Wearther : Weather-based Outfit Suggestion

Jiahong Li
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, United States
jli11@scu.edu

Piyush Kulkarni
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, United States
pkulkarni@scu.edu

Yuehan Cui
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, United States
ycui2@scu.edu

Almas Khan
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, United States
akhan3@scu.edu

***Abstract*—With the development of web technology and increased accuracy of meteorology, more and more weather forecasting apps have emerged in the market. As many of them have potent functionalities such as minute-level precipitation forecasts, and air quality indications, they seldom give accurate and detailed suggestions on what to wear and how much to wear based on the weather. We believe this functionality can significantly ease our lives by saving time in checking the weather and considering what to wear before leaving home every day. Therefore, we introduce a Wearther, a cloud-hosted web application with a modern UI design and powerful backend logic. It allows users to check real-time weather conditions, review weather-based clothing suggestions, and roll back account-based history data.**

***Index Terms*—weather, forecast, clothing, suggestion**

I. Introduction

A. The Outfit Suggestion System

In this project, we implemented an outfit suggestion system. It gives users suggestions on what to wear for a given day. Finding what to wear seems to be trivial, but it is actually not. It depends on lots of factors such as where you go, what the season outside is if you need to walk/bike a lot outside, if you are going to do sports, and most importantly, the weather. The temperature, wind, rain, UV index, and humidity are all important factors that need to be considered before deciding what you wear. It is a complex job for people and we need to do it everyday. So, in this project, we simplified this process. We implemented a suggestion scheme called the 26-degree outfit rule to help the user choose what to wear wisely. The system is also accessible, all user need is to register for an account and log in to the system. The system acquires the user's location from the device and retrieves weather from weather service, and suggestions are presented to the user. The user can then find a few groups of suggestions on what to wear and then they can decide themselves.

B. The Technical Challenges

We are implementing a full-stack website service in this project. It is a very involved project. From the user's perspective, it has a frontend UI design component, a frontend interaction component, a backend component, and an object storage component. But it is actually more than that. To support the website, there is a database component, a container image registry component, and an entire suite of cloud infrastructures as well. The interaction between components needs collaboration and testing. The difference in architecture on the local development machine and on a cloud server is bringing challenges to this project as well. The choice of deployment server and the deployment server orchestration needs testing to find the best performance-to-cost option.

II. Background and Related work

Humans have developed weather forecasting for thousands of years. It's been the most crucial prerequisite knowledge for agriculture and thus one of the most fundamental technologies that build up human civilization. Thus, to utilize technology with such importance, there have been numerous weather forecasting applications and tools in the market. Nevertheless, only a tiny portion of them offer users suggestions on their clothing based on the weather conditions. And some of the most representative ones will be introduced below.

A. Mobile Applications

If one searches for "weather forecast applications with clothing suggestion", Weather Fit [4] may probably be one of the top results. Being one of the best products in this field with a 4.6 out of 5 review score, Weather Fit has a modern UI design and informative features that are elegant and ease-to-use for smartphone users. Unfortunately, it only supports iOS devices and requires a monthly subscription fee, making it an exclusive and expensive utility tool for iPhone and iPad users. On the Android side, Weatherproof [6] has been one of the most popular choices on the google app store. It has a decent review score of 4.4 out of 5 and score-matching powerful features on weather forecasts and clothing suggestions. However, Weatherproof's UI design is not appealing at all. With a design style like the earliest mobile apps when iPhone was born, younger generations will find it hard to navigate and use. In addition, it's an application that is based solely on Android OS. Another application called Snafu [5] may provide better accessibility since it supports both iOS and Android. On the other hand, with a 2.8 out of 5 on the google app store, Snafu's core functions, weather forecast and clothing suggestions, often stop working or give totally incorrect dressing advice.

B. Web Applications

Regarding accessibility, web applications are often the most inclusive ones since all devices with internet connections and graphic browsers can visit them with ease. Furthermore, since the applications we discussed here only seek to provide weather and dressing information in a concise and informative manner, web applications are sufficient to provide interaction space and responses on small devices like smartphones. Developed in 2013, being one of the oldest websites that provides clothing suggestion features based on weather forecasts, Clothesforecast.com [1] fails to offer a clear and intuitive UI to show its dressing suggestions. In addition, it also placed many advertisements around the main panels, making the UI even harder to navigate. Moreover, Dailydressme.com [2] provides a more beautiful and concise UI that lists clothing suggestions and weather forecasts in one tall container. However, rather than giving inclusive and generalized suggestions, it provides users with outfit combos from an online store called Revolve, which only sells female clothing.

C. Insights

Based review of the above mobile and web applications' advantages and drawbacks, we come to a baseline to create an accessible web application that provides sufficient weather info as well as inclusive and diverse clothing suggestions. Furthermore, this application needs to have a modern UI design that is not only concise and easy to navigate but also appealing to use.

III. Approach

A. Frontend

The front-end of Wearther is built based on React framework. React is a framework developed by Facebook to modulize the front-end development logic using a new syntax called JSX. It empowers the front end to render only necessary parts - components that are actually changing and thus lead to a much higher performance and better efficiency. Together with React, a template called Material Dashboard 2 React [3] and various libraries are used. Some of the most significant libraries used in the front end are React Context, Material UI (MUI), and fullcalendar. In particular, React Context enables the front-end to store and get state dynamically and is mostly used for storage of login tokens, location info, and weather conditions fetched from the backend. MUI is used for better and smoother component designs since Google provides many performing and concise animations in it. In addition, fullcalendar offers the interactive calendar feature

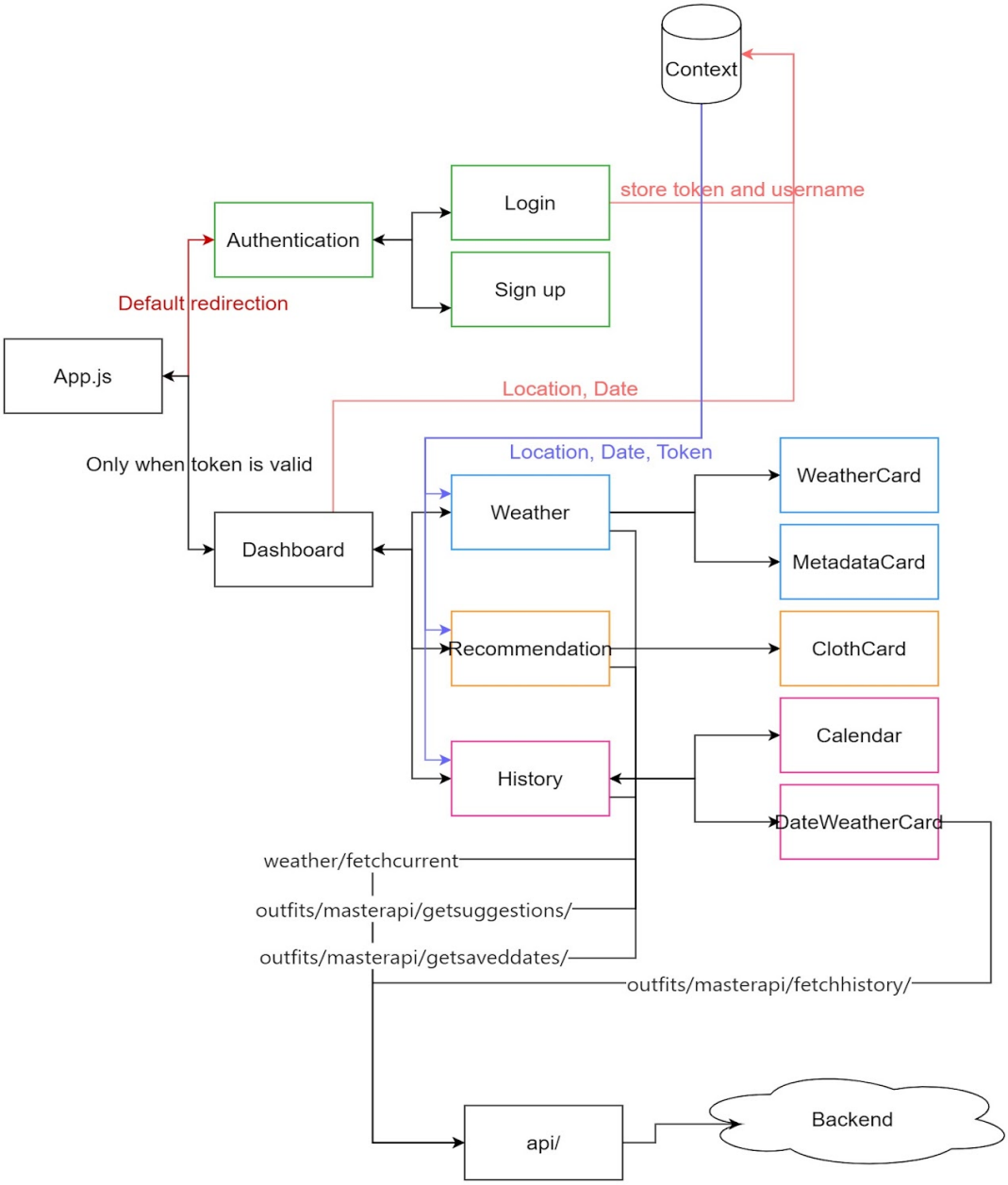

Fig. 1. Frontend Architecture

which can generate interactions such as date selections or month switching with ease.

As Figure 1 indicates, the front-end structure can be divided into two main components, the Authentication componens and the Dashboard component which are coordinated by App.js. Authentication component is the default page users navigated to without logging in. It includes two sub-components, login and signup, which provides UI elements for account login and register. Once users log in, a token will be validated from the backend and stored inside the React Context manager, and the Dashboard component will show up. A date data of today's date and a location data consist of latitude and longitude will be fetched from the user device after the Dashboard component is loaded. These two pieces of data will also be stored inside React Context manager with the token. Inside the Dashboard component, there are three sub-components, Weather, Recommendation, and History, each charges for a critical section

in the main page of Wearther. These three sub-components are dependent on the location, date, and token information from the React Context manager to be properly functional. Thus, there is a loading animation in Dashboard to prevent unnecessary loading before these three pieces of data are ready. To be more specific, Weather component provides users with detailed weather conditions and metadata. In the WeatherCard, today's weather, temperature, rain condition will be presented. And in the MetadataCard, the humidity, precipitation probability, and wind speed will be presented. Furthermore, the Recommendation component will show users the dressing suggestions generated from the backend logic. It includes a module called ClothCard, which shows all the recommended combinations based on today's weather. The final section of the Dashboard component is the History sub-component, which enables users to select previous visited date and check its weather condition with clothing recommendations. In specific, there are two modules in the History sub-component. The Calendar module is implemented using the fullcalendar library and provides users with a interactive calendar where they can switch through months and years, select dates, and re-locate to today. Another module is the DateWeatherCard. Placing below the Calendar module, it offers detailed review of the selected date's weather condition and clothing recommendations so that users can roll back and refer to their previous selections based on weather. From the figure, you may also notice the api calls, which are directed to an organzied folder that contains all the apis into isolated components to maintain a clean and secure coding manner.

B. Backend

The application's backend module – Birdy, was developed using Django, which is a Python-based web framework. Its rapid development, high scalability and enhanced security features made it the most suitable tool for building the backend service for our application. Django provides a facility to create multiple Django 'apps' within the project, each representing a discrete part of it. We utilized this feature to classify our backend services within multiple apps, resulting in a cleaner, highly scalable and reliable code. Figure 2 shows the architecture of Birdy.

An incoming HTTP request is directed to the desired service via the paths specified in urls.py. Upon arrival at the desired service, the request is then checked against the different endpoints within the Django app's urls.py file, and forwarded to the API Methods residing in the app's views.py file.
A Django model defines the structure of stored data in the database, including the field types, their maximum size, default values, selection list options, etc.
ORM (Object Relational Mapper) helps us to make queries to the database, by translating the Python query into SQL. These queries are stored in queries.py, while the models are stored in models.py files.
Serializers help us validate the fields within the incoming and outgoing data, and translate the response object into JSON format.
Below is a description of various services within the backend module:

1) Authentication: Django provides a built-in, ready-to-use session-based authentication system. However, it works only with the traditional HTML request-response cycle, making it unfavorable for our application. To comply with the application's API requirements, a JWT token-based authentication system was created. The newly created JWT service offers a more secure and faster authorization to users.This helped us offer services within the website based upon user's authorization class - registered user or admin.

2) Outfit Master: This is the core service of the application. Provided a temperature value, this service performs computations and outputs outfit suggestions in the response. *Suggestion Algorithm*
An algorithm was designed to compute suggestions based upon the available pool of products in the database. Each product is associated with a certain weather value. These products were fetched based upon their individual weather value, to add up to the difference between the ideal temperature and supplied temperature. Results were further classified into four categories :

1) Bottom
2) Top must
3) Top optional 1 and 2

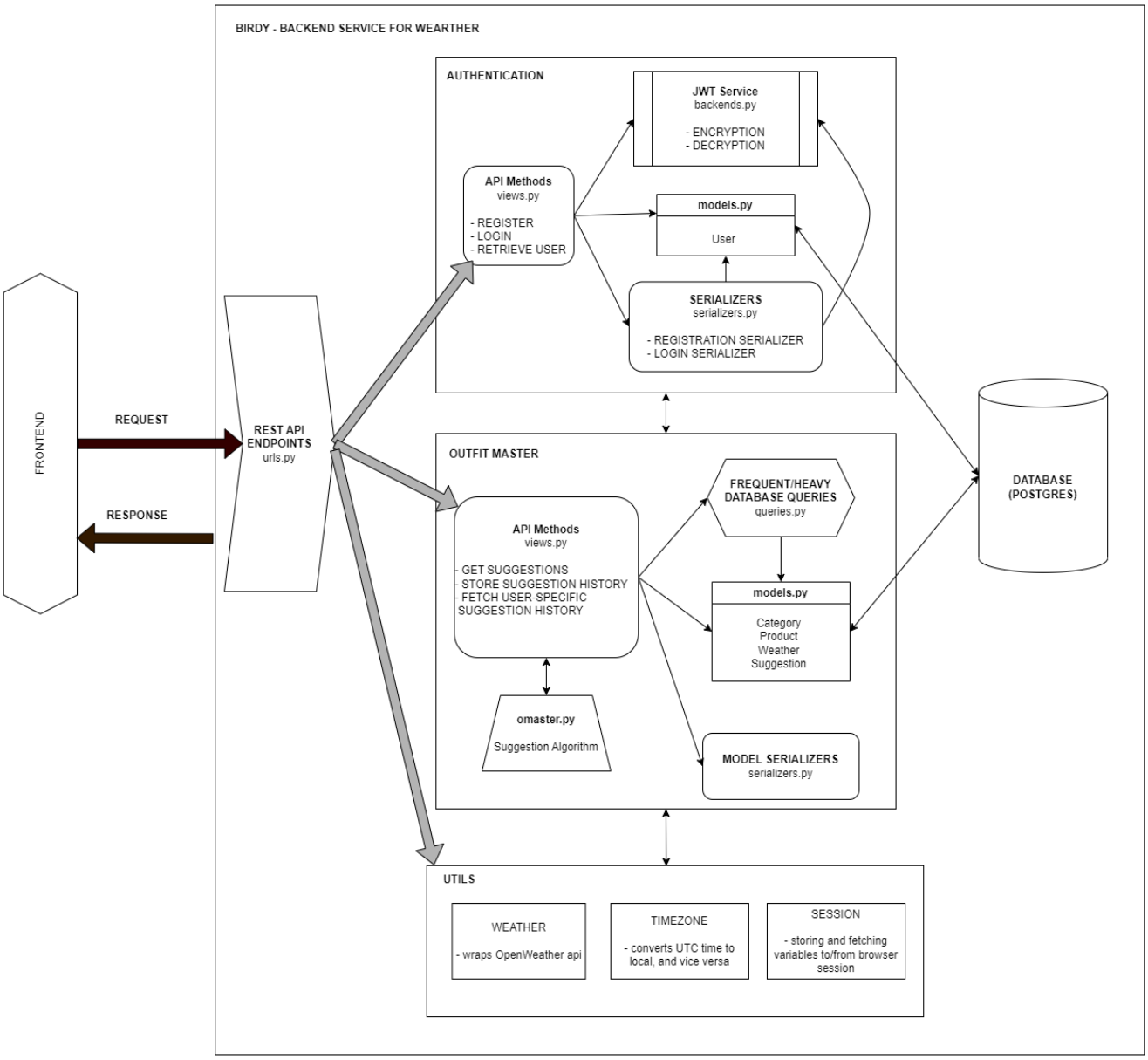

Fig. 2. Backend Service Architecture

Apart from the above mentioned core services, mentioned below are the utilities that serve as helper functions throughout the service, and also wrap third-party services.

3) Weather Utility: To fetch real-time weather data for a given location, we made use of OpenWeather, which provides an API to fetch data with very low latency. This API takes input as the user's location coordinates - latitude and longitude and sends the current weather as a JSON response. A helper function was developed to filter the weather data, as per front-end's requirements, and later associated with a separate custom endpoint.

4) Timezone Utility: This translates UTC timestamp received from the OpenWeather API response, to the user's local time zone in HH::MM:SS format.

5) Session Utility: This provides endpoints to store and fetch variables from the current user's browser session. This helps in passing data as variables, through different services within different

pages.

6) Testing and Code Security: The API endpoints were tested using Postman. To ensure desired output of individual helper functions and services, unit tests were written. Black and flake8 were used as pre-commit linting tools.

IV. Outcome

A. Cloud Architecture

The cloud architecture of our system is shown in figure 3. We are using OCI as our main cloud service provider because they offer the most free resources. We provisioned a VM, an Object Storage, and a Container Registry from the cloud provider. In

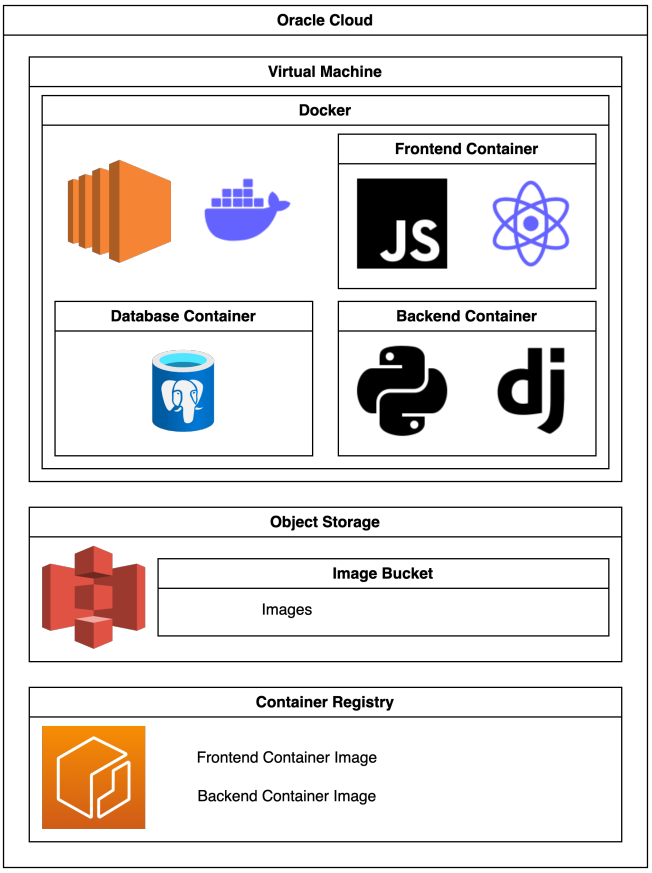

Fig. 3. Cloud Architecture

our virtual machine, we are running a Docker host that runs our containerized applications. In total we are using three containers. One for front-end, one for backend, and one for the database. The frontend container is running the react application that renders the website and handles user interaction. The backend container is running the Django application that provides user authentication, and data access operations. The database container is used by the backend service to store data.

The object storage bucket is used to host static image files to the user.

The container registry is used to store container image to be deployed to the docker host.

B. Deployment Method

We decided to containerize all of our services because that is the best way to provide environment idempotent deploy packages. A separate Dockerfile for the backend and the front-end is used to build the image separately. In our workflow, we develop our service in personal computers. When the codebase is ready for deployment, we build the container image on the personal computer and push them to the container registry, then from the virtual machine, we pull the image from the container registry and deploy it to the docker host. A challenge we encountered here is that the docker host is an ARM architecture machine which is different from what people have(x86). So building multi-arch images locally is a time-consuming process because software emulate is needed to build ARM images.

C. Runtime

1) Frontend Runtime: The front-end is implemented in the react framework and the project build to static files. Any service that can host can be used as the production server. In our project, we choose to use Nginx to serve the static files because it is the most widely used open source web server in the world.

2) Backend Runtime: The backend is implemented in Django with Python. It is not compiled but directly executed so the choice of a production server is more limited. After doing research, we choose gunicorn as the production server as that works seamlessly with the Django framework.

3) Database Runtime: We chose postgresql as our database service and we ran one instance of it directly using the image from Docker Hub. It is an open-source database that is widely used by the industry.

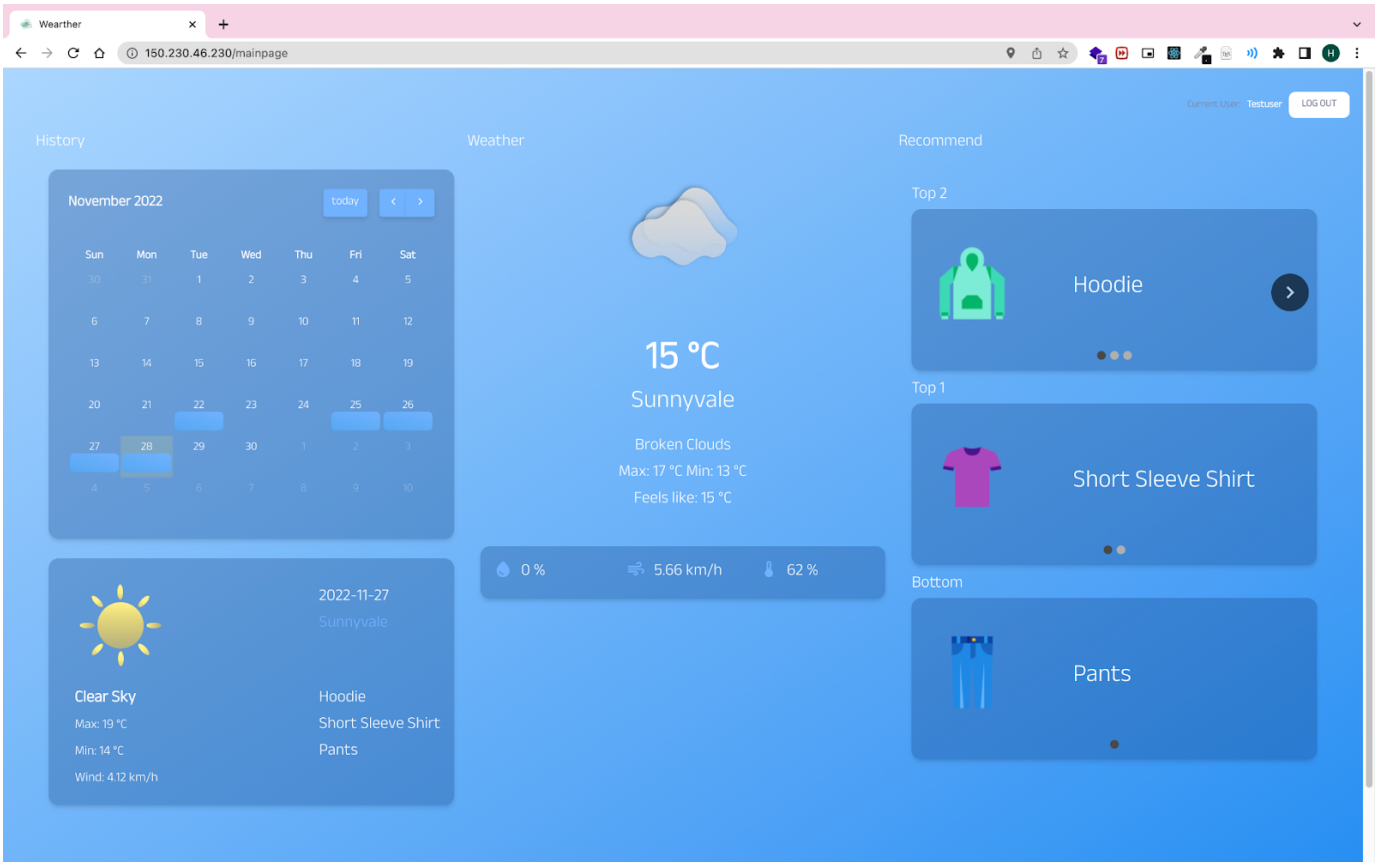

Fig. 4. Outcome website

D. Performance

1) Image size: The container image size is shown below

- Frontend: 33.8 MB. The frontend container image is very small because it only contains a few compiled static file and a nginx server.
- Backend: 1.01 GB. The backend container image is very large compared to the frontend image because it contains a full python runtime with lots of python packages and system packages installed.
- Database: 359 MB. The database image is an optimized image from the postgresql team.

2) Memory Consumption:

- Frontend: 3.96 MB. The frontend is only serving static files so it is using a small amount of memory.
- Backend: 55.57 MB. The backend service is more complex so it is using more memory.
- Database: 22.27 MB. The database service is complex in nature so it is using more memory.

V. Analysis and Future Work

A. Shortcomings

- The application does not provide a facility to switch between Fahrenheit/Celsius temperature options.
- The application currently does not consider users' suggestion history to predict future suggestions.

B. Improvement in the future

- Need to design a front-end module that would convert the existing Celsius temperature to Fahrenheit
- Need to implement a machine learning module that would learn and train its model from the past user choices and make suggestions accordingly.
- The application can incorporate a marketplace where the user may have an option to buy products right from suggestions list, via redirection to vendor's website.

VI. Conclusion

We successfully implemented all of the designed features and architecture as described in the proposal, with minimal modifications. These features are real-time weather checking, weather-based clothing recommendations, and account-based history reviewing. Furthermore, we achieved our baseline to create a modern-looking, scalable, and reliable web application that provides users with inclusive and adaptable outfit options based on informative real-time weather conditions. The codebase follows OOPS paradigm, making it easier to incorporate additional new features within the application.

VII. Github Links

A. *Frontend*

https://github.com/HarveyLijh/wearther-frontend

B. *Backend*

https://github.com/pikulkarni7/Wearther-Birdy

C. *Cloud*

https://github.com/Yuehanc/COEN241-Infra

References

[1] "Clothes forecast," Clothes Forecast - Provides information on appropriate clothes. [Online]. Available: https://clothesforecast.com/. [Accessed: 06-Dec-2022].

[2] "Daily dress me," Daily Dress Me. [Online]. Available: https://dailydressme.com/. [Accessed: 06-Dec-2022].

[3] "Overview: Material dashboard react @ creative tim," Creative Tim. [Online]. Available: https://www.creative-tim.com/learning-lab/react/overview/material-dashboard/. [Accessed: 07-Dec-2022].

[4] "Weather fit: IOS app that tells you what to wear," Weather Fit: iOS App That Tells You What to Wear. [Online]. Available: https://weatherfit.com/. [Accessed: 06-Dec-2022].

[5] "The weather wardrobe," Snafu, 21-Jun-2020. [Online]. Available: http://snafuapp.com/. [Accessed: 06-Dec-2022].

[6] "Weatherproof - what to wear? - apps on Google Play," Google. [Online]. Available: https://play.google.com/store/apps/details?id=com.changemystyle.weatherproof [Accessed: 06-Dec-2022].