1. **Project Overview**
    - Purpose
    - Role Assignment
    - Development Model
2. **Requirement Analysis**
    - Functional Requirements
        - User Account Management:
          Registration, login, and logout capabilities. Profile creation and editing, including qualifications for tutors, subjects of expertise, etc.
        - Tutor Information Posting:
          Tutors can post information about their teaching expertise, available timings, fee structures, etc.
          Tutors can update or delete their posted information.
        - Student Requirement Posting:
          Students can post their tutoring needs, including required subjects, level of education, learning goals, etc.
          Students can update or delete their requirements.
        - Matching System:
          The system should be able to match students with tutors based on requirements and qualifications, either automatically or manually.
          Search and filter capabilities to help users find suitable tutors or students.
        - Scheduling and Management of Lessons:
          Tutors and students can negotiate and schedule lesson times.
          Calendar and reminder functionalities for scheduled lessons.
        - Payment System:
          Secure processing of payments including course fee transactions and tutor earnings withdrawals.
        - Feedback and Review System:
          Students can rate and review tutors after lessons.
          Tutors can provide feedback on students' progress and performance.
        - Customer Support and Dispute Resolution:
          Support functionalities for user queries and assistance.
          Mechanisms to resolve disputes between tutors and students.
    - Non-Functional Requirements
        - Performance Requirements:
          System response time and processing efficiency.
          Support for many users and concurrent operations.
        - Usability:
          User-friendly interface and easy navigation.
          Availability of help documentation and user guides.
        - Reliability and Stability:
          Normal operational uptime of the system.
          Quick recovery capabilities.
        - Security:

Data encryption and privacy protection for users.
Prevention of unauthorized access and data breaches.

- Scalability:
  The system's ability to accommodate an increasing number of users and data volume.
  Architectural support for future expansion of features.
- Compatibility:
  Support for various browsers and devices.
  Consideration for integration with other systems, like social media logins.
- Maintainability and Code Quality:
  The system is easy to maintain and update.
  Code follows industry standards and is well documented.
- Legal and Regulatory Compliance:
  Compliance with relevant educational and privacy laws and regulations.
  Adherence to electronic payment and data protection standards.

3. **User Stories and Use Cases**
   - User Stories
   - Use Cases
4. **Technology Stack**
   - Frontend: Next.js
   - Backend: Flask
   - Database
5. **System Architecture and Design**
   - System Architecture Diagram
   - Class Diagrams

Student

student_id		Primary key
username		
password		
email		
grade		
timezone		
msg		

Tutor

tutorId		Primary key
username		
password		

email		
edu_background		
timezone		
available_time		
msg		

Subject

subject_id		Primary key
subject_name		

Tutor_subject

Tutor_subject_id		Primary key
tutor_id		
subject_id		

Student_post

post_id		Primary key
student_id		
student_name		
subject_id		
subject_name		
date		
msg		
salary		
available_time		
status		

Review

review_id		
student_id		
teacher_id		
rating		

msg		
date		

request

request_id		
sender_id		
receiver_id		
status		

- Frontend Architecture
- Backend Architecture
- Data Model
- Security Considerations

6. **API Documentation**
    - **Overview**: A brief description of the API's purpose and its main capabilities.
    - **Authentication and Authorization**: How clients authenticate with the API (e.g., token-based authentication).
    - **Endpoints**: Detailed description of each API endpoint, including:
        - **Path**: The URL path of the endpoint.
        - **Method**: HTTP method (GET, POST, PUT, DELETE, etc.).
        - **Parameters**: Required and optional parameters, their types, and descriptions.
        - **Request Examples**: Sample requests with example inputs.
        - **Response Format**: Description of the response structure, including status codes and payload format.
        - **Response Examples**: Sample responses for successful and error scenarios.
    - **Error Handling**: Explanation of error responses.
    - **Rate Limits**: Any limitations on the number of requests that can be made.
    - **Versioning**: How the API is versioned and updated.
    - **Best Practices**: Guidelines for developers using the API effectively.
7. **Development Guidelines**
    - Environment Setup
    - Code Structure
    - Coding Standards
    - Version Control
    - Testing
8. **Deployment and Maintenance**
    - Deployment Process
    - CI/CD Pipeline
    - Monitoring and Logging

- Maintenance Plan

9. Appendix