**Design Document: Understand**

**Version: 1.0**

**Date**: 8/16/2024

**1. Introduction**

**1.1 Purpose**

This design document outlines the architecture, data model, API design, and user interface (UI) considerations for the Understand platform. The document serves as a technical guide for developers and stakeholders to ensure that the system is designed in alignment with the project’s goals and requirements.

**1.2 Scope**

The scope of this document includes the design of the core features for the Understand platform: the resource directory, user ratings, and user authentication. It also covers the future expansion to include blog and forum functionalities.

**2. System Architecture**

**2.1 Overview**

The Understand platform will be built using a three-tier architecture, consisting of a front-end, a back-end, and a database layer. The front-end will be a React-based web application, the back-end will be powered by AWS services (including AWS Lambda and API Gateway), and the database layer will be managed using AWS RDS with PostgreSQL.

**2.2 Component Diagram**

*Insert a component diagram here that shows the interaction between the front-end, back-end, and database.*

**2.3 Front-End Architecture**

* **Framework**: React.js
* **State Management**: Context API or Redux (depending on complexity)
* **Routing**: React Router for managing navigation between different conditions and user pages.
* **Styling**: CSS Modules or Styled-Components for modular and maintainable styles.
* **Build Tool**: Webpack or Create React App (CRA) for bundling and managing dependencies.

**2.4 Back-End Architecture**

* **API Layer**:
  + **API Gateway**: Exposes RESTful API endpoints that the front-end can interact with.
  + **Lambda Functions**: Handle business logic, such as processing user ratings and managing content.
* **Authentication**:
  + **Amazon Cognito**: Manages user sign-up, login, and authentication. This service will also handle user sessions and authorization for accessing protected resources.
* **Content Management**:
  + **Resource Directory**: CRUD operations for managing links to external resources.
  + **Ratings**: Store and retrieve user ratings, calculate average ratings.

**2.5 Database Design**

* **Database Engine**: PostgreSQL hosted on AWS RDS.
* **Schema Overview**:
  + **Users Table**: Stores user data, including user credentials, profiles, and roles.
  + **Conditions Table**: Lists the conditions (e.g., Autism, ADHD) covered by the platform.
  + **Resources Table**: Stores external resource links, categorized by condition. Fields include id, condition\_id, title, description, url, and average\_rating.
  + **Ratings Table**: Captures user ratings for each resource, with fields like user\_id, resource\_id, and rating\_value.

**2.6 Data Flow**

1. **User Requests Resource Links**:
   * User navigates to a condition page.
   * Front-end fetches relevant resources from the back-end via API calls.
   * Back-end retrieves data from the Resources table in PostgreSQL.
   * Data is sent back to the front-end for display.
2. **User Submits a Rating**:
   * User rates a resource.
   * Front-end sends the rating data to the back-end.
   * Lambda function processes the rating, updates the Ratings table, and recalculates the average rating.
   * Updated rating is returned and displayed on the front-end.

**2.7 Security Considerations**

* **User Authentication**: Use Amazon Cognito for secure user authentication and management.
* **Data Encryption**: All sensitive data, such as user passwords and personal information, will be encrypted at rest and in transit.
* **Access Control**: Implement fine-grained access control using IAM roles to ensure that users can only perform actions for which they are authorized.

**3. API Design**

**3.1 API Overview**

The Understand platform will expose a set of RESTful APIs that allow the front-end to interact with the database via the back-end services.

**3.2 Endpoints**

* **Authentication**:
  + POST /auth/signup: Register a new user.
  + POST /auth/login: Log in an existing user.
* **Resources**:
  + GET /resources/:conditionId: Retrieve all resources for a specific condition.
  + POST /resources: Add a new resource (admin only).
  + PUT /resources/:id: Update an existing resource (admin only).
  + DELETE /resources/:id: Remove a resource (admin only).
* **Ratings**:
  + POST /ratings: Submit a rating for a resource.
  + GET /ratings/:resourceId: Retrieve the average rating for a specific resource.

**3.3 Request/Response Formats**

* **Authentication**:
  + **Request**: JSON format with user credentials.
  + **Response**: JSON with user token or error message.
* **Resources**:
  + **Request**: JSON with resource details for POST and PUT requests.
  + **Response**: JSON with resource details or confirmation message.
* **Ratings**:
  + **Request**: JSON with user\_id, resource\_id, and rating\_value.
  + **Response**: JSON with updated average rating.

**4. User Interface Design**

**4.1 User Interface Overview**

The UI will be clean, accessible, and responsive, focusing on ease of use for users who may not be tech-savvy. The design will follow modern UX principles, ensuring a smooth experience across different devices.

**4.2 Wireframes**

*Insert wireframes here if available.*

**4.3 Key Pages**

* **Home Page**: An overview of the platform with links to each condition’s resource page.
* **Condition Page**: A list of resources related to the selected condition, including the ability to rate resources.
* **User Profile Page**: Allows users to manage their accounts and view their ratings.
* **Admin Dashboard**: For managing resources, users, and content (if admin functionality is required).

**4.4 Accessibility**

The platform will adhere to WCAG 2.1 guidelines to ensure it is accessible to users with disabilities. This includes proper use of semantic HTML, keyboard navigability, and sufficient color contrast.

**5. Technology Stack**

**5.1 Front-End**

* **React.js**: For building the user interface.
* **React Router**: For managing navigation.
* **CSS Modules/Styled-Components**: For styling.

**5.2 Back-End**

* **AWS Amplify**: For hosting, user authentication, and deployment.
* **AWS Lambda**: For serverless functions handling business logic.
* **AWS API Gateway**: For managing API endpoints.

**5.3 Database**

* **AWS RDS (PostgreSQL)**: For data storage, providing scalability, reliability, and security.

**5.4 DevOps and Tools**

* **GitHub**: Version control and collaboration.
* **AWS Amplify**: Continuous deployment and hosting.
* **Docker**: (Optional) For containerizing applications during development.

**6. Security Considerations**

**6.1 Authentication and Authorization**

* Use Amazon Cognito for managing user authentication, ensuring that only registered users can submit ratings and access certain features.

**6.2 Data Protection**

* Implement SSL/TLS for data encryption in transit.
* Ensure all sensitive data is encrypted at rest using AWS-managed encryption keys.

**6.3 Regular Audits**

* Regularly review and audit security policies, IAM roles, and data access logs to ensure compliance with security standards.

**7. Future Enhancements**

**7.1 Blogs and Forums**

* Develop and integrate user-generated blogs and forums to enhance community engagement.

**7.2 Mobile Application**

* Explore the possibility of developing native mobile applications for iOS and Android.

**7.3 Advanced Analytics**

* Implement analytics features to track user engagement, resource popularity, and platform performance.

**8. Conclusion**

This design document outlines the technical architecture, data model, and user interface for the Understand platform. The project is designed to be scalable, secure, and user-friendly, with a focus on delivering valuable resources and supporting community engagement. As the project evolves, this document will be updated to reflect any changes in requirements or design.