Муниципальное бюджетное образовательное учреждение средняя общеобразовательная школа № 30 г. Пензы

XXVI научно-практическая конференция школьников города Пензы
«Я исследую мир»

Школьная информационно-коммуникационная система

Автор:
Сафронов Алексей,
учащийся 11А класса МБОУ СОШ № 30 г. Пензы
Научный руководитель:
В. А. Зайцев, учитель информатики высшей категории МБОУ СОШ № 30 г. Пензы

Пенза
2021

Содержание

Введение

В наше время информационные технологии стали неотъемлемой частью жизни каждого человека, организации или государства. Практически у каждого человека есть личное мобильное устройство (смартфон, планшет и т.д.). По статистике, люди используют мобильные устройства более пяти часов в день, и они, как правило, у каждого находятся под рукой, что очень удобно для ведения бизнеса или обычного общения.

В школах для взаимодействия учеников, учителей, работников школы и родителей используется большое количество сторонних приложений и сайтов. Многие процессы внутри школы недостаточно оптимизированы и автоматизированы. Например, информирование работников школы о важных распоряжениях руководства, сообщения техническому специалисту о проблемах в работе техники, заявки от классов на питание в столовую, оперативные вопросы учащихся учителю и т. п.

В связи с этим может стать хорошим решением создание своей собственной школьной сети в виде обычного приложения для мобильных устройств.

Цель проекта: Создание приложения школьной информационной сети для объединения и оптимизации функций взаимодействия различных участников образовательного процесса.

Задачи:

1. проанализировать существующие способы коммуникации и информирования, используемые в школьном процессе;
2. выделить основные функции создаваемого приложения;
3. проанализировать инструменты для создания нативных приложений, выбрать необходимые инструменты и изучить их работу и структуру;
4. создать Web-API сервис, нативное мобильное приложение, а также все необходимые вспомогательные приложения;
5. подключить и настроить online-сервис для публикации серверного Web-API приложения на нем, опубликовать мобильное приложение и провести его апробацию.

1. Теоретическая часть

1.1. Информационный анализ системы «Школа»

Школьный процесс – это особый механизм, работа которого строится на коммуникации различных категорий участников. Исследовать его изнутри наиболее удобно, когда ты являешься одним из его участников: все проблемы близки и потребности ощущаются на собственном опыте.

Процессы коммуникации в нашей школе были исследованы на основе опросов учащихся и учителей. Были выяснены основные способы коммуникации, которые существуют и выявлены их недостатки. На основе этих исследований была разработана структура приложения для коммуникации и информирования в образовательном процессе школы.

- Приложение должно включать в себя подсистему «***Пользователи***». Группы пользователей определяют права доступа пользователей к различным разделам. На начальном этапе предполагается создать группы:
  - *Администратор* – полный доступ к системе, в том числе управление другими пользователями и группами;
  - *Директор* – доступ ко всем подсистемам, для информирования и осуществления контроля над их работой;
  - *Секретарь* – доступ к системе приказов и документов;
  - *Учитель* – доступ к системе чатов, возможность создания заказов в подсистеме «Заказы в столовую», обращений в подсистеме «Заявки», просмотр назначенных на них приказов и документов;
  - *Ученик* – доступ к системе чатов для вопросов учителям;
  - *Работник столовой* – доступ к системе «Заказы в столовую»;
  - *Технический специалист* – доступ к системе «Заявки».

  Кроме выше обозначенных специфических доступов, всем пользователям доступны общие разделы системы (например, «Новости»). Пользователям предоставляются логин и пароль доступа. Одному пользователю может быть назначено несколько групп доступа. Регистрация новых пользователей доступна группе «Администратор». Редактирование данных доступно самому пользователю, кроме изменения группы.
- Подсистема «***Новости***» доступна всем группам. Новости импортируются из социальной сети «ВКонтакте», т.к. на её основе в нашей школе реализованы новости.
- «***Заказ в столовую***». Эта подсистема позволяет размещать заказы в столовую от классов, от учителей и т.д. Работник столовой может просматривать заказы и отмечать статусы их выполнения.
- «***Приказы и документы***». В этом разделе секретарь или директор публикует приказы по школе и назначает тех пользователей, кто будет их видеть (то есть для кого они предназначены), а учителя, заместители и т.д. заходят в данный раздел и видят те приказы, которые на них назначены, могут их почитать и нажать кнопку «Ознакомлен». Тем самым секретарь или директор могут вести мониторинг ознакомления со школьными документами.

- «***Заявки***». В этом разделе все пользователи, кроме учащихся могут оставлять заявки для специалистов (починить компьютер, замок в двери, свет и т.д.). Статусы заявок может изменять соответствующий специалист, который будет её выполнять.
- ***Подсистема чатов***. На её основе будут реализованы вопросы учащихся к учителям и общение пользователей в системе.

1.2. Подходы к разработке мобильных приложений

Анализ современных методов разработки мобильных приложений показал, что разработчики используют два основных подхода к созданию мобильных приложений: создание нативных программ и создание гибридных приложений. Оба способа имеют свои положительные и отрицательные стороны.

Нативные приложения — это приложения, которые создаются индивидуально под каждую платформу на родном для этой платформы языке. К примеру, приложения для Android пишутся на языке Java, а приложения для IOS – на Objective-C или заменяющем его языке Swift. Такие приложения являются сложными программами, в разработке которых чаще всего используют две команды: отдельно команда для разработки под IOS и команда для разработки под Android. Такие приложения могут быть как развлекательными, так и решениями бизнес-процессов. Нативные приложения пользуются всеми достоинствами нативной разработки:

1. Нативные приложения используют все ресурсы платформы, позволяя использовать все ее возможности.
2. Такие приложения являются максимально оптимизированными и производительными.
3. С помощью нативной разработки можно создать нативный пользовательский интерфейс, который ограничивается лишь фантазией разработчика.

Гибридная разработка подразумевает разработку сразу под все платформы. Преимущества такого способа сразу видны: разработка будет вестись в два раза быстрее за счет написания лишь одного экземпляра приложения сразу под все платформы. Однако такие приложения зачастую имеют множество ошибок и визуальных артефактов, так как разрабатывались они без целевой платформы. Такие приложения менее производительны и оптимизированы, а также их реализация ограничена языком программирования и инструментом создания, а не самой целевой платформой. Такие приложения хороши для управления небольшим объемом данных и предоставления к ним доступа пользователю, но для разработки средних или крупных проектов это будет плохим решением.

Помимо двух основных способов разработки, существует третий: *разработка веб-приложений*. Приложение создаётся в виде веб-сайта, который помещен в оболочку, которая устанавливается на мобильное устройство. Такие приложения легче всего поддерживать, и они могут функционировать на любой платформе. Этот способ меньше распространен среди разработчиков, чем нативная и гибридная разработка. Такие приложения имеют крайне ограниченные функции и не могут полноценно взаимодействовать с устройством, использовать его ресурсы, а без интернета они просто работать не будут. Поэтому данный способ разработки не позволит нам осуществить задуманное.

Для школьной сети необходимо создавать приложение сразу для всех платформ, чтобы доступ к нему был с любого устройства, но гибридная разработка все еще остается довольно сильно ограничивающей разработчика, из-за чего выбор был сделан в пользу нативной разработки. Однако, нативная разработка подразумевает разработку приложения в двух экземплярах на двух различных языках программирования, что является крайне сложной задачей для одного разработчика, поэтому было решено искать обходной путь. Подходящее решение было найдено в библиотеке Xamarin под язык C#.

1.3. Платформа Xamarin

Xamarin – это фреймворк для создания кроссплатформенных приложений на языке C#.

Этот фреймворк является продуктом компании Microsoft и в настоящее время активно развивается. Его преимущество заключается в том, что фактически требуется написать код приложения лишь один раз, но приложение будет иметь большую часть преимуществ нативной разработки, что означает доступ ко всем функциям целевых платформ. Данное решение позволяет писать всю бизнес-логику приложения только один раз, а также писать до 90% оставшегося кода также в единственном экземпляре. Оставшиеся 10% - это различные тонкости и настройки конкретных целевых платформ, однако даже эта часть кода пишется на языке C#, а не на различных родных для платформ языках.

Этот фреймворк является довольно молодым продуктом, однако он уже успел обзавестись своими соглашениями разработчиков о структуре приложений. Так, в подавляющей части разрабатываемых на этой платформе приложений используется архитектура MVVM – Model-View-ViewModel. Данная архитектура подразумевает в себе три основных элемента:

- Model – модель или же данные приложения, которые используются в нем. Это может быть объект пользователя или же объект какой-то записи в приложении.
- View – представление или же пользовательский интерфейс. Это то, что отображается пользователю, когда он взаимодействует с приложением.
- ViewModel – промежуточный слой между представлением и моделью данных. Он обеспечивает их взаимодействие и описывает в себе все методы, которые может использовать конкретная модель данных. К нему обращается представление, через которое получает доступ к модели данных или же к данным приложения.

Помимо языка C#, в платформе Xamarin используется язык разметки XAML. Это расширяемый язык разметки, основанный на языке разметки XML для декларативного программирования. В данном случае этот язык позволяет создавать пользовательский интерфейс приложения, который и будет видеть пользователь на протяжении всего времени пользования приложением. В данном языке присутствует возможность разделять элементы на различные страницы, которые могут содержать в себе разный контент. К примеру, страница-список или страница навигации.

1.4. База данных SQL

Все данные приложения необходимо где-то хранить. Для этой цели используются базы данных, которые позволяют структурировать данные в различных видах. Наиболее удобными базами данных для создания информационных систем являются *реляционные*, которые позволяют

хранить все данные в виде таблиц. Управление ими осуществляется с помощью языка SQL. Подавляющая часть информационных систем работает на таких базах данных.

1.5. API-приложение

Нативное приложение и база данных должны взаимодействовать между собой, однако размещать базу данных на устройстве пользователя будет не лучшей идеей, ибо такой способ не позволяет сделать данные приложения доступными всем пользователям и делает приложение offline-приложением. Взаимодействовать с удаленной базой данных можно напрямую, однако такой способ захламляет код нативного приложения и делает его более емким. Более того, способ отправки запросов напрямую является самым небезопасным способом доступа к данным, ибо такие запросы очень легко перехватить.

Решением этой проблемы является создание API-приложения — промежуточного слоя между базой данных и мобильным приложением. Так как мобильное приложение разрабатывается на языке программирования C#, создание API-приложения также будет вестись на данном языке программирования с помощью платформы ASP.NET.

ASP.NET – это платформа разработки веб-приложений от компании Microsoft. Эта платформа довольно популярна среди разработчиков благодаря активному развитию и большому функционалу платформы. Программная модель ASP.NET основывается на протоколе HTTP и использует его для взаимодействия между сервером и браузером.

Для мобильного приложения было решено создать Restfull API — это такое веб-приложение, которое посредством запроса к конкретным адресам выдает соответствующие запросам ответы. То есть, допустим, мы отсылаем GET-запрос на условный адрес server.net/restfull. Для этой страницы в самом API-приложении прописан запрос GET, который выдает нам данные в JSON формате о всех пользователях приложения. На стороне сервера этот запрос обрабатывается, API-приложение обращается к базе данных для получения соответствующих данных, после чего обрабатывает эти данные, приводит к JSON формату и отсылает его обратно в наше мобильное приложение, где эти данные снова обрабатываются и приводятся к виду, которое необходимо приложению. Таким образом работают все запросы мобильного приложения, будь то получение, удаление, добавление или изменение данных в базе данных.

Структура API-приложения основывается на паттерне MVC – Model-View-Controller. Данный паттерн подразумевает разделение проекта на три основные составляющие: модель данных, внешнее представление и контроллер, который обеспечивает связь между пользователем, в нашем случаем клиентом приложения, и остальными частями приложения.

Модель (Model) – является описанием конкретного объекта, хранящегося в базе данных. Эта составляющая сама по себе не содержит данные, она лишь описывается то, как они должны храниться и производит их валидацию. Модель может содержать в себе логику управления данными, однако она не должна содержать логики отображения данных в представлении и взаимодействия с пользователем.

Контроллер (Controller) – представляет из себя центральный компонент данного паттерна, который обеспечивает связь между пользователем и внутренностями приложения. Он обрабатывает запросы, которые делает пользователь, и, в зависимости от результатов обработки, отправляет пользователю определенный вывод, например, в виде представления.

Представление (View) – этот компонент не содержит в себе логики обработки запросов или управления данными. Представление управляет самим отображением. При создании веб-сайта этот компонент чаще всего содержит в себе HTML-страницу, однако в API он фактически отсутствует из-за ненадобности отображения пользователю какого-либо контента. В API представление заменяется JSON-ответами, которые содержат в себе какие-либо данные и с которыми в дальнейшем идет работа в сторонних приложениях.

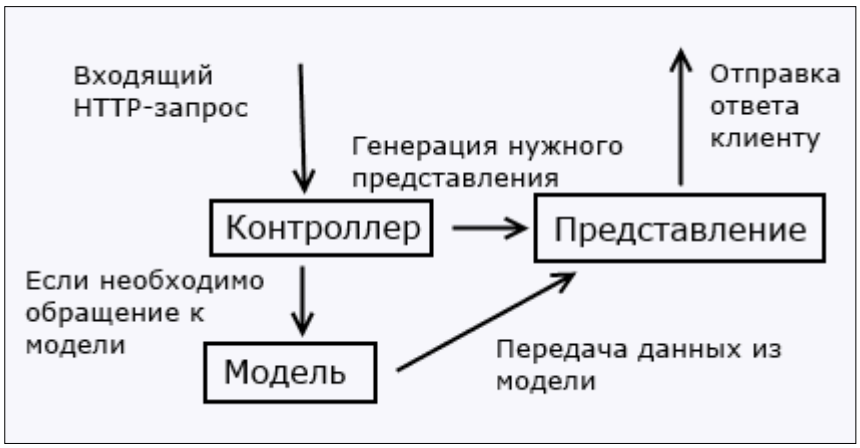

Рис 1. Схема работы MVC паттерна

Так как API-приложение по своей сути является обычным веб-сайтом, то, перейдя на определенные страницы в браузере, можно получить данные, которые привязаны к этой самой странице. Для предотвращения всеобщего доступа к данным используются множественные проверки пользователей и защита данных каким-либо ключом. В своем приложении я решил защитить API-запросы посредством дополнительной передачи токена и ключа пользователя, которые между собой связаны, в любом запросе, будь то GET, POST, PUT или DELETE запрос. Такой способ позволяет идентифицировать пользователя, путем сравнения передаваемых данных с данными, которые находятся в базе данных, и отклонять запрос в случае, если передаваемые ключи окажутся неверными. Сами ключи один раз передаются на устройство пользователя и хранятся на нем в защищенном виде в специальном хранилище Xamarin, и доступ к ним не имеет даже сам пользователь, поскольку с этими данными работает код приложения.

2. Практическая часть

2.1. Требования к API приложения

API является связующим звеном между мобильным приложением и данными, а значит, оно должно иметь представления для передачи данных. API приложение должно иметь реализацию следующей функциональности:

- ограниченная выборка пользователей по логину и паролю, а также возможность добавления данных в базу данных,
- выборка, вставка и изменение данных таблиц «Заявки на работу», «Приказы», «Оценки», «Заказы в столовую»,
- индивидуальная выборка по любым данным из различных таблиц для технических задач администраторов,
- API должно иметь систему защиты доступа к данным.

Данная функциональность реализуется посредством контроллеров и моделей данных, копирующих отдельные таблицы в базе данных.

2.2. Требования к нативному приложению

Согласно плану приложения, оно должно иметь следующую функциональность:

- *Система пользователей*: авторизация с помощью пароля, доступ к различным разделам приложения или же его ограничение, а также свой набор возможных действий.
- *Система подачи заявок в столовую*: пользователи с уровнем доступа «Учитель» и аналогичный ему должны иметь возможность подать заявку в столовую на свой класс (кол-во человек и номер класса), которую смогут просмотреть работники школьной столовой.
- *Система заявок*: в данном разделе все пользователи, кроме тех, кто имеет уровень доступа «Ученик» и «Родитель», должны иметь право подать заявку для специалиста (ремонт техники, материальных вещей и т. д.), который может просматривать эти заявки и устанавливать им свой статус выполнения.
- *Система приказов*: рассылка школьных приказов пользователям, список которых назначается самим рассылающим пользователем, а также отслеживание ознакомления пользователей с этим приказом.
- *Мессенджер*: приложение должно иметь базовую функциональность мессенджера, а именно систему чатов для обеспечения связи между пользователями приложения.
- *Система новостей*: таргетированный импорт новостей из социальных сетей, также каждый пользователь имеет доступ к меню «новости» и может как просматривать записи, так и создавать свои.

Приложение также должно иметь интуитивный и не перегруженный интерфейс, удобный обычному пользователю. Отдельные страницы доступны только пользователям с соответствующим уровнем доступа.

2.3. Разработка приложения

Для удобства разработки структуры приложения был выбран сервис diagrams.net, в котором была визуализирована база данных и все ее связи между таблицами.

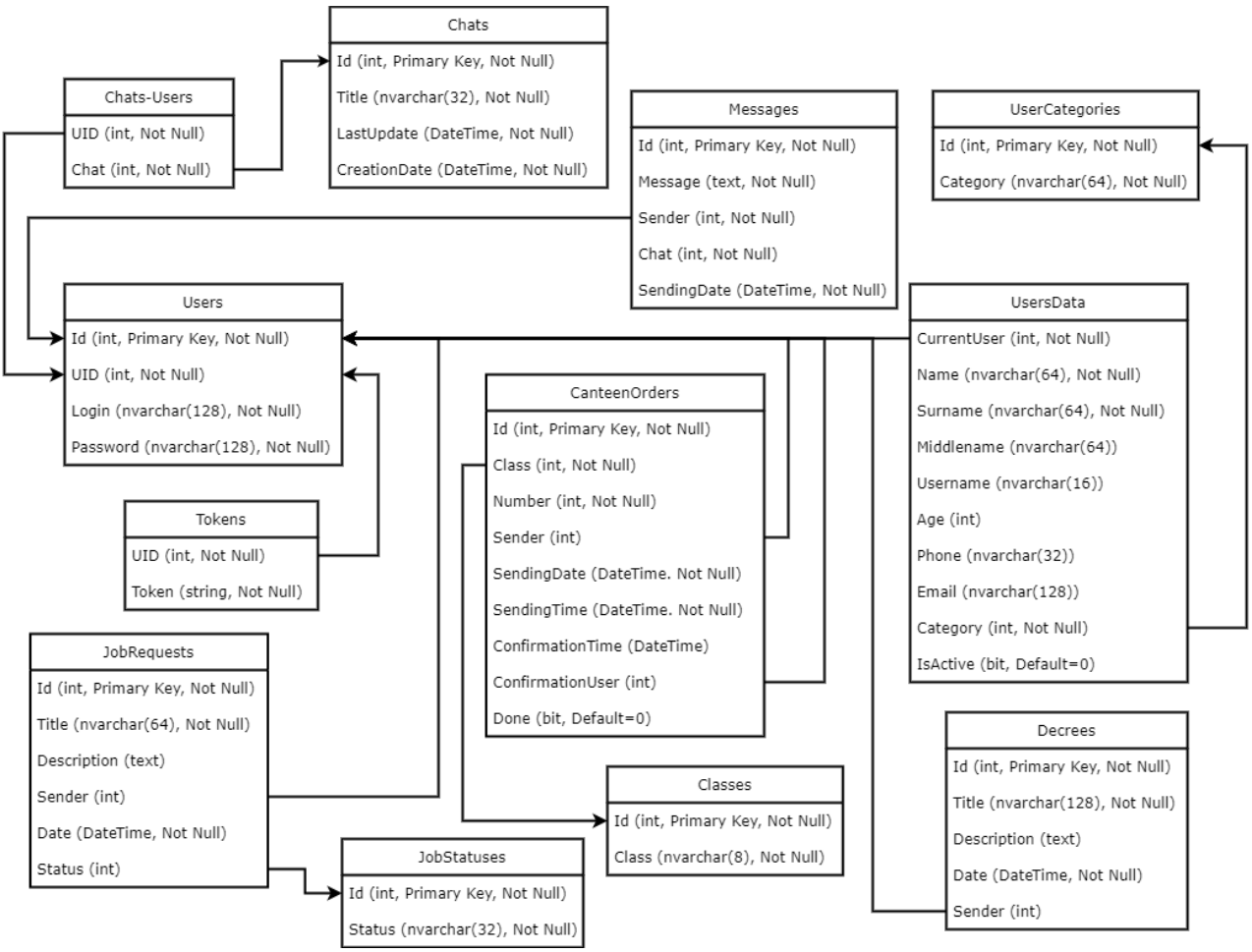

Рис 2. Часть схемы базы данных

По данной схеме была создана и настроена база данных. Также созданы модели в API приложении, описывающие отдельные таблицы базы данных.

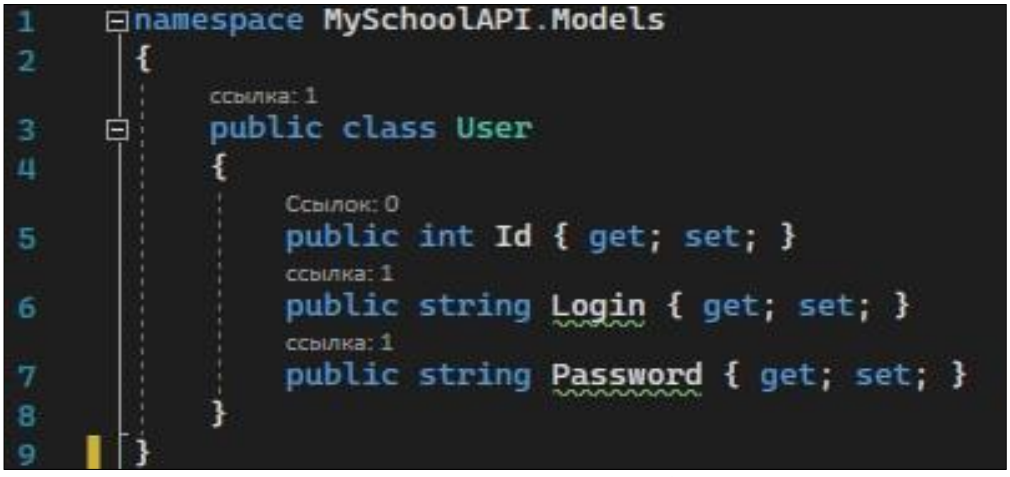

Рис 3. Пример модели таблицы

На основе этих моделей были созданы контроллеры-представления запросов к базе данных: общая выборка, выборка по данным, удаление и изменение данных.

Для передачи данных нативному приложению в проекте приложения были созданы константы для подключения к API-приложению, а также прописаны методы получения и передачи данных на основе контроллеров API-приложения. Получаемые данные передаются в представления, которые объединяют их в группы и отображают пользователям.

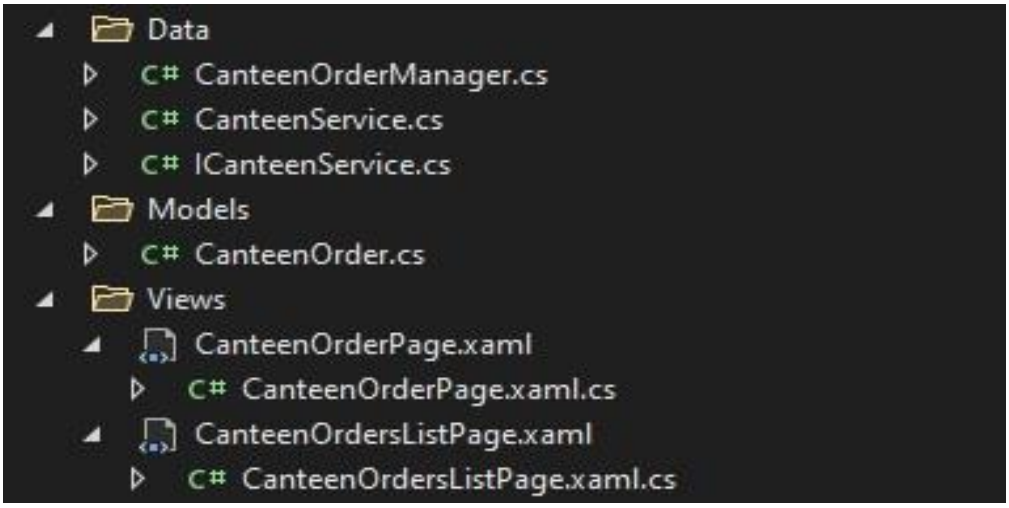

Рис 4. Часть структуры приложения

Представления «оборачивают» полученные данные в привычную пользователям оболочку и отображают итоговый результат, с которым взаимодействуют пользователи.

По такой схеме реализуются функции API и нативного приложения, перечисленные в требованиях.

Для создания базового дизайна приложения был использован сервис Proto IO, в котором был визуализирован пользовательский интерфейс. На ранней стадии разработки было решено не создавать сложный дизайн ради создания основы всего приложения.

Примеры исходного кода API представлены в приложении № 1.

Примеры исходного кода приложения представлены в приложении № 2.

Все остальные части приложения реализованы подобным образом.

2.4. Программа реализации проекта

Программа реализации проекта содержит в себе перечень необходимых этапов разработки, внедрения и апробации с примерными сроками реализации, перечень необходимых ресурсов, а также перспективные планы доработки и поддержки.

Этап	Описание	Сроки
Подготовительный	Работа над проблемой, формулирование цели и задач проекта, исследование целевого контингента участников и их потребностей, формирование основного вектора разработки	сентябрь-октябрь 2021 г.

Разработка	Изучение источников информации, выбор наиболее удобного способа реализации и продумывание функционала приложения, разработка структуры базы данных, написание кода API и клиентского приложений	октябрь-ноябрь 2021 г.
Тестирование	Проверка основных функций приложения, защищённости данных, возможных ошибок, стабильности и быстродействия	ноябрь 2021 г.
Доработка	Исправление ошибок, улучшение интуитивности интерфейса, доработка функционала	декабрь – январь 2021 г.
Апробация	Предоставление доступа к приложению ограниченному кругу пользователей для каждой роли, получение от них обратной связи	январь 2021 г.
Внедрение	Запуск приложения для использования в рамках школы	февраль – май 2021 г.

Ресурсы, необходимые для стабильной работы системы:

- сервер базы данных,
- специалист разработки и поддержки приложения,
- наличие стабильного доступа к сети Интернет,
- наличие персональных мобильных устройств у пользователей.

Оценка возможных рисков при реализации проекта:

- ✓ *Заинтересованность целевой аудитории пользователей.* Приложение оптимизирует процессы коммуникации пользователей, упрощает взаимодействие и сокращает временные затраты пользователей.
- ✓ *Защита данных.* Внутри системы существует разграничение прав доступа к определённым данным, доступ к приложению имеют только пользователи, зарегистрированные администратором. Внешние подключения исключаются. Данные хранятся на защищённом сервере на территории РФ.
- ✓ *Стабильность доступа к сети.* Доступ к сети Интернет доступен почти на всех современных мобильных устройствах и на всех тарифах провайдеров. Современное мобильное устройство почти нереально вообразить без доступа к сети Интернет.
- ✓ *Стабильность работы приложения и его поддержка.* В случае внедрения приложения в образовательный процесс планируется его поддержка и регулярные обновления.

Перспективный план возможных доработок приложения:

- Система электронного дневника: доступ к оценкам и домашнему заданию прямо из приложения, а также настраиваемые напоминания и общие уведомления.

- Автоматическая интеграция школьных новостей с различных внешних источников в приложение без дополнительных действий, а также разделение новостей на разные подкатегории.
- В поздней разработке допускается возможность расширения приложения с внутришкольного до межшкольного путем привязки всех данных к конкретной школе.

Заключение

В процессе работы над приложением были проанализированы существующие методы коммуникации между участниками образовательного процесса, выделены их достоинства и недостатки, сформированы необходимые функции мобильного приложения.

В процессе знакомства и изучения инструментов для создания приложений был выбран инструментарий и сформулирован план создаваемого приложения.

Был создан Web-API сервис, предоставляющий корректный доступ к данным по следующим направлениям: мессенджер, новости, пользователи, заказы в столовую. Также была разработана и реализована система проверки пользователей по секретному ключу, который запрашивается при каждом запросе и сравнивается с актуальным ключом в базе данных. Данная система позволяет эффективно защищать данные, затрачивая минимальное количество ресурсов.

Цель проекта достигнута, а задачи решены.

Работа над проектом продолжается в соответствии с Программой реализации.

Источники

1. Руководство по программированию для Xamarin Forms // METANIT URL: https://metanit.com/sharp/xamarin/
2. ASP.NET // ВикипедиЯ URL: https://ru.wikipedia.org/wiki/ASP.NET
3. XAML // ВикипедиЯ URL: https://ru.wikipedia.org/wiki/XAML
4. REST // ВикипедиЯ URL: https://ru.wikipedia.org/wiki/REST
5. C Sharp // ВикипедиЯ URL: https://ru.wikipedia.org/wiki/C_Sharp
6. Мобильные приложения: нативные, веб и гибридные // Agilie URL: https://agilie.com/ru/blog/mobilnyie-prilozhieniia-nativnyie-vieb-i-ghibridnyie
7. Подходы к разработке мобильных приложений // Xdan URL: https://xdan.ru/podkhody-k-razrabotke-mobilnykh-prilozhenij.html
8. Документация по ASP.NET // Microsoft URL: https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-6.0
9. Документация по Xamarin // Microsoft URL: https://docs.microsoft.com/ru-ru/xamarin/
10. Создание диаграмм (макет БД) // Diagrams URL: https://www.diagrams.net/
11. Множественные статьи с сайта Xamdev // Xamarin Developers URL: https://xamdev.ru/
12. К. Ю. Поляков, Е. А. Еремин Информатика // К. Ю. Поляков, Е. А. Еремин — Москва. «Просвещение» 2021г.

Приложение № 1. Примеры исходного кода API-приложения.

Типичный Get запрос к БД через API на примере получения списка записей новостей:

```
[HttpGet("uid={uid}&token={token}")]
Ссылок: 0
public JsonResult Get(int uid, string token)
{
    if (!TokenController.CheckToken(_configuration, uid, token))
    {
        return new JsonResult("Invalid token");
    }

    string query = @"
                    SELECT Articles.Id, Articles.Content, Articles.Author,
                    Articles.PublicationTime, Usersdata.Name AS AuthorName, Usersdata.Surname AS AuthorSurname
                    FROM Articles
                    LEFT JOIN Usersdata ON(Usersdata.CurrentUser=Articles.Author)
                    ORDER BY Articles.PublicationTime DESC
                    ";

    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("MySchoolAppCon");
    MySqlDataReader myReader;
    using (MySqlConnection myCon = new MySqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (MySqlCommand myCommand = new MySqlCommand(query, myCon))
        {
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
            myCon.Close();
        }
    }

    return new JsonResult(table);
}
```

Post запрос на примере добавления записи новостей:

```
[HttpPost("uid={uid}&token={token}")]
Ссылок: 0
public JsonResult Post(Article item, int uid, string token)
{
    if (!TokenController.CheckToken(_configuration, uid, token))
    {
        return new JsonResult("Invalid token");
    }

    string query = @"
                    INSERT INTO Articles(Content, Author, PublicationTime)
                    VALUES (@Content, @Author, @Time)
                    ";

    DataTable table = new DataTable();
    string sqlDataSource = _configuration.GetConnectionString("MySchoolAppCon");
    MySqlDataReader myReader;
    using (MySqlConnection myCon = new MySqlConnection(sqlDataSource))
    {
        myCon.Open();
        using (MySqlCommand myCommand = new MySqlCommand(query, myCon))
        {
            DateTime timeNow = DateTime.Now;
            myCommand.Parameters.AddWithValue("Content", item.Content);
            myCommand.Parameters.AddWithValue("Author", item.Author);
            myCommand.Parameters.AddWithValue("Time", timeNow);
            myReader = myCommand.ExecuteReader();
            table.Load(myReader);
            myReader.Close();
            myCon.Close();
        }
    }

    return new JsonResult("Added Successfully");
}
```

Приложение № 2. Примеры исходного кода пользовательского приложения.

Пример интерфейса (записи новостей), который описывает свой конкретный сервис (сервис работает с API):

```
Ссылок: 3
public interface IArticleService
{
    Ссылок: 2
    Task<List<ArticleExtendedModel>> RefreshDataAsync();

    Ссылок: 2
    Task<List<ArticleExtendedModel>> RefreshDataByUserAsync(int userId);

    Ссылок: 2
    Task SaveItemAsync(ArticleModel article);
}
```

Получения списка записей в сервисе на примере получения списка записей новостей (обращение к API):

```
Ссылок: 2
public async Task<List<ArticleExtendedModel>> RefreshDataAsync()
{
    List<ArticleExtendedModel> Items = new List<ArticleExtendedModel>();

    Uri uri = new Uri(Constants.RestBaseUrl + "article/" + Constants.GetTokenString());

    try
    {
        HttpResponseMessage response = await client.GetAsync(uri);
        string content = await response.Content.ReadAsStringAsync();

        Items = JsonSerializer.Deserialize<List<ArticleExtendedModel>>(content);
    }
    catch (Exception ex)
    {
        Debug.WriteLine("ERROR URL: " + uri);
        Debug.WriteLine(@"\t ERROR {0}", ex.Message);
    }

    return Items;
}
```

Сохранение записи в сервисе на примере сохранения записи новостей (обращение к API):

```
Ссылок: 2
public async Task SaveItemAsync(ArticleModel article)
{
    Uri uri = new Uri(Constants.RestBaseUrl + "article/" + Constants.GetTokenString());
    try
    {
        string json = JsonSerializer.Serialize<ArticleModel>(article, serializerOptions);
        StringContent content = new StringContent(json, Encoding.UTF8, "application/json");

        HttpResponseMessage response = null;

        response = await client.PostAsync(uri, content);

        if (response.IsSuccessStatusCode)
            Debug.WriteLine(@"\t Article successfully posted");
        else
        {
            Debug.WriteLine(response);
            Debug.WriteLine("ERROR URL: " + uri);
            Debug.WriteLine("ERROR JSON: " + json);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@"\t ERROR {0}", ex.Message);
        Debug.WriteLine("ERROR URL: " + uri);
    }
}
```

Менеджер, который пишется для каждого интерфейса и объявляется один раз на уровне приложения чтобы обращаться к нему для работы с API:

```
Ссылок: 3
public class ArticleManager
{
    IArticleService articleService;

    ссылка: 1
    public ArticleManager(IArticleService service)
    {
        articleService = service;
    }

    ссылка: 1
    public Task<List<ArticleExtendedModel>> GetTaskAsync()
    {
        return articleService.RefreshDataAsync();
    }

    ссылка: 1
    public Task<List<ArticleExtendedModel>> GetTaskByUserAsync(int userId)
    {
        return articleService.RefreshDataByUserAsync(userId);
    }

    ссылка: 1
    public Task SaveTaskAsync(ArticleModel item)
    {
        return articleService.SaveItemAsync(item);
    }
}
```