Contents

**No table of contents entries found.**

ВВЕДЕНИЕ

Сегодняшняя жизнь человека невозможна без применения современных технологий. Несколько лет назад существование смартфонов казалось нереальным, а сейчас многие люди не могут представить свою жизнь без этого устройства. Мобильные приложения предоставляют доступ к быстрой связи, электронной почте, банковским услугам, возможность заказывать такси, еду, одежду и многое другое. Безграничные возможности, которые предоставляются мобильными телефонами, не оставят равнодушным никого.

Выбор темы дипломной работы был обусловлен несколькими факторами. Во-первых, мобильные приложения являются одним из наиболее популярных и востребованных видов программного обеспечения в настоящее время. Это связано с тем, что все большее количество людей используют смартфоны и планшеты в повседневной жизни, и именно мобильные приложения позволяют им получать доступ к нужной информации и функциональности в любое время и в любом месте. Во-вторых, выбор темы связан с моим интересом к разработке приложений под iOS. Автор дипломной работы уже имел опыт работы с этой операционной системой и знает, какие инструменты и технологии можно использовать для создания качественных приложений.

Тема приложения – поиск и бронирование мест в кинотеатрах – была выбрана по нескольким причинам. Во-первых, это позволит мне продемонстрировать свои навыки в создании приложений, которые могут быть полезными для широкого круга пользователей. Во-вторых, кинотеатры являются популярным местом для проведения свободного времени, и приложение, которое позволяет быстро и удобно найти, и забронировать места на сеанс, будет востребовано.

Использование IUKit – одной из основных библиотек для разработки приложений под iOS – позволит продемонстрировать свои знания и навыки в работе с этой технологией. В целом, выбор темы дипломной работы был обоснованным и позволит мне получить ценный опыт в создании мобильных приложений под iOS.

Объектом исследования дипломной работы является процесс разработки мобильного приложения по поиску и бронированию мест в кинотеатры, а предметом исследования выступают уже существующие решения.

Целью данной дипломной работы является разработка мобильного приложения под операционную систему iOS с использованием IUKit, которое позволит пользователям быстро и удобно найти и забронировать места в кинотеатрах. В рамках работы будет проведен анализ требований и функциональности приложения, выбраны оптимальные инструменты и технологии для его разработки, реализованы основные функции и проведено тестирование приложения на соответствие заданным требованиям. Результатом работы будет готовое приложение, которое будет удобным и полезным для пользователей, а также демонстрация знаний и навыков в работе с IUKit.

Для обозначения степени готовности приложения формулируем следующие задачи:

1 Изучение требований и функциональности приложения для поиска и бронирования мест в кинотеатрах.

2 Анализ существующих решений и выбор оптимальных инструментов и технологий для разработки приложения.

3 Разработка интерфейса приложения с использованием IUKit.

4 Реализация основных функций приложения, таких как поиск кинотеатров, просмотр расписания сеансов, выбор мест в зале, бронирование билетов и оплата.

5 Тестирование приложения на соответствие заданным требованиям и исправление ошибок.

6 Оформление документации по разработке приложения и инструкции пользователя.

7 Демонстрация готового приложения и его возможностей.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ, АНАЛОГОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

Успешная разработка приложения для iOS по поиску и бронированию мест в кинотеатрах требует проведения всестороннего предварительного анализа и планирования. Важность этапа проектирования обусловлена необходимостью детальной проработки всех аспектов разрабатываемого мобильного приложения, что позволяет снизить риски и обеспечить качество готового продукта.

Ключевым фактором успешности проекта является корректная постановка и формулирование требований, которые определяют функциональность системы и предотвращают незапланированные изменения в процессе реализации.

1.1 Аналитический обзор литературных источников

Для формулирования требований, выбора подходящей архитектуры и последующего проектирования необходим комплексный аналитический обзор.

**1.1.1** Обзор целевых платформ. При разработке приложений для поиска и бронирования мест в кинотеатрах выбор целевой платформы играет ключевую роль. Мобильные платформы iOS получили широкое распространение благодаря своей надежности, безопасности и удобству использования. Такие платформы позволяют организовать быстрый доступ к информации, удобное бронирование билетов и предоставляют возможность взаимодействия с приложением через смартфон в любом месте, где есть подключение к интернету.

Клиентская часть проектируемого iOS-приложения будет реализована с помощью передового фреймворка UIKit, обеспечивающего высокую производительность и отзывчивость интерфейса. Использование библиотеки Swift позволит создать удобный и современный пользовательский интерфейс с высоким уровнем юзабилити, а URLSession упростит работу с запросами и состоянием приложения. Логика взаимодействия с сервером будет разработана с использованием архитектурного паттерна MVVM, известного своей гибкостью и удобством тестирования, что позволяет эффективно обрабатывать большое количество запросов.

Использование CoreData для локального хранения данных существенно ускорит работу системы, позволяя снизить нагрузку на сеть и обеспечить быструю выдачу данных пользователям. Такой подход особенно актуален при отсутствии стабильного интернет-соединения и для сохранения истории бронирований.

Кроме того, использование FirebaseSDK обеспечит удобное управление аутентификацией и хранением данных, а интеграция с API кинотеатров упростит работу с актуальным расписанием сеансов, обеспечив удобное получение информации о фильмах и наличии свободных мест.

**1.1.2** Обзор архитектурных стилей. Выбор подходящей архитектуры программного средства является ключевым решением, влияющим на производительность, масштабируемость и удобство поддержки системы. В рамках данного проекта выбрана архитектура MVVM, предполагающая четкое разделение презентационной логики и бизнес-логики приложения для поиска и бронирования мест в кинотеатрах.

Архитектура MVVM позволяет существенно упростить процесс разработки, тестирования и поддержки, благодаря разделению ответственности между компонентами приложения. Отделение логики представления от бизнес-логики позволяет снизить связанность кода, что повышает гибкость и расширяемость. Также этот подход упрощает процессы отладки и поиска ошибок, поскольку каждый компонент имеет четко определенную зону ответственности и может быть протестирован независимо от других.

Однако с ростом функциональности и увеличением нагрузки архитектура MVVM может приводить к усложнению ViewModels. Поэтому на более поздних этапах проекта возможно рассмотрение перехода на более сложный архитектурный паттерн VIPER, который позволит более гибко подходить к разделению обязанностей и управлению зависимостями, улучшая общую производительность и удобство сопровождения приложения.

**1.1.3** Проектирование баз данных. Эффективное проектирование базы данных является залогом стабильной работы приложения, особенно при работе с информацией о фильмах, кинотеатрах и бронированиях. В рамках данного проекта было выбрано сочетание локальной базы данных CoreData и облачного сервиса Firebase благодаря их гибкости и производительности при работе с различными типами данных.

Firebase позволяет динамически синхронизировать данные между устройствами, что особенно важно при работе с расписанием сеансов, свободными местами и бронированиями. Облачная база данных Firebase поддерживает масштабирование, обеспечивая эффективную работу приложения даже при значительном росте количества пользователей и хранящейся информации.

Важным этапом проектирования базы данных является определение структур данных (кинотеатры, фильмы, сеансы, пользователи, заказы, отзывы), которые позволяют систематизировать и оптимизировать хранение и извлечение данных. Использование CoreData в качестве локального хранилища для кэширования данных повышает быстродействие системы и позволяет работать с приложением даже при отсутствии интернет-соединения. Такое сочетание позволяет добиться оптимального баланса производительности и удобства использования системы.

**1.1.4** Мобильное приложение для iOS. Мобильные приложения для iOS представляют собой современный подход к разработке, который объединяет лучшие практики проектирования и оптимизации пользовательского опыта. Данный термин стал особенно актуальным после представления iPhone в 2007 году, когда компания Apple создала новый стандарт мобильных устройств и приложений для них.

Основная идея разработки приложений для iOS заключается в создании сервисов, которые обладают высокой производительностью, интуитивно понятным интерфейсом и широким функционалом для пользователей. Благодаря использованию современных технологий, таких как UIKit, CoreData и интеграция с облачными сервисами, приложения для поиска и бронирования мест в кинотеатрах позволяют обеспечивать быстрый доступ к актуальной информации о сеансах и свободных местах даже в условиях нестабильного интернет-соединения.

Компания Apple активно поддерживает и развивает экосистему iOS, предлагая различные инструменты и библиотеки, упрощающие разработку приложений. Одним из таких инструментов является Xcode, который автоматизирует процесс разработки, тестирования и публикации приложений, устраняя необходимость в дополнительных настройках. Он также позволяет гибко управлять процессом разработки, обеспечивая баланс между производительностью приложения и удобством его использования.

На данный момент приложения для iOS широко используются в различных сферах, включая электронную коммерцию, развлечения, социальные сети и бизнес-сервисы. Благодаря своим преимуществам, таким как высокая безопасность, отличная производительность и интеграция с экосистемой Apple, этот подход становится все более популярным среди разработчиков и компаний, ориентированных на качественное обслуживание пользователей.

1.2 Анализ существующих аналогов

На современном рынке существует множество решений и платформ, предназначенных для поиска и бронирования мест в кинотеатрах. Наиболее известными среди них являются KinoAfisha, Афиша Беларуси, Яндекс Афиша и Relax.by. Каждое из этих решений имеет свои преимущества и недостатки, которые необходимо учитывать при разработке нового продукта.

KinoAfisha отличается широким охватом кинотеатров в нескольких странах и информативностью, однако пользователи жалуются на сбои при оплате билетов и отсутствие фильтров, что затрудняет поиск фильмов и кинотеатров. Афиша Беларуси предоставляет пользователям информацию о фильмах, кинотеатрах и премьерах, а также возможность заказать билеты через приложение, но имеет проблемы с производительностью и актуальностью данных об афише.

Яндекс Афиша характеризуется высокой популярностью и надежностью, предлагая широкий спектр развлекательных мероприятий, включая кинотеатры, и возможность кешбэка с покупок билетов, но имеет региональное ограничение – сервис действует только на территории России. Relax.by представляет собой решение, популярное на белорусском рынке, однако функционал платформы относительно широкий и охватывает различные сферы развлечений, что может вызывать путаницу у пользователей при поиске конкретной активности.

Ключевым отличием разрабатываемого приложения является фокус исключительно на поиске и бронировании мест в кинотеатрах, что позволит предложить более интуитивный и удобный интерфейс, ориентированный на конкретную задачу. Такой подход даст возможность значительно упростить взаимодействие пользователя с приложением, сделав процесс выбора фильма и бронирования билетов максимально быстрым и комфортным.

Дополнительным преимуществом разрабатываемого приложения является эффективная интеграция с API кинотеатров, обеспечивающая пользователям доступ к актуальной информации о фильмах, расписании сеансов и наличии свободных мест. Эта функция будет гарантировать точность данных и своевременное обновление информации.

Также разрабатываемое решение будет включать удобную систему выбора и бронирования мест, позволяющую пользователям наглядно видеть расположение мест в зале и их статус (свободно/занято). Это позволит значительно повысить удобство использования приложения, предоставляя интуитивно понятный интерфейс для выбора оптимальных мест в кинозале.

Таким образом, уникальное сочетание узкой специализации, современного интерфейса и надежной системы бронирования позволит приложению занять прочные позиции на рынке, предлагая пользователям новый уровень удобства и эффективности при планировании похода в кино.

**1.3 Требования к проектируемому программному средству**

Формулирование требований к проектируемому программному средству является ключевым шагом на этапе подготовки к реализации проекта. Данный процесс включает в себя детальное описание функциональных и нефункциональных аспектов системы, которые обеспечат её работоспособность, эффективность и удобство использования. Чётко сформулированные требования позволят разработчикам ориентироваться в проекте, избегать возможных ошибок и неопределённостей.

**1.3.1** Назначение проекта. Проект предназначен для создания инновационного мобильного приложения для iOS, позволяющего эффективно осуществлять поиск и бронирование мест в кинотеатрах. Особенность приложения заключается в интеграции с API кинотеатров и удобной системе визуализации свободных мест в зале. Приложение направлено на повышение удобства и скорости бронирования билетов, что позволит обеспечить более качественный и комфортный пользовательский опыт, удовлетворяющий потребностям как любителей кино, так и кинотеатров.

**1.3.2** Основные функции. Основные функции системы включают широкий спектр возможностей для пользователей:

– регистрация и аутентификация с помощью Firebase;

– подробные профили пользователей для сохранения истории бронирований и предпочтений;

– поиск фильмов и кинотеатров с возможностью фильтрации по различным параметрам;

– просмотр расписания сеансов с актуальной информацией о времени, ценах и наличии мест;

– интерактивная схема зала для выбора и бронирования свободных мест;

– финансовые операции, включая оплату билетов через встроенные платежные системы с обеспечением безопасности платежей;

– адаптивный дизайн для различных моделей iPhone и iPad;

– управление профилями пользователей и историей заказов.

**1.3.3** Дополнительные функции. В дополнение к основным возможностям, приложение предусматривает реализацию следующих функций:

– система уведомлений о начале сеанса и напоминаний о забронированных билетах;

– интеграция с API "Кинопоиска" для получения подробной информации о фильмах, включая рейтинги, трейлеры и отзывы;

– возможность оставлять отзывы о фильмах и кинотеатрах, формируя базу пользовательских оценок;

**1.3.4** Требования к надежности. Для обеспечения надежной работы мобильного приложения для поиска и бронирования мест в кинотеатрах необходимо выполнить комплекс организационно-технических мероприятий. Это включает соблюдение рекомендаций Apple по разработке приложений, требования по защите персональных данных пользователей и обеспечение бесперебойной работы приложения даже при нестабильном интернет-соединении. Время восстановления системы после сбоя не должно превышать время, необходимое для перезагрузки приложения или обновления локальных данных. Также важно обеспечить интуитивно понятный интерфейс, не требующий специальных навыков от пользователя для эффективного использования.

Надежность системы обеспечивается следующими мерами:

– высокий уровень защиты пользовательских данных, включая безопасное хранение информации о платежах и личных данных;

– локальное кэширование данных с помощью CoreData для обеспечения доступа к информации даже при отсутствии интернет-соединения;

– система синхронизации данных с серверами Firebase и оперативное обновление информации о доступных сеансах и местах.

**1.3.5** Выбор инструментов разработки. Выбор инструментов разработки является одним из важнейших этапов подготовки к реализации проекта, поскольку напрямую влияет на эффективность, производительность и дальнейшую поддержку разработанного мобильного приложения.

Для реализации приложения выбран современный и популярный язык программирования Swift. Данный язык обладает высокой производительностью за счёт строгой типизации и эффективного управления памятью. Использование Swift позволяет создавать динамичные мобильные приложения с минимальным временем загрузки, высокой отзывчивостью интерфейса и простотой в масштабировании. Также Swift предоставляет возможность использовать современные подходы к разработке, что делает процесс создания приложения более гибким и удобным.

В сочетании со Swift будет применяться фреймворк UIKit, который предоставляет широкий набор готовых UI-компонентов и элементов управления для быстрого и удобного создания современного пользовательского интерфейса. UIKit отличается высоким уровнем кастомизации, поддержкой различных устройств и адаптивным дизайном, что позволяет создавать приложения, удобные в использовании на различных моделях iPhone и iPad.

Для управления состоянием и обработки запросов к серверу был выбран архитектурный паттерн MVVM (Model-View-ViewModel), обеспечивающий удобное и эффективное разделение ответственности между компонентами приложения. MVVM позволяет автоматизировать процессы получения, отображения и обновления данных, значительно упрощая процесс разработки и улучшая производительность приложения.

Для сетевого взаимодействия был выбран URLSession, встроенный в iOS API для выполнения HTTP-запросов. URLSession использует эффективную асинхронную архитектуру, что обеспечивает высокую производительность и лёгкость в расширении и поддержке системы. Данный инструмент особенно хорошо подходит для реализации приложений, которым требуется высокая скорость обработки большого количества запросов и высокая надёжность.

Для хранения данных в приложении будет использоваться CoreData и Firebase. CoreData является мощным и гибким фреймворком для локального хранения данных, который позволяет эффективно управлять информацией даже при отсутствии интернет-соединения. Firebase представляет собой облачную платформу, обеспечивающую удобную аутентификацию пользователей и синхронизацию данных. Такая комбинация особенно важна для проекта, где требуется как локальное хранение информации о бронированиях, так и синхронизация с актуальными данными о сеансах и местах.

Для разработки и управления кодом выбрана интегрированная среда разработки Xcode. Xcode обеспечивает удобный интерфейс, поддержку инструментов для отладки, интеграции с системами контроля версий и автоматизации задач. Эта среда позволяет разработчикам эффективно взаимодействовать с кодом и поддерживать высокую продуктивность работы.

Разработка и тестирование приложения будут осуществляться с использованием симуляторов iOS и реальных устройств. Использование симуляторов обеспечивает быстрое тестирование на различных моделях устройств, а тестирование на реальных устройствах позволяет выявить проблемы, связанные с производительностью и интерфейсом.

Для интеграции с API кинотеатров и получения информации о фильмах будет использоваться API "Кинопоиска". Это позволит предоставлять пользователям актуальную информацию о премьерах, рейтингах, трейлерах и других деталях фильмов.

Таким образом, выбранный набор инструментов и технологий обеспечивает все необходимые условия для успешной и эффективной разработки, последующего масштабирования и комфортной поддержки создаваемого мобильного приложения для поиска и бронирования мест в кинотеатрах.

2 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

Важным этапом разработки любого программного средства является его моделирование и формулировка функциональных требований. Этот этап необходим для детального описания взаимодействия различных элементов системы, а также для того, чтобы обеспечить ясное и точное понимание того, как будет функционировать проектируемое приложение в реальных условиях эксплуатации. Моделирование позволяет выявить возможные недостатки проектируемой системы на ранних этапах разработки, а также упрощает процесс разработки и дальнейшей поддержки программного продукта.

Процесс моделирования включает создание функциональной модели системы, описывающей возможные варианты взаимодействия пользователей и системы, а также разработку инфологической модели базы данных, отражающей структуру хранения и обработки данных в приложении.

2.1 Функциональная модель программного средства

Функциональная модель проектируемой системы описывает взаимодействие различных категорий пользователей с системой, а также действия, которые они могут выполнять в процессе работы с приложением. Такая модель позволяет наглядно представить весь функционал системы и помогает разработчикам и аналитикам более точно определить требования к проектируемому программному средству.

Функциональная модель проектируемого приложения включает подробное описание сценариев использования системы (use cases), с чётким указанием ролей пользователей, их прав доступа и возможных действий в приложении. Это позволяет обеспечить эффективную разработку и гарантировать высокое качество пользовательского опыта.

**2.1.1** Варианты использования программного средства. Система предполагает поддержку нескольких ролей пользователей, каждая из которых обладает своим набором прав и возможностей. Выделяются следующие роли: гость, зарегистрированный пользователь, администратор и менеджер кинотеатра.

Гость – это неавторизованный пользователь системы. Гость имеет возможность просматривать основную информацию о приложении, афишу фильмов, расписание сеансов, но без возможности бронирования. Также гость может проходить процесс регистрации и аутентификации. Процесс регистрации должен включать ввод основных данных (электронной почты, пароля) и дальнейшую проверку введённых данных на корректность и безопасность.

Зарегистрированный пользователь – авторизованный пользователь, прошедший процесс регистрации и авторизации. Основными действиями пользователя являются поиск фильмов и кинотеатров, просмотр расписания сеансов, выбор и бронирование мест, оплата билетов, а также управление своим профилем. Пользователь может просматривать историю своих заказов, оставлять отзывы о фильмах и кинотеатрах, а также получать персонализированные рекомендации. Система рекомендаций будет автоматически подбирать и предлагать наиболее релевантные фильмы, учитывая предпочтения пользователя и его предыдущую активность. Пользователь может также использовать реферальную систему, приглашая других участников и получая бонусные баллы для скидок на будущие билеты.

Администратор – это пользователь, обладающий наибольшими правами в системе. Администратор может управлять пользователями, добавлять, редактировать и удалять информацию о фильмах и кинотеатрах, а также осуществлять модерацию отзывов и контроль контента. Также администратор имеет доступ к управлению финансовыми операциями и мониторингу транзакций.

Менеджер кинотеатра – специалист, задача которого заключается в управлении расписанием сеансов и мест в конкретном кинотеатре. Менеджер имеет доступ к инструментам аналитики, позволяющим генерировать отчёты о заполняемости сеансов, популярности фильмов и визуализировать результаты продаж билетов. Менеджеры могут просматривать и экспортировать данные в различные форматы для дальнейшего анализа вне платформы.

Таким образом, разработанная функциональная модель проектируемого программного средства охватывает все аспекты взаимодействия пользователей с системой и позволяет обеспечить её эффективную работу, соответствие требованиям и высокий уровень удовлетворенности пользователей.

**2.1.2** Разработка инфологической модели базы данных. Разработка инфологической модели базы данных является необходимым этапом проектирования информационной системы, поскольку от правильности её реализации зависит эффективность и стабильность работы всего приложения. Инфологическая модель предназначена для структуризации и систематизации информации, используемой системой, а также для чёткого определения сущностей и взаимосвязей между ними.

В рамках проектируемого приложения для поиска и бронирования мест в кинотеатрах были выделены ключевые сущности и их характеристики:

Пользователь – центральная сущность системы, хранящая основную информацию о каждом пользователе: идентификатор, электронная почта, пароль (хешированный), профиль пользователя, дата регистрации, статус аккаунта (активен, заблокирован, удален), реферальный идентификатор (для бонусной системы), баланс пользователя и история транзакций.

Профиль – подробная информация о пользователе, которая включает личные данные, предпочтения по жанрам фильмов, любимые кинотеатры и другую информацию, необходимую для персонализации рекомендаций фильмов и предложений. Структура профиля содержит историю просмотров, историю бронирований и избранное.

Фильм – сущность, представляющая информацию о киноленте. Включает название, подробное описание, жанр, продолжительность, возрастное ограничение, рейтинг, актерский состав, режиссера, дату премьеры, трейлеры и постеры.

Кинотеатр – сущность, содержащая информацию о кинотеатре: название, адрес, контактную информацию, количество залов, схемы залов, рейтинг и отзывы пользователей.

Сеанс – связующая сущность между фильмом и кинотеатром, включающая информацию о дате и времени показа, зале, цене билетов, типе показа (2D, 3D, IMAX), статусе сеанса (активен, завершён, отменён).

Место – элемент зала кинотеатра, содержащий информацию о номере ряда, номере места, типе места (обычное, VIP, для людей с ограниченными возможностями), статусе места (свободно, забронировано, продано).

Бронирование – сущность, содержащая информацию о бронировании: идентификатор пользователя, идентификатор сеанса, выбранные места, дата и время бронирования, статус бронирования (подтверждено, отменено, оплачено), сумма к оплате.

Таким образом, разработанная инфологическая модель позволяет обеспечить полную, структурированную и эффективную работу с данными, что значительно повышает качество и надёжность проектируемого приложения.

* 1. Разработка спецификации функциональных требований

С учетом требований, определенных в подразделе 1.3, представим детализацию функций проектируемого ПС.

Для детализации функций рассмотрим основные требования, предъявляемые к каждой функции программного средства как с точки зрения внутренней организации системы, так и с точки зрения взаимодействия системы с пользователем.

При разработке спецификаций функциональных требований кроме всего прочего учитывались особенности платформы разработки.

**2.2.1** Регистрация и авторизация пользователей. Платформа должна обеспечивать простоту и безопасность регистрации и авторизации пользователей. Важным аспектом является предоставление возможности регистрации с использованием электронной почты и пароля с обязательным подтверждением электронной почты для предотвращения мошеннических регистраций.

Для удобства и расширения охвата аудитории предусмотрена возможность авторизации с помощью популярных социальных сетей, таких как Google и VK. Для дополнительной защиты учетных записей пользователей будет реализована двухфакторная аутентификация, что существенно повысит уровень безопасности системы.

**2.2.2** Управление профилем пользователя. Каждый пользователь приложения будет иметь возможность детально управлять информацией в своем профиле. Профиль будет содержать как базовые данные (имя, фамилия, дата рождения, пол), так и предпочтения по жанрам фильмов, любимым кинотеатрам и удобному времени посещения.

Пользователь сможет в любой момент редактировать и актуализировать эти данные, а также просматривать историю своих бронирований и оплаченных билетов. Предусмотрена возможность загрузки аватара для повышения персонализации профиля и комфортности использования приложения.

**2.2.3** Система рекомендаций релевантных фильмов. Основным конкурентным преимуществом приложения будет продвинутая система рекомендаций фильмов.

Эта система будет автоматически анализировать предпочтения пользователей, учитывать историю просмотров предыдущих фильмов и персональные интересы, после чего предлагать наиболее подходящие и интересные киносеансы. Уведомления о премьерах и новых сеансах будут отправляться пользователям как на электронную почту, так и внутри приложения, что позволит поддерживать высокий уровень вовлеченности пользователей.

**2.2.4** Поиск и бронирование мест. Процесс поиска и бронирования мест будет максимально простым и понятным, при этом обеспечивая высокое качество сервиса. Пользователь получит доступ к четко структурированному списку доступных и рекомендованных системой фильмов, составленному на основе профиля и ранее просмотренных фильмов.

Выбрав интересующий фильм, пользователь будет переходить к выбору кинотеатра, даты и времени сеанса. Затем система предоставит интерактивную схему зала с отображением свободных и занятых мест. Пользователь сможет выбрать желаемые места и добавить их в корзину для последующей оплаты.

Во время бронирования предусмотрена функция автоматического сохранения данных на каждом этапе, что исключит риск потери информации в случае прерывания интернет-соединения или закрытия приложения. Для обеспечения дополнительной мотивации пользователей предусмотрена индикация времени до начала сеанса и ограниченное время удержания брони.

**2.2.5** Административная панель управления. Административная панель управления станет центральным элементом для эффективного и удобного контроля за работой приложения. Администраторы смогут добавлять новые фильмы, кинотеатры, настраивать расписание сеансов и схемы залов, а также производить редактирование или архивирование устаревшей информации.

Встроенная система модерации позволит администраторам проверять качество и корректность отзывов пользователей, управлять их учетными записями, блокировать нарушителей и следить за активностью в приложении в режиме реального времени.

Для удобства работы с большими объемами информации в панели предусмотрены разнообразные аналитические инструменты, позволяющие формировать детализированные отчеты по активности пользователей, заполняемости сеансов и финансовым операциям. Дополнительно реализована возможность мониторинга транзакций и управления скидками и акциями.

**2.2.6** Бонусная система. Для стимулирования регулярных посещений кинотеатров и повышения вовлеченности пользователей приложение будет оснащено мощной бонусной системой. Пользователи будут получать бонусные баллы за каждое бронирование и приглашение новых участников. Накопленные баллы можно будет использовать для получения скидок на последующие покупки билетов или обменивать на специальные предложения от партнеров.

В панели управления бонусной системой будет доступна подробная статистика, отображающая количество накопленных баллов, историю их получения и использования, а также информация о доступных акциях и специальных предложениях.

**2.2.7** Безопасность и защита данных. Важнейшей задачей приложения является обеспечение надежной защиты персональных данных пользователей и конфиденциальности их платежной информации. Будет использовано многоуровневое шифрование всех данных, как в процессе передачи, так и при их хранении на сервере. Регулярное резервное копирование данных позволит оперативно восстанавливать информацию в случае технических сбоев.

Для предотвращения мошенничества и несанкционированного доступа предусмотрены механизмы мониторинга подозрительной активности и автоматического блокирования аккаунтов с выявленными нарушениями. Также приложение будет полностью соответствовать международным стандартам защиты данных, в том числе требованиям GDPR, обеспечивая пользователям прозрачность использования и хранения их личной информации.

**2.2.8** Поддержка и обратная связь. Для поддержания высокого уровня удовлетворенности пользователей будет предусмотрена удобная и эффективная система поддержки и обратной связи.

Пользователи смогут обратиться в службу поддержки посредством удобной формы обратной связи в приложении, через онлайн-чат в режиме реального времени или систему тикетов для решения сложных вопросов и технических проблем.

Встроенная система мониторинга и автоматизированного контроля качества работы позволит оперативно идентифицировать проблемы и максимально быстро устранять их.

Кроме того, пользователи получат возможность оставлять отзывы о фильмах и кинотеатрах, а также отправлять предложения по улучшению работы приложения, что позволит разработчикам постоянно совершенствовать систему и адаптировать её под актуальные потребности аудитории.

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Определение методологии разработки, архитектурного паттерна и языка программирования

Для реализации приложения был выбран актуальный язык программирования для нативной разработки мобильных приложений на iOS – Swift. Данный язык программирования был представлен на конференции Apple Worldwide Developers Conference (WWDC). Swift имеет возможность взаимодействия со своими предшественниками C и Objective-C, что делает разработку и поддержку уже существующих приложений более гибкой и удобной.

Выбор правильной архитектуры является ключевым фактором для успешного и качественного поддержания программного обеспечения. Архитектура представляет собой набор правил, которые помогают решать проблемы проектирования. Существует множество архитектурных паттернов, каждый из которых имеет свои преимущества и недостатки. Однако, хороший архитектурный подход должен обеспечивать следующие возможности:

1 Сбалансированное распределение обязанностей между блоками и строгое соблюдение правил. Это позволяет разбить сложный модуль на необходимые слои, упростить понимание кода другими разработчиками, повторно использовать сущности в других местах программы или проектах, а также быстро определять возникшие проблемы.

2 Тестируемость. Написанный код должен быть разбит на функциональные логические модули, которые можно будет тестировать, при этом не затрагивая другие.

3 Разбиение сложного модуля на слои. Если модуль перегружен, его нужно разбить на отдельные слои, каждый из которых будет выполнять свою задачу. Это позволит упростить тестируемость в будущем и позволит повторно использовать его компоненты.

4 Расширяемость. Расширяемость позволяет добавлять новый функционал в приложение, при этом не затрагивая (или затрагивая минимально) уже существующий код.

Для удовлетворения вышеописанных пунктов существует множество архитектурных паттернов, общих принципов программирования и методологий разработки программного обеспечения. Для начала пройдемся по ряду основных принципов программирования, так как их придерживаются, ну или стараются придерживаться, все разработчики, независимо от языка и области программирования.

3.1.1 Основные принципы программирования. Основными принципами программирования являются SOLID, DRY, KISS, YAGNI.

SOLID состоит из пяти принципов программирования:

1 Single Responsibility Principle (принцип единственной ответственности) – каждый класс должен иметь только одну ответственность и быть специализированным на выполнении этой задачи. Если класс задает направление движения машины, то этот класс не должен выполнять какие-либо другие задачи. Таким образом, данный принцип помогает разбивать общую конструкцию на независимые модули и уменьшать межмодульную связь.

2 Open-Closed Principle (принцип открытости/закрытости) – классы должны быть открыты для расширения, но закрыты для изменения. При необходимости добавления новой функции, уже существующий класс не модифицируется, а расширяется при помощи добавления нового кода. То есть, должна быть возможность расширять класс, без его изменения.

3 Liskov Substitution Principle (принцип подстановки Барбары Лисков) – объекты должны быть заменяемыми на экземпляры их подтипов без изменения корректности программы. Проще говоря, поведение наследующих классов не должно противоречить поведению, заданному базовым классом, то есть поведение наследующих классов должно быть ожидаемым для кода, использующего переменную базового типа.

4 Interface Segregation Principle (принцип разделения интерфейса) – клиенты не должны зависеть от методов, которые они не используют. Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы программные сущности маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться программные сущности, которые этот метод не используют.

5 Dependency Inversion Principle (принцип инверсии зависимостей) – высокоуровневые модули не должны зависеть от низкоуровневых модулей. Оба типа модулей должны зависеть от абстракций.

DRY расшифровывается как «Don’t Repeat Yourself». Этот принцип гласит, что каждая часть кода должна быть написана только один раз, чтобы избежать дублирования и упростить поддержку и изменение кода.

KISS расшифровывается как «Keep It Simple, Stupid». Этот принцип, который гласит, что код должен быть простым и понятным, чтобы упростить его понимание и поддержку.

YAGNI расшифровывается как «You Ain't Gonna Need It». Этот принцип, который гласит, что не следует создавать функциональность, которая не требуется в данный момент, чтобы избежать излишней сложности и ускорить разработку.

3.1.2 Методологии разработки. Огромную роль в разработке играет выбор методологии. Методология разработки – это набор принципов, подходов и инструментов, используемых при создании программного обеспечения. Эти методологии помогают организовать процесс разработки, управлять рисками и обеспечивать качество продукта.

Существует множество методологий разработки. Одними из самых популярных являются Waterfall, V-Model, Incremental Model, Spiral Model, Agile.

«Водопадная» или «каскадная» модель – последовательный подход к разработке, где каждый этап завершается перед началом следующего. Она подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В модели Waterfall легко управлять проектом. Благодаря ее жесткости, разработка проходит быстро, стоимость и срок заранее определены. Однако у такой модели есть и минусы. Каскадная модель будет давать отличный результат только в проектах с четко и заранее определенными требованиями и способами их реализации. Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта. Каскадную модель можно использовать только когда требования известны, среди них нет противоречий, и объем работы в проекте относительно небольшой. Ее диаграмма представлена на рисунке 3.1.

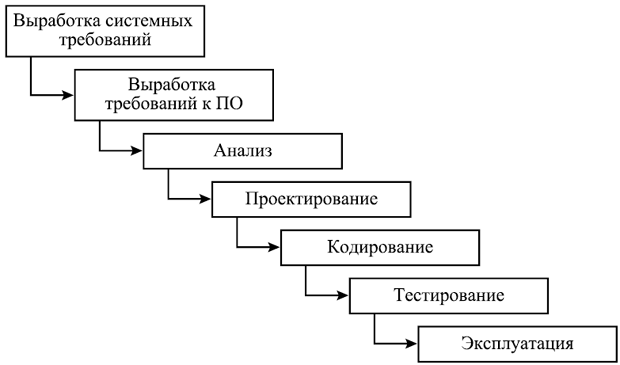


Рисунок 3.1 – Диаграмма модели Waterfall

V-Model схожа с каскадной, так как заимствовала из нее структуру «шаг за шагом». Ключевым отличием является то, что продукт тестируется на протяжении всего этапа разработки. V-модель подходит для проектов, в которых важна надёжность и цена ошибки очень высока. Например, при разработке подушек безопасности для автомобилей или систем наблюдения за пациентами в клиниках. Ее диаграмма представлена на рисунке 3.2.

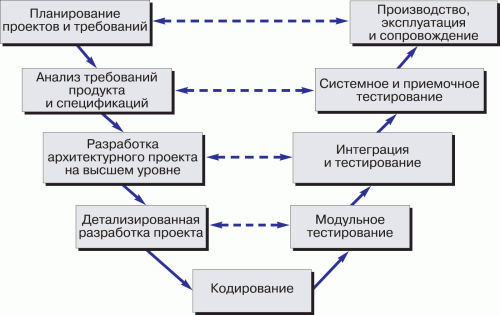


Рисунок 3.2 – Диаграмма модели V-Model

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система. Такая модель используется, когда основные требования к программе четко определены, в то время как некоторые детали могут дорабатываться со временем. Главным преимуществом этой модели является быстрый выход на рынок.

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия – работоспособна. Такая модель используется, если определена основная задача, но из-за масштаба проекта невозможно определить требования к деталям. Таким образом, детали проекта могут эволюционировать с течением времени. Диаграмма инкрементной и итерационной моделей представлена на рисунке 3.3.

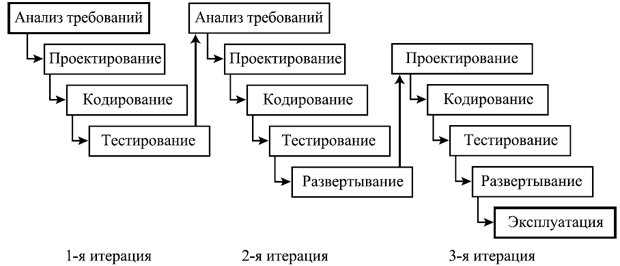


Рисунок 3.3 – Диаграмма итерационной модели

Используя спиральную модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка – цикла итераций – принимается решение, продолжать ли проект. Спиральная модель похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек или при необходимости научных исследований. Модель включает в себя 4 этапа для каждого витка: планирование, анализ рисков, конструирование, оценка результата и переход к новому витку. Спиральная модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование. Ее диаграмма представлена на рисунке 3.4.

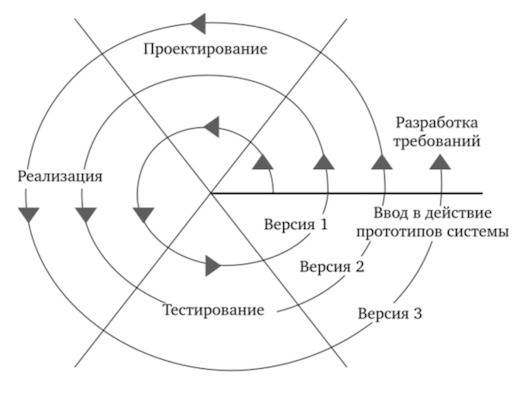


Рисунок 3.4 – Диаграмма спиральной модели

Agile (пер. «гибкий») – это методология разработки программного обеспечения, которая основывается на коллективной работе, частых итерациях и максимальной гибкости в изменении требований и планов.

Основные принципы Agile:

1 Люди и взаимодействие важнее процессов и инструментов.

2 Работающий продукт важнее исчерпывающей документации.

3 Сотрудничество с заказчиком важнее согласования условий контракта.

4 Готовность к изменениям важнее следования плану.

Agile используется не только в разработке программного обеспечения, но и в других областях бизнеса, таких как маркетинг, управление проектами и т.д. В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике. В основе такого типа – непродолжительные ежедневные встречи – «Scrum» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». Методология подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. Соответственно, в процессе реализации требования изменяются.

В процессе разработки приложения руководствовались принципами Agile. Разработка велась по первичному плану, который позволил нарисовать общую картину реализации. В процессе план менялся и дорабатывался: от некоторых функций приходилось отказаться ввиду невозможности реализации, а также добавлялись новые, которых в первичном плане не было.

Когда уже определен стек технологий, методология разработки и язык программирования, необходимо выбрать архитектурный паттерн (шаблон проектирования). Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения – повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. Архитектурные паттерны помогают создавать чистый и структурированный код, который легко понять и сопровождать. Это упрощает работу над приложением, ускоряет процесс разработки и уменьшает количество ошибок.

В контексте разработки мобильных приложения для операционной системы iOS самыми популярными являются архитектуры MVC, MVP, MVVM, VIPER. Каждая из них будет описана и проанализирована, в результате чего будет выбрана наиболее подходящая для выполнения поставленных задач.

MVC (Model-View-Controller). Один из самых популярных архитектурных паттернов, применяемых как при проектировании мобильных приложений для iOS, так и в других сферах разработки программного обеспечения. Популярность данного паттерна в iOS разработке также обусловлена тем, что Apple настоятельно рекомендует использовать именно его. В рамках данного архитектурного решения существуют три основных слоя – модель (Model), представление (View), контроллер (Controller). В классической реализации MVC модель отвечает за бизнес – логику и является хранилищем данных, предоставляя их представлению по команде контроллера (рисунок 3.5). Контроллер реагирует на действия пользователя и оповещает об этом модель. Представление отображает данные пользователю, полученные от модели. Основная проблема, практически сразу исключающая данное архитектурное решения для реализации поставленных задач, заключается в тесной связи между слоями: каждый из слоев знает о двух других. Это сразу же снижает возможность тестирования каждого из них.

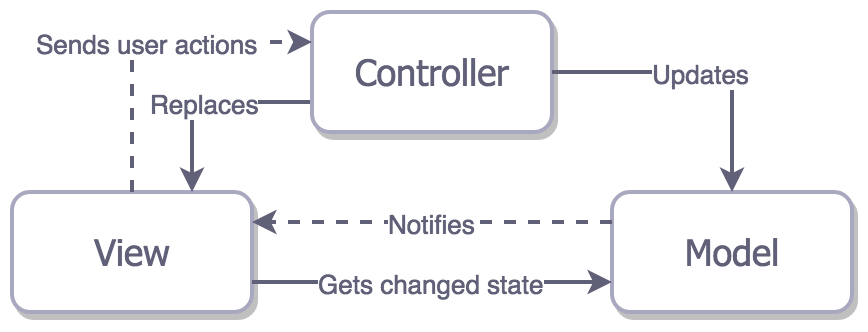


Рисунок 3.5 – Классическая реализация MVC

В разработке мобильных приложения для операционной системы iOS большая часть представлений наследуют базовый класс UIViewController, поэтому представление и контроллер участвуют как в отображении контента, так и в обработке действий пользователя. Это наглядно показано на рисунке 3.6. Из-за этого их достаточно сложно отделить друг от друга. Если большую часть данных и бизнес – логику еще представляется возможным разместить в модели, то отделить представление и контроллер в рамках данного архитектурного решения очень сложно, что ведет к одной серьезной проблеме – отсутствие возможности тестировать представление или контроллер по-отдельности.

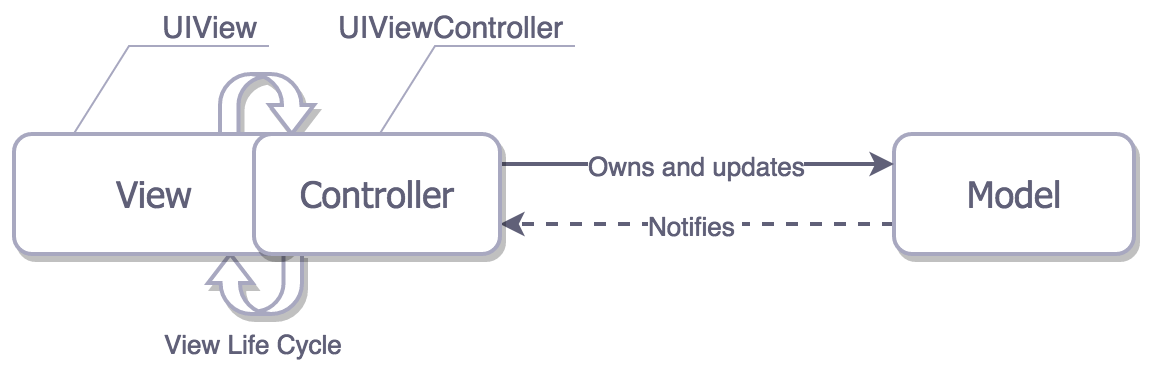


Рисунок 3.6 – MVC мобильных приложений

Исходя из вышесказанного, можно сказать, что с точки зрения разделения ответственности модель и контроллер разделены, в отличие от представления и контроллера. Протестировать получится только модель, потому что представление и контроллер тесно связаны друг с другом.

К плюсам можно отнести простоту реализации, из чего следует, что данный паттерн хорошо подходит для не больших программ, с не большим набором функционала.

К минусам относится разрастание Contorller’a, т.к. он реализует большинство функций приложения – работа с API, с базами данных, бизнес-логика приложения, и все это тесно связано с пользовательским интерфейсом, а в iOS разработке с UIKit – что означает, что для такого приложения сложно писать тесты.

MVP (Model-View-Presenter). Данный архитектурный паттерн решает основную проблему описанного ранее MVC. В концепции MVP представление отделено от модели, а также от любой логики.

Представление только реагирует на действия пользователя, сообщает об этом Presenter’у, который в свою очередь вызывает необходимые методы в модели. Модель отвечает за бизнес логику, а также хранит данные. Например, пользователь нажимает на кнопку, об этом действии представление оповещает Presenter, который потом обращается к модели для загрузки необходимых данных с сервера. После загрузки модель передает полученные данные в Presenter, который обрабатывает их и передает представлению для дальнейшего отображения пользователю.

Cточки зрения распределения обязанностей MVP имеет хорошее преимущество над тем же MVC. Каждый слой имеет лишь одну обязанность. Хорошая тестируемость описываемого архитектурного решения достигается за счет разделения обязанностей. Концепция MVP достаточно легка в понимании, однако требует написания довольно большого количества кода. Схема паттерна MVP представлена на рисунке 3.7.

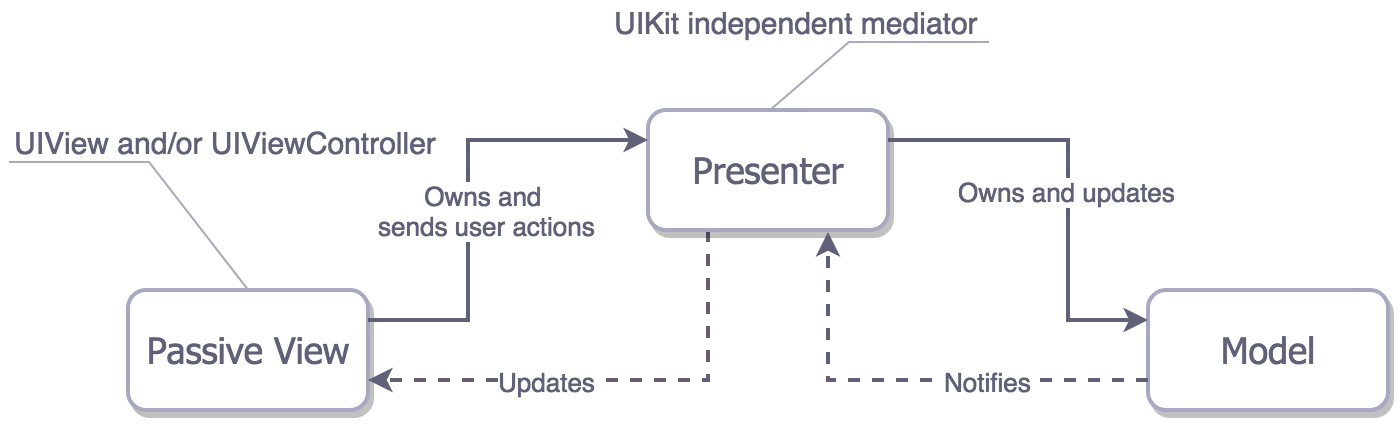
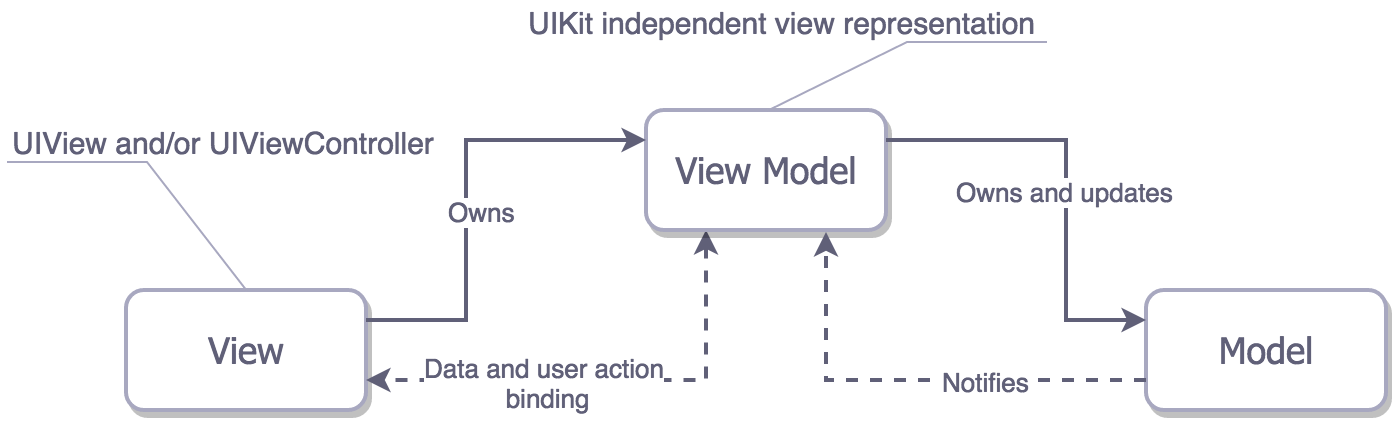


Рисунок 3.7 – Схема паттерна MVP

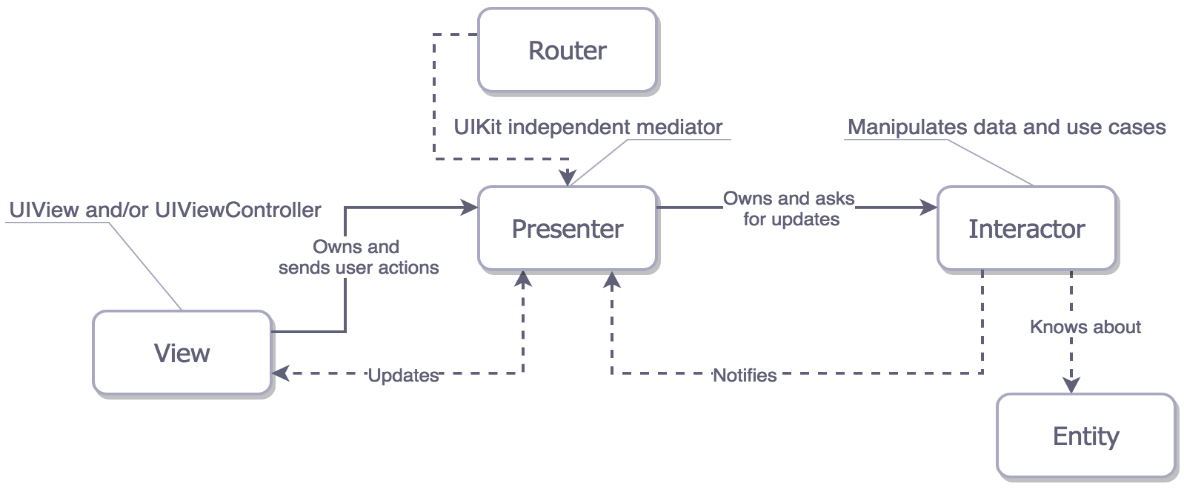
MVVM (Model-View-ViewModel). MVVM – это архитектурный паттерн, который похож на MVP. В нем также отсутствует прямая связь между представлением и моделью, однако в MVVM добавляется новый слой – ViewModel, который хранит состояние текущего представления и обновляет его при изменении модели. Представление и модель в MVVM выполняют такие же функции, что и в MVP. Однако новый слой несколько отличается от Presenter’a. ViewModel хранит состояние текущего представления. Вызывая методы из модели, ViewModel обновляет свое состояние, а вместе с ним, с помощью соответствующей привязки обновляется и само представление. Представление выступает в качестве наблюдателя за изменениями ViewModel и производит необходимые изменения в ответ на уведомления.

Распределение обязанностей в MVVM не такое сбалансированное, как в MVP, потому что представление имеет дополнительную обязанность в виде подписи и обработки изменений ViewModel. Данный архитектурный паттерн имеет высокий уровень тестируемости, обусловленный разделением ViewModel от представления.



VIPER (View-Interactor-Presenter-Entity-Router). VIPER является представлением «Чистой архитектуры» (Clean Architecture). Сила этого шаблона заключается в правильном распределении обязанностей по разным слоям. Представление, как и в некоторых вышеописанных паттернах, отвечает за отображение необходимого контента. Представление оповещает Presenter о действиях пользователя. Presenter в свою очередь обращается к Interactor’y и роутеру. Interactor отвечает за бизнес логику, а также управляет сущностями (Entities). Сущность – это модель данных, не содержащая какой-либо бизнес логики. Роутер ответственен за инициализацию и навигацию модуля.

Тщательное разделение слоев в VIPER’е позволяют легко расширять модуль, при этом никак его не модифицируя. С точки зрения разделения ответственностей является лидером среди упомянутых в работе архитектурных паттернов. Также VIPER также лидирует в тестируемости, благодаря сбалансированному разделению ответственности между слоями. Проблема с этим шаблоном заключается в том, что в большинстве сценариев он может быть излишним, поскольку нужно управлять множеством слоев, и это может быть запутанным и трудным в управлении.



Подробный анализ популярных архитектурных решений, используемых в мобильной разработке, позволил выделить основные плюсы и минусы каждого из них. В качестве архитектурного паттерна в рамках разработки мобильного приложения данной работы был выбран MVVM. Выбор обусловлен актуальностью паттерна, простотой в реализации и личном опыте, которого у меня в разработке еще, по ощущениям, недостаточно, чтобы придерживаться VIPER. К тому же, для приложения такого объема работы, VIPER может быть излишним, а MVVM обладает не меньшей степенью разделения обязанностей, расширяемости и тестируемости.

3.2 Использование IUKit для создания интерфейса и взаимодействия с пользователем

Создание интерфейсов в iOS разработке происходит с помощью двух основных фреймворков – это UIKit и SwiftUI. Оба фреймворка были разработаны компание Apple для разработчиков мобильных приложений. UIKit появился в 2007 году с выходом первого iPhone. UIKit является уникальным фреймворком, так как он был специально разработан для работы на устройствах Apple и включает в себя уникальные элементы интерфейса, такие как Navigation Bar и Tab Bar, которые стали стандартом для iOS-приложений. UIKit обеспечивает функциональные возможности для построения и управления пользовательским интерфейсом, анимацией, текстом, изображениями для ваших приложений, а также для обработки событий, происходящих во время работы пользователя. Если говорить проще, то UIKit **–** это набор основных возможностей, которые разработчики используют при построении интерфейса практически любого приложения.

SwiftUI **–** это новый фреймворк, представленный компанией Apple в 2019 году, который позволяет создавать пользовательский интерфейс для приложений на iOS, macOS, watchOS и tvOS с использованием Swift. Он предоставляет более простой и декларативный подход к созданию пользовательского интерфейса, чем UIKit, и позволяет разработчикам быстрее и легче создавать красивые и интерактивные приложения. SwiftUI также поддерживает многие элементы интерфейса, которые были введены в UIKit, такие как NavigationView и TabView.

Для разработки приложения был выбран UIKit по нескольким причинам:

1 SwiftUI еще довольно молодой фреймворк и в нем не реализованы многие функции и элементы, которые доступны в UIKit.

2 UIKit предоставляет больше возможностей в виду большего количества сторонних библиотек, которые могут упростить разработку приложения.

3 Несмотря на то, что Apple хочет перейти от UIKit’a к SwiftUI, на данный момент UIKit используется намного шире. Старые приложения все еще поддерживаются с помощью UIKit.

Стоит упомянуть навигационные паттерны в мобильной разработке. В приложении были использованы следующие паттерны:

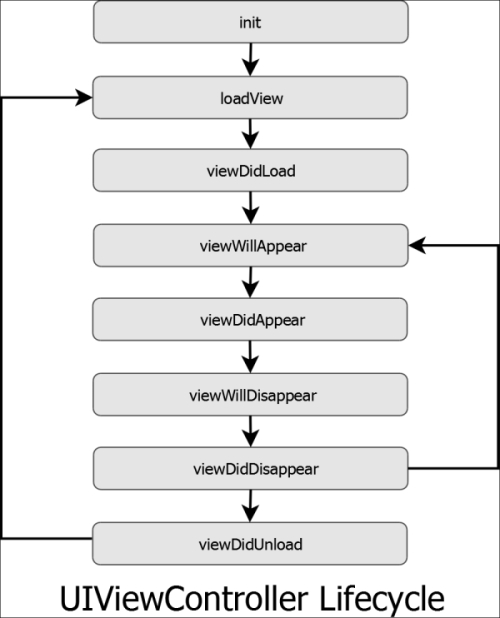
1 Tab Bar Controller. Это паттерн навигации, который позволяет переключаться между различными разделами приложения с помощью вкладок на нижней части экрана. Он используется для приложений, которые имеют несколько различных функциональных областей или разделов, например, социальные сети или музыкальные приложения.

2 Navigation Controller. Это стандартный паттерн навигации в iOS, который позволяет создавать иерархические списки и переходить между ними с помощью стека представлений. Он используется для приложений, которые имеют много уровней навигации, например, приложения для чтения новостей или каталогов товаров.

При запуске приложения пользователя встречает экран авторизации – LoginViewController. Если у пользователя еще нет аккаунта, то он может перейти на экран регистрации – RegisterVIewController. Если пользователь уже был авторизован, то повторно вводить данные не придется, а сразу откроется главный экран – HomeViewController.

Для определения авторизации пользователя проводится проверка на этапе загрузки LoginViewController’а. Результат проверки сохраняется в UserDefaults – это инструмент, предусмотренный для того, чтобы сохранять небольшие объемы информации, таких как настройки приложения, предпочтения пользователя или другие параметры, которые могут быть сохранены между сеансами работы приложения. Это удобный способ хранения данных, который не требует использования баз данных или файловой системы. UserDefaults может быть использован для сохранения простых типов данных, таких как числа, строки, булевы значения и массивы.

Сам по себе ViewController имеет несколько функций, которые вызываются поочередно, что образует в совокупности так называемый жизненный цикл ViewController’a. Перечень этих функций представлен на рисунке 3.10.



Жизненный цикл UIViewController начинается с инициализации, когда выделяется память для нового представления и настраиваются базовые параметры. Затем вызывается метод loadView, который загружает представление в поле view. Каждый UIViewController имеет поле view, которое является контейнером для отображения. Система загружает ранее созданное разработчиком представление и присваивает его полю view. Далее происходит вызов метода viewDidLoad, который добавляет основной контент и может начинать загрузку данных для отображения. После успешной загрузки представления происходит вызов метода viewWillAppear, который оповещает о том, что представление скоро будет показано пользователю, а метод viewDidAppear вызывается после появления представления на экране и может содержать логику появления клавиатуры для ввода данных. После того как пользователь производит переход к другому UIViewController’у или при исчезновении представления по любой другой причине, вызывается метод viewWillDisappear, который оповещает о скором исчезновении представления. Следующим этапом является вызов метода viewDidDissapear, который сигнализирует об исчезновении представления с экрана пользователя. Предпоследним этапом жизненного цикла UIViewController’а является вызов метода viewDidUnload, который в свою очередь сообщает о полной выгрузке представления из памяти системы. Финальный этап – деинициализация UIViewController’а, оповещающая о его полном освобождении из памяти. Методы класса UIViewController предоставляют отличную возможность для управления представлением на любом этапе жизненного цикла.

В приложении присутствует один Tab Bar, и именно он отвечает за глобальную навигацию в приложении. Всего в приложении представлено 4 «таба»:

1 HomeViewController. Это начальный экран приложения после авторизации пользователя. Здесь расположена актуальная афиша с постерами фильмов и кратким описанием. По нажатию на постер открывается экран с более подробной информацией о фильме: описание, оценка на сторонних ресурсах, отзывы пользователей, а также предоставлена возможность выбрать сеанс в кинотеатре.

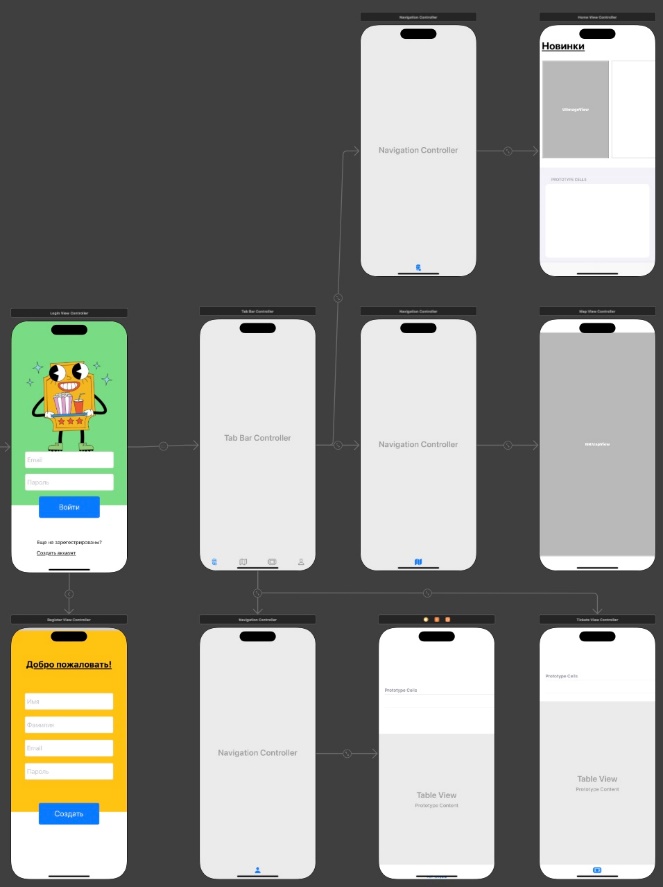
2 MapViewController. Это второй «таб» в приложении. На этом экране расположена карта с помеченными на ней кинотеатрами. При нажатии на метку кинотеатра открывается экран с дополнительной информацией. Здесь представлены фото кинотеатра, оценки и отзывы пользователей, а также контактная информация (адрес, телефон).

3 TicketsViewController. Третий «таб» куда поступают все заказанные билеты, производится их подсчет. Также здесь производится оплата.

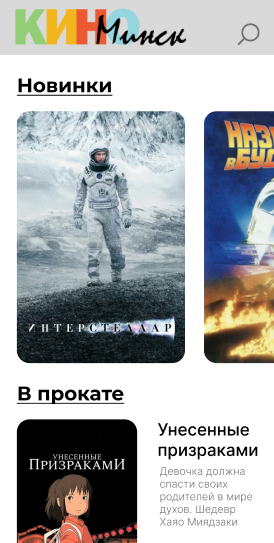
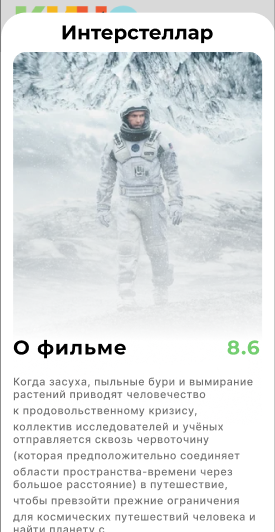
4 ProfileViewContoller. Последний четвертый «таб» в приложении, в котором расположена вся информация пользователя. Он может ее редактировать при помощи экрана настроек.

В каждый из вышеперечисленных «табов» вложен NavigationController, который позволяет удобно и быстро переходить между экранами.

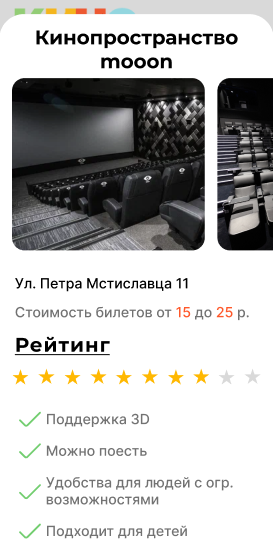
В общем виде навигация представлена на рисунке 3.11.



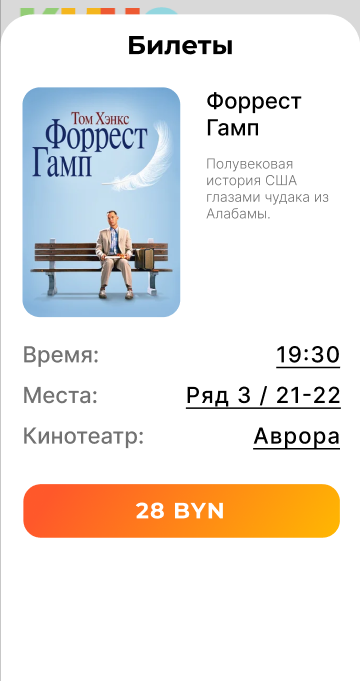
Интерфейс HomeViewController’а построен с помощью элементов UITableView (таблица) и UICollectionView (коллекция). Коллекция расположена в «шапке» таблицы, что способствует фокусу внимания пользователя на премьеры фильмов. В самой таблице представлены фильмы в прокате. По нажатию на постер любого фильма откроется экран с детальной информацией, календарем сеансов, списком кинотеатров, в котором этот фильм прокатывается, а также полем с отзывами пользователей. Деинициализация экрана с деталями фильма происходит по «смахиванию» вправо, так как вся эта конструкция включена в NavigationController.

MapViewController содержит элемент интерфейса MKMapView, который дает возможность просматривать карты. На карте отмечены кинотеатры города Минска, по нажатию на которые открывается экран с деталями о кинотеатре. Здесь представлены фотографии, вложенные в UICollectionView, адрес кинотеатра, диапазон цен на билеты, рейтинг и отзывы пользователей.



TicketsViewController содержит заголовок, UITableView для отображения забронированных билетов и считает сумму для оплаты. В ячейке таблицы отображаются все необходимые данные: название фильма, постер, кинотеатр, места, время сеанса. Забронированные билеты сохраняются в локальную базу данных с помощью CoreData и не пропадут после закрытия приложения. По нажатию на кнопку оплаты, все забронированные билеты переносятся в личный кабинет – в профиль пользователя.



ProfileViewController содержит информацию, которая может быть полезна для пользователя: оплаченные билеты, историю отзывов, просмотров, а также кнопку выхода из учетной записи.

3.3 Интеграция с базой данных и API кинотеатров

В приложении были использованы локальные и удаленные базы данных.

Локальная база данных используется для сохранения забронированных билетов в клиенте приложения. Она реализована с помощью фреймворка CoreData. CoreData - это фреймворк для управления объектами в iOS-приложении. Он позволяет сохранять, изменять и извлекать данные из базы данных SQLite или других источников данных. CoreData предоставляет мощный набор инструментов для работы с данными, включая запросы, отслеживание изменений, версионирование и многое другое.

CoreData может быть использовано в приложениях, где требуется хранение большого объема данных. Это может быть приложение для социальных сетей, приложение для учета финансов или любое другое приложение, где необходимо сохранять большое количество информации.

Кроме того, CoreData может быть использовано для создания сложных структур данных, таких как деревья или графы.

Удаленная база данных в приложении представлена сторонним сервисом FireBase. Firebase – это платформа, разработанная компанией Google, которая предоставляет набор инструментов для разработки мобильных приложений, веб-приложений и аналитики. Она позволяет разработчикам быстро создавать и развертывать приложения, используя облачные ресурсы, а также предоставляет инструменты для анализа и улучшения работы приложений. Также FireBase позволяет синхронизировать данные между несколькими клиентами.

Для хранения информации о пользователях и кинотеатрах был использован сервис FireBase. Структура баз данных представлена так:

– база данных кинотеатров: содержит информацию о кинотеатрах, включая название, адрес, контактную информацию, расписание сеансов, цены на билеты, а также фильмы в прокате;

– база данных сеансов: содержит информацию о сеансах в кинотеатрах, включая даты, время, цены, свободные и занятые места в зале;

– база данных пользователей: содержит информацию о пользователях, включая логин, пароль, имя, фамилию, историю заказов и активные заказы;

– база данных заказов: содержит информацию о заказах пользователей, включая номер заказа, дату и время заказа, фильм, кинотеатр, сеанс и выбранные места;

– база данных отзывов: содержит информацию об отзывах пользователей о фильмах и кинотеатрах, включая текст отзыва и рейтинг.

Что касается API кинотеатров и фильмов из актуальной афиши, получить доступ не представилось возможным, так как все они приватные. Поэтому в ходе разработки, пришлось самостоятельно заносить актуальные позиции в базы данных.

Для получения детальной информации о фильмах, такой как рейтинг, оценки и постеры, используется API «Кинопоиска».

Для составления HTTP запросов используется URLSession – это API, предоставляемое iOS для выполнения сетевых запросов и получения данных из сети. Для использования URLSession необходимо создать экземпляр URLSession с помощью конструктора, который принимает параметры конфигурации сессии. Затем можно создать объект URLRequest, который содержит информацию о запросе, такую как URL, метод запроса, заголовки и тело запроса. После создания объекта URLRequest его можно передать в метод URLSession.dataTask(with:), который создаст задачу для выполнения запроса. Этот метод возвращает объект URLSessionDataTask, который можно запустить методом resume(). Когда задача выполнена, URLSession вызовет свой делегат, где можно обработать полученные данные или ошибки. Делегат должен реализовать методы URLSession:dataTask:didReceiveData: и URLSession:task:didCompleteWithError:. В замыкании, переданном методу dataTask, обрабатываются полученные данные или ошибки. Если получена ошибка, выводится ее описание в консоль. Если данные получены успешно, они обрабатываются в блоке кода, который следует за проверкой наличия данных.

6 ТЕХНИКО–ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА

6.1 Характеристика программного средства, разрабатываемого для собственных нужд

Созданный проект представляет собой мобильное приложение для удобной покупки билетов в кинотеатрах с интегрированной системой персональных рекомендаций и программой лояльности, включающей интеллектуальный анализ пользовательских предпочтений.

Целью разработки данного мобильного приложения является оптимизация процесса выбора и покупки билетов в кино, предоставление персонализированных рекомендаций фильмов, а также повышение лояльности клиентов через внедрение бонусной системы и специальных предложений. Данная разработка может использоваться как индивидуальными посетителями кинотеатров, так и сетями кинотеатров для эффективного управления продажами и аналитики.

В отличие от существующих решений, данная разработка предлагает уникальное сочетание интуитивного интерфейса выбора мест, системы рекомендаций на основе искусственного интеллекта и программы лояльности, что позволяет значительно улучшить пользовательский опыт и увеличить частоту посещения кинотеатров.

Разработчиками данного мобильного приложения являются UI/UX дизайнер, мобильный разработчик и специалист по анализу данных.

Предполагаемая модель монетизации приложения основана на комиссии с каждого проданного билета. Дополнительным источником дохода станет реклама премьер фильмов, продажа попкорна и напитков через приложение с функцией предзаказа, а также партнерские программы с кинопрокатчиками и сетями кинотеатров.

В качестве каналов продвижения планируется использовать таргетированную рекламу в социальных сетях, контекстную рекламу и SEO-оптимизацию. Особый акцент будет сделан на коллаборации с кинокритиками и блогерами, обзоры новинок кино и интеграцию с популярными сервисами оценки фильмов. Важным направлением станет офлайн-реклама непосредственно в кинотеатрах и кросс-маркетинг с партнерами.

В результате ожидается создание современного мобильного приложения для киноманов, которое будет предоставлять полный цикл взаимодействия с кинотеатром - от выбора фильма на основе персональных предпочтений до покупки билетов и получения бонусов за посещение

6.2 Расчет инвестиций на разработку программного средства для собственных нужд

В процессе разработки было задействовано четыре исполнителя: UI/UX дизайнер, мобильный разработчик и специалист по анализу данных.

Основная заработная плата исполнителей рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (6.1) |

где *n* – количество исполнителей, занятых разработкой конкретного ПО;

Кпр – коэффициент, учитывающий процент премий (Кпр = 1,5);

Зч*i* – часовая заработная плата *i*-го исполнителя, р.;

*ti* – трудоёмкость работ, выполняемых *i*-м исполнителем, ч.

По данным Министерства социальной защиты и труда в месяц рабочее время составляет 168 часов.

Расчет основной заработной платы исполнителей представлен в таблице 6.1.

Таблица 6.1 – Затраты на основную заработную плату

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячная заработная плата, р. | Часовая заработная плата, р. | Трудоемкость работ, ч. | Итого, р. |
| UI/UX дизайнер | 1945,44 | 11,58 | 64 | 741,12 |
| Мобильный разработчик | 2256,24 | 13,43 | 168 | 2256,24 |
| Специалист по анализу данных | 1842,96 | 10,97 | 168 | 1842,96 |
| Итого | | | | 4840,32 |
| Премия (50% от заработной платы) | | | | 2420,16 |
| Всего затрат на основную заработную плату | | | | 7260,48 |

После подстановки значений в формулу (6.1) основная заработная плата составит:

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по формуле:

|  |  |
| --- | --- |
|  | (6.2) |

где Нд – норматив дополнительной заработной платы (20%);

Зо – затраты на основную заработную плату, р.

После подстановки значений в формулу (6.2) размер дополнительной заработной платы исполнителей составит:

Отчисления в фонд социальной защиты населения и на обязательное страхование определяются в соответствии с действующими законодательными актами по формуле:

|  |  |
| --- | --- |
|  | (6.3) |

где – норматив отчислений в фонд социальной защиты населения и на обязательное страхование, (34 + 0,6%).

После подстановки значений в формулу (6.3), отчисления в фонд социальной защиты населения и на обязательное страхование составят:

Прочие затраты рассчитываются по формуле:

|  |  |
| --- | --- |
|  | (6.4) |

где – норматив прочих затрат (30%).

После подстановки значений в формулу (6.4), прочие расходы составят:

Общая сумма затрат по всем статьям сметы () на разработку ПО рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (6.5) |

После подстановки значений в формулу (6.5), сумма затрат по всем статьям сметы составит:

6.3 Расчет экономического эффекта от использования программного средства для собственных нужд

Данное мобильное приложение позволяет сократить время на выбор фильмов и покупку билетов в кинотеатры, что значительно упрощает процесс планирования досуга и повышает качество кинематографического опыта.

Экономическим эффектом от использования приложения станет оптимизация расходов на посещение кинотеатров, получаемая за счет автоматизированной системы поиска выгодных предложений, участия в программе лояльности и накопления бонусов, а также минимизации рисков покупки билетов по завышенным ценам или на неудобные места в зале. Дополнительная экономия достигается за счет предложения пользователю специальных акций, скидок на предварительную покупку билетов и комбо-наборов с попкорном и напитками, а также возможности выбора оптимального времени сеанса с учетом текущего ценообразования и загруженности зала.

Экономия затрат на заработную плату при использовании программного средства в расчете на объем выполняемых работ определяется по формуле:

|  |  |
| --- | --- |
|  | (6.6) |

где – коэффициент, учитывающий процент премий, (1.5);

‒ трудоемкость выполнения работ сотрудниками до и после внедрения программного средства, ч;

‒ часовой оклад (часовая тарифная ставка) сотрудника, использующего программное средство, (18, р.);

– плановый объем работ, выполняемых сотрудником (50);

– норматив дополнительной заработной платы, (20 %);

– ставка отчислений от заработной платы, включаемых в себестоимость, (34,6%).

По формуле (6.6) экономия на заработной плате и начисления на заработную плату составят:

Экономическим эффектом при использовании программного средства является прирост чистой прибыли, полученной за счет экономии на текущих затратах компании и рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (6.7) |

Прирост текущих затрат, связанных с использованием ПО произойдет из-за необходимости сопровождения ПО.

Таким образом, вычисленный по формуле (6.7) прирост чистой прибыли составит:

Внедрение программного средства планируется в июле 2025 г.

6.4 Расчет показателей экономической эффективности разработки и использования программного средства в организации

Для расчета показателей эффективности инвестиций в разработку необходимо использовать дисконтирование путём умножения соответствующих результатов и затрат на коэффициент дисконтирования соответствующего года t, который определяется по формуле:

|  |  |
| --- | --- |
|  | (6.8) |

где *E*Н– норма дисконта (9,5%);

*t* – порядковый номер года, доходы и затраты которого приводятся к расчетному году (2025 – 1, 2026 – 2, 2027 – 3, 2028 – 4, 2029 – 5, 2030 – 6);

*tp*– расчетный год, к которому приводятся доходы и инвестиционные затраты (*tp* =1).

Коэффициенты дисконтирования по годам расчетного периода по формуле (6.8) составят:

Чистый дисконтированный доход рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (6.9) |

где *n* – расчётный период, лет;

*Рt* – результат (экономический эффект – прибыль или чистая прибыль), полученный в году t, р.;

*Зt* – затраты (инвестиции – затраты на разработку (модернизацию) или на приобретение и внедрение ПО) в году *t*, р.

Расчёт показателей эффективности инвестиций по разработке программного обеспечения представлен в таблице 6.2.

Tаблица 6.2 – Расчёт эффективности инвестиционного проекта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатель | Значение по годам расчетного периода | | | | | |
| 1-й год | 2-й год | 3-й год | 4-й год | 5-й год | 6-й год |
| Результат |  |  |  |  |  |  |
| Прирост чистой прибыли, руб. | 7415,83 | 7415,83 | 7415,83 | 7415,83 | 7415,83 | 7415,83 |
| Дисконтированный результат, руб. | 7415,83 | 6772,14 | 6184,80 | 5648,64 | 5158,45 | 4710,54 |
| Затраты |  |  |  |  |  |  |
| Инвестиции (затраты) в реализацию решения, руб. | 13905,27 | - | - | - | - | - |
| Дисконтированные инвестиции, руб. | 13905,27 | - | - | - | - | - |
| Чистый дисконтированный доход по годам, руб. | -6489,44 | 6772,14 | 6184,80 | 5648,64 | 5158,45 | 4710,54 |
| Чистый дисконтированный доход нарастающим итогом | -6489,44 | 282,7 | 6467,50 | 12116,14 | 17274,59 | 21985,13 |
| Коэффициент дисконтирования | 1 | 0,9132 | 0,834 | 0,7617 | 0,6956 | 0,6352 |

Рентабельность инвестиций определяется по формуле:

|  |  |
| --- | --- |
| , | (6.10) |

где Пчср – среднегодовая величина чистой прибыли за расчётный период, р., которая определяется по формуле:

|  |  |
| --- | --- |
| , | (6.11) |

где – чистая прибыль, полученная в году t, р.

Подставив значения в формулу (6.11), получим среднегодовую величину чистой прибыли за расчётный период:

Подстановкой числовых значений в формулу (6.10) вычислим рентабельность инвестиций:

6.5 Выводы по результатам расчетов

В результате технико-экономического обоснования разработки и использования приложения были получены следующие данные:

– чистый дисконтированный доход за шесть лет использования программного средства составит 21985,13 р.;

– затраты на разработку программного продукта окупятся на третий год использования;

– рентабельность инвестиций составляет 53,33% в год.

В результате расчета технико-экономического обоснования инвестиций в разработку мобильного приложения для продажи билетов в кинотеатрах можно сделать вывод, что разработка считается эффективной. Это объясняется тем, что период возврата инвестиций определяется третьим годом реализации программного продукта.

ЗАКЛЮЧЕНИЕ

В ходе разработки приложения по поиску и бронированию мест в кинотеатры была проделана большая работа по анализу, планированию, дизайну, проектированию и разработке.

В первой главе данной дипломной работы были проанализированы существующие мобильные приложения на схожую тему или те, которые включают поиск и бронирование билетов в кинотеатры в отдельный блок. Благодаря отзывам пользователей в AppStore, у существующих приложений были выявлены плюсы, которые были внедрены в приложение, и минусы, которые были исправлены.

Во второй главе был составлен план или «Roadmap», который во многом упростил создание приложения. Четкий план действий позволял фокусироваться на отдельных задачах, не отвлекаясь на другие блоки. Каждая задача разбивалась на более мелкие, что в итоге приводило к продуктивной работе. Был разработан дизайн интерфейса, с ориентировкой на лучшие решения в этой области. Был проведен глубокий анализ в выборе лучших навигационных шаблонов и интуитивно понятного пользовательского интерфейса. Выбор стиля, приятного пользователю, располагает к приложению, что увеличит месячный оборот.

Третья глава дипломной работы посвящена проектированию приложения с использованием анализа уже известных архитектурных паттернов и выбран наилучший вариант шаблона – MVVM. Для приложения такого масштаба выбранный шаблон не уступает по возможностям в расширяемости и тестируемости. В то же время он не усложняет разработку программных реализаций такого типа. Выбран также современный язык программирования Swift и актуальный фреймворк UIKit. Выбор данного фреймворка позволил интегрироваться в приложение FireBaseSDK для дальнейшей работы с удаленной базой данных. Также проведена работа с сетью, а именно создание и распределение API запросов.