МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПРОСМОТРА РАСПИСАНИЯ ЗАНЯТИЙ ВУЗОВ ПОД УСТРОЙСТВА ЭКОСИСТЕМЫ APPLE

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Архипова Кирилла Олеговича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2025

ВВЕДЕНИЕ

С развитием современных технологий и растущей популярностью умных устройств, мобильных приложений и программного обеспечения улучшение его качества и эффективности становится критически важным. Мобильные устройства и гаджеты, например, такие как iPhone и Apple Watch, позволяют людям экономить время и упрощать выполнение повседневных задач благодаря удобным интерфейсам и функционалу. Например, взаимодействие с устройствами стало интуитивно понятным: звонки можно принимать с помощью одного касания часов или кнопки на наушниках, а данные синхронизируются между всеми устройствами пользователя.

Такие достижения возможны благодаря использованию современных языков программирования и инструментов разработки, которые упрощают создание надёжных и масштабируемых приложений. Одним из ключевых языков для экосистемы Apple является Swift, обеспечивающий разработчиков мощными и гибкими средствами для реализации сложных задач, включая построение интерфейсов, управление данными и синхронизацию между устройствами. Для создания полнофункционального и удобного приложения разработчику зачастую требуется изучить и использовать широкий спектр библиотек и фреймворков. Например, для построения интуитивно понятных интерфейсов используется большое количество различных библиотек, причем каждая из них используется под определенную платформу и имеет свои преимущества и недостатки. Также, многие современные приложения предоставляют набор виджетов, которые позволяют получать быстрый доступ к необходимой информации и для таких случаев тоже должны использоваться отдельные фреймворки. Известно, что важную роль в разработке любого приложения играют базы данных, которые обеспечивают хранение, доступ и управление данными, как в масштабе небольших приложений, так и массивных с несколькими миллионами строк информации. А с появлением интернета и его технологий, данные начали передаваться между устройствами по сети, а также появилась возможность получать доступ к информации в интернете с мобильных устройств и обрабатывать текстовые данные, получаемые из HTML-страниц. Для этого в язык программирования интегрируются инструменты, библиотеки, специально адаптированные для работы со Swift. Они существенно расширяют возможности разработчиков, позволяя создавать приложения, которые удовлетворяют требованиям современного

пользователя.

Целью настоящей работы является создание приложения и виджетов для него на языке программирования Swift для устройств на базе операционных систем iOS, iPadOS, macOS и watchOS с использованием технологии баз данных SwiftData, а также поддержкой парсинга расписания с нескольких сайтов вузов.

Поставлены несколько задач:

- рассмотреть языки разработки приложений под устройства Apple;
- рассмотреть язык программирования Swift;
- проанализировать имеющиеся инструменты технологий баз данных для приложений на платформе iOS;
- рассмотреть технологии парсинга для приложений на платформе iOS;
- рассмотреть фреймворки SwiftUi, SwiftData, WidgetKit, SwiftSoup, WatchConnectivity;
- рассмотреть другие вспомогательные средства разработки приложений для iOS;
- создать приложение, способное загружать (парсить), хранить, удалять и просматривать расписание пар в вузах, для iOS, iPadOS, macOS и watchOS на языке Swift, а также дополнить его виджетом.

Структура и объём работы. Бакалаврская работа состоит из введения, 3 разделов, заключения, списка использованных источников и 4 приложений. Общий объем работы – 92 страницы, из них 65 страниц – основное содержание, включая 43 рисунка и 1 таблицу, CD диск в качестве приложения, список использованных источников информации – 21 наименование.

Первый раздел «Основные технологии разработки приложений» посвящен описанию основных определений используемых в работе, а также теоретической информации, касающейся инструментов разработки.

До 2014 года для разработки приложений на устройства Apple использовался Objective-C, который почти не менялся с момента его появления в 1980-х. В 2014 году Apple представила новый язык программирования — Swift. Теперь для разработки приложений под устройства Apple в основном используется язык Swift и его фреймворки, но имеются и иные возможности вести разработку, например, с помощью других языков на таких фреймворках как Kotlin Multiplatform Mobile, React Native, Flutter, Xamarin, BeeWare и Kivy.

Одним из важных и необходимых инструментов современной разработки

являются фреймворки и интегрированные среды разработки. Первые расширяют инструментарий разработки, вторые ответственны за редактирование кода, подсказки в процессе его написания, компиляцию в готовое решение, поиск ошибоки и отладку.

SwiftUi — это фреймворк, который является основой приложения, поскольку помогает создать и настроить пользовательский интерфейс и реакции на взаимодействие с пользователем. SwiftUI был первоначально представлен в июне 2019 года на презентации WWDC19 и предназначался для полной замены или небольшого дополнения, как на то время казалось и так достаточно простого UIKit.

Основные типы баз данных включают: реляционные базы данных, с языком для управления и манипуляции данными в реляционных базах данных SQL, NoSQL базы данных, предлагающие альтернативные подходы к хранению, организации и обработке данных и NewSQL базы данных сочетающие свойства реляционных баз данных с масштабируемостью NoSQL решений. Для iOS приложений существуют следующие решения баз данных SQLite, реляционная база данных, встроенная в iOS и используемая через API фреймворка Core Data, SwiftData, новая библиотека, выполняющая те же функции что и CoreData, но чуть проще и быстрее, Realm, имеющая собственный высокоэффективный движок и облачные решения Firebase Realtime Database и Firestore.

Парсинг HTML — это процесс извлечения и обработки данных из вебстраниц, представленных в формате HTML. Для эффективного парсинга используются инструменты, такие как CSS-селекторы и XPath. CSS-селекторы — это механизм, который позволяет находить элементы HTML-документа, основываясь на их атрибутах. XPath (XML Path Language) — это язык запросов для выборки элементов из XML или HTML-документов. Он позволяет находить элементы, используя их иерархическое расположение в документе. Для работа с HTML-документами на Swift представлены библиотеки: Kanna, SwiftSoup, Fuzi. Также существуют устаревшие Hpple, NDHpple и Ji.

Второй раздел «Программные средства» посвящен детальному описанию кодовой базы инструментов разработки.

Мультиплатформенный и современный язык программирования Swift, использует синтаксис без точки с запятой (;), не требует явного указания типов, сохраняя строгую типизацию. Имеет поддержку выводимых типов, а также тип

данных Tuple, который содержит множество данных разных типов. Присутствуют опциональные типы данных, пользовательские библиотеки и фреймворки.

Интегрированная среда разработки Xcode включает в себя редактор кода, инструменты для отладки, тестирования и анализа производительности, а также симулятор для тестирования приложений. Он поддерживает не только программирование на Swift, но и на других языках, например Java, C, Python и C++, имеет связь с Git, функцию предсказания кода с помощью генеративной модели и занимает меньше 15 ГБ.

Фреймворк SwiftUi содержит в себе множество различных элементов, отвечающих за построение пользовательского интерфейса и взаимодействие с ним. Например классы, операторы, методы, объекты состояний, команды сохранения и инициализации данных.

Фреймворк SwiftData интегрирует в себя более масштабную и устоявшуюся технологию, CoreData. Он позволяет создавать пользовательские объекты, определять, как они связаны друг с другом, извлекать их с помощью фильтрации и сортировки и синхронизировать с iCloud. Чтобы воспользоваться фреймворком необходимо импортировать его в файлы, где используются его компоненты с помощью $\mathrm{import\ SwiftData}$. Далее необходимо создать модель данных, являющуюся классом в Swift. В модели можно использовать как обычные типы Swift, так и типы, усложненные структурами и классами.

Для взаимодействия с фреймворком и его компонентами можно пользоваться такими словами-аннотациями как $\mathrm{@Model}$, $\mathrm{@Attribute(.unique)}$, $\mathrm{@Query}$, переменной окружения $\mathrm{ModelContext()}$, а также ее методами $\mathrm{insert()}$ и $\mathrm{delete()}$.

Фреймворк SwiftSoup позволяет получить доступ к элементам через метод $\mathrm{select()}$ или получит текстовое содержимое с помощью $\mathrm{text()}$, а для работы с атрибутами используется метод $\mathrm{attr()}$. Библиотека также предоставляет функционал для навигации по иерархии DOM с помощью $\mathrm{parent()}$, $\mathrm{children()}$ и $\mathrm{siblingElements()}$.

Фреймворк WatchConnectivity позволяет взаимодействовать с умными часами с помощью своих команд и протоколов. Таковыми являются протокол WCSessionDelegate, методы session, sessionDidBecomeInactive, sessionDidDeactivate и класс WCSession. Существуют особенные передаваемые данных. Для взаимодействия между часами и смартфоном фреймворк предлагает 3 различных способа, каждый из которых предназначен для определенного случая обмена

данными: ApplicationContext, UserInfo, SendMessage.

WidgetKit представляет собой фреймворк для разработки виджетов, но также используется для настройки взаимодействия смартфона с его виджетами. Виджеты позволяют отображать ключевую информацию из приложений на домашнем экране и экране блокировки устройства, обеспечивая пользователям быстрый доступ к данным без необходимости открывать само приложение. Виджеты имеют свои «семьи», с определенными типами и размерностями. Для управления обновлением данных применяется `TimelineProvider`. Виджеты могут отображать как статическую, так и динамически изменяющуюся информацию, а также изменять отображаемую информацию в зависимости от пользовательской настройки виджета. Для высокой энергоэффективности механизм изменения виджетов устроен как последовательность картинок. И у них имеются разные политики обновления: `.atEnd`, `.after(date:)` и `.never`. Обновление может вызываться методом одного из классов WidgetKit: `WidgetCenter.shared.reloadTimelines(ofKind:)` или `WidgetCenter.shared.reloadAllTimelines()`.

Третий раздел «Описание разработки приложения» посвящен проектированию и реализации приложения, его компонентов и расширений. Описано проектирование модели данных, связи компонентов приложения и их взаимодействия. Установлен поэтапный план разработки приложений.

Изначально были установлены проанализированы требования и сценарии использования основного приложения, основная идея проекта, смысл и задача приложения. А именно, загрузка и отображение расписания, как в основном приложении, так и на часах и виджетах. Была детализирована структура расписания, ее свойства, основанные на ключевых функциональных возможностях.

Одной из задач было создание логики, которая смогла бы постоянно поддерживать актуальную информацию о расписании в приложении на часах и в виджетах. Для отправки данных на часы был выбран подход с использованием ApplicationContext, для синхронизации данных с виджетами реализуется подход с использованием локального хранилища. Каждый подход реализуется отдельным объектом, ответственным за те или иные действия связанные с компаньоном или виджетами. Для обобщения идеи синхронизации и избежания дублирования кода с похожей логикой был создан отдельный объект.

Для парсинга сайтов по конкретным факультетам, группам и учителям

были созданы соответствующие структуры. Для поддержки нескольких вузов было реализовано общее решение — единый модуль. Парсеры написаны с использованием фреймворка SwiftSoup.

Были спроектированы экраны пользовательского интерфейса основного приложения, приложения-компаньона watchOS с учетом особенностей Apple Watch. Особое внимание уделено проектированию пользовательского интерфейса виджетов, поскольку они должны отображать только самую необходимую информацию. Были проанализированы сценарии использования виджетов и спрототипированы их дизайн и механизм наполнения данными.

Для виджетов на домашнем экране с соответствующими им семьями был спроектирован вариант, где наполнение представляет из себя строчки, в каждой из которых располагается важная информация о занятии в какой либо момент времени. Эти виджеты были разделены на 2 типа, соответствуя специфике своего назначения и функционала.

Виджеты на экране блокировки и их семьи также поделены на несколько типов со своим назначением. Один из виджетов прямоугольной версии перенял наполнение от виджета на домашнем экране. Также спроектированы различные версии круглого и прямоугольного типов виджетов.

Создание интерфейса основного приложения включало в себя создание основного View с элементами SwiftUI и наполнение экранами настройки и расписания. Экран настроек отвечающий за соответствующий функционал был наделен объектом SettingsManager, отвечающим за хранение настроек, а также был использован модификатор onChange, позволяющий реагировать на изменения в объектах или визуальных элементах. Экран расписания представлен TabView и наполнен днями недели, выбранным днем и расписанием на этот день.

Несколько пар в одно и то же время обобщены визуальным элементом, который содержит такие параметры как информация о паре, закрепленная пара и активная отображаемая пара, которые передаются во View из вне. Представлена элементом TabView с которым можно выполнять прямое взаимодействие По удержанию пары вызывается контекстное меню и действия закрепить пару, открыть детальное отображение с возможностью изменить параметры, скрыть пару и открыть настройки видимости занятий.

Элемент пар одного дня также получает информацию о парах в этот день

из вне. Представлен прокручиваемым списком из нескольких идущих подряд элементов пар в одно время.

Для обеспечения удобного выбора объектов было реализовано универсальное шаблонное представление — SearchablePickerView. Оно реализовывает возможнсть производить выбор и поиск в списке похожих данных.

С целью повышения удобства первичного взаимодействия с приложением был реализован экран с обучающим материалом для ознакомления с приложением. Данная функция реализована в виде последовательности полноэкранных представлений, каждое из которых акцентирует внимание на ключевых возможностях и особенностях приложения. Каждое представление содержит различное наполнение с интерактивными возможностями, присутствуют механизмы переключения экранов с помощью кнопок «Далее» и «Назад», а также возможность пропустить обучение. После прохождения в локальном хранилище пользовательских данных сохраняется флаг, предотвращающий повторное отображение обучающего интерфейса при последующих запусках приложения.

Для хранения данных произведена интеграция и настройка фреймворка SwiftData. Расписаны поэтапные шаги с описанием объектов, методов и иных элементов кода.

Создание интерфейса приложения-компаньона watchOS состоит в основном из портирования и адаптировани экранов iOS под watchOS с внесением изменений и заменой некоторых элементов. Так например, для представления занятий в одно время была изменена логика, чтобы отображалось только одно актуальное занятие.

По спроектированым прототипам были созданы виджеты. Они поделены на несколько типов и для каждого из них были созданы поставщики данных по времени. Разделение на типы и поставщики объясняется энергоэффективностью и более чистой архитектурой без лишних данных. С учетом спроектированного интерфейса, было создано отдельное универсальное шаблонное представление строки занятия. Были реализованы различные виды виджетов домашнего экрана и экрана блокировки. Для ручной настройки содержимого некоторых виджетов был создан AppIntent. AppIntent представляет собой механизм настройки виджетов, позволяющий пользователям персонализировать отображаемую информацию.

Произведена демонстрация работы приложения, его функционала и его

версий на других платформах, а также его расширений в виде виджета и приложения компаньона в условиях работы на симуляторе.

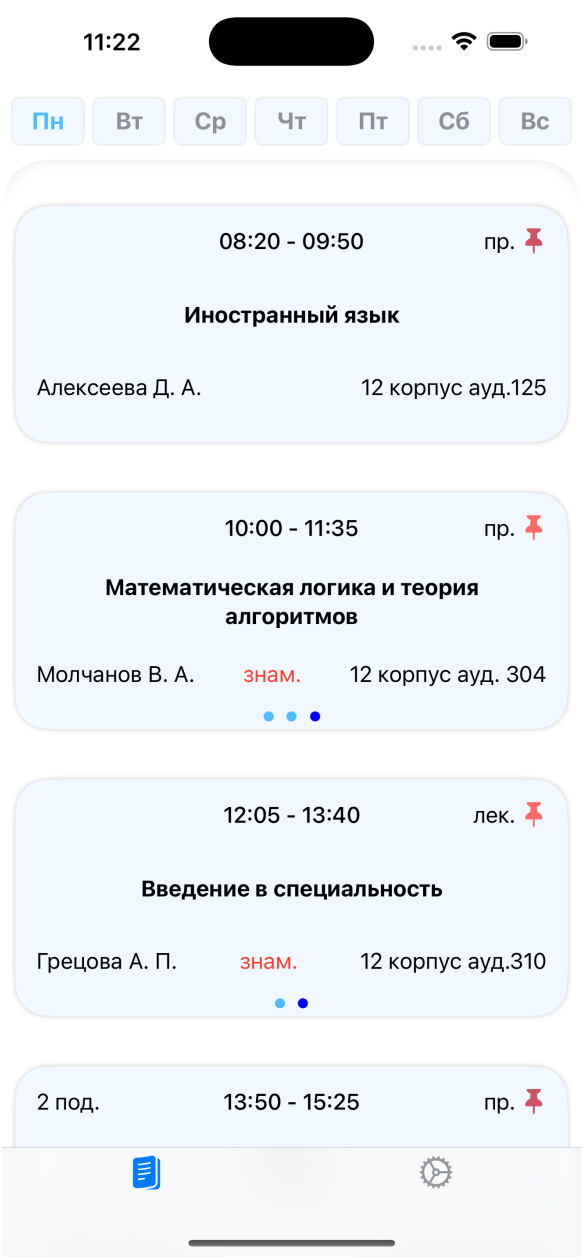

Рисунок 1 – Вид расписания основного приложения

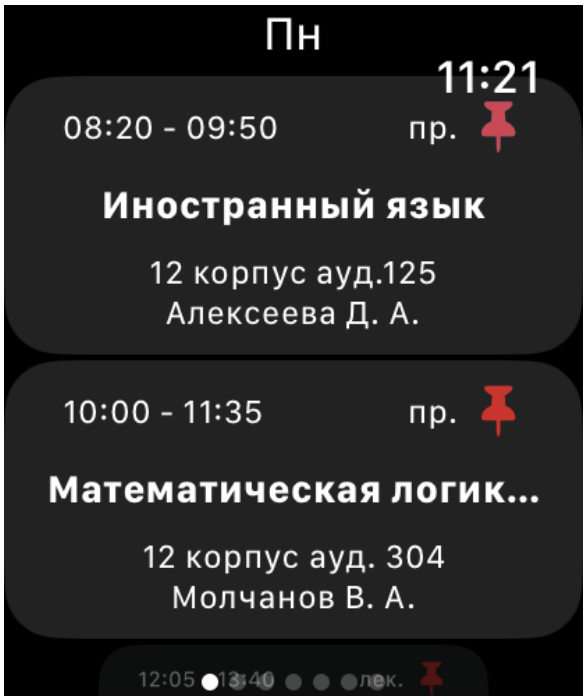

Рисунок 2 – Вид расписания на экране часов

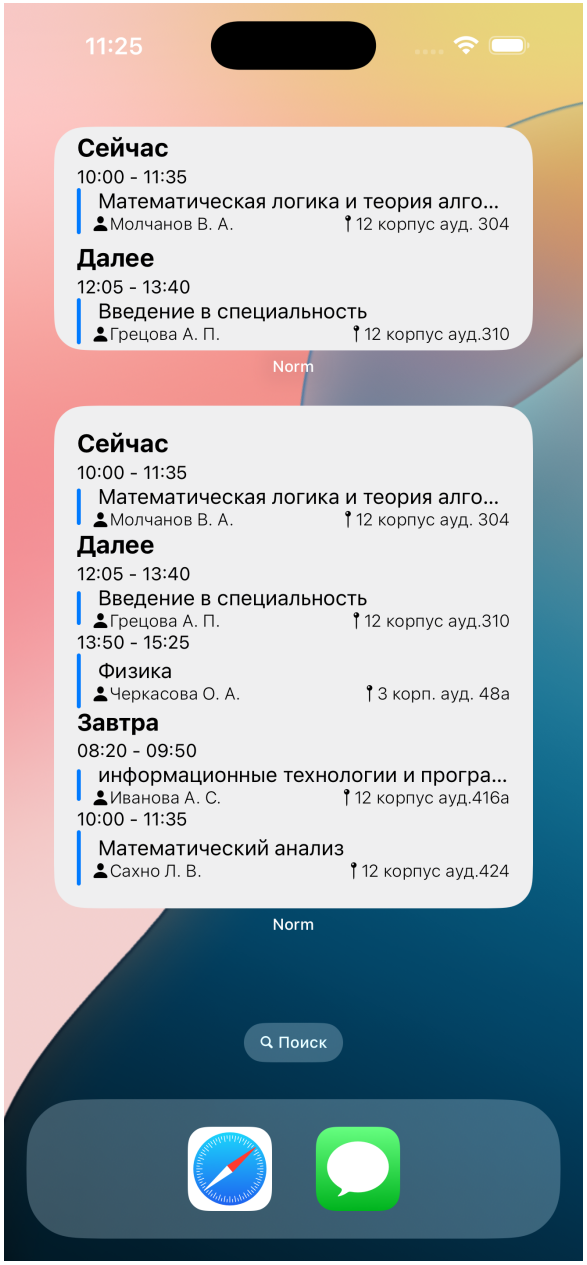

Рисунок 3 – Вид расписания на виджетах домашнего экрана

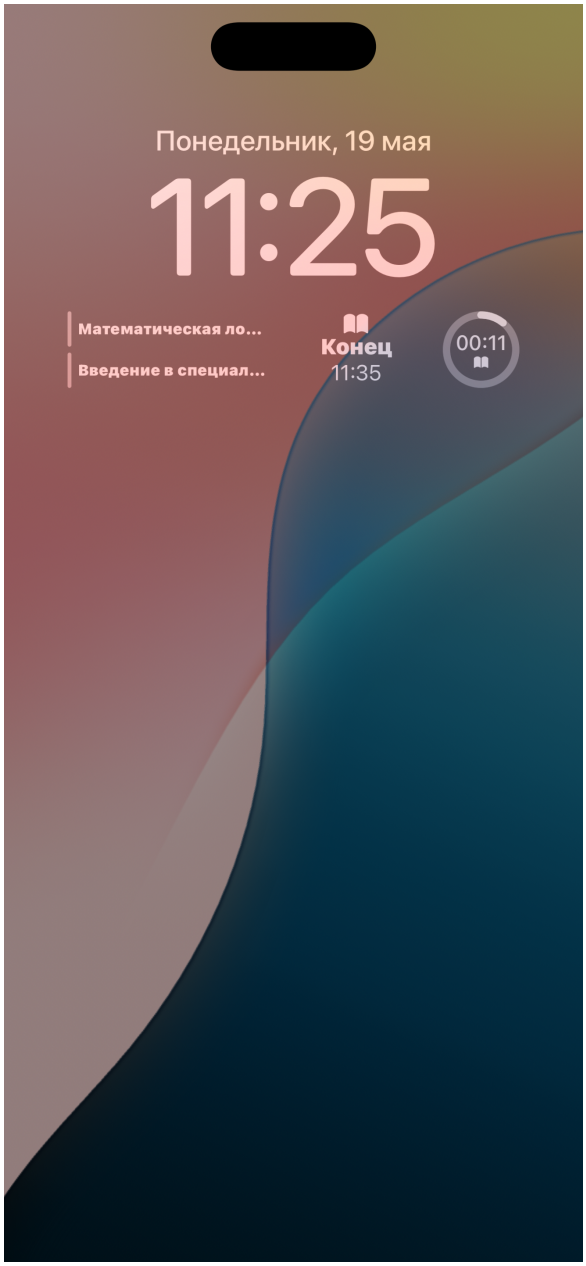

Рисунок 4 – Вид расписания на виджетах экрана блокировки

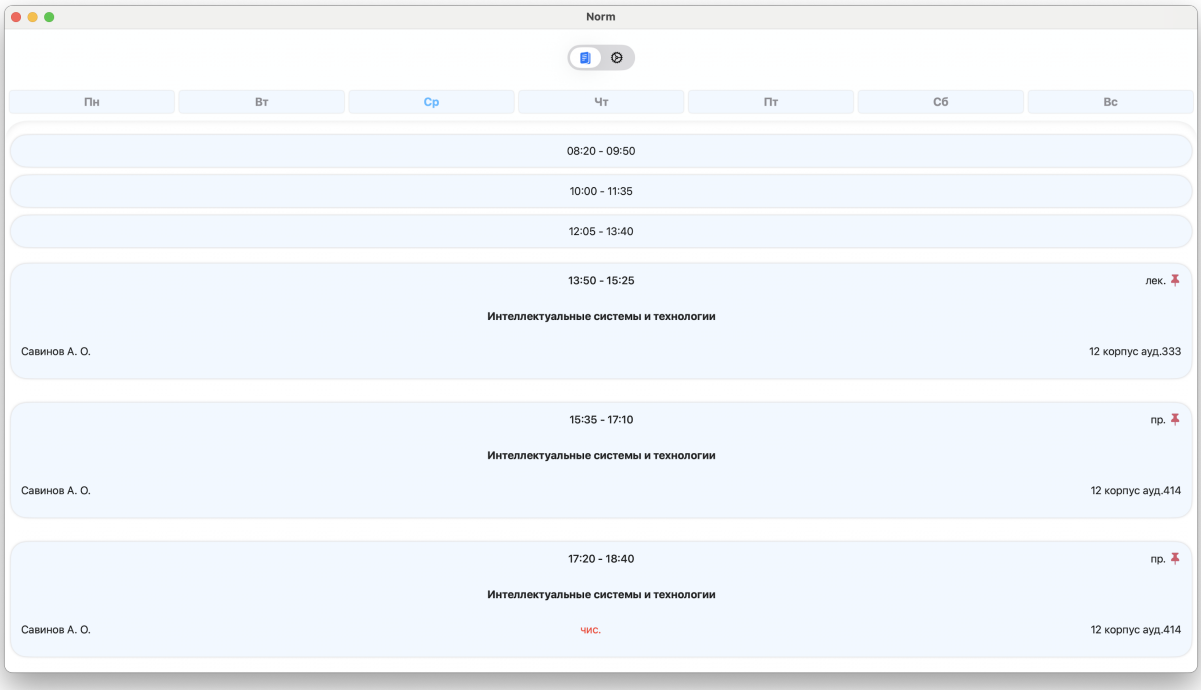

Рисунок 5 – Вид основного приложения на macOS

Рисунок 6 – Вид основного приложения на iPadOS (вертикальная ориентация)

ЗАКЛЮЧЕНИЕ

В рамках данной дипломной работы было разработано современное мультиплатформенное приложение для просмотра и управления учебным расписанием с использованием актуальных технологий экосистемы Apple. Проект представляет собой комплексное решение, включающее мобильное и десктоп приложение, систему виджетов для домашнего экрана и экрана блокировки, а также приложение-компаньон для Apple Watch.

Разработка осуществлялась на языке Swift с применением таких фреймворков как SwiftUI для построения интерфейсов, SwiftData для работы с локальными данными, WidgetKit для реализации виджетов и WatchConnectivity для синхронизации между устройствами. Особое внимание было уделено обеспечению стабильной и понятной пользователю работы парсера расписания на основе SwiftSoup.

Полученное в результате работы приложение демонстрирует эффективное сочетание современных технологий разработки для платформ экосистемы Apple. Реализованные функции позволяют пользователям удобно отслеживать учебное расписание, изменять его параметры под свои требования и иметь быстрый доступ к актуальной информации через систему виджетов и приложение для часов.

Проведенная работа подтверждает возможность создания полноценных образовательных решений с использованием рассмотренных технологий. Разработанное приложение соответствует современным требованиям к мобильным приложениям в части производительности, удобства использования и функциональности. Полученные результаты могут служить основой для дальнейшего развития проекта и адаптации под различные образовательные учреждения.