Climate Watch

ICS 372 - 01

Jordan Mielke, Mohamed Yusuf, Steve Hartmann

What problem are we solving?

- Incomplete or outdated weather information for local locations
- Difficulty in planning daily activities or coordinating travel without accurate and up-to-date forecasts

How are we solving that problem?

- Providing Detailed Forecasts
  - Highs and lows for not only that day, but also the next five days
- Extensive Details
  - On wind speed, humidity, visibility and dew point
- Comprehensive Planning
  - Ideal for organizing a family vacation and daily activities

Functional Requirements & How each was satisfied (Pt. 1)

- Display Current Weather for City
  - Upon launching the app, a pop-up is shown giving the user the ability to read current weather data using the user's location
- Search by ZIP Code
  - Enter a ZIP code into the search
  - If you enter an invalid ZIP, you will get an error message with examples
- Unit Conversions
  - When accessing the user preferences screen, you will be able to toggle between units

Functional Requirements & How each was satisfied (Pt. 2)

- Responsive UI Components
  - Interacting with various UI components will ensure a responsive UI
- Location Services
  - When loading app you will be prompted with location tracking
- Weather Cache
  - When searching for ZIP codes, it will check previously entered ZIPs

Non-Functional Requirements & How each was satisfied (Pt. 1)

- JAVAFX for GUI
    - Our application uses JavaFX to create a user-friendly graphical interface
- Implement Weather API for weather Data
    - We utilize the OpenWeather API (https://openweathermap.org) to fetch real-time weather data

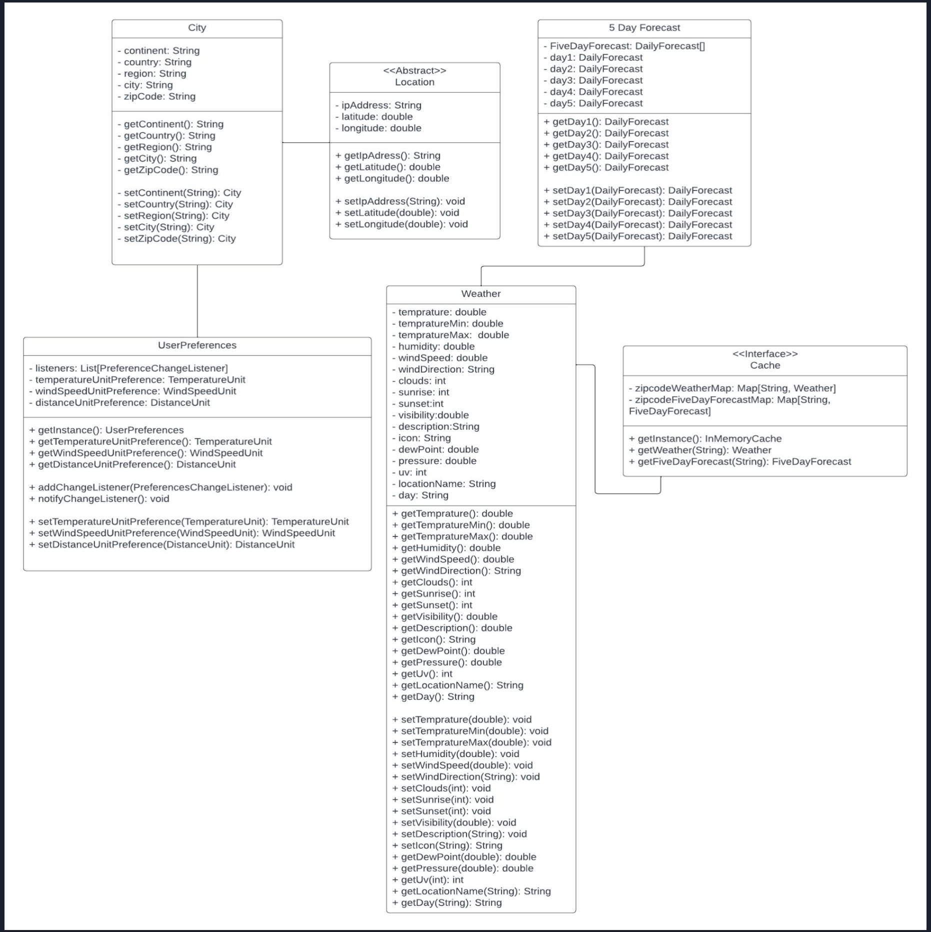
Non-Functional Requirements & How each was satisfied (Pt. 2)

- Use IP geolocation API for converting user IP address to location
  - Our application incorporates the Ipstack API (https://ipstack.com) to convert the user's IP address into a geographical location
- Error Handling
  - We have implemented comprehensive input validation and error handling mechanisms

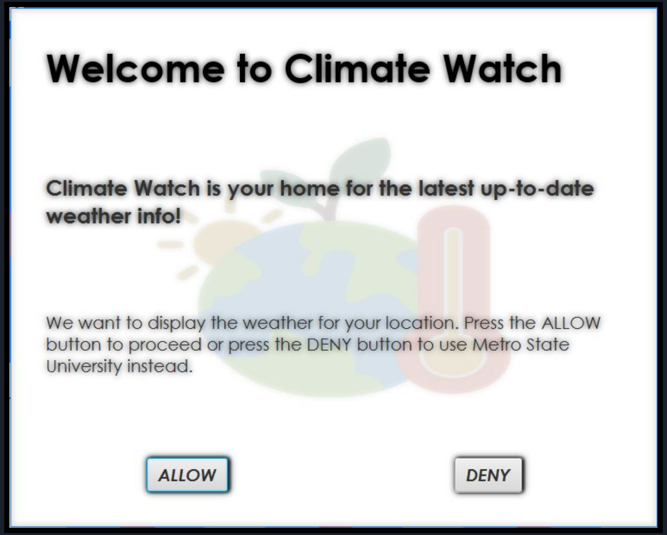
How is the application implemented internally?

- Model
  - Contains classes such as City, Weather, and User, representing the data model, as well as strongly typed enums to hold units and conversions for temperature, distance, and wind speed
- Controller
  - HomeController, UserPrefController, and WelcomeController to manage user interactions
- Service
  - CityApiService and WeatherApiService to handle REST API calls for fetching city and weather data
- Utils
  - Utility classes like IpUtils, TimeUtils, and ZipCodeUtils to provide auxiliary functionalities
- Cache
  - An in-memory cache as intermediate storage for fetched weather data to reduce REST API calls

UML Diagram


City
- continent: String
- country: String
- region: String
- city: String
- zipCode: String
- getContinent(): String
- getCountry(): String
- getRegion(): String
- getCity(): String
- getZipCode(): String
- setContinent(String): City
- setCountry(String): City
- setRegion(String): City
- setCity(String): City
- setZipCode(String): City
<<Abstract>>
Location
- ipAddress: String
- latitude: double
- longitude: double
+ getIpAdress(): String
+ getLatitude(): double
+ getLongitude(): double
+ setIpAddress(String): void
+ setLatitude(double): void
+ setLongitude(double): void
5 Day Forecast
- FiveDayForecast: DailyForecast[]
- day1: DailyForecast
- day2: DailyForecast
- day3: DailyForecast
- day4: DailyForecast
- day5: DailyForecast
+ getDay1(): DailyForecast
+ getDay2(): DailyForecast
+ getDay3(): DailyForecast
+ getDay4(): DailyForecast
+ getDay5(): DailyForecast
+ setDay1(DailyForecast): DailyForecast
+ setDay2(DailyForecast): DailyForecast
+ setDay3(DailyForecast): DailyForecast
+ setDay4(DailyForecast): DailyForecast
+ setDay5(DailyForecast): DailyForecast
UserPreferences
- listeners: List[PreferenceChangeListener]
- temperatureUnitPreference: TemperatureUnit
- windSpeedUnitPreference: WindSpeedUnit
- distanceUnitPreference: DistanceUnit
+ getInstance(): UserPreferences
+ getTemperatureUnitPreference(): TemperatureUnit
+ getWindSpeedUnitPreference(): WindSpeedUnit
+ getDistanceUnitPreference(): DistanceUnit
+ addChangeListener(PreferencesChangeListener): void
+ notifyChangeListener(): void
+ setTemperatureUnitPreference(TemperatureUnit): TemperatureUnit
+ setWindSpeedUnitPreference(WindSpeedUnit): WindSpeedUnit
+ setDistanceUnitPreference(DistanceUnit): DistanceUnit
Weather
- temprature: double
- tempratureMin: double
- tempratureMax: double
- humidity: double
- windSpeed: double
- windDirection: String
- clouds: int
- sunrise: int
- sunset:int
- visibility:double
- description:String
- icon: String
- dewPoint: double
- pressure: double
- uv: int
- locationName: String
- day: String
+ getTemprature(): double
+ getTempratureMin(): double
+ getTempratureMax(): double
+ getHumidity(): double
+ getWindSpeed(): double
+ getWindDirection(): String
+ getClouds(): int
+ getSunrise(): int
+ getSunset(): int
+ getVisibility(): double
+ getDescription(): double
+ getIcon(): String
+ getDewPoint(): double
+ getPressure(): double
+ getUv(): int
+ getLocationName(): String
+ getDay(): String
+ setTemprature(double): void
+ setTempratureMin(double): void
+ setTempratureMax(double): void
+ setHumidity(double): void
+ setWindSpeed(double): void
+ setWindDirection(String): void
+ setClouds(int): void
+ setSunrise(int): void
+ setSunset(int): void
+ setVisibility(double): void
+ setDescription(String): void
+ setIcon(String): String
+ getDewPoint(double): double
+ getPressure(double): double
+ getUv(int): int
+ getLocationName(String): String
+ getDay(String): String
<<Interface>>
Cache
- zipcodeWeatherMap: Map[String, Weather]
- zipcodeFiveDayForecastMap: Map[String, FiveDayForecast]
+ getInstance(): InMemoryCache
+ getWeather(String): Weather
+ getFiveDayForecast(String): FiveDayForecast

User Interface (Pt. 1)

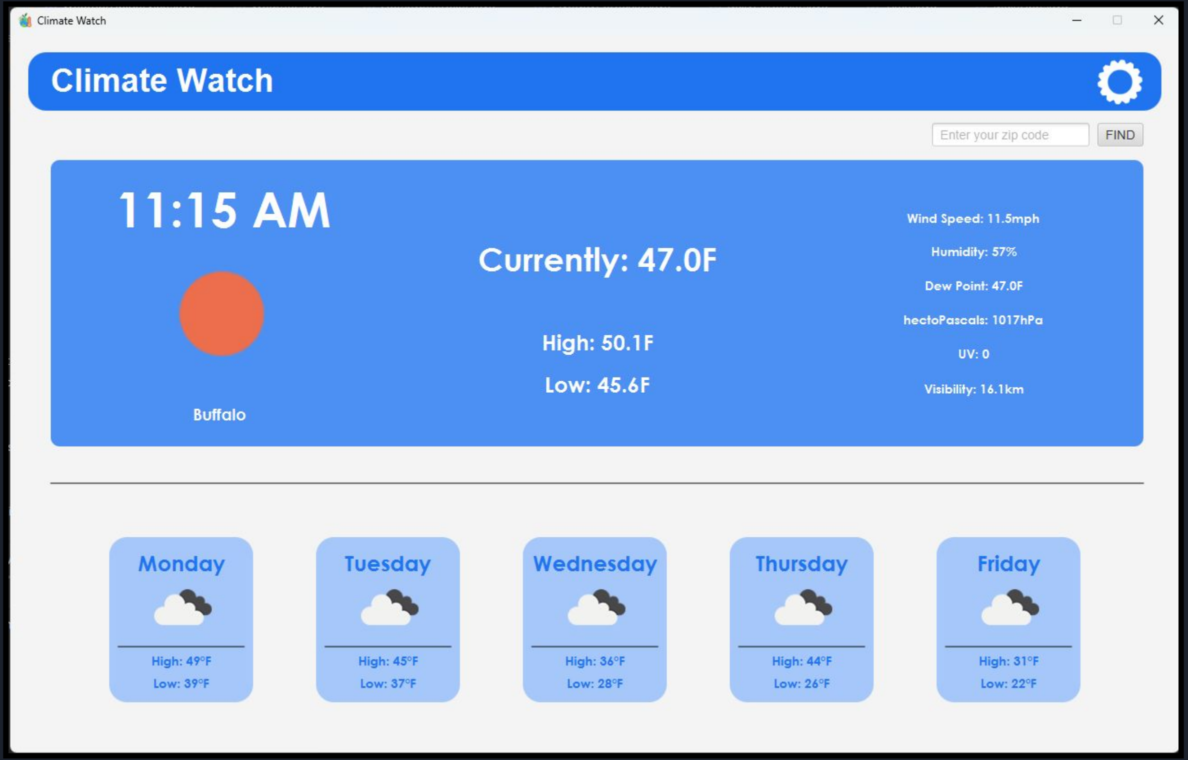


Welcome to Climate Watch

Climate Watch is your home for the latest up-to-date weather info!

We want to display the weather for your location. Press the ALLOW button to proceed or press the DENY button to use Metro State University instead.

User Interface (Pt. 2)



Climate Watch

Climate Watch

Enter your zip code FIND

11:15 AM

Currently: 47.0F

High: 50.1F

Low: 45.6F

Buffalo

Wind Speed: 11.5mph

Humidity: 57%

Dew Point: 47.0F

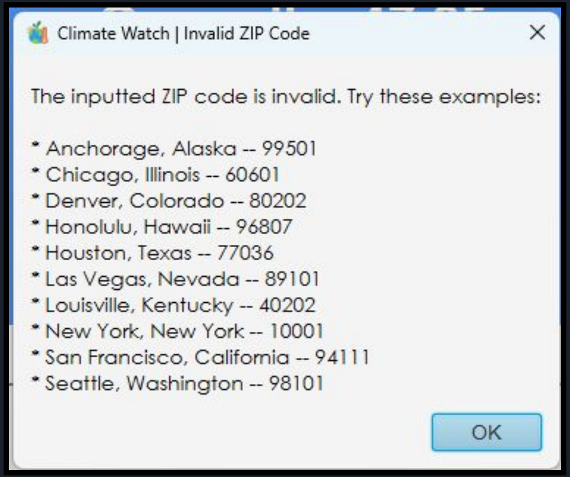
hectoPascals: 1017hPa

UV: 0

Visibility: 16.1km

Monday	Tuesday	Wednesday	Thursday	Friday
High: 49°F	High: 45°F	High: 36°F	High: 44°F	High: 31°F
Low: 39°F	Low: 37°F	Low: 28°F	Low: 26°F	Low: 22°F

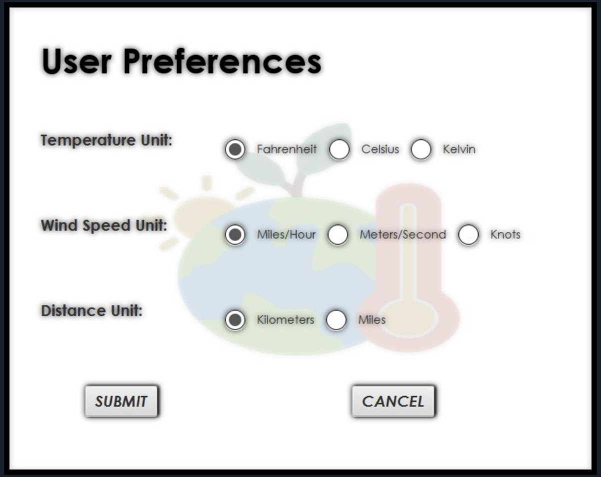
User Interface (Pt. 3)



The inputted ZIP code is invalid. Try these examples:

* Anchorage, Alaska -- 99501
* Chicago, Illinois -- 60601
* Denver, Colorado -- 80202
* Honolulu, Hawaii -- 96807
* Houston, Texas -- 77036
* Las Vegas, Nevada -- 89101
* Louisville, Kentucky -- 40202
* New York, New York -- 10001
* San Francisco, California -- 94111
* Seattle, Washington -- 98101

User Interface (Pt. 4)



User Preferences

Temperature Unit:	[✓ Fahrenheit] Celsius Kelvin
Wind Speed Unit:	[✓ Miles/Hour] Meters/Second Knots
Distance Unit:	[✓ Kilometers] Miles

What went well?

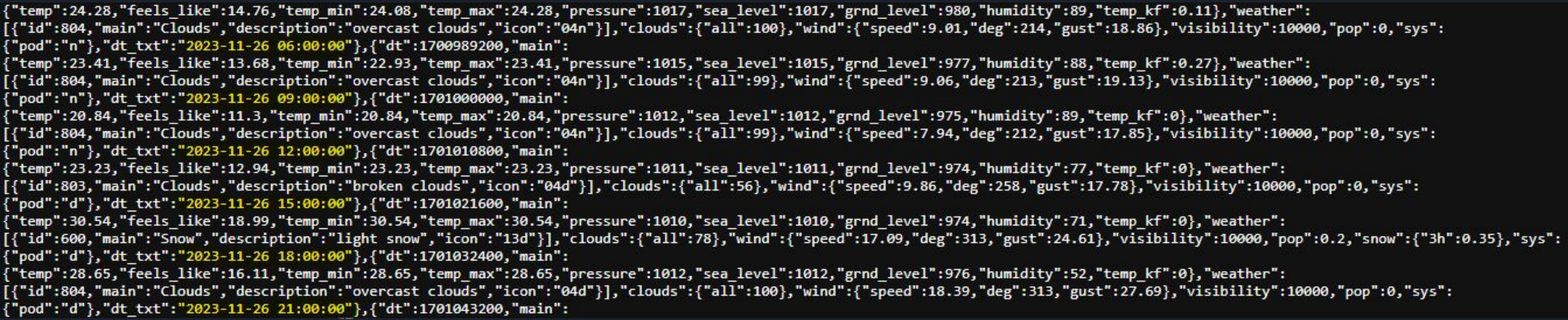
- Modular Development
  - Facilitated independent work
  - Reduced conflicts
- JavaFX UI Design
  - Aesthetically pleasing user interface
  - Enhanced user friendliness
- Effective Communication
  - Shared phone numbers
  - Group chat with ongoing process
  - Weekly Friday meetings

What didn’t go well?

- API Integration Challenges
  - Issues with incorporating some features from the REST API’s
- Timezone Data limitations
  - Five day forecast was a little rough. They had it in a format that did not fit our particular needs
- Task organization
  - Did not have a clear outline of each person's tasks, could have been improved with a system similar to JIRA

What didn’t go well?

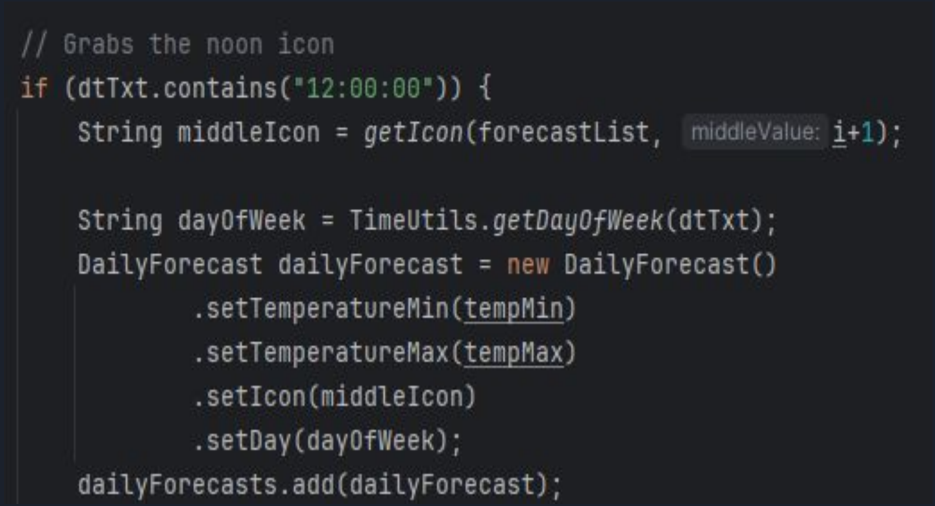
- In regards to the challenges with the timezone data limitations:



```
{"temp":24.28,"feels_like":14.76,"temp_min":24.08,"temp_max":24.28,"pressure":1017,"sea_level":1017,"grnd_level":980,"humidity":89,"temp_kf":0.11},"weather":
[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04n"}],"clouds":{"all":100},"wind":{"speed":9.01,"deg":214,"gust":18.86},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2023-11-26 06:00:00"},{"dt":1700989200,"main":
{"temp":23.41,"feels_like":13.68,"temp_min":22.93,"temp_max":23.41,"pressure":1015,"sea_level":1015,"grnd_level":977,"humidity":88,"temp_kf":0.27},"weather":
[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04n"}],"clouds":{"all":99},"wind":{"speed":9.06,"deg":213,"gust":19.13},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2023-11-26 09:00:00"},{"dt":1701000000,"main":
{"temp":20.84,"feels_like":11.3,"temp_min":20.84,"temp_max":20.84,"pressure":1012,"sea_level":1012,"grnd_level":975,"humidity":89,"temp_kf":0},"weather":
[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04n"}],"clouds":{"all":99},"wind":{"speed":7.94,"deg":212,"gust":17.85},"visibility":10000,"pop":0,"sys":
{"pod":"n"},"dt_txt":"2023-11-26 12:00:00"},{"dt":1701010800,"main":
{"temp":23.23,"feels_like":12.94,"temp_min":23.23,"temp_max":23.23,"pressure":1011,"sea_level":1011,"grnd_level":974,"humidity":77,"temp_kf":0},"weather":
[{"id":803,"main":"Clouds","description":"broken clouds","icon":"04d"}],"clouds":{"all":56},"wind":{"speed":9.86,"deg":258,"gust":17.78},"visibility":10000,"pop":0,"sys":
{"pod":"d"},"dt_txt":"2023-11-26 15:00:00"},{"dt":1701021600,"main":
{"temp":30.54,"feels_like":18.99,"temp_min":30.54,"temp_max":30.54,"pressure":1010,"sea_level":1010,"grnd_level":974,"humidity":71,"temp_kf":0},"weather":
[{"id":600,"main":"Snow","description":"light snow","icon":"13d"}],"clouds":{"all":78},"wind":{"speed":17.09,"deg":313,"gust":24.61},"visibility":10000,"pop":0.2,"snow":{"3h":0.35},"sys":
{"pod":"d"},"dt_txt":"2023-11-26 18:00:00"},{"dt":1701032400,"main":
{"temp":28.65,"feels_like":16.11,"temp_min":28.65,"temp_max":28.65,"pressure":1012,"sea_level":1012,"grnd_level":976,"humidity":52,"temp_kf":0},"weather":
[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04d"}],"clouds":{"all":100},"wind":{"speed":18.39,"deg":313,"gust":27.69},"visibility":10000,"pop":0,"sys":
{"pod":"d"},"dt_txt":"2023-11-26 21:00:00"},{"dt":1701043200,"main":
```

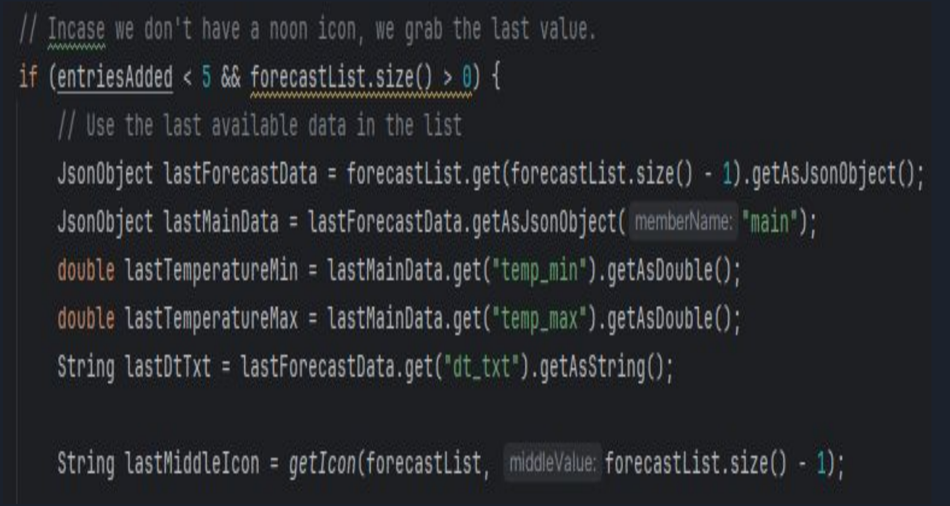
What didn’t go well?

- Solution to have the five-day forecast fit our needs:



```
// Grabs the noon icon
if (dtTxt.contains("12:00:00")) {
    String middleIcon = getIcon(forecastList, middleValue: i+1);

    String dayOfWeek = TimeUtils.getDayOfWeek(dtTxt);
    DailyForecast dailyForecast = new DailyForecast()
            .setTemperatureMin(tempMin)
            .setTemperatureMax(tempMax)
            .setIcon(middleIcon)
            .setDay(dayOfWeek);
    dailyForecasts.add(dailyForecast);
```



```
// Incase we don't have a noon icon, we grab the last value.
if (entriesAdded < 5 && forecastList.size() > 0) {
    // Use the last available data in the list
    JsonObject lastForecastData = forecastList.get(forecastList.size() - 1).getAsJsonObject();
    JsonObject lastMainData = lastForecastData.getAsJsonObject( memberName: "main");
    double lastTemperatureMin = lastMainData.get("temp_min").getAsDouble();
    double lastTemperatureMax = lastMainData.get("temp_max").getAsDouble();
    String lastDtTxt = lastForecastData.get("dt_txt").getAsString();

    String lastMiddleIcon = getIcon(forecastList, middleValue: forecastList.size() - 1);
```

What would we do differently

- Better API Integration planning
  - Develop a better plan around the API return values, which would have been very helpful for the five-day forecast
- Have better task organization
  - This would have helped show each members current tasks along with future tasks
  - This would have also helped with our time management on the list view that would have contained five US and five non-US cities

What do we wish we knew at the beginning

- Wish we understood how tricky the API would be for our particular use
- Wish we also knew how tricky it would be to create UI screens using the built in JavaFX UI element
    - A lot of trial and error
    - Deciding which UI elements to use required a lot of digging through JavaDocs, using online tutorials, and articles
    - We used the SceneBuilder tool, but it wasn’t integrated into IntelliJ very well

What did we learn throughout the project

- Complexity Management
  - Valuable experience in breaking down complex tasks into manageable components
- Integration Expertise
  - Learned effective integration of various components, especially external features
- Significance of early planning
  - Understanding the critical role of early planning
  - Steps to take to plan an effective development process
- Coding Principles
  - Follow the KISS principle
  - You spend most of your time on maintainability rather than writing new code
- **The more challenging you make things to understand, the more difficult it is to make changes in the future**

How did our team organize work?

- Weekly class time, along with weekly meetings on Friday
  - We used this to set goals and update on progress of each person's tasks
  - Ensured that everyone was on the same page
  - Allowed us to communicate if we were struggling with implementing a task

How did we share our code?

- GitHub
  - Allowed us to share code effectively by being able to use features like branches, pull requests, merge
  - Allowed us to work on different aspects of the project simultaneously without overriding each others work

How did our team come up with features?

- Brainstorming sessions
  - Feature development was done in brainstorming sessions on Mondays
  - Further into the semester we were able to independently work on features throughout the week
- Discussion and refinement
  - These independent features were then discussed on Fridays
  - Made sure they aligned with the overall project vision and requirements

How did we test features?

- Independently
  - Each member was in charge of testing their developed features
  - Testing was done before committing any changes to the main branch
- Stress testing
  - Once a change was made, the other two members would each stress test those changes, which allowed us to maintain integrity of the application

How did we decide when the feature was complete?

- Completion Criteria
  - Feature was deemed complete after it has meet the criteria previous discussed on either Monday or Friday
  - Also after each member had fully tested the feature to make sure it was working as intended
- Milestone Checks
  - We also looked at the milestones reports to make sure nothing was missed and we covered all of our bases

Jordan’s Responsibilities

- Creating the outlines for each milestone
- Creating the home screen of the app
- Five-day forecast using REST API calls, and implementation
- Additional thorough stress testing the app to ensure a reliable app

Mohamed's Responsibilities

- Hosting the Repository for the Application and Zoom Meetings
- Creating an account to access OpenWeather API data
- Transferring the Initial UML Diagram into code
- Implementing User Preferences through application-wide conversions

Steve’s Responsibilities

- Welcome modal
- User preferences modal
- IP address lookup using lookup URLs
- City lookup using REST API call and implementation
- In-memory cache to reduce weather REST API calls
- Code organization and cleanup

Conclusion

- Our journey with this project has been a learning experience
- Acknowledging successes, challenges, and the continuous on improvements
- Glad we got to apply software development principles and enhance our skills

DEMO

Any Questions?