9530

St. MOTHER THERESAENGINEERING COLLEGE

COMPUTER SCIENCE ENGINEERING

NM-ID:

B3914712467804994 2449EA6BFCBF718

REG NO: 953023104138

DATE: 22-09-2025

Completed the project named as

Phase 3

FRONT END TECHNOLOGY

LIVE WEATHER DASHBOARD

SUBMITTED BY,

P. YARWIN YAHAV

9361906857

Phase 3: MVP Implementation

Overview

This Phase 3 document focuses solely on the Minimum Viable Product (MVP) implementation activities. It intentionally excludes project-identifying details already present in earlier phases. The included cover/header page from the provided reference PDF will be used as the document's front page.

Scope & Objectives

• Implement core MVP features based on requirements and design from previous phases. • Provide a clear implementation plan, folder structure, key components, sample code snippets, testing and deployment guidance.

Implementation Plan

- Bootstrap project and install required tooling.
- Create core components and modules required for the MVP.
- Implement service modules to interact with external APIs and normalize responses.
- Implement state management and persistence for user-facing features.
- Add loading/error states and basic responsive styles.
- Optional: create a lightweight backend proxy to protect API credentials and enable caching.
- Test thoroughly (manual + automated) and prepare for deployment.

Suggested Tech Stack

Frontend: Modern JS framework (React/Vue) with component-based structure. Styling: Utility CSS (e.g., Tailwind) or scoped CSS modules. APIs: External third-party APIs for data; a small server (Node/Express) is optional for key protection and caching. Testing: Unit tests (Jest), integration tests (React Testing Library), and manual acceptance tests.

Recommended Folder Structure

src/ ■■ components/ ■■ context/ or store/ ■■ services/ ■■ styles/ ■■ App.jsx / main.js

Key Components & Responsibilities

Search/Input component: triggers data retrieval. Display component(s): render main data and short-term forecasts. History component: persist and present recent searches. Service module: single place to call external APIs and return normalized data. Context/store: central state management and persistence.

Sample Service (concept)

```
/* Example service (pseudo-code) */ async function getData(query, apiKey) { const res =
await
fetch(`https://api.example.com/data?q=${encodeURIComponent(query)}&appid;=${apiKey}`); if
(!res.ok) throw new Error('Fetch failed'); return res.json(); }
```

State Management (concept)

```
/* Concept: central provider that exposes currentData, history and actions */ const
[current, setCurrent] = useState(null); const [history, setHistory] = useState(() =>
JSON.parse(localStorage.getItem('history') || '[]'));
```

Testing Approach

Manual tests: functional flows (valid/invalid inputs), network failures, mobile responsiveness. Automated: unit tests for service functions and components. Acceptance criteria: core search returns data, display contains expected fields, history persists and is clickable.

Deployment

Build and deploy as a client application (Vercel / Netlify / GitHub Pages) or as a client + small server (Render / Railway). If a server is used, ensure environment variables are used for API keys and that endpoints are rate-limited/cached.

Timeline & Deliverables

Week 1: Setup and core components. Week 2: Service integration and state management. Week 3: Styling, responsiveness, testing. Week 4: Final testing and deployment.

Acceptance Criteria

1. Searching returns current data and short-term forecast. 2. Display shows name/date/condition/temp/humidity/wind (as applicable). 3. History is persisted and clickable. 4. UI is responsive.

References

This document uses the first page of the provided reference PDF as the front cover/header. For detailed requirements and design, refer to Phase 1 and Phase 2 documents.