WeatherTrack

Applicazioni e Servizi Web

Gian Luca Nediani - 0001075900 {gianluca.nediani@studio.unibo.it}

9 giugno 2024

Indice

Elenco delle figure

1 Introduzione

L'obiettivo del progetto WeatherTrack è realizzare un'applicazione web che permetta agli utenti di interagire con i dati meteo raccolti in tempo reale da sensori IoT.

Tramite un'interfaccia web gli utenti potranno autenticarsi, visualizzare i dati in tempo reale per le posizioni per cui nel sistema sono presenti sensori, oltre che visualizzare dati storici e una mappa che mostra le posizioni disponibili. Inoltre, l'applicazione include un sistema di notifica di anomalie meteo: oltre a poter visualizzare lo storico delle anomalie, gli utenti possono registrare delle anomalie personalizzate per cui venire notificati in tempo reale. Infine è presente una funzionalità di richieste attraverso la quale gli utenti possono inoltrare richieste per nuove posizioni e funzionalità. Le richieste, così come gli account degli utenti, sono gestite dagli amministratori dell'applicazione. Il sistema disporrà dunque di una distinzione di ruoli nel proprio sistema di autenticazione.

Il progetto si pone come obiettivo quello di produrre l'intero sistema a partire dai sensori, che saranno qui simulati, fino ad arrivare al server e l'interfaccia web.

2 Requisiti

L'obiettivo principale del progetto WeatherTrack è quello di sviluppare un sistema distribuito in grado di raccogliere e elaborare dati meteorologici in tempo reale da una rete di sensori IoT e di fornire agli utenti un'interfaccia web-based per visualizzare e analizzare i dati raccolti.

2.1 Requisiti funzionali

- Raccolta, elaborazione e visualizzazione in tempo reale dei dati dai sensori IoT simulati nell'applicazione web.
- Archiviazione dei dati dei sensori in un database per l'analisi storica.
- Funzionalità dell'applicazione web per visualizzare dati meteorologici in tempo reale, statistiche aggregate e rilevamento di anomalie meteorologiche.
- Possibilità di visualizzare una mappa su cui sono evidenziate le posizioni dei sensori, con la possibilità di vedere una preview e di avere accesso con un click ad informazioni più approfondite.
- Possibilità per gli amministratori di gestire gli account utente e le richieste presentate dagli utenti. Tali richieste possono riguardare sia funzionalità aggiuntive che sensori in nuove località.
- Possibilità per gli utenti di gestire il proprio account e di inviare richieste.
- Creazione di anomalie meteorologiche personalizzate visualizzate in tempo reale se rilevate dal sistema, nonché possibilità di visualizzare anomalie passate archiviate nel sistema.
- Autenticazione utente per controllare l'accesso all'applicazione web.

2.2 Requisiti non funzionali

Oltre ai requisiti funzionali, sono presenti dei requisiti non funzionali di cui si dovra tenere conto in fase di design:

- Il sistema deve essere scalabile per garantire un corretto funzionamento anche all'aumentare del volume di: sensori, dati, utenti.
- Il sistema deve essere flessibile e modulare per garantire la facilità nell'aggiunta di feature in quanto uno dei requisiti evidenziati precedentemente è la possibilità per gli utenti di richiedere nuove feature.
- Deve essere realizzata una implementazione di un sensore simulato in grado di produrre dati verosimili in maniera continua ed inviarli al sistema per l'archiviazione e la visualizzazione in tempo reale sull'interfaccia web

- Il sistema deve avere un meccanismo di autenticazione che garantisca la possibilità di avere più ruoli per gli utenti in quanto sono presenti due tipologie di utenti: normali e amministratori.
- Usabilità: il sistema deve essere facile da usare e da navigare per tutti gli utenti, indipendentemente dalle loro abilità tecniche

3 Design

Si è fatto uso di un Design di tipo User Centered con un approccio KISS, al fine di garantire un'esperienza utente intuitiva e priva di frustrazioni. Inoltre, è stato adottato un approccio Mobile First in quanto la maggior parte degli utenti accede alla piattaforma tramite dispositivi mobili e si vuole assicurare una navigazione fluida e una visualizzazione ottimale su schermi di piccole dimensioni.

Il progetto è stato realizzato da un singolo sviluppatore utilizzando la metodologia Agile. Il processo è stato strutturato in sprint, con l'obiettivo di consegnare funzionalità incrementali ad ognuno di essi. Per tenere traccia dello stato di avanzamento dei lavori e gestire le attività, è stato utilizzato Trello, un'applicazione web che consente di creare schede per ogni task, organizzarle in colonne che rappresentano le fasi del processo e spostarle avanti e indietro per mostrare il loro progresso. In questo modo, il singolo sviluppatore è stato in grado di mantenere una visione chiara e aggiornata sullo stato del progetto.

3.1 Design architetturale

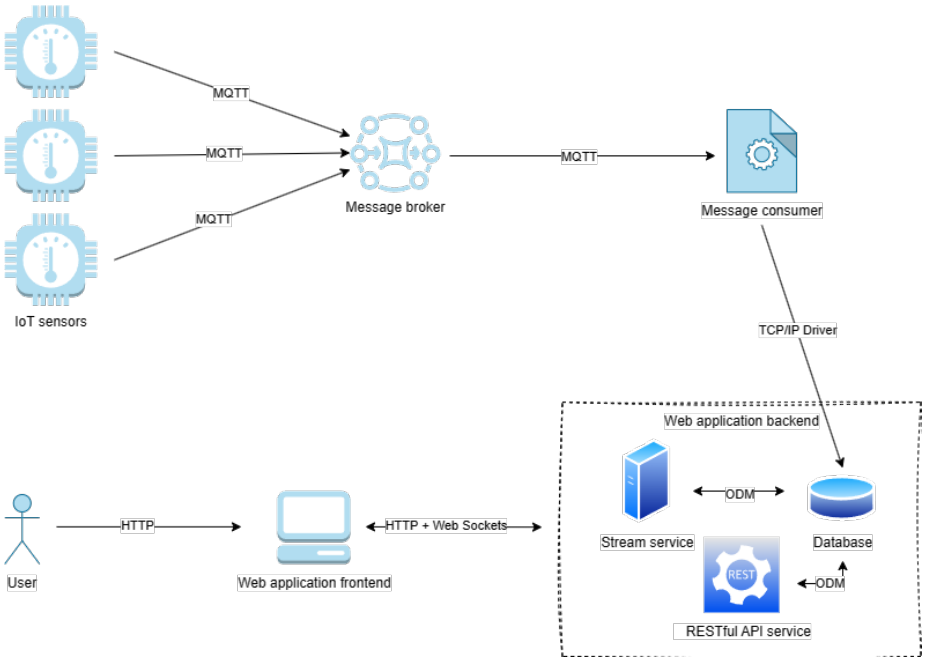

Figura 1: Panoramica del design architetturale

La panoramica dell'architettura di sistema mostrata in figura illustra i componenti di alto livello del sistema WeatherTrack. Ciascuno dei componenti è im-

plementato in un singolo container, consentendo una facile scalabilità, sviluppo e gestione del sistema.

Gli script dei sensori e dei consumer sono responsabili rispettivamente della raccolta e dell'elaborazione dei dati meteorologici. Lo script dei sensori raccoglie i dati meteorologici da dispositivi IoT simulati e li invia al broker di messaggi. Lo script dei consumer riceve i dati dal broker di messaggi, li elabora e li memorizza nel database. Vi è poi broker di messaggi che funge da intermediario tra i sensori e il consumer, consentendo una comunicazione asincrona e disaccoppiando i componenti. Ciò permette al sistema di gestire grandi volumi di dati e fornisce tolleranza ai guasti in caso di errori dei componenti.

L'applicazione web è composta da un frontend e un backend. Il frontend fornisce un'interfaccia utente per visualizzare e analizzare i dati meteorologici raccolti, oltre che accedere alle altre funzionalità del sistema, mentre il backend fornisce un'API RESTful per consentire al frontend di interagire con il sistema e gli stream per la trasmissione di dati in tempo reale.

L'utilizzo dei container per il deployment dei componenti fornisce una soluzione scalabile e portatile per il sistema. Ogni componente viene dispiegato come singolo container, consentendo una facile gestione e scalabilità del sistema. I container sono connessi utilizzando una rete locale.

3.2 Design interfacce

Vengono qui mostrati i mockup delle principali interfacce dell'applicazione web, realizzati in fase di analisi dei requisiti e design del sistema e serviti da riferimento durante lo sviluppo dell'applicazione web.

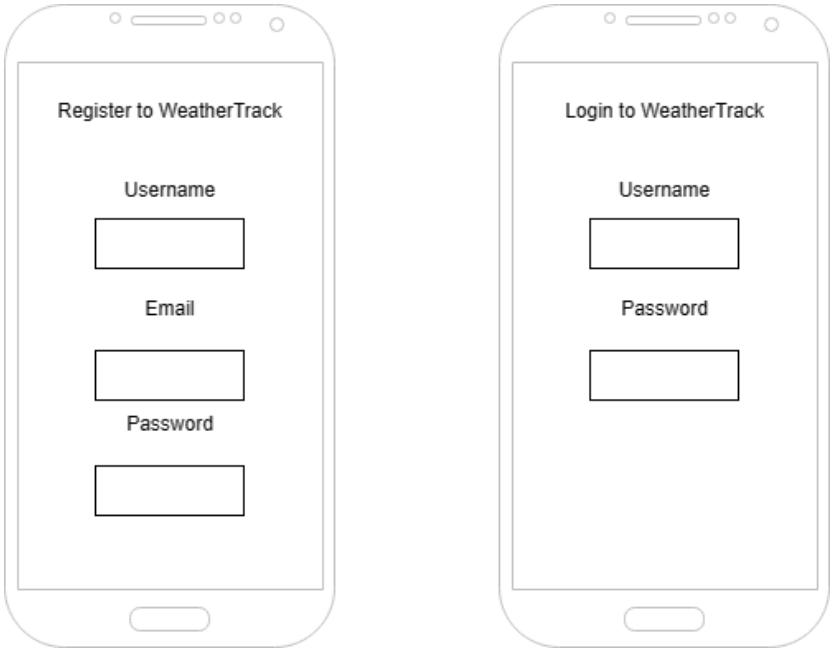

Figura 2: Mockup per l'autenticazione

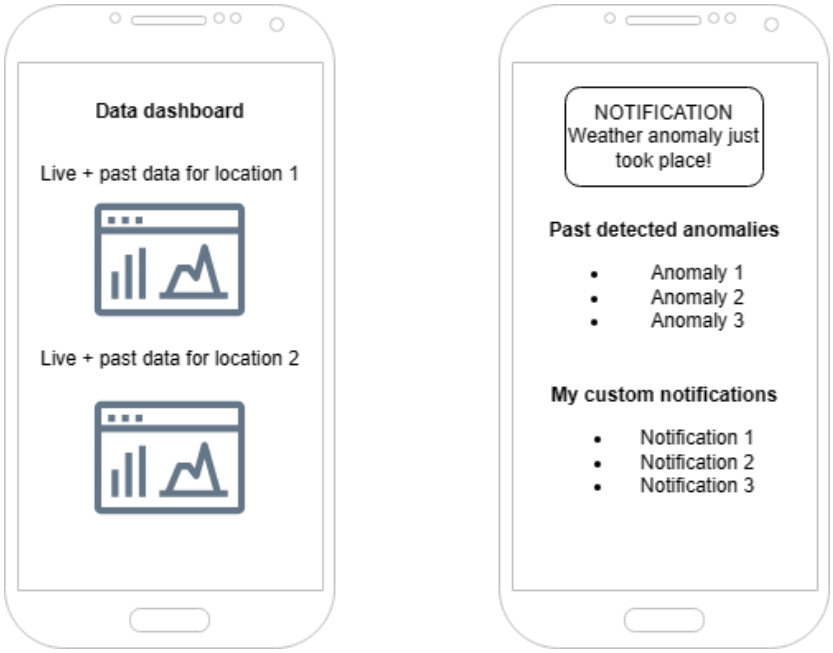

Figura 3: Mockup vista dati e anomalie

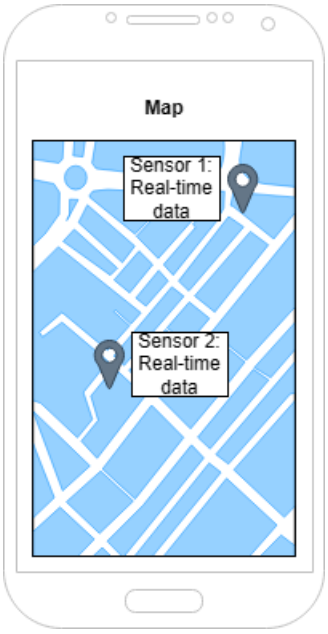

Figura 4: Mockup mappa sensori

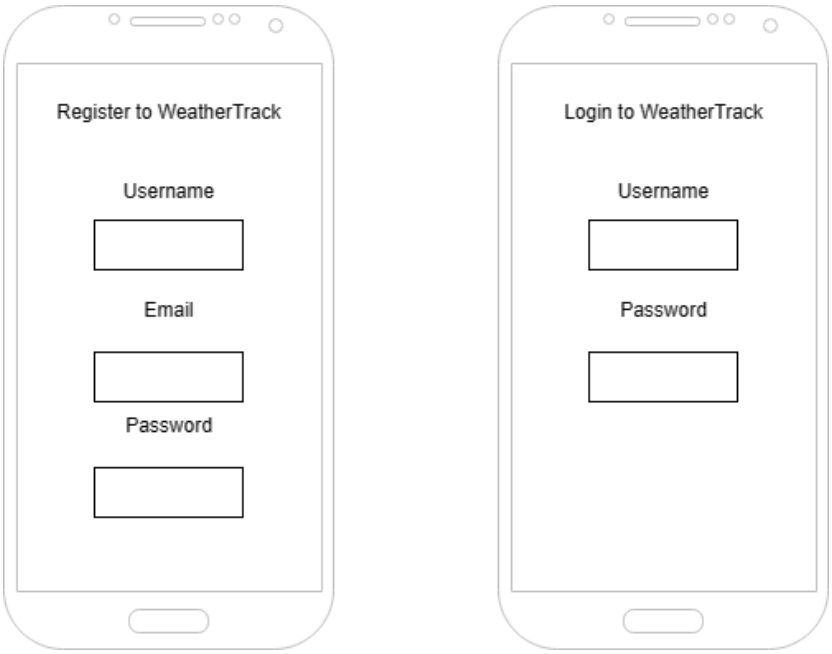

Figura 5: Mockup pagine amministratore

4 Tecnologie

4.1 Stack tecnologico

L'applicazione web è stata realizzata utilizzando lo stack MEVN, che comprende MongoDB (M), Express.js (E), Vue.js (V), e Node.js (N). Questo stack, oggi molto popolare, è stato scelto per la sua flessibilità, scalabilità e per la sua capacità di gestire sia il lato client che server dell'applicazione web con lo stesso linguaggio.

Più nello specifico di seguito vengono riportate le scelte che compongono l'intero stack tecnlogico:

- *Node.js + Express.js* per lo sviluppo del backend dell'applicazione web. Node.js è stato scelto perché è una piattaforma leggera e scalabile per la costruzione di applicazioni lato server, e il suo modello I/O non bloccante è ideale per gestire molteplici connessioni concorrenti. Express.js è stato scelto come framework web perché è semplice, flessibile e ampiamente utilizzato nella community Node.js.
- *Vue.js* [3] per lo sviluppo del frontend dell'applicazione web. Vue.js è stato scelto perché è un framework leggero e facile da imparare per la costruzione di interfacce utente dinamiche. Ha anche buone prestazioni e un ecosistema in crescita di plugin e strumenti.
- *MongoDB* [5] come database. MongoDB è stato scelto perché è un database NoSQL scalabile e flessibile che può gestire grandi volumi di dati non strutturati. Ha anche buone prestazioni e supporta ricche capacità di interrogazione. Inoltre, la sua funzione di change streams è molto adatta per la consegna di aggiornamenti in tempo reale.
- *Docker* [1] come gestore di container. Docker è stato scelto perché consente un facile rilascio e scalabilità delle applicazioni in container isolati. Semplifica anche il processo di sviluppo e testing fornendo un ambiente coerente su macchine diverse.
- *Docker-compose* [2] per l'orchestrazione dei container. Docker-compose è stato scelto invece di *Kubernetes* perché è più semplice da utilizzare e più adatto per implementazioni su piccola scala. Si integra anche bene con Docker e consente una gestione facile di applicazioni multi-container.
- *RabbitMQ* [8] come broker di messaggi.
- *Python* per gli script dei sensori e dei consumatori, in quanto fornisce un modo semplice ed efficiente per sviluppare e gestire gli script e un ampio ecosistema di librerie per dati e sistemi IoT.

4.2 Framework front-end

Di seguito vengono elencati i framework utilizzati per l'implementazione del front-end dell'applicazione web:

- **axios**: Libreria per effettuare richieste HTTP verso il backend, utilizzata per comunicare con il server e recuperare dati o effettuare operazioni CRUD.
- **bootstrap**: Framework CSS utilizzato per la creazione di interfacce utente responsive e moderne, semplificando lo sviluppo del frontend.
- **chart.js**: Libreria JavaScript per la creazione di grafici e visualizzazioni dati interattive, utilizzata per visualizzare dati in modo chiaro e comprensibile agli utenti.
- **chartjs-adapter-date-fns**: Adattatore per l'utilizzo di date con Chart.js.
- **jwt-decode**: Utilizzato per decodificare i token JWT lato client e ottenere informazioni sull'utente autenticato senza la necessità di effettuare una chiamata al server.
- **socket.io-client**: Client JavaScript per Socket.IO, utilizzato per la comunicazione in tempo reale tra client e server tramite WebSocket o altre tecnologie di trasporto.
- **vue-router**: Libreria Vue per la gestione delle rotte dell'applicazione, che consente la navigazione tra le diverse viste dell'applicazione seguendo un approccio single page application (SPA).
- **vuex**: Libreria Vue per la gestione dello stato dell'applicazione, che semplifica la gestione dei dati condivisi tra i componenti Vue.

4.3 Framework back-end

Di seguito vengono elencati i framework utilizzati per l'implementazione del back-end dell'applicazione web:

- **bcryptjs**: Utilizzato per l'hashing delle password degli utenti prima di salvarle nel database.
- **cors**: Utilizzato per gestire le richieste di origine incrociata (CORS) e consentire la comunicazione tra server e client su domini diversi.
- **jest**: Framework di test JavaScript utilizzato per testare il codice backend.
- **jsonwebtoken**: Utilizzato per la generazione e la gestione dei token JWT (JSON Web Token) per l'autenticazione basata su token.
- **mongoose**: ODM (Object Data Modeling) per MongoDB utilizzato per semplificare l'interazione con il database MongoDB.
- **socket.io**: Libreria JavaScript utilizzata per la comunicazione in tempo reale tra client e server tramite WebSocket o altre tecnologie di trasporto.

- **supertest**: Utilizzato per testare le API HTTP in modo semplice durante lo sviluppo.
- **swagger-jsdoc**: Utilizzato per generare automaticamente la documentazione API utilizzando annotazioni Swagger.

5 Codice

5.1 API

WeatherTrack fornisce un insieme di API RESTful per interagire con il backend del sistema tramite il suo server web.

Si è fatto uso di Swagger per documentare le API, semplificando la comprensione e l'interazione con il sistema. Swagger è un framework software open-source utilizzato per documentare e testare APIs RESTful. La specifica Swagger definisce gli endpoint dell'API, i formati di richiesta e risposta e i requisiti di autenticazione. La documentazione Swagger per il sistema WeatherTrack viene generata automaticamente dal codice utilizzando annotazioni Swagger. La documentazione generata è inclusa nel repository e può anche essere generata eseguendo il seguente comando (richiede *node*):

```
node app-backend/swagger.js
```

5.2 Middleware

Sia nel backend che nel frontend è stato fatto uso di un middleware.

5.2.1 Middleware backend

Per garantire la sicurezza dell'applicazione, è stato implementato un middleware di autenticazione e autorizzazione utilizzando JSON Web Token (JWT). Questo middleware verifica la validità dei token di accesso inviati nelle richieste API e controlla se l'utente ha i permessi necessari per accedere alle risorse richieste. Inoltre, un controllo aggiuntivo è stato introdotto per limitare l'accesso a determinate funzionalità ai soli utenti amministratori. Questa soluzione centralizzata semplifica la gestione dell'autenticazione e dell'autorizzazione in tutta l'applicazione, migliorando la sicurezza e riducendo la duplicazione di codice.

Di seguito vengono mostrati i due metodi del middleware:

```
  const verifyToken = async (req, res, next) => {
try {
  const token = req.headers.authorization;
  if (!token) {
    console.log("Verify token: No token provided");
    return res.status(403).send(
    { message: "No token provided" });
  }

  const decoded = jwt.verify(token, config.JWT_SECRET);
  const user = await
  User.findById(decoded.userId, { password: 0 });
  if (!user) {
    return res.status(404).send(
```

```
      { message: "No user found" });
    }
    if (user.role === "admin") {
      req.isAdmin = true;
    }

    next();
  } catch (error) {
    return res.status(500).send(
    { message: "Internal server error" });
  }
};

const isAdmin = (req, res, next) => {
  if (!req.isAdmin) {
    console.log("Is Admin: Require Admin Role!");
    return res.status(403).send(
    { message: 'Require Admin Role!' });
  }
  next();
};
```

Viene anche mostrato un esempio di come questo middleware sia aggiunto alle rotte dei router per proteggerle:

Rotta non protetta:

```
router.post("/notifications", notificationController.create);
```

Rotta protetta richiedente autenticazione nel sistema:

```
router.post("/notifications",
verifyToken, notificationController.create);
```

Rotta protetta richiedente autenticazione come amministratore nel sistema:

```
router.post("/notifications",
verifyToken, isAdmin, notificationController.create);
```

5.2.2 Middleware frontend

Il middleware dell'API frontend è implementato utilizzando Axios. Tale middleware specifica una volta l'URL per il server e imposta un interceptor per tutte le richieste API in uscita. L'interceptor controlla se un utente è attualmente autenticato cercando un token memorizzato nel sistema di gestione dello stato Vuex. Se è presente un token, viene aggiunto agli header della richiesta come token Bearer di autorizzazione. Questo assicura che tutte le successive richieste API saranno autenticate e autorizzate dal server, fornendo un ulteriore livello di sicurezza. Centralizzando la gestione del token nel middleware, si semplifica anche il codice in altre parti dell'applicazione (principio DRY). Il seguente listato mostra l'implementazione del middleware frontend :

```
const api = axios.create({
baseURL: "http://localhost:3000",
});
api.interceptors.request.use(
(config) => {
const token =
store.state.authModule.token;
if (token) {
config.headers.Authorization = token;
}
return config;
},
(error) => {
return Promise.reject(error);
}
);
export default api;
```

5.3 Aggiornamenti in tempo reale

L'applicazione sfrutta Socket.IO per permettere la comunicazione in tempo reale tra il server e il client, potendo così mostrare dati in tempo reale agli utenti.

L'implementazione lato server imposta namespace separati per i flussi di dati e le notifiche di anomalie, utilizzando i change stream [4] di MongoDB per rilevare e inviare nuovi dati e anomalie ai client connessi. Viene anche utilizzata una cache di chiavi per far sì che al verificarsi delle condizioni che fanno scattare una notifica per un'anomalia meteo, la stessa notifica non venga inviata ripetutamente all'utente.

Lato client, un modulo Vuex gestisce le connessioni socket per diverse località e il flusso di notifiche di anomalie. Il modulo recupera le località disponibili dal server e stabilisce socket di dati separati per ciascuna località. Inoltre, imposta un listener per il flusso di notifiche di anomalie, visualizzando una notifica visiva all'utente al ricevimento di un nuovo evento di anomalia. Gli utenti impostano notifiche personalizzate tramite una apposita interfaccia selezionando località, attributi e soglie. Oltre alle notifiche push in tempo reale, gli utenti possono visualizzare uno storico delle anomalie generate secondo i parametri da loro impostati.

Gli aggiornamenti in tempo reale vengono visualizzati nell'interfaccia utente attraverso componenti reattivi. Un componente si registra al socket dei dati corrispondente alla sua località e aggiorna in tempo reale i dati visualizzati quando vengono raccolti dai sensori. Un altro componente ascolta gli le anomalie sul flusso di notifiche e mostra una notifica temporanea all'utente quando viene rilevata un'anomalia registrata dall'utente.

Questa architettura di comunicazione in tempo reale garantisce che l'applicazione possa fornire efficacemente informazioni in tempo reale agli utenti,

consentendo loro di monitorare i cambiamenti nei dati e le anomalie man mano che si verificano, senza la necessità di ricaricare la pagina o effettuare polling.

6 Test

6.1 Unit test

Per testare i vari moduli che compongono il back-end sono stati creati degli unit test utilizzando Jest. Jest è un framework di testing JavaScript sviluppato da Facebook; fornisce funzionalità come il testing degli snapshot, il mocking e l'analisi della copertura del codice, rendendolo una soluzione completa per il testing delle applicazioni JavaScript. L'analisi degli unit test, eseguita aggiungendo il comando *–coverage* allo script Jest di test, mostra una code coverage dell'84% del codice back-end.

Tramite *GitHub Actions* è stata messa a punto una pipeline che esegua automaticamente i test ogni volta che nuovo codice viene aggiunto al repository, rendendo così lo sviluppo di nuove feature più semplice e sicuro.

6.2 Test manuali

Durante lo sviluppo del sistema, è stato utilizzato Postman come strumento per condurre test e debug manuali sulle API. Postman è una piattaforma di sviluppo API che consente di inviare richieste HTTP a un server e di visualizzare le risposte in modo facile e intuitivo.

6.3 Test con utenti

Il sistema è stato testato con l'aiuto di volontari che impersonassero le due categorie di utenti (utente normale e amministratore). Grazie al riscontro da loro fornito è stato possibile validare i requisiti, uno su tutti l'usabilità. Per valutare questo requisito sono infatti state poste agli utenti domande basate sul decalogo di euristiche di Nielsen [6]. Le risposte a tali domande hanno dato esiti positivi.

7 Deployment

Il sistema è stato sviluppato con una logica basata su componenti, e ciascuno di questi componenti è stato progettato per essere distribuito in un container utilizzando *Docker*. L'orchestrazione dei componenti e la gestione delle loro interazioni è stata gestita con lo strumento *docker-compose* di *Docker*. Per quanto riguarda l'automazione della build, *pip* è stato utilizzato come sistema di gestione dei pacchetti per le parti del sistema sviluppate in *Python* (sensore e consumer di messaggi), mentre *npm* è stato utilizzato per le parti sviluppate in *JavaScript* (backend e frontend dell'applicazione web).

Il repository contiene anche dump del database in formato JSON che vengono importati durante la creazione dell'immagine del database. Questi contengono dati di dimostrazione insieme ad alcuni utenti pre-registrati per testare il sistema. Il Dockerfile del database, oltre ad importare i dati di dimostrazione, inizializza anche il replica set, che è richiesto sia per la funzionalità di aggiornamento in tempo reale che per garantire la fault tolerance del sistema.

Per testare lo scenario dell'utente normale, utilizzare:

- Nome utente: *user*.
- Password: *user*.

Per testare lo scenario dell'amministratore, utilizzare:

- Nome utente: *admin*.
- Password: *admin*.

Inoltre, poiché i sensori sono simulati attraverso chiamate API, è richiesta una chiave API OpenWeatherMap [7]. La seguente chiave API può essere utilizzata per testare il sistema e deve essere aggiunta fra le virgolette del parametro *API_KEY*: *f947a05b36f9e57f9c15 fcc3ec250d1c*.

Di seguito le istruzioni per avviare il sistema:

1. Installare *Docker* e *docker-compose*.
2. Clonare il repository del progetto.
3. Spostarsi nella root del repository.
4. Aprire il file *secrets.txt* al path *weathertrack/config*. Aggiungere la chiave dimostrativa f947a05b36f9e57f9c15fcc3ec250d1c fra le virgolette.
5. Eseguire il comando *docker-compose up –build*, questo costruirà e avvierà tutti i container del sistema.
6. Una volta che i container sono attivi e in esecuzione, è possibile accedere all'applicazione web navigando su *http://localhost:5173* nel browser web.
7. Per fermare il sistema, eseguire il comando *docker-compose down* nella directory root della repository.

8 Conclusioni

In conclusione, WeatherTrack è un'applicazione web efficace e scalabile per la raccolta e la visualizzazione di dati meteorologici in tempo reale da sensori IoT. L'utilizzo di un approccio User Centered e Mobile First garantisce un'esperienza utente intuitiva, come certificato dai test con gli utenti. Il metodo Agile ha invece permesso di consegnare funzionalità incrementali.

Il sistema è stato progettato per essere flessibile e modulare, con l'utilizzo di container per il deployment. La scelta dello stack MEVN si è rivelata vincente, grazie alla sua flessibilità, scalabilità e disponibilità di framework e librerie.

WeatherTrack ha dimostrato di essere un'applicazione web efficace e scalabile, che soddisfa i requisiti definiti in fase di analisi. Cimentarsi in questo progetto ha permesso al sottoscritto di affinare le tecniche di progettazione e sviluppo di applicazioni web, con un occhio di riguardo per l'utente e l'usabilità e soprattutto facendo esperienza con lo stack MEVN, oggi ampiamente utilizzato per via delle sue qualità sopra descritte.

Riferimenti bibliografici

[1] Docker, Inc. Docker. https://www.docker.com/.

[2] Docker, Inc. Docker Compose. https://docs.docker.com/compose/.

[3] Evan You. Vue.js - a progressive JavaScript framework. https://vuejs.org/.

[4] MongoDB . MongoDB Manual - Change Streams. https://www.mongodb.com/docs/manual/changeStreams/.

[5] MongoDB, Inc. MongoDB. https://www.mongodb.com/.

[6] Nielsen Norman Group. 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics.

[7] OpenWeatherMap. OpenWeatherMap - Weather API. https://openweathermap.org/.

[8] RabbitMQ. RabbitMQ - a message broker. https://www.rabbitmq.com/.