**Software Development Document (SDD)**

**Project Title**: Tutor Application System

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define and describe the software design for the Tutor Application System. It serves as a blueprint for the development team to implement the system in a consistent, maintainable, and scalable manner.

**1.2 Scope**

This web-based platform allows students to browse available tutors and subjects, book sessions, and manage their learning schedules. Tutors can set their availability, manage bookings, and interact with students. The system also provides an admin interface for user and subject management.

**2. System Overview**

* **User Roles**: Student, Tutor, Admin
* **Core Functionalities**:
  + User Authentication & Authorization
  + Subject Browsing and Filtering
  + Session Scheduling and Management
  + Availability Management for Tutors
  + Admin Control Panel

**3. Architecture Design**

**3.1 Overview**

The application uses a **three-tier architecture** for maintainability, scalability, and separation of concerns:

* **Presentation Layer** (Frontend)
* **Business Logic Layer** (Backend APIs)
* **Data Access Layer** (Database)

**3.2 Component Breakdown**

**1. Frontend (Client-Side)**

* **Technology**: React.js / HTML / CSS / JavaScript
* **Role**: Provides the user interface for students, tutors, and admins.
* **Responsibilities**:
  + Display dynamic content
  + Send HTTP requests to backend
  + Handle client-side validations and routing

**2. Backend (Server-Side)**

* **Technology**: Java with Spring Boot
* **Role**: Acts as the business logic handler, API provider, and security enforcer.
* **Responsibilities**:
  + Handle RESTful API requests
  + Enforce role-based access
  + Manage session booking logic
  + Handle database operations via JPA/Hibernate

**3. Database**

* **Technology**: MySQL
* **Role**: Persistent data storage
* **Responsibilities**:
  + Store user details, sessions, subjects, timetables
  + Ensure data consistency and referential integrity

**3.3 Architectural Diagram (Text Representation)**

[Client Browser]

|

[React Frontend (UI)]

|

[Spring Boot REST APIs]

|

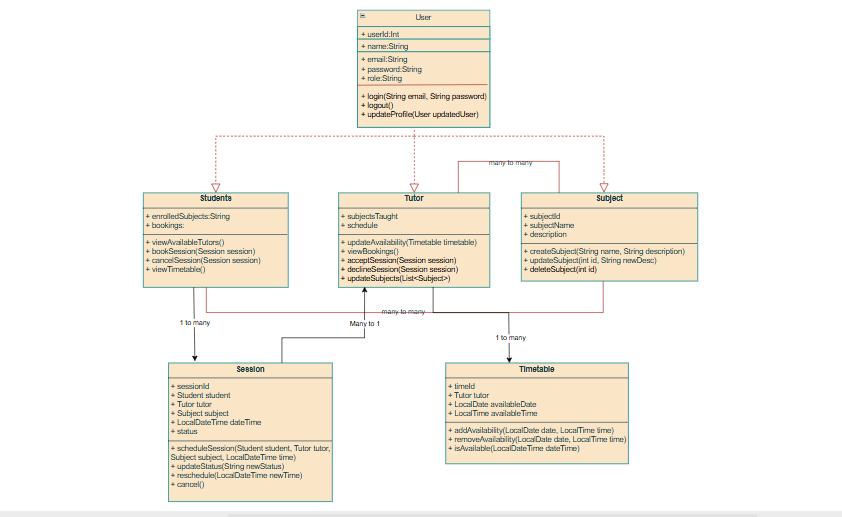
[Service Layer → Repository Layer]

|

[MySQL Database]

**4. Class Diagram**

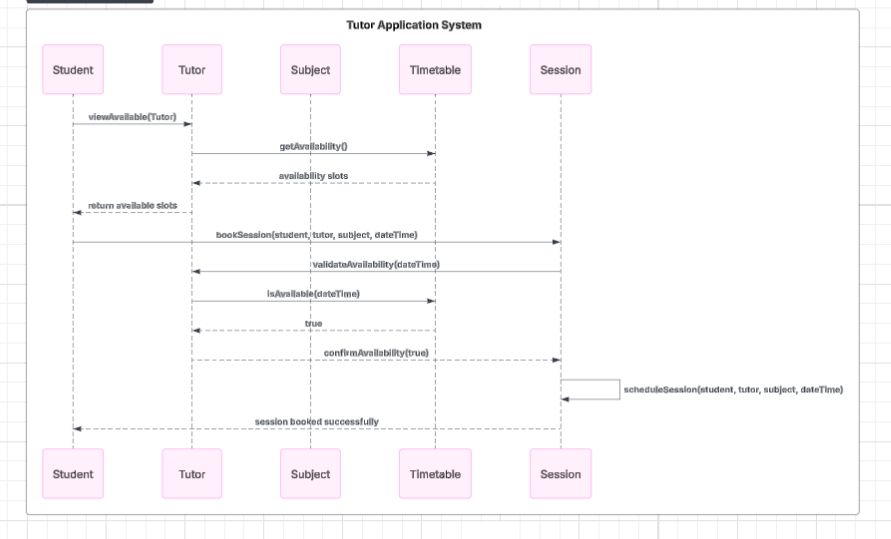
* **User**
  + Fields: userId, name, email, password, role
* **Student extends User**
  + enrolledSubjects: List<Subject>
  + sessions: List<Session>
* **Tutor extends User**
  + subjectsTaught: List<Subject>
  + timetable: List<Timetable>
* **Subject**
  + subjectId, name, description
* **Session**
  + sessionId, student, tutor, subject, dateTime, status
* **Timetable**
  + timeId, tutor, availableDate, availableTime



**5. Sequence Diagram – Booking a Session**

**Actors**: Student, System, Tutor  
**Steps**:

1. Student logs in
2. Selects a subject and tutor
3. Views available slots from tutor’s timetable
4. Chooses a time and confirms session
5. Backend checks availability → books session
6. Tutor is notified of new booking



**6. Database Design – ER Attributes (Key Tables)**

* **User** (user\_id, name, email, password, role)
* **Subject** (subject\_id, name, description)
* **Session** (session\_id, student\_id, tutor\_id, subject\_id, date\_time, status)
* **Timetable** (time\_id, tutor\_id, available\_date, available\_time)

**7. Performance**

**7.1 Goals**

* Ensure **fast API response times** under normal and peak loads
* Achieve **concurrent user support** with minimal resource conflict
* Maintain **smooth UI interactions** through efficient frontend-backend communication

**7.2 Design Considerations**

* **Caching**: Frequently accessed data such as subject lists and tutor profiles can be cached at the server or client-side to reduce DB queries.
* **Lazy Loading**: Load tutor profiles or subject data only as needed, improving initial load speed.
* **Asynchronous Communication**: REST API calls and confirmation emails will be handled asynchronously (e.g., using @Async in Spring) to avoid blocking operations.
* **Pagination & Filtering**: For listing tutors and sessions, implement pagination and server-side filtering to improve performance on large datasets.
* **Connection Pooling**: Utilize HikariCP for managing database connections efficiently.
* **Load Handling**: Use Spring Boot’s actuator and monitoring tools (like Prometheus/Grafana) to track performance and scale services if hosted on cloud platforms.

**7.3 Benchmarks (Expected)**

|  |  |
| --- | --- |
| **Metric** | **Target Value** |
| API Response Time | ≤ 2 seconds (95% cases) |
| Concurrent Users Support | Up to 1000 users |
| DB Query Response Time | ≤ 500 milliseconds |
| System Uptime | 99.9% |

**Non-Functional Requirements**

* 🔐 **Security**: Password encryption using BCrypt
* 📱 **Responsiveness**: UI adjusts to screen size
* 🧩 **Scalability**: Can handle increasing user load
* 🧩 **Performance**: API response ≤ 2s
* 🧩 **Role Management**: Access control enforced in backend

**Assumptions and Constraints**

* All users must have internet connectivity.
* Users are expected to register and maintain valid profiles.
* Sessions once booked cannot overlap for the same tutor.
* Tutors manage their own availability.

**Conclusion**

This Software Design Document provides a structured and performance-oriented blueprint for the development of the Tutor Application System. By adhering to layered architecture, RESTful principles, and responsive design, the system aims to offer a seamless experience to students and tutors while ensuring maintainability and scalability for future growth.