OOSD PART 2 – INDIVIDUAL CONTRIBUTION AND REFLECTIVE REPORT

Evidence of Work and Outline

My primary responsibility was implementing the backend logic and database integration for the student accommodation application. This involved:

- Translating UML diagrams into usable code structures, creating a clear correlation between the planned architecture and the implemented system.
- Establishing database connectivity and ensuring the persistent storage of accommodation data.
- Applying design patterns to enhance code maintainability and scalability.

Reflection on Contribution

Skills and Knowledge Applied:

Object-Oriented Programming (OOP): The translation of UML diagrams into code required a solid understanding of OOP principles. By applying encapsulation, inheritance, and polymorphism, I created a robust system architecture that mirrored our initial designs closely.

Database Management: Implementing JDBC for database connectivity called upon my knowledge of SQL and data management. I was responsible for designing the database schema, which involved normalizing the database to reduce redundancy and ensure data integrity.

Problem-Solving: Throughout the project, I encountered several unforeseen challenges, particularly in integrating different system components. Leveraging my problem-solving skills, I managed to find effective solutions that kept the project on

track. Persistence of data was a critical requirement for our application, ensuring that accommodation information and system changes were reliably stored and retrieved. We achieved this through:

Relational Database: We used a relational database management system (RDBMS) for storing data. This choice was based on the structured nature of our data and the need for complex queries.

JDBC API: Java Database Connectivity (JDBC) API was used to interact with the database. This allowed us to execute SQL statements from within our Java application, facilitating data persistence.

Transaction Management: To ensure data integrity, especially in operations involving multiple steps (like booking accommodation), we implemented transaction management. This ensured that all parts of a transaction were completed successfully before committing the changes to the database.

Design Patterns

The use of design patterns in our system was instrumental in addressing common software design issues:

Singleton Pattern: We applied the Singleton pattern for database connection management. This ensured that only one database connection instance was created and shared across the application, optimizing resource usage.

Factory Method Pattern: For creating different types of user interfaces based on the user role (student, admin), we used the Factory Method pattern. This provided flexibility in managing various user interactions while maintaining loose coupling between the interface creation and its use.

Observer Pattern: To handle notifications (e.g., accommodation availability changes), we implemented the Observer pattern. This allowed various parts of our application to observe and react to events without being directly linked, enhancing modularity and reusability.

Conclusion

Reflecting on my contribution to the student accommodation application project, I recognize the importance of a solid foundation in OOP, database management, and the application of design patterns. These skills were not only crucial in executing my tasks but also in contributing to a cohesive and efficient team effort. The technical challenge of ensuring data persistence taught me valuable lessons in database design and transaction management. Furthermore, the practical application of design patterns reinforced their value in creating scalable and maintainable software systems. This project has significantly contributed to my growth as a software developer, highlighting the importance of theoretical knowledge and its practical application in solving real-world problems.