Team 1:

Uveimar Sandoval: usandova@gmu.edu

Farith Bascope: fbascope@gmu.edu

Daniel Curtis: dvurtis7@gmu.edu

Rajeep Karki: rkarki6@gmu.edu

Sammy Miller: smille77@gmu.edu

Omar Naseem: [onaseem@gmu.edu](mailto:onaseem@gmu.edu)

Joshua Reyes: jreyes36@gmu.edu

October 29, 2023

CS 321: Software Engineering

Professor Wes Masri

GTA: Bhargavi Janga

Project Deliverable 3 – Software Architecture

**Software Architecture Diagrams**

* High Level View
* Frontend Components
* Backend Components

A diagram of a software development process

Description automatically generated

A diagram of a website

Description automatically generated

A diagram of a database

Description automatically generated

**Component Descriptions**

Frontend Website (HTML, CSS, JavaScript with ApexCharts):

* **Role**: The frontend website is responsible for providing a user interface for interacting with the data and presenting it in a visually appealing manner using ApexCharts.
* **API**: The frontend website is built using HTML, CSS, and JavaScript. It incorporates ApexCharts to render data visualizations. JavaScript is used to make asynchronous requests to the Flask backend server, fetching data from the database, and updating the charts without the need for full page reloads. HTML provides the structure and content of web pages, CSS styles the content, and JavaScript enhances interactivity.

Backend Server (Python):

* **Role**: The frontend website is built using HTML, CSS, and JavaScript. It incorporates ApexCharts to render data visualizations.JavaScript is used to make asynchronous requests to the Flask backend server, fetching data from the database, and updating the charts without the need for full page reloads. HTML provides the structure and content of web pages, CSS styles the content, and JavaScript enhances interactivity.
* **API**: The backend exposes a set of HTTP endpoints (API) for the frontend to interact with. These endpoints include functionalities such as querying the database for specific data, processing and transforming data, and sending it back to the frontend.

Database:

* **Role**: The database stores the data that your website needs to display in charts. It is in a relational database, MySQL. MySQL is used for storing and managing the data. The Flask application interacts with the MySQL database to retrieve and store information. It can use SQL queries within our Python code to fetch data from the database and insert new data when scraping information from external websites.

Web Scraping (Beautiful Soup, Python):

* Beautiful Soup, a Python library, is used for web scraping. It fetches data from external websites, parses the HTML, and extracts the relevant data. This component can be scheduled as a periodic task or triggered by certain events to keep the database up-to-date with the latest data from external sources.
* In terms of architecture behavior, the web scraping component periodically scrapes data from external websites and inserts it into the MySQL database. This can be scheduled using other tools or manually coded in. The Flask server provides an API to the frontend for data retrieval and other actions. The frontend uses JavaScript to make requests to the Flask server, which in turn interacts with the MySQL database to retrieve data. ApexCharts within the frontend dynamically renders data visualizations for the user. This architecture ensures a responsive web application that fetches, stores, and presents data efficiently to the end user while keeping the database updated with the latest information from external websites.

**Important Non-functional Requirements**

Reliability:

* **Why it's important**: Reliability ensures that the website’s functions work consistently and without frequent failures, providing a positive user experience.
* **How the architecture helps**:
  + Using a well-established Python web framework like Flask, which is known for its reliability and stability.
  + Employing proper error handling to identify and resolve issues quickly.
  + Storing data in a reliable database system like MySQL, which offers data integrity and fault tolerance features.

Performance:

* **Why it's important**: Performance affects the speed and responsiveness of our website, which is critical for user satisfaction.
* **How the architecture helps**:
  + Separating frontend and backend allows to scale each component independently for improved performance.
  + ApexCharts provides interactive data visualizations without compromising performance.

Maintainability:

* **Why it's important**: Maintainability ensures that the website can be easily updated, extended, and maintained over time without excessive effort or cost.
* **How the architecture helps**:
  + Clear separation of frontend and backend components makes it easier to update or replace one without affecting the other.
  + Proper documentation and well-organized code in Flask and JavaScript will facilitate maintenance.
  + Using a Python-based stack promotes code maintainability and reuse.

Usability:

* **Why it's important**: Usability ensures that the website is user-friendly and easy to navigate. It’s crucial for users of Fantasy Fanatic to get accustomed to our website easily.
* **How the architecture helps**:
  + The separation of concerns between frontend and backend enables frontend developers to focus on user interface and experience design.
  + Proper data visualization with ApexCharts enhances data comprehension and user engagement.

Responsiveness:

* **Why it's important**: A smooth user experience is needed, especially when interacting with data visualizations.
* **How the architecture helps**:
  + Efficient server-side processing in Flask and optimized frontend code ensure that users receive timely responses.
  + ApexCharts provides interactive and responsive data visualizations for a more engaging user experience.

**Deliverable 3 Contributions:**

* Joshua Reyes: Software Architecture Diagrams
* Omar Naseem: Component roles and APIs
* Joshua Reyes: Non-functional requirements