1. Software Architecture (40%)

**System Overview:**

The Stuber system is structured around a client-server model. The backend is built using Node.js with MySQL as the database, while the frontend utilizes React Native for a mobile application. The app connects student drivers and riders by allowing users to post or book rides.

**Major Components:**

1. Frontend (React Native):

- Responsible for the UI and interaction with users.

- Key functionalities: login/signup screens, ride posting, ride booking, profile management, and notifications.

2. Backend (Node.js):

- Handles business logic, API requests, and database interactions.

- Key functionalities: managing authentication, storing rides and profiles, and handling notifications.

3. Database (MySQL):

- Stores user data, ride details, and driver/rider profiles.

- Major tables:

- `Users` (ID, name, email, studentID, role)

- `Rides` (rideID, driverID, pickup, dropoff, time, seats)

- `Bookings` (bookingID, rideID, riderID, status)

**Interfaces between Components:**

- \*\*API (Backend ↔ Frontend):\*\* RESTful API for data communication.

- \*\*Database ↔ Backend:\*\* SQL queries to retrieve, update, and store data.

**Assumptions:**

- The app assumes stable network connections for real-time updates and bookings.

- The user base will initially be small, which reduces scalability concerns.

**Architectural Decisions & Alternatives:**

1. Database Choice:

- Chosen: MySQL

- Alternative: MongoDB (NoSQL)

- Pros: Easier to handle unstructured data, faster development.

- Cons: Less reliable for transactions and structured data.

2. Backend Framework:

- Chosen: Node.js

- Alternative: Django (Python)

- Pros: Django comes with built-in admin features and a more robust ORM.

- Cons: Steeper learning curve for the team; more setup required for API-based applications.

---

2. Software Design (30%)

**Detailed Component Breakdown:**

1. Frontend (React Native):

- Components:

- `AuthenticationScreen` (handles user login/signup).

- `RidePostingScreen` (allows drivers to post rides).

- `RideBookingScreen` (allows riders to search and book rides).

- `ProfileScreen` (displays user profiles).

- `NotificationSystem` (handles alerts and notifications).

- Responsibilities:

- Provide intuitive UI, interact with API, and ensure smooth navigation between screens.

2. Backend (Node.js):

- Packages/Modules:

- `AuthController` (handles user authentication).

- `RideController` (handles ride creation and management).

- `BookingController` (handles ride bookings).

- `NotificationController` (handles notifications).

- Responsibilities:

- Handle API requests, interact with the database, ensure data integrity, and implement business logic.

3. Database (MySQL):

- Schema:

- `Users`: Handles user data.

- `Rides`: Stores ride details.

- `Bookings`: Manages ride reservations.

- \*\*Responsibilities:\*\*

- Store and retrieve data efficiently, maintain relationships between tables.

---

3. Coding Guidelines (10%)

For each language used in the project:

- JavaScript (Node.js): Follow the BlaBlaCar JavaScript Style Guide. This guide is widely adopted, covers best practices, and enforces consistency.

- React Native (JavaScript): Same as above, as React Native uses JavaScript.

- SQL (MySQL): Follow SQL Style Guide. It ensures that queries are readable and maintainable.

Enforcement:

- Regular code reviews and automated linting tools (e.g., ESLint for JavaScript) will be used to enforce the coding standards.

---

4. Process Description (20%)

\*\*i. Risk Assessment:\*\*

1. \*\*Integration Challenges:\*\*

- Likelihood: High, Impact: Medium

- Evidence: Previous coordination delays between teams.

- Steps: Regular meetings, well-documented API.

- Detection: Cross-team code integration tests.

- Mitigation: Clear API documentation.

2. \*\*Performance Issues:\*\*

- Likelihood: Medium, Impact: High

- Evidence: Growing user base.

- Steps: Early load testing.

- Detection: Monitoring performance metrics.

- Mitigation: Database query optimization.

3. \*\*Security Vulnerabilities:\*\*

- Likelihood: Medium, Impact: High

- Evidence: Sensitive user data.

- Steps: Secure password storage (hashing using bcrypt).

- Detection: Penetration testing.

- Mitigation: Implementing strong authentication and encryption.

4. \*\*Backend Overload:\*\*

- Likelihood: Low, Impact: High

- Evidence: Backend bottlenecks.

- Steps: Use caching strategies, server load balancing.

- Detection: Monitoring server load.

- Mitigation: Optimize API calls.

5. \*\*UI/UX Delays:\*\*

- Likelihood: Medium, Impact: Medium

- Evidence: Complex UI interactions.

- Steps: Early prototype testing.

- Detection: Usability testing feedback.

- Mitigation: Prioritize essential features.

\*\*ii. Project Schedule:\*\*

| **Milestone** | **Task** | **Estimate** | **Dependencies** |
| --- | --- | --- | --- |
| API Design | Complete authentication API | 2 weeks | None |
| Database Schema | Design MySQL schema | 1 week | API Design |
| Frontend Authentication | Implement login/signup screens | 1 week | Database Schema |
| Ride Posting System | Develop ride posting functionality | 2 weeks | API Design, Database Schema |
| Integration Testing | Full system integration test | 3 weeks | All features completed |

\*\*iii. Team Structure:\*\*

- \*\*Alice:\*\* Backend Engineer (Node.js, API)

- \*\*Haytham:\*\* Backend Engineer (Node.js, MySQL)

- \*\*Yuzhou:\*\* Frontend Engineer (React Native)

- \*\*Wells:\*\* Frontend Engineer (React Native)

\*\*iv. Test Plan & Bugs:\*\*

- \*\*Unit Testing:\*\* Each component (e.g., `AuthController`, `RideController`) will be tested individually.

- \*\*Integration Testing:\*\* API integration with frontend, ensuring correct data flow.

- \*\*Usability Testing:\*\* Conducted with beta users to ensure ease of navigation and feature accessibility.

\*\*v. Documentation Plan:\*\*

- \*\*User Guide:\*\* Step-by-step instructions for using the app, including ride posting and booking.

- \*\*Developer Guide:\*\* Documentation of the API, code structure, and setup.

- \*\*Help Menu:\*\* In-app help section for quick troubleshooting.