Course: PA1484

Date: 2025-11-21

Place: Blekinge Tekniska Högskola (campus)

Participants:

- Hampus Ek
- Kaiser Ahmad Samim
- Max Bacharach
- My Lundblad
- Robert Schrewelius

The Project Plan (Part 1 + Part 2)

- "Weather Data Visualization and Interaction on ESP32"

Overview

The purpose of this project is to plan for the development of a system that can fetch weather data from SMHI's API and display it on an ESP32 microcontroller. The system should allow users to interact with the data, navigate with the touch screen, and configure API call settings.

Refined requirements

The group discussed and agreed upon the following refined requirements for the project:

1. The ESP32 must fetch weather data from **SMHI's API**.
2. The system must display:
   a. Seven-day forecast
   b. Simple line graph showing the last 30 days of weather development
3. The system must support **touchscreen navigation** including the function to swipe between screens.
4. Users must be able to select:
   a. City
   b. Weather parameter (e.g., temperature, precipitation, wind)
   c. Forecast options
5. The code must follow best practices:
   a. Modular structure (functions in .hpp files, minimal logic in main.cpp)
   b. Use of templates where possible for reusability
   c. Flexible design to avoid hardcoding solutions
6. API requests must be limited:
   a. Auto-refresh every 5 minutes
   b. Refresh on start-up
   c. Optional manual refresh button if time allows
7. Include configuration system for user settings (city, parameters).

ID	Requirement	Priority
FR1	When the system starts, the ESP32 shall fetch current weather data from the SMHI API (http://opendata.smhi.se/apidocs/).	High
FR2	When the user selects the 7-day forecast view, the system shall display temperature and weather icons for the next seven days at 12:00 each day.	Medium

FR3	When the user opens the history view, the system shall display a simple line graph showing the last 30 days of average temperature data.	Medium
FR4	When the user slides a finger on the touchscreen, the system shall navigate between available screens (e.g. forecast - history).	Medium
FR5	When the user opens the setting menu, the system shall allow selection of the city, weather parameter (temperature, precipitation, wind), and forecast options.	Medium
FR6	When the user updates preferences, the system shall store and apply those configuration settings on subsequent start-ups.	Medium
FR7	When data refresh is triggered, the system shall update API data automatically every five minutes and once on start-up; a manual refresh button shall be available if time permits.	Medium
FR8	When implemented, the system's code shall follow best practices: modular structure (functions in .hpp files, minimal logic in main.cpp) and temples for reusability.	High
FR9	When the system configuration is saved, the design shall avoid hard-coded values to maintain flexibility and ease of future adaptation.	High

Tabel 1.1 - Functional Requirements for the Weather Data Visualization System

ID	Requirement	Priority
NF1	The code shall follow a suitable coding standard.	High
NF2	The code shall include inline comments and markdown documentation in the repository.	High
NF3	The system shall comply with SMHI API license terms.	High
NF4	The system shall respond to user input within three seconds.	Medium

Tabel 1.2 - Non-Functional Requirements

Risk management

The group identified and agreed upon the following risks and mitigation strategies:

- **Risk 1:** Hardcoded code increases workload in the long run.
  - **Solution:** Keep code modular and flexible from the start.
  - Likelihood (1-3): 1
  - Severity (1-3): 2
- **Risk 2:** Bugs missed during development.
  - **Solution:** Perform small, incremental tests frequently before adding new features.
  - Likelihood (1-3): 2
  - Severity (1-3): 2
- **Risk 3:** ESP32 overload due to too many API calls.
  - **Solution:** Limit auto-refresh to 5-minute intervals and keep data handling lightweight.
  - Likelihood (1-3): 1
  - Severity (1-3): 3
- **Risk 4:** Health issues or absence of group members.
  - **Solution:** If a member experiences health problems and **communicates** this to the group in advance, the workload will be temporarily redistributed among the remaining members. The affected member will rejoin at full capacity once able. This ensures project continuity while respecting personal well-being.
  - Likelihood (1-3): 2
  - Severity (1-3): 2
  - If a member **does not communicate** their health-related unexpected absence, they will receive a strike. After two strikes, the issue will be brought up by a supervisor. Escalation may involve reassignment of tasks, adjustment of grading, or, in severe cases, removal from the project team – depending on what the supervisor deems fit.
  - Likelihood (1-3): 1
  - Severity (1-3): 2
  - If a member fails to attend scheduled meetings or work sessions without any communication, this results in immediate escalation after **one** strike as such behavior is unacceptable and undermines trust in the group and deliverables. Escalation may involve reassignment of tasks, adjustment of grading, or, in severe cases, removal from the project team – depending on what the supervisor deems fit.
  - Likelihood (1-3): 1
  - Severity (1-3): 2

Rationale:

- Health issues are unavoidable but manageable with communication.
- Unannounced absences are preventable and damage teamwork.
- Clear rules and a strike-based escalation system ensure fairness, accountability, and project stability.

Memo – Communication with customer

2025-09-26: Kick-off Meeting
Participants: Team 1 + Client
The group noted the following instructions received during the kickoff:

- The system must show today's forecast, 7-days, and a 30-day line graph.
- Code must remain flexible and modular.
- Auto-refresh should be limited to once every 5 minutes.
- A manual refresh button is desirable if time permits.
- Users should be able to navigate via touchscreen gestures.
- System configurations must be included (city, weather parameters).
- The client may request changes along the way; flexibility is crucial.

2025-10-13: Workshop 2, Second Meeting
Participants: Team 1 + Client
The following points were discussed and agreed upon:

- The boot-up screen must display the program version and group number for three seconds.
- Forecast data should also include weather condition symbols (sun, rain, snow etc.)

Contribution statement

- **Max Bacharach** – Led the meeting, contributed to requirements refinement, and role assignments. Took the meeting notes and documented decisions.
- **My Lundblad** – Edited and structured the meeting notes.
- **Kaiser Ahmad Samim** – Contributed to discussion on risk management and hardware responsibilities.
- **Hampus Ek** – Proposed modular coding practices and GitHub management strategy.
- **Robert Schrewelius** – Supported refined requirements and GitHub management strategy.

Work Breakdown Structure (WBS)

Outlining main responsibilities and task areas for requirements and defining the different development processes.

1. **Initiation**
   1.1 Toolchain Set-up – Download VS Code & platform IO. To be able to run & code.
   1.2 Project Plan – General development planning.

2. **Design**
   2.1 System Architecture & Design – System and Data functionality
      2.1.1 ESP32 System Architecture & Boot-up – Adds adaption to ESP32 System
      2.1.2 Screen Navigation Design – Design to be able to navigate between screens
      2.1.3 Data Model for Forecast & History – Formatting of weather data

   2.2 UI/UX Design – Tile design & visualization.
      2.2.1 Render Version + Group Number – First screen showing group names & version number.
      2.2.2 Forecast Screen – Display weather for upcoming week.
      2.2.3 History Screen – Display weather for monthly history.
      2.2.4 User Interaction Tests (sliders, swiping) – Interactions with sliders & menus.

3. **Execution**
   3.1 API Layer Implementation – API implementation
      3.1.1 SMHI API Base – Class for API functionality
      3.1.2 Data Fetching Methods – Fetching data from API

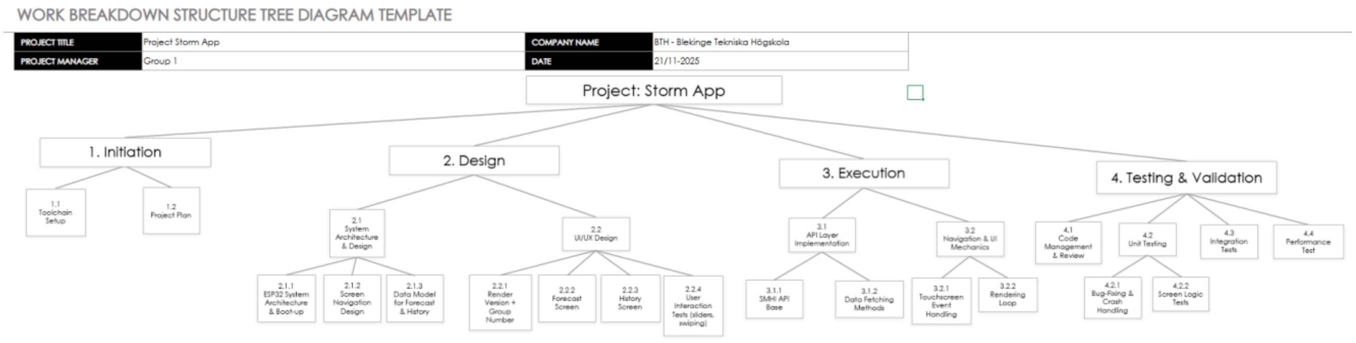
   3.2 Navigation & UI Mechanics – Handles the tiles functionality
      3.2.1 Touchscreen Event Handling – Interaction with touchscreen
      3.2.2 Rendering Loop - Updating tiles for UI

4. **Testing & Validation**
   4.1 Code Management & Review - Reviewing of the code before merge
   4.2 Unit Testing - Test of the code before integration
      4.2.1 Bug-Fixing & Crash Handling – Fixing the bugs that comes up from testing
      4.2.2 Screen Logic Tests - Test of screen functions
   4.3 Integration Tests - Test after adding new functionalities in merged
   4.4 Performance Test – Testing stability and performance

Note!

Time estimation and progress in % in the GANTT chart.

Figure 1 – Diagram of WBS



WORK BREAKDOWN STRUCTURE TREE DIAGRAM TEMPLATE

PROJECT TITLE	Project Storm App	COMPANY NAME	BTH - Blekinge Tekniska Högskola
PROJECT MANAGER	Group 1	DATE	21/11-2025

Project: Storm App
1. Initiation
1.1 Toolchain Setup
1.2 Project Plan
2. Design
2.1 System Architecture & Design
2.1.1 ESP32 System Architecture & Boot-up
2.1.2 Screen Navigation Design
2.1.3 Data Model for Forecast & History
2.2 UI/UX Design
2.2.1 Render Version + Group Number
2.2.2 Forecast Screen
2.2.3 History Screen
2.2.4 User Interaction Tests (sliders, swiping)
3. Execution
3.1 API Layer Implementation
3.1.1 SMHI API Base
3.1.2 Data Fetching Methods
3.2 Navigation & UI Mechanics
3.2.1 Touchscreen Event Handling
3.2.2 Rendering Loop
4. Testing & Validation
4.1 Code Management & Review
4.2 Unit Testing
4.2.1 Bug-Fixing & Crash Handling
4.2.2 Screen Logic Tests
4.3 Integration Tests
4.4 Performance Test

Structure & Roles

The group assigned the following roles and responsibilities:

- **Chairman – Max Bacharach** Keeps the group on track with deadlines and otherwise, ensures focus on deliverables, and works with the secretary on documentation.
- **Secretary – My Lundblad** Records formal meeting notes and project decisions. Ensures that they reach all members of the group. Assists the chairman with administrative tasks such as keeping track of deadlines, agendas and documentation requirements.
- **ESP Handler & Code Tester – Kaiser Ahmad Samim** Responsible for ESP32 hardware availability in labs, performs stress testing of the code, and reports results.
- **Code Managers & GitHub Managers – Hampus Ek and Robert Schrewelius** Maintain code quality and consistency. Ensure the GitHub project is updated. Each validates the other's pull/merge actions.
- **Coders – All members** Implement functionality according to requirements, in collaboration with managers and tester.

Note!
All members are responsible and will participate, but the roles are assigned in order to ensure that at least one person has the final responsibility to make sure that specific sections run smoothly.