UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Internet Software Architecture (ISA) (4CS017)

Report Writing

Student Name: Naomi Thing

Student ID: 2332244

Group: L4ACG4

Module Leader: Mr. Bishal Khadka

Submitted to: Mr. Pradeep Mani Dixit

Submitted Date: 2023/05/21.

Acknowledgement

I would like to express my sincere gratitude to the entire team involved in the “Internet Software Architecture” module for their invaluable guidance throughout the process. Mr. Pradeep Mani Dixit, our lecturer and tutor, deserves special recognition for his ability and aid in the development of our weather app. This module has significantly improved our skills in HTML, JavaScript, PHP, and database management, allowing us to expand our capabilities in these areas. I would also like to thank Herald College Kathmandu for providing us with an exceptional learning platform that has greatly contributed to our knowledge and understanding of this field.

Table of Contents

1. Weather App

1.1 Prototype 1

We were tasked with creating a weather web application that would retrieve weather data from an API key for the first prototype. The goal was to create a functional application that displays real-time weather conditions, including temperature, humidity, wind speed, precipitation, and other pertinent data. Our guidelines emphasized the importance of ensuring that the application successfully retrieves data from the API and effectively displays current weather conditions to users.

To begin developing the weather app, I concentrated on front-end design using HTML and CSS. I decided to make a container div that would hold all the necessary weather information. I used flexbox's flexible layout capabilities to effectively position and arrange elements within the HTML structure. I tried to achieve a visually appealing and responsive design of the weather application's interface by using flexbox features.

I continued to develop the back-end functionality of the weather app using JavaScript after I finished designing the front-end. To retrieve weather data, I integrated the open-weather API into the application. I saved the fetched weather data in separate variables based on their respective values, such as temperature, humidity, and wind speed, in the back-end scripting. This enabled me to display the values of these variables dynamically in the front-end. I used various conditions to improve the overall design of the site and to add visual elements. These conditions enabled the weather app to change the background color and weather icon based on Open Weather API weather conditions. I attempted to provide users with a more immersive and visually informative experience by using these conditions, where the appearance of the application would reflect the current displayed weather conditions.

1.2 Prototype 2

The goal of prototype 2 was to prove server-side caching by implementing a server-side method of retrieving and storing weather data. We were told to pull the data from PHP, save it in a MySQL database, and then extract the weather data for display. The task specifically asked us to present current weather data as well as weather data from the previous seven days. To carry out this, we used PHP to create a server-side script that retrieves weather data. The data was then saved in a MySQL database for easy retrieval and caching. We perfected the weather application's performance and response time by using server-side caching.

Overall, prototype 2 emphasized server-side caching via PHP and MySQL integration, allowing the extraction and display of current weather data as well as historical weather data for the previous seven days.

I began by changing the layout of the weather app's front-end design for prototype 2 to accommodate the display of weather data for the previous seven days. I added a new container specifically designed to display this historical weather data. I included weather data. I included elements in the new container to display the day names, minimum and maximum temperatures. Users would be able to easily show and understand the weather of the previous week if these elements were combined. This feature not only improved the user experience by supplying a comprehensive overview of weather trends, but it also supplied valuable information on temperature and weather conditions changes over time.

Overall, this change to the weather app's front-end design enabled an efficient presentation of weather data for the previous seven days, including basic information such as day names, and temperature ranges.

I began by writing PHP scripts to create a table in the database and fetch weather data from the API, following the instructions for fetching data in PHP and saving it to a MySQL database. I extracted the required weather data from the loaded data and entered it into a database table. I read this data from a database table and listed it in a hidden div-box to make it available in the front-end.

I extracted the weather data and organized systematically using JavaScript and DOM selectors. I displayed the data in front of the weather app now that the data was properly managed. The data from the hidden div-box was extracted and presented to the user in a structured and visually appealing format using JavaScript.

1.3 Prototype 3

We were asked to prove the use of a local API repository in Prototype-3 to reduce excessive API access and make the site available offline. Except for some other condition changes made via JavaScript, there have been no changes to the front-end design since Prototype-2. I chose to create an offline backup site for the backend.

To carry out this, I created a backup page that displays the most recently searched data in the weather app. Using service workers, I caught offline backup HTML, CSS, and JavaScript files. When the network connection is lost, the weather app displays a cached backup offline page.

I used a local API repository to ensure weather data availability even when I was offline. I saved the weather information from the most recently searched areas to local storage. Instead of using an API to retrieve weather data, the site now uses locally stored data when available.

By incorporating these features, the website can run smoothly without relying too heavily on API access, and users can still access weather information even when they are not connected to the internet.

1.4 UML Diagram of Prototype 3

1.4.1 Deployment Diagram

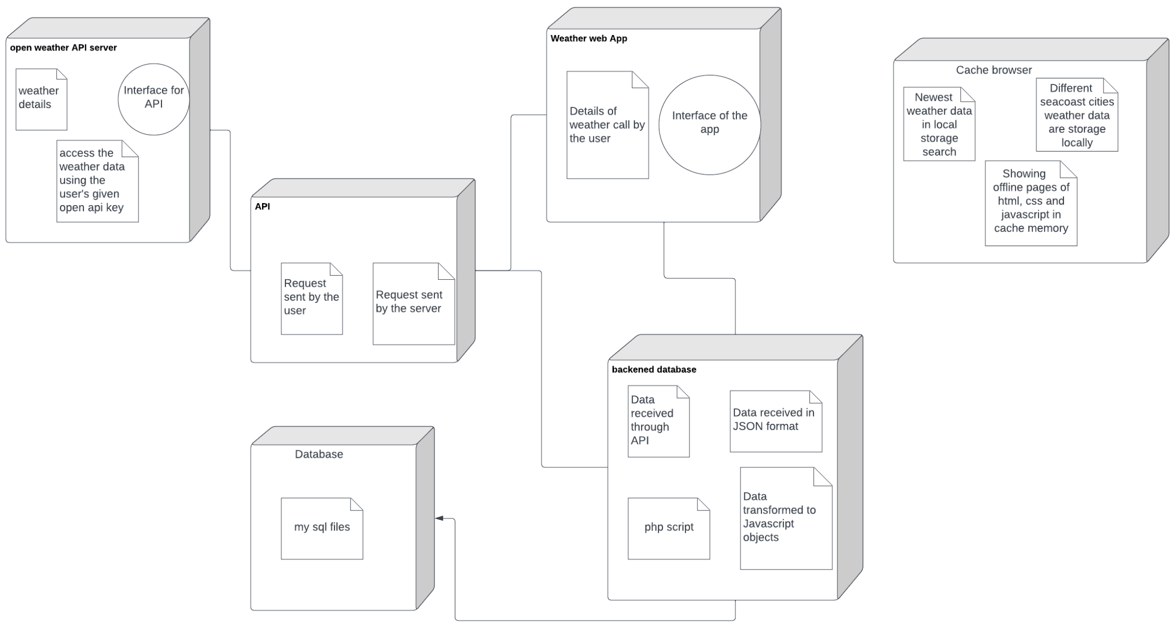
open weather API server
weather details
Interface for API
access the weather data using the user's given open api key
API
Request sent by the user
Request sent by the server
Weather web App
Details of weather call by the user
Interface of the app
Cache browser
Newest weather data in local storage search
Different seacoast cities weather data are storage locally
Showing offline pages of html, css and javascript in cache memory
backened database
Data received through API
Data received in JSON format
php script
Data transformed to Javascript objects
Database
my sql files

Figure 1 Deployment Diagram

1.4.2 Activity Diagram

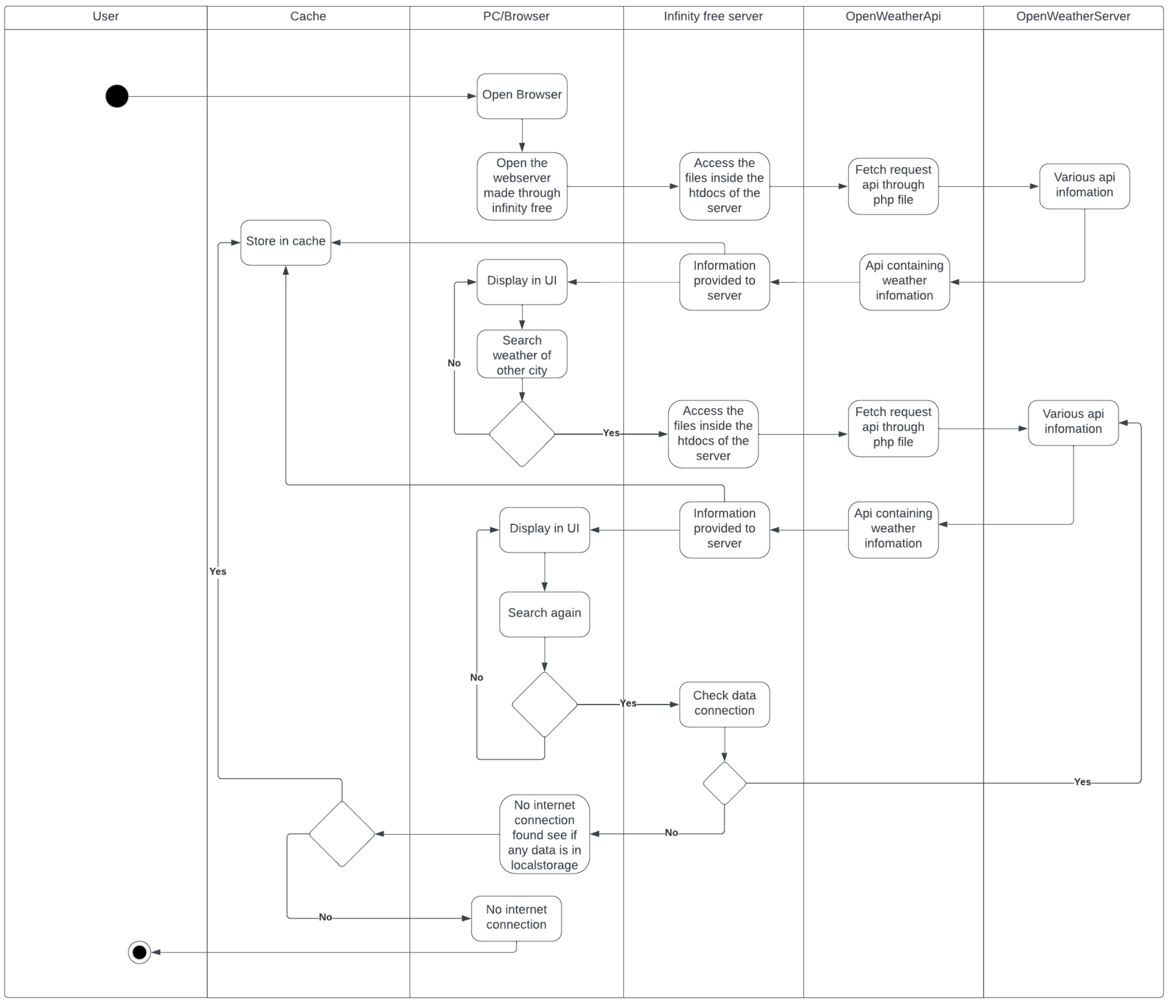
User
Cache
PC/Browser
Infinity free server
OpenWeatherApi
OpenWeatherServer
Open Browser
Open the webserver made through infinity free
Access the files inside the htdocs of the server
Fetch request api through php file
Various api infomation
Store in cache
Display in UI
Information provided to server
Api containing weather infomation
Search weather of other city
No
Yes
Access the files inside the htdocs of the server
Fetch request api through php file
Various api infomation
Display in UI
Information provided to server
Api containing weather infomation
Yes
Search again
No
Yes
Check data connection
Yes
No internet connection found see if any data is in localstorage
No
No internet connection
No

Figure 2 Activity Diagram

1.4.3 Sequence Diagram

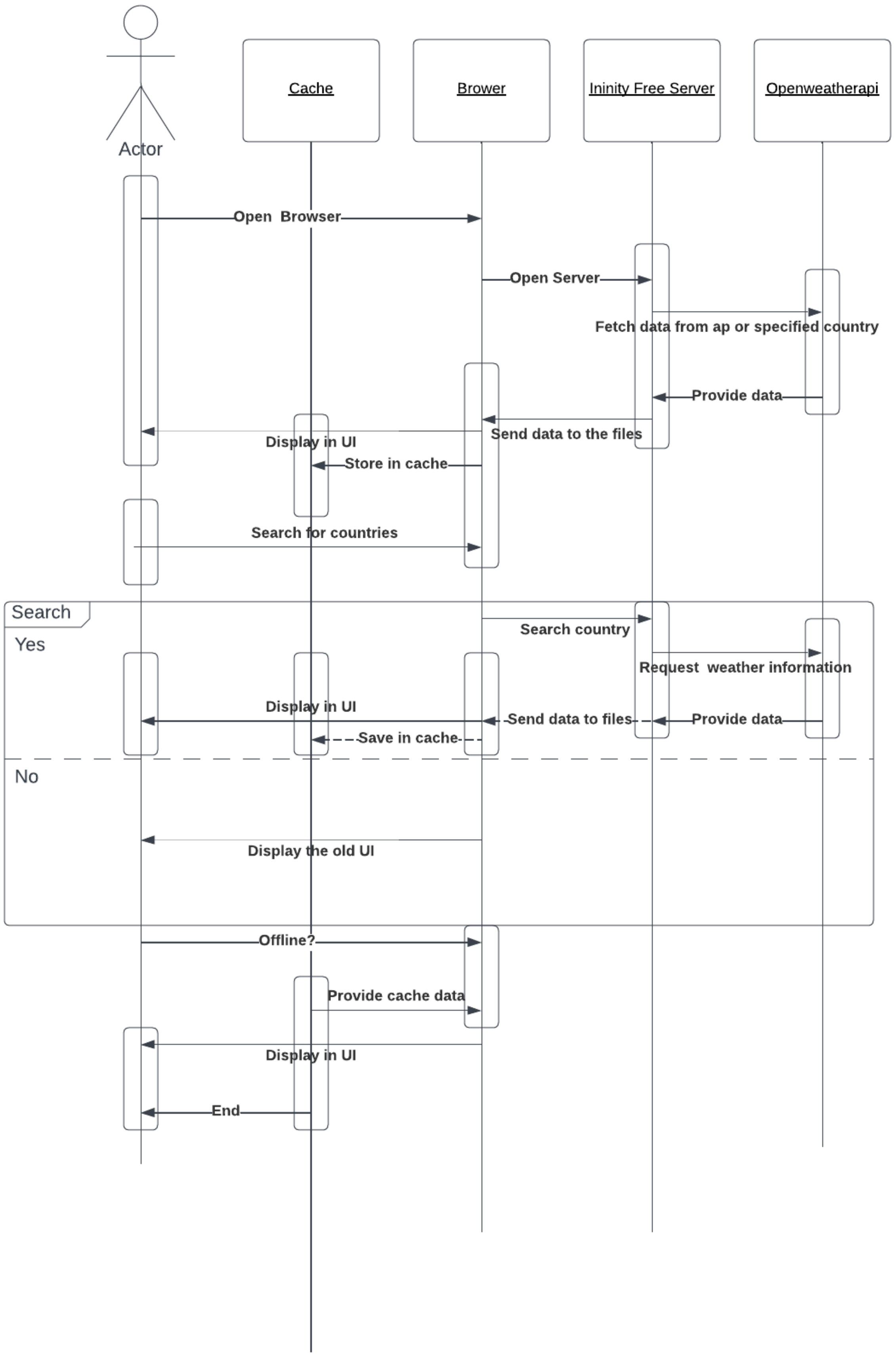

Figure 3 Sequence Diagram

1.5 Link to the final Prototype (3)

Link: http://cafe2.epizy.com/