МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 “Программное обеспечение информационной

безопасности мобильных систем”

Специализация Инженер-программист

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Мобильное приложение для салона оптики “Klingen Opticals”»

Выполнил студент Тарайкович Владислав Игоревич

(Ф.И.О.)

Руководитель проекта асс. Нистюк О.А. (учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В . (учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс. Нистюк О.А. (учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2021

Реферат

Пояснительная записка курсового проекта содержит 34 страницы пояснительной записки, 8 иллюстраций, 7 источников литературы, 2 приложения.

Цель курсового проекта – разработка программных модулей, которые будут корректно взаимодействовать между собой и смогут обеспечить безошибочное функционирование приложения для салона оптик.

В первой главе проводится анализ предметной области и формирование функционала.

Вторая глава посвящена процессу проектирования системы и описанию технологий, которые будут использованы в проекте.

В третьей главе описаны подробности реализации.

В четвертой главе описано тестирование и проверка работоспособности получившихся приложений.

В пятой главе описывается руководство пользователя.

В заключении приведены результаты проделанной работы.

Содержание

[Реферат 2](#_Toc72469752)

[Введение 5](#_Toc72469753)

[1 Анализ и моделирование предметной области, формирование функциональных требований к проектируемому мобильному приложению 6](#_Toc72469754)

[1.1 Исследование предметной области, выявление основных проблем 6](#_Toc72469755)

[1.2 Разработка функциональных требований 6](#_Toc72469756)

[1.3 Технические требования 7](#_Toc72469757)

[1.4 Итоги раздела 7](#_Toc72469758)

[2 Проектирование приложения 8](#_Toc72469759)

[2.1 Архитектура 8](#_Toc72469760)

[2.2 Сущности в базе данных и связи между ними 8](#_Toc72469761)

[2.3 Серверная часть приложения 9](#_Toc72469762)

[2.3.1 Уровни. API, Service, Repository 10](#_Toc72469763)

[2.3.2 Объекты, использующиеся на различных уровнях, и их особенности 10](#_Toc72469764)

[2.3.3 Безопасность. Аутентификация и авторизация 11](#_Toc72469765)

[2.3.4 Хранение фото 11](#_Toc72469766)

[2.3.5 Разработка документации 12](#_Toc72469767)

[2.4 Клиентская часть приложения 12](#_Toc72469768)

[2.4.1 Виджеты, управление состоянием виджетов, декларативный UI, фрагменты и диалоги, навигация 13](#_Toc72469769)

[2.4.2 Внедрение зависимостей 14](#_Toc72469770)

[2.4.3 Логика использования приложения 14](#_Toc72469771)

[2.4.4 Дизайн и расположение элементов 14](#_Toc72469772)

[2.5 Веб-приложение 16](#_Toc72469773)

[2.6 Итоги раздела 16](#_Toc72469774)

[3 Реализация приложения 17](#_Toc72469775)

[3.1 Серверная часть 17](#_Toc72469776)

[3.1.1 Конфигурация 17](#_Toc72469777)

[3.1.2 Реализация REST API. Контроллеры 17](#_Toc72469778)

[3.1.3 Сервисы. 18](#_Toc72469779)

[3.1.4 Репозитории. Взаимодействие с БД 18](#_Toc72469780)

[3.1.5 DTO. Конвертация разных типов объектов 18](#_Toc72469781)

[3.1.6 JWT. Аутентификация и авторизация 19](#_Toc72469782)

[3.2 Клиентская часть 20](#_Toc72469783)

[3.2.1 Виджеты, разметка, диалоги. 20](#_Toc72469784)

[3.2.2 Внедрение зависимостей 21](#_Toc72469785)

[3.3 Итоги раздела 21](#_Toc72469786)

[4 Тестирование, проверка работоспособности и анализ полученных результатов 22](#_Toc72469787)

[5 Руководство по использованию 23](#_Toc72469788)

[Заключение 29](#_Toc72469789)

[Список литературы 30](#_Toc72469790)

[Приложение А 31](#_Toc72469791)

[Приложение Б 34](#_Toc72469792)

Введение

Цель данного курсового проекта – разработка мобильного приложения салона оптики, сервера для него, а также веб-клиента.

Задачи курсового проекта:

1. Изучение программных средств, аналогичных теме курсового проекта

Данная задача позволит изучить аналоги данному программному средству, разобраться в их минусах, плюсах. Найти отличия между аналогами и данным программным средством.

1. Проектирование приложения

Проектирование приложения даёт возможность увидеть предварительную концепцию приложения, позволяет спланировать разработку программного средства и соответствующе распределить ресурсы и время, которые будут потрачены на разработку приложения.

1. Реализация программного средства

В курсовом проекте будет реализовано приложение для салона оптики. Данная тема актуальна в век информационных технологий, где люди часто делают выбор в пользу мобильных приложений для онлайн покупок, так как у каждого есть компьютер или смартфон с доступом в Интернет.

1. Тестирование разработанного продукта

Важный этап разработки, позволяющий проверить функциональность приложения и его реакцию на исключительные ситуации, например, ввод данных неподходящего формата, неверные данные авторизации и др.

1. Разработка руководства пользователя

Данная задача подразумевает разработку руководства пользователя, призванное помочь пользователю разобраться с работой приложения и его основным функционалом. Это является важной частью разработки, поскольку определённые пользователи заинтересованы перед использованием ознакомиться с руководством пользователя, чтобы изучить функционал приложения.

После прохождения всех перечисленных этапов, должно получиться полностью рабочее и отлаженное приложение.

1 Анализ и моделирование предметной области, формирование функциональных требований к проектируемому мобильному приложению

1.1 Исследование предметной области, выявление основных проблем

В современном мире мобильные приложения для совершения онлайн покупок занимают значительную часть рынка. Людям гораздо проще ознакомиться с каталогом товаров и узнать информацию об их наличии, не выходя из дома, тем более у каждого под рукой есть компьютер или смартфон, имеющие доступ в сеть Интернет. За счет онлайн магазинов увеличивается клиентская база, состоящая преимущественно из молодежи. Причем популярностью пользуются не только приложения, в которых собраны различные категории товаров. Многие приложения нацелены на определённую сферу, одной из которых является медицинская сфера.

Изучив предлагаемые сервисом Google Play мобильные приложения на данную тематику, я заметил, что на рынке очень мало приложений, предоставляемых салонами оптик. Большинство из них находятся на российском рынке. Что касается Беларуси, то аналогов на рынке найдено не было.

Как отмечалось ранее, главным плюсом данного мобильного приложения является то, что воспользоваться им сможет любой желающий, ведь для этого понадобиться только смартфон с доступом в Интернет, имеющийся практически у любого человека. Таким образом, взяв во внимание текущее положение дел, было принято решение разработать подобное приложение, главной целью которого будет возможность просмотра каталога и новостей, а также запись на консультацию специалиста в определенный салон оптики.

1.2 Разработка функциональных требований

После того как выяснены основные проблемы в области и определён фундаментальный функционал, необходимо его конкретизировать и преобразовать в функциональные требования, а также разработать дополнительные функции для того, чтобы улучшить пользовательский опыт.

В ходе использования приложения, пользователей можно разделить на две роли: сами клиенты, имеющие доступ к просмотру каталога и новостей, а также записи на консультацию и администраторы, которые могут добавлять, изменять и удалять каталог товаров.

Помимо самого приложения для смартфона, необходимо сделать сервер, который будет ответственен за реализацию всей логики, поддержку аутентификации и авторизации и взаимодействию с базой данных.

В итоге был сформирован следующий функционал:

* просмотр каталога товаров и новостей;
* добавление продуктов в каталог;
* добавление новостей;
* запись на консультацию в определенный салон;
* добавление товаров и новостей в избранное;
* возможность изменения общей информации о товаре (название, брэнд, модель, описание, фото и т.д.);
* возможность изменения общей информации о новости (заголовок, описание, изображение);
* возможность удаления товаров и новостей;
* поиск и фильтр товаров;
* наличие нескольких ролей: «пользователь», «администратор».

1.3 Технические требования

Практически все функции мобильного приложения требуют взаимодействия с сервером, который в свою очередь взаимодействует с базой данных, следовательно, для работы приложения требуется стабильное подключение к интернету. От работоспособности перечисленных сервисов целиком зависит работоспособность самого приложения.

Для клиента мобильного приложения, с учётом того, что одной из основных целей является привлечение большего количества пользователей, выбраны системы Android и iOS. Для разработки приложения был выбран фреймворк Flutter от Google. Его главным преимуществом является возможность кроссплатформенно разработки. Нам не нужно разрабатывать нативные приложения под разные операционные системы на разных языках программирования, достаточно знать один Dart. Несмотря на то, что данный фреймворк очень молодой, для него имеется большое количество готовых библиотек, книг, статей и всевозможных инструкций, что, без сомнений, облегчает процесс разработки приложения. Операционная система Android имеет множество версий, чем ниже версия системы, под которую будет разрабатываться приложение, тем больше устройств смогут её запустить, но тем меньше новых функций будет доступно при разработке. Таким образом, оптимальным выбором оказалось версия под номером 26.

Серверная часть приложения будет разработана с использованием фреймворка NestJS, а язык, использующийся при разработке – TypeScript, он же понадобится и для запуска сервера. Этот фреймворк был выбран в силу того, что он является очень прогрессивным и удобным для обеспечения работы серверной части, а также имеющим много коммерческих проектов, написанных с его использованием. Также вместе с сервером будет работать СУБД PostgreSQL 13.

Как мобильное приложение, так и сервер не требуют много ресурсов для своей работы, поэтому ставить вопрос о комплектующих, необходимых для работы приложения и сервера, не имеет смысла.

1.4 Итоги раздела

По итогам данного раздела была проанализирована предметная область, в ней выявлены ключевые проблемы, которые необходимо устранить, и на основе этого был сформирован функционал программы, а также определены базовые технологии для его реализации.

2 Проектирование приложения

Проектирование приложения – самая важная часть разработки, ведь именно на этапе проектирования выявляются как мелкие огрехи, так и крупные недоработки, исправление которых на этапе его развёртывания будет крайне затруднительно и накладно. Важность проектирования в данной курсовой работе очень высока ещё и потому, что разрабатываться будут три компонента: мобильное приложение, серверная часть, а также веб-приложение. Необходимо продумать то, как они будут без помех взаимодействовать друг с другом.

2.1 Архитектура

Для разработки подобного типа проекта было решено использовать стандартную трёхуровневую архитектуру, так как такая архитектура позволяет добиться низкой связности уровней, что, в свою очередь, обеспечивает для приложения надёжность, простоту долгосрочной поддержки, а также оперативное устранение неисправностей при необходимости.

Клиент представляет из себя либо мобильное, либо веб-приложение с пользовательским интерфейсом, которое будет совершать REST-запросы на сервер и обрабатывать их в читаемый для пользователя вид. Также на клиенте реализовывается логика обработки полученных от сервера сообщений об изменении таблицы лидеров и местоположении участников, отображении информации на карте.

Сервер будет выполнять мероприятия по поддержке аутентификации и авторизации, а также будет ответственным за всё взаимодействие с базой данных.

2.2 Сущности в базе данных и связи между ними

Изучение предметной области и составленный впоследствии функционал показали, что можно выделить 9 таблиц, которые позволяют полностью охватить обозначенный функционал (рис. 2.1). Из них 6 основных:

* пользователь (User) – таблица, предназначенная для хранения основных данных о пользователе, таких как имя, телефон, электронная почта, пароль;
* новости (News) – таблица, предназначенная для хранения новостей;
* продукт (Product) – таблица, предназначенная для хранения данных о продукте;
* консультация (Consultation) – в этой таблице хранятся данные о консультациях;
* салоны (Salon) – таблица, предназначенная для хранения информации о салонах оптики;

роль (Role) – таблица нужна для того, чтобы хранить данные о существующих ролях;

Остальные таблицы вспомогательные, и нужны для того, чтобы поддерживать отношения типа «многие-ко-многим».

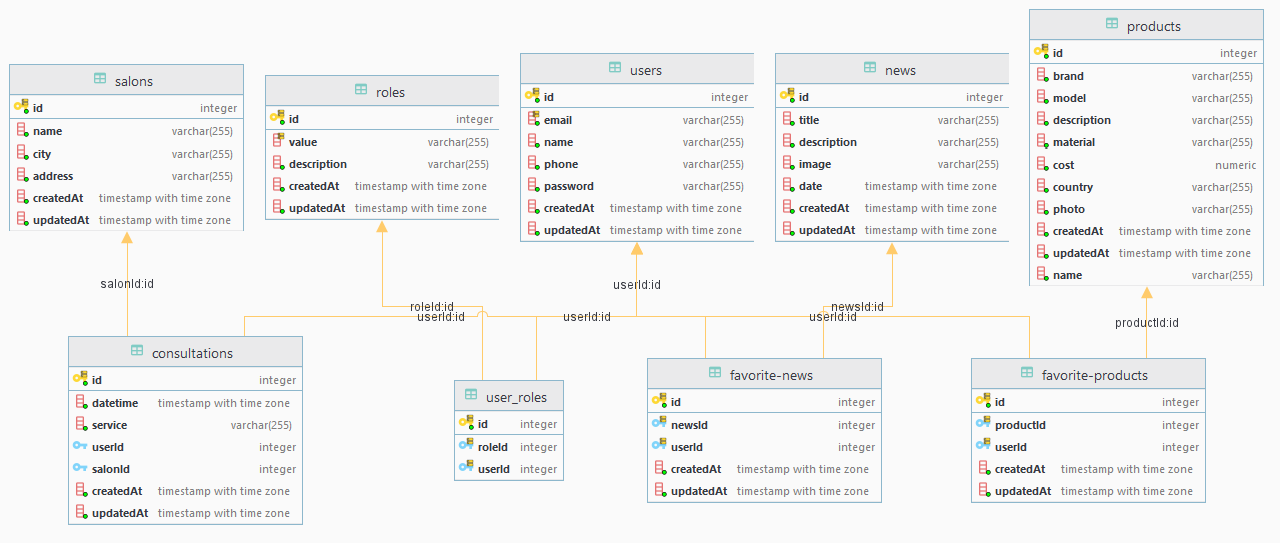


Рисунок 2.1 – Схема базы данных

Отдельно стоит упомянуть способ хранения картинок в базе данных. Там их хранение осуществляется в виде относительных ссылок на файловую систему внутри сервера, либо абсолютных ссылок на Интернет-ресурс, откуда её будет необходимо качать для просмотра.

2.3 Серверная часть приложения

При проектировании классов и интерфейсов для серверной части приложения необходимо стараться держать низкий уровень избыточности и связности между составляющими частями, а также усиленно следить за корректностью кода, поскольку от этого будет зависеть работа всех подключенных клиентов. Добиться всего этого можно с помощью грамотного распределения логики сервера по разным уровням, определяя формат объектов, которые будут использоваться на каждом уровне, и преобразования объектов при передаче с уровня на уровень, а также используя подходящие паттерны проектирования.

Для того, чтобы обмениваться данными между клиентами и сервисами был выбран архитектурный стиль REST. По своему определению он широко использует возможности протокола HTTP и поэтому не имеет состояния (stateless). Это современный подход, использующийся в большинстве веб-сервисов и поэтому имеющий множество примеров реализаций различных функций на его основе, таких как аутентификация, выполнение CRUD-операций и т.д. В REST-приложениях в подавляющем большинстве случаев используется JSON для передачи данных, его также было решено взять как формат данных при передаче между клиентами и сервером.

2.3.1 Уровни. API, Service, Repository

Приняв во внимание вышенаписанное, лучше всего разбить серверную логику на следующие уровни API, Service и Repository.

API-уровень. Это самый верхний уровень сервера, и на нём будут располагаться классы-контроллеры, методы которых будут описывать URL-адреса, на которые можно будет сделать вызов, и их характеристику. Также эти методы будут преобразовывать данные из формата, пригодного для сериализации, в формат, пригодный для сохранения данных в БД и работы с ними как с бизнес-объектами.

Service-уровень. На этом уровне находятся классы, ответственные за бизнес-логику, среди них как те, которые просто делегируют вызов к репозиторию, так и те, которые выполняют более сложную логику.

Repository-уровень. Он содержит в себе классы-репозитории, использующиеся, когда необходимо совершить какие-либо операции с базой данных. Методы этих классов могут обращаться к БД как с помощью автоматически сгенерированных SQL-запросов, так и с помощью написанных непосредственно программистом. Стоит упомянуть, что создание данных классов в NestJS не требуется, так как для нас эти классы предоставляет пакет sequelize.

2.3.2 Объекты, использующиеся на различных уровнях, и их особенности

Использование различных типов объектов для предоставления данных является хорошей практикой при разработке любых типов программных продуктов, а при наличии разных уровней в пределах одного приложения, без такого подхода обойтись попросту невозможно. Для этого использовались два паттерна одновременно: DTO-объект и Immutable-объект. Следуя первому паттерну, мы будем проектировать классы, содержащие только базовые типы, либо те типы, которые легко сериализуются. Его преимущество фактически уже было названо: понятная и предсказуемая сериализация, то есть разработчик всегда будет знать, как экземпляр класса будет выглядеть после превращения его в JSON-строку. Второй паттерн было решено применить по причине как раз-таки тех ограничений, что накладывает первый паттерн. DTO-объекты содержат в себе данные исключительно для передачи, более с ними никакие операции совершать не рекомендуется. Вывод о том, что такой объект должен быть неизменяемым, очевиден, более того, по мере роста проекта это поможет защитить программиста от дальнейших ошибок.

Второй тип объекта, которые будет использоваться на сервере – Entity-объект. Его задачей служит хранение данных, которые либо в дальнейшем будут записаны в БД, либо, наоборот, только что оттуда скачаны. Такой тип объектов наряду с классами-репозиториями является частью ORM-системы. Он также хорошо подходит для использования в тех местах, где выполняется бизнес логика. С этим типом объектов будут в той или иной степени работать все уровни сервера. Уровень API будет связан с ним только тем, что будет преобразовывать в этот тип DTO-объект, чтобы потом отдать его Service-уровню, например, при получении POST-запроса, и наоборот, преобразовывать из Entity в DTO, при запросе данных. Service-уровень будет использовать его как полноценный объект, хранящий в себе все необходимые данные для работы бизнес логики. На этом уровне объекты будут подвергаться любым изменениям, а затем передаваться в Repository-уровень.

Всегда, когда речь идёт о различных объектах, по-разному предоставляющим одну и ту же информацию, и их взаимодействии, необходимо обозначить то, как будет происходить конвертация. Традиционно для это используются специальные классы, называемые «мапперами», содержащие в себе логику конвертации из одного типа объекта в другой. Так как задача крайне тривиальная, существует уже достаточно большой выбор библиотек, которые позволяют в той или иной мере автоматизировать и упростить этот процесс.

2.3.3 Безопасность. Аутентификация и авторизация

Важнейшим аспектом проектирования серверной части приложения является то, как пользователи будут входить в систему и как система будет понимать, что очередной присланный запрос прислан уже находящимся в системе пользователем.

Стандартными подходами при реализации этих функций является либо аутентификация с помощью логина и пароля, при этом используя собственную БД в качестве их места хранения, либо использование сторонних сервисов, предоставляющих своё API для того, чтобы можно было осуществить вход. Это может быть API на основе протокола OAuth, либо какие-то собственные разработки. При рассмотрении данного вопроса лучше всего обратить внимание на то, какими будут приложение-клиенты, то, в каком окружении, на каких системах они будут запускаться. В нашем случае это приложение под Android и веб-клиент. В связи с этим было принято решение о авторизации с использованием JWT-токена.

После того, как пользователь совершит вход в приложение, необходимо поддерживать его статус и позволять выполнять запросы к API, в то время как запросы не аутентифицированных пользователей – отклонять. Как было описано ранее, серверная часть работает без сохранения состояния по протоколу HTTP, в таком случае достаточно распространённой практикой является использование JWT-токена. Он идеально вписывается в описанные методы взаимодействия с клиентами и поможет поддержать требуемый уровень безопасности.

2.3.4 Хранение фото

Реализовать функционал хранения фотографий можно как и вручную, храня их на собственном сервере, так и воспользовавшись уже готовым решением в облаке, такими как AWS S3, Azure Storage. В рамках работы над данным курсовым проектом, был выбран первый вариант, так как он предполагает наименьшие трудозатраты, однако соответствующие классы на серверной части приложения сделаны таким образом, что хранилище фотографий можно будет при необходимости легко сменить с локального на удалённое либо облачное.

2.3.5 Разработка документации

Для документации был использован Swagger – это технология, которая позволяет документировать REST-сервисы. Она представляет интерфейc между клиентскими приложениями (Android/iOS и Веб-клиент), и сервером (рис. 2.2).



Рисунок 2.2 – Swagger схема

2.4 Клиентская часть приложения

Клиентская часть приложения не менее важна, чем серверная. Это то, с чем будет контактировать пользователь, то, от чего в первую очередь зависит будут ли впечатления от использования положительными или отрицательными. Здесь стоит понимать, исходя из специфики проекта, в первую очередь разрабатывается мобильное приложение и самые строгие требования со стороны пользователя будут предъявляться именно к нему.

Точно так же, как и серверная часть приложения, как и практически любой другой проект, который имеет риск сильно разрастись в будущем, лучше всего разбить логику клиентской части приложения на части таким образом, чтобы на этой основе можно было каждую часть реализовывать последовательно и независимо от других частей. Приняв во внимание все требования к функционалу, а также лучшие практики и паттерны, использующиеся при разработке Android-приложений, логику клиентской части приложения можно разделить на две категории: включающую в себя взаимодействие с пользовательским интерфейсом и ту, которая связана с бизнес-логикой.

В первую категорию будут входить:

* виджеты и разметка экранов;
* виджеты и разметка диалогов;
* адаптеры списков;
* стили, иконки.

Во вторую:

* классы, ответственные за внедрение зависимостей;
* функции, отправляющие запросы к API;
* прочие дополнительные и вспомогательные классы.

Понятно, что иерархически классы из категории с пользовательским интерфейсом стоят ниже, чем классы из категории с бизнес-логикой, и именно они будут пользоваться классами из второй категории.

2.4.1 Виджеты, управление состоянием виджетов, декларативный UI, фрагменты и диалоги, навигация

Так как было принято решение разрабатывать приложение на Flutter, были изучены различные способы управления состояниями виджетов, так как все объекты во флаттере являются виджетами, которые изменяют свое состояние и перерисовываются во время работы приложения. В больших проектах есть смысл в использовании такого паттерна проектирования как BLoC, который является посредником между уровнем представления и базой данных. В данном приложении было принято решение использовать метод setState() для контроля состояний.

Виджетами является всё то, что связано с интерфейсом: код, реагирующий на нажатия кнопок и на остальные события, код инициализации состояния, очистки состояния, валидация. Они по большей части выполняют работу по отображению пользователю какого-либо динамического контента, путём взаимодействия с соответствующей ему моделью, а также в ответ на действия пользователя. Важно отметить, что приложение будет построено целиком с использование Stateless и Statefull виджетах. Это стандартная практика при разработке приложений на Flutter, так как такая архитектура позволяет легко использовать одни и те же виджеты по нескольку раз, появляется возможность применять продвинутые библиотеки с навигацией и манипулировать их жизненным циклом.

Разметка в разработке на Flutter представляет собой декларативный UI, в которых с помощью классов виджетов описываются различные элементы интерфейса. Все сложные виджеты строятся на основе простых путем наследования одного от другого.

Навигация – очень важный для мобильного приложения элемент, при её проектировании необходимо поддерживать логическую связь между виджетами, давать возможность перейти на какой-либо виджет из одних виджетов и не позволять это делать из других. Необходимый список виджетов будет таким:

* с главным экраном, который будет содержать навигационное меню;
* со списком новостей;
* со списком товаров;
* с добавлением и редактированием новости;
* с добавлением и редактированием товара;
* с записью на консультацию;
* с профилем;
* с детальным описанием продукта;
* и некоторое количество диалогов, которые, по сути, тоже являются виджетами и участвуют в навигации наравне с ними.

Для навигации будет использоваться Navigator – это стандартный класс, предназначенная для того, чтобы упростить переходы между виджетами. С её помощью можно будет также строить граф переходов, где будут отображены стрелками направления переходов.

2.4.2 Внедрение зависимостей

Одной из самых распространённых практик при разработке мобильных приложений – это использование различных библиотек, внедряющихся виде зависимостей в pubsec файл. Для Flutter существует множество библиотек, предоставляющих различный функционал.

Во время разработки приложения были использованы такие библиотеки как:

* flutter\_http и dio, необходимые для выполнения запросов от приложения к серверу;
* shared\_preferences, для хранения JWT токена;

2.4.3 Логика использования приложения

Как уже было написано ранее, типов пользователей всего два: пользователь и администратор. Как только регистрируется новый клиент, ему даются права обычного пользователя. Имея такие права, клиент может просматривать новости и продукты, записываться на консультацию в салон и просматривать профиль.

Пользователь заходит в приложение, изучив продукты и новости он может добавить понравившееся в избранное, которое помечается закрашенной иконкой. При записи на консультацию у пользователя есть возможность выбрать один из доступных салонов, время консультации и услугу.

Администратор в свою очередь имеет возможность добавлять и изменять информацию, а также просматривать список консультации в веб-приложении.

2.4.4 Дизайн и расположение элементов

В виду своей кроссплатформенности, во Flutter присутствуют элементы Material Design для устройств, работающих по ОС Android и Cupertino для IOS, так как все приложения в App Store должны иметь схожий дизайн и не путать пользователей.

Приложение имеет в себе 4 главных экрана, являющиеся отправными точками для переходов на все остальные виджеты: экран с новостями, продуктами, записью на консультацию и с профилем. Для того, чтобы перемещаться между ними, используется ряд с кнопками в нижней части экрана, сделано это по той причине, что на рынке сейчас доминируют телефоны с большой диагональю, и дотянуться до нижней части экрана всегда проще всего. Перемещение по фрагментам назад осуществляется либо с помощью аппаратной кнопки назад, либо с помощью его программного аналога в левом верхнем углу.

При выборе цветовой гаммы приложения и вида интерфейса важно помнить для какой аудитории делается приложения и какие цели те или иные цвета и геометрические формы преследуют. Основной цвет приложения – зелёный, выбран он был потому, что в теории дизайна он является символом безопасности, а это именно то, что хотят чувствовать люди обращаясь в медицинские учреждения.

В приложении используются различного вида кнопки, как имеющие заливку, так и нет. Первый тип используется для тех кнопок, которые делают какие-то существенные действия: добавление продуктов, новостей, выполняют вход в приложение и т.д. Второй же тип используется для менее важных действий, например, добавления товара в избранное или переход на новый экран. Такое разделение позволяет заострить внимание пользователя на важных действиях и не распылять его внимание на менее существенные.

Наличие виджетов, которые содержат в себе несколько типов полей ввода, а также имею несколько способов предоставлять пользователю информации, предоставляет дополнительные сложности при проектировании их дизайна, и при работе над ними нужно быть особенно внимательным, их нужно составлять таким образом, чтобы уместить достаточно количество информации на экран мобильного телефона, при этом не запутав пользователя. Для решения такой проблемы чаще всего используется группировка элементов, она может осуществляться, например, с помощью использования разделителей. Следуя последним трендам в дизайне, решено было использовать разделение контента на своеобразные «карточки» с закруглёнными углами. При этом стоит учитывать, что более натурально будет выглядеть, если карточка сверху будет иметь закруглённые углы только внизу, карточка посередине – и внизу, и вверху, а карточка внизу – только вверху. Цвет для основного заднего фона белый, а для заднего фона карточек зеленый, что позволяет их выделять на общем фоне.

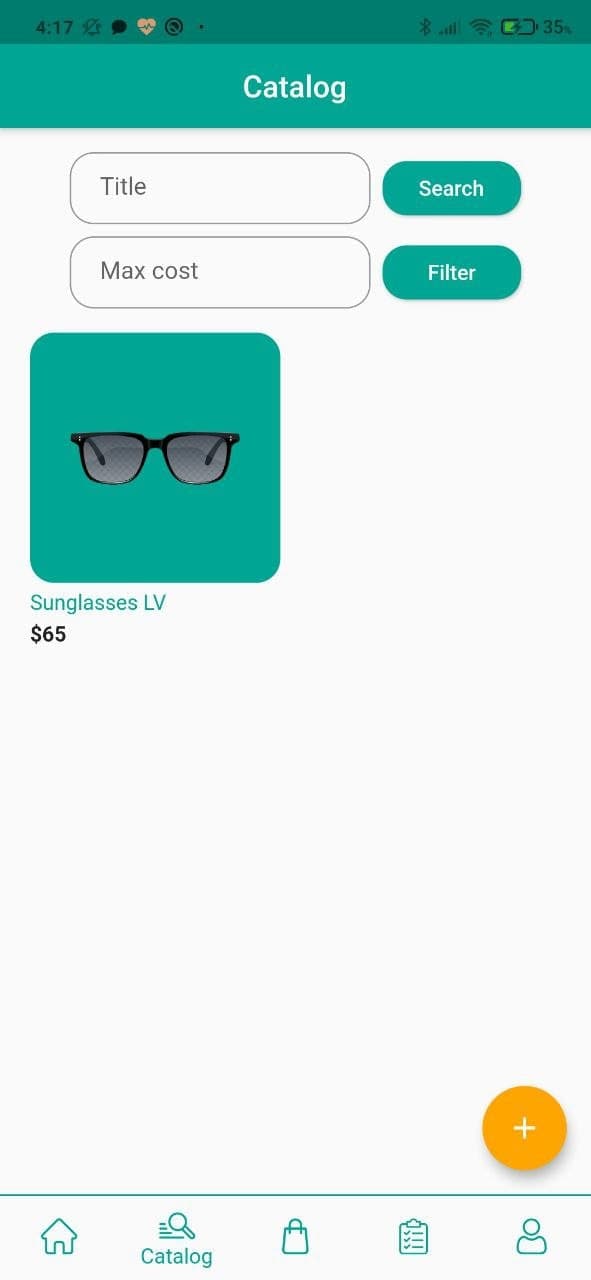
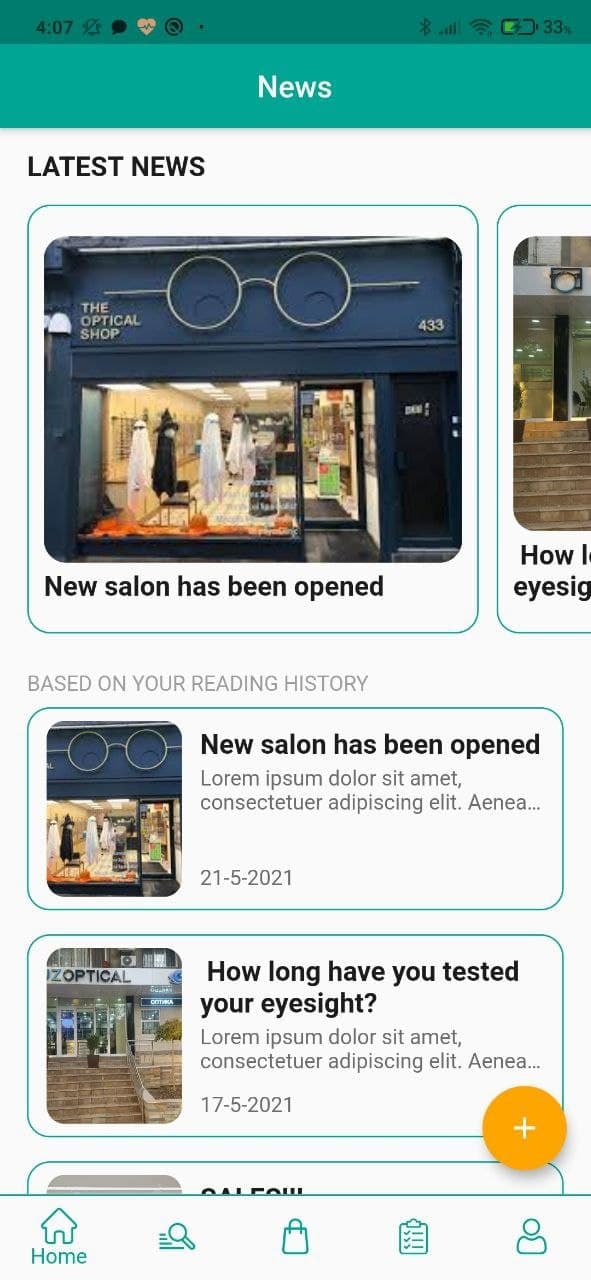
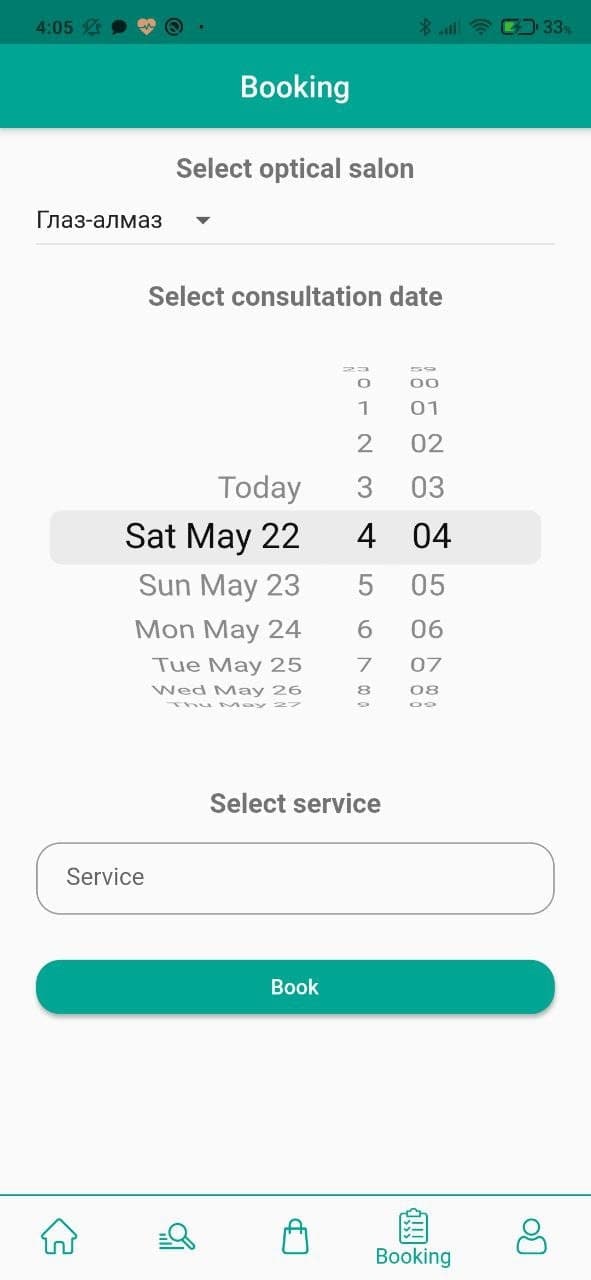


Рисунок 2.3 – Три основных экрана

2.5 Веб-приложение

В данной курсовой работе веб-приложение занимает самую малую часть. Оно целиком предназначено для пользователя с ролью «Администратор» и будет представлять собой страницу со входом, а после успешного входа – таблицу со списком салонов, где у него будет возможность менять их некоторые данные. Для реализации веб-приложения будет использоваться Vanilla JS.

2.6 Итоги раздела

По итогу данного раздела, у нас имеется детально проработанный план разработки приложения, как серверной и клиентской части, так и веб-приложения. Проработана внутренняя архитектура самих частей приложения, проработаны вопросы того, какого вида классы там будут использоваться, а также вопросы связи всех частей между собой и вопросы, связанные с безопасностью. Проект готов к реализации.

3 Реализация приложения

3.1 Серверная часть

Для разработки серверной части приложения использовался язык программирования NodeJS, с использованием фреймворка NestJS, который включает в себя язык TypeScript.

3.1.1 Конфигурация

Конфигурация сервера осуществляется за счет содержания в нем .env файлов, которые в зависимости от переданной переменной среды NODE\_ENV выбираются с помощью npm-пакета cross-env. Env файл отвечает за то, в каком режиме будет работать сервер: production или development. Так же там хранятся логин, пароль и адрес сервера, на котором расположена БД, а также ключи.

В файле tsconfig.json описана настройка Typescript.

Файл nest-cli.json отвечает за настройку nest-cli.

Файлы eslint и prettier отвечают за качество кода.

В классе main.ts содержатся метод, настраивающий сервер. В нем создается объект сервера подключается конфигурация Swagger и устанавливаются глобальные middleware и запускается сервер на порту хранящимся в env файле. Помимо всего прочего, там настраивается CORS для корректного доступа с веб-клиента.

Отдельно стоит уделить внимание форматам даты и времени, которые будут использоваться при сериализации. Это важный момент, так как очевидно, что тип, представляющий время и дату, является сложным, а значит, что на разных платформах, в разных языках программирования, этот тип может быть интерпретирован совершенно по-разному, поэтому возникает необходимость введения определённого единого стиля. Таковым был выбран стандарт ISO 8601, так как он является самым распространённым и имеет поддержку во всех используемых языках программирования. Он выглядит так: «YYYY-MM-DDThh:mm:ss».

В NestJS огромную роль играет настройка классов модулей. Они определяют контроллеры, которые используются в этом модуле, сервисы, которые используются в самом модуле, а также сервисы, которые предоставляет сам этот модуль при его импорте, модели, модули.

3.1.2 Реализация REST API. Контроллеры

Классы-контроллеры являются представлением API. У них есть методы, помеченные специальными декораторами, которые вызываются, когда на сервер поступает HTTP-запрос. Есть условный шаблон, по которому описывается каждый контроллер. Указывается декоратор над классом контроллера, характеризующая его. В нашем случае, это будет та, которая характеризует его как REST-контроллер. Далее внутри класса идут методы, каждый из которых помечен аннотацией, которая показывает, какой тип HTTP-метода соответствует этому методу класса, какой тип данных он принимает и какой возвращает. Благодаря NestJS, нет необходимости в том, чтобы осуществлять самостоятельную валидацию, данных, приходящих в виде JSON. Достаточно указать в аргументах метода, класс DTO в котором описать с помощью декораторов ограничения на входные данные. NestJS с помощью глобального middleware ValidationPipe проверит входные данные и в случае если входные данные окажутся не валидными отправит ответ со статусом ошибки 400 и пояснительным текстом. В контроллеры с помощью механизма Dependency Injection встраиваются сервисы, которые в нем используются.

3.1.3 Сервисы.

Сервисы – классы, которым контроллеры делегируют выполнение различных действий, после поступления на них запроса. Они также помечаются своим особым декоратором, но скорее для того, чтобы фреймворк смог создать по ним объекты и включить их в свой пул для последующего внедрения зависимостей.

Как было уже описано ранее, большинство сервисов лишь выполняют различные действия с репозиториями, иногда лишь делая небольшие преобразования в объектах, чтобы ORM-система смогла правильно их обработать. Работа сервисов совершается в неблокирующем асинхронном режиме. В конструктор сервиса так же встраиваются различные модели.

3.1.4 Репозитории. Взаимодействие с БД

Для того, чтобы взаимодействовать с базой данных, используется ORM sequelize-typescript. Для этого создаются Модели – классы, представляющие собой таблицы в базе данных, у которых поля соответствуют колонкам. Эти классы принимают на вход методы репозиториев. Репозиторий в нашем случае – это сами классы моделей, которые позволяют работать с ними как с репозиториями.

Каждый класс модели помечается декоратором Table в котором можно указать название таблицы, которой он будет соответствовать.

Описание полей класса также сопровождается декораторами в которых указывается: тип данных, ограничение целостности, ограничения типа данных. Также можно описать декораторами схему для документации.

3.1.5 DTO. Конвертация разных типов объектов

DTO – важнейшая часть API-системы, так как эти объекты представляют то, как должны будут выглядеть запросы и ответы к ней. Легко понять тот факт, что если запросы будут по большей части одинаковыми, то ответы могут отличаться как по своему наполнению, так и по структуре. Например, если приходит запрос на получение одного пользователя, то не имеет смысла отправлять вместе с ним полную информацию о всех консультациях, в которых он поучаствовал, так как большинство из отправленных данных не будет использоваться получателем и канал передачи будет впустую нагружаться. Также будут возникать проблемы как с многочисленными зависимостям, если они будут тянуть друг друга, так и с циклическими зависимостям, которые будут вызывать переполнение стека. Одним из решений перечисленных проблем является определение класса для каждого из типов запросов. Главным его недостатком является то, что придётся писать достаточно большое количество мало чем отличающегося кода. Тем не менее, преимуществ такой подход несёт больше, поэтому был выбран именно он. Для реализации этого подхода использовались:

* классы для каждого тела запроса;
* классы, которые наследовали либо все поля либо часть из них.

Пример такого класса можно увидеть в приложении Б.

При валидации используется npm-пакет class-validator.

Так же в приложении реализована загрузка тела с Content-Type двух видов multipart/form-data и application/json. Так как при загрузке информации с телом переданным как multipart/form-data вся информация приходит в виде строки, испольщовался пакет class-transorm для преобразования типа строк в тип указынный в DTO с помощью декораторов @Transform

3.1.6 JWT. Аутентификация и авторизация

Как было написано ранее, для нужд аутентификации и авторизации будет использоваться JWT-токен. При аутентификации, он будет формироваться и отправляться аутентифицирующемуся клиенту, затем этот клиент должен будет его отправлять с каждым запросом, а задача сервера – проверять этот токен на валидность.

Для реализации аутентификации использовался пакет passport. Было реализовано несколько стратегий (так в терминологии passport.js называют способ аутентификации).

LocalStrategy – стратегия в которой пользователь аутентифицируется по логину и паролю.

JwtStrategy – стратегия в которой пользователь аутентифицируется по содержащемуся в заголовке Authorization, bearer токен.

Аутентификация пользователя к методам контроллера в NestJs реализована с помощью Guards. В них как раз и реализован функционал из пакета passport.

Для авторизации пользователя использовались так же Guards которые проверяли на наличие необходимой роли в jwt токене, который добавляется после прохождения JwtAuthGuard в запрос как поле user.

Код RolesGuard представлен в приложении Б.

3.2 Клиентская часть

Разработка приложений под Android имеет свою специфику и во многом отличается от разработки для других платформ. Изначально в SDK идут только самые базовые компоненты, которые очень сложно использовать для того, чтобы реализовать требуемый функционал, однако существует большое множество библиотек и фреймворков, предоставляемых в том числе и разработчиками Google.

Самые главные сторонних компонентов, которые использовались, это:

* flutter\_http – библиотека, упрощающая совершение вызовов к API;
* dio – так же библиотека, упрощающая совершение вызовов к API;
* shared\_preferences – библиотека, для хранения пар ключ-значение (используется для хранения токена);
* Future– библиотека для выполнения асинхронного кода;
* Material и Cupertino – библиотеки, предоставляющая компоненты из Material UI и Cupertino.

Языком программирования был выбран Dart.

3.2.1 Виджеты, разметка, диалоги.

Так как в проекте используется DataBinding, то при создании вида каждого фрагмента приходится использовать специальные классы, которые генерируются этим фреймворком. Для этого, а также для того, чтобы сократить количество остального повторяющегося кода, был создан абстрактный класс StandardBoundFragment. Туда, в качестве generic-параметров, посылается класс ViewModel и класс, который обеспечивает привязку данных. Также в качестве аргументов конструктора туда посылаются методы создания класса с привязками, связывания ViewModel, внедрения зависимостей и уничтожения связанного компонента. Далее в нужных местах эти методы вызываются, а также инициализируются поля, которые, как показала практика, будут использоваться почти в каждом фрагменте: поле с классом связывания и поле с ViewModel. Также этот класс предоставляет два метода для реализации: initElements и subscribeUi. Первый нужен для того, чтобы инициализировать UI-компоненты, а второй, чтобы подписаться на обновления полей из ViewModel.

Диалоги, разработанные в данном проекте, тоже являются виджетами. Они используются при вызове функции удаления элемента для того чтобы подтвердить операцию. Диалоговое окно представляет собой объект класса AlertDialog, который включает виджет Text – для отображения текста сообщения и виджеты MaterialButton – для расположения в нес кнопок отмены и подтверждения. При нажатии кнопку подтвердить вызывается функция delete, в теле которой происходит отправка запроса на удаление. После удаления сервер возвращает удаленный объект и при помощи метода setState() происходит перерисовка виджета.

3.2.2 Внедрение зависимостей

При разработке мобильной части приложения повсеместно используется внедрение зависимостей. Помимо того, что оно предоставляет единое место для создания объектов, он также предоставляет возможности взаимодействия с жизненным циклом фрагментов, что крайне полезно в случае, если необходимо иметь ViewModel, использующуюся более чем в одном фрагменте, например, когда создаётся новое соревнование. Классы для внедрения зависимостей расположены в специально созданной для этого директории и могут разделяться на три типа: компоненты, модули и области видимости. Помимо этого, используется собственный класс, который называется ComponentKeeper, он содержит в себе все компоненты, обёрнутые в класс DestroyableComponent. Всё это нужно для того, чтобы можно было управлять жизненным циклом компонентов, уничтожая их в нужный момент (например, при выходе из фрагмента). Области видимости же нужны для того, чтобы Dagger понимал, когда используется новый компонент, а когда тот же самый. Каждому фрагменту соответствует свой компонент, при этом все они объединены в один большой, принадлежащий всему приложению, который создаётся в классе MyApplication, и через который создаются все остальные компоненты.

3.3 Итоги раздела

В итоге была разработана мобильная и серверная часть приложения, а также проверено и отлажено их взаимодействие. Основная сложность была в грамотной реализации поддержки соревновательных сессий, чтобы одни клиенты исправно отправляли координаты на сервер, а другие клиенты их получали, а также в том, чтобы правильно реализовать обработку этих локаций. Также много сложностей возникало с правильным структурированием всех взаимодействующих классов в мобильном приложении, что было, несмотря на это, успешно выполнено.

4 Тестирование, проверка работоспособности и анализ полученных результатов

Тестирование проекта выполнялось в течение разработки посредством таких средств, как отладка и логгирование. После введения нового либо изменения существующего компонента программы, он и зависящий от него код также подвергался тестированию. Анализ полученных результатов происходил по факту получения ошибок и проблемные места сразу исправлялись. После завершения разработки программы по нескольку раз были подвергнуты испытаниям все элементы управления в пользовательском интерфейсе, также они тестировались по ходу написания записки. Если происходили ошибки, то их причина и местоположение в коде легко фиксировались благодаря ведению логов.

После написания приложения также проводились различные тесты, и проверки, такие как проверка на корректное завершение соревнования, проверка на корректную работу таймера, проверка корректного создания соревнования и команды, а также их удаления и изменения.

В итоге приложение стало хорошо отлаженным и готовым к полноценному использованию.

5 Руководство по использованию

Для того, чтобы установить приложение на телефон, Вам необходимо иметь версию Android 7.0 и выше (версию API 22 и более). Версию Вашего телефона можно узнать в настройках.

Если Ваша версия удовлетворяет требованиям, то можно устанавливать приложение на телефон.

После установки приложения на экране появится иконка и подпись “Forums” (рис.5.1)

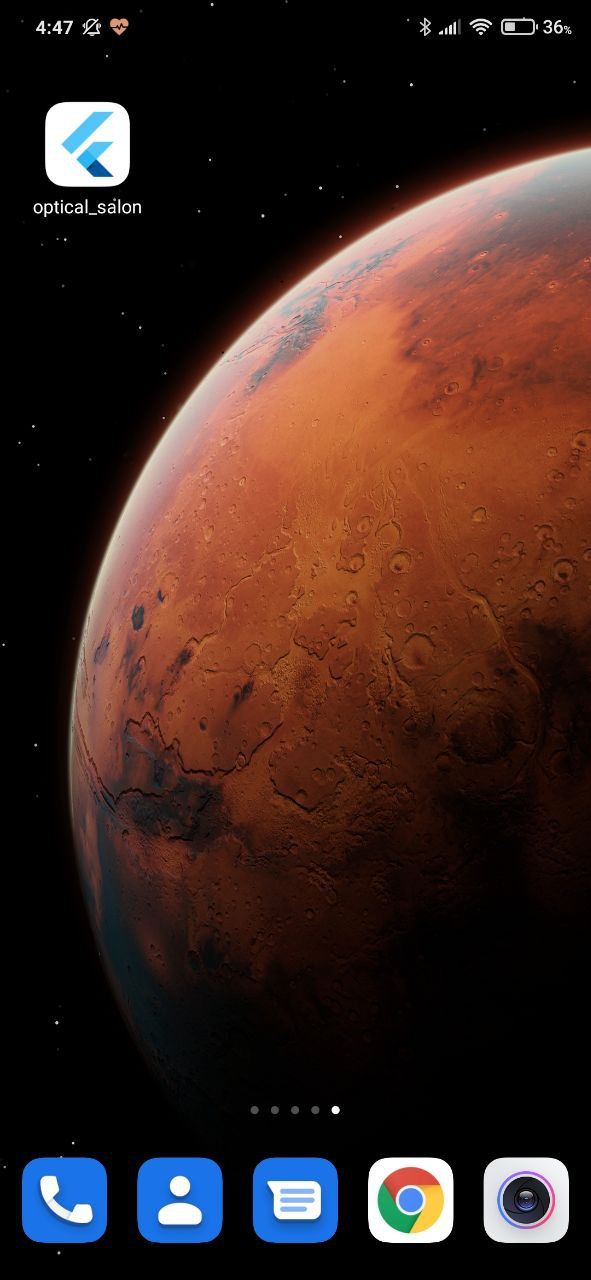


Рис 5.1 Иконка приложения

Изначально открывается экран, на котором находится форма авторизации, которая имеет заголовок, два текстовых поля и две кнопки: регистрации и авторизации(рис.5.2).

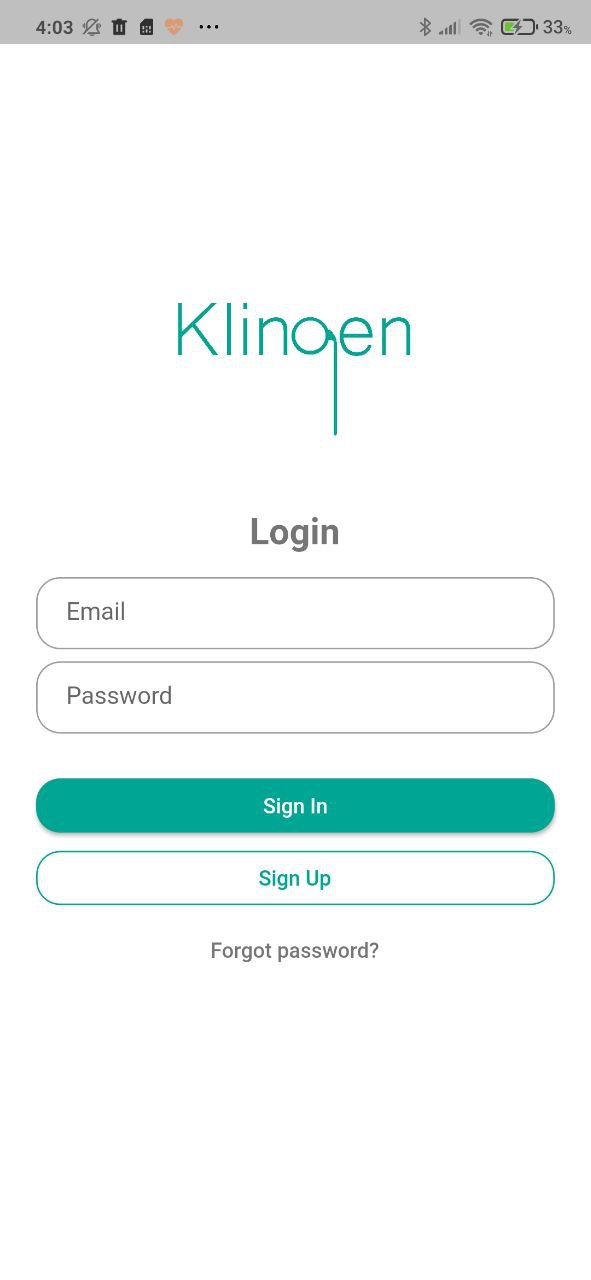


Рисунок 5.2 – Форма авторизации

Перед первым использованием приложения Вам необходимо зарегистрироваться. Для этого нужно заполнить поля логин и пароль валидными данными и нажать на кнопку “Sign Up”.

Вы успешно прошли регистрацию и можете зайти в свой аккаунт. Для этого Вы заполняете email (имя пользователя) и вводите пароль (password) на форме входа, после нажатия клавиши “SING IN” Вы переходите в приложение.

Сразу открывается главный экран, где отображаются новости (рис.5.3). Все новости представляют собой список, где каждая новость имеет свое название, дату создания, изображение и описание. Добавлять новые новости и удалять их может только администратор.

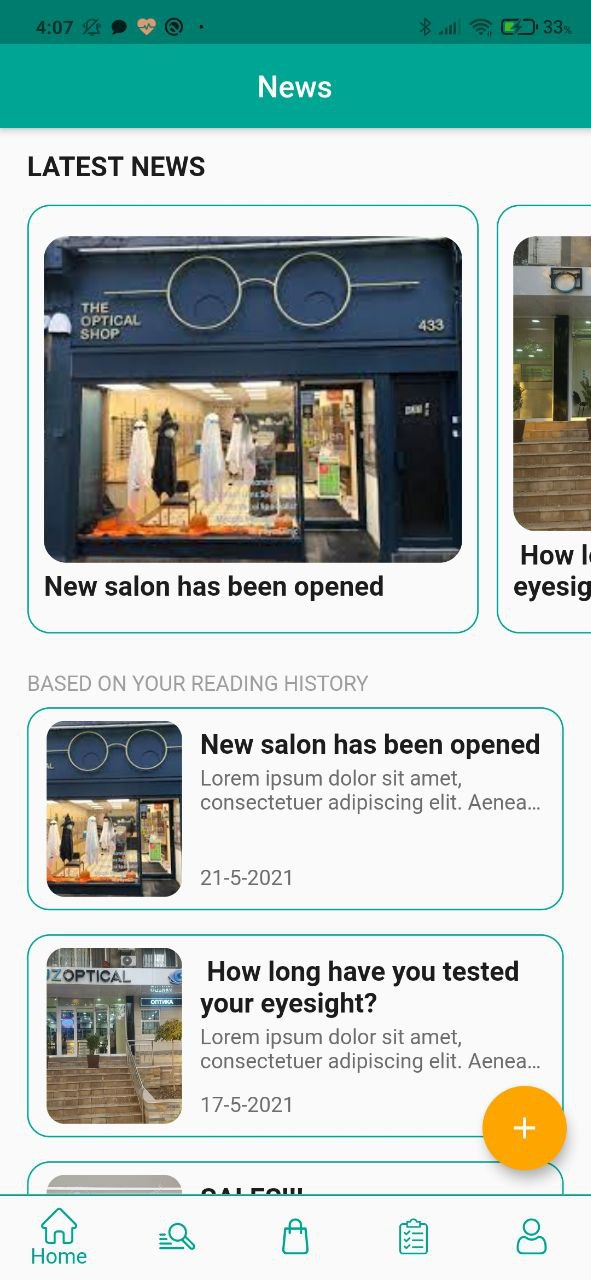


Рисунок 5.3 – Список новостей

Для добавления нового форума Вам необходимо нажать на кнопку “Upload”. После нажатия Вы перейдете на страницу создания форума, где нужно будет заполнить три поля “Title”, “Image”, “Description”. Все поля являются обязательными, и в случае не валидных пользователю будет выдано предупреждение. Это позволит избежать ошибок пользователя при отправке данных на сервер. (рис. 5.4)

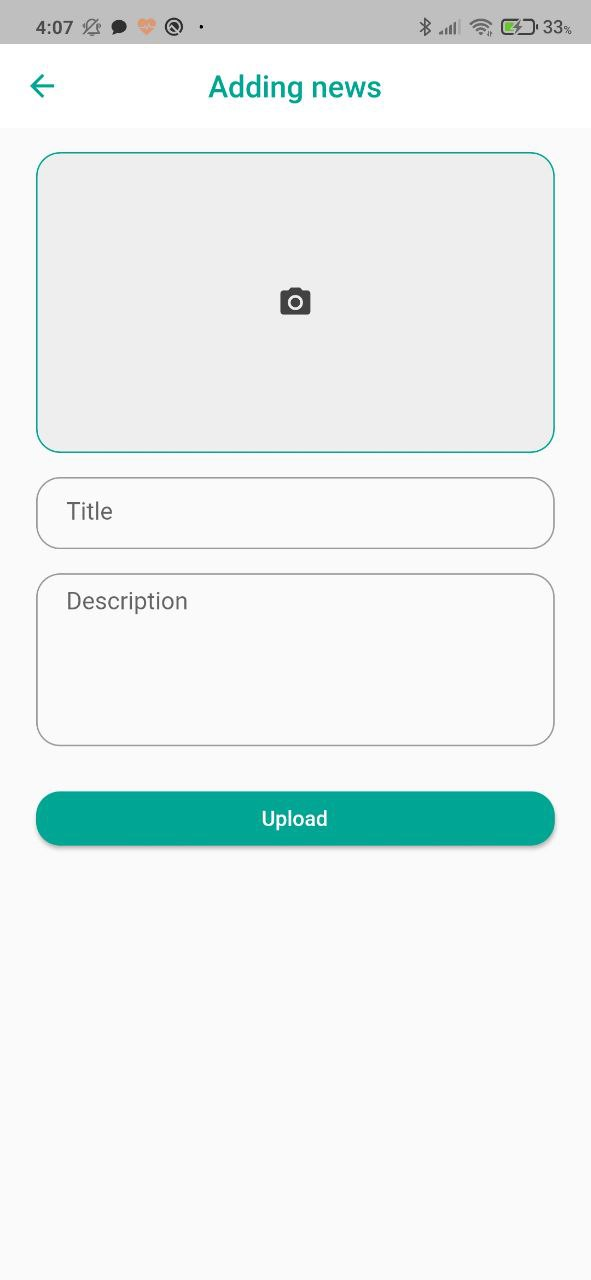


Рисунок 5.4 – Добавление новости

После нажатия кнопки загрузки, будет совершен переход на главную страницу с новостями. Все элементы списка оформлены в виде карточек и при нажатии на определенный элемент списка будет открыт экран с подробным описанием продукта. Также на данном экране можно осуществить операции изменения и удаления новости, а также добавить новость в избранное кликнув на иконку сердечка и удалить из избранного повторным нажатием. Перед удалением появляется диалоговое окно с подтверждением удаления, что так же позволит избежать ошибки пользователя. Операции изменения происходит таким же образом как и добавление. (рис. 5.5).

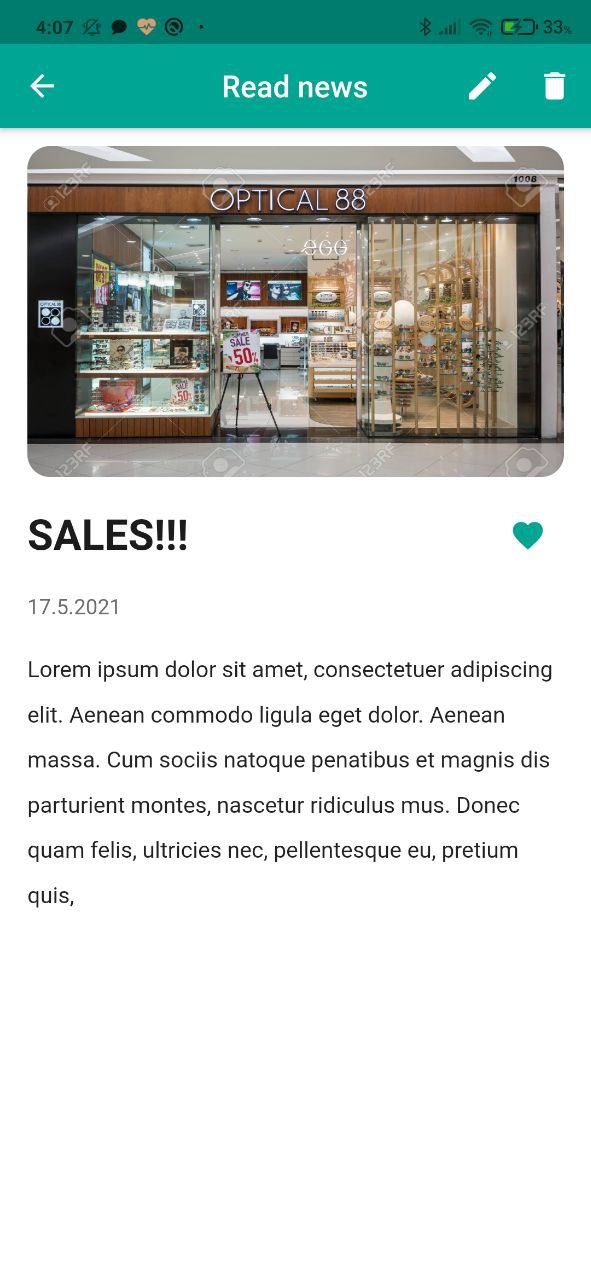


Рисунок 5.5 – Страница чтения новости

Добавление, изменение и удаление продуктов осуществляется по такой же логике. Экран записи на консультацию также представлен в виде формы. Для записи необходимо выбрать салон в выпадающем меню, указать время и необходимую услугу.

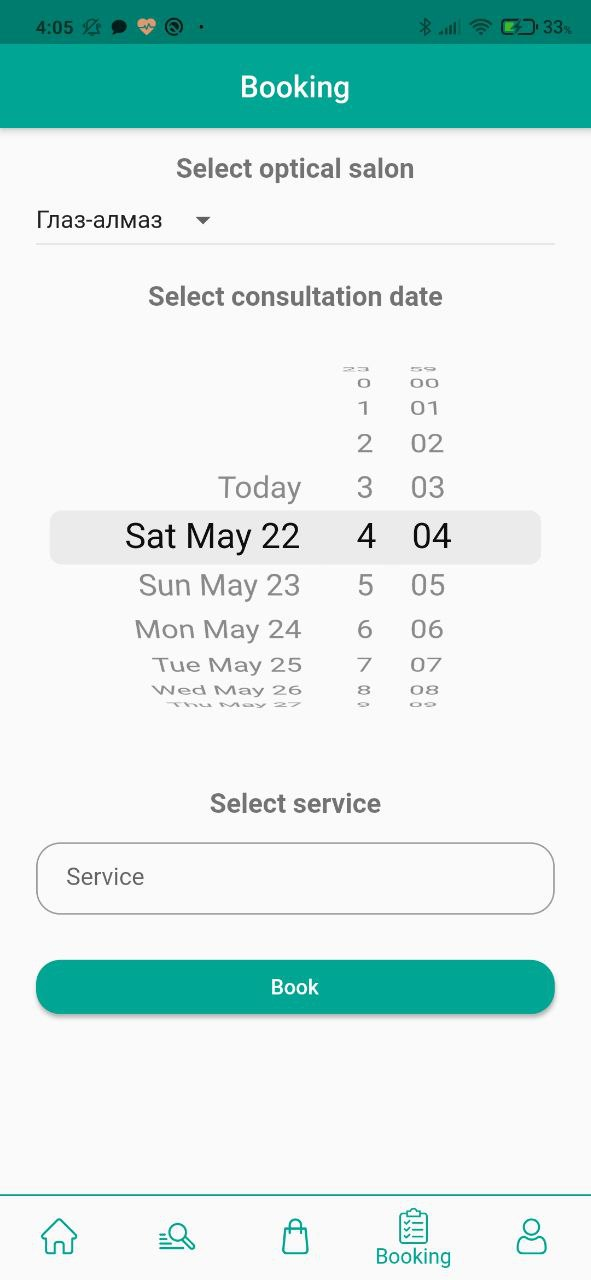


Рисунок 5.6 = Запись на консультацию

Последним экраном является профиль, на котором просто выводиться информация об авторизованном пользователе.

Заключение

При работе над курсовым проектом было проведено изучение предметной области, выявление в ней проблем и составление функционала программного средства, с их учётом, после чего последовало проектирование и разработка. Также в ходе работы проанализировано и проработано множество сторон и нюансов проектирования и разработки мобильного приложения, сервера, а также веб-приложения, например, выбор подходящих паттернов и технологий, составление и тестирование алгоритмов, структурирование проекта, построение пользовательского интерфейса, синхронизации данных и т.д.

В итоге были реализован весь заявленный функционал и готовое приложение вполне можно назвать подходящим для реального использования. Как мобильная, так и серверная части проекта имеют хороший потенциал для будущих модификаций во многом благодаря грамотному структурированию, внимательностью при написании кода, а также выборе известных и проверенных сопутствующих технологий. Было получено большое количество опыта по работе с такими фреймворками как Flutter и NestJS, а также PostgreSql, улучшены навыки по проектированию, а также применению и связыванию друг с другом различных технологий.

Список литературы

1. REST-API [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/483202/>. – Дата доступа: 16.05.2021.
2. StudFiles [Электронный ресурс]. – Режим доступа: https://studfiles.net/preview/page:2/. – Дата доступа: 16.05.2021.
3. Трехуровневая архитектура — Википедия [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Трехуровневая\_архитектура. – Дата доступа: 16.05.2021.
4. Документация по Android [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://developers.google.com/android/> – Дата доступа: 16.05.2021;
5. Мартин, Р. Чистая архитектура и искусство разработки программного обеспечения / Р. Мартин. – Санкт-Петербург: Питер, 2019. – 352 с
6. Metanit Java [Электронный ресурс]. – Режим доступа: <https://metanit.com/java/android/>. – Дата доступа: 16.05.2021
7. Голощапов А. Google Android: программирование для мобильных устройств / Голощапов А. — СПб.: БХВ-Петербург, 2010. — 4

Приложение А

import { ApiProperty } from '@nestjs/swagger';

import { Transform } from 'class-transformer';

import { IsString, IsNotEmpty, MaxLength, Min } from 'class-validator';

export class CreateProductDto {

@ApiProperty({

example: 'Очки',

description: 'Название',

maxLength: 40,

})

@IsString({ message: 'Должно быть строкой' })

@IsNotEmpty({ message: 'Обязательное поле' })

@MaxLength(40, { message: 'Не больше 40' })

name: string;

@ApiProperty({

example: 'Nivea',

description: 'Brand',

maxLength: 40,

})

@IsString({ message: 'Должно быть строкой' })

@IsNotEmpty({ message: 'Обязательное поле' })

@MaxLength(40, { message: 'Не больше 40' })

brand: string;

@ApiProperty({

example: 'БАВ2345',

description: 'Модель',

maxLength: 40,

})

@IsString({ message: 'Должно быть строкой' })

@IsNotEmpty({ message: 'Обязательное поле' })

@MaxLength(40, { message: 'Не больше 40' })

model: string;

@ApiProperty({

example: 'Lorem daasdas фывыфвыфв',

description: 'Описание',

maxLength: 255,

})

@IsString({ message: 'Должно быть строкой' })

@IsNotEmpty({ message: 'Обязательное поле' })

@MaxLength(255, { message: 'Не больше 255' })

description: string;

@ApiProperty({

example: 1,

description: 'Цена',

minimum: 0,

})

@Transform((value) => {

if (+value.value) {

return +value.value;

}

return -1;

})

@IsNotEmpty({ message: 'Обязательное поле' })

@Min(0)

cost: number;

@ApiProperty({

example: 'Беларусь',

description: 'Страна',

maxLength: 40,

})

@IsString({ message: 'Должно быть строкой' })

@IsNotEmpty({ message: 'Обязательное поле' })

@MaxLength(40, { message: 'Не больше 40' })

country: string;

@ApiProperty({

example: 'Металл',

description: 'Материал',

maxLength: 40,

})

@IsString({ message: 'Должно быть строкой' })

@IsNotEmpty({ message: 'Обязательное поле' })

@MaxLength(40, { message: 'Не больше 40' })

material: string;

@ApiProperty({ type: 'string', format: 'binary', required: false })

photo: any;

}

Приложение Б

import { Injectable, CanActivate, ExecutionContext } from '@nestjs/common';  
import { Reflector } from '@nestjs/core';  
import { Role } from '../enums/role.enum';  
import { ROLES\_KEY } from '../decorators/roles.decorator';  
  
@Injectable()  
export class RolesGuard implements CanActivate {  
 constructor(private reflector: Reflector) {}  
  
 canActivate(context: ExecutionContext): boolean {  
 console.log('RolesGuard');  
 const requiredRoles = this.reflector.getAllAndOverride<Role[]>(ROLES\_KEY, [  
 context.getHandler(),  
 context.getClass(),  
 ]);  
 if (!requiredRoles) {  
 return true;  
 }  
 const { user } = context.switchToHttp().getRequest();  
 console.log(user);  
 if (!user) {  
 return false;  
 }  
 return requiredRoles.some((role) => user.roles?.includes(role));  
 }  
}