

МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Кафедра

вычислительной техники

МОиПЭВМ

Направление подготовки

09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка Android-приложения учёта рабочего времени с применением

Jetpack Compose

Студент

(подпись, дата)

Колягин Владислав Сергеевич

(ФИО полностью)

Руководитель

(подпись, дата)

Гурьянов Л.В.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

(фамилия, инициалы)

Пенза, 2024

МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Кафедра

вычислительной техники

МОиПЭВМ

«Утверждаю»

Заведующий кафедрой

_____ А.Ю. Козлов «__»

_____ 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Колягин Владислав Сергеевич гр. 20ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Разработка Android-приложения учёта рабочего времени с применением Jetpack Compose
Тема утверждена приказом ПГУ № _____ от «__» _____ 2024 г.
3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Гурьянов Л.В.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
 - 1) Назначение: Автоматизация учёта рабочего времени
 - 2) Среда выполнения: Android 8.0
Языковые средства разработки: Kotlin
Инструментальные средства: Android Studio, draw.io
Технологии разработки: ООП, Чистая архитектура
 - 3) Основные функции разрабатываемого программного обеспечения:
 - а) создание рабочего графика;
 - б) создание списка перерывов;
 - в) отслеживание времени работы и перерывов;
 - г) вывод статистики рабочего времени сотрудников;
 - д) создание напоминаний для сотрудника.

5. Перечень подлежащих разработке вопросов:

- 1) Анализ предметной области и постановка задачи;
- 2) Планирование разработки и расчет бюджета;
- 3) Разработка требований и проектирование;
- 4) Реализация;
- 5) Контроль качества программного обеспечения;
- 6) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10%	01.03.2024 – 07.03.2024	
2	Планирование разработки и расчет бюджета	10%	08.03.2024 – 15.03.2024	
3	Проектирование программных средств	10%	15.03.2024 – 02.04.2024	
4	Реализация программных средств	30%	03.04.2024 – 02.05.2024	
5	Тестирование программных средств	20%	03.05.2024 – 18.05.2024	
6	Оформление пояснительной записки	20%	19.05.2024 – 01.06.2024	

Дата выдачи задания «22» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить

Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 77 страниц, 13 рисунков, 4 таблицы, 13 использованных источников и 2 приложения.

УЧЕТ РАБОЧЕГО ВРЕМЕНИ, ANDROID, JETPACK COMPOSE, UI, UX, ЧИСТАЯ АРХИТЕКТУРА, MVVM, MVI, UML.

Объект исследования – Android-приложение для учета рабочего времени WorkTracker, реализованное с использованием декларативного инструмента для создания графического интерфейса Jetpack Compose.

Цель выпускной квалификационной работы – разработать Android-приложение с использованием часто применяемых библиотек, а также архитектурой, следующей принципам чистой архитектуры и проверить применимость Jetpack Compose в рамках разработанной системы.

Технологии разработки – разработка проводилась на языке программирования Kotlin с использованием Jetpack Compose, Compose Destinations, Hilt, Coroutines и Room. В качестве среды разработки использовался редактор программного кода Android Studio.

Среда исполнения – Android 8.0 и выше.

Результаты работы – в ходе выполнения выпускной квалификационной работы было разработано Android-приложение WorkTracker. Был сделан вывод о том, что Jetpack Compose может быть использован в качестве альтернативы для View.

					ПГУ 09.03.04 – 08ВР201.08 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Колягин В.С.			Разработка Android-приложения учёта рабочего времени с применением Jetpack Compose	Лит.	Лист	Листов
Пров.		Гурьянов Л.В.					4	77
Н. Контр.		Такташкин Д.В.			Пояснительная записка	Группа 20ВП1		
Утв.		Козлов А.Ю.						

Содержание

Введение.....	6
1 Анализ предметной области и постановка задачи на разработку.....	7
1.1 Анализ предметной области разработки	7
1.2 Обзор существующих систем	7
1.3 Постановка задачи на разработку	9
1.4 Планирование разработки и расчет бюджета	9
2 Анализ требований на разработку программных средств	13
2.1 Анализ функциональных требований.....	13
2.2 Требования к архитектуре программных средств	15
2.3 Требования к пользовательскому интерфейсу программных средств...	17
2.4 Обоснование выбора технологий и средств разработки.....	18
2.5 Требования к аппаратно-программному обеспечению.....	19
3 Разработка программных средств	20
3.1 Разработка архитектуры программных средств	20
3.2 Разработка уровня представления.....	21
3.3 Разработка уровня домена.....	23
3.4 Разработка уровня данных	24
4 Тестирование приложения WorkTracker.....	26
4.1 Выбор методологии тестирования	26
4.2 Функциональное тестирование	26
4.3 Нефункциональное тестирование	33
4.3.1 Unit тестирование	33
4.3.2 Метрика SLOC	37
4.3.3 Sonar	39
Заключение	40
Список использованных источников	41
Приложение А (справочное) Глоссарий	43
Приложение Б (рекомендуемое) Код программы.....	45

Введение

В современном мире все больше людей начинает пользоваться мобильными телефонами. Большую часть рынка операционных систем для мобильных устройств занимают две системы: Android и IOS. С ростом популярности мобильных устройств увеличивается и количество мобильных приложений.

При проектировании проекта Android разработчикам приходится решать задачу выбора архитектуры и используемых библиотек. До 28 июля 2021 года в каждом приложении использовались View, однако после разработчикам стала доступна библиотека Jetpack Compose, являющаяся альтернативой View.

В рамках данной дипломной работы будет рассмотрен пример использования Jetpack Compose при разработке мобильного приложения отслеживания рабочего времени. В ходе разработки будут достигнуты следующие цели:

- провести анализ предметной области;
- провести анализ конкурентов;
- провести расчет бюджета проекта;
- провести анализ требований;
- спроектировать архитектуру приложения;
- спроектировать пользовательский интерфейс приложения;
- выбрать технологические средства;
- реализовать приложение;
- провести тестирование приложения.

Поставленные цели предполагают разработку полноценного ПО, что позволяет проверить возможность внедрения нового инструмента в процесс разработки приложения.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области разработки

Предметной областью является отслеживание рабочего времени сотрудника компании. Рабочее состояние может быть четырех типов: работа, пауза, обед, а также состояние вне рабочего времени. Для сотрудника важно знать сколько он находится в каждом из состояний для корректного составления отчетов о проделанной работе. Отчёты могут быть как ежедневными, так и ежемесячными, поэтому необходимо предоставить возможность просматривать суммарное время в каждом из состояний для выбранных промежутков времени.

Состояние паузы можно разделить на два под состояния: запланированная пауза и незапланированная. К запланированным паузам относятся промежутки, которые сотрудник заранее запланировал, а к незапланированным относится время, в которое сотрудник не должен был находиться в состоянии паузы, например, при слишком долгом нахождении в состоянии обеда.

Отслеживать текущее рабочее состояние необходимо каждый рабочий день, поэтому необходимо максимально упростить этот процесс. Также необходимы напоминания о необходимости актуализировать текущее состояние.

Некоторые предприятия блокируют доступ в сеть Интернет, поэтому сотрудник должен иметь возможность изменять текущее состояние без доступа к сети.

Исходя из описанных выше требований, можно сделать вывод, что самым подходящим форматом ПО является мобильное приложение.

1.2 Обзор существующих систем

Перед началом разработки необходимо проанализировать существующие системы, выявить их сильные и слабые стороны. Эта информация поможет при дальнейшем планировании разработки системы.

Исследуя существующие аналоги, был сделан вывод о том, что все приложения занимают небольшую часть рынка. При отсутствии явного лидера

среди конкурентов, разработанное приложение может занять часть рынка благодаря своим отличиям от уже существующих систем.

Прямыми конкурентами разрабатываемому ПО являются «Учёт рабочего времени. WorkTimeManager», «Workhours – учёт рабочего времени» и «График рабочих смен – Alfa Work. Календарь смен». Данные аналоги имеют наибольшую популярность в магазине приложений RuStore.

WorkTimeManager имеет следующую функциональность:

- позволяет отслеживать рабочее время за день и месяц;
- позволяет отслеживать время перерывов за день;
- позволяет отслеживать дни на больничном, дни в отпуске, а также дни в командировке;
- позволяет подсчитывать зарплату;
- позволяет составлять список дел.

Слабыми сторонами данного конкурента является отсутствие напоминаний, отсутствие ведения статистики перерывов более чем за день, ограничение на просмотр статистики только днём и месяцем.

Сильными сторонами данного конкурента является более подробная статистика пропущенных дней, наличие возможности подсчитывать зарплату и возможности составлять список дел.

Workhours позволяет:

- фиксировать рабочее время за день;
- подсчитывать заработок;
- указывать отпускные и больничные;
- работать с проектами и заданиями.

Слабыми сторонами данного приложения является отсутствие отслеживания перерывов, статистика суммарного времени работы доступна только за день или месяц, отсутствуют напоминания.

Сильными сторонами являются возможность подсчитывать заработок и работать с проектами и задачами.

Alfa Work обладает следующей функциональностью:

- обладает несколькими цветовыми темами;
- позволяет планировать рабочие смены на месяц;
- подсчитывает заработок;
- позволяет создавать заметки;
- показывает диаграмму, отражающую количество рабочих дней и дней отдыха;
- имеет встроенный будильник.

Слабыми сторонами проекта являются отсутствие возможности фиксировать фактическое время работы и перерывы, отсутствует возможность просмотра статистики за произвольный период времени.

Сильными сторонами проекта являются подсчёт заработка, возможность смены темы, наличие диаграммы и будильник.

Исходя из выше сказанного, можно сделать вывод, что разрабатываемое приложение имеет особенности, которые позволяют ему быть конкурентноспособным на рынке приложений.

1.3 Постановка задачи на разработку

Целью разработки приложения является автоматизация подсчета времени отдыха и работы сотрудника. Помимо этого, приложение должно показывать напоминания для сотрудника и позволять отслеживать общее время работы за необходимые промежутки времени. Приложение должно иметь интуитивно понятный интерфейс, позволяющий изменять текущее рабочее состояние с минимальными временными затратами.

1.4 Планирование разработки и расчет бюджета

Для создания нового проекта необходимо спланировать процесс разработки системы. В процессе реализации проекта может потребоваться коррекция и поддержка актуальности планов разработки.

Одной из наиболее популярных диаграмм для планирования разработки приложения является диаграмма Ганта. Эта диаграмма и будет использована при планировании разработки приложения WorkTracker. Это удобный и простой в использовании инструмент для управления задачами проекта, оценки используемых ресурсов и составления взаимосвязей работ. В данной диаграмме, представляющей собой столбчатую диаграмму, столбцы представляют собой даты, горизонтальные линии – задачи, длительность которых отображена длиной линии. Также для каждой задачи указаны взаимосвязи с другими задачами и ответственные за задачу.

Для представления планирования разработки был выбран Microsoft Project 2007. С его помощью можно заполнять список используемых ресурсов, а также задавать список задач проекта. Заполненный лист ресурсов представлен на рисунке 1.

		Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполыз.	Начисление	Базовый календарь	Код
1		Аналитик	Трудовой		A		100%	100,00 P/ч	180,00 P/ч	0,00 P	Пропорциональное	Стандартный	
2		Программист	Трудовой		П		100%	250,00 P/ч	500,00 P/ч	0,00 P	Пропорциональное	Стандартный	
3		Дизайнер	Трудовой		Д		100%	200,00 P/ч	400,00 P/ч	0,00 P	Пропорциональное	Стандартный	
4		Тестировщик	Трудовой		Т		100%	120,00 P/ч	240,00 P/ч	0,00 P	Пропорциональное	Стандартный	
5		Телефон	Материальный		Тел			18 000,00 P		0,00 P	Пропорциональное		
6		Компьютер	Материальный		К			75 000,00 P		0,00 P	Пропорциональное		
7		Печать	Затраты		Пе						Пропорциональное		
8		Интернет	Затраты		И						Пропорциональное		

Рисунок 1 – Ресурсы проекта

В процессе планирования необходимо выделить основные задачи, подзадачи, длительность их выполнения, а также распределить ресурсы по задачам. В Microsoft Project данные о задачах представлены с специальной таблице. Данная таблица представлена на рисунке 2.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Название ресурса
1	Начало проекта	0 дней	Пн 01.03.24	Пн 01.03.24		Компьютер(0); Телефон(0)
2	- Анализ предметной области и постановка задач на разработку	11 дней	Пн 01.03.24	Пн 15.03.24		
3	Анализ аналогов разработки	2 дней	Пн 01.03.24	Пн 04.03.24	1	Аналитик(100%); Дизайнер(40,00 P)
4	Сравнительный анализ систем-конкурентов	7 дней	Вт 05.03.24	Ср 13.03.24	3	Аналитик(50%); Дизайнер(50%); Интернет(140,00 P)
5	Постановка задачи на разработку системы	2 дней	Пн 14.03.24	Пн 15.03.24	4	Программист
6	Анализ предметной области и постановка задачи на разработку завершены	0 дней	Пн 15.03.24	Пн 15.03.24	5	
7	- Разработка требований к приложению	12 дней	Пн 18.03.24	Вт 09.04.24	6	
8	Разработка функциональных требований	5 дней	Пн 18.03.24	Вт 22.03.24	7	Аналитик(100,00 P)
9	Разработка требований к архитектуре приложения	3 дней	Пн 25.03.24	Ср 27.03.24	8	Программист(100,00 P)
10	Разработка требований к интерфейсу приложения	7 дней	Пн 25.03.24	Вт 02.04.24	9	Дизайнер(140,00 P)
11	Разработка требований к архитектуре программного инструмента	1 день	Чт 28.03.24	Чт 28.03.24	10	Программист(100,00 P)
12	Выбор средств разработки системы	2 дней	Пн 29.03.24	Пн 01.04.24	11	Программист(100,00 P)
13	Разработка требований к приложению завершена	0 дней	Вт 02.04.24	Вт 02.04.24	12	
14	- Разработка приложения	22 дней	Ср 03.04.24	Чт 02.05.24	13	
15	Разработка главного экрана	7 дней	Ср 03.04.24	Чт 11.04.24	14	Программист(100,00 P)
16	Разработка экрана настроек рабочего графика	5 дней	Пн 12.04.24	Чт 18.04.24	15	Программист(100,00 P)
17	Разработка уведомлений	3 дней	Пн 19.04.24	Вт 23.04.24	16	Программист(100,00 P)
18	Разработка экрана настроек уведомлений	2 дней	Ср 24.04.24	Чт 25.04.24	17	Программист(100,00 P)
19	Разработка экрана статистики	5 дней	Пн 26.04.24	Чт 02.05.24	18	Программист(100,00 P)
20	Разработка приложения завершена	0 дней	Чт 02.05.24	Чт 02.05.24	19	
21	- Тестирование приложения	5 дней	Пн 03.05.24	Чт 09.05.24	20	
22	Выбор метода тестирования	1 день	Пн 03.05.24	Пн 03.05.24	21	Тестировщик
23	Проведение тестирования	4 дней	Пн 06.05.24	Чт 09.05.24	22	Тестировщик
24	Тестирование приложения завершено	0 дней	Чт 09.05.24	Чт 09.05.24	23	Тестировщик
25	- Завершение проекта	12 дней	Пн 06.05.24	Пн 27.05.24	24	
26	Составление документации	10 дней	Пн 06.05.24	Чт 23.05.24	25	Аналитик(100,00 P); Программист(100,00 P); Интернет(200,00 P)
27	Ввод в эксплуатацию	2 дней	Пн 24.05.24	Пн 26.05.24	26	Программист(100,00 P)
28	Проект завершен	0 дней	Пн 27.05.24	Пн 27.05.24	27	
29	Конец проекта	0 дней	Пн 27.05.24	Пн 27.05.24	28	

Рисунок 2 – Перечень задач и ресурсов проекта

Для описанных задач и ресурсов Microsoft Project создаёт графическое представление календарного плана проекта. На рисунке 3 представлена диаграмма Ганта для заданных данных.

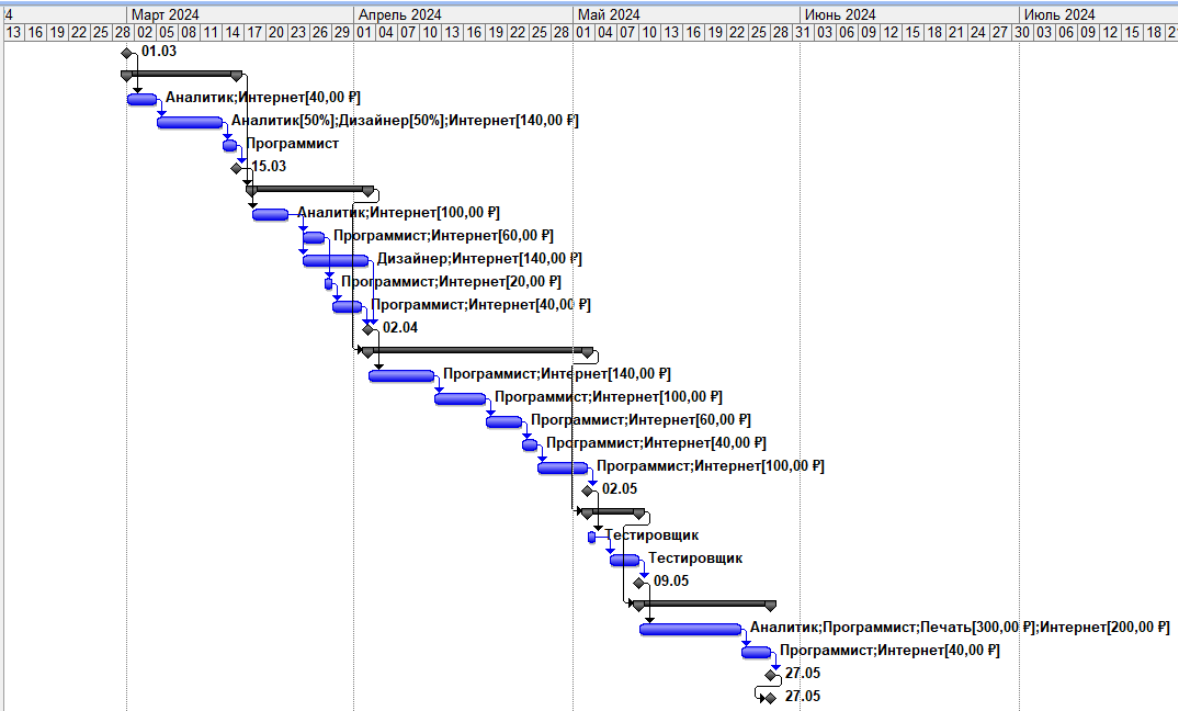


Рисунок 3 – Диаграмма Ганта

Microsoft Project 2007 способен генерировать отчёты, помогающие следить за текущим состоянием проекта: текущее состояние проектных затрат, информация о запланированных, оставшихся, фактических, совокупных, базовых затратах и т. д. Отчёт, представленный на рисунке 4, показывает необходимый бюджет для реализации проекта.

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
26	Составление документации	0,00 P	Пропорциональное	28 500,00 P	0,00 P	28 500,00 P
15	Разработка главного экрана	0,00 P	Пропорциональное	14 140,00 P	0,00 P	14 140,00 P
10	Разработка требований к интерфейсу	0,00 P	Пропорциональное	11 340,00 P	0,00 P	11 340,00 P
16	Разработка экрана настроек рабочего стола	0,00 P	Пропорциональное	10 100,00 P	0,00 P	10 100,00 P
19	Разработка экрана статистики	0,00 P	Пропорциональное	10 100,00 P	0,00 P	10 100,00 P
4	Сравнительный анализ систем-конкурентов	0,00 P	Пропорциональное	8 540,00 P	0,00 P	8 540,00 P
9	Разработка требований к архитектурам	0,00 P	Пропорциональное	6 060,00 P	0,00 P	6 060,00 P
17	Разработка уведомлений	0,00 P	Пропорциональное	6 060,00 P	0,00 P	6 060,00 P
8	Разработка функциональных требований	0,00 P	Пропорциональное	4 100,00 P	0,00 P	4 100,00 P
12	Выбор средств разработки системы	0,00 P	Пропорциональное	4 040,00 P	0,00 P	4 040,00 P
18	Разработка экрана настроек уведомлений	0,00 P	Пропорциональное	4 040,00 P	0,00 P	4 040,00 P
27	Ввод в эксплуатацию	0,00 P	Пропорциональное	4 040,00 P	0,00 P	4 040,00 P
5	Постановка задачи на разработку системы	0,00 P	Пропорциональное	4 000,00 P	0,00 P	4 000,00 P
23	Проведение тестирования	0,00 P	Пропорциональное	3 840,00 P	0,00 P	3 840,00 P
11	Разработка требований к аппаратуре	0,00 P	Пропорциональное	2 020,00 P	0,00 P	2 020,00 P
3	Анализ актуальности разработки	0,00 P	Пропорциональное	1 640,00 P	0,00 P	1 640,00 P
22	Выбор метода тестирования	0,00 P	Пропорциональное	960,00 P	0,00 P	960,00 P
1	Начало проекта	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
6	Анализ предметной области и постановка задачи	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
13	Разработка требований к приложению	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
20	Разработка приложения завершена	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
24	Тестирование приложения завершено	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
28	Проект завершен	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
29	Конец проекта	0,00 P	Пропорциональное	0,00 P	0,00 P	0,00 P
		0,00 P		123 520,00 P	0,00 P	123 520,00 P

Рисунок 4 – Отчет о бюджете проекта

По результатам отчета видно, что затраты на реализацию приложения составили 123 520,00 р. На рисунке 5 представлен отчёт об общих сведениях по проекту.

<u>Даты</u>			
Начало:	Пт 01.03.24	Окончание:	Пн 27.05.24
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
<u>Длительность</u>			
Текущий план:	62 дней	Оставшаяся:	62 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	62 дней	Процент завершения:	0%
<u>Трудозатраты</u>			
Текущий план:	624 ч	Оставшиеся:	624 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	624 ч	Процент завершения:	0%
<u>Затраты</u>			
Текущий план:	123 520,00 Р	Оставшиеся:	123 520,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	123 520,00 Р		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи	29	Трудовые ресурсы:	6
Выполняющиеся задачи	0	Трудовые ресурсы с прев. досл	0
Завершенные задачи:	0	Материальные ресурсы:	2
Итого задач:	29	Итого ресурсов:	8

Рисунок 5 – Отчет об общих сведениях по проекту

Исходя из отчета, длительность проекта, с учетом праздников и выходных дней составила 87 дней. Используя полученные данные о сроках и бюджете проекта, можно сделать вывод, что приложение WorkTracker является экономически выгодным.

2 Анализ требований на разработку программных средств

2.1 Анализ функциональных требований

Для управления текущим рабочим состоянием пользователь должен знать текущее состояние и иметь возможность его изменить. После изменения состояния пользователь может выйти из приложения или же вновь изменить состояние.

Пользователь может находиться в следующих состояниях:

- рабочий день не запланирован;
- рабочий день не начат;
- опоздание;
- рабочий день активен;
- пауза;
- обед;
- переработка;
- рабочий день завершен.

На рисунке 6 представлена диаграмма вариантов использования, показывающая основные возможные взаимодействия сотрудника и системы [1].

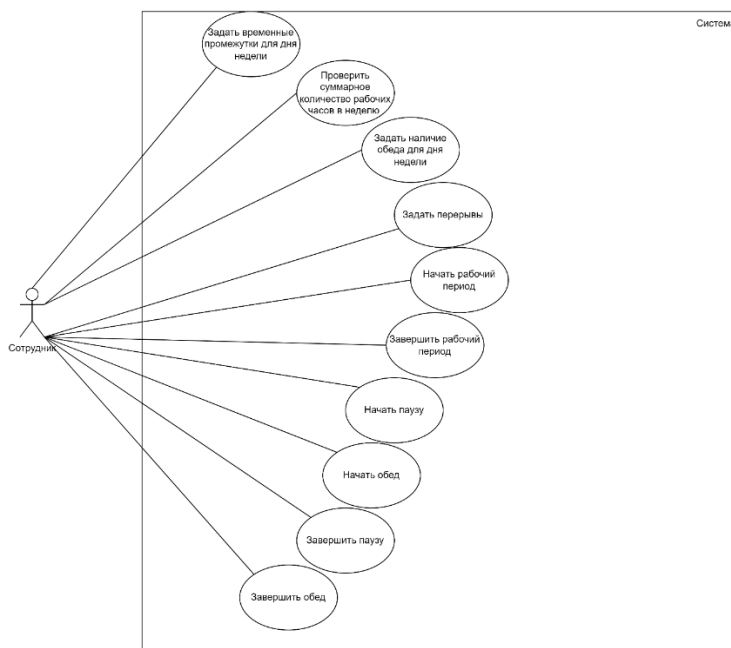


Рисунок 6 – Диаграмма основных вариантов использования

Согласно диаграмме, сотрудник может запланировать рабочие промежутки, перерывы и обеды, а также начинать и заканчивать их. Помимо этого, для удобства составления расписания сотруднику будет выведена информация о текущем запланированном количестве рабочих часов в неделю.

Основная функциональность приложения будет осуществляться благодаря переходу между данными состояниями. Некоторые переходы будут инициированы пользователем, другие произойдут при наступлении определенного времени. На рисунке 7 представлена диаграмма деятельности, демонстрирующая обработку переходов, которые совершает пользователь.

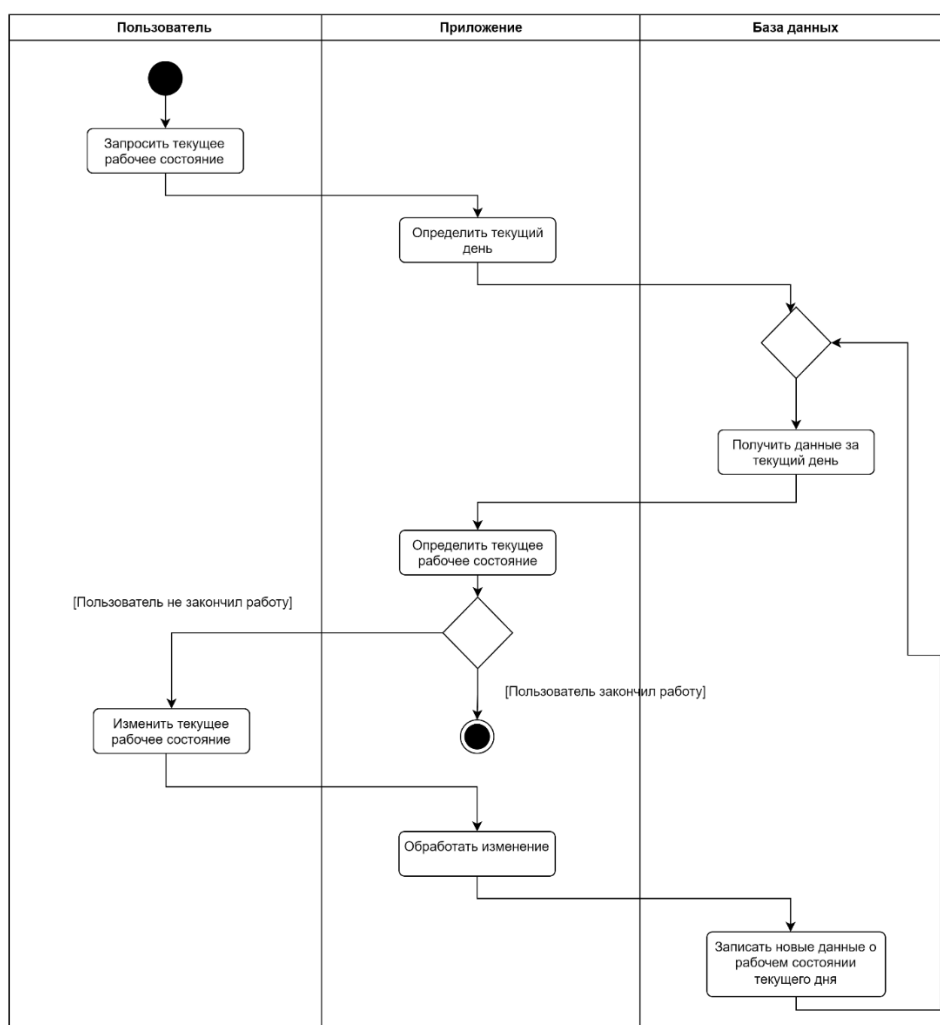


Рисунок 7 – Диграмма деятельности для обработки переходов между состояниями

На представленной диаграмме отображен процесс изменения текущего состояния пользователем. При входе в приложение, оно обращается в базу

данных и, используя полученную информацию, вычисляет текущее рабочее состояние и показывает его пользователю. После получения текущего состояния пользователь может либо завершить работу с приложением, либо изменить состояние. При изменении состояния информация об этом должна быть записана в базу данных и должна произойти обработка данных с учетом новой информации, после чего обновленное состояние будет отображено пользователю.

Для корректной работы с текущими состояниями у пользователя должна быть возможность составить план рабочих периодов для недели. Для обеспечения гибкости системы количество рабочих периодов в дне может быть больше одного. Пользователю необходима возможность выбирать время начала и завершения каждого рабочего периода, а также указывать плановое время обеда и его наличие в конкретный день. Для упрощения составления расписания, пользователю должно быть отображено суммарное время работы за неделю.

Для изменения текущего состояния пользователю могут быть необходимы напоминания. Приложение должно напоминать пользователю о необходимости начать и завершить работу, а также о необходимости отметить время фактического времени обеда. Следует разделить уведомление о начале рабочего дня и о начале очередного рабочего периода. Поскольку каждый пользователь имеет потребность в индивидуальном времени напоминания, то необходимо создать возможность настройки напоминаний.

Помимо отслеживания текущего рабочего состояния необходимо обеспечить сохранение и отображение суммарных значений времени, которое было проведено в рабочем состоянии, а также в паузах. Вывод этой информации должен быть обеспечен для выбранного пользователем произвольного промежутка времени.

2.2 Требования к архитектуре программных средств

Проанализировав функциональные требования, можно заметить, что работа приложения построена на изменении рабочего состояния пользователя.

Для разрабатываемой системы необходимо выбрать архитектуру, которая могла бы учитывать эту особенность.

Такой архитектурой является Model-View-Intent (MVI) [2]. Объединив данную архитектуру с Model-View-ViewModel (MVVM) мы сможем получить преимущества от обеих архитектур [3]. MVI обеспечит удобную работу с состояниями и переходами между состояниями, повысит тестируемость кода, а также сделает приложение полностью реактивным. MVVM позволит отделить логику от отображения и от базы данных.

У данных архитектур имеется два одинаковых компонента:

- представление (View), которое представляет собой пользовательский интерфейс;
- модель (Model), которая является моделями данных в приложении.

Отличаются архитектуры последним компонентом. В MVI этот компонент является намерением (Intent). Этот компонент отвечает за передачу намерений пользователя в систему, которые инициируют смену состояния. Для MVVM компонентом является ViewModel. Данный компонент отвечает за обработку данных от представления и модели. После обработки данные либо сохраняются в системе, либо отображаются пользователю.

Объединив данные архитектуры, получим архитектуру, представленную на рисунке 8.

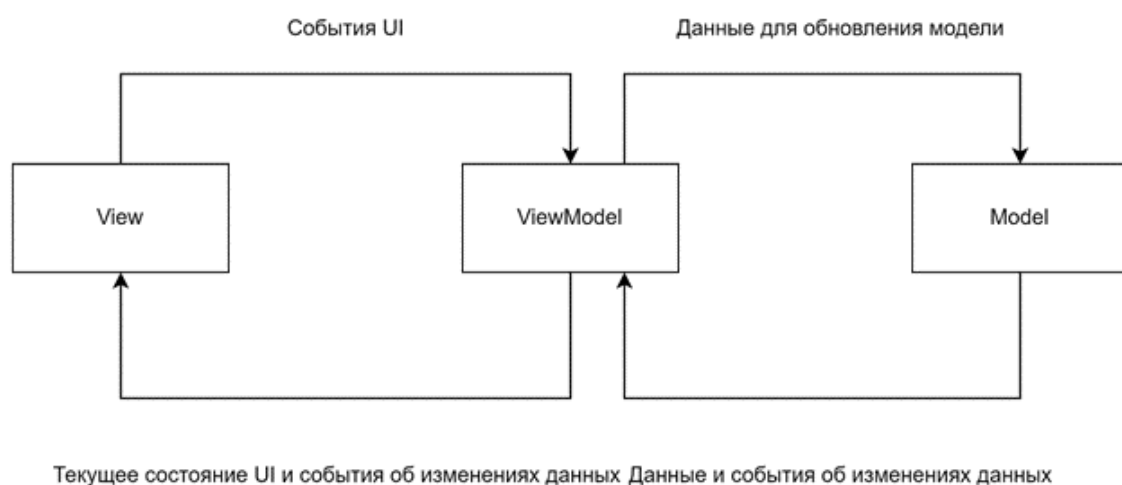


Рисунок 8 – Архитектура приложения

В полученной архитектуре представление и ViewModel взаимодействуют с помощью обмена событиями. Во ViewModel хранится единое состояние, которое изменяется при получении событий из представления и модели и передается для отображения представлению.

2.3 Требования к пользовательскому интерфейсу программных средств

Для обеспечения наилучшего пользовательского опыта, пользовательский интерфейс должен быть интуитивно понятным и соблюдать основные правила UI/UX. Компоненты интерфейса должны быть спроектированы исходя из принципов Material Design.

Интерфейс приложения должен быть представлен несколькими экранами:

- главный экран;
- экран настроек рабочих периодов;
- экран настроек уведомлений;
- экран статистики.

На главном экране должен быть отображен текущие состояние работы пользователя, запланированные перерывы, а также элементы управления состоянием. Управление состоянием должно производиться нажатием по кнопке. Возможные переходы из текущего состояния должны определяться автоматически. Помимо этого, должна присутствовать возможность создавать, удалять и редактировать запланированные перерывы.

На экране настройки рабочих периодов пользователю должен быть представлен функционал создания, редактирования и удаления рабочих периодов. Также пользователь должен иметь возможность указать наличие обеда во время работы в определенный день. На экране должно присутствовать суммарное количество рабочих часов за неделю.

Экран настройки уведомлений должен обеспечивать возможность настройки времени отправки напоминаний. Должны присутствовать настройки

утреннего уведомления, напоминания об обеде, напоминания перед началом и окончанием рабочего периода.

На экране статистики пользователь должен иметь возможность выбрать временной интервал для отображения статистики. Статистика представляет собой время, которое сотрудник провел в состоянии работы, запланированном и незапланированном перерыве.

2.4 Обоснование выбора технологий и средств разработки

Согласно требованиям к интерфейсу, функциональным требованиям, а также архитектурным требованиям, были определены необходимые технологии и средства разработки мобильного android приложения.

Разработка нативных Android-приложений происходит на языке программирования Kotlin. Он обладает удобным синтаксисом и имеет активную поддержку [4].

Для реализации пользовательского интерфейса была выбрана библиотека Jetpack Compose. Данная библиотека позволяет реализовывать графический интерфейс с помощью декларативного подхода [5]. Декларативный подход разработки упрощает процесс разработки интерфейса, уменьшает количество возможных ошибок так как позволяет не разрабатывать логику перехода одного состояния интерфейса в другое. Также данный подход подходит под выбранную архитектуру поскольку также работает с намерениями и состояниями. Данная библиотека имеет активную поддержку от Google и постепенно заменяет подход к созданию интерфейса с помощью View.

Для работы с базой данных была выбрана библиотека Room. Она позволяет работать с базой данных SQLite и является популярным решением для работы с базой данных [6].

Для обеспечения качества программного кода, он должен быть реализован, следуя принципам SOLID [7]. Для этого необходимо решение для работы с зависимостями в коде. Для этого была выбрана библиотека Hilt. Она также обладает активной поддержкой от Google и обеспечивает ленивую

инициализацию ViewModel при внедрении в функции Jetpack Compose, чем выигрывает у аналогичных решений [8].

Для реализации асинхронных обращений к базе данных были выбраны Coroutines, встроенные в Kotlin. На текущий момент они являются универсальным решением для реализации асинхронных операций, обладают возможностью настройки контекста выполнения операций и значительно упрощают понимание работы асинхронного кода [9].

Построение диаграмм осуществляется с помощью языка UML 2.0 в графическом редакторе draw.io. Данное обеспечение позволяет легко создавать различные UML диаграммы.

Для разработки выбрана среда разработки Android Studio от компании Google, которая обеспечивает быстрый и точный анализ кода, рефакторинг кода, его простую отладку и автоматическое исправление форматирования по заданному шаблону. Разработка осуществляется под управлением операционной системы Windows 10, запуск и просмотр разрабатываемого приложения – на эмуляторах, поставляемых Android Studio.

2.5 Требования к аппаратно-программному обеспечению

Для запуска приложения необходимо мобильное устройство с операционной системой Android. Минимальной версией Android для работы предложения является версия Android 8.0. Данное ограничение позволит приложению корректно работать на большинстве используемых устройств.

Требования к оборудованию:

- 4 GB оперативной памяти системы;
- 50 MB памяти на персистентном хранилище системы;
- процессор Dimensity 900 и лучше или аналоги.

Разрабатываемое приложение может быть запущено на всех современных устройствах, которые удовлетворяют требованию по программному обеспечению.

3 Разработка программных средств

3.1 Разработка архитектуры программных средств

Проанализировав требования к разрабатываемому приложению, было решено опираться на принципы «Чистой архитектуры». Весь разрабатываемый код был разделен на уровни:

- уровень представления;
- уровень домена;
- уровень данных.

При этом классы уровня домена должны зависеть только от классов того же уровня, а классы уровня представления и данных будут зависеть от уровня домена.

Уровень представления работает с пользовательским интерфейсом приложения. Компоненты данного уровня состоят из экранов, состояний, эффектов и ViewModel для этих экранов. Состав этих компонентов был получен путем анализа пункта 2.2. Уровень представления использует определенные на уровне домена модели данных. Данные модели могут быть получены как в виде единственного объекта, так и в виде потока объектов. Это позволяет обеспечить независимость уровня домена от уровня представления.

Уровень домена содержит объявление используемых моделей данных, а также интерфейсы репозитория. Объявленные классы используются на всех уровнях приложения, а реализации репозитория находятся на уровне данных. Часто данный уровень содержит классы UseCase, которые содержат основную бизнес-логику приложения, но при реализации программных средств было принято решение не использовать данные классы, перенести бизнес-логику во ViewModel. Это решение было принято из-за избыточности реализации этих классов в разрабатываемой системе.

Уровень данных содержит логику работы приложения с базой данных. На нем объявлены соответствующие модели и интерфейсы взаимодействия, которые используются только на данном уровне. Уровень данных реализует

интерфейсы репозитория, объявленные на уровне домена, что позволяет уровню представления получать данные из базы данных используя соответствующие интерфейсы.

Исходный код разработанного приложения представлен в приложении Б.

3.2 Разработка уровня представления

При реализации уровня представления использовались следующие библиотеки Jetpack Compose и Compose Destinations. Вторая библиотека используется для организации навигации между экранами, созданными с помощью Jetpack Compose.

Выбор Jetpack Compose оправдан архитектурными особенностями разрабатываемого приложения. При изменении состояния экрана приложения пользовательский интерфейс должен отреагировать на изменение данных. При императивном подходе разработчику необходимо реализовать логику переходов из одного состояния в другое. Данная логика может быть достаточно сложна в реализации и может содержать ошибки. Использование декларативного подхода к реализации интерфейса позволяет не реализовывать логику перехода между состояниями, позволив разработчику избегать значительного числа возможных ошибок.

Jetpack Compose использует Composable функции для создания интерфейса. В данных функциях вызываются другие Composable функции, создавая дерево компонентов. Данное дерево имеет отличную оптимизацию, поэтому Jetpack Compose не уступает по производительности императивному аналогу. Пример реализации Composable функции представлен в листинге 1.

Листинг 1 – Composable функция, реализующая содержание карточки с результатами работы

```
@Composable
fun ResultsScreenContent(
    state: CardState.Results,
    onClickStartWork: () -> Unit = {},
    onClickDeleteEvent: (WorkEvent, Int) -> Unit = { _, _ -> },
    onAddPeriod: (Int) -> Unit = {},
    onClickEvent: (Int, WorkEvent) -> Unit = { _, _ -> },
```

```

) {
    Column {
        HeaderDay(
            modifier = Modifier
                .fillMaxWidth()
                .padding(start = 17.dp, top = 26.dp, end = 28.dp, bottom
= 15.dp),
            day = state.day
        )
        WorkTimer(
            time = state.statistic.workTime,
            title = stringResource(id = R.string.time_worked),
        )
        StartButton(
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = 12.dp, vertical = 0.dp),
            onClickStartWork = onClickStartWork,
            buttonActive = true,
            startEarly = false,
            contentColor = MaterialTheme.colors.primaryVariant
        )
        EventList(
            modifier = Modifier
                .fillMaxWidth()
                .padding(12.dp),
            events = state.events,
            day = state.day.ordinal,
            onClickDeleteEvent = onClickDeleteEvent,
            onAddPeriod = onAddPeriod,
            onClickEvent = onClickEvent
        )
    }
}

```

Данная функция принимает в качестве параметров объект состояния карточки результатов, а также набор функций, позволяющий получать информацию о взаимодействии пользователя с карточкой. Представленная функция последовательно вызывает ранее объявленные функции, передавая необходимые параметры. Все элементы интерфейса в данной функции будут отображены в виде колонки.

Jetpack Compose обладает большим количеством плюсов, но имеет недостаток. Данным недостатком является способ реализации навигации между экранами приложения. Для реализации навигации необходимо использовать большое количество шаблонного кода, в котором могут возникать ошибки на этапе выполнения программы. Данную проблему позволяет решить Compose Destinations. Данная библиотека предоставляет набор классов и функций-

расширений для реализации навигации. Также представленная библиотека использует генерацию кода для создания классов, позволяющих осуществлять передачу параметров при навигации. Библиотека позволяет разработчику использовать несколько аннотаций для создания графа навигации.

Для реализации экрана приложения используется определенный набор основных компонентов:

- Composable функция, содержащая в имени Screen и отмеченная аннотацией Destination;
- класс состояния экрана, который наследуется от класса базового состояния;
- класс события экрана, который наследуется от класса базового события;
- класс ViewModel, который наследуется от базового класса ViewModel и работает с состояниями и событиями экрана.

Composable функция получает в качестве параметра ViewModel. Передача ViewModel происходит с помощью использования параметра по умолчанию, который представляет собой результат вызова функции `hiltViewModel`, предоставляемой библиотекой для внедрения зависимостей Hilt. Composable функция получает текущее состояние экрана, а также получает события от ViewModel. Действия пользователя при взаимодействии с интерфейсом передаются для обработки во ViewModel.

3.3 Разработка уровня домена

При реализации уровня домена не использовались библиотеки. Классы данного уровня должны зависеть только от внутренних классов. На данном уровне находятся модели данных и интерфейсы репозитория. От данных классов зависят все остальные уровни.

3.4 Разработка уровня данных

При реализации уровня данных использовалась библиотека Room для работы с локальной базой данных SQLite. На рисунке 9 отображена физическая диаграмма базы данных.

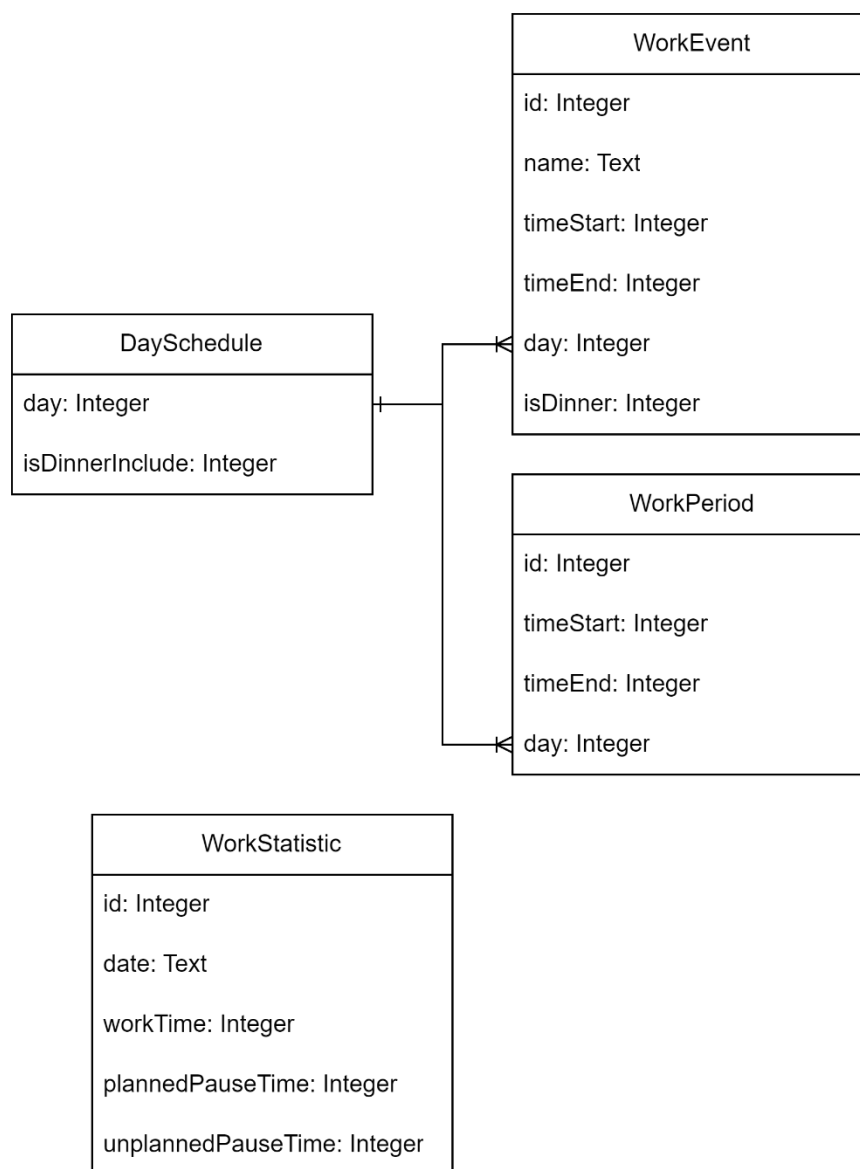


Рисунок 9 – Физическая диаграмма базы данных

На диаграмме представлены следующие сущности:

- сущность статистики за день (WorkStatistic);
- сущность дня недели (DaySchedule);
- сущность запланированных перерывов (WorkEvent);

- сущность запланированных рабочих периодов (WorkPeriod).

SQLite не имеет некоторых привычных типов данных, поэтому на данной диаграмме логические колонки представлены целочисленным типом данных, а даты хранятся в виде текста. Room позволяет уменьшить влияние этих ограничений благодаря использованию преобразователей типов. Помимо этого, Room скрывает низкоуровневые обращения к базе данных благодаря генерации кода, аннотациям и интерфейсам.

При использовании Flow в качестве возвращаемого значения функции интерфейса, Room создает триггеры для необходимых таблиц и позволяет получать поток данных. Это позволяет всегда отображать актуальное состояние системы на уровне представления.

4 Тестирование приложения WorkTracker

4.1 Выбор методологии тестирования

При тестировании Android приложений необходимо проверять не только корректность выполнения функций, но и правильность отображения выходных данных на пользовательском интерфейсе.

Существует несколько типов тестирования [10]:

- функциональное;
- нефункциональное.

К функциональному тестированию относятся такие виды тестирования, которые позволяют проверить выполнены ли функциональные требования к системе. Функциональное тестирование представлено на всех уровнях системы: модульном, интеграционном, системном и приемочном.

Нефункциональное тестирование включает виды тестирования, проверяющие нефункциональные аспекты приложения, а именно производительность, надежность, безопасность и удобство пользования. Эти метрики могут быть выражены в различных величинах.

Для тестирования разрабатываемого приложения использованы как функциональное тестирование, так и нефункциональное.

4.2 Функциональное тестирование

Для проведения функционального тестирования была составлена диаграмма State & Transition (S&T) [11]. Данная диаграмма используется для визуального представления возможных состояний объекта и условий перехода из одного состояния в другое. Для отображения должна быть выбрана сущность, от состояния которой зависит состояние системы. С использованием диаграммы S&T может быть составлена таблица переходов, используя которую будут составлены тест-кейсы. На рисунке 10 представлена диаграмма для такой сущности в разрабатываемой системе.

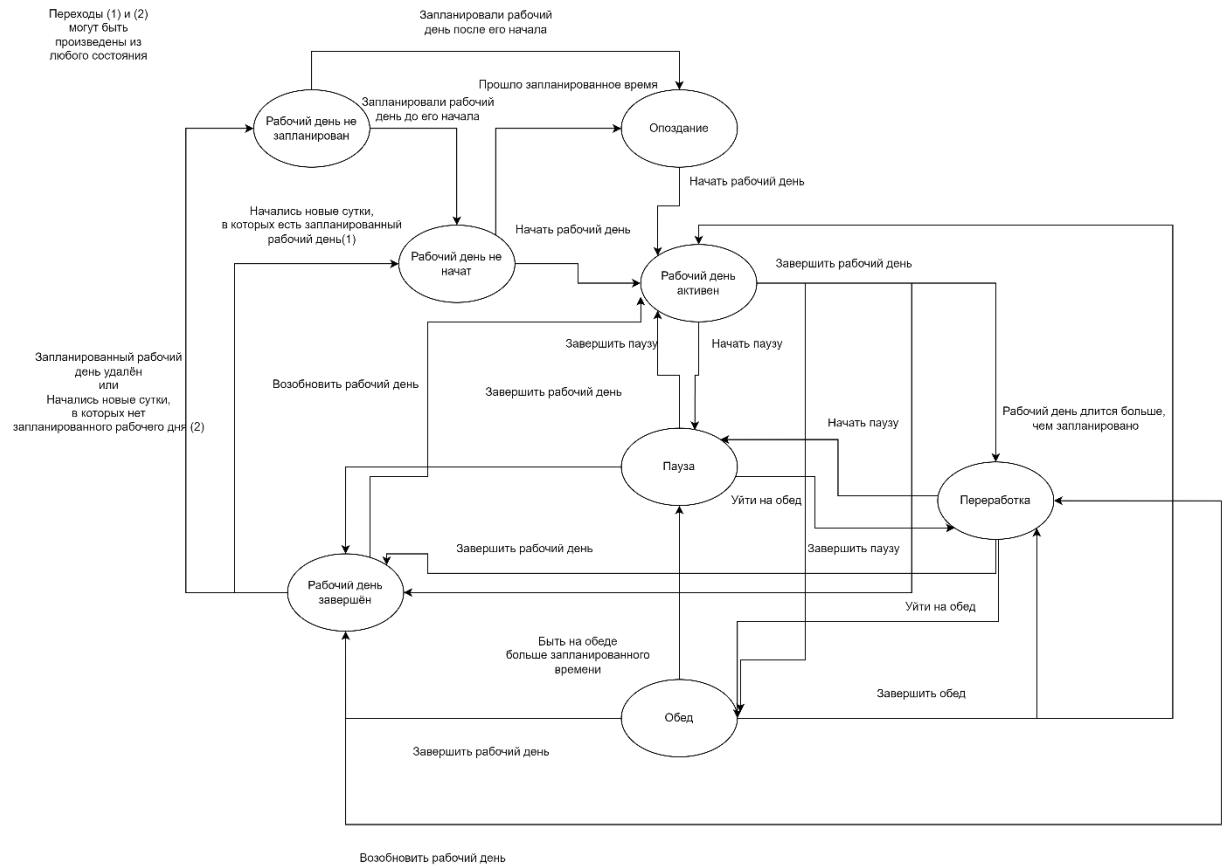


Рисунок 10 – Диаграмма S&T

На представленной диаграмме отображены все возможные состояния для такого объекта системы, как текущее рабочее состояние. Также на данной диаграмме представлены все возможные переходы между существующими состояниями. Обозначения переходов представлены в таблице 1.

Таблица 1 – Таблица обозначения переходов

Обозначение	Расшифровка
П1	Запланировали рабочий день до его начала
П2	Запланировали рабочий день после его начала
П3	Прошло запланированное время начала
П4	Начали рабочий день
П5	Начали паузу
П6	Завершили паузу

Продолжение таблицы 3

Состояние	Переход														
	П1	П2	П3	П4	П5	П6	П7	П8	П9	П10	П11	П12	П13	П14	П15
C2			C3	C4									C1	C1	C2
C3				C4									C1	C1	C2
C4					C5		C6			C7	C8		C1	C1	C2
C5						C4					C8		C1	C1	C2
C6								C4	C5		C8		C1	C1	C2
C7					C5		C6				C8		C1	C1	C2
C8												C4	C1	C1	C2

С использованием таблицы 3 были составлены тест-кейсы для тестирования переходов между состояниями. Тест-кейсы представлены в таблице 4.

Таблица 4 – Тест-кейсы

№	Входные данные	Выходные данные
1	Не запланировать рабочий день	Рабочий день не запланирован
2	Запланировать рабочий день до его начала	Рабочий день не начат
3	Запланировать рабочий день после его начала	Опоздание
4	Запланировать рабочий день до его начала и начать рабочий день	Рабочий день начат
5	Запланировать рабочий день до его начала, начать рабочий день и начать паузу	Пауза
6	Запланировать рабочий день до его начала, начать рабочий день, начать паузу и закончить паузу	Рабочий день начат

Продолжение таблицы 4

№	Входные данные	Выходные данные
7	Запланировать рабочий день до его начала, начать рабочий день и начать обед	Обед
8	Запланировать рабочий день до его начала, начать рабочий день, начать обед и закончить обед	Рабочий день начат
9	Запланировать рабочий день до его начала, начать рабочий день, начать обед и быть на обеде больше запланированного времени	Пауза
10	Запланировать рабочий день до его начала, начать рабочий день, начать обед, быть на обеде больше запланированного времени, завершить паузу	Рабочий день начат
11	Запланировать рабочий день до его начала, начать рабочий день и рабочий день длится больше, чем запланировано	Переработка

Продолжение таблицы 4

№	Входные данные	Выходные данные
12	Запланировать рабочий день до его начала, рабочий день длится больше, чем запланировано, начать рабочий день и начать паузу	Пауза
13	Запланировать рабочий день до его начала, начать рабочий день, рабочий день длится больше, чем запланировано, начать паузу и закончить паузу	Переработка
14	Запланировать рабочий день до его начала, начать рабочий день, рабочий день длится больше, чем запланировано и начать обед	Обед
15	Запланировать рабочий день до его начала, начать рабочий день, рабочий день длится больше, чем запланировано, начать обед и закончить обед	Переработка
16	Запланировать рабочий день до его начала, начать рабочий день и завершить рабочий день	Рабочий день завершён

Окончание таблицы 4

№	Входные данные	Выходные данные
17	Запланировать рабочий день до его начала, начать рабочий день, завершить рабочий день и дожждаться следующих суток, в которых есть запланированный рабочий день	Рабочий день не начат
18	Запланировать рабочий день до его начала, начать рабочий день, завершить рабочий день и возобновить рабочий день	Рабочий день начат
19	Запланировать рабочий день до его начала, начать рабочий день и текущий рабочий день удалён	Рабочий день не запланирован
20	Запланировать рабочий день до его начала, прошло запланированное время	Опоздание
21	Запланировать рабочий день до его начала, начать рабочий день, завершить рабочий день и дожждаться следующих суток, в которых нет запланированного рабочего дня	Рабочий день не запланирован

Результаты проведения данных тест-кейсов показали корректную и безошибочную работу программы.

4.3 Нефункциональное тестирование

4.3.1 Unit тестирование

Для Unit тестирования Android приложений используются JUnit и Mockito.

JUnit – фреймворк для модульного тестирования программного обеспечения на языке Java [12].

Mockito позволяет создавать виртуальные объекты (mock'и) с простейшей реализацией. Такие виртуальные объекты позволяют тестировать код, не прибегая к реальным объектам.

Библиотека Mockito работает вместе с JUnit и может быть использована для активации работы в unit-тестах.

Для тестирования асинхронного кода с использованием Coroutines используется дополнительная библиотека `kotlinx-coroutines-test`. Она позволяет запускать тесты в специализированном `CoroutineScope`.

В листинге 2 представлено unit тестирования основного репозитория приложения.

Листинг 2 – Unit тестирование репозитория `ScheduleRepository`

```
package ru.kolyagin.worktracker.data.repositories

import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toPersistentList
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.test.runTest
import org.junit.Assert.*
import org.junit.Before
import org.junit.Test
import org.mockito.kotlin.mock
import org.mockito.kotlin.times
import org.mockito.kotlin.verify
import org.mockito.kotlin.whenever
import ru.kolyagin.worktracker.data.database.dao.ScheduleDao
import ru.kolyagin.worktracker.data.database.entities.DayScheduleEntity
import ru.kolyagin.worktracker.data.database.entities.WorkEventEntity
import ru.kolyagin.worktracker.data.database.entities.WorkPeriodEntity
import ru.kolyagin.worktracker.data.database.entities.mapToDomain
import ru.kolyagin.worktracker.domain.models.DayWorkInfo
import ru.kolyagin.worktracker.domain.models.Time
import ru.kolyagin.worktracker.domain.models.WorkPeriod
import java.time.DayOfWeek
```

```

class ScheduleRepositoryImplTest {

    private val dao = mock<ScheduleDao>()

    private val repository = ScheduleRepositoryImpl(
        dao
    )

    private val day = DayOfWeek.FRIDAY

    private val workEventEntities = listOf(
        WorkEventEntity(
            id = 1,
            name = "Name",
            day = day.ordinal,
            isDinner = false,
            timeEnd = Time.fromMinutes(10),
            timeStart = Time.fromMinutes(1)
        )
    )

    private val dayScheduleEntities = listOf(
        DayScheduleEntity(
            day = day.ordinal,
            isDinnerInclude = false
        )
    )

    private val workPeriod = WorkPeriod(
        id = 1,
        timeEnd = Time.fromMinutes(10),
        timeStart = Time.fromMinutes(1),
    )

    @Before
    fun before() {
        whenever(dao.getWorkEvents()).thenReturn(
            flow {
                emit(workEventEntities)
            }
        )

        whenever(dao.getDaySchedule()).thenReturn(
            flow {
                emit(dayScheduleEntities)
            }
        )

        whenever(dao.getWorkPeriods()).thenReturn(
            flow { emit(emptyList()) }
        )
    }

    @Test
    fun schedule() = runTest {
        val result = repository.schedule()
        verify(dao, times(1)).getDaySchedule()
        verify(dao, times(1)).getWorkPeriods()
    }
}

```

```

        verify(dao, times(1)).getWorkEvents()
        assertEquals(
            DayWorkInfo(
                day = day,
                isDinnerInclude = false,
                events = workEventEntities.map { it.mapToDomain() }
            ).toPersistentList(),
            result.first().first().periods = persistentListOf()
        )
    }

    @Test
    fun addPeriod() = runTest {
        val day = DayOfWeek.FRIDAY
        repository.addPeriod(day)
        verify(dao, times(1)).addPeriod(day.ordinal)
    }

    @Test
    fun deletePeriod() = runTest {
        val id = 1L
        repository.deletePeriod(WorkPeriod(id, Time.fromMinutes(1),
            Time.fromMinutes(2)))
        verify(dao, times(1)).deletePeriod(id)
    }

    @Test
    fun setTime() = runTest {
        repository.setTime(
            workPeriod,
            day
        )
        verify(dao, times(1)).setTime(
            WorkPeriodEntity(
                workPeriod.id,
                workPeriod.timeStart,
                workPeriod.timeEnd,
                day.ordinal
            )
        )
    }

    @Test
    fun setDinner() = runTest {
        repository.setDinner(day.ordinal, true)
        verify(dao, times(1)).setDinner(day.ordinal, true)
    }

    @Test
    fun addWorkEvent() = runTest {
        val event = workEventEntities.first().mapToDomain()
        repository.addWorkEvent(day.ordinal, event)
        verify(dao, times(1)).addEvent(
            day.ordinal,
            event.name,
            event.isDinner,

```

```

        event.timeStart,
        event.timeEnd
    )
}

@Test
fun deleteWorkEvent() = runTest {
    val event = workEventEntities.first().mapToDomain()
    repository.deleteWorkEvent(event)
    verify(dao, times(1)).deleteEvent(event.id)
}

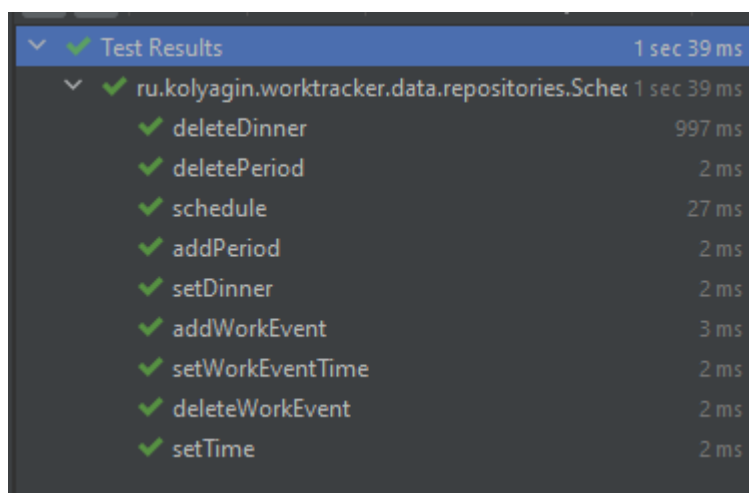
@Test
fun deleteDinner() = runTest {
    repository.deleteDinner(day.ordinal)

    verify(dao, times(1)).deleteDinner(day.ordinal)
}

@Test
fun setWorkEventTime() = runTest {
    val event = workEventEntities.first().mapToDomain()
    repository.setWorkEventTime(event, day.ordinal)
    verify(dao, times(1)).changeEvent(
        workEventEntities.first().copy(day = day.ordinal)
    )
}
}

```

На рисунке 11 приведены результаты выполнения тестирования.



Test Results	1 sec 39 ms
ru.kolyagin.worktracker.data.repositories.Scher	1 sec 39 ms
deleteDinner	997 ms
deletePeriod	2 ms
schedule	27 ms
addPeriod	2 ms
setDinner	2 ms
addWorkEvent	3 ms
setWorkEventTime	2 ms
deleteWorkEvent	2 ms
setTime	2 ms

Рисунок 11 – Результаты тестирования

В результате Unit тестирования ошибок не было выявлено, что означает, что репозиторий реализован корректно.

4.3.2 Метрика SLOC

Количество строк кода (англ. Source Lines of Code – SLOC) – это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода [13]. Подсчет данной метрики проведен в функциях `schedule`, `setTime` и `addWorkEvent`, служащих для работы с расписанием пользователя. Функции приведены в листинге 3.

Листинг 3 – Функции `schedule`, `setTime` и `addWorkEvent`

```

override fun schedule(): Flow<ImmutableList<DayWorkInfo>> =
    // Комбинируем все потоки данные для формирования расписания
    combine(
        scheduleDao.getDaySchedule(),
        scheduleDao.getWorkPeriods(),
        scheduleDao.getWorkEvents()
    ) { dayScheduleEntities, workPeriodEntities, events ->
        // Группируем данные по дню
        val groups = workPeriodEntities.groupBy { it.day }
        val eventGroups = events.groupBy({ it.day }) {
it.mapToDomain() }
        // Создаём список с расписанием на каждый день
        dayScheduleEntities.map { scheduleEntity ->
            val eventsList =
eventGroups[scheduleEntity.day]?.toImmutableList()
                ?: persistentListOf()
            DayWorkInfo(
                day = DayOfWeek.values()[scheduleEntity.day],
                periods = groups[scheduleEntity.day]?.map {
it.mapToDomain() }
                    ?.toImmutableList()
                    ?: persistentListOf(),
                isDinnerInclude =
scheduleEntity.isDinnerInclude,
                events = eventsList
            )
        }.toImmutableList()
    }

override suspend fun setTime(
    workPeriod: WorkPeriod, dayOfWeek: DayOfWeek
) {
    scheduleDao.setTime(
        WorkPeriodEntity(
            workPeriod.id, workPeriod.timeStart,
workPeriod.timeEnd, dayOfWeek.ordinal
        )
    )
}

override suspend fun addWorkEvent(dayOfWeek: Int, event: WorkEvent)
{

```

```

// Если переданный день = -1, то добавляем событие во все дни
if (dayOfWeek == -1) {
    for (i in 0 until 6)
        scheduleDao.addEvent(
            i,
            event.name,
            event.isDinner,
            event.timeStart,
            event.timeEnd
        )
} else {
    scheduleDao.addEvent(
        dayOfWeek,
        event.name,
        event.isDinner,
        event.timeStart,
        event.timeEnd
    )
}
}

```

Физическими строками считаются все непустые строки текстового файла.

Измеряя логические строки кода, предпринимается попытка посчитать количество собственно операторов в программе.

В данной программе имеем:

- 54 физических строк кода;
- 27 логических строк кода;
- 5 строк комментариев.

Используя плагин Statistic для Android Studio, можно получить общее количество строк кода в разработанном ПО. На рисунке 12 представлено общее количество строк и количество строк, содержащее исходный код программы.

Overview <> kt <> properties <> xml		
Source File ▲	Total Lines	Source Code Lines
ActivityCallbacks.kt	34	25
AddButton.kt	48	46
AlarmNotificationsManager.kt	294	274
AnalyticsEvent.kt	13	5
AppDatabase.kt	28	25
Application.kt	17	13
BaseViewModel.kt	207	103
Total:	6776	5994

Рисунок 12 – Результаты сканирования проекта с помощью плагина Statistic

Итоговая программа обладает оптимальным количеством строк исходных кодов.

4.3.3 Sonar

Используя плагин для Android Studio SonarLint, была произведена проверка кода на соответствие существующим стандартам разработки. На рисунке 13 представлены результаты проверки кода.

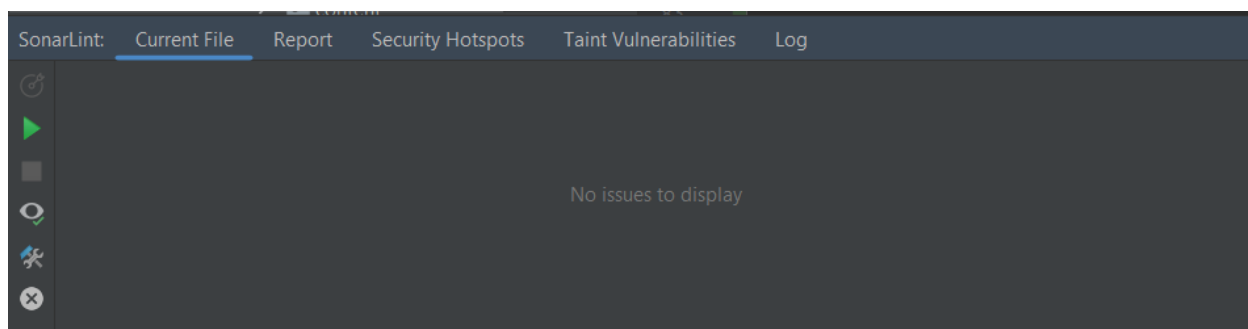


Рисунок 13 – Результаты проверки кода с помощью Sonar

Проверка показала, что код не содержит нарушений.

Заключение

В рамках данной дипломной работы было разработано мобильное Android-приложение WorkTracker.

Был проведен анализ предметной области учета рабочего времени. Был составлен список конкурентов, проанализированы их сильные и слабые стороны.

Перед разработкой программного обеспечения было произведено планирование разработки, а также функционально-стоимостной анализ разработки программных средств. Дата начала работ была определена как 01.03.24, а дата окончания работ – 27.05.24. Бюджет проекта составил 123 520 рублей.

После реализации приложение было протестировано с использованием диаграммы S&T, а также с использованием метрик SLOC и плагина Sonar. Тестирование показало корректность работы ПО.

В ходе разработки была применена новая библиотека для создания пользовательского интерфейса Jetpack Compose. Данная библиотека позволила упростить реализацию спроектированной архитектуры, уменьшив возможное количество ошибок. С помощью Jetpack Compose был реализован простой и удобный интерфейс, позволяющий конкурировать на рынке мобильных приложений. Обнаруженный недостаток при создании графа навигации был решен использованием дополнительной библиотеки Compose Destination.

Данная библиотека имеет совместимость с другими часто используемыми библиотеками, такими как Hilt, Room и Coroutines. Это позволяет использовать Jetpack Compose не только в новых, но и в уже существующих проектах.

Исходя из вышесказанного, можно сделать вывод, что Jetpack Compose является полноценной заменой View и может быть использован при разработке Android-приложений.

Список использованных источников

1. Прошкина, Е. Н. Язык моделирования информационных процессов UML 2 / Прошкина Е. Н., Балашова И. Ю. – П. : Изд-во ПГУ, 2018 – 112 с.
2. Сравниваем готовые решения для реализации MVI-архитектуры на Android // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/simbirsoft/articles/661185/> (дата обращения: 01.06.2024)
3. Использование шаблона MVVM (Model-View-ViewModel) в Android // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/dataart/articles/272737/> (дата обращения: 01.06.2024)
4. Kotlin Docs // Kotlinlang [Электронный ресурс]. URL: <https://kotlinlang.org/docs/home.html> (дата обращения: 01.06.2024)
5. Compose Docs // Android Developers [Электронный ресурс]. URL: <https://developer.android.com/develop/ui/compose> (дата обращения: 01.06.2024)
6. Save data in a local database using Room // Android Developers [Электронный ресурс]. URL: <https://developer.android.com/training/data-storage/room> (дата обращения: 01.06.2024)
7. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2021. – 352 с.
8. Dependency injection with Hilt // Android Developers [Электронный ресурс]. URL: <https://developer.android.com/training/dependency-injection/hilt-android> (дата обращения: 01.06.2024)
9. Coroutines // Kotlinlang [Электронный ресурс]. URL: <https://kotlinlang.org/docs/coroutines-overview.html> (дата обращения: 01.06.2024)
10. Виды и типы тестирования: подробный разбор // Test engineer [Электронный ресурс]. URL: <https://testengineer.ru/vidy-tipy-testirovaniya/> (дата обращения: 01.06.2024)

11. State & Transition Diagram – что это и как применять // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/548192/> (дата обращения: 01.06.2024)

12. Build local unit tests // Android Developers [Электронный ресурс]. URL: <https://developer.android.com/training/testing/local-tests> (дата обращения: 01.06.2024)

13. Гурьянов, Л. В. Выпускная квалификационная работа: стандарты и программные технологии. Учебно-методическое пособие для бакалавров по направлениям 09.03.04 «Программная инженерия», 09.03.02 «Информационные системы и технологии» / Гурьянов Л. В., Князев В. Н., Балашова И. Ю., Дзюба Е. А., Попова Н. А., Такташкин Д. В., Самуйлов С. В., Макарычев П. П. – П. : Изд-во ПГУ, 2020 – 182с.

Приложение А
(справочное)
Глоссарий

View – класс, наследники которого позволяют создавать графический интерфейс для Android-приложений.

UI/UX – это проектирование любых пользовательских интерфейсов в которых удобство использования так же важно как и внешний вид.

Material Design – дизайн-система для создания интерфейсов программного обеспечения и приложений, разработанная компанией Google.

Декларативный подход – подход к решению задачи, при котором задаётся спецификация решения, то есть описывается ожидаемый результат, а не способ его получения.

Императивный подход – подход к решению задачи, при котором задаётся способ получения необходимого результата.

SQLite – компактная встраиваемая СУБД.

SOLID (Single responsibility, Open–closed, Liskov substitution, Interface segregation, Dependency inversion) – мнемонический акроним, введённый Майклом Фэзерсом для первых пяти принципов, названных Робертом Мартином, которые означали 5 основных принципов объектно-ориентированных проектирования и программирования.

Приложение Б
(рекомендуемое)
Код программы

Листинг Б.1 – Файл Application.kt

```
package ru.kolyagin.worktracker.ui

import dagger.hilt.android.HiltAndroidApp
import ru.kolyagin.worktracker.ui.notifications.ActivityCallbacks
import javax.inject.Inject

@HiltAndroidApp
class Application : android.app.Application() {

    @Inject
    lateinit var activityCallbacks: ActivityCallbacks

    override fun onCreate() {
        super.onCreate()
        registerActivityLifecycleCallbacks(activityCallbacks)
    }
}
```

Листинг Б.2 – Файл MainActivity.kt

```
package ru.kolyagin.worktracker.ui

import android.os.Bundle
import androidx.activity.compose.setContent
import androidx.appcompat.app.AppCompatActivity
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.MaterialTheme
import androidx.compose.material.ModalBottomSheetValue
import androidx.compose.material.Surface
import androidx.compose.material.rememberModalBottomSheetState
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.navigation.compose.rememberNavController
import androidx.navigation.plusAssign
import com.google.accompanist.navigation.material.BottomSheetNavigator
import com.google.accompanist.navigation.material.ExperimentalMaterialNavigationApi
import com.google.accompanist.navigation.material.ModalBottomSheetLayout
import com.google.accompanist.systemuicontroller.rememberSystemUiController
import com.ramcosta.composedestinations.DestinationsNavHost
import com.ramcosta.composedestinations.animations.rememberAnimatedNavHostEngine
import dagger.hilt.android.AndroidEntryPoint
import ru.kolyagin.worktracker.ui.theme.WorkTrackerTheme

@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
    @OptIn(ExperimentalMaterialNavigationApi::class,
        ExperimentalAnimationApi::class,
        ExperimentalMaterialApi::class)
```

```

    )
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            WorkTrackerTheme {

rememberSystemUiController().setStatusBarColor(MaterialTheme.colors.primaryVariant)

                val navController = rememberNavController()

                val sheetState = rememberModalBottomSheetState(
                    initialValue = ModalBottomSheetValue.Hidden,
                    skipHalfExpanded = true
                )
                val modalBottomNavigator = remember {
BottomSheetNavigator(sheetState) }
                navController.navigatorProvider += modalBottomNavigator
                ModalBottomSheetLayout(
                    bottomSheetNavigator = modalBottomNavigator,
                    sheetShape = RoundedCornerShape(topStart = 40.dp,
topEnd = 40.dp),
                ) {
                    Surface(
                        modifier = Modifier.fillMaxSize(),
                        color = MaterialTheme.colors.background
                    ) {
                        DestinationsNavHost(
                            modifier = Modifier
                                .fillMaxSize(),
                            navController = navController,
                            navGraph = NavGraphs.root,
                            engine = rememberAnimatedNavHostEngine()
                        )
                    }
                }
            }
        }
    }
}

```

Листинг Б.3 – Файл MainScreen.kt

```

package ru.kolyagin.worktracker.ui.main

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.Card
import androidx.compose.material.Icon

```

```

import androidx.compose.material.IconButton
import androidx.compose.material.MaterialTheme
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.DateRange
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.ramcosta.composedestinations.annotation.Destination
import com.ramcosta.composedestinations.annotation.RootNavGraph
import com.ramcosta.composedestinations.navigation.DestinationsNavigator
import ru.kolyagin.worktracker.R
import ru.kolyagin.worktracker.domain.models.Time
import ru.kolyagin.worktracker.ui.destinations.SettingsScreenDestination
import
ru.kolyagin.worktracker.ui.destinations.StatisticBottomSheetDestination
import ru.kolyagin.worktracker.ui.main.content.DinneringScreenContent
import ru.kolyagin.worktracker.ui.main.content.PauseScreenContent
import ru.kolyagin.worktracker.ui.main.content.ResultsScreenContent
import ru.kolyagin.worktracker.ui.main.content.WorkStartScreenContent
import ru.kolyagin.worktracker.ui.main.content.WorkingScreenContent
import ru.kolyagin.worktracker.ui.settings.models.PeriodPart
import ru.kolyagin.worktracker.ui.theme.OnPrimaryHighEmphasis
import ru.kolyagin.worktracker.ui.utils.BaseMaterialTimePickerBuilder
import ru.kolyagin.worktracker.ui.utils.rememberFragmentManager
import ru.kolyagin.worktracker.ui.views.Spacer
import ru.kolyagin.worktracker.ui.views.TopBar

@RootNavGraph(start = true)
@Destination
@Composable
fun MainScreen(
    navigator: DestinationsNavigator,
    viewModel: MainViewModel = hiltViewModel()
) {
    val fragmentManager = rememberFragmentManager()
    val state by viewModel.screenState.collectAsStateWithLifecycle()
    val context = LocalContext.current
    LaunchedEffect(Unit) {
        viewModel.event.collect { event ->
            when (event) {
                is MainEvent.OpenSettings -> {
                    navigator.navigate(SettingsScreenDestination.route)
                }

                is MainEvent.AddEventTime -> {
                    BaseMaterialTimePickerBuilder
                        .setHour(12)
            }
        }
    }

```

```

        .setMinute(0)
        .build().apply {
            addOnPositiveButtonClickListener {
                viewModel.onTimePicked(Time(hour,
minute), PeriodPart.START)
                BaseMaterialTimePickerBuilder
                    .setHour(14)
                    .setMinute(0)
                    .build().apply {
                        addOnPositiveButtonClickListener
{
viewModel.onTimePicked(Time(hour, minute), PeriodPart.END)
                    }
                    show(fragmentManager,
"MainScreen")
                }
            }
        }
        show(fragmentManager, "MainScreen")
    }
}

is MainEvent.ChangeEventTime -> {
    BaseMaterialTimePickerBuilder
        .setHour(event.timeStart.hours)
        .setMinute(event.timeStart.minutes)
        .build().apply {
            addOnPositiveButtonClickListener {
                viewModel.onTimeChanging(Time(hour,
minute), PeriodPart.START)
                BaseMaterialTimePickerBuilder
                    .setHour(event.timeEnd.hours)
                    .setMinute(event.timeEnd.minutes)
                    .build().apply {
                        addOnPositiveButtonClickListener
{
viewModel.onTimeChanging(Time(hour, minute), PeriodPart.END)
                    }
                    show(fragmentManager,
"MainScreen")
                }
            }
        }
        show(fragmentManager, "MainScreen")
    }
}

}

}

val toolbarHeight = 100.dp
val goToStatistic = remember {
    {
        navigator.navigate(StatisticBottomSheetDestination) {
            launchSingleTop = true
        }
    }
}
Box(

```

```

        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(),
    ) {
        val scrollState = rememberScrollState()
        Image(
            modifier = Modifier
                .fillMaxSize(),
            painter = painterResource(id = R.drawable.header),
            contentScale = ContentScale.FillBounds,
            contentDescription = null
        )
        Column(Modifier.verticalScroll(scrollState)) {
            TopBar(
                modifier = Modifier
                    .fillMaxWidth()
                    .height(toolbarHeight),
                title = stringResource(id = R.string.work_periods)
            ) {
                IconButton(
                    onClick = goToStatistic
                ) {
                    Icon(
                        imageVector = Icons.Default.DateRange,
                        contentDescription = null,
                        tint = OnPrimaryHighEmphasis
                    )
                }

                IconButton(onClick = viewModel::onClickOpenSettings) {
                    Icon(
                        painter = painterResource(id =
R.drawable.settings),
                        contentDescription = null,
                        tint = Color.Unspecified
                    )
                }
            }
            Spacer(size = 40.dp)
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .fillMaxHeight()
                    .clip(RoundedCornerShape(topStart = 56.dp, topEnd =
56.dp))
                    .background(MaterialTheme.colors.background)
                    .padding(top = 16.dp)
            ) {
                state.days.forEach {
                    Card(
                        modifier = Modifier
                            .fillMaxSize()
                            .padding(start = 16.dp, end = 16.dp, bottom =
12.dp),
                        shape = RoundedCornerShape(40.dp),
                        backgroundColor = OnPrimaryHighEmphasis,
                        elevation = 0.dp
                    ) {

```

```

        when (val currentState = it) {
            is CardState.WorkStart ->
                WorkStartScreenContent(
                    state = currentState,
                    onClickStartWork =
viewModel::onClickStartWork,
                    onClickDeleteEvent =
viewModel::onClickDeleteEvent,
                    onAddPeriod = viewModel::onClickAddEvent,
                    onClickEvent = viewModel::onClickEvent,
                    onClickShowMore = goToStatistic,
                )

            is CardState.Dinnering ->
                DinneringScreenContent(
                    state = currentState,
                    onClickReturnFromDinner =
viewModel::onClickReturnFromDinner,
                    onClickDeleteEvent =
viewModel::onClickDeleteEvent,
                    onClickEvent = viewModel::onClickEvent,
                    onAddPeriod = viewModel::onClickAddEvent,
                    onClickEndWork =
viewModel::onClickFinishWork,
                    onClickShowMore = goToStatistic,
                )

            is CardState.Working -> WorkingScreenContent(
                state = currentState,
                onClickStartPause =
viewModel::onClickStartPause,
                onClickGoToDinner =
viewModel::onClickGoToDinner,
                onClickDeleteEvent =
viewModel::onClickDeleteEvent,
                onClickEvent = viewModel::onClickEvent,
                onAddPeriod = viewModel::onClickAddEvent,
                onClickEndWork =
viewModel::onClickFinishWork,
                onClickShowMore = goToStatistic,
            )

            is CardState.Pause -> PauseScreenContent(
                state = currentState,
                onClickEndPause =
viewModel::onClickEndPause,
                onClickDeleteEvent =
viewModel::onClickDeleteEvent,
                onClickEvent = viewModel::onClickEvent,
                onAddPeriod = viewModel::onClickAddEvent,
                onClickEndWork =
viewModel::onClickFinishWork,
                onClickShowMore = goToStatistic,
            )

            is CardState.Results -> ResultsScreenContent(
                state = currentState,

```

```
viewModel::onClickStartWork,
viewModel::onClickDeleteEvent,
    )
    }
    }
    }
    }
    }
    }
    }
    }
    }
    }
```

Листинг Б.4 – Файл MainScreenState.kt

```
package ru.kolyagin.worktracker.ui.main

import kotlinx.collections.immutable.PersistentList
import kotlinx.collections.immutable.persistentListOf
import ru.kolyagin.worktracker.domain.models.TimeWithSeconds
import ru.kolyagin.worktracker.domain.models.WorkEvent
import ru.kolyagin.worktracker.domain.models.WorkStatistic
import ru.kolyagin.worktracker.utils.base.State
import java.time.DayOfWeek

data class MainScreenState(
    val days: PersistentList<CardState> = persistentListOf()
) : State()

sealed class CardState(open val day: DayOfWeek) {

    /**
     * Состояние перед началом рабочего периода
     */
    data class WorkStart(
        override val day: DayOfWeek,
        val buttonActive: Boolean,
        val buttonStartEarly: Boolean,
        val events: PersistentList<WorkEvent>,
    ) : CardState(day)
}
```

```

        val time: TimeWithSeconds? = null,
        val late: Boolean = false
    ) : CardState(day)

/**
 * Состояние во время работы
 */
data class Working(
    override val day: DayOfWeek,
    val events: PersistentList<WorkEvent>,
    val time: TimeWithSeconds,
    val isDinnerEnable: Boolean = true,
    val overwork: Boolean = false
) : CardState(day)

/**
 * Состояние во время рабочей паузы
 */
data class Pause(
    override val day: DayOfWeek,
    val events: PersistentList<WorkEvent>,
    val time: TimeWithSeconds
) : CardState(day)

/**
 * Состояние во время обеда в рабочий период
 */
data class Dinnering(
    override val day: DayOfWeek,
    val events: PersistentList<WorkEvent>,
    val time: TimeWithSeconds
) : CardState(day)

/**
 * Состояние после завершения последнего периода работы

```

```

        */
    data class Results(
        override val day: DayOfWeek,
        val statistic: WorkStatistic,
        val events: PersistentList<WorkEvent>,
    ) : CardState(day)
}

```

Листинг Б.5 – Файл MainEvent.kt

```

package ru.kolyagin.worktracker.ui.main

import ru.kolyagin.worktracker.domain.models.Time
import ru.kolyagin.worktracker.utils.base.Event

sealed class MainEvent : Event() {
    object OpenSettings : MainEvent()
    object AddEventTime : MainEvent()
    class ChangeEventTime(val timeStart: Time, val timeEnd: Time) :
        MainEvent()
}

```

Листинг Б.6 – Файл MainViewModel.kt

```

package ru.kolyagin.worktracker.ui.main

import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.collections.immutable.PersistentList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toPersistentList
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.isActive
import ru.kolyagin.worktracker.domain.models.DayWorkInfo
import ru.kolyagin.worktracker.domain.models.Time
import
ru.kolyagin.worktracker.domain.models.Time.Companion.toTimeWithSeconds
import ru.kolyagin.worktracker.domain.models.TimeWithSeconds

```

```

import
ru.kolyagin.worktracker.domain.models.TimeWithSeconds.Companion.toSeconds

import
ru.kolyagin.worktracker.domain.models.TimeWithSeconds.Companion.toTimeWithSeconds

import ru.kolyagin.worktracker.domain.models.WorkEvent
import ru.kolyagin.worktracker.domain.models.WorkState
import ru.kolyagin.worktracker.domain.repositories.PreferenceRepository
import ru.kolyagin.worktracker.domain.repositories.ScheduleRepository
import
ru.kolyagin.worktracker.domain.repositories.WorkStatisticRepository
import ru.kolyagin.worktracker.ui.notifications.NotificationsManager
import ru.kolyagin.worktracker.ui.settings.models.PeriodPart
import ru.kolyagin.worktracker.utils.Constants
import ru.kolyagin.worktracker.utils.base.BaseViewModel
import ru.kolyagin.worktracker.utils.log.Logger
import java.time.DayOfWeek
import java.time.LocalDate
import java.time.LocalTime
import java.time.temporal.ChronoUnit
import javax.inject.Inject

@HiltViewModel
class MainViewModel @Inject constructor(
    logger: Logger,
    private val scheduleRepository: ScheduleRepository,
    private val preferenceRepository: PreferenceRepository,
    private val workStatisticRepository: WorkStatisticRepository,
    private val notificationsManager: NotificationsManager
) : BaseViewModel<MainScreenState, MainEvent>(MainScreenState(), logger)
{
    private var timerJob: Job? = null
    private var schedule: DayWorkInfo? = null
    private var events: Map<Int, List<WorkEvent>> = mapOf()

    private var selectedDayOfWeek: Int? = null
    private var selectedWorkEvent: WorkEvent? = null

```

```

private var addingEvent: WorkEvent = WorkEvent(
    id = 0,
    timeStart = Time(13, 0),
    timeEnd = Time(14, 0),
    name = Constants.BREAK,
    isDinner = false
)

init {
    updateState {
        val listOfDays = DayOfWeek.values().toList()
        it.copy(days = listOfDays.map { day ->
            CardState.WorkStart(
                day = day,
                buttonActive = false,
                buttonStartEarly = true,
                events = persistentListOf(),
                time = null
            )
        }).toPersistentList())
    }
}

init {
    val listOfDays = DayOfWeek.values().toList()
    scheduleRepository.schedule().subscribe {
        timerJob?.cancel()
        timerJob = launchViewModelScope {
            val currentDay = LocalDate.now().dayOfWeek.ordinal
            schedule = it.getOrNull(currentDay)
            events = it.associate { Pair(it.day.ordinal, it.events) }
            val startWorkRanges =
                schedule?.let { schedule ->
                    getListOfTimeRangesStartWork(schedule)
                }.map {
it.start.toTimeWithSeconds()..it.endInclusive.toTimeWithSeconds() }

```

```

        }?.sortedBy { it.start }
    val workingRanges =
        schedule?.let { schedule ->
            getListOfTimeRangesWorking(schedule)
                ?.map {
it.start.toTimeWithSeconds()..it.endInclusive.toTimeWithSeconds() }
                }?.sortedBy { it.start }
        val currentEvents =
events[currentDay]?.toPersistentList() ?: persistentListOf()

    do {
        val isDinnerEnable = schedule?.isDinnerInclude ==
true &&
            preferenceRepository.isDinnerEnableToday
        val currentTime = LocalTime.now()
        val time = currentTime.toTimeWithSeconds()
        val currentState = getCurrentState(
            startWorkRanges = startWorkRanges,
            workingRanges = workingRanges,
            time = time,
            currentDayOfWeek = listOfDays[currentDay],
            currentTime = currentTime,
            totalTime = schedule?.totalTime,
            isDinnerEnable = isDinnerEnable,
            workEvents = currentEvents
        )
        val daysWithoutCurrent = (listOfDays.subList(
            currentDay + 1, listOfDays.size
        ) + listOfDays.subList(0, currentDay)).map { day ->
            val dayEvent =
                events[day.ordinal]?.toPersistentList() ?:
persistentListOf()

            CardState.WorkStart(
                day = day,
                buttonActive = false,
                buttonStartEarly = true,
                events = dayEvent,

```

```

        time = null
    )
}
updateState { state ->
    state.copy(
        days = (listOf(currentState) +
daysWithoutCurrent).toPersistentList()
    )
}
    delay(ChronoUnit.MILLIS.between(LocalTime.now(),
currentTime.plusSeconds(1)))
    } while (isActive)
}
}
}

fun onClickOpenSettings() {
    trySendEvent(MainEvent.OpenSettings)
}

fun onClickStartWork() {
    preferenceRepository.currentWorkState = WorkState.Working
    updateCardState()
    notificationsManager.rescheduleNotifications()
}

fun onClickFinishWork() {
    launchViewModelScope {
        when (preferenceRepository.currentWorkState) {
            WorkState.Pause -> {
                updateTimeOfPause()
            }

            WorkState.Dinner -> {
                updateTimeOfPause()
            }
        }
    }
}

```

```

        WorkState.Working -> {
            workStatisticRepository.addWorkTime(
                LocalDate.now(),
                LocalTime.now()
                    .toTimeWithSeconds() -
                preferenceRepository.timeOfCurrentStateSet
            )
        }

        else -> {}
    }

    preferenceRepository.currentWorkState = WorkState.Worked
    updateCardState()
    notificationsManager.rescheduleNotifications()
}

fun onClickStartPause() {
    launchViewModelScope {
        workStatisticRepository.addWorkTime(
            LocalDate.now(),
            LocalTime.now().toTimeWithSeconds() -
            preferenceRepository.timeOfCurrentStateSet
        )
        preferenceRepository.currentWorkState = WorkState.Pause
        updateCardState()
        notificationsManager.rescheduleNotifications()
    }
}

fun onClickEndPause() {
    updateTimeOfPause()
    preferenceRepository.currentWorkState = WorkState.Working
    updateCardState()
    notificationsManager.rescheduleNotifications()
}

```

```

    }

    fun onClickGoToDinner() {
        launchViewModelScope {
            workStatisticRepository.addWorkTime(
                LocalDate.now(),
                LocalTime.now().toTimeWithSeconds() -
                preferenceRepository.timeOfCurrentStateSet
            )
            preferenceRepository.currentWorkState = WorkState.Dinner
            preferenceRepository.isDinnerEnableToday = false
            updateCardState()
            notificationsManager.rescheduleNotifications()
        }
    }

    fun onClickReturnFromDinner() {
        onClickEndPause()
    }

    private fun updateTimeOfPause() {
        val pauseStart = preferenceRepository.timeOfCurrentStateSet
        val pauseStop = LocalTime.now().toTimeWithSeconds()
        calculateAndSavePauses(pauseStart, pauseStop)
    }

    private fun calculateAndSavePauses(
        pauseStart: TimeWithSeconds,
        pauseStop: TimeWithSeconds
    ) = launchViewModelScope {
        val listOfPauses =

        (events[LocalDate.now().dayOfWeek.ordinal]?.toPersistentList() ?:
        persistentListOf())
            .map {
                it.timeStart.toTimeWithSeconds()..it.timeEnd.toTimeWithSeconds() }
        val pause = pauseStart..pauseStop
    }

```

```

        listOfPauses.foldRight(TimeWithSeconds.fromSeconds(0)) { elem,
plannedTime ->
            when {
                elem.start in pause && elem.endInclusive in pause -> {
                    plannedTime + (elem.endInclusive - elem.start)
                }

                elem.start in pause -> {
                    plannedTime + (pauseStop - elem.start)
                }

                elem.endInclusive in pause -> {
                    plannedTime + elem.endInclusive - pauseStart
                }

                pauseStart in elem && pauseStop in elem -> {
                    plannedTime + (pauseStop - pauseStart)
                }

                else -> {
                    plannedTime
                }
            }
        }.let { plannedTime ->
            workStatisticRepository.addPlannedPauseTime(LocalDate.now(),
plannedTime)

            val unplanned = (pauseStop - pauseStart) - plannedTime
            if (unplanned.toSeconds() > 0)

workStatisticRepository.addUnplannedPauseTime(LocalDate.now(), unplanned)
        }
    }

fun onClickDeleteEvent(workEvent: WorkEvent, day: Int) {
    launchViewModelScope {
        if (workEvent.isDinner) {

```

```

        scheduleRepository.setDinner(day, false)
    }
    scheduleRepository.deleteWorkEvent(workEvent)
}

fun onClickEvent(day: Int, workEvent: WorkEvent) {
    selectedDayOfWeek = day
    selectedWorkEvent = workEvent
    trySendEvent(MainEvent.ChangeEventTime(workEvent.timeStart,
workEvent.timeEnd))
}

fun onClickAddEvent(day: Int) {
    selectedDayOfWeek = day
    trySendEvent(MainEvent.AddEventTime)
}

fun onTimePicked(time: Time, periodPart: PeriodPart) {
    selectedDayOfWeek?.let { dayOfWeek ->
        launchViewModelScope {
            addingEvent.let {
                if (periodPart == PeriodPart.START) {
                    addingEvent = it.copy(timeStart = time)
                } else {
                    addingEvent = it.copy(timeEnd = time)
                    if (addingEvent.timeStart >= addingEvent.timeEnd)
{
                        addingEvent = it.copy(
                            timeStart = addingEvent.timeEnd, timeEnd
= addingEvent.timeStart
                        )
                    }
                    scheduleRepository.addWorkEvent(dayOfWeek,
addingEvent)
                }
            }
        }
    }
}

```

```

        }
    }
}

fun onTimeChanging(time: Time, periodPart: PeriodPart) {
    selectedDayOfWeek?.let { dayOfWeek ->
        launchViewModelScope {
            var newEvent: WorkEvent = addingEvent
            selectedWorkEvent?.let {
                newEvent = if (periodPart == PeriodPart.START) {
                    it.copy(timeStart = time)
                } else {
                    it.copy(timeEnd = time)
                }
                if (newEvent.timeStart >= newEvent.timeEnd) {
                    newEvent =
                        it.copy(timeStart = newEvent.timeEnd, timeEnd
= newEvent.timeStart)
                }
                scheduleRepository.setWorkEventTime(newEvent,
dayOfWeek)
            }
            selectedWorkEvent = newEvent
        }
    }
}

private fun getListOfTimeRangesStartWork(currentSchedule:
DayWorkInfo): List<ClosedRange<Time>>? {
    val startWorkTime = currentSchedule.periods.minByOrNull {
it.timeStart }?.timeStart
    val startTimes = currentSchedule.periods.map { it.timeStart
}.drop(1)
    val endTimes = currentSchedule.periods.map { it.timeEnd }
    val periods = startTimes.mapIndexed { index, time ->
        endTimes[index]..time
    }
}

```

```

        return startWorkTime?.let {
            periods + listOf(
                Time.fromMinutes(0)..startWorkTime
            )
        }
    }

    private fun getListOfTimeRangesWorking(currentSchedule: DayWorkInfo):
    List<ClosedRange<Time>>? {
        return currentSchedule.periods.map { it.timeStart..it.timeEnd
    }.takeIf { it.isNotEmpty() }
    }

    private fun updateCardState() = launchViewModelScope {
        val currentDay = LocalDate.now().dayOfWeek.ordinal
        val startWorkRanges =
            schedule?.let { schedule ->
                getListOfTimeRangesStartWork(schedule)
                    ?.map {
it.start.toTimeWithSeconds()..it.endInclusive.toTimeWithSeconds() }
                    }?.sortedBy { it.start }
        val workingRanges = schedule?.let { schedule ->
            getListOfTimeRangesWorking(schedule)
                ?.map {
it.start.toTimeWithSeconds()..it.endInclusive.toTimeWithSeconds() }
                }?.sortedBy { it.start }
        val currentTime = LocalTime.now()
        val time = currentTime.toTimeWithSeconds()
        val listOfDays = DayOfWeek.values().toList()
        val isDinnerEnable = schedule?.isDinnerInclude == true &&
            preferenceRepository.isDinnerEnableToday
        val currentState = getCurrentState(
            startWorkRanges,
            workingRanges,
            time,
            listOfDays[currentDay],
            currentTime,

```

```

        schedule?.totalTime,
        isDinnerEnable,
        events[currentDay]?.toPersistentList() ?: persistentListOf()
    )
    val daysWithoutCurrent = (listOfDays.subList(
        currentDay + 1, listOfDays.size
    ) + listOfDays.subList(0, currentDay)).map { day ->

        val dayEvent = events[day.ordinal]?.toPersistentList() ?:
persistentListOf()
        CardState.WorkStart(
            day = day,
            buttonActive = false,
            buttonStartEarly = true,
            events = dayEvent,
            time = null
        )
    }
    updateState { state ->
        state.copy(
            days = (listOf(currentState) +
daysWithoutCurrent).toPersistentList()
        )
    }
}

private suspend fun getCurrentState(
    startWorkRanges: List<ClosedRange<TimeWithSeconds>>?,
    workingRanges: List<ClosedRange<TimeWithSeconds>>?,
    time: TimeWithSeconds,
    currentDayOfWeek: DayOfWeek,
    currentTime: LocalTime,
    totalTime: Time?,
    isDinnerEnable: Boolean,
    workEvents: PersistentList<WorkEvent>
) = when {

```

```

        startWorkRanges == null || workingRanges == null || totalTime ==
null -> {

            CardState.WorkStart(
                day = currentDayOfWeek,
                buttonActive = false,
                buttonStartEarly = true,
                events = workEvents,
                time = null
            )
        }

        startWorkRanges.any { time in it } -> {
            if (preferenceRepository.currentWorkState ==
WorkState.Worked) {
                preferenceRepository.currentWorkState =
WorkState.NotWorking
            }
            getStateForStartWork(
                startWorkRanges,
                currentTime,
                time,
                currentDayOfWeek,
                totalTime,
                isDinnerEnable,
                workEvents
            )
        }

        workingRanges.any { time in it } -> {
            getStateForWorking(
                workingRanges,
                currentTime,
                time,
                currentDayOfWeek,
                totalTime,
                isDinnerEnable,

```

```

        workEvents
    )
}

else -> {
    getStateForEndWork(currentDayOfWeek, currentTime, totalTime,
isDinnerEnable, workEvents)
}
}

private suspend fun getStateForStartWork(
    startWorkRanges: List<ClosedRange<TimeWithSeconds>>,
    currentTime: LocalTime,
    time: TimeWithSeconds,
    currentDayOfWeek: DayOfWeek,
    totalTime: Time,
    isDinnerEnable: Boolean,
    workEvents: PersistentList<WorkEvent>
) = when (preferenceRepository.currentWorkState) {
    WorkState.NotWorking, WorkState.Worked -> {
        val range = startWorkRanges.first { time in it }
        val timeBeforeWork =
            range.endInclusive - currentTime.toTimeWithSeconds()
        CardState.WorkStart(
            day = currentDayOfWeek,
            buttonActive = true,
            buttonStartEarly = true,
            events = workEvents,
            late = false,
            time = timeBeforeWork
        )
    }

    WorkState.Working -> {
        val statistic =
workStatisticRepository.getStatistic(LocalDate.now())

```

```

        if (totalTime.toTimeWithSeconds() > statistic.workTime) {
            CardState.Working(
                currentDayOfWeek,
                workEvents,
                statistic.workTime + currentTime.toTimeWithSeconds()
- preferenceRepository.timeOfCurrentStateSet,
                isDinnerEnable,
            )
        } else {
            CardState.Working(
                currentDayOfWeek,
                workEvents,
                statistic.workTime + currentTime.toTimeWithSeconds()
- preferenceRepository.timeOfCurrentStateSet -
totalTime.toTimeWithSeconds(),
                isDinnerEnable,
                true
            )
        }
    }

    WorkState.Dinner -> {
        CardState.Dinnering(
            currentDayOfWeek,
            workEvents,
            currentTime.toTimeWithSeconds() -
preferenceRepository.timeOfCurrentStateSet
        )
    }

    WorkState.Pause -> {
        CardState.Pause(
            currentDayOfWeek,
            workEvents,
            currentTime.toTimeWithSeconds() -
preferenceRepository.timeOfCurrentStateSet
        )
    }

```

```

    }
}

private suspend fun getStateForWorking(
    workingRanges: List<ClosedRange<TimeWithSeconds>>,
    currentTime: LocalTime,
    time: TimeWithSeconds,
    currentDayOfWeek: DayOfWeek,
    totalTime: Time,
    isDinnerEnable: Boolean,
    workEvents: PersistentList<WorkEvent>
) = when (preferenceRepository.currentWorkState) {
    WorkState.NotWorking -> {
        val range = workingRanges.first { time in it }
        val timeLate = currentTime.toTimeWithSeconds() - range.start
        CardState.WorkStart(
            day = currentDayOfWeek,
            buttonActive = true,
            buttonStartEarly = false,
            events = workEvents,
            time = timeLate,
            late = true
        )
    }

    WorkState.Working -> {
        val statistic =
workStatisticRepository.getStatistic(LocalDate.now())
        if (totalTime.toTimeWithSeconds() > statistic.workTime) {
            val timeWork =
                statistic.workTime + currentTime.toTimeWithSeconds()
- preferenceRepository.timeOfCurrentStateSet
            CardState.Working(
                day = currentDayOfWeek,
                events = workEvents,
                time = timeWork,

```

```

        isDinnerEnable = isDinnerEnable,
    )
} else {
    CardState.Working(
        currentDayOfWeek,
        workEvents,
        statistic.workTime + currentTime.toTimeWithSeconds()
- preferenceRepository.timeOfCurrentStateSet -
totalTime.toTimeWithSeconds(),
        isDinnerEnable,
        true
    )
}

}

WorkState.Worked -> {
    val rangeIndex = workingRanges.indexOfFirst { time in it }
    if (rangeIndex == workingRanges.size - 1) {
        val statistic =
workStatisticRepository.getStatistic(LocalDate.now())
        CardState.Results(
            day = currentDayOfWeek,
            statistic = statistic,
            events = workEvents
        )
    } else {
        val range = workingRanges[rangeIndex + 1]
        val time = range.start - currentTime.toTimeWithSeconds()
        CardState.WorkStart(
            day = currentDayOfWeek,
            buttonActive = true,
            buttonStartEarly = false,
            events = workEvents,
            time = time,
        )
    }
}

```

```

        }
    }

    WorkState.Dinner -> {
        CardState.Dinnering(
            currentDayOfWeek,
            workEvents,
            currentTime.toTimeWithSeconds() -
            preferenceRepository.timeOfCurrentStateSet
        )
    }

    WorkState.Pause -> {
        CardState.Pause(
            currentDayOfWeek,
            workEvents,
            currentTime.toTimeWithSeconds() -
            preferenceRepository.timeOfCurrentStateSet
        )
    }
}

private suspend fun getStateForEndWork(
    currentDayOfWeek: DayOfWeek,
    currentTime: LocalTime,
    totalTime: Time,
    isDinnerEnable: Boolean,
    workEvents: PersistentList<WorkEvent>
) =
    when (preferenceRepository.currentWorkState) {
        WorkState.NotWorking, WorkState.Worked -> {
            val statistic =
            workStatisticRepository.getStatistic(LocalDate.now())

            CardState.Results(
                day = currentDayOfWeek,
                statistic = statistic,
            )
        }
    }

```

```

        events = workEvents
    )
}

WorkState.Working -> {
    val statistic =
workStatisticRepository.getStatistic(LocalDate.now())
    if (totalTime.toTimeWithSeconds() > statistic.workTime) {
        CardState.Working(
            currentDayOfWeek,
            workEvents,
            statistic.workTime +
currentTime.toTimeWithSeconds() -
preferenceRepository.timeOfCurrentStateSet,
            isDinnerEnable,
        )
    } else {
        CardState.Working(
            currentDayOfWeek,
            workEvents,
            statistic.workTime +
currentTime.toTimeWithSeconds() -
preferenceRepository.timeOfCurrentStateSet -
totalTime.toTimeWithSeconds(),
            isDinnerEnable,
            true
        )
    }
}

WorkState.Dinner -> {
    CardState.Dinnering(
        currentDayOfWeek,
        workEvents,
        currentTime.toTimeWithSeconds() -
preferenceRepository.timeOfCurrentStateSet
    )
}

```

```

        WorkState.Pause -> {
            CardState.Pause(
                currentDayOfWeek,
                workEvents,
                currentTime.toTimeWithSeconds() -
                preferenceRepository.timeOfCurrentStateSet
            )
        }
    }
}

```

Листинг Б.7 – Файл ScheduleRepository.kt

```

package ru.kolyagin.worktracker.domain.repositories

import kotlinx.collections.immutable.ImmutableList
import kotlinx.coroutines.flow.Flow
import ru.kolyagin.worktracker.domain.models.DayWorkInfo
import ru.kolyagin.worktracker.domain.models.WorkEvent
import ru.kolyagin.worktracker.domain.models.WorkPeriod
import java.time.DayOfWeek

interface ScheduleRepository {
    fun schedule(): Flow<ImmutableList<DayWorkInfo>>

    suspend fun addPeriod(
        dayOfWeek: DayOfWeek
    )

    suspend fun deletePeriod(
        workPeriod: WorkPeriod,
    )

    suspend fun setTime(
        workPeriod: WorkPeriod, dayOfWeek: DayOfWeek
    )
}

```

```

suspend fun setDinner(
    dayOfWeek: Int, isDinnerInclude: Boolean
)

suspend fun addWorkEvent(
    dayOfWeek: Int, event: WorkEvent
)

suspend fun deleteWorkEvent(
    workEvent: WorkEvent
)

suspend fun deleteDinner(
    dayOfWeek: Int
)

suspend fun setWorkEventTime(
    workEvent: WorkEvent, dayOfWeek: Int
)
}

```

Листинг Б.8 – Файл ScheduleRepositoryImpl.kt

```

package ru.kolyagin.worktracker.data.repositories

import kotlinx.collections.immutable.ImmutableList
import kotlinx.collections.immutable.persistentListOf
import kotlinx.collections.immutable.toImmutableList
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.combine
import ru.kolyagin.worktracker.data.database.dao.ScheduleDao
import ru.kolyagin.worktracker.data.database.entities.WorkPeriodEntity
import ru.kolyagin.worktracker.data.database.entities.mapToDomain
import ru.kolyagin.worktracker.domain.models.DayWorkInfo
import ru.kolyagin.worktracker.domain.models.WorkEvent
import ru.kolyagin.worktracker.domain.models.WorkPeriod

```

```

import ru.kolyagin.worktracker.domain.models.mapToEntity
import ru.kolyagin.worktracker.domain.repositories.ScheduleRepository
import java.time.DayOfWeek
import javax.inject.Inject

class ScheduleRepositoryImpl @Inject constructor(
    private val scheduleDao: ScheduleDao
) : ScheduleRepository {
    override fun schedule(): Flow<ImmutableList<DayWorkInfo>> =
        // Комбинируем все потоки данные для формирования расписания
        combine(
            scheduleDao.getDaySchedule(),
            scheduleDao.getWorkPeriods(),
            scheduleDao.getWorkEvents()
        ) { dayScheduleEntities, workPeriodEntities, events ->
            // Группируем данные по дню
            val groups = workPeriodEntities.groupBy { it.day }
            val eventGroups = events.groupBy({ it.day }) {
it.mapToDomain() }
            // Создаём список с расписанием на каждый день
            dayScheduleEntities.map { scheduleEntity ->
                val eventsList =
eventGroups[scheduleEntity.day]?.toImmutableList()
                ?: persistentListOf()
                DayWorkInfo(
                    day = DayOfWeek.values()[scheduleEntity.day],
                    periods = groups[scheduleEntity.day]?.map {
it.mapToDomain() }
                        ?.toImmutableList()
                        ?: persistentListOf(),
                    isDinnerInclude =
scheduleEntity.isDinnerInclude,
                    events = eventsList
                )
            }.toImmutableList()
        }
}

```

```

override suspend fun setTime(
    workPeriod: WorkPeriod, dayOfWeek: DayOfWeek
) {
    scheduleDao.setTime(
        WorkPeriodEntity(
            workPeriod.id, workPeriod.timeStart,
workPeriod.timeEnd, dayOfWeek.ordinal
        )
    )
}

override suspend fun addWorkEvent(dayOfWeek: Int, event: WorkEvent)
{
    // Если переданный день = -1, то добавляем событие во все дни
    if (dayOfWeek == -1) {
        for (i in 0 until 6)
            scheduleDao.addEvent(
                i,
                event.name,
                event.isDinner,
                event.timeStart,
                event.timeEnd
            )
    } else {
        scheduleDao.addEvent(
            dayOfWeek,
            event.name,
            event.isDinner,
            event.timeStart,
            event.timeEnd
        )
    }
}

override suspend fun addPeriod(dayOfWeek: DayOfWeek) {
    scheduleDao.addPeriod(dayOfWeek.ordinal)
}

```

```

    }

    override suspend fun deletePeriod(workPeriod: WorkPeriod) {
        scheduleDao.deletePeriod(workPeriod.id)
    }

    override suspend fun setDinner(dayOfWeek: Int, isDinnerInclude:
Boolean) {
        scheduleDao.setDinner(dayOfWeek, isDinnerInclude)
    }

    override suspend fun deleteWorkEvent(workEvent: WorkEvent) {
        scheduleDao.deleteEvent(workEvent.id)
    }

    override suspend fun deleteDinner(dayOfWeek: Int) {
        scheduleDao.deleteDinner(dayOfWeek)
    }

    override suspend fun setWorkEventTime(
        workEvent: WorkEvent, dayOfWeek: Int
    ) {
        scheduleDao.changeEvent(
            workEvent.mapToEntity().copy(day = dayOfWeek)
        )
    }
}

```