Report on the Meteorological Web Application - Exus - Kamil Szczepanik

For this assignment, I used the **FastAPI** framework, integrating with a full-stack project template from GitHub. The goal was to build a solution for streaming, managing, and forecasting weather data from various sources, including IoT sensors and user inputs via a frontend. Here's a summary of the approach and implementation.

Choice of Technology:

I selected **FastAPI** as the core backend framework for several reasons:

1. **Real-Time Data Handling**: FastAPI natively supports asynchronous programming, which is essential for efficient WebSocket connections. This makes it well-suited for handling continuous data streams, such as those from IoT sensors.
2. **High-Performance**: FastAPI is known for being one of the fastest Python frameworks available, making it perfect for the low-latency needs of this project.
3. **Type Hints & Validation**: Python's type hints, combined with FastAPI's automatic request validation, simplified the development process by making API endpoints clearer and catching potential bugs early.
4. **Automatic Documentation**: FastAPI's auto-generated Swagger and ReDoc API documentation greatly sped up development, particularly in ensuring that all API endpoints were clear and well-documented.

Development Workflow:

To expedite development, I leveraged the **full-stack-fastapi-template**. This template provides a robust setup that includes Dockerized services, a PostgreSQL database, and a full FastAPI and React integration. This saved time that would have otherwise been spent on initial project setup, allowing me to focus more on the core functionality.

Implementation Overview:

In just 2 days, I was able to implement several key features:

1. **Current Weather and Time**: A feature displaying real-time weather data for a city.
2. **Forecast for Future Days**: Users can add, edit, or delete forecasts for upcoming days for specific cities, giving them control over the forecasted data for multiple future days.
3. **Weather History**: A view of a city's weather history with detailed information (temperature, humidity, and wind).

Database Design:

The database is structured around three key models:

- **Meteorological Stations**: id, name latitude, longitude, date of installation
- **Weather Forecast**: id, date, high and low temperature, wind, humidity, city_id, user_id
- **Weather History**: id, city_id, humidity, wind, temperature, date
- **Sensors**: id, name, city, category, measurement, unit, date

Each IoT device (sensor) should send JSON data. The data points are stored in the database, ready to be streamed to the frontend via WebSockets or REST.

WebSocket Integration:

Although time constraints limited the full WebSocket integration, I have started implementing WebSocket endpoints that will allow users to subscribe to updates from specific cities and receive real-time weather data once the sensors begin sending information.

While I did not yet connect the IoT sensor simulation to the WebSocket streams, the mechanism is in place to broadcast data to all connected clients whenever new data is added to the database. This will allow the frontend to display updated weather data in real time.

Future Improvements:

Given more time, I would:

- **Complete WebSocket Integration**: Implement WebSocket connections to stream real-time updates for current weather and weather history directly to the frontend.
- **Mock IoT Sensors**: Create a script to simulate the real-time behavior of IoT sensors, sending measurements every 10 seconds and storing them in the database.
- **Testing and Code Quality**: Add more unit and integration tests to ensure robustness. Additionally, I'd improve code quality and make the frontend more responsive across different screen sizes.
- **Responsive Frontend Design:** Implement responsive design to quality of use on different platforms.

In conclusion, I am quite satisfied with the solution, especially given the short timeframe. The project foundation is solid, and with a bit more time, I could have fully implemented the remaining features. This project provided valuable experience with FastAPI(since this is the first time that I use it), and I believe the current structure is a strong base for building a scalable, real-time meteorological data platform.