**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное автономное образовательное   
учреждение высшего образования   
«Санкт-Петербургский политехнический университет Петра Великого»**

**ИНСТИТУТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

|  |  |
| --- | --- |
|  | **Допустить к защите**  Заместитель директора по  учебно-методической работе  \_\_\_\_\_\_\_\_\_\_\_ \_\_Е. Г. Конакина\_\_\_\_\_  (Подпись) (И.О.Ф.)  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |
|  |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Тема Разработка приложения прогноза погоды

специальность 09.02.07 группа 42919/3

Студент (ка) Кокин Н.В.

(подпись) (ФИО)

Руководитель

(подпись) (ФИО)

Санкт-Петербург

2022

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc104825483)

[1 Общая часть 5](#_Toc104825484)

[1.1 Описание предметной области 5](#_Toc104825485)

[1.2 Требование к приложению 5](#_Toc104825486)

[1.3 Анализ рынка существующих решений 6](#_Toc104825487)

[1.4 Аналитический обзор СУБД 7](#_Toc104825488)

[1.5 Обоснование и выбор методики, технологии и инструментальных средств разработки 16](#_Toc104825498)

[2 Специальная часть 20](#_Toc104825505)

[2.1 Описание структуры приложения 20](#_Toc104825506)

[2.2 Объектно-ориентированное проектирование системы 24](#_Toc104825509)

[2.3 Разработка пользовательского интерфейса 26](#_Toc104825512)

[2.4 Реализация системы 31](#_Toc104825516)

[3 Экономическая часть 32](#_Toc104825517)

[3.1 Область применения программного продукта и его преимущества перед аналогичным программным продуктом 32](#_Toc104825518)

[3.2 Трудоемкость разработки программного продукта, квалификация исполнителя и его оклад 32](#_Toc104825519)

[3.3 Расчет затрат на разработку 34](#_Toc104825520)

[3.4 Расчет затрат на внедрение 38](#_Toc104825526)

[3.5 Расчет цены и прибыли 41](#_Toc104825532)

[4 Техника безопасности и охрана труда 44](#_Toc104825533)

[4.1 Анализ условий труда программиста, работающего в ООО «Мониторинг Цен» 44](#_Toc104825534)

[4.2 Расчет искусственного освещения рабочего места программиста в ООО «Мониторинг Цен» 46](#_Toc104825535)

[4.3 Электробезопасность ООО «Мониторинг Цен» 48](#_Toc104825536)

[4.4 Пожарная безопасность в ООО «Мониторинг Цен» 50](#_Toc104825537)

[ЗАКЛЮЧЕНИЕ 53](#_Toc104825538)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 54](#_Toc104825539)

[Приложение А 56](#_Toc104825540)

[Приложение Б 58](#_Toc104825541)

[Приложение В 71](#_Toc104825542)

# ВВЕДЕНИЕ

Мобильные телефоны давно перестали быть чем-то необычным и, в настоящее время, набирают все большую популярность. Мобильные телефоны великолепно справляются со своей функцией – являются средством коммуникации между людьми. При этом, недавно появившиеся, но уже прочно вошедшие в нашу жизнь смартфоны настолько функциональны, что трудно сказать, чего они не умеют: это и плеер, и фотоаппарат, и возможность использования Интернет-ресурсов, и прочее. По сути, все смартфоны стали небольшой копией компьютера, который постоянно можно иметь при себе.

Клиент ООО «Мониторинг Цен» обратились с задачей на разработку приложения для просмотра прогноза погоды. С клиентом было принято решение на разработку мобильного приложения. Клиент предоставил интерфейс приложения.

Целью дипломного проекта является реализация мобильного приложения, который поможет отслеживать прогноз погоды из сервиса, предоставляющего доступ к этим данным.

# Общая часть

## Описание предметной области

ООО «Мониторинг Цен» - предприятие, которое занимается разработкой различных видов программного обеспечения, таких как, например, разработка парсеров сайтов и мобильных приложений.

При разработке приложения с клиентом обсуждается дизайн и функционал приложения. Если клиент не имеет готового дизайна, то формируется техническое задание к дизайну в зависимости от функционала и желаний заказчика и советов дизайнера.

При формировании технического задания функционала приложения разработчик смотрит на сложность реализации функционала, возможность реализации функционала и т.п.

Для разрабатываемого приложения клиентом был предоставлен интерфейс и требования по функционалу.

## Требование к приложению

Клиентом были выдвинуты следующие требования к функционалу мобильного приложения:

* Возможность поиска города;
* Почасовой просмотр прогноза погоды на день;
* Просмотр погоды по дням;
* Возможность просмотра подробного прогноза погоды за час;
* Возможность настраивать единицы измерения для некоторых показателей погоды;
* Возможность менять частоту обновления виджета на главном экране.

## Анализ рынка существующих решений

Проанализировав рынок, были найдены решения, которые имеют схожие задачи:

* Bastion7 Weather Live Wallpapers – это приложение позволяет смотреть погоду используя два источника данных: METAR and MET Norway и Dark Sky. На главном экране отображается погода на каждые два часа и погода на каждый день в течении недели, позволяет менять единицы измерения. Приложение имеет на фоне живые обои и позволяет поставить их на главный экран телефона;
* Geometric Weather – приложение отображает информацию о погоде в виде блоков: почасовая погода, погода по дням, информация о восходе и закате и дополнительная информация о погоде. Отображение блоков можно настраивать. На выбор имеет три погодных провайдера: accuweather.com, OpenWeather и MeteoFrance. Позволяет настраивать единицы измерения;
* Gismeteo – приложение прогноза погоды от одноименного провайдера, которое отображает прогноз погоды на каждые 3 часа, так же имеется ежедневный прогноз. Позволяет изменить язык приложения и единицы измерения. Отображает специфичные параметры погоды, например, геомагнитную обстановку;

У данных приложений имеется избыточная функциональность, у некоторых приложений имеются проблемы с производительностью. Решением этих проблем станет разработка собственного приложения, которое будет отображать необходимую информацию о погоде.

## Обоснование и выбор методики, технологии и инструментальных средств разработки

### Kotlin

Kotlin - [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), объектно-ориентированный, высокоуровневый [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) со строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ, работающий поверх [Java Virtual Machine](https://ru.wikipedia.org/wiki/Java_Virtual_Machine). Также компилируется в [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) и в [исполняемый код](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D0%BD%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) ряда платформ через инфраструктуру [LLVM](https://ru.wikipedia.org/wiki/LLVM). язык программирования общего назначения.

Авторы ставили целью создать язык более лаконичный и типобезопасный, чем [Java](https://ru.wikipedia.org/wiki/Java), и более простой, чем [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Следствием упрощения по сравнению со Scala стали также более быстрая [компиляция](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и лучшая поддержка языка в [IDE](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8). Язык полностью совместим с Java, что позволяет Java-разработчикам постепенно перейти к его использованию; в частности, язык также встраивается [Android](https://ru.wikipedia.org/wiki/Android), что позволяет для существующего android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

### Koin

Koin — библиотека, реализующая шаблон Dependency Injection (внедрение зависимости). Это библиотека внедрения зависимостей для разработчиков Kotlin, который создает экземпляры различных объектов приложения и внедряет их в классы.

### Coroutines

Coroutine (сопрограмма) - Удобная библиотека, позволяющая отправлять запросы на сервер, делать запросы к БД, выполнять тяжеловесные операции, не блокируя главный поток, что позволяет не зависать приложению, например, при загрузке данных с сервера. Сопрограмма – [Программный](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) модуль, особым образом организованный для обеспечения взаимодействия с другими модулями по принципу [кооперативной многозадачности](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C): модуль приостанавливается в определённой точке, сохраняя полное состояние (включая [стек вызовов](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2%D0%BE%D0%B2) и [счётчик команд](https://ru.wikipedia.org/wiki/%D0%A1%D1%87%D1%91%D1%82%D1%87%D0%B8%D0%BA_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4)), и передаёт управление другому, тот, в свою очередь, выполняет задачу и передаёт управление обратно, сохраняя свои стек и счётчик.

### Navigation Component

Navigation Component – Архитектурный компонент Navigation позволяет упростить реализацию навигации между экранами назначения (destinations) в приложении.

### SQLite (Room)

SQLite - это встроенная библиотека, которая реализует автономный, бессерверный, транзакционный механизм СУБД SQL.

SQLite не использует парадигмы [клиент-сервер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), то есть [движок](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%85%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой [библиотеку](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций ([API](https://ru.wikipedia.org/wiki/API)) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; [ACID](https://ru.wikipedia.org/wiki/ACID)-функции достигаются в том числе за счёт создания файла журнала.

Room – библииотека, позволяющая более просто работать с SQLite. Room выполняет большую часть своей работы во время компиляции, создавая API-интерфейс поверх встроенного SQLite API, поэтому вам не нужно работать с Cursor или ContentResolver.

### Android Studio

Android Studio – программа, являющаяся средой разработки приложений для мобильной платформы Android. Во время создания приложений и утилит, пользователь программного обеспечения может наблюдать за изменениями в проекте, в режиме реального времени.

В программу встроен эмулятор, позволяющий проверить корректную работу приложения на устройствах с разными экранами, с различными соотношениями сторон. Отличительная особенность эмулятора – просмотр приблизительных показателей производительности при запуске приложения на самых популярных устройствах.

В программе реализованы все современные средства для упаковки кода, его маркировки.

# Специальная часть

## Описание структуры приложения

На главном экране приложения отображается краткая информация о

На главном экране приложения будет отображаться краткая информация за каждый час текущего дня, за ближайшие дни и подробная информация за текущий час. Данные о погоде на ближайшие дни будут сохранены в БД, чтобы потом отобразить их, если у пользователя будет отсутствовать интернет.

Вверху окна будет кнопка поиска города, ведущая в окно поиска города, правее будет написан город, для которого сейчас отслеживается погода, справа будет кнопка настроек, ведущая в окно настроек.

В окне поиска будет отображаться список найденных городов по запросу на поиск, в нем можно выбрать город, погода которого будет отслеживаться.

В настройках можно открыть окно выбора единиц измерения для отображения для давления, скорости ветра и температуры. Так же там можно поменять погодного провайдера.

В пунктах «Инфологическая модель» и «Даталогическая модель» будет описано проектирование и реализации таблиц в базу данных.

### Инфологическая модель

Требуется разработать базу данных в которой будет храниться информация о погоде.

Для прогноза на день будет храниться дата дня, температура утром, температура ночью, скорость ветра, влажность, атмосферное давление, видимость, облачность.

Для прогноза за час будет храниться время, температура, скорость ветра, влажность, атмосферное давление, видимость и облачность.

* Сущность «Day Forecast» атрибуты: IdDayForecast, Date, DayTemperature, NightTemperature, WindSpeed, Humidity, AtmospherePressure, Visibility, Cloudness;

На рисунке 1 представлена блок-схема базы данных.

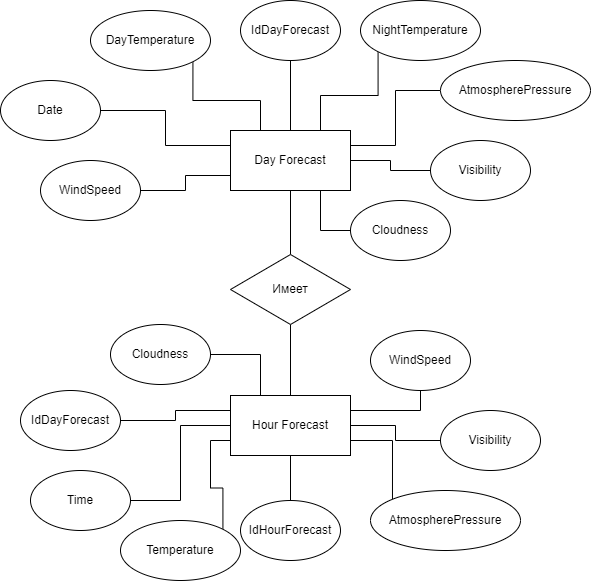


Рисунок 1 – Инфологическая модель

### Даталогическая модель

На основе Инфологической модели определим отношения между сущностями и типы данных для атрибутов.

* Сущность «Day Forecast» имеет отношение 1: М с сущностью «Hour Forecast»

#### Сущность «Day Forecast»

Имеются следующие типы данных для атрибутов:

* IdDayForecast. Числовой тип. Создает уникальный номер автоматически, когда новая запись вставляется в таблицу;
* Date. Тип даты;
* DayTemperature. Числовой тип. Диапазон должен быть от -100 до 100;
* NightTemperature. Числовой тип. Диапазон должен быть от -100 до 100;
* WindSpeed. Числовой тип. Диапазон должен быть от 0 до 250;
* Humidity. Числовой тип. Диапазон должен быть от 0 до 100;
* AtmospherePressure. Числовой тип. Диапазон должен быть от 600 до 1200;
* Visibility. Числовой тип. Диапазон должен быть от 0 до 1000;
* Cloudness. Числовой тип. Диапазон должен быть от 1 до 10.

#### Сущность «Hour Forecast»

Имеются следующие типы данных для атрибутов:

* IdHourForecast. Числовой тип. Создает уникальный номер автоматически, когда новая запись вставляется в таблицу;
* Date. Тип даты;
* Temperature. Числовой тип. Диапазон должен быть от -100 до 100;
* WindSpeed. Числовой тип. Диапазон должен быть от 0 до 250;
* Humidity. Числовой тип. Диапазон должен быть от 0 до 100;
* AtmospherePressure. Числовой тип. Диапазон должен быть от 600 до 1200;
* Visibility. Числовой тип. Диапазон должен быть от 0 до 1000;
* Cloudness. Числовой тип. Диапазон должен быть от 1 до 10;
* IdDayForecast. Числовой тип. Является внешним ключом. Связан с таблицей «Day Forecast».

На рисунке 2 представлена даталогическая модель.

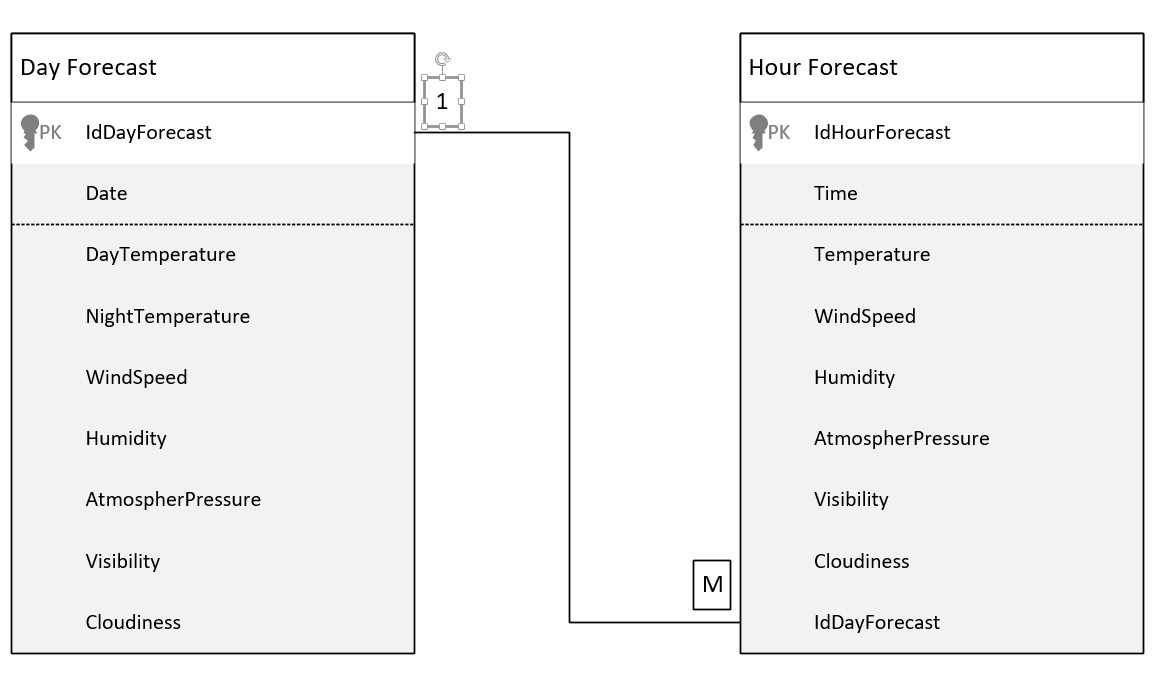


Рисунок 2 – Даталогическая модель

## Объектно-ориентированное проектирование системы

Проектирование программного продукта выполнялось на основе объектно-ориентированной разработки с использованием UML-диаграмм.

### Диаграмма прецедентов

На рисунке 3 представлен пользователь приложения прогноза погоды. В самой системе у него есть полный доступ к имеющемуся функционалу, с помощью которого он может осуществлять поиск города для отображения погоды в этом городе, менять настройки приложения, а так же выбирать час для отображения подробного прогноза.

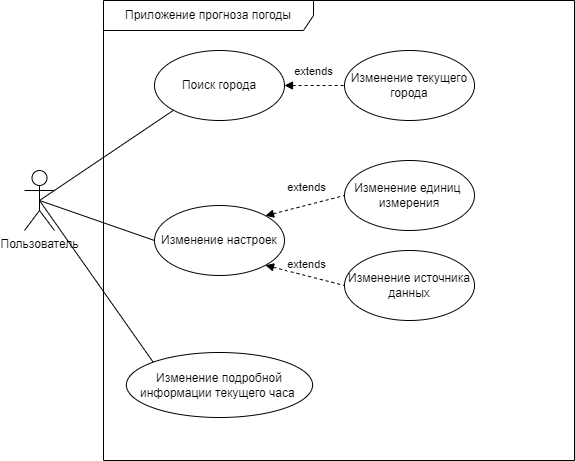


Рисунок 3 – Диаграмма прецедентов

### Диаграмма последовательности

На рисунке 4 рассмотрена работа функции «Добавления».

Когда сотрудник отдела заполнив форму нажимает на кнопку «добавление записи», клиентская часть передает запрос на сервер. Для начала эти данные проходят проверку на валидацию, после чего они направляются в базу данных. При создании у самой записи создается автоматически уникальный ключ.

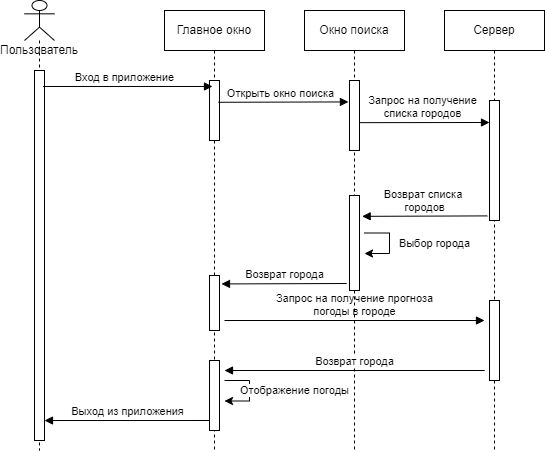


Рисунок 4 – Диаграмма последовательности

## Разработка пользовательского интерфейса

При разработке пользовательского интерфейса были созданы следующие окна:

* Главное окно
* Окно поиска
* Окно настроек
* Окно настроек единиц измерений
* Виджет для главного экрана

### Главная окно

На главном окне находятся почасовой прогноз погоды, прогноз на день, подробный прогноз на час (Рисунок 5).



Рисунок 5 – Главное окно

На панели инструментов расположены две кнопки: кнопка перехода на окно поиска и кнопка перехода на окно настроек, и название города шапке (Рисунок 6).



Рисунок 6 – Панель инструментов на главном окне

### Окно поиска

На панели инструментов находится поиск (Рисунок 7), при вводе названия города и нажатии на кнопку «поиск» появляется список городов (Рисунок 8).



Рисунок 7 – Поиск на панели инструментов

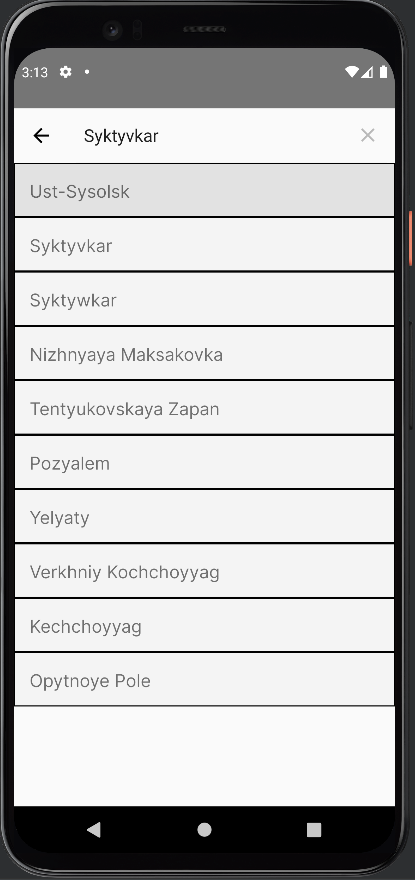


Рисунок 8 – Окно поиска

### Окно настроек

В окне настроек (Рисунок 9) можно настроить время обновления виджета и поменять единицы измерения

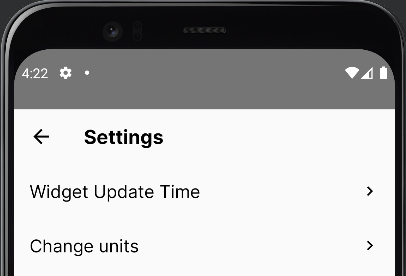


Рисунок 9 – Окно настроек

### Окно настроек единиц измерений

В окне настроек единиц измерения можно поменять три единицы измерения: температуру, скорость ветра, давление (Рисунок 10). При нажатии на одну из единиц измерения будет показано диалоговое окно с возможностью выбора единиц измерений (Рисунок 11).

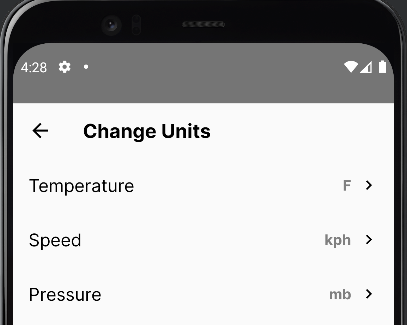


Рисунок 10 – Окно настроек единиц измерений

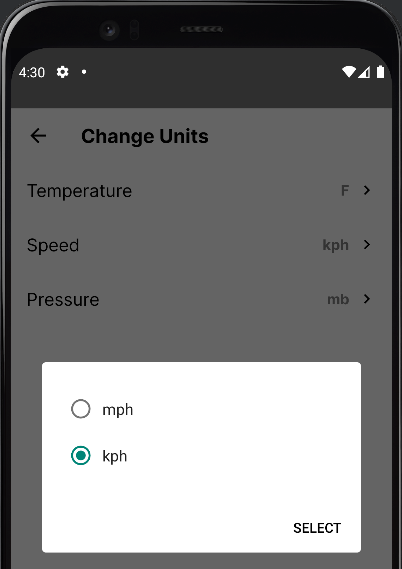


Рисунок 11 – Диалог выбора единицы измерения скорости

### Виджет для главного экрана

Виджет достаточно компактно вмещает в себя базовую информацию о погоде на текущий час (Рисунок 12).

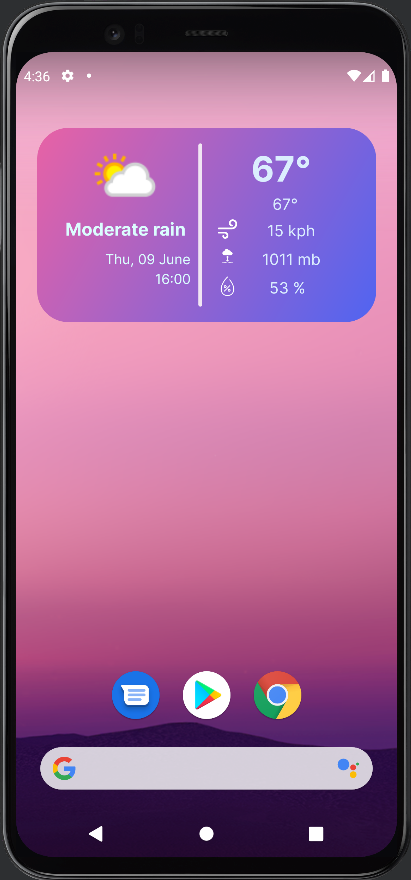


Рисунок 12 – Виджет на главном экране

## Реализация системы

Реализация системы представлена в «Приложении В».

# Экономическая часть

## Область применения программного продукта и его преимущества перед аналогичным программным продуктом

Система предназначена для просмотра информации о погоде в выбранном городе.

Достоинствами системы являются простота функционала и интерфейса.

## Трудоемкость разработки программного продукта, квалификация исполнителя и его оклад

Трудоемкость разработки можно определить в таблице 1. Строка «Всего» отображает общую трудоемкость разработки.

Таблица 1 – Трудоемкость разработки программного продукта

| **Наименование этапа** | **Условное обозначение** | **Трудоемкость выполнения этапа, час** |
| --- | --- | --- |
| Описание задания | То | 16 |
| Разработка UML - диаграмм | Тд | 10 |
| Разработка Базы данных | Тбд | 4 |
| Проектирование программы | Тпп | 234 |
| Отладка программы | Топ | 52 |
| Оценка качества программы | Тоц | 8 |
| Оформление документации | Тд | 28 |
| Всего | Тобщ | 352 |

Разработчики программного продукта представлены в таблице 2

Таблица 2 – Разработчики программного продукта

|  |  |  |  |
| --- | --- | --- | --- |
| **Исполнители** | **Оклад, руб.** | **Часовая тарифная ставка, руб/час** | **Количество сотрудников** |
| Разработчик -программист | 10 000 | 56,82 | 1 |
| Руководитель проекта | 15 000 | 85,23 | 1 |
| Инженер по внедрению | 12 000 | 68,18 | 1 |

Часовая тарифная ставка ЧТС, руб./час., определяется исходя из месячного оклада, количества рабочих дней в месяце и продолжительности рабочего по формуле:

(4)

где Ом – оклад исполнителя в месяц, руб. /мес.;

Д – количество рабочих дней в месяце (для расчета Д = 22 раб. дня);

Тс – продолжительность рабочего дня (для расчета Тс = 8час.)

Часовая тарифная ставка разработчик-программист:

ЧТС = 10000 / (8 \* 22) = 56,82 руб.

Часовая тарифная ставка руководителя проекта:

ЧТС = 15000 / (8 \* 22) = 85,23 руб.

Часовая тарифная ставка инженера по внедрению

ЧТС = 12000 / (8 \* 22) = 68,18 руб.

В таблице 3 показана стоимость технических средств для разработки системы.

Таблица 3 – Стоимость технических средств разработки

| **Наименование компонента** | **Цена, руб.** | **Количество, шт.** | **Стоимость, руб.** |
| --- | --- | --- | --- |
| Ноутбук Honor MagicBook 16 | 85000 | 1 | 85000 |
| Мышь беспроводная Logitech PRO X SUPERLIGHT | 900 | 1 | 900 |
| Итого: | | | 85900 |

В таблице 4 представлены затраты на расходные материалы.

Таблица 4 – Планируемые затраты на расходные материалы (Р)

| **Затраты** | **Стоимость** | **Количество** | **Сумма, руб.** |
| --- | --- | --- | --- |
| Интернет | 1200 руб./месяц | 2 месяцев | 2 400 |
| Электричество | 5,71 руб./КВт\*ч | 352 КВт\*ч | 2 009,92 |
| Итого: | | | 4 409,92 |

## Расчет затрат на разработку

Исходные данные, связанные с разработкой программного продукта приведены в таблице 5.

Таблица 5 – Исходные данные

| **Наименование** | **Обозначение** | **Значение** |
| --- | --- | --- |
| Оклад разработчика | Ор | 25000 руб. |
| Время разработки | Трп | 1 месяцев |
| Машинное время разработки | Тмч | 0,5 месяца |
| Коэффициент дополнительной заработной платы | Кд | 0,11 |
| Коэффициент страховых взносов | Кст | 0,54 |
| Количество единиц техники | Q | 1 шт. |
| Себестоимость содержания техники | См/ч | 10 руб./час |
| Коэффициент готовности техники | Кгт | 0,97 |
| Число рабочих дней в месяце | ЧРД | 22 дня |
| Продолжительность смены | Тст | 8 часов |
| Коэффициент сменности | Ксм | 1 |
| Коэффициент транспортных расходов | Кт | 0 |
| Коэффициент накладных расходов | Кнр | 0,46 |

Расчет полных затрат на разработку проектного решения (программного продукта) осуществляется по формуле:

Зрп = Зот + Зст + Зэвм + Зсп + Зрм + Знр, руб. (5)

Зрп = 27 750 + 14 985 + 853,6 + 85 900 + 4 409,92 + 12 765 = 146 664 руб.

где Зот – затраты на оплату труда разработчика (разработчиков), руб.;

Зст – страховые взносы по оплате труда во внебюджетные фонды, руб.;

Зэвм – затраты, связанные с содержанием вычислительной техники, руб.;

Зсп – затраты на технические средства, руб.;

Зрм – затраты на расходные материалы, необходимые при разработке программного продукта, руб.;

Знр – затраты по накладным расходам, приходящиеся на разработку программного продукта, руб.

### Затраты на оплату труда разработчиков (Зот), руб.

Размер фонда оплаты труда разработчиков (Зот) рассчитывается по формуле:

Зот = Ор \* Трп (1+Кд), руб (6)

где Ор – месячный оклад разработчика проектного решения, руб./мес.;

Трп – время разработки проектного решения разработчиком, мес.(час.) включает в себя машинное время работы над проектом (Тмрп)

Кд – коэффициент дополнительной заработной платы разработчика;

Зот = 25 000 \* 2 \* (1 + 0,11) = 27 750 руб

### Затраты по страховым вносам (Зст), руб.

Сумма страховых взносов определяется по формуле:

Зст = Кст \*Зот, руб (7)

где Кст – коэффициент страховых взносов для расчета отчислений во внебюджетные фонды.

Зст = 0,54 \* 27 750 = 14 985 руб

### Затраты по содержанию ЭВМ (Зэвм), руб

Затраты, связанные с эксплуатацией и содержанием ЭВМ, определяются по формуле:

Зэвм = Тмрп \* Кгт \* Q\* Cм/ч, руб. (8)

Зэвм = 88 \* 0,97 \* 1 \* 10 = 853,6 руб.

где Тмрп – машинное время на разработку проектного решения, час;

Кгт – коэффициент готовности техники;

Q – количество условных единиц, используемой техники;

Cм/ч – стоимость машино-часа, эксплуатации оборудования, руб. в час.

Так как машинное время может измеряться в месяцах, а себестоимость машино-часа за один час, то машинное время необходимо перевести в часы.

Перевод рабочего времени в часы осуществляется по формуле:

Тмрп = Тмч \* Чрд \* Тсм \* Ксм, час (9)

где Тмч – рабочее время в месяцах;

Чрд – число рабочих дней в месяце;

Тсм – продолжительность рабочей смены;

Ксм – количество рабочих смен.

Тмрп = 0,5 \* 22 \* 8 \* 1 = 88 час

### Затраты на расходные материалы (Зрм), руб.

Затраты на расходные материалы необходимые для разработки проектного решения определяются по формуле:

Зрм = , руб. (11)

затраты на расходные материалы, руб.;

Кт – коэффициент транспортных расходов

Зрм = 4 409,92 \* (1 + 0) = 4 409,92 руб.

### Затраты по накладным расходам (Знр)

Затраты по накладным расходам определяются по формуле:

Знр = Кнр \* Зот, руб. (12)

где Кнр – коэффициент накладных расходов, принимается для расчета по данным предприятия;

Зот – затраты по оплате труда, руб.

Знр = 0,46 \* 27 750 = 12 765 руб.

Таблица 6 – Результаты вычислений

| **Наименование** | **Обозначение** | **Значение** |
| --- | --- | --- |
| Оплата труда | Зот | 27 750 руб.; |
| Страховые взносы | Зст | 14 985 руб.; |
| Содержание ЭВМ | Зэвм | 853,6 руб.; |
| Затраты на специальные программы | Зсп | 85900 руб.; |
| Расходные материалы | Зрм | 4 409,92 руб.; |
| Накладные расходы | Знр | 12 765 руб.; |
| Итого затрат на разработку | Зрп | 146 664 руб.; |

Произведя вычисления было выявлено, что полные затраты на разработку составляют 146 664 рублей. Промежуточные результаты вычислений представлены в таблице 6.

## Расчет затрат на внедрение

Исходные данные, связанные с разработкой программного продукта приведены в таблице 7.

Таблица 7 – Исходные данные

| **Наименование** | **Обозначение** | **Значение** |
| --- | --- | --- |
| Оклад инженера по внедрению | Ор | 12 000 руб. |
| Время необходимое для внедрения | Трп | 1 месяца |
| Коэффициент дополнительной заработной платы | Кд | 0,11 |
| Коэффициент страховых взносов | Кст | 0,54 |
| Затраты на разработку | Зрп | 146 664 руб. |
| Себестоимость содержания техники | См/ч | 10 руб./час |
| Коэффициент готовности техники | Кгт | 0,97 |
| Число рабочих дней в месяце | ЧРД | 22 дня |
| Продолжительность смены | Тст | 8 часов |
| Коэффициент сменности | Ксм | 1 |
| Коэффициент накладных расходов | Кнр | 0,46 |

Затраты на внедрение являются для организации единовременными и формируют величину капиталовложений в данный проект.

Затраты на внедрение проектного решения (Квпр) рассчитываются по формуле:

Квпр = Зрп +Зот +Зст +Зэвм +Знр, руб. (13)

Квпр = 146 664 + 13 320 + 7 192,8 + 1707,2 + 6 127,2 = 175 011 руб.

где Зрп – затраты на приобретение программного обеспечения (включают стоимость разработанного ПП, а также других существующих ПП, необходимых для функционирования системы), руб.;

Зот – затраты на оплату туда работников, занятых внедрением проекта, руб.;

Зст – отчисления во внебюджетные фонды с заработной платы работников, занятых внедрением проекта, руб.;

Зэвм – затраты, связанные с эксплуатацией ЭВМ при внедрении проектного решения, руб.;

Знр – накладные расходы, руб.

### Затраты на приобретение программного обеспечение (Зпо)

Зрп = 146 664 руб.

### Затраты на оплату работников, занятых внедрением проекта (Зот)

Затраты на оплату труда работников, занятых внедрением разработанного проекта, включают в себя заработную плату сотрудников, участвующих непосредственно в установке и отладке системы.

Зотв= Окл. Разработчика\*Те\*(1+Кд)), руб. (14)

Те–трудоемкость внедрения, т.е. время необходимое для внедрения проекта, час., дни, мес.

Кд –коэффициент дополнительной заработной платы.

Зотв = 12 000\* 1 \* (1 + 0,11) = 13 320 руб.

### Затраты по страховым взносам (Зст)

Зст =13 320 \* 0,54 = 7 192,8 руб.

### Затраты по содержанию ЭВМ (Зэвм), руб

Зэвм = 176 \* 0,97 \* 10 \* 1 = 1 707,2 руб.

Тмрп = 1 \* 22 \* 8 \* 1 = 176 час

### Затраты по накладным расходам (Знр)

Затраты по накладным расходам определяются по формуле:

Знр = КнрЗотв, руб (15)

где Кнр – коэффициент накладных расходов

Зотв – затраты по оплате труда работников, занятых внедрением.

Знр = 0,46 \* 13 320 = 6 127,2 руб.

Таблица 8 – Результаты вычислений

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Обозначение** | **Значение** |
| Оплата труда | Зотв | 13 320 руб. |
| Страховые взносы | Зст | 7 192,8 руб. |
| Содержание ЭВМ | Зэвм | 1 707,2 руб. |
| Накладные расходы | Знр | 6 127,2 руб. |
| Итого затрат на внедрение | Зрпв | 175 011 руб. |

Произведя вычисления было выявлено, что полные затраты на внедрение составляют 175 011 рублей. Промежуточные результаты вычислений представлены в таблице 8.

## Расчет цены и прибыли

Исходные данные, связанные с разработкой программного продукта приведены в таблице 9.

Таблица 9 – Исходные данные

| **Наименование** | **Обозначение** | **Значение** |
| --- | --- | --- |
| Коэффициент рентабельности | Кр | 0,33 |
| Коэффициент налога на добавочную стоимость | Кндс | 0,2 |
| Ставка налога | Кнп | 0,2 |
| Стоимость затрат на разработку и внедрение | Квпр | 175 011 руб. |

Цена программного продукта, который разработан одной организаций по заказу другой и не предназначен для тиражирования, определяется по формуле:

Цпп = Квпр + Ппл + НДС, руб. (16)

Цпп = 175 011 + 57 753,6 + 46 552,9 = 279 318 руб.

где Ппл – планируемая прибыль рассчитывается по формуле:

Ппл = Квпр × Кр, руб. исходя из нормативной рентабельности (коэффициента)

Ппл = 175 011 \* 0,33 = 57 753,6 руб

НДС – налог на добавленную стоимость определяется исходя из Кндс = 0.2 по формуле

НДС = (Квпр + Ппл) × Кндс, руб. (17)

НДС = (175 011 + 57 753,6) \* 0,2 = 46 552,9 руб

Каждое предприятие с полученной прибыли перечисляет государству налог на прибыль. На сегодня ставка налога 20% от полученной прибыли, и определяется по формуле:

НП = Ппл × Кнп, руб. (18)

НП = 57 753,6 \* 0,2 = 11 550,7 руб

Чистая прибыль, остающаяся в распоряжении предприятия

ЧП = Ппл – НП, руб. (19)

ЧП = 57 753,6 – 11 550,7 = 46 202,9 руб

Поступление в бюджет складываются из налога на прибыль и НДС

ПБ = НП + НДС, руб. (20)

ПБ = 11 550,7 + 46 552,9 = 58 103,7 руб

Таблица 10 – Результаты вычислений

| **Наименование** | **Обозначение** | **Значение** |
| --- | --- | --- |
| Плановая прибыль | Ппл | 57 753,6 руб. |
| Налог на добавочную стоимость | НДС | 46 552,9 руб. |
| Цена программного продукта | Цпп | 279 318 руб. |
| Налог на прибыль | НП | 11 550,7 руб. |
| Чистая прибыль | ЧП | 46 202,9 руб. |
| Поступление в бюджет | ПБ | 58 103,7 руб. |

Произведя вычисления было выявлено, что чистая прибыль будет составлять 46 202,9 рублей. Промежуточные результаты вычислений представлены в таблице 10.

ВЫВОД: на основании произведенных расчетов, можно сделать вывод, что на разработку продукта будет затрачено 146 664 рублей, а с учетом внедрения 175 011 рублей. Цена готового продукта будет равна 279 318 рубля, что в свою очередь позволит получить чистую прибыль в размере от 46 202,9 рублей.

# Техника безопасности и охрана труда

## Анализ условий труда программиста, работающего ООО «Мониторинг Цен»

Рабочее место программиста – это его рабочий кабинет, либо часть помещения, в котором он проводит большую часть работы. Совершенно очевидно, что чем лучше приспособлено рабочее место для выполнения функций программиста, тем производительнее и эффективнее будет его трудовая деятельность.

Программист имеет собственное рабочее место, оборудованное ПЭВМ. Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также – расстоянию спинки до переднего края сиденья.

Рабочая поверхность стола сделана из дерева. На столе размещены ЭВМ, монитор и принтер. Площадь поверхности стола предполагает размещение монитора, устройств ввода/вывода и рабочей зоны с местом расположения оперативной документации.

Кабинет, в котором расположено рабочее место, представляет собой помещение площадью 30 м2, длиной 6 м, шириной 5 м и высотой 4 м.

Для создания благоприятных условий для зрительного восприятия кабинет оформлен в светло-бежевом цвете.

Для создания и поддержания независимо от наружных условий оптимальных значений температуры, влажности, чистоты воздуха, в холодное время используется водяное отопление, в теплое время года применяется кондиционирование воздуха.

Режим работы программиста – односменный с пятидневной рабочей неделей и нормированным рабочим днем. Продолжительность ежегодного отпуска – 28 календарных дней.

При идентификации вредных производственных факторов было установлено, что опасными могут являться:

* Шум и вибрация, источником которого является оргтехника;
* Электромагнитные поля и излучения от дисплея ПК;
* Статическое электричество, накапливаемое на клавиатуре, дисплее, корпусе системного блока;
* Электрический ток в электрических сетях;
* Запыленность рабочей зоны, так влажная уборка рабочей поверхности проводиться не ежедневно, а только по мере загрязнения;
* Статические перегрузки, так как работа программиста связана с неизменной статической позой, иногда неудобной позе (работа за ПК), что может привести к искривлению позвоночника, остеохондрозу, застою в органах малого таза;
* Перенапряжение анализаторов, в большей степени зрительного, что может привести к различным нарушениям зрения (снижение остроты, спазм и уменьшение запаса аккомодации), помимо этого повышение зрительной нагрузки может приводить к общему утомлению, возникновению головной боли, ухудшению самочувствия;
* Умственное перенапряжение, так как в процессе деятельности программист анализирует и перерабатывает огромное количество административных сетей, разрабатывает программные продукты, поддерживает технику в рабочем состоянии.

В целях выявления вредных и (или) опасных производственных факторов и осуществления мероприятий по приведению условий труда в соответствие государственными нормативными требованиями охраны трудаежегодно проводится аттестация рабочих мест по условиям труда - оценка условий труда на рабочих местах. Аттестации подлежат все имеющиеся в организации рабочие места.

При аттестации рабочих мест проводят оценку условий труда, гигиеническую оценку и оценку травмоопасности рабочих мест. При этом учитывают наличие средств коллективной защиты, обеспеченность работников средствами индивидуальной защиты и определяют эффективность этих средств.

Согласно санитарным правилам СП 2.2.3670-20 "Санитарно-эпидемиологические требования к условиям труда", дисплей должен располагаться на расстоянии 60-70 см, но не менее 50 см от глаз. Между боковыми поверхностями мониторов должно быть не менее 1,2 м. При использовании жидкокристаллических дисплеев на каждый компьютер должно приходиться не менее 4,5 м2 площади. На дисплей ПК не должен попадать прямой солнечный свет во избежание бликов и повышенной нагрузки на зрение.

Поэтому в кабинете, где работает программист, стоит только один компьютер, а комплектующая оргтехника расположена достаточно далеко непосредственно от самого специалиста. Это обусловлено тем, что все копировальные аппараты, принтеры и факс в периоды своей работы выделяют не только вредные токсичные газы, но и шумы и вибрацию. Данные факторы оказывают гораздо меньшее негативное влияние, нежели излучение от компьютера, но также являются вредными и могут нанести вред здоровью сотрудника.

Для снижения напряжения с глаз, а также для снятия мышечного напряжения во время рабочего дня проводится проветривание в течении 10 – 15 минут каждые 1,5 часа.

## Расчет искусственного освещения рабочего места программиста в ООО «Мониторинг Цен»

Через зрительный анализатор человек получает около 80% из общего объема информации. Качество поступающей информации во многом зависит от освещения: неудовлетворительное количественно или качественно, оно не только утомляет зрение, но и вызывает утомление организма в целом. Кроме того, нерациональное освещение может явиться причиной травматизма: плохо освещенные опасные зоны, слепящие источники света и блики от них, резкие тени ухудшают видимость настолько, что вызывают полную потерю ориентировки работающих, снижают производительность труда и увеличивают брак продукции. Поэтому необходимо достаточное искусственное освещение для рабочего места.

Условия работы для расчета показателя освещенности рабочего места программиста являются:

* Помещение размером 30 квадратных метров;
* Высота помещения 4 метра, длина 6 метра, ширина 5 метров;
* Лампы люминесцентные (дневного света) в количестве 7 штук, новые;
* Крупногабаритная мебель отсутствует.

Расчет общего равномерного искусственного освещения горизонтальной рабочей поверхности выполняется методом коэффициента использования светового потока. Световой поток (лм) одной лампы рассчитывается по следующей формуле:



где Ен - нормируемая минимальная допустимая освещенность по СП 1.3330.2016, для помещения 2000 лк;

S - площадь освещаемого помещения, м2;

z - коэффициент неравномерности освещения, z = 1,1;

K - коэффициент запаса, учитывающий снижение освещенности в процессе эксплуатации вследствие загрязнения и старения ламп и светильников, а также снижения отражающих свойств поверхностей помещения для кабинета будет равен 1,3;

Nc - число светильников в помещении;

- коэффициент затенения, обычно 1;

= i - коэффициент использования светового потока

Коэффициент использования светового потока, давший название методу расчета, определяется по индексу помещения i в зависимости от типа светильника и коэффициентов отражения света от потолка, стен и пола:



где А, В - длина и ширина помещения в плане, м;

Нс - высота подвеса светильников над рабочей поверхностью, м.

Согласно СП 52.13330.2016 величина полученного светового потока достаточно для данного помещения.

## Электробезопасность в ООО «Мониторинг Цен»

При поступлении на работу сотрудник должен пройти инструктаж по технике безопасности и электробезопасности. Работника знакомят с основными правилами по технике безопасности, предлагают внимательно прочитать действующие на предприятии инструкции, поясняя при этом отдельные правила и требования.

Инструктаж по технике безопасности при выполнении конкретной работы проводит руководитель соответствующего производственного участка, показывая инструктируемому правильные безопасные приемы работы.

Повторный инструктаж проводится для рабочих независимо от их квалификации, стажа и опыта работы, не реже 1 раза в 6 месяцев по программе инструктажа на рабочем месте.

При нарушении работающими правил и инструкций по технике безопасности, технологической и производственной дисциплины, а также в случаях изменения технологического процесса или вида работы проводят дополнительные инструктажи.

Все инструктажи оформляются записями в специальном журнале с указанием номеров или шифров инструкций. Журнал о проведении инструктажа хранится у руководителя подразделения.

Работник на рабочем месте должен соблюдать общие меры электробезопасности. При этом запрещается:

- Прикасаться к задней панели системного блока при включенном питании;

- Допускать попадание влаги на поверхность системного блока, монитора, рабочую поверхность клавиатуры, дисководов, принтеров и др. устройств;

- Производить самостоятельно вскрытие и ремонт оборудования;

- Пользоваться неисправными электроприборами и электропроводкой;

- Ремонтировать электроприборы самостоятельно;

- Подвешивать электропровода на гвоздях, металлических и деревянных предметах, перекручивать провод, закладывать провод и шнуры на водопроводные трубы и батареи отопления, вешать что-либо на провода, вытягивать за шнур вилку из розетки;

- Прикасаться одновременно к персональному компьютеру и к устройствам, имеющим соединение с землей (радиаторы отопления, водопроводные краны, трубы и т.п.);

- Пользоваться самодельными электронагревательными приборами и электроприборами с открытой спиралью;

- Наступать на переносимые электрические провода, лежащие на полу.

При работе с электроприборами и оргтехникой (персональные компьютеры, принтеры, сканеры, копировальные аппараты, факсы, бытовые электроприборы, приборы освещения):

- Автоматические выключатели и электрические предохранители должны быть всегда исправны;

- Изоляция электропроводки, электроприборов, выключателей, штепсельных розеток, ламповых патронов и светильников, а также шнуров, с помощью которых включаются в электросеть электроприборы, должны быть в исправном состоянии;

- Электроприборы необходимо хранить в сухом месте, избегать резких колебаний температуры, вибрации, сотрясений;

- Для подогрева воды пользоваться сертифицированными электроприборами с закрытой спиралью и устройством автоматического отключения, с применением несгораемых подставок.

## Пожарная безопасность в ООО «Мониторинг Цен»

Инструктаж по пожарной безопасности проводится по программе, разработанной инженером по охране труда ООО «Мониторинг Цен», с учетом требований стандартов, правил, норм и инструкций о мерах пожарной безопасности. Продолжительность инструктажа устанавливается в соответствии с утвержденной программой. Инструктаж по пожарной безопасности, как правило, проводится совместно с инструктажем по технике безопасности и в те же сроки.

Инструктаж по пожарной безопасности проходят все работники организации, независимо от их образования, стажа работы по данной профессии или должности, временные работники, командированные, обучающиеся и студенты, прибывшие на практику.

О проведении инструктажа по пожарной безопасности работник, проводивший инструктаж, делает запись в журнале учета проведения инструктажей по пожарной безопасности, с обязательной подписью инструктируемого и инструктирующего. При регистрации внепланового инструктажа по пожарной безопасности указывают причину его проведения.

Все работники организаций должны допускаться к работе только после прохождения противопожарного инструктажа, а при изменении специфики работы проходить дополнительное обучение предупреждению и тушению возможных пожаров в порядке, установленном руководителем.

На проведение первичного противопожарного инструктажа необходимо отводить не менее 1 ч. Инструктируемые должны ознакомиться:

- с действующими на объекте правилами пожарной безопасности и инструкциями;

- с производственными участками, наиболее опасными в пожарном отношении, где запрещается курить, применять открытый огонь;

- с возможными причинами возникновения пожара и мерами его предупреждения;

- с практическими действиями в случае возникновения пожара - вызов пожарной помощи, использование первичных средств пожаротушения, место расположения ближайшего телефона и ознакомление с правилами поведения в случае возникновения пожара, эвакуации людей и материальных ценностей.

При первичном инструктаже инструктирующий обязан рассказать о производственных установках с повышенной пожарной опасностью, мерах предотвращения пожаров и загораний, указать место курения, ознакомить вновь поступившего с имеющимися на объекте средствами пожаротушения, показать ближайший телефон и объяснить правила поведения в случае возникновения пожара.

Проведение противопожарного инструктажа в обязательном порядке должно сопровождаться практическим показом способов использования имеющихся на объекте средств пожаротушения (противогазы, респираторы, огнетушители и т.д.).

Соблюдение рассмотренных в данном разделе мероприятий по охране труда и технике безопасности в ООО «Мониторинг Цен» позволяет снизить утомляемость и травматизм, повысить производительность труда, обеспечить комфортные условия трудовой деятельности специалиста, работающего в должности программиста.

# ЗАКЛЮЧЕНИЕ

Реализованная в рамках дипломного проекта приложение прогноза погоды смогла решить все поставленные перед ней задачи. В приложении был реализован весь функционал. Приложение упростило получение прогноза погоды для пользователей мобильных устройств на базе операционной системы Android.

В приложении было успешно реализованы поиск городов, получение данных о прогнозе для выбранного города, настройка приложения и виджет для главного экрана.

После внедрения системы стало ясно, какие именно недостатками она обладает и что еще в ней необходимо реализовать. За счет всей этой полученной информации возможно дальнейшее улучшение системы путем добавления в нее нового функционала, такого как поиска города по карте, отображение в окне уведомлений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Стив Макконнелл. Совершенный код / Стив Макконнелл. : Питер, 2019. – 867 с.
2. Роберт Мартин. Идеальный программист. Как стать профессионалом разработки ПО / Роберт Мартин. : Питер, 2018. – 214 с.
3. Koin – a smart injection library [Электронный ресурс] Режим доступа : https://insert-koin.io/, свободный. – Загл. с экрана.
4. Роберт Мартин. Чистый код. Создание, анализ и рефакторинг / Роберт Мартин. : ДМК Пресс, 2016. – 431 c.
5. Энтони Молинаро. SQL. Сборник рецептов / Энтони Молинаро. : Символ-Плюс, 2009. – 672 с.
6. Алан Бьюли. Изучаем SQL / Алан Бьюли. : Питер, 2007. – 308 с.
7. Kotlin, обрабатываем исключения в корутинах правильно [Электронный ресурс] Режим доступа : <https://habr.com/ru/post/664444/>, свободный. – Загл. с экрана.
8. Kotlin. Basic syntax [Электронный ресурс] Режим доступа : https://kotlinlang.org/docs/basic-syntax.html, свободный. – Загл. с экрана.
9. Metanit, Руководство по языку программирования Kotlin [Электронный ресурс] Режим доступа : https://metanit.com/kotlin/tutorial/, свободный. – Загл. с экрана.
10. Metanit, Руководство по SQLite [Электронный ресурс] Режим доступа : https://metanit.com/sql/sqlite/, свободный. – Загл. с экрана.
11. ГОСТ 19.106-78. Требования к программным документам, выпаленным печатным способом. – Введ. 1980-01-01. – Москва : Изд-во стандартов, 1978. – 10 с.
12. ГОСТ 19.104-78. Основные надписи. – Введ. 1980-01-01. – Москва : Изд-во стандартов, 1978. – 6 с.
13. ГОСТ 19.105-78. Общие требования к программным документам. – Введ. 1980-01-01. – Москва: Изд-во стандартов, 1978. – 2 с.
14. ГОСТ 19.201-78. Техническое задание. Требования к содержанию и оформлению. – Введ. 1980-01-01. – Москва : Изд-во стандартов, 1978. – 3 с.
15. ГОСТ 19.101-77. Виды программ и программных документов. – Введ. 1980-01-01. – Москва: Изд-во стандартов, 1977. – 3 с.
16. ГОСТ 19.102-77. Стадии разработки. – Введ. 1980-01-01. – Москва : Изд-во стандартов, 1977. – 2 с.

# Приложение А

(справочное)

В таблице А.1 предоставлены аспекты программного обеспечения.

Таблица А.1 – Характеристика проблемно-ориентированного программного обеспечения

| **Показатель** | **Значение** |
| --- | --- |
| Среда разработки | Android Studio |
| Технология программирования | ООП |
| Язык программирования | Kotlin |
| Модель описания данных | Реляционный |
| СУБД | SQLite |
| Количество таблиц в базе данных | 2 |
| Количество полей в базе данных | 20 |
| Тип данных в базе данных | Varchar, integer, datetime |
| Тип связей между таблицами в базе данных | 1 : М |

# 

# Приложение В

(справочное)

**Application.kt**

class Application : Application() {  
  
 companion object {  
 private lateinit var weakReferenceContext: WeakReference<Context>  
  
 val context: Context  
 get() = weakReferenceContext.get()!!  
 }  
  
 override fun onCreate() {  
 super.onCreate()  
 weakReferenceContext = WeakReference(this)  
  
 val module = module {  
  
 single<PropertyTracker> { PropertyTrackerImpl }  
 single<SettingsRepository> { SettingsRepositoryImpl(get()) }  
 single<WeatherApi> { createRetrofit(WeatherApiService::class.java) }  
  
 viewModel { MainViewModel(get(), get(), get()) }  
 viewModel { SearchViewModel(get(), get()) }  
 viewModel { SettingsViewModel() }  
 viewModel { UnitsViewModel(get()) }  
  
 }  
  
 startKoin {  
 androidContext(this@Application)  
 modules(module)  
 }  
 }  
  
 private fun <T> createRetrofit(serviceClass: Class<T>) : T {  
 return Retrofit.Builder()  
 .baseUrl("https://api.weatherapi.com/v1/")  
 .addConverterFactory(GsonConverterFactory.create())  
 .client(createOkHttpInterceptor())  
 .build()  
 .create(serviceClass)  
 }  
  
 private fun createOkHttpInterceptor() : OkHttpClient {  
 return OkHttpClient.Builder()  
 .addInterceptor { chain ->  
 val url = chain.request().url().newBuilder()  
 .addQueryParameter("key", "0a6f744e5dde4449a73141307212111")  
 .build()  
  
 chain.proceed(chain.request().newBuilder().url(url).build())  
 }.build()  
 }  
}

**MainFragment.kt**

class MainFragment : Fragment() {  
  
 private val viewModel: MainViewModel by viewModel()  
 private lateinit var binding: FragmentMainBinding  
  
 private lateinit var hoursAdapter: HoursWeatherAdapter  
 private lateinit var daysAdapter: DaysWeatherAdapter  
  
 override fun onCreateView(inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?): View {  
  
 binding = FragmentMainBinding.inflate(inflater, container, false)  
  
 activity?.actionBar?.hide()  
  
 initAdapters()  
  
 viewModel.liveDataCityName.observe(viewLifecycleOwner) { cityName ->  
 binding.toolbarCityName.text = cityName  
 }  
 viewModel.liveDataSelectedHour.observe(viewLifecycleOwner) { hour ->  
 setCurrentWeather(hour)  
 }  
 viewModel.liveDataSelectedDay.observe(viewLifecycleOwner) { day ->  
 hoursAdapter.setListModel(day.hours)  
 }  
 viewModel.liveDataDays.observe(viewLifecycleOwner) { days ->  
 daysAdapter.setListModel(days)  
 }  
  
 binding.toolbarSearchButton.setOnClickListener {  
 NavHostFragment.findNavController(this)  
 .navigate(R.id.action\_fragment\_main\_to\_fragment\_search)  
 }  
 binding.toolbarSettingsButton.setOnClickListener {  
 NavHostFragment.findNavController(this)  
 .navigate(R.id.action\_fragment\_main\_to\_fragment\_settings)  
 }  
  
 return binding.root  
 }  
  
 private fun initAdapters() {  
 hoursAdapter = HoursWeatherAdapter { selectedHour ->  
 viewModel.changeSelectedHour(selectedHour)  
 }  
 daysAdapter = DaysWeatherAdapter { selectedDay ->  
 viewModel.changeSelectedDay(selectedDay)  
 }  
  
 binding.hourlyForecastRecyclerView.adapter = hoursAdapter  
 binding.dailyForecastRecyclerView.adapter = daysAdapter  
 }  
  
 @SuppressLint("SetTextI18n")  
 private fun setCurrentWeather(model: HourWeatherModel) {  
 binding.currentWeatherTemperatureTextView.text = model.temperature.asTemperature()  
 binding.currentWeatherFeelsLikeTextView.text =  
 "Feels like ${model.feelLikeTemperature.asTemperature()}"  
 binding.currentWeatherImageView.loadImage(  
 model.weatherIconUrl.replace("64x64", "128x128"),  
 model.weatherIconUrl)  
 binding.currentWeatherTypeTextView.text = model.weatherType  
 binding.currentWeatherDayTextView.text =  
 viewModel.liveDataSelectedDay.value!!.date.format(DateTimeFormatter.ofPattern("EE, dd MMMM"))  
 binding.currentWeatherTimeTextView.text = model.time.toString()  
 binding.currentWeatherWindTextView.text =  
 "${model.windSpeed} ${viewModel.units.speedUnit.asString()}"  
 binding.currentWeatherPressureTextView.text =  
 "${model.pressure} ${viewModel.units.pressureUnit.asString()}"  
 binding.currentWeatherHumidityTextView.text = "${model.humidity}%"  
 }  
  
 override fun onDestroy() {  
 viewModel.unregisterTrackingProperties()  
 super.onDestroy()  
 }  
}

**SearchFragment.kt**

class SearchFragment : Fragment() {  
  
 private val viewModel: SearchViewModel by viewModel()  
 private lateinit var binding: FragmentSearchBinding  
  
 private lateinit var searchAdapter: SearchResultAdapter  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View {  
 binding = FragmentSearchBinding.inflate(inflater, container, false)  
  
 searchAdapter = SearchResultAdapter {  
 viewModel.setCurrentCity(it)  
 findNavController().popBackStack()  
 }  
 binding.searchResultRecyclerView.adapter = searchAdapter  
  
 viewModel.liveDataSearchResult.observe(viewLifecycleOwner) { cityList ->  
 searchAdapter.setCityList(cityList)  
 }  
  
 binding.backButton.setOnClickListener { findNavController().popBackStack() }  
 binding.searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {  
  
 override fun onQueryTextChange(newText: String?): Boolean { return false }  
  
 override fun onQueryTextSubmit(cityName: String?): Boolean {  
 if (cityName == null || cityName.length <= 3)  
 return false  
  
 viewModel.searchCity(cityName)  
 return true  
 }  
 })  
  
 return binding.root  
 }  
  
}

**SettingsFragment.kt**

class SettingsFragment : Fragment() {  
  
 private val viewModel: SettingsViewModel by viewModel()  
 private lateinit var binding: FragmentSettingsBinding  
  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View {  
 binding = FragmentSettingsBinding.inflate(inflater, container, false)  
  
 binding.backButton.setOnClickListener {  
 findNavController().popBackStack()  
 }  
 binding.changeUnitsContainer.setOnClickListener {  
 NavHostFragment.findNavController(this)  
 .navigate(R.id.action\_fragment\_settings\_to\_fragment\_settings\_units)  
 }  
 binding.widgetUpdateTimeContainer.setOnClickListener {   
 NavHostFragment.findNavController(this)  
 .navigate(R.id.action\_fragment\_settings\_to\_fragment\_time\_settings)  
 }  
  
 return binding.root  
 }  
  
}

**UnitsFragment.kt**

class UnitsFragment : Fragment() {  
  
 private val viewModel: UnitsViewModel by viewModel()  
 private lateinit var binding: FragmentSettingUnitsBinding  
  
 @SuppressLint("CheckResult")  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View {  
  
 binding = FragmentSettingUnitsBinding.inflate(inflater, container, false)  
  
 viewModel.liveDataTemperature.observe(viewLifecycleOwner) { temperatureType ->  
 binding.temperatureUnitTextView.text = temperatureType.asString()  
 }  
 viewModel.liveDataSpeed.observe(viewLifecycleOwner) { speedType ->  
 binding.speedUnitTextView.text = speedType.asString()  
 }  
 viewModel.liveDataPressure.observe(viewLifecycleOwner) { pressureType ->  
 binding.pressureUnitTextView.text = pressureType.asString()  
 }  
  
 binding.backButton.setOnClickListener {  
 viewModel.saveChanges()  
 findNavController().popBackStack()  
 }  
 binding.temperatureUnitContainer.setOnClickListener {  
 val temperatureTypes = TemperatureType.values()  
  
 showSelectUnitDialog(temperatureTypes.map { it.asString() },  
 temperatureTypes.indexOf(viewModel.liveDataTemperature.value)) {  
 viewModel.setTemperatureType(it.asTemperatureType())  
 }  
 }  
 binding.speedUnitContainer.setOnClickListener {  
 val speedTypes = SpeedType.values()  
  
 showSelectUnitDialog(speedTypes.map { it.asString() },  
 speedTypes.indexOf(viewModel.liveDataSpeed.value)) {  
 viewModel.setSpeedType(it.asSpeedType())  
 }  
 }  
 binding.pressureUnitContainer.setOnClickListener {  
 val pressureTypes = PressureType.values()  
  
 showSelectUnitDialog(pressureTypes.map { it.asString() },  
 pressureTypes.indexOf(viewModel.liveDataPressure.value)) {  
 viewModel.setPressureType(it.asPressureType())  
 }  
 }  
  
 return binding.root  
 }  
  
 @SuppressLint("CheckResult")  
 private fun showSelectUnitDialog(units: List<String>, initValue: Int, onSelected: (String) -> Unit) {  
 val dialog = MaterialDialog(requireContext()).show {  
  
 listItemsSingleChoice(items = units,  
 initialSelection = initValue,  
 waitForPositiveButton = true,  
 selection = object : SingleChoiceListener {  
 override fun invoke(dialog: MaterialDialog, index: Int, text: CharSequence) {  
 onSelected.invoke(text.toString())  
 }  
 })  
 positiveButton(text = "Select")  
 }  
 // HACK: Чтобы были видны кнопки  
 dialog.view.buttonsLayout?.actionButtons?.forEach{  
 it.updateTextColor(resources.getColor(R.color.black, null))  
 }  
  
 }  
}

**CurrentDayForecastWidget.kt**

class CurrentDayForecastWidget : AppWidgetProvider() {  
  
 override fun onUpdate(  
 context: Context,  
 appWidgetManager: AppWidgetManager,  
 appWidgetIds: IntArray  
 ) {  
 val koin = KoinJavaComponent.getKoin()  
  
 val weatherApi = koin.get<WeatherApi>()  
 val settingsRepository = koin.get<SettingsRepository>()  
  
 runBlocking {  
 launch {  
 val cityName = settingsRepository.currentCity  
 val response = weatherApi.getWeatherInfo(cityName, -1)  
  
 if (!response.isSuccessful)  
 return@launch  
  
 val currentDayModel = response.body()!!  
 .forecast!!  
 .forecastday[0]  
 .toDayModel(settingsRepository.units)  
  
 for (appWidgetId in appWidgetIds) {  
 updateAppWidget(context, appWidgetManager, appWidgetId, currentDayModel)  
 }  
 }  
 }  
 }  
}  
  
internal fun updateAppWidget(  
 context: Context,  
 appWidgetManager: AppWidgetManager,  
 appWidgetId: Int,  
 model: DayWeatherModel  
) {  
 val views = RemoteViews(context.packageName, R.layout.widget\_current\_day\_forecast)  
 val currentHour = model.hours[LocalTime.now().hour]  
  
 // HACK: закончить уже хочется  
 runBlocking {  
 val result = withContext(Dispatchers.IO) {  
 Picasso.get().load(currentHour.weatherIconUrl).get()  
 }  
 views.setImageViewBitmap(R.id.current\_weather\_image\_view, result)  
 }  
  
 views.setTextViewText(R.id.current\_weather\_type\_text\_view, model.weatherType)  
 views.setTextViewText(R.id.current\_weather\_day\_text\_view, model.date.format(DateTimeFormatter.ofPattern("EE, dd MMMM")))  
 views.setTextViewText(R.id.current\_weather\_time\_text\_view, currentHour.time.toString())  
 views.setTextViewText(R.id.current\_weather\_temperature\_text\_view, currentHour.temperature.asTemperature())  
 views.setTextViewText(R.id.current\_weather\_feels\_like\_text\_view, currentHour.feelLikeTemperature.asTemperature())  
 views.setTextViewText(R.id.current\_weather\_wind\_text\_view, "${currentHour.windSpeed} ${model.units.speedUnit.asString()}")  
 views.setTextViewText(R.id.current\_weather\_pressure\_text\_view, "${currentHour.pressure} ${model.units.pressureUnit.asString()}")  
 views.setTextViewText(R.id.current\_weather\_humidity\_text\_view, "${currentHour.humidity} %")  
 appWidgetManager.updateAppWidget(appWidgetId, views)  
}

**DaysWeatherAdapter.kt**

class DaysWeatherAdapter(private val onDayItemClick: ((DayWeatherModel) -> Unit)? = null)  
 : RecyclerView.Adapter<DaysWeatherAdapter.DayWeatherViewHolder>() {  
  
 class DayWeatherViewHolder(private val itemBinding: ItemDaysForecastBinding,  
 private val onDayItemClick: ((DayWeatherModel) -> Unit)? = null)  
 : RecyclerView.ViewHolder(itemBinding.root) {  
  
 private lateinit var model: DayWeatherModel  
  
 init {  
 itemBinding.root.setOnClickListener { onDayItemClick?.invoke(model) }  
 }  
  
 @SuppressLint("SetTextI18n")  
 fun setModel(model: DayWeatherModel) {  
 this.model = model  
 itemBinding.dayTextView.text =  
 model.date.format(DateTimeFormatter.ofPattern("EE"))  
 itemBinding.shortMonthTextView.text =  
 model.date.format(DateTimeFormatter.ofPattern("dd.MM"))  
 itemBinding.temperatureTextView.text = model.averageTemperature.asTemperature()  
 itemBinding.weatherImageView.loadImage(model.weatherIconUrl)  
 itemBinding.specialInfoTextView.text =  
 "${model.maxWindSpeed} ${model.units.speedUnit.asString()}"  
 }  
 }  
  
 private var daysWeatherList: List<DayWeatherModel>? = null  
  
 fun setListModel(daysWeatherList: List<DayWeatherModel>) {  
 this.daysWeatherList = daysWeatherList  
 notifyDataSetChanged()  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DayWeatherViewHolder {  
 val inflater = LayoutInflater.from(parent.context)  
 val viewBinding = ItemDaysForecastBinding.inflate(inflater, parent, false)  
  
 return DayWeatherViewHolder(viewBinding, onDayItemClick)  
 }  
  
 override fun onBindViewHolder(holder: DayWeatherViewHolder, position: Int) {  
 holder.setModel(daysWeatherList!![position])  
 }  
  
 override fun getItemCount(): Int {  
 return daysWeatherList?.size ?: 0;  
 }  
}

**HoursWeatherAdapter.kt**

class HoursWeatherAdapter(private val onHourItemClick: ((HourWeatherModel) -> Unit)? = null)  
 : RecyclerView.Adapter<HoursWeatherAdapter.HourWeatherViewHolder>() {  
  
 class HourWeatherViewHolder(private val itemBinding: ItemHoursForecastBinding,  
 private val onHourItemClick: ((HourWeatherModel) -> Unit)? = null)  
 : RecyclerView.ViewHolder(itemBinding.root) {  
  
 private lateinit var model: HourWeatherModel  
  
 init {  
 itemBinding.root.setOnClickListener { onHourItemClick?.invoke(model) }  
 }  
  
 @SuppressLint("SetTextI18n")  
 fun setModel(model: HourWeatherModel) {  
 this.model = model  
 itemBinding.hourTextView.text =  
 model.time.format(DateTimeFormatter.ofPattern("HH:mm"))  
 itemBinding.temperatureTextView.text = model.temperature.asTemperature()  
 itemBinding.weatherImageView.loadImage(model.weatherIconUrl)  
 }  
 }  
  
 private var hoursWeatherList: List<HourWeatherModel>? = null  
  
 fun setListModel(hoursWeatherList: List<HourWeatherModel>) {  
 this.hoursWeatherList = hoursWeatherList  
 notifyDataSetChanged()  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): HourWeatherViewHolder {  
 val inflater = LayoutInflater.from(parent.context)  
 val viewBinding = ItemHoursForecastBinding.inflate(inflater, parent, false)  
  
 return HourWeatherViewHolder(viewBinding, onHourItemClick)  
 }  
  
 override fun onBindViewHolder(holder: HourWeatherViewHolder, position: Int) {  
 holder.setModel(hoursWeatherList!![position])  
 }  
  
 override fun getItemCount(): Int {  
 return hoursWeatherList?.size ?: 0  
 }  
}

**SearchResultAdapter.kt**

class SearchResultAdapter(private val onCityClick: ((String) -> Unit)? = null)  
 : RecyclerView.Adapter<SearchResultAdapter.SingleSearchResultViewHolder>() {  
  
 class SingleSearchResultViewHolder(private val itemBinding: ItemSearchResultBinding,  
 private val onCityClick: ((String) -> Unit)? = null)  
 : RecyclerView.ViewHolder(itemBinding.root) {  
  
 init {  
 itemBinding.root.setOnClickListener {  
 onCityClick?.invoke(itemBinding.cityNameTextView.text.toString())  
 }  
 }  
  
 fun setCityName(cityName: String) {  
 itemBinding.cityNameTextView.text = cityName  
 }  
 }  
  
 private var cityList: List<String>? = null  
  
 fun setCityList(cityList: List<String>) {  
 this.cityList = cityList  
 notifyDataSetChanged()  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SingleSearchResultViewHolder {  
 val inflater = LayoutInflater.from(parent.context)  
 val viewBinding = ItemSearchResultBinding.inflate(inflater, parent, false)  
  
 return SingleSearchResultViewHolder(viewBinding, onCityClick)  
 }  
  
 override fun onBindViewHolder(holder: SingleSearchResultViewHolder, position: Int) {  
 holder.setCityName(cityList!![position])  
 }  
  
 override fun getItemCount(): Int {  
 return cityList?.size ?: 0;  
 }  
}

**MainViewModel.kt**

class MainViewModel(private val weatherApi: WeatherApi,  
 private val settingsRepository: SettingsRepository,  
 private val propertyTracker: PropertyTracker)  
 : ViewModel(), PropertyTracker.PropertyChangedListener {  
  
 private val \_\_locale: Location? = null  
  
 private var \_units: UnitsSettings  
 val units: UnitsSettings  
 get() = \_units  
  
 private val \_liveDataCityName = MutableLiveData<String>()  
 val liveDataCityName: LiveData<String>  
 get() = \_liveDataCityName  
  
 private val \_liveDataSelectedHour = MutableLiveData<HourWeatherModel>()  
 val liveDataSelectedHour: LiveData<HourWeatherModel>  
 get() = \_liveDataSelectedHour  
  
 private val \_liveDataSelectedDay = MutableLiveData<DayWeatherModel>()  
 val liveDataSelectedDay: LiveData<DayWeatherModel>  
 get() = \_liveDataSelectedDay  
  
 private val \_liveDataDays = MutableLiveData<List<DayWeatherModel>>()  
 val liveDataDays: LiveData<List<DayWeatherModel>>  
 get() = \_liveDataDays  
  
 init {  
 viewModelScope.launch { loadForecast() }  
 \_units = settingsRepository.units  
 registerTrackingProperties()  
 }  
  
 private suspend fun loadForecast() {  
 val cityName = settingsRepository.currentCity  
 if (cityName.isEmpty())  
 return  
  
 loadForecastIn(cityName)  
 }  
  
 private suspend fun loadForecastIn(cityName: String) {  
 val response = weatherApi.getWeatherInfo(cityName, 3)  
 if (response.isSuccessful)  
 setForecast(response.body()!!)  
 }  
  
 // Отрефакторить  
 private fun setForecast(forecastDto: DailyForecastDto) {  
 val days = forecastDto.forecast!!.forecastday.map { it.toDayModel(units) }  
 val defaultHours = days[0].hours  
 // Исправить на дату  
 val timeHour = LocalTime.now().hour  
 val current = defaultHours.first { it.time.hour == timeHour }  
  
 \_liveDataCityName.value = forecastDto.location!!.name!!  
 \_liveDataDays.value = days  
 \_liveDataSelectedDay.value = days[0]  
// \_liveDataHoursOfCurrentDay.value = defaultHours  
 \_liveDataSelectedHour.value = current  
 }  
  
 fun changeSelectedDay(model: DayWeatherModel) {  
 \_liveDataSelectedDay.value = model  
 // Отрефакторить  
 \_liveDataSelectedHour.value = model.hours.first { hour ->  
 (\_\_locale ?: LocalTime.now().hour) == hour.time.hour  
 }  
 }  
  
 fun changeSelectedHour(model: HourWeatherModel) {  
 \_liveDataSelectedHour.value = model  
 }  
  
  
  
 /\* Trackers \*/  
 fun registerTrackingProperties() {  
 propertyTracker.addListenerTo(SettingsRepositoryImpl.CURRENT\_CITY\_PROPERTY, this)  
 propertyTracker.addListenerTo(SettingsRepositoryImpl.UNITS\_PROPERTY, this)  
 }  
 fun unregisterTrackingProperties() {  
 propertyTracker.removeListenerFrom(SettingsRepositoryImpl.CURRENT\_CITY\_PROPERTY, this)  
 propertyTracker.removeListenerFrom(SettingsRepositoryImpl.UNITS\_PROPERTY, this)  
 }  
 override fun propertyChanged(propertyName: String) {  
 if (propertyName == SettingsRepositoryImpl.CURRENT\_CITY\_PROPERTY)  
 viewModelScope.launch { loadForecast() }  
  
 if (propertyName == SettingsRepositoryImpl.UNITS\_PROPERTY) {  
 \_units = settingsRepository.units  
 viewModelScope.launch { loadForecast() }  
 }  
 }  
  
}

**SearchViewModel.kt**

class SearchViewModel(private val weatherApi: WeatherApi,  
 private val settingsRepository: SettingsRepository) : ViewModel() {  
  
 private val \_liveDataSearchResult = MutableLiveData<List<String>>()  
 val liveDataSearchResult: LiveData<List<String>>  
 get() = \_liveDataSearchResult  
  
 fun searchCity(cityName: String) {  
 viewModelScope.launch {  
 val responseSearchCity = weatherApi.searchCity(cityName)  
 if (responseSearchCity.isSuccessful)  
 \_liveDataSearchResult.value = responseSearchCity.body()!!.map { it.name!! }  
 }  
 }  
  
 fun setCurrentCity(cityName: String) {  
 settingsRepository.currentCity = cityName  
 }  
  
}

**SettingsViewModel.kt**

class SettingsViewModel : ViewModel()

**UnitsViewModel.kt**

class UnitsViewModel(private val settingsRepository: SettingsRepository) : ViewModel() {  
  
 private var localUnits: UnitsSettings  
  
 private val \_liveDataTemperature = MutableLiveData<TemperatureType>()  
 val liveDataTemperature: LiveData<TemperatureType>  
 get() = \_liveDataTemperature  
  
 private val \_liveDataSpeed = MutableLiveData<SpeedType>()  
 val liveDataSpeed: LiveData<SpeedType>  
 get() = \_liveDataSpeed  
  
 private val \_liveDataPressure = MutableLiveData<PressureType>()  
 val liveDataPressure: LiveData<PressureType>  
 get() = \_liveDataPressure  
  
 init {  
 val units = settingsRepository.units  
 localUnits = UnitsSettings(units.temperatureUnit,  
 units.speedUnit,  
 units.pressureUnit)  
  
 \_liveDataTemperature.value = units.temperatureUnit  
 \_liveDataSpeed.value = units.speedUnit  
 \_liveDataPressure.value = units.pressureUnit  
 }  
  
 // HACK  
 private var hasChanges = false  
 fun setTemperatureType(temperatureType: TemperatureType) {  
 if (liveDataTemperature.value == temperatureType)  
 return  
  
 hasChanges = true  
 localUnits = UnitsSettings(temperatureType, localUnits.speedUnit, localUnits.pressureUnit)  
 \_liveDataTemperature.value = temperatureType  
 }  
 fun setSpeedType(speedType: SpeedType) {  
 if (liveDataSpeed.value == speedType)  
 return  
  
 hasChanges = true  
 localUnits = UnitsSettings(localUnits.temperatureUnit, speedType, localUnits.pressureUnit)  
 \_liveDataSpeed.value = speedType  
 }  
 fun setPressureType(pressureType: PressureType) {  
 if (liveDataPressure.value == pressureType)  
 return  
  
 hasChanges = true  
 localUnits = UnitsSettings(localUnits.temperatureUnit, localUnits.speedUnit, pressureType)  
 \_liveDataPressure.value = pressureType  
 }  
  
 fun saveChanges() {  
 if (hasChanges)  
 settingsRepository.units = localUnits  
 }  
}

**PropertyTrackerImpl.kt**

object PropertyTrackerImpl : PropertyTracker {  
  
 private val listenersMap:  
 MutableMap<String, MutableList<PropertyTracker.PropertyChangedListener>> = mutableMapOf()  
  
 override fun addListenerTo(propertyName: String,  
 listener: PropertyTracker.PropertyChangedListener) {  
 if (!listenersMap.containsKey(propertyName))  
 listenersMap[propertyName] = arrayListOf()  
  
 listenersMap[propertyName]!!.add(listener)  
 }  
  
  
 override fun removeListenerFrom(propertyName: String,  
 listener: PropertyTracker.PropertyChangedListener  
 ) {  
 if (listenersMap.containsKey(propertyName))  
 if (listenersMap[propertyName]!!.count() == 1)  
 listenersMap.remove(propertyName)  
 else  
 listenersMap[propertyName]!!.remove(listener)  
 }  
  
 override fun raisePropertyChanged(propertyName: String) {  
 if (listenersMap[propertyName] == null)  
 return  
  
 for (listeners in listenersMap[propertyName]!!)  
 listeners.propertyChanged(propertyName)  
 }  
}

**SettingsRepositoryImpl.kt**

class SettingsRepositoryImpl(private val propertyTracker: PropertyTracker) : SettingsRepository {  
  
 companion object {  
 private const val PREFERENCE\_NAME = "APP\_PREFERENCE"  
 private const val CITY\_KEY = "CITY\_KEY"  
 private const val UNITS\_KEY = "UNITS\_KEY"  
  
 const val CURRENT\_CITY\_PROPERTY = "SettingsRepositoryImpl.currentCity"  
 const val UNITS\_PROPERTY = "SettingsRepositoryImpl.units"  
 }  
  
  
 private val sharedPreferences  
 = Application.context.getSharedPreferences(PREFERENCE\_NAME, Context.MODE\_PRIVATE)  
  
 override var currentCity: String  
 get() {  
 return sharedPreferences.getString(CITY\_KEY, null) ?: "Saint-Petersburg"  
 }  
 set(value) {  
 sharedPreferences.edit().putString(CITY\_KEY, value).commit()  
 propertyTracker.raisePropertyChanged(CURRENT\_CITY\_PROPERTY)  
 }  
  
 override var units: UnitsSettings  
 get() {  
 val unitsJson = sharedPreferences.getString(UNITS\_KEY, null)  
 if (unitsJson == null) {  
 units = UnitsSettings.DEFAULT  
 return UnitsSettings.DEFAULT  
 }  
  
 return Utils.gson.fromJson(unitsJson, UnitsSettings::class.java)  
 }  
 set(value) {  
 sharedPreferences.edit().putString(UNITS\_KEY, Utils.gson.toJson(value)).commit()  
 propertyTracker.raisePropertyChanged(UNITS\_PROPERTY)  
 }  
}

**WeatherApiService.kt**

interface WeatherApiService : WeatherApi {  
  
 @GET("forecast.json")  
 override suspend fun getWeatherInfo(@Query("q") cityName: String,  
 @Query("days") countDays: Int): Response<DailyForecastDto>  
  
 @GET("search.json")  
 override suspend fun searchCity(@Query("q") cityName: String): Response<List<Location>>  
}