WeatherWatch

Let the weather set the scene.

Project document

Full-stack group 4 members

Sadie Jones, Viktoria Lauri, Safa Zia, Alberta Uwadiae, Ana (Kiki) Villarreal,Jessica James

Roadmap of the Project

Introduction **3**
Aims and objectives of the project 3
Background **3**
Specifications and design **4**
Technical requirements 4
Tech stack 4
Non-technical requirements 4
System design and architecture **5**
Key sections: 6
Implementation and execution **6**
Development methodology 6
CI/CD and version control 6
Team roles and collaboration 6
Development flow 7
Key challenges and solutions 7
Testing and evaluation **7**
Testing strategy 7
Functional and user testing 7
System limitations 8
Future improvements 8
Conclusion **8**
Project objectives 9
Final results 9
Alignment with objectives 9
Performance of the final product 9
Conclusion 9
Screenshots 10

Introduction

Aims and objectives of the project

This project aims to develop a web application that recommends movies based on the user's local weather. WeatherWatch is built on the idea that different weather conditions can influence mood and the type of entertainment we seek. By connecting real-time weather data with curated film suggestions, the app offers a personalised movie experience that matches the atmosphere outside of the user's screen.

The name *WeatherWatch* combines two core concepts: "weather," which drives the app's functionality, and "watch," referring both to viewing films and monitoring weather conditions. This simple name captures the app's essence, suggesting films that feel just right for the moment, whether it's a cosy drama for a rainy day or a light-hearted comedy on a sunny afternoon.

Our goal is to deliver a full-stack solution using APIs to fetch weather and movie data, with features including genre filtering by decade, a shuffle function, and the ability to update user details. The app will be responsive, accessible, and easy to use.

Beyond the product itself, this project is also a learning opportunity for us as a team to collaborate effectively, apply what we've learned in front-end and back-end development, and follow a realistic, iterative approach to delivering a working solution

---

Background

The concept behind WeatherWatch is rooted in the way weather subtly shapes our choices, especially when it comes to comfort and entertainment. Rather than selecting a film at random, the app introduces a layer of mood-based logic by linking movie suggestions to local weather conditions. This creates a more curated and relatable experience for users.

To make this possible, we integrated two key external data sources: OpenWeatherMap, which provides live weather information, and TMDb (The Movie Database), which supplies movie details. These datasets are processed and matched on the backend to return recommendations that reflect current weather types, such as clear, cloudy, rainy, or snowy.

From early on, we defined a few core rules for how the system should behave:

- Each weather condition maps to one or more movie genres (e.g., rainy → drama or romance, sunny → comedy or adventure, snowy → fantasy or family movies, etc.).
- The user must be able to request another suggestion within the matched genre if they do not feel like watching or have already seen the first suggestion that comes up.
- Users can narrow down their results by release year ranges, giving them control over the era or aesthetic they’re in the mood for.

This logic helped us focus our development and shape how the API interactions and user experience would flow.

The app is designed for users of all ages. Age verification is essential to ensure that movie recommendations are appropriate for the user’s age group, especially when considering films with age ratings. To handle this, the app implements a simple verification process that prompts users to confirm their age upon registration

---

Specifications and design

Technical requirements

System functionality and constraints

- The application must support at least **100 concurrent users** without performance degradation.
- User authentication must be secure, with **encrypted credentials** and **session management** using JWT.
- The system must respond to user actions within **1–2 seconds** under normal load.
- All features must work on **modern desktop and mobile browsers** (Chrome, Firefox, Safari, Edge).
- The UI must meet basic **accessibility standards** (keyboard navigation, readable contrast, screen reader compatibility).
- The app must gracefully handle errors from external APIs (e.g., rate limits, failed weather fetch).

- The database must persist user data, including credentials, age, and preferences.
- The system must prevent underage users from receiving inappropriate film suggestions based on age restrictions.

Tech stack

Frontend
- **React** – for building dynamic, component-based user interfaces
- **Axios** – for making HTTP requests to the backend and external APIs
- **React Router** – for client-side routing
- **JWT-decode** – to read user information from JWT tokens on the client side
- **React Testing Library & Jest** – for writing and running unit tests on React components
- **Babel** – to compile modern JavaScript for broader browser compatibility

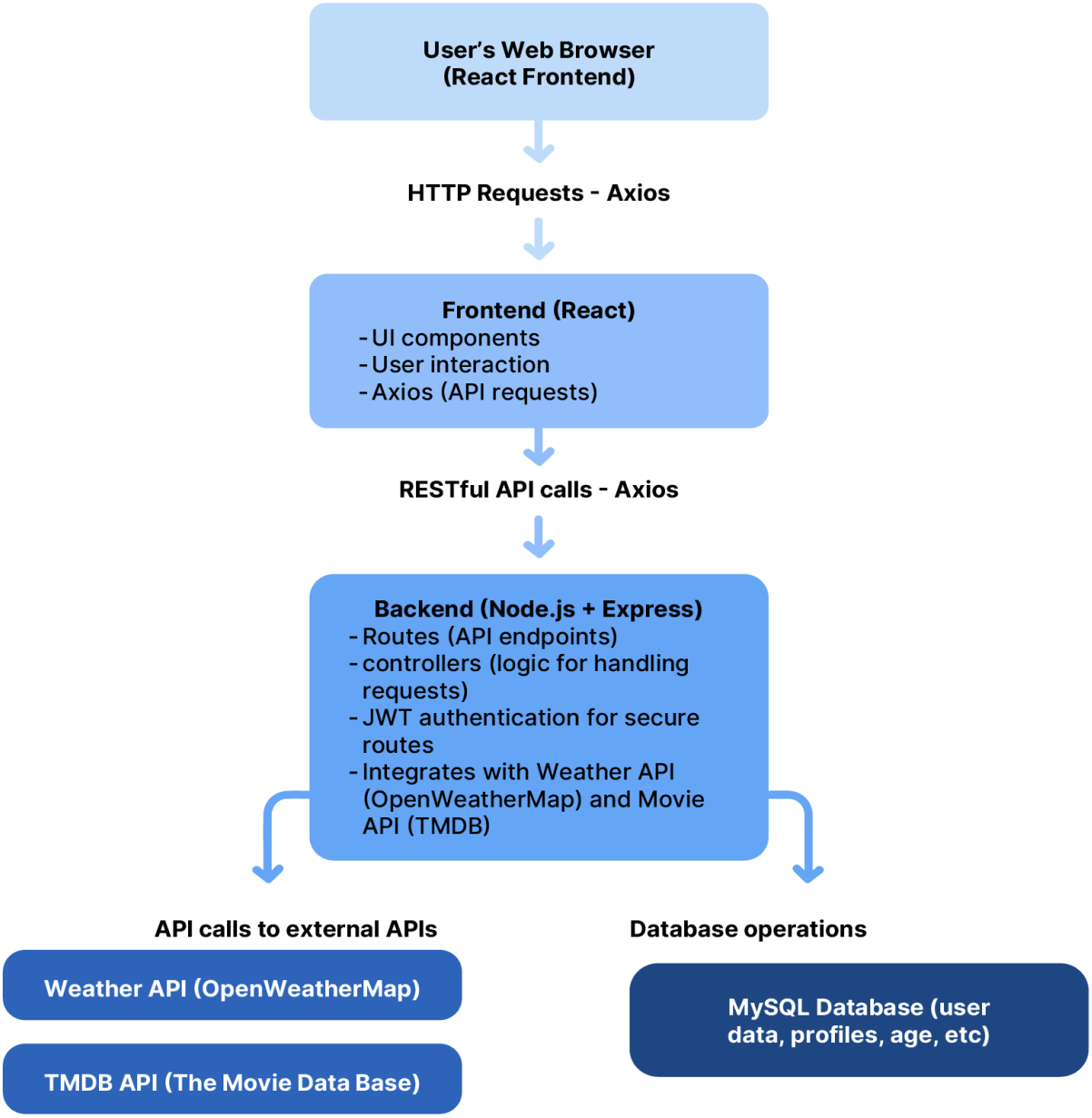
Backend
- **Node.js & Express** – to handle server-side logic, routing, and API integration
- **MySQL2** – to interact with the MySQL database efficiently
- **jsonwebtoken** – to generate and verify JWT tokens for user authentication
- **cors** – to enable secure cross-origin resource sharing between frontend and backend
- **supertest** – to test HTTP endpoints and backend behavior

Utilities and tools
- **dotenv** – to manage environment variables securely
- **Nodemon** – for automatic server reloading during development
- **concurrently** – to run backend and frontend servers with a single command
- **VS Code** – code editor
- **Git + GitHub** – version control with branching for features and bug fixes
- **Postman** – for testing API endpoints manually during development

Non-technical requirements

- **User experience and design consistency:**
  The interface should feel intuitive and engaging, with a consistent visual design and layout that aligns with user expectations. Interactive feedback (e.g., loading indicators, error prompts) should guide the user clearly.
- **Personalization:**
  The app should tailor content to the individual by incorporating user profile data and real-time weather, creating a sense of relevance and connection.
- **Maintainability:**
  Code should follow industry best practices with meaningful comments, consistent formatting, and modular structure to support easy future development and debugging.
- **Team collaboration:**
  Development should be version-controlled and branch-based to support collaborative workflows, allowing clear integration of features and timely resolution of bugs.
- **Scalability of development process:**
  Reusable components and clear separation of concerns (frontend/backend/API services) should enable smooth addition of future features and team scaling.
- **Documentation and handover:**
  The system should be documented well enough that new developers can onboard quickly and understand project structure, dependencies, and deployment steps.

System design and architecture



Key sections:

- **User's web browser (React frontend)**: This is where users interact with the app. It makes requests to the backend to fetch weather data and movie recommendations.
- **Frontend (React) [/client]**: Handles the user interface (UI) and client-side logic. It uses Axios to make requests to the backend for weather, movie, and user data; React Router for navigation; and a context component to hold and manage shared/global state.
- **Backend (Node.js + Express) [/server]**:
  - Manages API endpoints for handling data requests (registration, login, profile page, and movie/weather queries).

- Ensures secure authentication with **JWT**.
- Integrates with external APIs (OpenWeatherMap and TMDb) to retrieve weather data and movie recommendations.
- Uses dotenv for managing environment variables

- **Weather API (OpenWeatherMap)**: This external API provides weather data, including weather conditions like clear, rainy, or cloudy.
- **Movie API (TMDb)**: This external API is used to fetch movie details such as genres, ratings, overviews, posters, certification details, and release years based on the weather.
- **MySQL database**: Stores user data, including credentials, profile information, and age for age-appropriate recommendations. The schema also includes tables for future features, including a watchlist and weather logs

---

Implementation and execution

Development methodology

The project followed an Agile-inspired workflow, using iterative development cycles and regular team check-ins (via scheduled virtual meetings and Slack) to discuss blockers, review progress, and plan the next sprint. Work was organized through shared excel files to track tasks.

All contributions followed a branching strategy (feature/bugfix branches), with pull requests reviewed by team members to ensure code quality, catch bugs early, and maintain consistency across the codebase. Paired programming and small group sessions were used throughout, especially during complex feature development and bug resolution.

CI/CD and version control

We used GitHub for version control and collaborative development. Our CI workflow also included the use of **.env** files to securely manage environment variables.

Although fully automated deployment was out of scope for the project timeline, we laid the foundation for a CI/CD pipeline, allowing easy future integration of test coverage checks and automated deployment workflows.

Team roles and collaboration

- **Backend Development** (Viktoria, Sadie, Safa, Jessica): API routes, JWT-based authentication, database integration, external API connections.
- **Frontend Development** (Viktoria, Sadie, Alberta, Safa): UI components, React routing, logic for filtering and movie selection.
- **Styling and UX** (Sadie,Ana (Kiki)): Wireframes, CSS styling, layout design, accessibility tweaks.
- **Testing** (Viktoria, Sadie): Unit and integration testing across both frontend and backend.

Splitting into sub-teams allowed us to make good progress in parallel streams while keeping communication focused and efficient. Regular merge reviews ensured cohesive progress.

Development flow

We adopted a **backend-first approach**: the API and database logic were built and tested independently, allowing frontend development to proceed with reliable endpoints and mock data where necessary. This approach enabled us to identify and fix integration bugs early.

Key challenges and solutions

- **Asynchronous API handling:** Ensured robust error handling and fallbacks for rate limits or failed responses.
- **API data transformation:** Created utility functions to map TMDb and OpenWeatherMap data into frontend-friendly formats.
- **Alignment of backend with frontend needs:** Regular syncs between teams helped resolve mismatches in data structures or expected responses.
- **User experience adjustments:** Late-stage refinements were made to improve clarity of prompts and layout responsiveness based on feedback

---

Testing and evaluation

Testing strategy

We applied a combination of **manual**, **automated**, and **peer testing** approaches to evaluate the system's functionality and user experience:

- **Manual testing**: Conducted across major browsers (Chrome, Edge, Safari) to verify responsiveness and consistent behavior.
- **Backend testing**: API endpoints tested using **Postman** to validate route logic, authentication, and error handling.
- **UI testing**: Dev tools were used to inspect rendering, responsiveness, accessibility, and performance bottlenecks.

Functional and user testing

- Core features (user authentication, weather-based movie suggestion and filtering functionality) were tested using real and mocked data.
- Edge cases were considered and handled.
- A peer testing and feedback loop was used throughout the process, with users providing insights into usability, navigation, and clarity of messages/prompts.

System limitations

Despite meeting core requirements, the current system has the following known limitations:

- **Basic recommendation Logic**: Movie suggestions are currently based on a fixed mapping of weather types to genres. There is no personalized or machine-learned recommendation logic implemented.
- **Session-based watchlist**: The user watchlist is not yet built or persisted in the database.
- **Limited offline support**: The app does not function offline, as it relies heavily on external APIs for weather and movie data.
- **Minimal accessibility testing automation**: While the app was designed to be accessible (keyboard navigation, readable contrast), automated accessibility testing tools like Lighthouse or Axe were not used due to time constraints.

Future improvements

To further enhance functionality, user experience, and system robustness, the following improvements are proposed:

1. Watchlist and favourites (persistent storage)

- Enable users to save favourite movies to a persistent watchlist tied to their account, allowing access across sessions and devices.
- Include a "watched" toggle or history to avoid repetition in recommendations.

2. Enhanced filtering options

- Add more sophisticated filters (e.g. current user mood, popularity, movie rating) so users can tailor recommendations further.

3. UI and UX enhancements

- Introduce a dark mode toggle for improved accessibility and user preference.
- Optimize mobile responsiveness with refined layouts and touch-friendly controls.
- Add micro-interactions and transitions to enhance engagement and polish.

4. Smarter recommendation engine

- Move beyond static weather-genre mapping by incorporating machine learning models to analyze user behavior and preferences over time.
- Personalize suggestions using collaborative filtering or content-based filtering methods.

5. Accessibility and internationalization

- Integrate automated accessibility testing tools (e.g., Axe, Lighthouse) into CI pipeline.
- Prepare for multi-language support with i18n libraries, enabling a wider user base.

6. CI/CD integration and testing automation

- Implement CI/CD pipelines using GitHub Actions to automate testing, linting, and deployment.
- Expand test coverage with React Testing Library, Jest, and Supertest, and incorporate code coverage reports to monitor quality.

7. Caching and performance optimization

- Cache frequent API responses (e.g. popular movies or weather conditions) to reduce load times and API request volume.
- Introduce lazy loading and code splitting for better performance on slower networks.

Conclusion

WeatherWatch meets the majority of the objectives set out at the project's inception. Our main aim was to create a web application that recommends movies based on real-time weather data. We have achieved this through seamless API integration, user authentication, and a responsive, accessible interface.

Project objectives

1. Provide movie suggestions based on the current weather conditions in the user's location.
2. Secure user authentication and personalized recommendations tailored to user preferences, such as age restrictions and weather types.
3. A user interface that is both responsive across devices and meets basic accessibility standards.
4. The app must support at least 100 concurrent users without performance degradation and respond within 1-2 seconds under normal load.
5. Foundation for future enhancements, including a watchlist, more sophisticated recommendation logic, and integration of additional filters.

Final results

1. We successfully integrated OpenWeatherMap and TMDb APIs to deliver real-time movie suggestions based on weather conditions. The weather-to-genre mapping was implemented as initially planned, with suggestions adjusted for user age restrictions.

2. A secure JWT-based authentication system was implemented, ensuring user privacy and access to personalized recommendations based on weather and age data. The app currently filters movies by genre, release year, and age-appropriate content.
3. WeatherWatch meets accessibility standards, with features such as readable contrast. The UI adapts well to different sizes of desktop screens, ensuring a broad user base can interact with the application without barriers.
4. The application performs efficiently under normal load, with minimal latency (1-2 seconds response time).
5. The app's modular architecture and clean codebase provide a good foundation for future features such as a persistent watchlist, advanced filtering options, and AI-based recommendations. The system's database schema is designed to support these future functionalities, and the codebase is well-structured for easy updates.

Alignment with objectives

WeatherWatch aligns closely with the objectives set at the beginning of the project. The system meets the functional requirements of real-time weather-based recommendations and secure user authentication. While the current recommendation logic is static, we have laid the groundwork for implementing smarter, personalized suggestions in the future.

Additionally, the app fulfills the non-functional objectives related to performance, scalability, and accessibility. It performs within the expected load range, providing a responsive experience even as we continue to add features. The app's design is accessible to a wide range of users, ensuring that it is usable by people of different abilities.

Performance of the final product

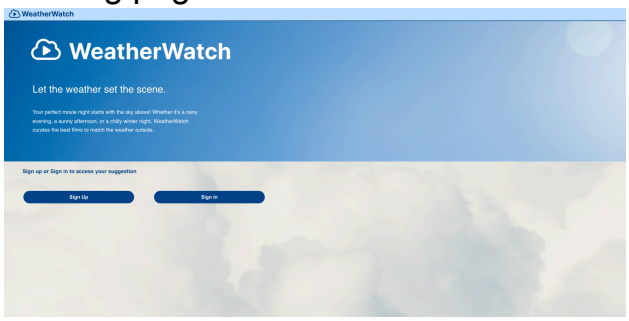
WeatherWatch consistently responds to user requests by fetching weather data and delivering movie recommendations in well under two seconds. Error rates remain low thanks to robust fallback handling for API failures. Resource usage stays within acceptable bounds on our development server. The modular architecture ensures that, with minimal adjustments, the system can maintain this level of performance as traffic grows.

Conclusion

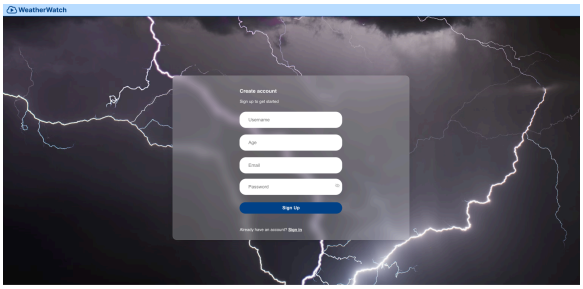
This project has been a rewarding collaborative effort that allowed our team to learn and apply full-stack development principles in a real-world context. We each brought different strengths and worked cohesively to deliver a functioning, user-friendly application. Along the way, we deepened our technical skills, improved our communication and problem-solving as a group, and gained hands-on experience in project planning, version control, and agile development. WeatherWatch is not only a reflection of what we've learned, but also a creation which we're proud to have built together.

Screenshots of core features

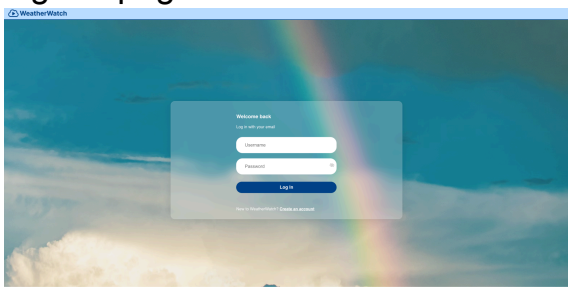
Landing page



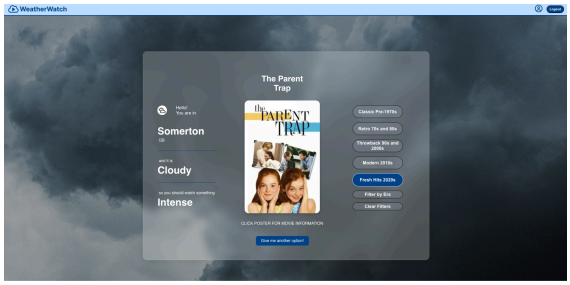
Sign-up page



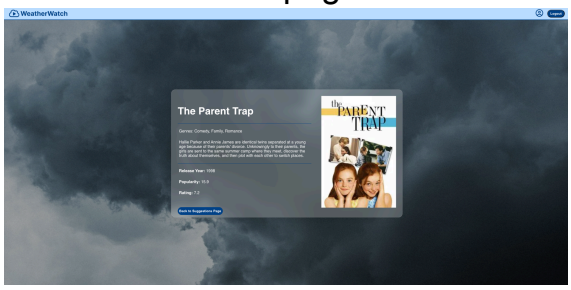
Sign-in page



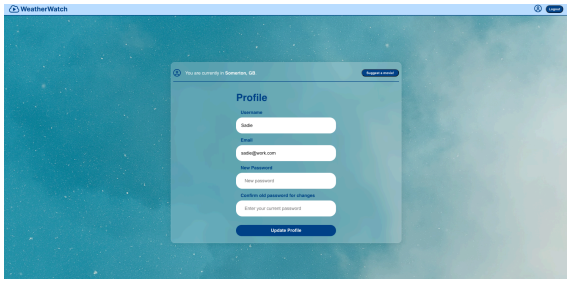
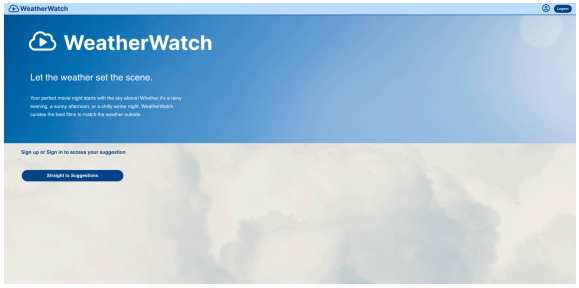
Movie recommendation page



Movie information page



Profile page



Landing page, when logged in