|  |
| --- |
| **SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING**  **(An autonomous Institution affiliated to Anna University)**  **OMR, Kalavakkam, Chennai**  [**www.ssn.edu.in**](http://www.ssn.edu.in)  **—---------------------------------------------------------------------------------------------------------** |

**FandomHub: Multi-Tenant Fandom Content Management System**

**Version 2.0**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | Dr. R. Kanchana  Dr. S. Sasirekha |  |  |
| **Role** |  |  |  |
| **Signature** |  |  |  |
| **Date** | 3rd Nov 2023 |  |  |

1. ***Objective***

It is proposed to develop a versatile and user-centric Multi-Tenant Fandom Content Management System (FandomHub) that empowers users to create, share, and engage in fan-centric content. The objectives are listed below:

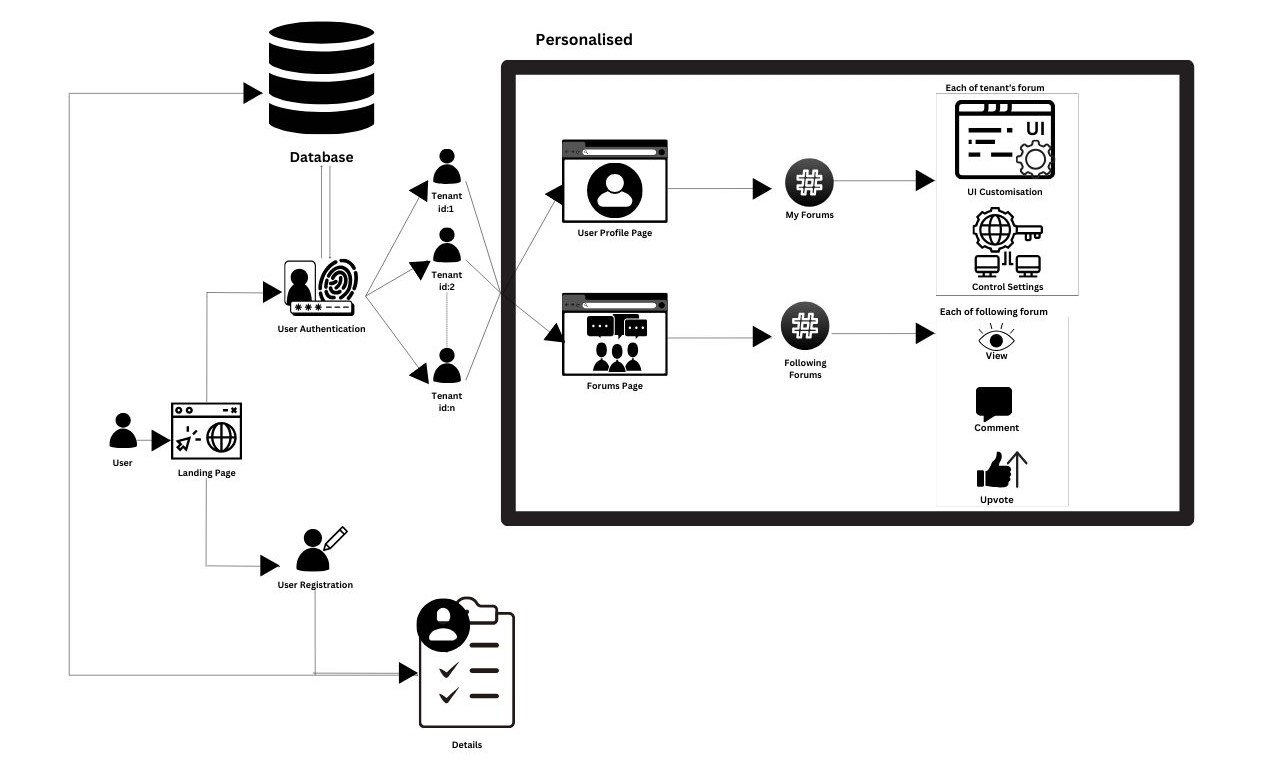
* To provide a platform where users can seamlessly post articles, comment on content, incorporate voting systems, discussion threadsand create dedicated fan forums.
* To enable users to customize the user interface (UI) using predefined templates, fostering a vibrant and unique online space for fan communities to thrive.
* To deliver an intuitive and flexible system that caters to the dynamic needs of diverse fan communities while maintaining data isolation and security.

1. ***Approach***

The specified requirements for the multi-tenant SaaS Forum platform will be implemented using the modules shown in Fig. 1. The approach of each module is explained below.

**1. User Management and Authentication:**

* Step 1: Implement user registration and login functionality using a robust authentication system like JWT or OAuth.
* Step 2: Create user profiles with necessary information (e.g., username, profile picture).
* Step 3: Integrate role-based access control to distinguish between regular users, moderators, and administrators.

****

### **Fig. 1. System Architecture**

The user flow in the FandomHub website begins with users visiting the landing page, where they can choose to log in, sign up, or explore available fan communities. After registration or login, users access their personalized profile, and they can create their own fandom pages with customizable UI to create a unique experience. They can select a specific fan community, participate in discussions, and utilize the voting system to engage with the platform’s diverse content. Users have the flexibility to navigate the site, customize their experiences, and interact with their chosen fan communities throughout their visit.

### **2. Content Management:**

* Step 1: Develop a form or editor for users to create and edit articles.
* Step 2: Implement a categorization system allowing users to assign genres or topics to their articles.
* Step 3: Enable users to view, comment on, and interact with articles posted by others.

### **3. Voting Systems and Discussion Threads:**

* Step 1: Integrate voting buttons (upvote and downvote) for articles and comments.
* Step 2: Implement the logic to record and display vote counts for each article and comment.
* Step 3: Create discussion threads under articles to facilitate user engagement and conversations.

### **4. Forum Creation and Management:**

* Step 1: Develop a feature to allow users to create new fan forums, defining topics or fandoms.
* Step 2: Provide customization options within predefined UI templates for each forum.
* Step 3: Implement forum moderation features for administrators and moderators.

### **5. UI Customization:**

* Step 1: Develop predefined UI templates with customizable elements (themes, colors, fonts).
* Step 2: Allow users to choose a template and customize within the specified parameters.
* Step 3: Implement a preview mechanism so users can visualize the changes they make to the UI.

### **6. Security Measures:**

* Step 1: Implement data validation and sanitization to prevent SQL injection and other attacks.
* Step 2: Use encryption for sensitive data, both at rest and in transit, ensuring secure communication.
* Step 3: Conduct security audits and penetration testing to identify vulnerabilities and address them.

### **7. Performance Optimization:**

* Step 1: Optimize database queries through indexing and query optimization.
* Step 2: Implement caching strategies (e.g., Redis caching) to reduce load times and improve scalability.
* Step 3: Monitor system performance and conduct performance testing to identify bottlenecks and optimize resource usage.

### **8. Testing and Quality Assurance:**

* Step 1: Develop a comprehensive testing suite covering unit tests, integration tests, and end-to-end tests.
* Step 2: Conduct usability testing to ensure the platform is intuitive and user-friendly.
* Step 3: Address bugs, glitches, and usability issues through iterative development and bug-fixing.

**Data Isolation Patterns:**

**Database Schema Per Tenant:**

FandomHub employs a separate database schema for each tenant, guaranteeing complete data isolation. This approach ensures that the data of one tenant is physically separated from others, and access is restricted to only the relevant schema.

**Row-Level Data Isolation:**

In shared databases, a tenant\_id column is added to relevant tables, allowing row-level data isolation. All queries are filtered based on the tenant\_id, guaranteeing that data retrieval remains exclusive to the tenant making the request.

**Shared Database with Views:**

While sharing a database, FandomHub implements database views that act as filters for data based on the current tenant. This approach provides the appearance of distinct databases while utilizing a single physical database, maintaining data isolation.

**Request/Response Isolation Patterns:**

**Tenant Identification Middleware:**

FandomHub employs a middleware layer that extracts the tenant identifier from incoming requests and uses it to route the request to the relevant tenant's data. This ensures that each request interacts solely with data pertinent to the requesting tenant.

**Response Filtering:**

The system employs response filters to prevent inadvertent inclusion of data related to one tenant in responses to other tenants. This layer of security guarantees that tenant-specific data remains protected in APIs and web pages.

**UI Identification and Customization Patterns:**

**Predefined Templates:**

FandomHub offers predefined templates and themes that tenants can choose from to customize the appearance of their site. These templates dictate the structure, styles, and layout, ensuring consistent customization options.

**Customization Boundaries:**

Customization boundaries are clearly defined to prevent tenants from making changes that could affect the overall system's integrity. This approach safeguards core functionality and the experiences of other tenants while promoting individual customizability.

**Site Identification Patterns:**

**Subdomains**:

Each tenant is assigned a unique subdomain, enhancing user-friendliness and aiding in tenant identification. For example, "tenant1.fandomhub.com" and "tenant2.fandomhub.com" represent distinct fan communities.

**Custom Domains:**

Tenants have the option to use their custom domains (e.g., "tenant1.com" or "community2.org") to access their respective sites. FandomHub configures DNS settings to direct these custom domains to the correct tenant within the system.

***3. Service Catalog***

The details of different services are tabulated below.

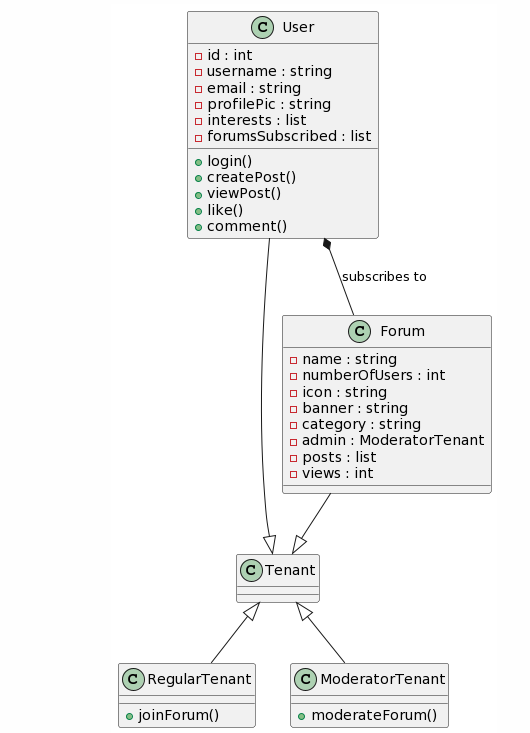
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HTTP Method** | **Service Endpoint** | **Body/Parameter** | **Brief Description** | **Comments** |
| POST | /api/auth/login | { username, password } | User login. | Requires authentication. |
| POST | /api/auth/logout | - | User logout. | Requires authentication. |
| POST | /api/auth/register | { username, email, password } | User registration. |  |
| GET | /api/user/profile | - | Retrieve user profile. | Requires authentication. |
| PUT | /api/user/profile | { firstName, lastName, avatar, gender, ... } | Update user profile. | Requires authentication. |
| POST | /api/user/reset-password | { email } | Request a password reset email. |  |
| GET | /api/content/articles | - | Retrieve a list of articles. | Pagination and filtering options. |
| POST | /api/content/articles | { title, content, category, ... } | Create a new article. | Requires authentication and appropriate permissions. |
| GET | /api/content/articles/{articleId} | - | Retrieve a specific article. |  |
| PUT | /api/content/articles/{articleId} | { title, content, ... } | Update an article. | Requires authentication and appropriate permissions. |
| DELETE | /api/content/articles/{articleId} | - | Delete an article. | Requires authentication and appropriate permissions. |
| POST | /api/content/images | { imageFile } | Upload and manage images. | Requires authentication and appropriate permissions. |
| GET | /api/content/themes | - | Retrieve available themes/templates. |  |
| POST | /api/content/themes | { themeData } | Create or customize a theme. | Requires authentication and appropriate permissions. |
| GET | /api/content/categories | - | Retrieve a list of content categories. |  |
| POST | /api/content/categories | { name, description, ... } | Create a new content category. | Requires authentication and appropriate permissions. |
| GET | /api/community/forums | - | Retrieve a list of discussion forums. | Pagination and filtering options. |
| POST | /api/community/forums | { name, description, ... } | Create a new discussion forum. | Requires authentication and appropriate permissions. |
| GET | /api/community/comments | - | Retrieve comments on articles and discussions. | Pagination and filtering options. |
| POST | /api/community/comments | { articleId, text, ... } | Add a comment to an article or discussion. | Requires authentication and appropriate permissions. |
| GET | /api/community/notifications | - | Retrieve user notifications. | Pagination and filtering options. |

1. ***Design Considerations / Models***

**4.1 Class Diagram**

The class diagram is shown in Fig 2. The responsibility of each class and its interaction with other classes are detailed below.

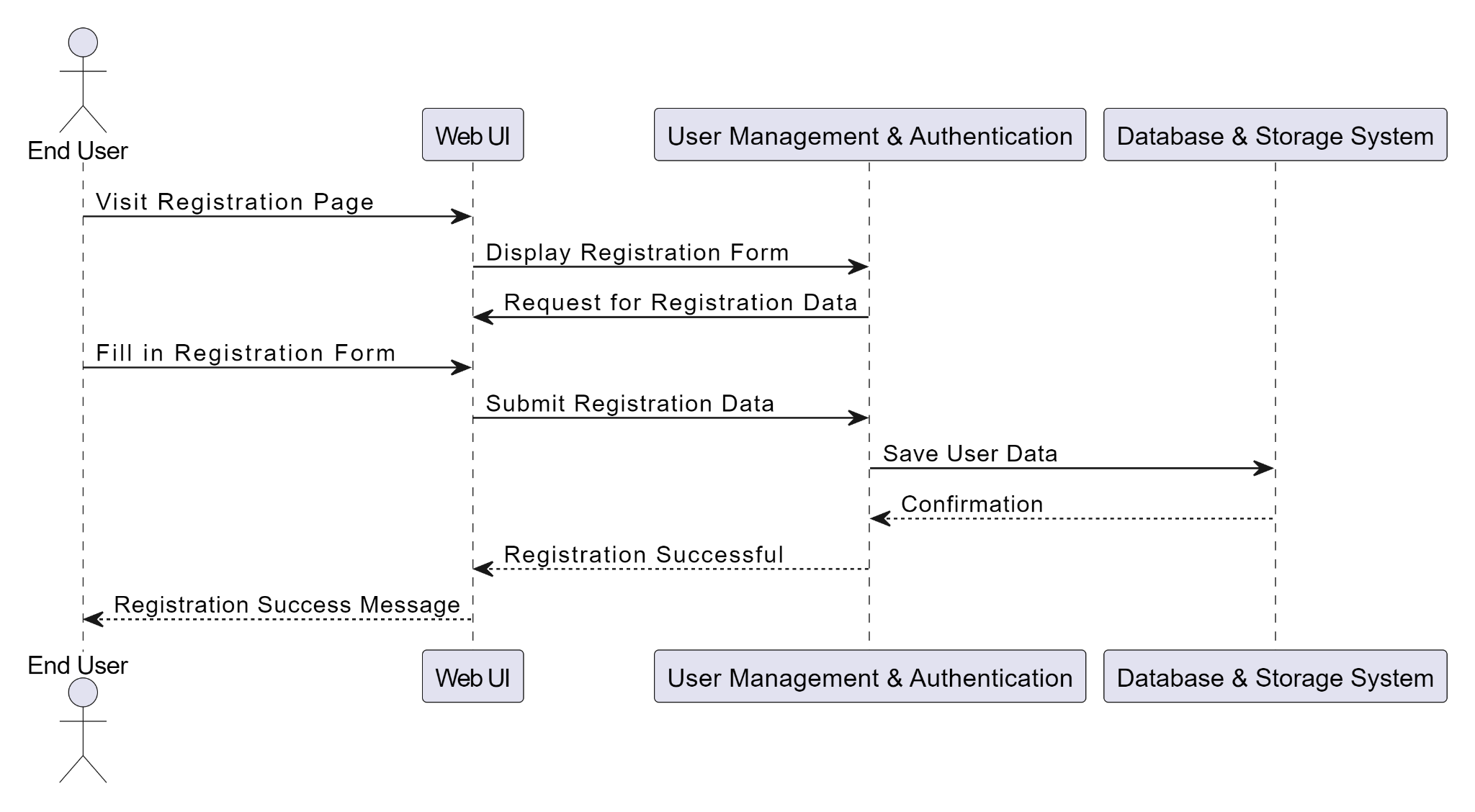
|  |  |  |
| --- | --- | --- |
| **Class Name** | **Responsibility** | **Collaborators** |
| User | Post authentication, a User object will be created, which will contain the id, username, email, profilePic, interests and the forums they are subscribed to. Based on this information, the viewPost() function will allow users to view the various posts. Furthermore, joinFourm(), createPost(), like(), comment() will use the id and username to do their respective operations | Admin is a child class of User. User subscribes to Fourm. |
| Admin | A child of user with additional admin functions such as createForum(), manageUsers(), managePosts(), manageComments() and manageForum(), all of which will take the forum ID as a parameter, allowing the admin to perform various data operations. | Child of User |
| Forum | Stores the id, name, icon, numberOfUsers, banner, category, admin, posts and views as the data. These data will be passed on through an object to User and Admin. |  |



**Fig. 2. Class Diagram**

**4.2 Sequence Diagram**

The interaction between the modules and the tenents are depicted in the sequence diagram shown in Fig. 3.

****

**Fig. 3 Message sequencing between modules**

***4.3 Data Model***

The schema of different tables involved are presented below.

Table Name : User Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Constraints** | **Description** | **Comments** |
| id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the user. |  |
| username | VARCHAR(50) | UNIQUE | User's username. |  |
| email | VARCHAR(100) | UNIQUE | User's email address. |  |
| password\_hash | VARCHAR(255) |  | Hashed password for authentication. |  |
| first\_name | VARCHAR(50) |  | User's first name. |  |
| last\_name | VARCHAR(50) |  | User's last name. |  |
| avatar\_url | VARCHAR(255) |  | URL to the user's avatar image. |  |
| gender | ENUM('Male', 'Female', 'Other') |  | User's gender. |  |
| created\_at | DATETIME |  | Timestamp of user registration. |  |
| updated\_at | DATETIME |  | Timestamp of last profile update. |  |

Table Name : Articles Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Constraints** | **Description** | **Comments** |
| id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the article. |  |
| title | VARCHAR (255) |  | Title of the article. |  |
| content | TEXT |  | Content of the article. |  |
| author\_id | INT | FOREIGN KEY | ID of the user who authored the article. | References 'User' model. |
| category\_id | INT | FOREIGN KEY | ID of the article's category. | References 'Category' model. |
| created\_at | DATETIME |  | Timestamp of article creation. |  |
| updated\_at | DATETIME |  | Timestamp of last article update. |  |

Table Name: Forum Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Constraints** | **Description** | **Comments** |
| id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the forum. |  |
| name | VARCHAR(100) |  | Name or title of the forum. |  |
| description | TEXT |  | Description of the forum's purpose. |  |
| created\_at | DATETIME |  | Timestamp of forum creation. |  |
| updated\_at | DATETIME |  | Timestamp of last forum update. |  |

Table Name: Comment Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | Data Type (Length) | Constraints | Description | Comments |
| id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the comment. |  |
| text | TEXT |  | The content of the comment. |  |
| author\_id | INT | FOREIGN KEY | User ID of the comment author. | References 'User' model. |
| article\_id | INT | FOREIGN KEY | ID of the article being commented on. | References 'Article' model. |
| forum\_id | INT | FOREIGN KEY | ID of the forum being commented on. | References 'Forum' model. |
| created\_at | DATETIME |  | Timestamp of comment creation. |  |

***5. Error Codes & Messages on various validation scenarios***

The error scenarios and intended error messages are tabulated below.

|  |  |  |
| --- | --- | --- |
| **Error Scenario** | **Error Code** | **Error Message** |
| Bad Request | 400 | The request is malformed or contains invalid data. |
| Unauthorized | 401 | Authentication is required to access this feature. |
| Forbidden | 403 | You do not have permission to customize this theme. |
| Not Found | 404 | The requested content or resource could not be found. |
| InternalServer Error | 500 | An unexpected server error has occurred. Please try again later. |
| Service Unavailable | 503 | The service is temporarily unavailable. We are working to resolve this issue. |
| Gateway Timeout | 504 | The server, acting as a gateway, did not receive a timely response. |

***6. UI Wireframes***

The proposed design of the user interface is shown in Fig. 4.

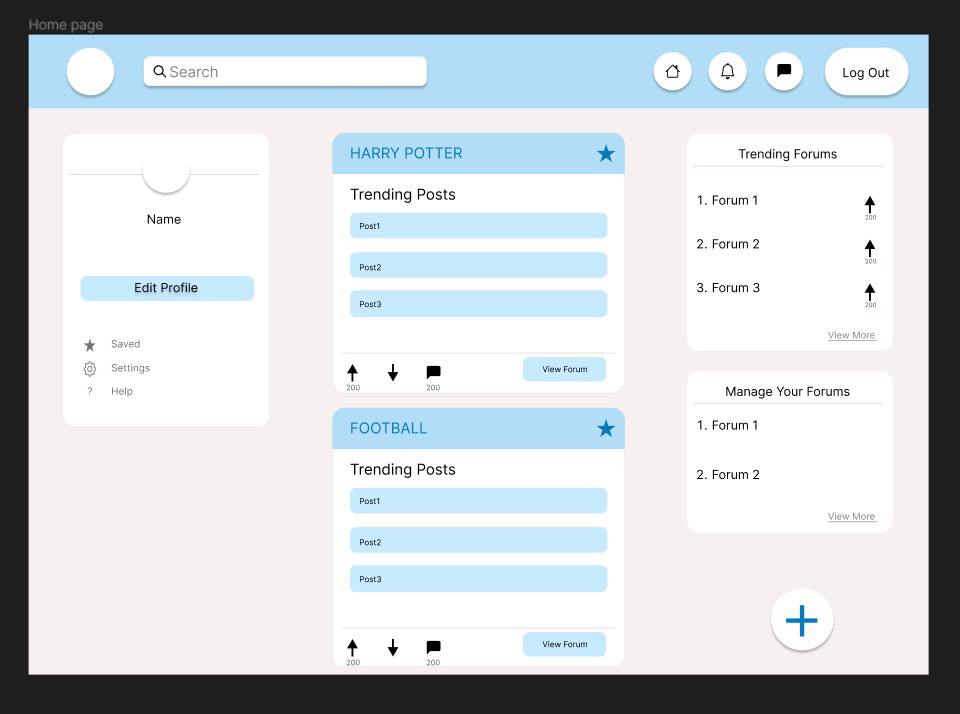
**

Fig. 4. User Interface Design

**\*\*\*\*\***