**Prepper Application Documentation**  
**Prepper – real time disaster prepping information.**

**Introduction**

Prepper is a disaster preparedness web application providing users with real-time weather updates, earthquake alerts, air quality data, and curated news articles.

Built with Node.js and Express.js, it leverages APIs like Weatherbit and Event Registry for live data integration. MySQL serves as the data layer for storing user preferences and fetched data, ensuring persistence and scalability.

It provides users with tailored alerts, news updates, and survival tips, all accessible via a user-friendly dashboard. By leveraging modern APIs, databases, and Node.js-based backend architecture, Prepper integrates real-time data into an interactive, web-based interface. Key features include customizable user preferences, geolocation-based searches, and personalized disaster updates, ensuring users are always prepared for emergencies.

I wanted to develop this app to be more aware of how to survive in major disasters, and to also keep track of your current surroundings.

**High-Level Architecture**

The application consists of:

* Frontend: EJS templates for dynamic, server-side rendered views.
* Backend: Node.js with Express for server logic and routing.
* Database: MySQL for structured data storage.
* APIs: Integration with Weatherbit, USGS, Open-Meteo, and Event Registry APIs for real-time data.

**GitHub Repository**: [Prepper on GitHub](https://github.com/Nousagi44/Prepper)

* <https://github.com/Nousagi44/Prepper>

**Live Application**: [Prepper Dashboard](https://www.doc.gold.ac.uk/usr/154/dashboard)

* <https://www.doc.gold.ac.uk/usr/154/>

Security

SQL Injection Protection: All database queries use parameterized statements.  
Password Security: Bcrypt is used to hash passwords securely.  
Session Management:

Sessions stored securely with expiration and secure cookies.

Cookies are configured with the HttpOnly and Secure flags to prevent client-side scripts from accessing session data and to ensure cookies are transmitted only over HTTPS.

Session Expiry: Sessions automatically expire after 10 minutes of inactivity, reducing the risk of unauthorized access.

Session ID Regeneration: Session IDs are regenerated upon login to prevent session fixation attacks.

Input Validation: Frontend and backend sanitization ensures only valid data is processed.  
Error Handling Errors are logged securely without exposing details to the user.

**Access Control**

Route Protection:

Sensitive routes, such as /dashboard and /data/\*, are protected using middleware to ensure only authenticated users can access them.

API routes are secured with session-based authentication.

**Setup and Installation**

Prerequisites

1. Node.js (v14+)
2. MySQL
3. Git

Steps

1. Clone the repository:

git clone https://github.com/Nousagi44/Prepper.git

cd Prepper

npm install

Configure the database:

* Import database.sql into your MySQL server.
* Update the database connection in index.js:

const db = mysql.createConnection({

host: 'localhost',

user: 'prepper\_app',

password: 'YourPassword',

database: 'prepper\_db',

});

npm start

**Database Schema**

**Overview**

Prepper uses a MySQL database with the following tables:

1. users: Manages user accounts.
2. user\_preferences: Stores user-specific settings.
3. weather\_data: Historical weather data for users.
4. earthquake\_data: Earthquake records by location.
5. air\_quality\_data: Stores air quality indices.
6. news\_data: Curated news articles related to disaster topics.

A screenshot of a computer

Description automatically generated

**Core Features**

**1. Weather Updates**

* Fetches data from Weatherbit API.
* Displays temperature, humidity, UV index, and conditions.

**2. Earthquake Alerts**

* Retrieves data from the USGS Earthquake API.
* Lists recent earthquakes near the user’s location.

**3. Air Quality Reports**

* Provides air quality indices from Open-Meteo.

**4. News Updates**

* Uses Event Registry API to deliver news articles categorized into disaster preparedness, global conflict, and financial risks.

**Links and Logins**

* URL of deployed app: <https://www.doc.gold.ac.uk/usr/154>
* GitHub repository: <https://github.com/Nousagi44/Prepper>
* Demo Credentials:
  + Username:tim
  + Password: 123456

**API Documentation**

Base URL

http://localhost:8000/api/weather

[{"id":9,"user\_id":1,"city":"Lat: 51.4424832, Lon: -0.0360448","temperature":"6.70","humidity":90,"conditions":"Few clouds","uv\_index":"0.0","retrieved\_at":"2024-12-10T17:54:47.000Z"},

Alternatively for use on the virtual server

<https://www.doc.gold.ac.uk/usr/154/api/weather>  
<https://www.doc.gold.ac.uk/usr/154/api/earthquakes>   
<https://www.doc.gold.ac.uk/usr/154/api/airquality>  
<https://www.doc.gold.ac.uk/usr/154/api/news>

Architecture

The Prepper app uses:

* Application Tier: Node.js and Express.js for server-side logic.
* Data Tier: MySQL with UTF-8 support for multilingual compatibility.
* API Integrations: Weatherbit, Open-Meteo, and Event Registry for real-time data fetching,
* NEWSAI.API for real-time local news fetching.

Data Model

Diagram and Description

The database contains relational tables:

* users: stores user details.
* weather\_data, earthquake\_data, air\_quality\_data, news\_data: store real-time fetched data.

User-Facing Functionality

1. User Authentication:
   * Registration and login functionalities with password hashing using bcrypt.
   * Session management for secure access.
2. Dashboard:
   * Links to weather, air quality, earthquake, and news information.
   * Includes links to YouTube guides.
3. Dynamic Data Fetching:
   * Weather, air quality, earthquake data, and news articles fetched based on user location or preferences.
   * Data is stored in the database for user-specific retrieval.
4. Customizable Preferences:
   * Users can set default cities, alert thresholds, and notification preferences.
5. Responsive UI:
   * EJS templates styled with CSS for consistent and user-friendly interfaces.
6. Error Handling:
   * Errors displayed in a user-friendly manner with secure logging.

Templates

Prepper uses EJS templates for dynamic content rendering:

* views/index.ejs: Landing page.
* views/dashboard.ejs: User dashboard.
* views/weather.ejs: Weather data.
* views/earthquakes.ejs: Earthquake updates.
* views/airquality.ejs: Air quality details.
* views/news.ejs: Curated news.

Styling

All styles are in public/main.css. The design focuses on:

* Simplicity and accessibility.
* Responsive layouts for mobile and desktop.

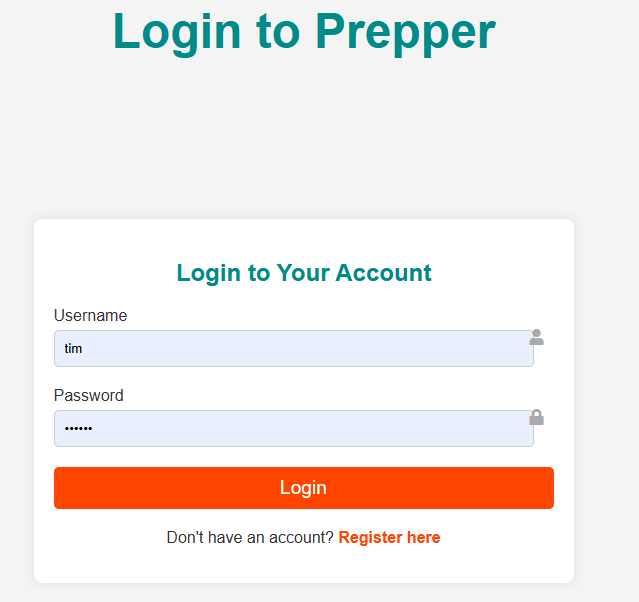
User Functionality

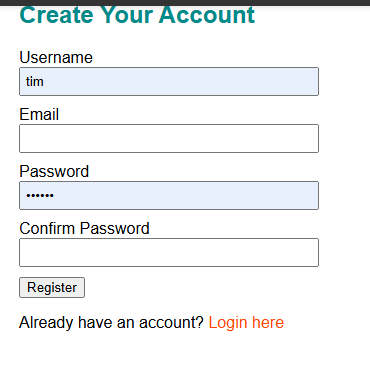
Overview with Screenshots

Landing page

A screenshot of a website

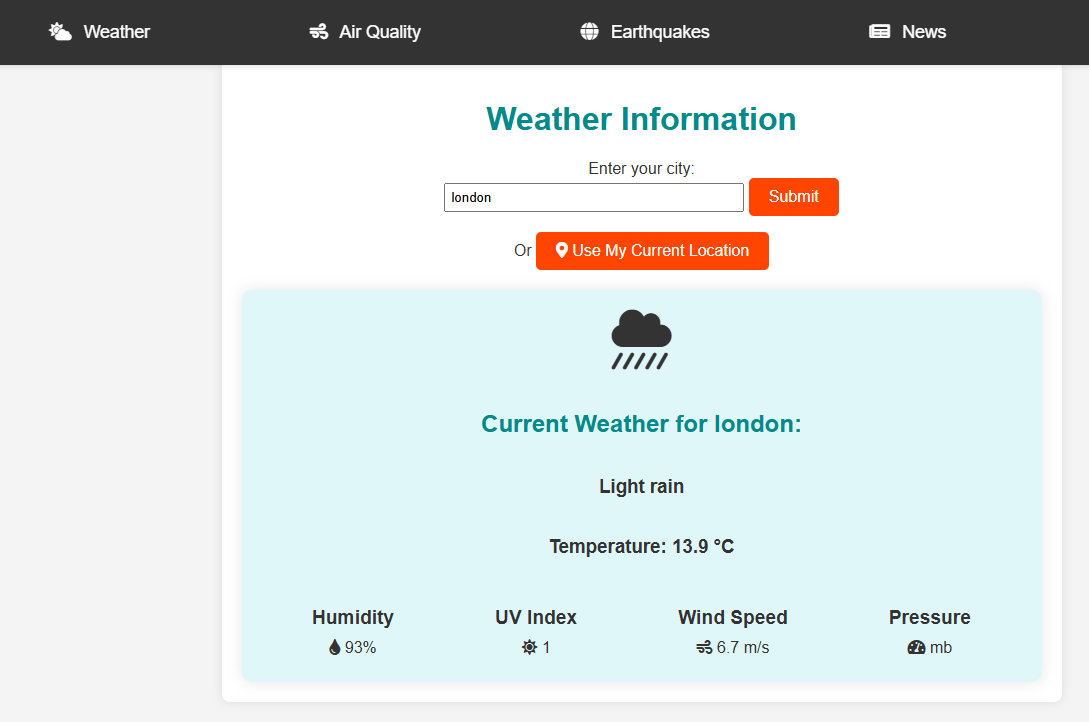
Description automatically generated

* Registration and Login: Users can register and log in with secure credentials.

Prepper home page


Dashboard: A central hub home page to prepper

Weather Updates: Current temperature, humidity, and UV index by city or user location.



Air Quality: Displays AQI and pollutant levels.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedEarthquakes: Lists recent earthquakes, magnitude, and distance from the user.

News: Categorized survival-related news.

A screenshot of a web page

Description automatically generated

A screenshot of a video chat

Description automatically generated

YouTube Guides:

Handpicked YouTube playlists in a carousel style

Based on

Urban Survival, Global Conflict Survival, Disaster Survival, Prepping

Deployment

The application is hosted on a virtual server at [Prepper Dashboard](https://www.doc.gold.ac.uk/usr/154/dashboard).

(<https://www.doc.gold.ac.uk/usr/154>)

Hosting Details

* Server: Goldsmiths’ virtual server
* Process Manager: pm2 to keep the Node.js server running.

**Improvements**

If I were to suggest a few improvements, I could make given more time.

First, it would be beneficial to include live shopping data for essential survival items from local stores. Additionally, integrating APIs, such as the police API, could help users monitor crime in their neighbourhoods. Traffic APIs would assist in selecting safer routes, while charging station APIs would be valuable for users with electric vehicles.

More importantly, providing users with a way to obtain a PDF of important updates would ensure that vital data remains accessible even without internet access. Lastly, I believe that improving YouTube's loading time could enhance its responsiveness, particularly by optimizing how images are fetched and displayed.

Acknowledgments

This project uses data and frameworks from:

* [Weatherbit API](https://www.weatherbit.io/)
* [USGS Earthquake API](https://earthquake.usgs.gov/)
* <https://newsapi.ai/documentation?tab=introduction>
* <https://open-meteo.com/en/docs/air-quality-api>
* <https://open-meteo.com/en/docs/geocoding-api>

Additional resources:

* "Express in Action" by Evan Hahn
* "Learning JavaScript Design Patterns" by Addy Osmani