Anime Streaming Website

Contents

1. Introduction

1. 1 Why this high level design document

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The HLD will:

- Present all of the design aspects and define them in detail
- Describe the user interface being implemented
- Describe the hardware and software interfaces
- Describe the performance requirements
- Include design features and the architecture of the project
- List and describe the non-functional attributes like: o Security
-
    - Reliability
    - Maintainability
    - Portability
    - Reusability
    - Application compatibility
    - Resource utilization
    - Serviceability

1. 2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

1. 3 Defination

Database: Collection of all the information monitored by this system

IDE: Integerated development environment

2. General description

2.1 Product perspective

Creating an anime streaming website involves several key product perspectives to ensure a user-friendly experience, effective management, and smooth operations.

2.2 Problem statements

Fragmented Content Sources: Anime fans struggle to find a centralized platform for streaming, requiring visits to multiple websites for different content.

Limited Search Functionality: Existing platforms offer basic search features, limiting users' ability to discover anime based on specific preferences.

Outdated or Inaccurate Data: Many streaming services rely on manually updated content, leading to inconsistencies and outdated anime information.

Poor User Experience: Multi-source platforms create a disjointed experience, with users navigating through various websites to find their desired content.

Scalability Issues: Handling large amounts of anime data and traffic efficiently is a challenge for platforms with extensive content libraries.

2.3 Proposed solutions

Centralized Streaming Platform: Aggregate anime content from multiple sources into one platform for easy access and seamless streaming.
Advanced Search and Filters: Implement robust search features with filters for genre, popularity, and more, improving content discovery.
Real-Time Web Scraping: Use automated web scraping to fetch up-to-date and accurate anime data, ensuring content is current.
User-Friendly Interface: Design a cohesive and intuitive UI with smooth navigation and an enhanced user experience for streaming.
Scalable Infrastructure: Build a scalable backend using Angular and cloud services to handle large traffic and vast content libraries efficiently.

2.4 Further improvements

Enhancements for the anime streaming website include personalized recommendations, history feature, like,favourites features, a mobile app, loyalty programs, two-factor authentication, cloud infrastructure for scalability, integration with delivery services, and eco-friendly options. These features improve user experience, security, and overall operational efficiency.

2.5 Technical requirements

Angular Framework: Use Angular for building a responsive and dynamic frontend with efficient data binding and component-based architecture.

Web Scraping Tools: Implement Python-based web scraping libraries (e.g., BeautifulSoup, Scrapy) to extract anime data from multiple websites.

Backend API: Develop a RESTful API (e.g., using Flask or Node.js) to handle requests and serve scraped data to the frontend.

Database Integration: Use a database (e.g., MySQL, MongoDB) to store scraped anime data, user profiles, and viewing history.

Cloud Hosting & Deployment: Deploy the application on cloud platforms (e.g., AWS, Render) for scalability, performance, and availability.

2.6 Data requirements

Anime website which you want to scrape

2.7 Tools used

Angular: A powerful frontend framework used to build the responsive and dynamic user interface, providing seamless interaction and navigation for the anime streaming platform.

Python (BeautifulSoup/Scrapy): These web scraping libraries are used to collect real-time anime data from various external websites, ensuring accurate and up-to-date content.

Node.js/Express: Used for building the backend server, handling API requests, and facilitating communication between the frontend, database, and external data sources.

MongoDB/MySQL: NoSQL or relational databases used to store anime metadata, user data, and other relevant information for efficient querying and scalability.

AWS/Render: Cloud platforms used for deploying and hosting the application, ensuring high availability, load balancing, and the ability to scale based on user demand and traffic.

2.8 Constraints

Legal and Copyright Issues: Web scraping may violate the terms of service of some websites, and distributing copyrighted anime content without proper licenses can lead to legal challenges.

Data Accuracy and Consistency: Web scraping relies on the structure of external websites, which may change over time, leading to inconsistencies or data extraction failures.

Performance and Scalability: Handling large volumes of data and high traffic requires efficient backend systems and databases to ensure smooth performance and scalability under heavy loads.

API Rate Limits: Web scraping may be restricted by rate limits or IP blocking from external websites, impacting the frequency and reliability of data extraction.

Complexity in Maintenance: Maintaining a web scraping system can be complex due to frequent website updates, requiring constant monitoring and adjustments to ensure data accuracy and platform functionality.

2.9 Assumptions

Availability of Scraping Sources: It is assumed that the websites providing anime data will allow scraping or provide consistent data formats, ensuring smooth data extraction.

Stable Data Structure: The project assumes that the structure of the scraped websites will remain relatively stable, reducing the need for frequent changes in the scraping logic.

Legal Permissions: It is assumed that all required legal permissions and licensing for using and displaying the anime content are obtained, either through scraping or partnerships.

User Internet Connectivity: The project assumes that users will have sufficient internet connectivity to stream anime content without significant buffering or delays.

Scalable Hosting: It is assumed that the cloud hosting infrastructure (e.g., AWS, Render) will be able to handle increased traffic and large amounts of content without performance degradation.

3.2 Event log

User Activity Logging: Track user interactions such as logins, searches, and content viewing to provide insights into user behavior and preferences.

Error Logging: Capture and log errors or issues that occur on the website, including failed data scraping, API errors, or frontend performance issues.

Data Scraping Logs: Maintain logs of web scraping activities, including successful and failed scraping attempts, to monitor the reliability of external sources and ensure data accuracy.

Performance Monitoring: Log performance-related data such as page load times, server response times, and streaming quality to identify potential bottlenecks or areas for optimization.

Security Events: Log potential security events, including unauthorized access attempts, to monitor and respond to any suspicious activities or vulnerabilities in the system.

3.3 Error handling

Data Scraping Failures: Implement error handling for failed web scraping attempts, with retry mechanisms or fallback solutions in place if the target website structure changes or if there are network issues.
API Response Errors: Handle errors from backend API requests, including timeouts, bad responses, or server failures, and display appropriate error messages to the user with options to retry.
Frontend Errors: Catch JavaScript errors in the Angular frontend, such as undefined variables or failed component renders, and provide user-friendly fallback UI to ensure a smooth experience.
Authentication and Authorization Failures: Ensure proper handling of login or registration failures, such as incorrect credentials or expired sessions, with clear error messages and guidance for users.
Database Connection Issues: Implement error handling for database connectivity problems, with retries or fallback mechanisms to ensure minimal downtime and data retrieval errors for users

3.4 Performance

Lazy Loading: Implement lazy loading in Angular to load content and components only when required, reducing initial load times and improving overall performance.
Efficient Data Caching: Cache API responses and scraped data locally to minimize the number of repeated requests to the backend or external sources, improving the user experience and reducing server load.
Optimized Images & Media: Use techniques like image compression and adaptive streaming to ensure efficient delivery of media content, enhancing loading times and minimizing bandwidth usage.
Server-Side Pagination: Implement pagination or infinite scrolling on large datasets (e.g., anime titles) to load content in smaller chunks, improving performance and reducing memory usage.
Content Delivery Network (CDN): Leverage a CDN to distribute static assets like images, CSS, and JavaScript across multiple servers, improving load times for global users by reducing latency.

3.5 Reusuability

Component-Based Architecture: Use Angular's component-based architecture to create reusable UI components (e.g., video player, search bar, card view), ensuring consistent design and easy maintenance.
Service Layer for API Communication: Implement reusable Angular services to manage API calls for anime data, user profiles, and streaming content, ensuring the same logic is applied across multiple components.
Reusable Scraping Modules: Develop reusable Python scraping modules or functions to collect data from different anime sources, making it easy to add new data sources with minimal changes to the core scraping logic.
State Management with NgRx: Use state management (e.g., NgRx) to centralize the application's state, allowing for reusable, consistent, and easily testable application state management across various features.
Customizable UI Components: Design flexible and customizable UI components for different screen sizes (e.g., responsive grids, adaptable banners), allowing easy integration and reusability across different sections of the website.

3.6 Application compatibility

Cross-Browser Support: The website will be compatible with modern browsers such as Chrome, Firefox, Safari, and Edge, ensuring a wide user base.
Responsive Design: The site will be optimized for mobile, tablet, and desktop devices, offering a consistent user experience across screen sizes.
API Compatibility: The backend API will be designed to work seamlessly with the Angular frontend, ensuring smooth communication and data flow.
Third-Party Integrations: Integration with third-party services (e.g., video streaming, social sharing) will be compatible and easy to manage.
Cross-Platform Accessibility: The website will be accessible across different operating systems, such as Windows, macOS, and Linux, with no issues.

3.7 Resource utilization

Efficient Data Caching: Use local storage and caching strategies to reduce unnecessary API calls and optimize server resource usage.
Load Balancing: Implement load balancing techniques to distribute traffic evenly across servers, improving performance and minimizing resource consumption.
Optimized Media Streaming: Use adaptive streaming protocols to adjust video quality based on the user's internet speed, minimizing bandwidth consumption.
Memory Management: Efficient handling of large datasets (e.g., anime lists) to ensure minimal memory usage and reduce lag or crashes.
Cloud Resource Management: Leverage cloud-based solutions for storage and computing resources, allowing scalable resource utilization as user demand grows.

4. Conclusion

The anime streaming website, built with Angular and web scraping, will be highly compatible across various platforms and browsers. The architecture will ensure efficient resource utilization through optimized caching, media streaming, and scalable backend solutions. This approach aims to deliver a seamless and responsive user experience while managing resources effectively.