**Software Requirements Specification (SRS) and Implementation Details for CrowdAcademy**

**1. Introduction**

1.1 Purpose: The purpose of this document is to outline the specifications for the development of CrowdAcademy, a crowd-sourcing learning platform. CrowdAcademy enables users to propose challenges or questions on various topics, fostering a community-driven approach to learning. Other users can respond by creating lessons, which may be offered for free or for a fee, and can be conducted either online or in-person. Users can register for these lessons, enhancing their learning experiences.

1.2 Scope: CrowdAcademy will encompass features such as user registration, challenge submission, lesson creation, lesson enrollment, payment processing, and user communication. It will cater to both online and offline learning modalities, providing a comprehensive platform for collaborative learning.

1.3 Definitions, acronyms, and abbreviations:

* SRS: Software Requirements Specification
* MVP: Minimum Viable Product

**2. Overall Description**

2.1 Product Perspective: CrowdAcademy will be a standalone web application accessible via standard web browsers. It will interact with payment gateways for transaction processing and may integrate with third-party APIs for additional functionality.

2.2 Product Features:

* User Registration: Users can create accounts and build profiles.
* Challenge Submission: Users can propose challenges or questions on diverse topics.
* Lesson Creation: Users can craft lessons in response to challenges/questions.
* Lesson Enrollment: Users can register for lessons.
* Payment Processing: Users can make payments for paid lessons.
* Communication: Users can interact, ask questions, and provide feedback.
* Search and Filter: Users can search for specific challenges, lessons, or topics and filter results.
* Notifications: Users receive updates about lesson activities, registrations, and other relevant events.

2.3 User Classes and Characteristics:

* **Regular Users**: Individuals seeking to learn and engage in challenges and lessons.
* **Lesson Creators**: Users who generate lessons in response to challenges/questions.
* **Administrators**: Personnel managing the system, moderating content, and resolving disputes.

**3. Specific Requirements**

3.1 External Interface Requirements

3.1.1 User Interfaces: CrowdAcademy will feature a user-friendly web interface accessible across devices, ensuring responsiveness and ease of use.

3.1.2 Hardware Interfaces: The system will be accessible through standard web browsers on desktops, laptops, tablets, and smartphones.

3.1.3 Software Interfaces: Integration with payment gateways for secure transaction processing and potential utilization of APIs for extended functionality.

3.1.4 Communication Interfaces: Support for email notifications and in-app messaging to facilitate user communication.

3.2 Functional Requirements

3.2.1 User Registration:

* Users can register by providing necessary information (e.g., name, email, password).
* Authentication mechanisms enable secure user logins.

3.2.2 Challenge Submission:

* Users can propose challenges/questions by specifying title, description, and relevant tags/topics.

3.2.3 Lesson Creation:

* Users can create lessons in response to challenges/questions.
* Lesson creators provide title, description, duration, format (online or in-person), location (if applicable), and pricing (if applicable) for the lesson.

3.2.4 Lesson Enrollment:

* Users can view available lessons and enroll.
* Payment options are available for paid lessons.

3.2.5 Payment Processing:

* Integration with payment gateways for secure transaction processing.
* Users can manage payment methods and view transaction history.

3.2.6 Communication:

* Users can interact via messaging.
* Lesson creators can communicate with enrolled users for updates or additional information.

3.2.7 Search and Filter:

* Users can search for challenges, lessons, or topics using keywords.
* Filtering options based on criteria such as topic, format, and price.

3.2.8 Notifications:

* Users receive notifications about lesson updates, registrations, payments, and other activities.

3.3 Non-Functional Requirements

3.3.1 Performance:

* Responsive system capable of handling multiple concurrent users.
* Minimal response times for various actions.

3.3.2 Security:

* Robust authentication and authorization mechanisms.
* Encryption for payment transactions and compliance with industry standards.

3.3.3 Scalability:

* Design considerations for accommodating a growing user base and increasing data volume.

3.3.4 Usability:

* Intuitive user interface design.
* Accessibility adherence for users with disabilities.

3.3.5 Reliability:

* Minimal downtime with error handling mechanisms.
* Data integrity and backup procedures to prevent loss.

**4. Implementation Details**

4.1 Frontend Development with React:

**Technology Stack:** React.js, HTML, CSS, JavaScript

**Description:** The frontend development of CrowdAcademy will focus on creating an intuitive and responsive user interface to facilitate seamless interaction for users.

* **User Interface Design:** Designing visually appealing and user-friendly interfaces using HTML and CSS to ensure an optimal user experience across devices.
* **Component Development:** Implementing reusable components using React.js to modularize the frontend codebase and enhance maintainability.
* **State Management:** Leveraging React's state management capabilities to manage user interactions, form submissions, and application state.
* **Integration with Backend:** Consuming backend APIs via asynchronous requests (e.g., Axios) to fetch data dynamically and update the UI accordingly.
* **Responsive Design:** Implementing responsive design principles using CSS media queries to ensure compatibility and usability across various screen sizes and devices.

4.2 Backend Development with Flask:

**Technology Stack:** Flask (Python)

**Description:** Flask will serve as the backend framework for CrowdAcademy, handling server-side logic, API endpoints, and database interactions.

* **Setup Flask App:** Initialize a Flask application, configure routes, and set up middleware for request processing.
* **User Authentication:** Implement user authentication using Flask-Login or JWT to manage user sessions and secure access to protected routes.
* **API Endpoints:** Define RESTful API endpoints using Flask-Restful or Flask-RESTPlus to facilitate communication between the frontend and backend.
* **Data Validation:** Implement data validation and sanitization to ensure data integrity and prevent security vulnerabilities.

4.3 Database Management with MongoDB:

**Technology Stack:** MongoDB

**Description:** MongoDB will serve as the NoSQL database for storing user data, challenges, lessons, and other application-related information.

* **Database Connection:** Connect Flask application to MongoDB using the PyMongo library to establish a connection and perform database operations.
* **Schema Design:** Design MongoDB collections to store data in a flexible, document-based format suitable for the application's requirements.
* **Data Management:** Implement CRUD (Create, Read, Update, Delete) operations to manage data within MongoDB collections.

4.4 Cloud-Hosted MongoDB Database with MongoDB Atlas:

**Technology Stack:** MongoDB Atlas

**Description:** MongoDB Atlas will provide cloud-hosted MongoDB database management, offering scalability, reliability, and automated backups.

* **Create Atlas Cluster:** Set up a MongoDB Atlas cluster on the cloud platform of choice, specifying configuration options such as storage size, region, and replication.
* **Connect Flask to Atlas:** Configure Flask application to connect to the MongoDB Atlas cluster using connection strings provided by Atlas.
* **Security Configuration:** Implement security measures such as IP whitelisting, authentication mechanisms, and encryption at rest to secure data stored in MongoDB Atlas.

4.5 Media Storage with Google Drive:

**Technology Stack:** Google Drive API

**Description:** Google Drive will serve as the media storage solution for CrowdAcademy, allowing users to upload and access media files such as images, videos, and documents.

* **Google Drive API Integration:** Integrate Google Drive API into the Flask application to enable file upload, download, and management functionalities.
* **File Upload:** Implement functionality for users to upload media files to their Google Drive accounts directly from the CrowdAcademy application.
* **File Management:** Develop features to manage uploaded files, such as organizing files into folders, renaming files, and deleting files as needed.
* **Access Control:** Configure access permissions to ensure that only authorized users can access and modify their own files stored on Google Drive.

By following this detailed implementation plan, CrowdAcademy can leverage Flask for backend development, MongoDB and MongoDB Atlas for database management, and Google Drive for media storage, providing users with a robust and scalable learning platform.