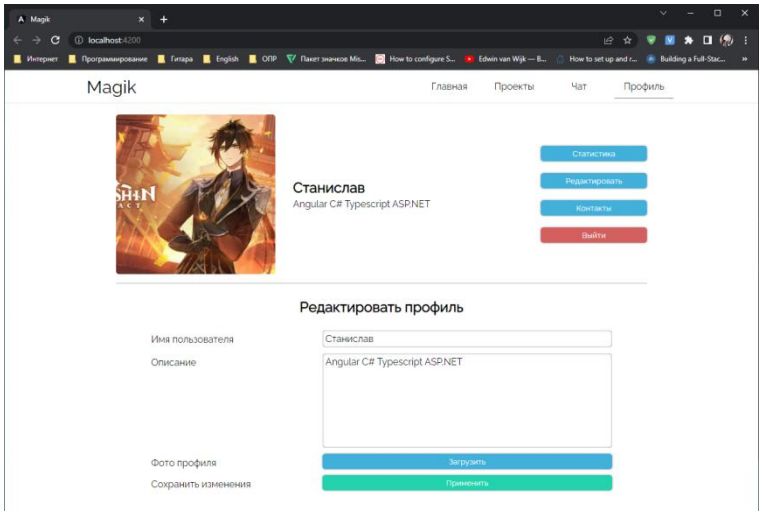

Рисунок 4.3 – Внешний вид окна профиля пользователя

Следующей вкладкой для рассмотрения является вкладка групповых пользовательских чатов. Внешний вид активного пользовательского чата изображён на рисунке 4.4.

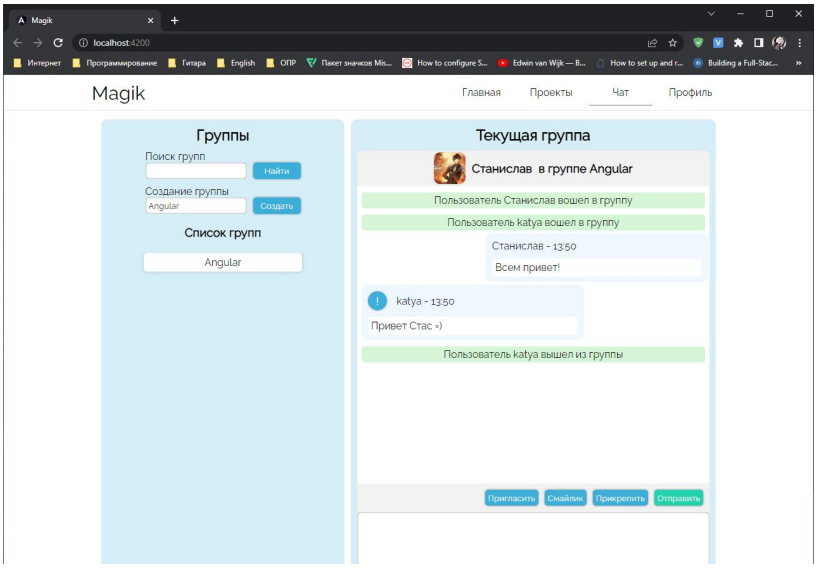

Рисунок 4.4 – Внешний вид активного пользовательского чата

Основной вкладкой приложения является вкладка проектов пользователей. При переходе на эту вкладку пользователь видит свои области проектов и может выбрать любую из них нажатием левой кнопки мыши по иконке области.

После нажатия на область проектов пользователь видит список всех проектов этой области. При этом цвет проектов и их стадий показывает

срочность их выполнения. Чем большую срочность выполнения имеет проект или его стадия, тем более красным является цвет. При полном выполнения проекта или стадии цвет проекта или стадии становится зелёным. При частичном выполнении цвет устанавливается в диапазоне от красного до зелёного в зависимости от прогресса выполнения. Внешний вид проектов, их областей и стадий изображён на рисунке 4.5.

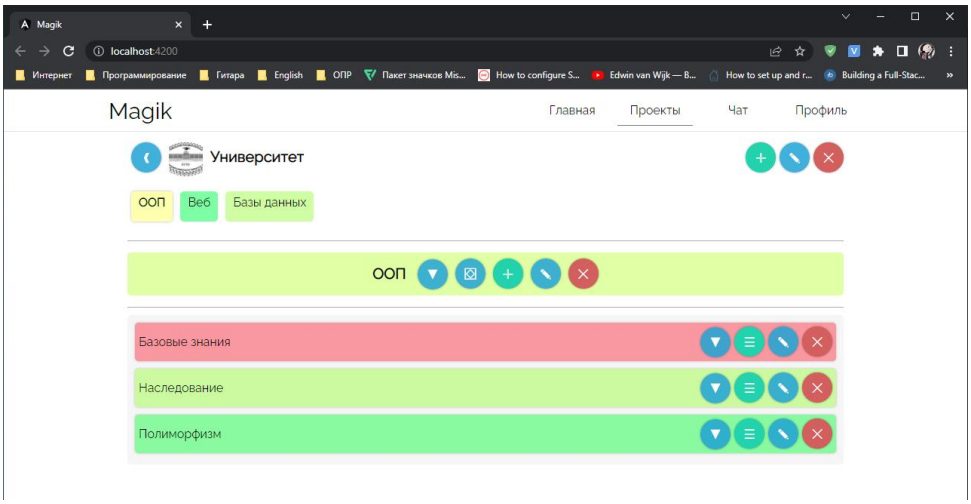

Рисунок 4.5 – Внешний вид проектов и их стадий

Каждый проект имеет набор карт для запоминания и самоконтроля знаний по этому проекту. Внешний вид модального окна карточек изображён на рисунке 4.6.

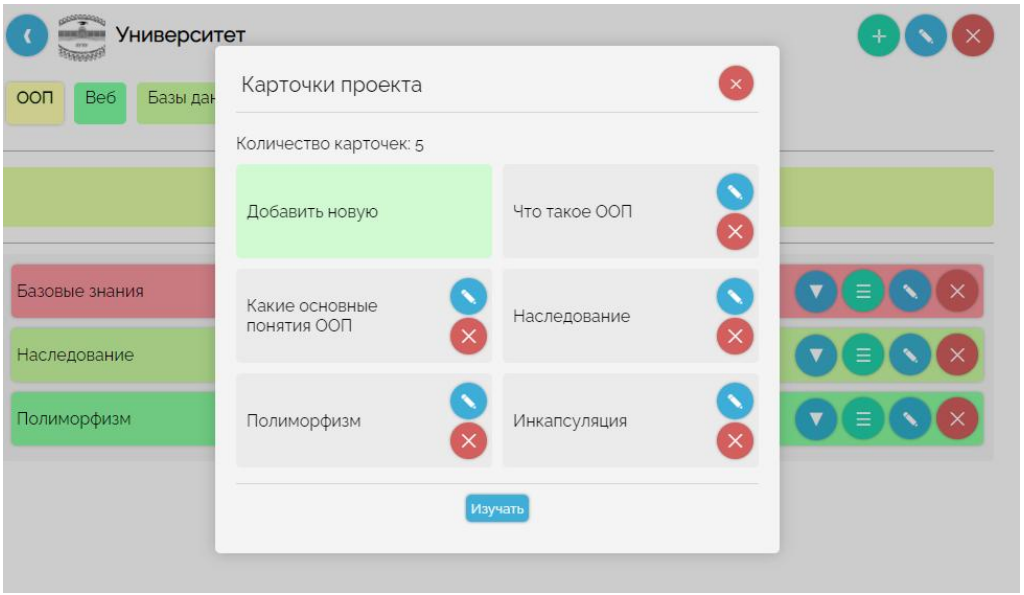

Рисунок 4.6 – Внешний вид модального окна карточек проекта

В приложении реализован функционал вложений. Благодаря ему возможно добавить в стадии проектов различные данные, относящиеся к проекту, такие как: ссылки и таблицы.

Внешний вид стадии с вложениями изображён на рисунке 4.7.

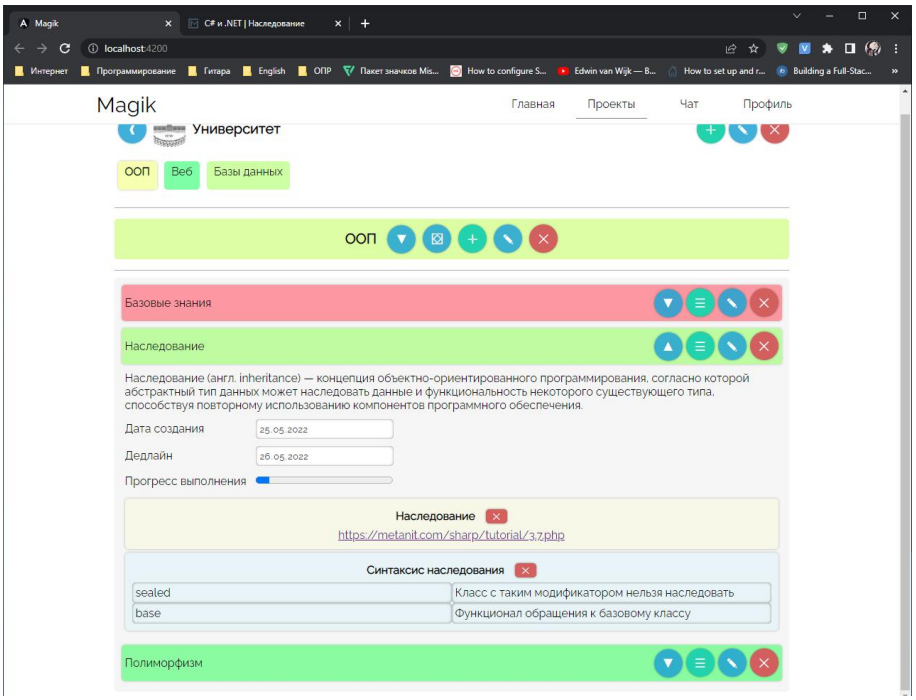

Рисунок 4.7 – Внешний вид стадии проекта с вложениями

Типовые операции, осуществляемые в приложении описаны далее.

4.2 Верификация проведения типовых операций

Для взаимодействия с системой пользователь должен зарегистрироваться или авторизироваться. Внешний вид окна авторизации изображён на рисунке Д.1.

После вхождения в систему пользователь видит главную страницу приложения. Внешний вид главной страницы приложения изображён на рисунке Д.2.

Далее пользователь может зайти в свой профиль и отредактировать необходимую информацию. Внешний вид профиля пользователя изображён на рисунке Д.3.

После этого пользователь может зайти во вкладку проекты и добавить необходимые ему области проектов. Области проектов изображены на рисунке Д.4.

Далее пользователь может нажать на область проектов и выбрать проект, с которым он хочет взаимодействовать. Выбранная область проектов и проект изображены на рисунке Д.5.

Пользователь может добавить удалить и редактировать область проектов, проект и стадию проекта. Также пользователь может раскрыть стадию проекта и просмотреть её информацию и вложения. Внешний вид раскрытой стадии изображён на рисунке Д.6.

При нажатии на кнопку вложений открывается список вложений пользователя. Внешний вид окна вложений изображён на рисунке Д.7.

В данном окне пользователь может добавить вложение в стадию, редактировать, удалить или создать новое вложение. Внешний вид окна создания нового вложения изображён на рисунке Д.8.

Также пользователь может добавить карточки для проекта, изменить их, удалить и приступить к их изучению. Внешний вид окна карточек проекта изображён на рисунке Д.9. Внешний вид изучения карточек изображён на рисунке Д.10.

Взаимодействие пользователей в пользовательских групповых чатах изображено на рисунке Д.11.

Верификация работы приложения показала, что всё работает так как нужно и все функции реализованы.

4.3 Автоматизированное тестирование программных модулей

Для того, что бы убедиться что такая сложная система будет работать как ожидалось, необходимо периодически проводить её тестирование. Проводить тестирование вручную не является возможным из-за постоянно возрастающей сложности системы. Решением является написание модульных тестов.

Практически любая современная технология позволяет писать модульные тесты, так как без них невозможно продолжать разработку продукта.

Для реализации модульных тестов в *C#* принято использовать *xUnit*. *xUnit.net* – это бесплатный инструмент модульного тестирования с открытым исходным кодом для *.NET*, написанный первоначальным автором *NUnit*.

Эта технология позволяет провести быстрые асинхронные тесты для различных модулей кода, слабо связанных между собой.

Чаще всего вместе с *xUnit* используется технология *Moq*. Она позволяет заменить доступ к источнику данных на описание возвращаемых данных в случае вызова определенного метода. Это необходимо для того, что бы у каждого теста был свой контекст исполнения и они не пересекались друг с другом и не влияли на результаты выполнения друг друга.

Angular также предоставляет функционал для тестирования работы компонентов. При генерации компонентов вместе с ними создаются файлы разметки, стилей и тестов. То есть в этом фреймворке сразу подразумевается, что программист будет тестировать то, что он разработал.

Эти тесты работают похожим образом, как и модульные тесты на сервере, однако они обладают более широким контекстом исполнения и выполняются вместе с остальными компонентами приложения.

Однако модульных тестов недостаточно для верификации работы приложения. Чаще всего разрабатывается дополнительная документация по приложению. По этой документации проводятся ручные тесты или пишутся автоматизированные тесты, моделирующие взаимодействие реального пользователя с системой.

5 ЭКОНОМИЧЕСКАЯ ЧАСТЬ

6 ОХРАНА ТРУДА

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг основных классов авторизационного сервиса

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг основных классов ресурсного сервиса

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг основных классов сервиса чатов

ПРИЛОЖЕНИЕ Г

(обязательное)

Листинг основных компонентов клиентского приложения

ПРИЛОЖЕНИЕ Д
(обязательное)
Графический интерфейс программного комплекса

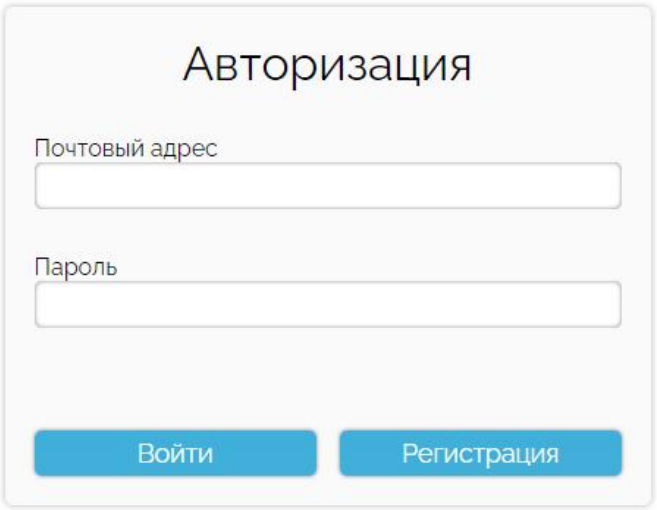

Рисунок Д.1 – Внешний вид окна авторизации

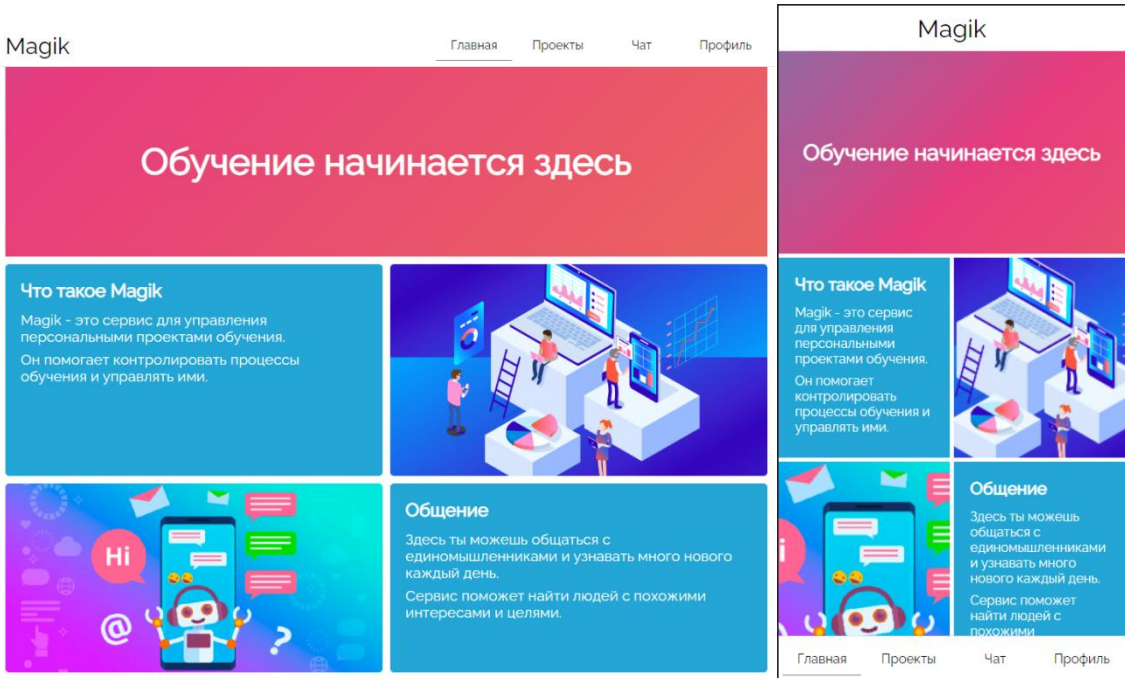

Рисунок Д.2 – Внешний вид главного окна приложения

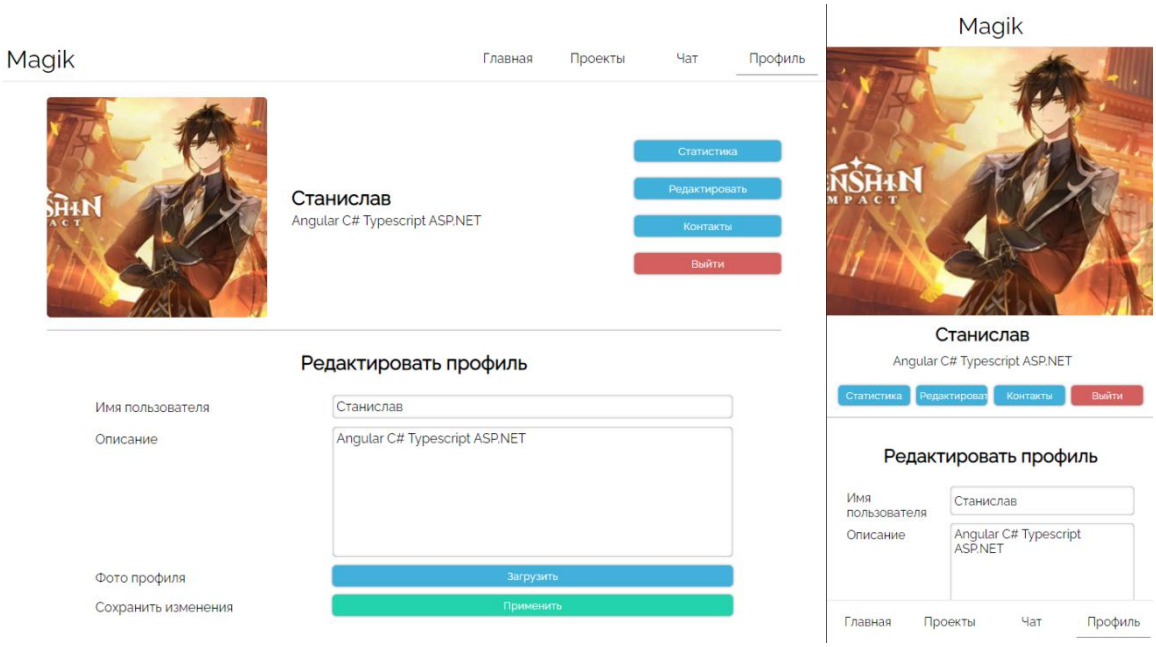

Рисунок Д.3 – Внешний вид профиля пользователя

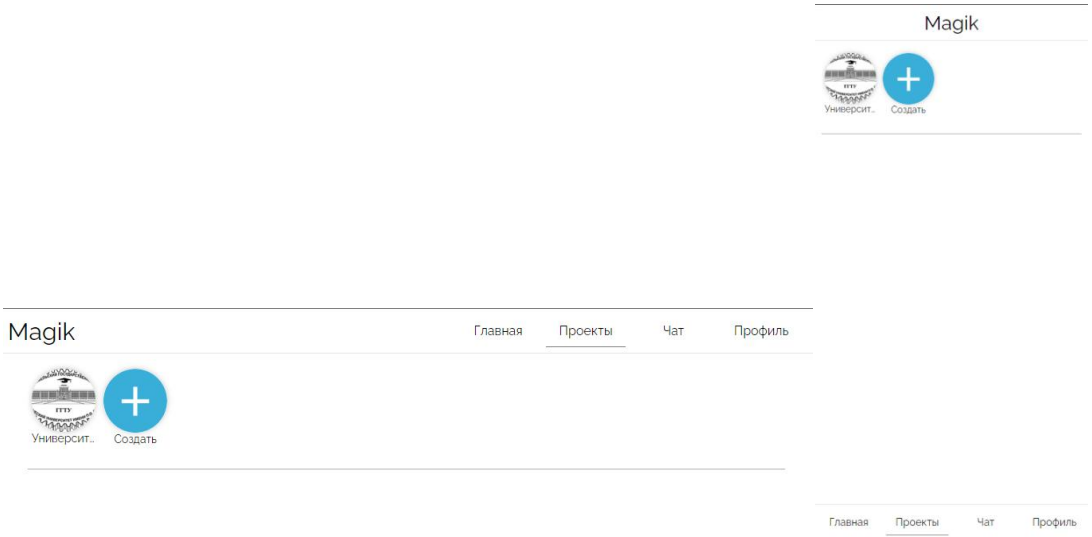

Рисунок Д.4 – Внешний вид областей проектов приложения

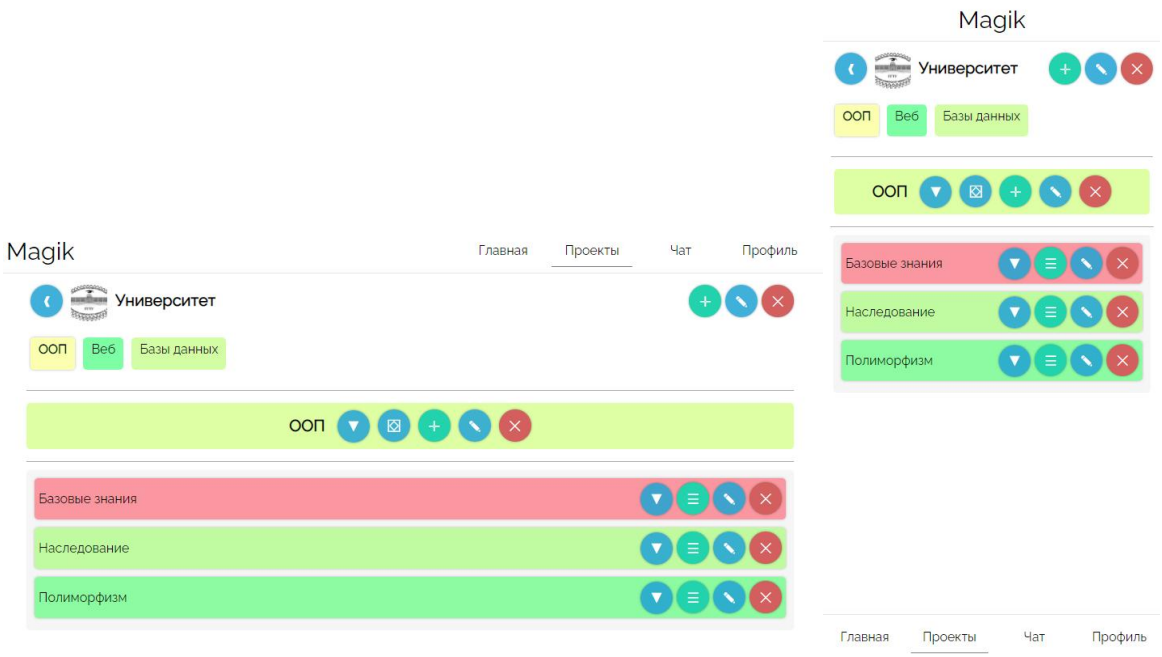

Рисунок Д.5 – Внешний вид выбранной области проектов и проекта

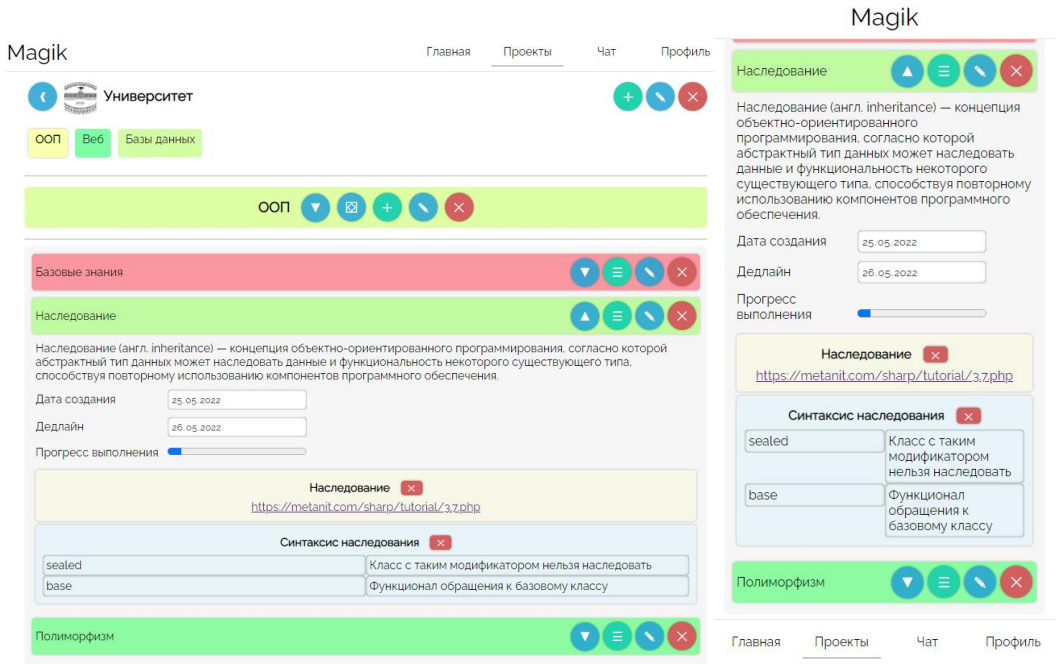

Рисунок Д.6 – Внешний вид раскрытой стадии проектов

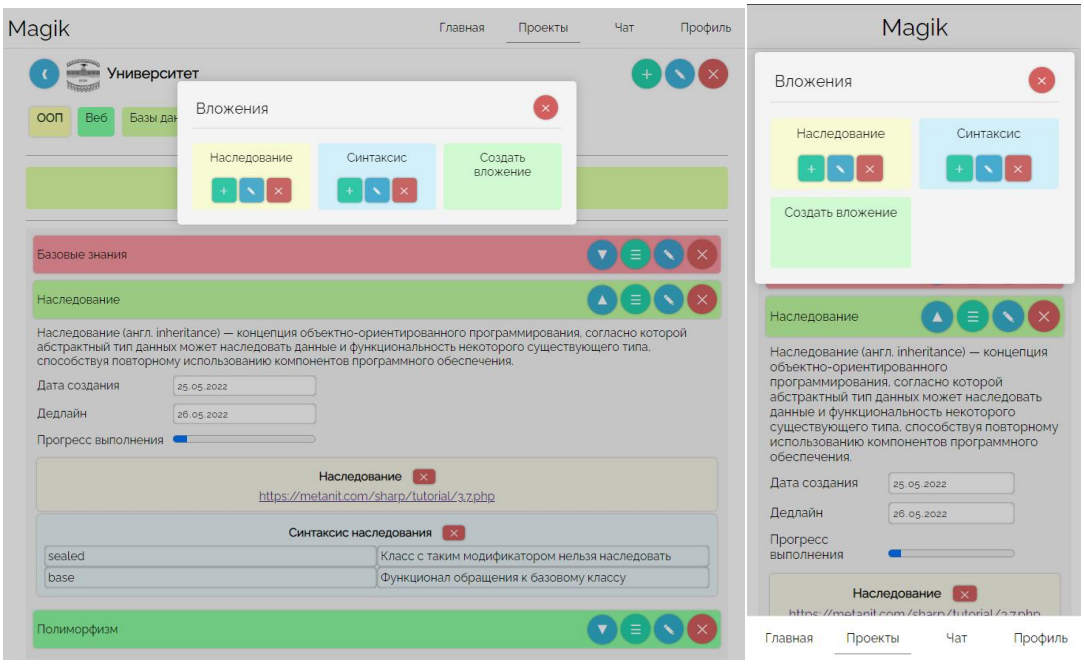

Рисунок Д.7 – Внешний вид окна вложений

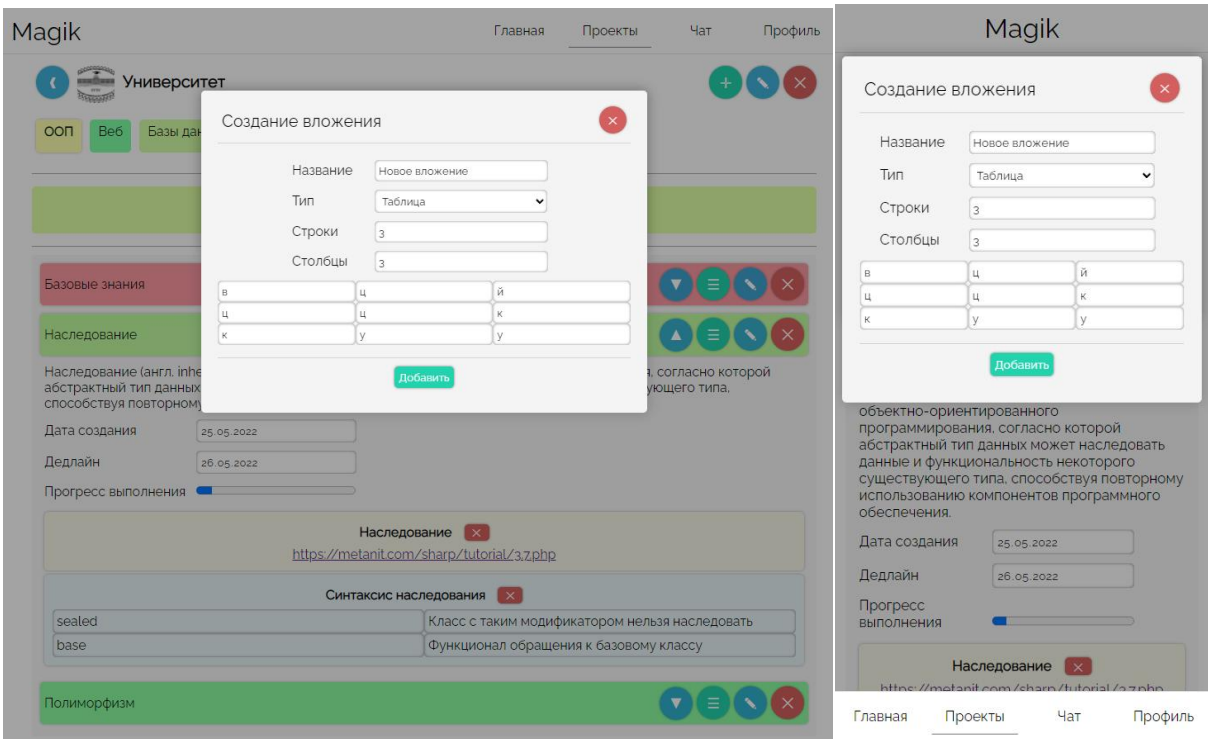

Рисунок Д.8 – Внешний вид окна создания вложения

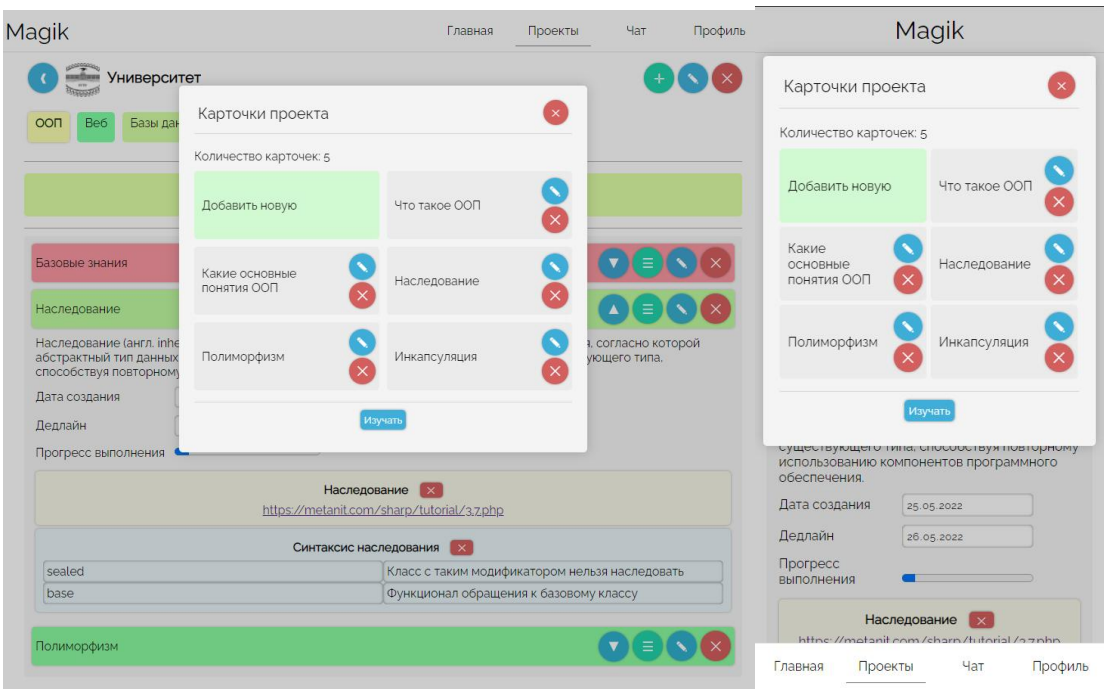

Рисунок Д.9 – Внешний вид окна карточек проекта

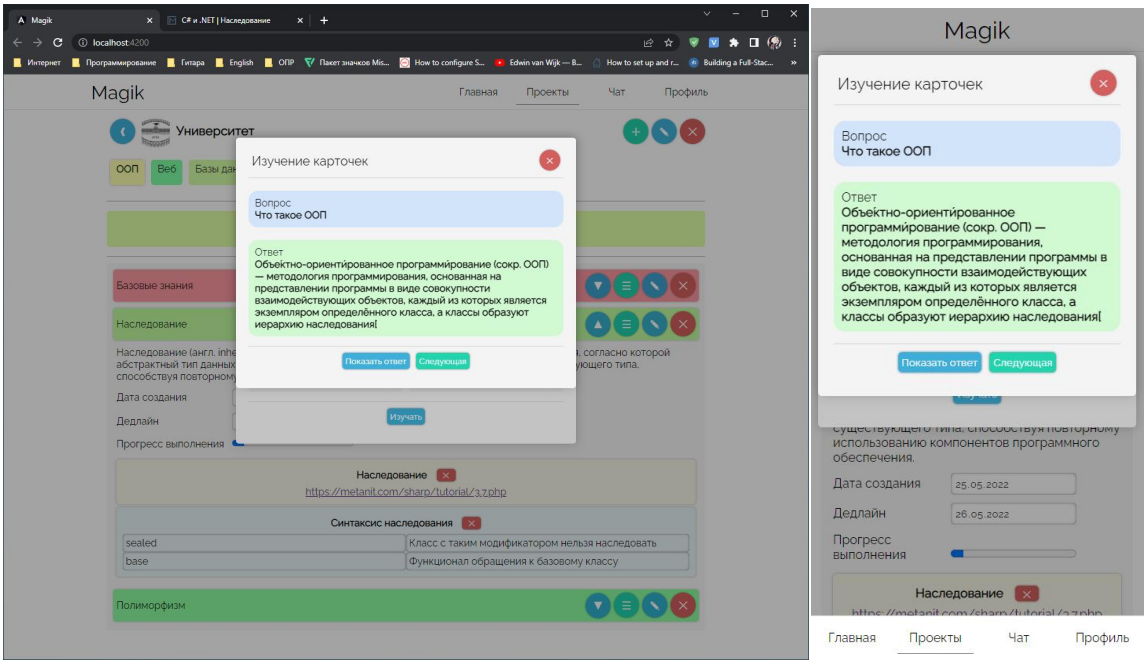

Рисунок Д.10 – Внешний вид изучения карточек проекта

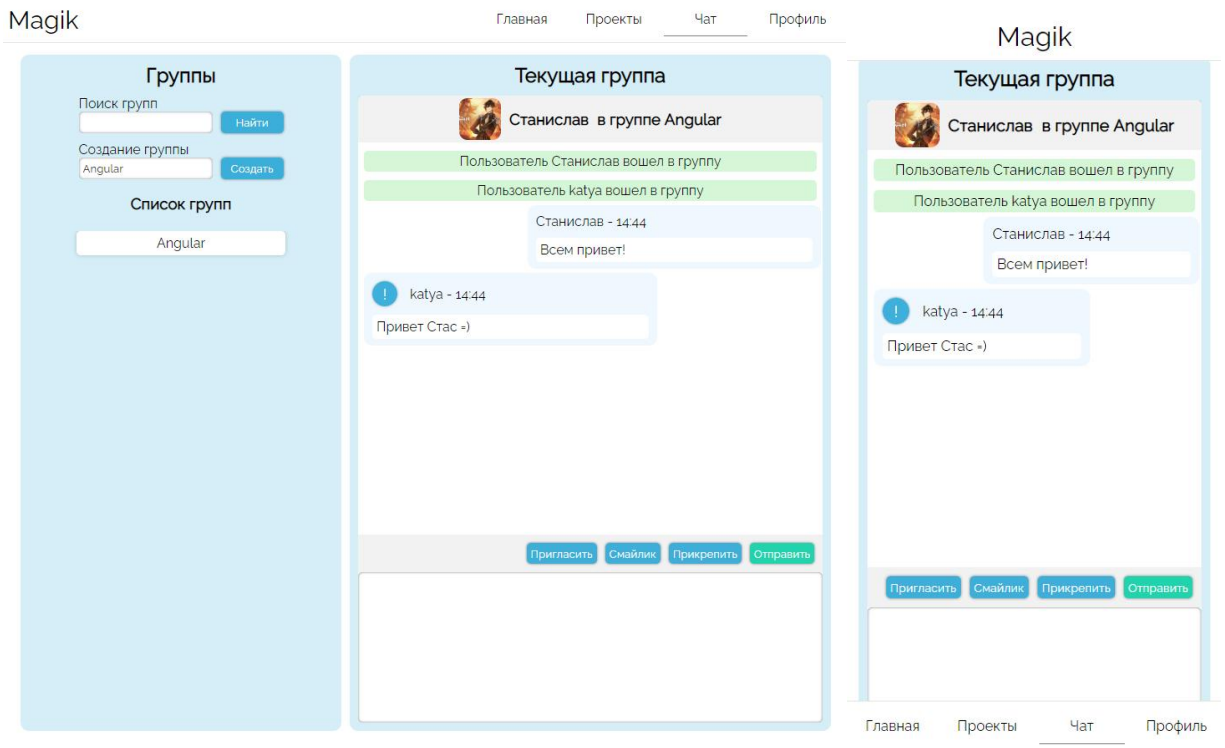

Рисунок Д.11 – Внешний вид взаимодействия пользователей в групповых чатах