## Assignment: Enhance UI with Detailed Weather Information

### Objective

Your goal is to expand the weather application by displaying more detailed information from the weather data response. Specifically, you will parse additional fields from the **`days`** array in the JSON response and present this data in a user-friendly manner in the app's UI.

### Background

The weather API provides a comprehensive dataset, including maximum and minimum temperatures, humidity, precipitation, and more, for each day. This assignment focuses on utilizing this data to give users a more detailed view of the weather forecast.

### Requirements

**Data Class Modification**: Extend the **`DayInfo`** data class within **`WeatherData`** to include additional fields such as **`humidity`**, **`precip`**, and **`snow`**. These fields are present in the JSON response and represent the day's humidity percentage, precipitation volume, and snow volume, respectively.
**UI Enhancement**: Modify the existing UI or create a new Composable function to display the newly added weather details. Ensure that the information is presented in an easily readable and understandable format for the user.
**Data Display Logic**: Implement logic in the UI to conditionally display weather details. For example, only show snow volume if the **`snow`** value is greater than 0.

### Steps

**Update the `DayInfo` Data Class**: Add new properties to **`DayInfo`** to match the additional data you plan to display (e.g., **`humidity`**, **`precip`**, **`snow`**). Ensure these new properties correctly correspond to the structure of the JSON response.

kotlin Copy code

```kotlin
data class DayInfo( val datetime: String, val temp: Double, val conditions: String, val humidity: Double, val precip: Double, val snow: Double? // Nullable, as snow might not be present every day )
```

**Modify the UI**: In your Jetpack Compose UI, add new Composables or update existing ones to display the new weather details. Consider creating a detailed view for each day that shows all the relevant weather information.
Example:

kotlin Copy code

```kotlin
@Composable fun WeatherDetail(dayInfo: DayInfo) { Column(modifier = Modifier.padding(8.dp)) { Text(text = "Date: ${dayInfo.datetime}") Text(text = "Temp: ${dayInfo.temp}°C") Text(text = "Humidity: ${dayInfo.humidity}%") Text(text = "Precipitation: ${dayInfo.precip}mm") dayInfo.snow?.let { Text(text = "Snow: ${it}cm") } } }
```

**Implement Conditional Logic**: Ensure that your UI gracefully handles optional or variable data, such as showing snow information only when applicable.
**Testing**: Run your application and navigate to the screen where the detailed weather information is displayed. Verify that all the new data fields are correctly parsed and shown.

Tips

Familiarize yourself with the JSON response structure to understand how the data is organized and how it can be accessed in your Kotlin code.
Use Jetpack Compose's layout and styling capabilities to make the detailed weather information clear and visually appealing.
Consider user experience—avoid overwhelming users with too much data at once. Find a balance between detail and simplicity.

Submission

After implementing the enhancements, test your application thoroughly to ensure that the new weather details are displayed correctly and without errors. Share your updated code through a pull request or as specified by your instructor, including screenshots of the updated UI and a brief description of your changes.

---

This assignment encourages students to dig deeper into the data provided by the API and find meaningful ways to present it, enhancing their understanding of data parsing, Kotlin data classes, and UI development with Jetpack Compose.