InfoPulse Software System (IPSS)

v1.0

Software Architecture and Design

Author: Joshua Bonner

Contact: jbb5882@psu.edu

Master of Software Engineering

SWENG 894 (SUM 2024) – Capstone Experience

1. System Overview

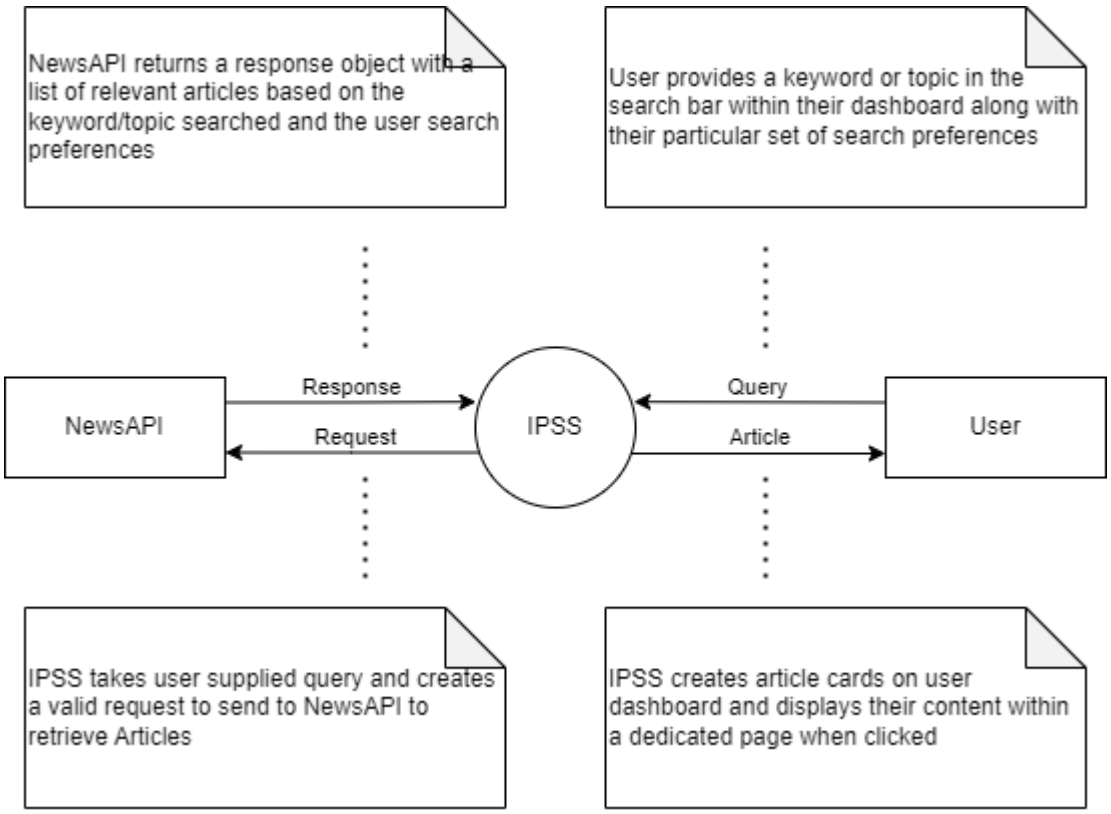
a. Purpose

This document will serve as a reference point for the architectural design decisions made throughout the development of the InfoPulse Software System (IPSS). It will provide context around how the system interacts with various components and its third party APIs. In addition, it will address how the nonfunctional requirements are satisfied by the architectural design of the system.

b. Stakeholder Use

Stakeholders can utilize this document by verifying their requirements are satisfied from an architectural design standpoint. As it is reviewed, it would be beneficial to annotate questions not answered by the specification, which engineers can then answer throughout the development process in keeping with Agile principles.

c. System Context Diagram



2. Architecture Overview

a. System Scope

IPSS will feature user authentication, personalized news feeds, search functionalities, and secure data handling. The GUI will be built using the Vue.js framework leveraging its Composition API, while the backend implementation will leverage Python's FastAPI framework with Pydantic models. User data and search preferences will be stored in a PostgreSQL Database and news article metadata content will be stored in an Elastic Search index for quick retrieval, persisted through user sessions.

b. Overview

IPSS is a web application leveraging the latest in web technologies to provide the user with quick access to the most current news pulled from NewsAPI, one of the most popular sources available to developers with API endpoints to cover any need. Users can utilize this web application to keep themselves up to date on the latest information on a particular topic without the normal distractions, like popups or advertisements, at a quick glance and promptly.

For ease of reference, the Nonfunctional Requirements will be reiterated below:

3.1.1.1 – IPSS shall provide a web application GUI that is intuitive and free of advertisements, following standard UI/UX design methodologies.

3.2.2.1 – IPSS shall load search content within 2 seconds.

3.2.2.2 – IPSS shall maintain latency no greater than 100 milliseconds.

3.2.3.1 – IPSS shall encrypt all sensitive user data in transit and at rest.

3.2.3.2 – IPSS shall provide failover mechanisms with verbose error statements.

3.2.4.1 – IPSS shall provide documentation for both end users and developers.

3. Architecture Specification

a. Functional Responsibilities and Rationale

i. IPSS Backend

Python –

Scripting language primarily used for the IPSS backend implementation. This language was chosen given its ease of use and familiarity with building quick prototype applications. Python is one of the most used scripting languages and

has a significant amount of support and libraries enabling it to build complex software from web applications to AI models.

FastAPI –

FastAPI is a REST API framework known for its speed, making it one of the fastest available. This was the main reason for its selection to meet stakeholder NFRs 3.2.1.1, 3.2.2.2, and 3.2.3.1. FastAPI handles all request/response objects between IPSS and its subsystems.

ElasticSearch (ES) –

ElasticSearch is a powerful open-source search and analytics engine designed for handling large volumes of data quickly and efficiently. It will be leveraged within the IPSS to store article content that can be searched on after it is retrieved from NewsAPI sources. NFRs addressed by the addition of ES are 3.2.2.1 and 3.2.2.2.

PostgreSQL –

PostgreSQL is a powerful open-source relational database management system known for its robustness, extensibility, and standards compliance. It was chosen because of these attributes and to facilitate the relations between a user and their search preferences within the IPSS.

NewsAPI –

NewsAPI is a powerful, easy-to-use RESTful web service that allows developers to access a wide range of news articles and headlines from various sources worldwide. It aggregates news from multiple publishers, providing a unified interface to search and retrieve news content programmatically. NewsAPI will be the primary source of articles served up to the user based on their search preferences and the topic/keyword searched on.

ii. IPSS Frontend

Javascript/Typescript –

Javascript is the primary language used in web application development. Typescript is a superset of the language that provides static typing that enhances error handling and helps to prevent bugs. Additionally, it provides object-oriented structures that can be used to extend upon primitive Javascript types. This language is used to define the frontend logic within the IPSS.

Vue3 Composition API –

Vue is a progressive Javascript framework used for building user interfaces and single-page applications. Vue3 is the latest version of the Vue framework which allows for more flexible and reusable code by enabling developers to group

related logic together in a single function. Vue3 was selected given these attributes that satisfy NFRs 3.1.1.1 and 3.2.4.1.

Vuetify –

Vuetify is a popular open-source component framework for Vue.js that follows the Material Design guidelines by Google. It provides a comprehensive suite of UI components and tools to help developers create beautiful, responsive, and feature-rich web applications. This component library seamlessly integrates with the rest of the IPSS frontend architecture and is used to help satisfy NFR 3.1.1.1.

Pinia –

Pinia is a state management library for Vue.js, designed as a modern and lightweight alternative to Vuex. It leverages Vue 3's Composition API and provides a simpler and more intuitive way to manage and share state across components in a Vue application. Pinia allows IPSS to manage the state of the current user and their search preferences as well as their personalized dashboard with previously searched article content.

Axios –

Axios is a popular, promise-based HTTP client for JavaScript that can be used both in the browser and in Node.js environments. It simplifies making HTTP requests, handling responses, and working with APIs. All IPSS request/response objects on the frontend are handled by Axios.

b. Constraints/Trade-Offs

A great deal of open-source web application technologies are leveraged in the IPSS. As such, there are several potential drawbacks to consider:

- **Complexity in Integration:**
  - Integrating multiple libraries and frameworks can introduce significant complexity. Ensuring that all components work seamlessly together may require additional configuration and custom code.
  - Each library has its own learning curve. Developers need to be proficient with each tool, which can slow down the development process initially.
- **Performance Issues:**
  - Using high-level abstractions provided by libraries like Vuetify or Pinia can sometimes introduce performance overhead, especially if not used optimally.
  - Ensuring efficient data flow between components, especially when dealing with large datasets from ElasticSearch or PostgreSQL, can be challenging.

- **Dependency Management:**
  - Keeping all dependencies up-to-date and compatible with each other can be challenging. An update in one library might introduce breaking changes affecting other parts of the stack.
  - Regularly updating libraries to patch security vulnerabilities is crucial, but this can sometimes introduce bugs or incompatibilities.
- **Maintenance and Technical Debt:**
  - Over time, the accumulation of quick fixes and workarounds can lead to technical debt, making the application harder to maintain and extend.
  - Relying on open-source libraries means depending on the community for updates and support. If a library is abandoned or not well-maintained, it can become a liability.
- **Development and Debugging:**
  - With multiple libraries in use, debugging issues can become complex. Problems might arise at the intersection of different technologies, making it harder to identify and resolve issues.
  - Setting up and configuring tools like Docker for a smooth development workflow requires expertise and can be time-consuming.

C. Prioritized Drivers

The following list of architectural drivers can be used to understand why certain libraries were utilized and how they address stakeholder concerns resolving around NFRs.

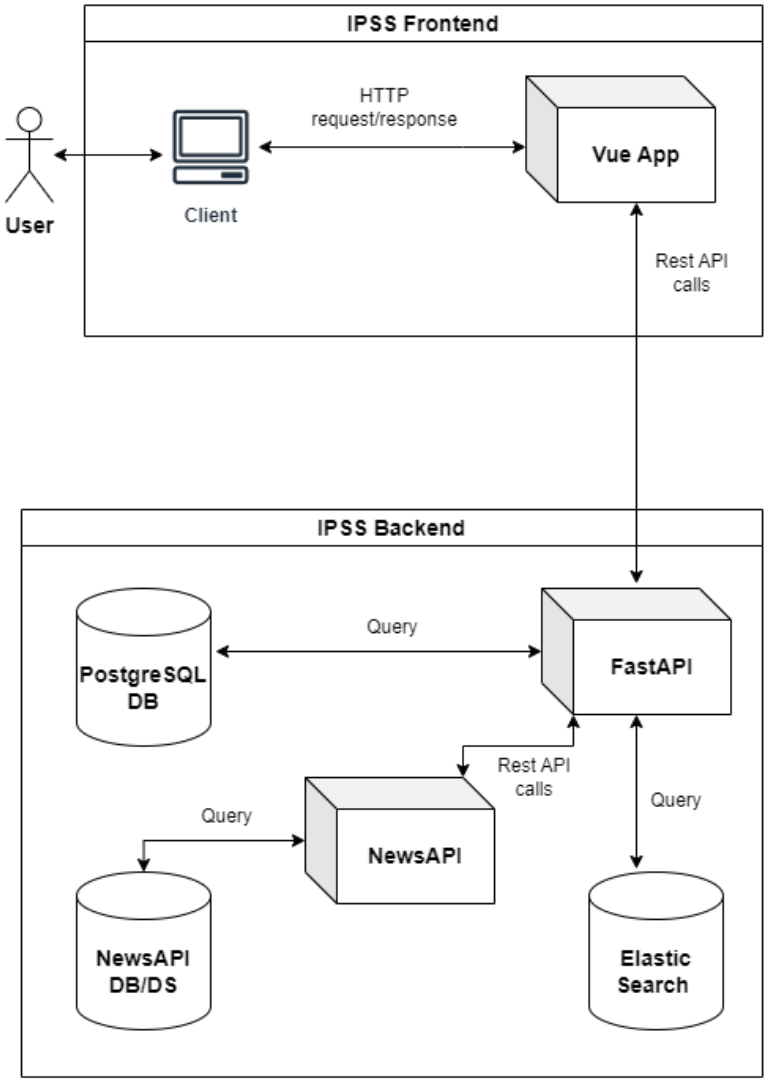
- **Performance:**
  - FastAPI's high performance and asynchronous capabilities, along with the efficient querying capabilities of ES and PostgreSQL are critical in ensuring the adequate performance of the IPSS.
- **Reliability:**
  - Data consistency is crucial in a system that employs many different technologies, especially moving throughout its many forms from request/response objects to models defined in the various databases/datastores. FastAPI and Axios help to facilitate this.
- **Security:**
  - Implementing robust security mechanisms to protect sensitive user data and ensure users have appropriate access rights. FastAPI security libraries are utilized to ensure security throughout the IPSS.

- **Usability:**
    - Usability is a core driver for the IPSS in that it is essential to its overall functionality. Vue3 and Vuetify ensure optimal user experience with its adherence to standard design principles.

4. Architecture Representation Diagrams (high/low)

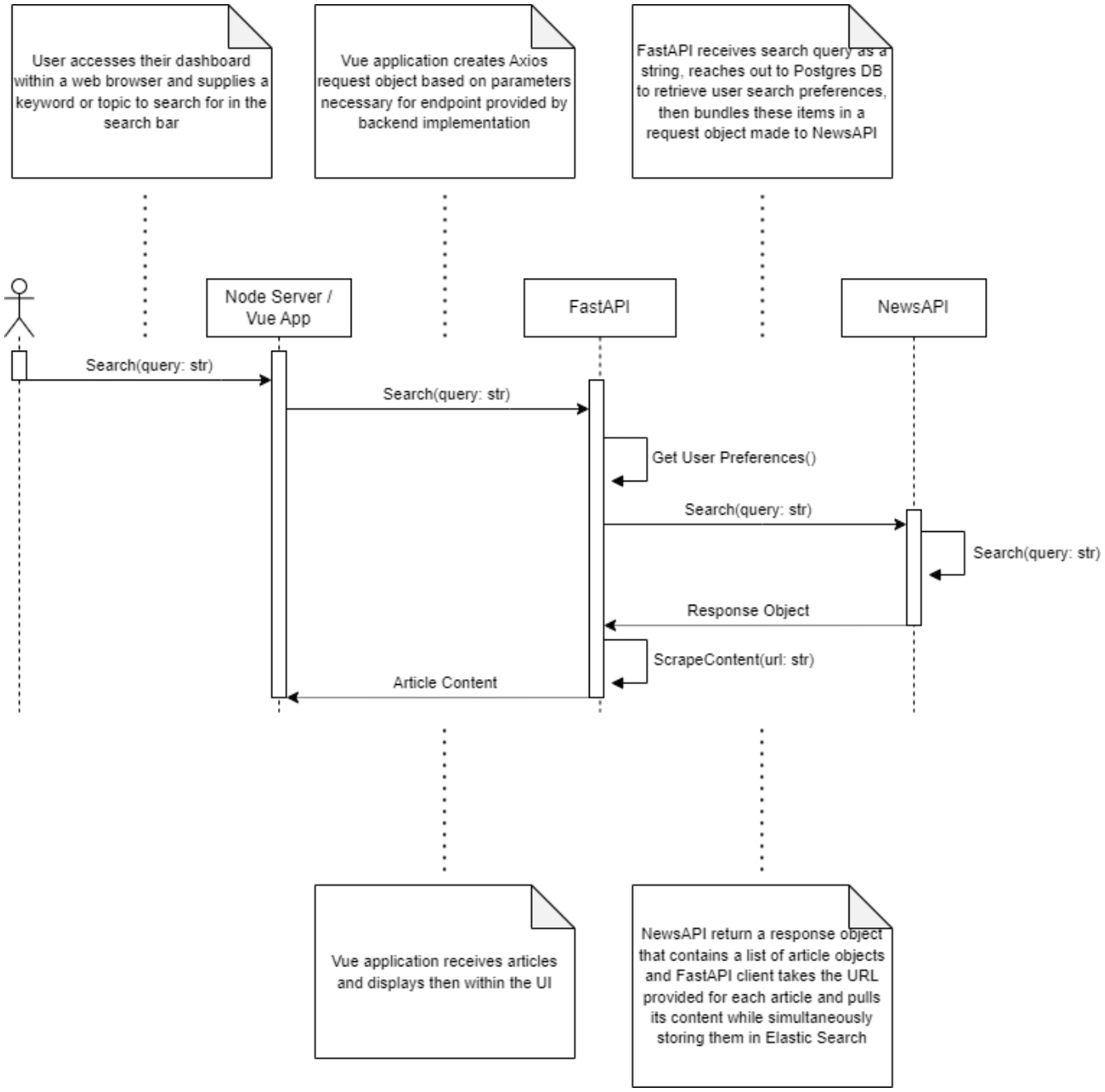
IPSS Architecture Overview

The following diagram illustrates the basic software architecture for the InfoPulse Software System.



- **Web Client** will be whichever browser is utilized by the user, typically this is Google Chrome or Mozilla Firefox.
- Request and response will be facilitated by HTTP through the **Web Client** to the **Vue App** instance.
- **Vue App** will be a standard instance of a Vue3 Application utilizing the Composition API.
- Rest API calls will organize the request/response objects passed to and from the **Vue App** instance to the backend **FastAPI** implementation.
- The **FastAPI** implementation will make Rest API calls to the **NewsAPI** and store content in both a **PostgreSQL** database instance and Elastic Search indexes.
- **NewsAPI** will serve up articles based on the search criteria provided by the user by reaching out to its internal database/datastore instances, named here in this diagram as **NewsAPI DB/DS**
- **Elastic Search** will house all the articles, as JSON objects, returned by the **NewsAPI** for quick reference and to serve up to the front-end **Vue App.**
- **PostgreSQL DB** will store user information regarding previous searches, preferences, and account information.

IPSS Sequence Diagram



IPSS Class Diagram