COMP 4960: Software Engineering
E. Banach, S. Sollauer, G. Powell

# Software Design Document

## Weather Prediction & Clothing Recommendation Application

Revision B
11 April 2023

# Table of Contents

**1 Revision History** **3**
**2 Introduction** **3**
2.1 Document Purpose 3
2.2 Product Overview 3
2.3 Product Functionality 4
2.3.1 Feature I: Weather Overview Display 4
2.3.2 Feature II: Forecast Detail Display 4
2.3.3 Feature III: Clothing Preparation Display 4
2.4 Acronyms and Abbreviations 5
**3 System Architecture** **6**
3.1 Overall Architecture 6
3.2 Components Mapping 7
3.1.1 Functional Requirements 7
3.2.2 Non-Functional Requirements 10
3.3 Technology Stack Selection 12
**4 System Design** **13**
4.1 UI 13
4.2 Class Diagram 14
4.3 Sequence Diagram 15
4.4 Database 16
**5 Test Plan** **16**
5.1 Function #1 - getLocation(location) 16
5.2 Function #2 - getWeather(location) 16
5.3 Function #3 - generateRecommended(weather_data) 16
5.4 Function #4 - updateRecommended(clothing_list) 17
5.5 Function #5 - updateClothingChoices(clothing_list) 17
5.6 Function #6 - updateClothingTable(clothing_choices) 17
5.7 Function #7 - generateRecommended(weather_data, clothing_choices) 18

# 1 Revision History

| Date | Revision | Description | Author(s) |
|---|---|---|---|
| 3/23/2023 | A | Template created, first draft done | Eric Banach, Sarah Sollauer, George Powell |
| 4/11/2023 | B | Architecture Fixes | Eric Banach, Sarah Sollauer, George Powell |
| | | | |
| | | | |

# 2 Introduction

## 2.1 Document Purpose

The Software Design Document (SDD) is intended to provide a detailed description of our system's design and architecture. Our application, WeatherCoach, is designed to provide users with assistance in choosing appropriate outfits based on the current weather in their specified location. This SDD will serve as a guide for our team as we develop our software application. It will outline the architecture model we have selected and the testing methods we plan to use.

## 2.2 Product Overview

The system our team is planning to develop is a program to measure error in weather forecasting. For any given day's weather forecast, we want to know "Was this weather forecast wrong?" and "If so, how wrong was it?". This will be displayed as a "Prediction error" of both degrees (℉) and chance of precipitation (%). A 3-day weather forecast will also be displayed for functionality as a weather app/website.

The program will also recommend an outfit choice for the given weather. Based on a user's location and that location's weather, it will display a printed list of recommended articles of clothing such as "t-shirt, long pants, sunglasses" or perhaps "winter coat, heavy pants, thermal layers, boots." This will aid people who struggle with decision making early in the morning and will use technology to create a frictionless and accurate morning routine.

Many people have an interest in proving technology wrong; from trying to beat the GPS on one's way home to realizing a snowstorm was predicted on a bright and sunny day. We hope this program can satisfy that curiosity many people have: How much better is technology than our intuition or suspicions?

## 2.3 Product Functionality

### 2.3.1 Feature I: Weather Overview Display

The Weather Overview Display (WOD) is the default screen that clearly displays the weather in the selected location. The display only shows the basic information for the day about weather conditions such as hourly temperatures, current conditions, etc.

### 2.3.2 Feature II: Forecast Detail Display

The Forecast Detail Display (FDD) is a more in depth version of the WOD, which includes more data about the weather for the week including wind chill, UV index, etc. The user will be able to select which day in the next week they wish to view.

### 2.3.3 Feature III: Clothing Preparation Display

The Clothing Preparation Display (CPD) will recommend clothing choices based on the current weather at a user-specified location to optimize user comfort when going out. The display will show a printed list of recommended articles of clothing such as “t-shirt, long pants, sunglasses” or perhaps “winter coat, heavy pants, thermal layers, boots.” This will aid people who struggle with decision making early in the morning and will use technology to create a frictionless and accurate morning routine.

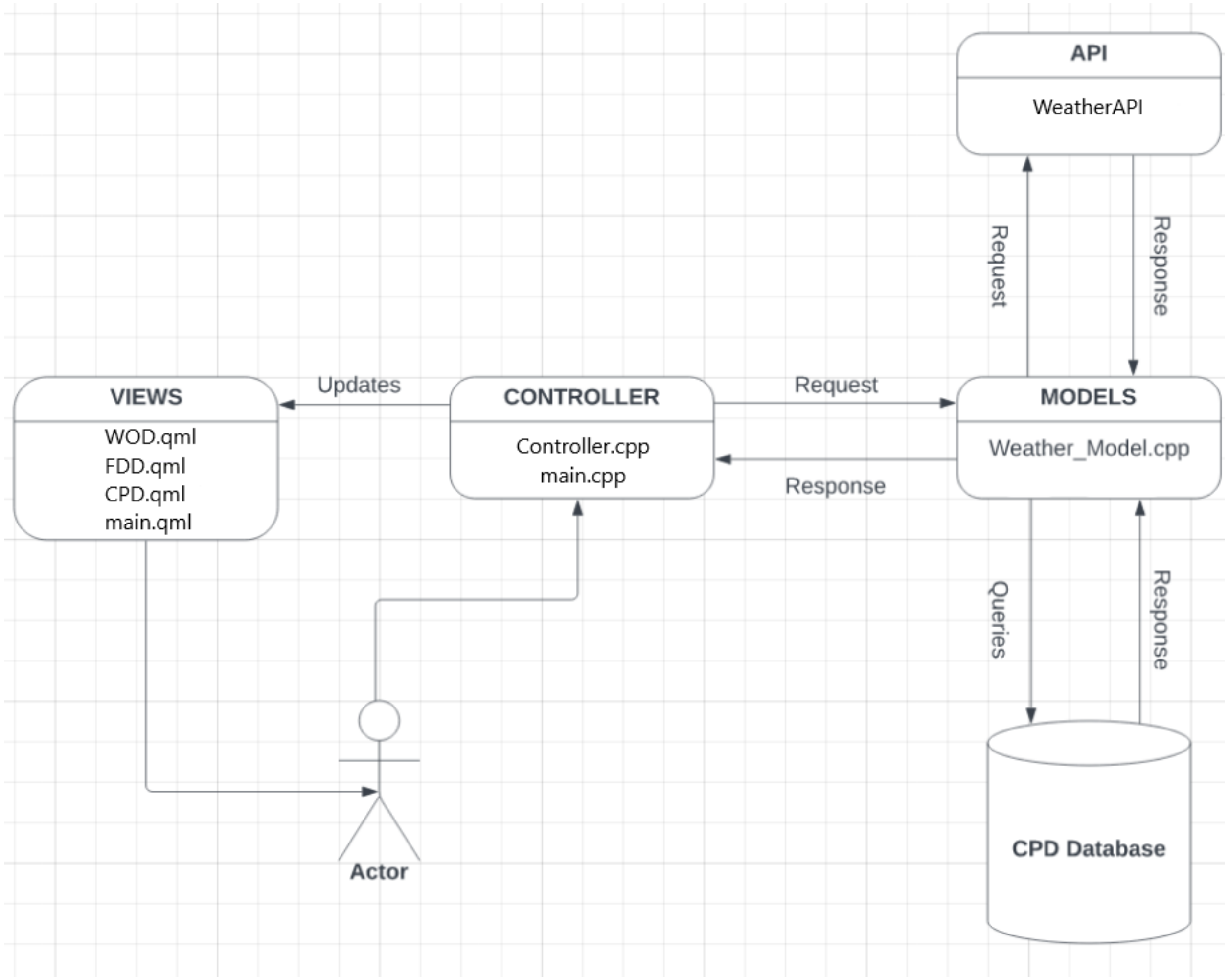
## 2.4 Acronyms and Abbreviations

| | | |
|---|---|---|
| CPD | - | Clothing Preparation Display |
| FDD | - | Forecast Detail Display |
| WOD | - | Weather Overview Display |

# 3 System Architecture

## 3.1 Overall Architecture

The project will follow the MVC (Model View Controller) architecture. The focus of MVC architecture is the separation of a view that the user will see, a controller that takes user input and passes data on to the model, and the model which does data processing and storage. The project is currently using QT with QML for the view, and will likely use C++ for the model and controller part of the application. MVC is a natural choice as QML already promotes the breakdown of components in this way.



## 3.2 Components Mapping

### 3.1.1 Functional Requirements

| Req. Number | Requirement | Mapped Component |
|---|---|---|
| **3.1.1 Graphical User Interface** | | |
| **3.1.1.1 Weather Overview Display** | | |
| 3.1.1.1.1 | The WOD shall display the current temperature at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.2 | The WOD shall display the current conditions (rain, snow, cloud, etc) at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.3 | The WOD shall display the current time at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.4 | The WOD shall display hourly temperatures for at least the next 24 hours at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.5 | The WOD shall display hourly conditions for at least the next 24 hours at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.6 | The WOD shall display a background image that reflects the current conditions at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.7 | The WOD shall display the times of sunrise and sunset at a user-selected location. | **View (WOD.qml)** |
| 3.1.1.1.8 | The WOD shall display the currently user-selected location. | **View (WOD.qml)** |
| **3.1.1.2 Forecast Detail Display** | | |
| 3.1.1.2.1 | The FDD shall display a forecast for the next 7 days at a user-selected location. | **View (FDD.qml)** |

| | | |
|---|---|---|
| 3.1.1.2.2 | The display shall allow the user to select a day within the next 7 days to view the forecast for that day in detail. | **Controller (main.cpp)** |
| 3.1.1.2.3 | A detail view for a day shall include hourly temperatures, wind chill, sky cover, UV index, humidity, precipitation chance, air quality, rain accumulation, snow accumulation, and sleet accumulation. | **View (FDD.qml)** |
| 3.1.1.2.4 | The FDD shall display a detail view for the current day as a default. | **View (FDD.qml)** |
| 3.1.1.2.5 | The FDD shall display the currently selected location. | **View (FDD.qml)** |
| **3.1.1.3 Clothing Preparation Display** | | |
| 3.1.1.3.1 | The CPD shall display a recommendation for a jacket layer, shirt layer, pants layer, shoes layer, and accessories layer based on the weather of the currently selected location. | **View (CPD.qml)** |
| 3.1.1.3.2 | The CPD shall display the currently selected location. | **View (CPD.qml)** |
| 3.1.1.3.3 | The CPD shall display the current temperature at the user-selected location. | **View (CPD.qml)** |
| 3.1.1.3.4 | The CPD shall display an indication of the trend of the temperature for the rest of the day. | **View (CPD.qml)** |
| **3.1.2 Clothing Preparation Database** | | |
| **3.1.2.1 Database Initialization** | | |
| 3.1.2.1.1 | The system shall use SQL language for the database management system. | **Model (CPD_Model.cpp)** |
| 3.1.2.1.2 | The database shall contain at least one table. | **Backend DB** |
| 3.1.2.1.3 | The database shall store at least one article of clothing for each layer described in 3.1.1.3.1. | **Backend DB** |

| | | |
|---|---|---|
| 3.1.2.1.4 | The database shall store weather conditions. | **Backend DB** |
| 3.1.2.1.5 | The database shall store temperatures. | **Backend DB** |
| **3.1.2.2 Linking Clothes to Weather Conditions** | | |
| 3.1.2.2.1 | Each clothing item shall have at least one paired weather condition. | **Backend DB** |
| 3.1.2.2.2 | Each clothing item shall have at least one paired temperature range. | **Backend DB** |
| **3.1.2.3 Adding User Clothing** | | |
| 3.1.2.3.1 | The database shall be updated with the users choices of clothing. | **Model (CPD_Model.cpp)** |
| **3.1.3 Weather API** | | |
| **3.1.3.1 Retrieving Data** | | |
| 3.1.3.1.1 | The application shall retrieve the forecast at a user-selected location for the next 7 days from a weather API. | **Model (Weather_Model.cpp)** |
| 3.1.3.1.2 | The application shall retrieve the current weather conditions at a user-selected location from a weather API. | **Model (Weather_Model.cpp)** |
| 3.1.3.1.3 | The application shall retrieve the current temperature at a user-selected location from a weather API. | **Model (Weather_Model.cpp)** |
| **3.1.4 Hardware Compatibility** | | |
| **3.1.4.1 Mobile Devices** | | |
| 3.1.4.1.1 | The application shall be developed for Android mobile devices. | **N/A** |
| 3.1.4.1.2 | The application will be supported on IOS mobile devices. | **N/A** |

### 3.2.2 Non-Functional Requirements

| Req. Number | Requirement | Mapped Component |
|---|---|---|
| **3.2.1 Graphical User Interface** | | |
| **3.1.1.1 General UI** | | |
| 3.2.1.1.1 | All weather displays (WOD, FDD) shall contain a hamburger menu to select the location for which the weather data is displayed. | **View (WOD.qml, FDD.qml) Controller (main.cpp)** |
| 3.2.1.1.2 | The CPD shall contain a hamburger menu to select what clothes the user owns to be used for clothing recommendations. | **View (CPD.qml) Controller (main.cpp)** |
| 3.2.1.1.3 | The CPD hamburger menu shall only save user choices when a jacket layer, shirt layer, pants layer, shoes layer, and accessories layer have been chosen. | **Controller (main.cpp)** |
| 3.2.1.1.4 | All displays shall have a menu bar that allows the user to switch between the WOD, FDD, and CPD. | **View (*.qml) Controller (main.cpp)** |
| 3.2.1.1.5 | All main displays (WOD, FDD, CPD) shall allow the user to swipe to change the currently shown display. | **Controller (main.cpp)** |
| **3.2.2 Data Updating** | | |
| **3.2.2.1 Database Updates** | | |
| 3.2.2.1.1 | All updates to the clothing database shall be completed in less than 5 seconds. | **Model (CPD_Model.cpp)** |
| **3.2.2.2 Weather API** | | |
| 3.2.2.2.1 | All calls to the weather API shall be returned within 5 seconds. | **Model (Weather_Model.cpp)** |

| | | |
|---|---|---|
| 3.2.2.2.2 | All calls to the weather API shall return the most recent forecast data. | **Model (Weather_Model.cpp)** |
| 3.2.2.2.3 | All calls to the weather API shall return the most recent temperature data. | **Model (Weather_Model.cpp)** |
| 3.2.2.2.4 | All calls to the weather API shall return the most recent conditions data. | **Model (Weather_Model.cpp)** |
| **3.2.3 Availability** | | |
| **3.2.3.1 Availability due to API** | | |
| 3.2.3.1.1 | The application shall be available as long as the weather API is available. | **N/A** |

## 3.3 Technology Stack Selection

For the Views, the project will use Qt Quick for the development of all Graphical User Interfaces (GUI). Qt Quick uses QML and is specifically designed for the development of mobile applications and GUIs. The Qt Designer Studio that comes with the installation of Qt comes with a graphical drag +and drop designer that allows the user to drag GUI components directly where they are needed. This allows for quick creation of components that can be accessed through the controller and used to modify and update data in the backend database. While our group is generally unfamiliar with QML, the Designer Studio is difficult to master but very easy to pick up as a beginner.

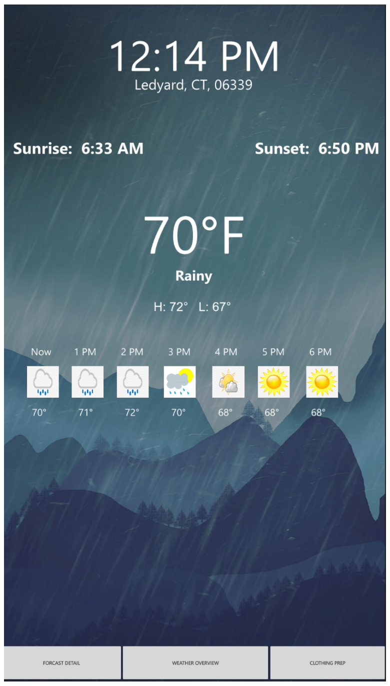
For the Model & Controller, the project will use Qt with C++. These controllers manage all user input and pass information to and from the backend database. Qt allows for easy integration between the GUI with QML and the C++ backend. Creating new classes to link to the QML files is quick and is easily configured within the QML project file.  C++ is also a language our team will be much more familiar with due to its similarity to other Object Oriented Languages such as Java. As a result, writing the controller code will be more efficient with C++ than using alternative languages we are less familiar with such as Python. The model will also be linked to a database which can be done simply using the C++ SQLite libraries.

For the backend Database, we will use SQLite. SQLite is a simple version of SQL that does not require hosting a server for the database and as such can be stored directly on the targeted system. SQLite has built-in libraries in the Qt framework and as a result will be easy to integrate into the application. These libraries allow us to give the database queries directly from our Model so we can avoid creating another interface to do so.

# 4 System Design

## 4.1 UI

### 4.1.1 WOD:



12:14 PM

Ledyard, CT, 06339

Sunrise: 6:33 AM

Sunset: 6:50 PM

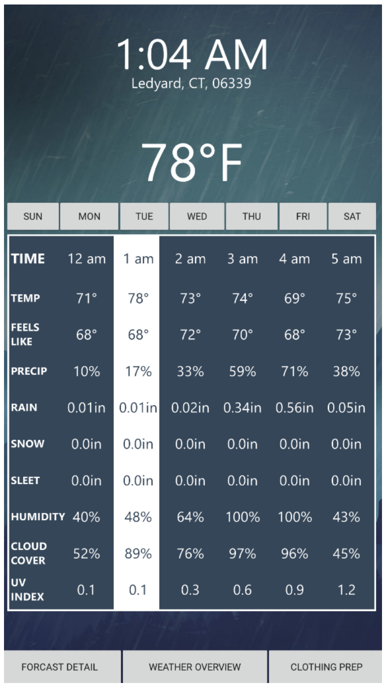
70°F

Rainy

H: 72° L: 67°

| Now | 1 PM | 2 PM | 3 PM | 4 PM | 5 PM | 6 PM |
|---|---|---|---|---|---|---|
| 70° | 71° | 72° | 70° | 68° | 68° | 68° |

### 4.1.2 FDD:



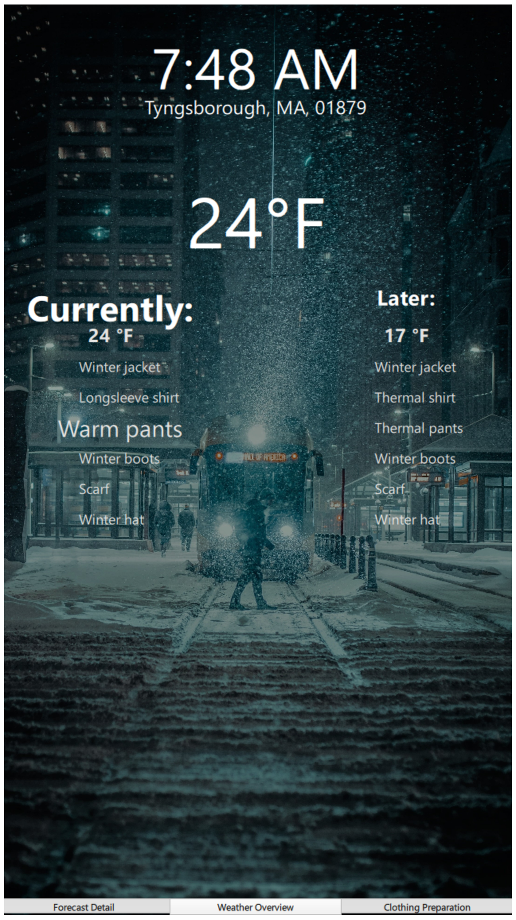
1:04 AM

Ledyard, CT, 06339

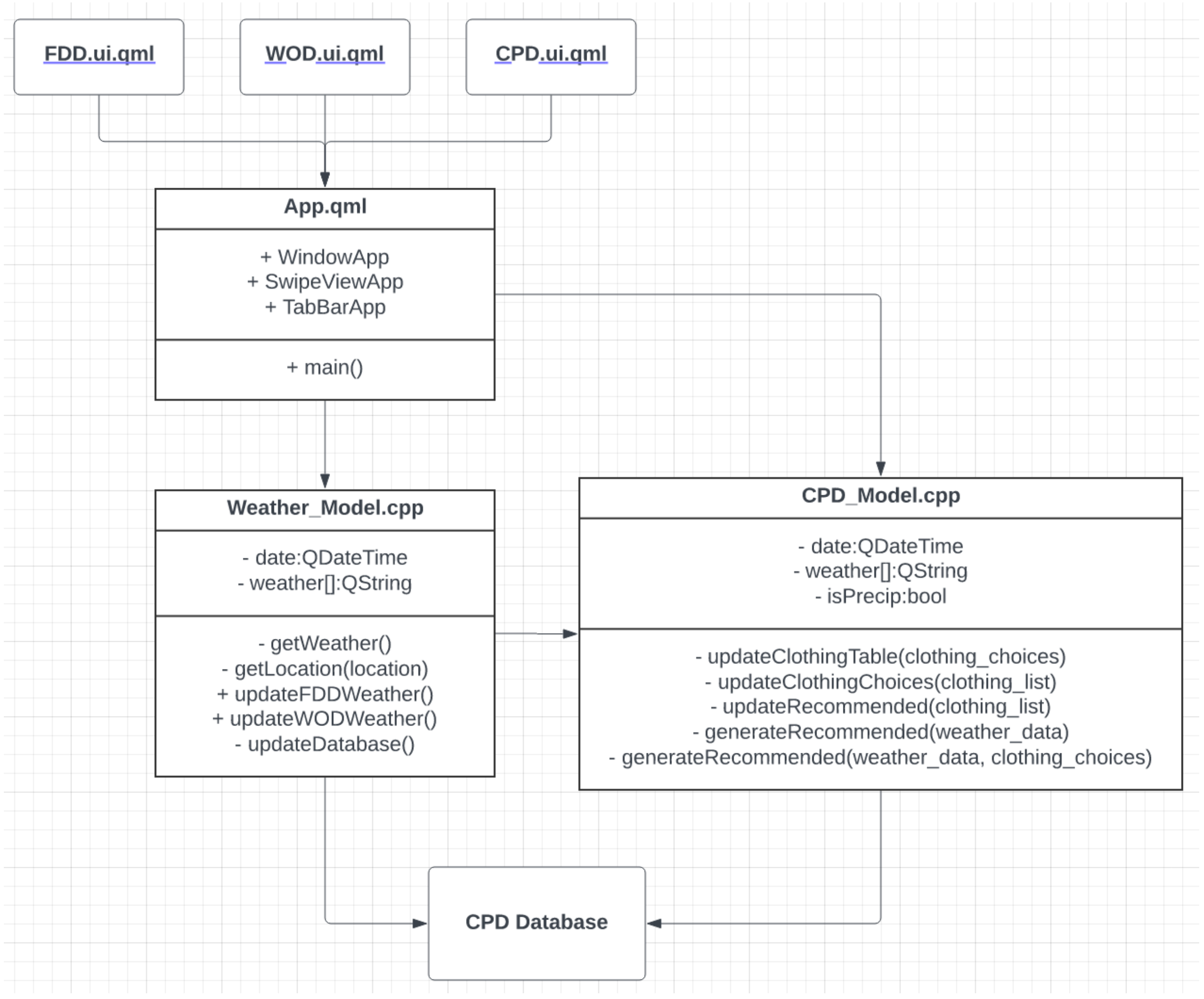
78°F

| TIME | 12 am | 1 am | 2 am | 3 am | 4 am | 5 am |
|---|---|---|---|---|---|---|
| TEMP | 71° | 78° | 73° | 74° | 69° | 75° |
| FEELS LIKE | 68° | 68° | 72° | 70° | 68° | 73° |
| PRECIP | 10% | 17% | 33% | 59% | 71% | 38% |
| RAIN | 0.01in | 0.01in | 0.02in | 0.34in | 0.56in | 0.05in |
| SNOW | 0.0in | 0.0in | 0.0in | 0.0in | 0.0in | 0.0in |
| SLEET | 0.0in | 0.0in | 0.0in | 0.0in | 0.0in | 0.0in |
| HUMIDITY | 40% | 48% | 64% | 100% | 100% | 43% |
| CLOUD COVER | 52% | 89% | 76% | 97% | 96% | 45% |
| UV INDEX | 0.1 | 0.1 | 0.3 | 0.6 | 0.9 | 1.2 |

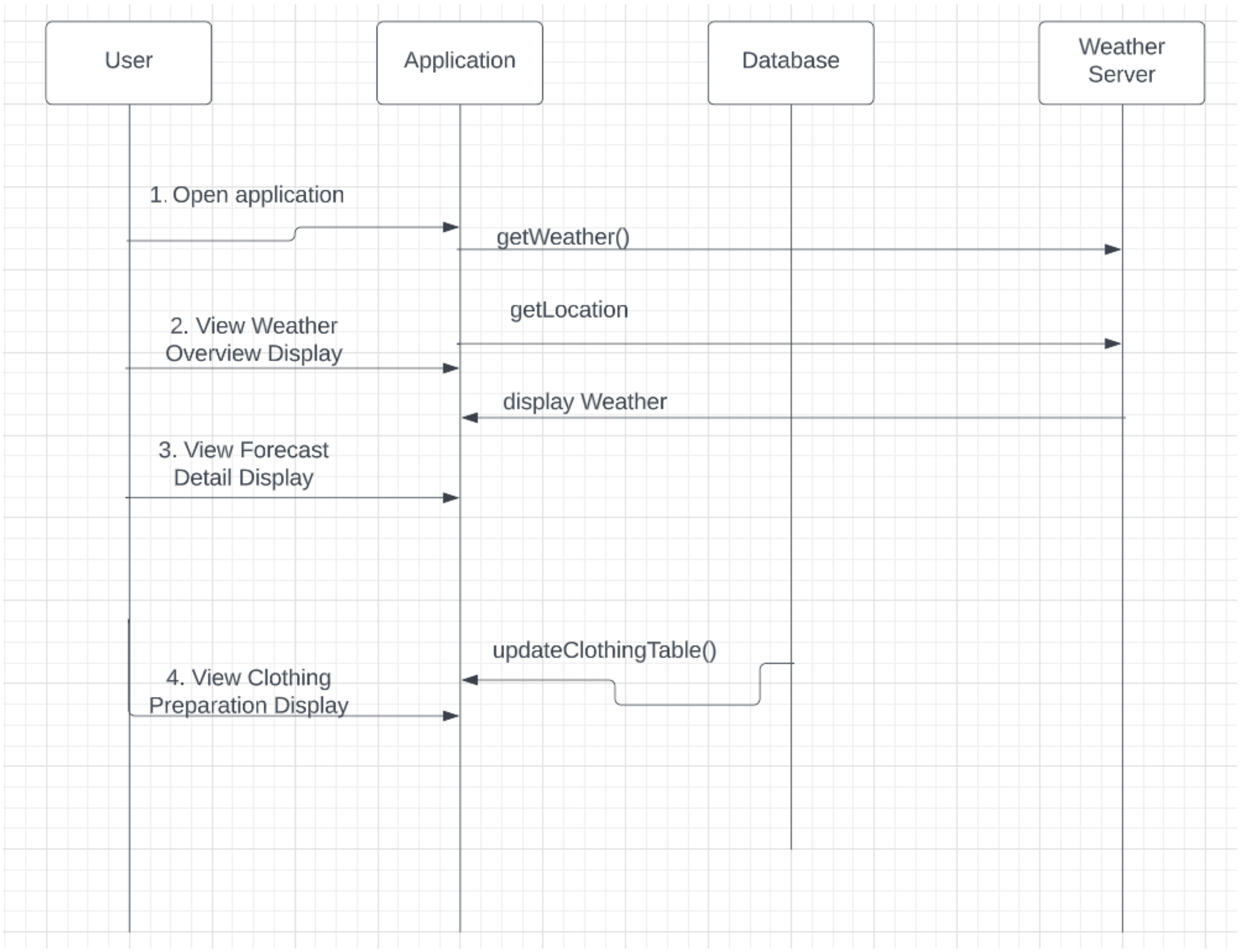
4.1.3 CPD:


7:48 AM
Tyngsborough, MA, 01879
24°F
Currently:
24 °F
Winter jacket
Longsleeve shirt
Warm pants
Winter boots
Scarf
Winter hat
Later:
17 °F
Winter jacket
Thermal shirt
Thermal pants
Winter boots
Scarf
Winter hat
Forecast Detail
Weather Overview
Clothing Preparation

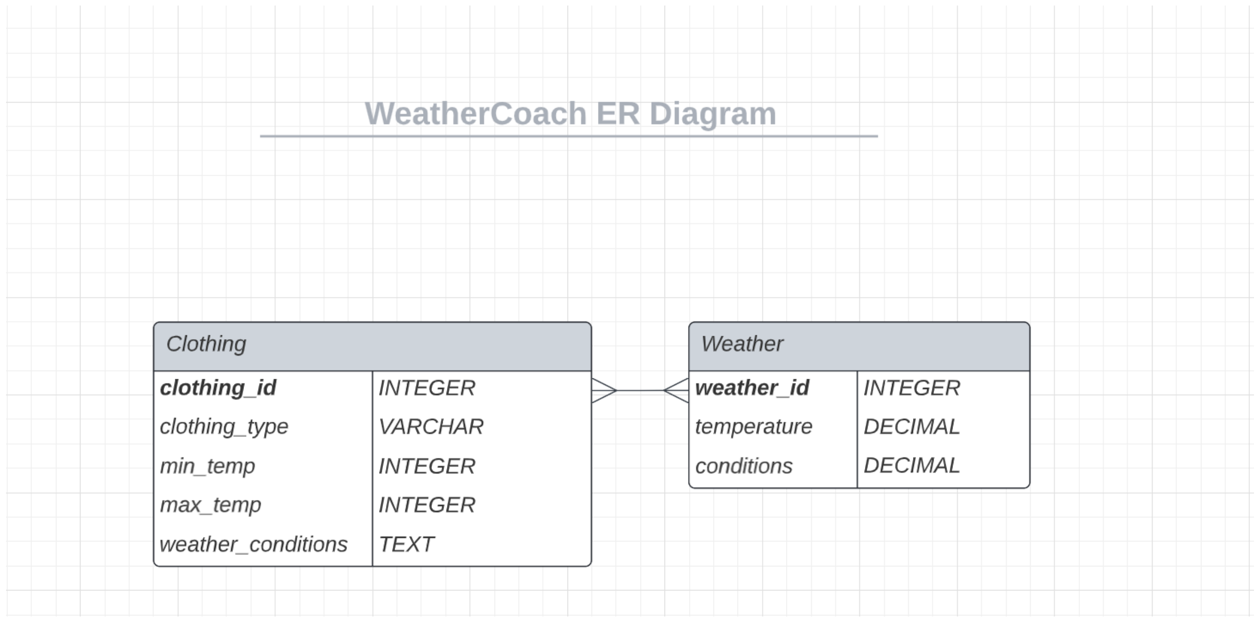
## 4.2 Class Diagram



## 4.3 Sequence Diagram



## 4.4 Database



# 5 Test Plan

## 5.1 Function #1 - getLocation(location)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| *1* | *Take in a zip code as user input and return the associated city.* | ‘02770’ | Return the location “Rochester, Massachusetts” as text. |
| *2* | *Take in a city name as user input.* | “Rochester, Massachusetts” | Return the location “Rochester, Massachusetts” as text. |

## 5.2 Function #2 - getWeather(location)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| *1* | *Takes a location as input and returns appropriate data for this location* | “New York, NY” | Return a dictionary that contains the current weather information for New York City. |

## 5.3 Function #3 - generateRecommended(weather_data)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| *1* | *Function returns a list of clothing items appropriate for the given weather conditions (cool, rainy)* | weather_data = {‘temperature’: 60, ‘precipitation’: 80%} | [‘long sleeve’, ‘raincoat’, ‘long pants’, ‘umbrella’] |
| *2* | *Function returns a list of clothing items appropriate for the given weather conditions (warm, no precipitation)* | weather_data = {‘temperature’: 80, ‘precipitation’: 0%} | [‘short sleeves’, ‘shorts’] |

## 5.4 Function #4 - updateRecommended(clothing_list)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| *1* | *Uses the clothing_list to generate an image on the CPD.* | [“jacket”, “boots”, “hat”] | The function should display the clothing recommendations as a visual with images of each item of clothing. |
| *2* | *Given an empty array, clothing_list* | [] | The function should display a generic image of someone with a t-shirt, pants, and shoes if an empty array is returned. |

## 5.5 Function #5 - updateClothingChoices(clothing_list)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| *1* | *Updates clothing choices list on the CPD that is seen in the hamburger menu.* | [“jacket”, “hat”, “boots”, “long sleeve”, “short sleeves”, “long pants”, “shorts”] | The function should update the clothing choices to the items selected by the user in the CPD, the selected items should show up as checked off. |

## 5.6 Function #6 - updateClothingTable(clothing_choices)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| *1* | *The database is updated with the items of clothing selected by the user.* | [‘hat’, ‘long sleeves’, ‘long pants’, ‘shorts’, ‘dress’, ‘raincoat’, ‘umbrella’] | clothing_choices table should be updated in the DB with the selected data |

| | | | |
|---|---|---|---|
| 2 | *The database is updated and set to a default when no choice is made by the user.* | [] | clothing_choices table should be set to set to the default, which includes every item of clothing available |

## 5.7 Function #7 - generateRecommended(weather_data, clothing_choices)

| *No.* | *Test case* | *User input* | *Pass criteria* |
|---|---|---|---|
| 1 | *Function returns a list of clothing items appropriate for the given weather conditions(cool, rainy) based on the user's clothing selection.* | weather_data = {'temperature': 60, 'precipitation': 80%}<br><br>clothing_choices = ['hat', 'long sleeves', 'long pants', 'shorts', 'dress', 'raincoat', 'umbrella'] | ['long sleeves', 'raincoat', 'long pants', 'umbrella'] |
| 2 | *Function returns a list of clothing items appropriate for the given weather conditions (warm, no precipitation) based on the user's clothing selection.* | weather_data = {'temperature': 80, 'precipitation': 0%}<br><br>clothing_choices = ['hat', 'short sleeves', 'long sleeves', 'long pants', 'shorts', 'dress', 'raincoat', 'umbrella'] | ['short sleeves', 'shorts'] |