Plus Points in Implementation (Overall Evaluation Criteria)

1. Authentication:
- Implement robust user authentication protocols to ensure secure access.

2. Cost Estimation - Time and Space:
- Conduct a thorough analysis of time and space complexity in the system.
- Utilize efficient algorithms and data structures to optimize both time and space requirements.

3. Handling System Failure Cases:
- Implement fault-tolerant mechanisms to address system failures.
- Employ backup and recovery strategies for data integrity.
- Develop comprehensive error recovery procedures to minimize downtime.

4. Object-Oriented Programming Language (OOPS):
- Choose a robust OOPS language for structured and modular code.
- Leverage OOPS principles such as encapsulation, inheritance, and polymorphism for maintainability and extensibility.

5. Trade-offs in the System:
- Clearly define and document trade-offs made during system design.
- Evaluate and communicate the rationale behind architectural and design decisions.
- Consider trade-offs in terms of performance, scalability, and maintainability.

6. System Monitoring:
- Implement comprehensive monitoring tools to track system performance.
- Utilize real-time dashboards and logging mechanisms to promptly identify and address issues.

7. Caching:
- Integrate caching mechanisms to enhance system response times.
- Utilize caching for frequently accessed data to reduce database load.
- Implement cache eviction policies for optimal resource utilization.

8. Error and Exception Handling:
- Develop a robust error and exception handling framework.
- Provide meaningful error messages for effective debugging.
- Regularly review and update error-handling strategies based on system usage patterns.

Instructions for Project submissions:

Document Format:
- Combine textual explanations, screenshots, and code snippets for clarity.
- Organize information in a structured manner, following a logical flow.

Demonstration:
- Include a demonstration video showcasing key features of the ride-sharing platform.
- Alternatively, use screenshots to visually highlight the user interface and functionality.

Case Study : Unified Billing & Reporting Platform for Multi-Client, Multi-Vendor Operations

MoveInSync enables employee commute solutions for multiple corporate clients through a network of transportation vendors.
Each **client** has **multiple vendors**, and each vendor may operate under a **different billing model** — for example:

- **Package Model**: A fixed monthly cost for a certain number of trips or kilometers.
- **Trip Model**: Billing based on the number of trips made or distance covered.
- **Hybrid Models**: Some combination of the above.

Employees take daily trips to and from the office using these vendors. These trips may exceed standard limits (extra kilometers or hours), which require **incentive adjustments** — either as additional payouts to vendors or incentives to employees.

MoveInSync's billing team must generate accurate, auditable billing reports for:

- **Clients** – Monthly summary for all trips and vendor payments.
- **Vendors** – Detailed statements of payable trips and incentives.
- **Employees** – Summary of earned incentives for extra hours/trips.

Currently, the billing and reporting process is **manual, fragmented, and error-prone** due to varied models and data sources.

Your Task

You are tasked with designing a **Billing and Reporting System** for MoveInSync that can:

1. Handle **multiple clients** and **multiple vendors** with **different billing models**.
2. Accurately **compute trip-level and incentive-based billing**.
3. Generate **client-, vendor-, and employee-level reports** that are clear, reliable, and auditable.
4. Present a **usable UI dashboard** where internal users (MoveInSync operations and finance teams) can:
    - View trip data summaries.
    - Configure billing models per client/vendor.
    - Generate and export reports for each stakeholder type.
5. **Design an intuitive and insightful UI** featuring relevant statistics, visual charts, and key performance indicators to make billing and reporting easy to understand.
6. **Implement multi-tenant architecture** with role-based access control — supporting distinct views and permissions for **Admin**, **Vendor**, and **Employee** users.
7. **Ensure secure and authenticated APIs** that strictly isolate tenant data, preventing any unauthorized access or data leakage across different clients or vendors.