Designing and Implementing a Backend API Structure:
Design Proposal

Stewart Charles Fisher II
Student ID: 25020928
QLS ID: FIS20772963
25020928@students.lincoln.ac.uk

April 2024

UNIVERSITY OF
LINCOLN

School of Computer Science
University of Lincoln
United Kingdom

Submitted in partial fulfilment of the requirements for the Degree of MComp Computer Science

Word Count: 2784

Contents

Chapter 1

Background

In the modern era, where a personalised experience is becoming increasingly valued, I propose a new API solution that brings a personal weather report to you. My API, named *Weather App*, is designed to seamlessly combine location retrieval, weather data reception, and AI-driven narrative generation together with user-friendly access methods, to provide a user with a simple, quick, and easy solution to get their news about the weather.

The primary objective of *Weather App* is to showcase the new possibilities of integrating new, cutting edge AI technologies, such as large language models, to both accelerate and customise the user experience. By leveraging the state-of-the-art advancements in natural language processing, *Weather App* aims to deliver a weather update service in a format that is not only informative but also engaging and personable.

Weather App will harness the power of OpenAI's GPT-3.5 Turbo Chat Completions API model, enabling it to generate descriptive messages using the context of the user's location and their current weather conditions. This integration of advanced AI capabilities will allow *Weather App* to provide a user with a level of personalisation and familiarity that was previously unattainable using traditional weather forecasting applications.

The following proposal shall outline the initial design of the API, the estimates of the costs involved, and a prospective deployment protocol. By establishing the key features and functionalities of *Weather App*, I hope to demonstrate its potential to improve the way in which you can interact with weather forecasting.

Chapter 2

Design

The *Weather App* API currently encompasses a total of 10 routes, facilitating tasks such as:

- User registration
- Authentication
- Profile management
- Weather data retrieval

2.1 Registering a User

This endpoint facilitates the registration process for new users. When a user registers, they will provide a unique username, a unique email address, and a password. These details are validated against the database to ensure that the username and email are unique, and frontend validation is included to validate for username formatting. Upon a successful validation, the user's password is securely hashed using the bcrypt hashing function before being stored in the database, to maintain confidentiality. The bcrypt hashing function is ideal for its attack threat resistance, use of password salting, and adaptability. The registration process may develop further to include measures to prevent abuse, such as CAPTCHA challenges to verify that the user is not a bot. These measures help maintain the integrity of the user database and protect against unauthorised access.

Currently, uniqueness and email validity are the only requirements for a user's account parameters. Future implementations should include features such as password character content verification, and minimum lengths.

Once registered, a user will receive a confirmation message to their client indicating a successful registration. They can then proceed to log in to the application using their credentials. Upon registration, a user's credit balance is set to 2000 credits. The user registration endpoint is the cornerstone in onboarding new users and ensuring a smooth experience for all users of the application.

2.2 Logging in a User

The login endpoint enables a registered user to authenticate themselves with the server and gain access to the application's authorisation protected resources. A

user will provide their username and password, which are then verified against the stored credentials in the database. The stored password hash is compared against the login input to verify the credentials.

Upon a successful authentication, the user is issued a JWT access token, which serves as a digital key granting access to the protected routes within the application. This token is valid for a period of 30 minutes, after which the user must reauthenticate themselves to obtain a new token. This mechanism helps prevent unauthorised access and enhances the overall security of the system. The time limit also aids security; a token that doesn't expire can become a breach threat if the token is stolen, leading to permanent access to a user's data. A breach of this kind could potentially be a violation of the GDPR and the Data Protection Act 2018.

As with the registration endpoint, the login endpoint could also consider integrating CAPTCHA challenges to aid in user verification.

2.3 Getting a User Profile

The endpoint for retrieving a user's profile information allows a frontend engineer to access and display a user's account details to the user. This information includes the user's username and the credit balance of their account. This feature is essential for users to manage their account settings, update their personal information, and view their current credit balance.

Upon making a request to retrieve their profile, the user's identity is verified using their authentication token to ensure that only authorized users can access the information. Once authenticated, the server retrieves the user's profile data from the database and returns it in the response.

Overall, the endpoint for getting a user's profile plays a crucial role in dynamically providing users with access to their account information and allowing them to manage their accounts within the application.

2.4 Updating a User Profile

The endpoint for updating a user's profile allows users to modify their account username as needed. While this does not currently include the option to change the user's email address or password, these are considerations for future improvements.

To initiate an update, the user submits a request with the desired username change. The client validates that the username parameter is acceptable, before the server then validates the changes against the database to ensure they comply with the uniqueness constraints.

Once validated, the server applies the requested changes to the user's profile, updating the corresponding data in the database. If successful, the server returns a confirmation message indicating that the changes have been applied successfully. The client will then reload the page with the updated username displayed to the user. To accompany the updated details, a new JWT access token is issued along with a JWT refresh token, which allows the user to obtain a new access token without having to reauthenticate.

2.5 Deleting a User Profile

The endpoint for deleting a user's profile allows users to permanently remove their account and associated location data from the system; if the location data is not removed, a future account of the same name could access this information. Currently, this action is irreversible, however there are no plans to implement an account recovery process. The account deletion process is a necessary feature, due to the GDPR and Data Protection Act 2018 stipulating a user's right to erasure.

To delete their profile, the user submits a request to the server indicating their intent to delete their account. The server then verifies the user's identity using their JWT access token and proceeds to delete all records associated with the user from the database. This includes removing the user's profile and their associated locations; access tokens are revoked upon account deletion. Once the deletion process is complete, the server returns a confirmation message to the client indicating that their account has been successfully deleted.

2.6 Getting a User's Credit Balance

The endpoint for retrieving a user's credit balance allows a frontend engineer to display a user's credit balance available in their account. Credits are used as a form of currency within the API to make a weather report request.

To retrieve their credit balance, the client submits a request to the server, which verifies their identity with the JWT access token, retrieves the current user's details, and finds the current credit balance from the database. The server then returns this information to the user in the response, allowing the client to display the user's available credits.

2.7 Purchasing User Credits

The endpoint for purchasing user credits allows users to acquire additional credits for their account, to continue using the weather requests. A user is limited to adding 1-500 credits to their account.

To purchase credits, the user submits a request to the server, indicating the desired amount of credits to acquire. The server then adds the purchased credits to their account balance.

Once the transaction is complete, the server returns a confirmation message to the client indicating that the credits have been successfully added to their account, and displays the new credit balance.

Currently, the endpoint does not take payment for credit purchase. Future implementations of the API should include such services. An ideal external API to use for this would be the Stripe API. The Stripe API would allow not only for user's to make one-time purchases of credits, but would also allow the facilitation of credits subscriptions via recurring billing (Stripe, 2023).

2.8 Getting a User's Locations

The endpoint for retrieving a user's locations allows a frontend engineer to display a list of locations that a user has previously added to their account. This feature is

useful for users who frequently access weather information for multiple locations and wish to easily manage and access their saved locations within the application.

To retrieve their locations, the client submits a request to the server, which verifies the user and retrieves the list of locations associated with their account from the database. The server then returns this list to the client in the response, allowing the locations to be displayed to the user in a tabular format.

Currently, the list of locations can return the following information:

- Temperature
- Location
- Coordinates
- Timestamp
- AI generated description

In addition to displaying the list of locations, the endpoint may also provide additional details such as the name of each location, its geographical coordinates, or any other relevant information stored in the database. This helps users identify and manage their saved locations more effectively within the application. Pagination was implemented for the table and endpoint to reduce overloading the API requests, using a limit of 10 rows per table page.

2.9 Adding a New Location

The endpoint for adding a new location allows users to add their current location and weather conditions to their account. This feature primarily revolves around the use of 3 external APIs:

- IPData: IP Geolocation API
- OpenWeatherMap: One Call API 3.0
- OpenAI: GPT-3.5 Turbo Chat Completions

To add a new location, the user submits a request to the server to retrieve the weather of their current location they wish to add. The server then validates the request and proceeds to approximate the user's location using their IP address; for testing purposes, the IP address of the server is used. Once the IP address is geolocated, the coordinates are used to retrieve the weather conditions. The weather conditions are extracted from the response package, and given to the chat completion API with the instructions in Listing 2.1.

```
{
    "role": "user",
    "content": f"You are a weather assistant, and I want you to
       extend the following sentence with a little message about
       what to wear: {weather_info}",
}
```

Listing 2.1: The instruction given to the Chat Completions API

The response from OpenAI is what is finally provided to the user in the output table. Using the OpenWeatherMap API with the OpenAI API allows for other information to be given to the assistant, to add more narrative for the user.

Credits are deducted from the user balance when they make a request. For testing purposes, the credit cost is set to a single credit, however the future implementations of the API should consider credit costs adjusted to account primarily for the costs of the three external API calls.

2.10 Deleting a User's Location

The endpoint for deleting a user's location allows users to remove unwanted or outdated locations from their account, helping them keep their list of saved locations organised and up to date. This feature is useful for users who no longer wish to access weather information for certain locations or who have accidentally added duplicate locations to their account.

To delete a location, the user submits a request to the server to delete a particular row from the table containing the location they wish to delete. The server then verifies the request and proceeds to remove the specified location from the user's account in the database. Once the location has been successfully deleted, the server returns a confirmation message to the client indicating that the location has been removed from the user's account and updates the onscreen table.

2.11 Database

The database currently used in the prototype currently uses SQLite. SQLite is a good choice for small scale applications, providing a lightweight database. Unlike most client-server databases, a SQLite database is serverless and does not require a dedicated server process. It is also provides faster transactional speeds.

As the userbase of the API grows, it would be ideal to consider other database options, such as PostgreSQL. Unlike SQLite, a PostgreSQL database is capable of handling concurrent accesses of large volumes, provides a wider range of data types, and provisions dedicated security features that SQLite does not provide (Team, 2023).

Chapter 3

Pricing Model

Weather App could adopt a variety of pricing models to cater to different user types:

Freemium: A user should at least be provided with a minimal credit balance, per day or per month, so that new customers can be attracted to the service in hopes of user retention.

Pay-as-you-go: A user is charged directly based upon their actual usage of the service. A pay-as-you-go model allows the user to flexibly use the service, without a long term tie-in, catering to potential customers who haven't fully bought into the product yet.

Subscription based: A user will pay a monthly or annual subscription plan with recurring payments, selecting for a specific tier that caters to their usage level; each tier provides a certain credit balance per day. A subscription model would provide a predictable revenue source, and further encourages long term usage from the customer.

Future implementations of the service could implement features that provide priority access to users, based upon their tier. An example of this would be subscription-based customers being given priority to the service over freemium users.

Chapter 4

Deployment

Deploying the *Weather App* API involves making it accessible to users while ensuring reliability, scalability, and security. A robust deployment strategy is essential to achieve these goals and provide a seamless experience for API consumers.

4.1 Development Environment

Selecting the appropriate deployment environment is a key step in the deployment process:

On-premises deployment: If it is deemed that the API requires strict compliance or specific infrastructure needs, it is possible to opt for on-premises deployment. This would involve hosting the API on servers located within your organisation's data centre, which would provide full control over security and performance.

Cloud-based deployment: If scalability, reliability, or global accessibility is the primary concern, leveraging cloud platforms such as AWS, Azure, or Google Cloud. Cloud services such as these provide managed infrastructure, auto-scaling capabilities, and built-in security features, making them ideal for hosting a scalable version of the *Weather App* API.

Hybrid: A hybrid deployment would combine elements of both cloud-based and on-premises deployment, allowing the organisation to leverage the benefits of both approaches. For example, sensitive user data or payment details may be stored on-premises, while less sensitive components such as the user's saved locations are hosted in the cloud for scalability.

4.2 Containerisation

Containerisation via Docker would offer a powerful solution for deploying and managing the *Weather App* API, providing benefits such as portability, scalability, and simplified orchestration. Currently, Docker is employed by the prototype to ensure cross-platform compatibility and package integrity.

Docker allows developers to package the *Weather App* API, the client. and their dependencies into lightweight, portable containers that can run consistently across different environments. This encapsulation ensures that the API behaves

consistently regardless of the underlying infrastructure, reducing the risk of compatibility issues and simplifying the deployment process. Furthermore, Docker containers allow for automated software security analysis in CI/CD, via the Docker Scout service. Using Docker Scout, vulnerabilities can be identified and addressed earlier without requiring a dedicated anti-threat team.

Kubernetes, an open-source container orchestration platform, would complement Docker by providing advanced features for managing the containerised application when operating at scale. Kubernetes automates tasks such as deployment, scaling, and load balancing, which would allow the *Weather App* API to be deployed with ease and efficiency (Hat, 2020). By leveraging Kubernetes, the API can achieve:

Scalability: Kubernetes would enable automatic scaling of the *Weather App* API based on demand. It monitors resource usage and adjusts the number of containers dynamically to accommodate changes in traffic, ensuring optimal performance and cost efficiency.

Accessibility: Kubernetes provides built-in mechanisms for ensuring high availability of the *Weather App* API. It would automatically restart containers that fail or become unresponsive, minimising the downtime and ensuring uninterrupted service for our customers.

Rolling release updates: Kubernetes supports rolling updates, which would allow us to deploy new versions of the *Weather App* API without any downtime. It gradually replaces old containers with new ones, ensuring a smooth transition and minimising disruption to our customers.

Load balancing and service discovery: Kubernetes includes features for service discovery and load balancing, allowing clients to access the *Weather App* API reliably. It automatically assigns IP addresses to containers and distributes incoming traffic evenly across multiple instances, improving reliability and performance.

Overall, containerisation using both Docker and Kubernetes would offer an efficient approach to deploying and managing the *Weather App* API. By encapsulating the API in containers and leveraging Kubernetes for orchestration, you can achieve greater agility, scalability, and reliability in their deployment infrastructure.

Bibliography

Hat, R. (2020, March). What is kubernetes? *www.redhat.com*. Retrieved April 19, 2024, from https://www.redhat.com/en/topics/containers/what-is-kubernetes

Stripe. (2023, April). Payment apis explained: What businesses should know — stripe. *stripe.com*. Retrieved April 19, 2024, from https://stripe.com/gb/resources/more/payment-apis-101

Team, A. A. (2023, June). Postgresql vs sqlite: The ultimate database showdown. *Astera*. Retrieved April 19, 2024, from https://www.astera.com/knowledge-center/postgresql-vs-sqlite/