**Group Report: Agile Development for Cedar Woods Accommodation System**

**1. Team Roles, Coordination, and Communication**

Our team consisted of three members, and we collaborated to design and implement the Cedar Woods Accommodation System as part of our Object-Oriented Systems Development module. This project required us to apply Agile development methodologies while utilizing UML modelling and Java-based object-oriented programming to build a functional software system.

To align with Agile practices, we structured our team with the following roles:

* **Scrum Master**: Facilitated team discussions, ensured tasks were completed within sprints, and managed coordination.
* **Product Owner**: Defined system requirements, gathered feedback, and ensured that the features aligned with project needs.
* **Developers & Testers**: Implemented Java code based on UML diagrams, developed the Graphical User Interface (GUI), and conducted rigorous testing.

We coordinated through group chats and used Trello to manage tasks and track progress. Meetings were held periodically to review development progress and address challenges. This ensured a smooth workflow and adherence to Agile principles.

A screenshot of a computer

AI-generated content may be incorrect.

**2. Iteration Plan (Sprint Plan) – Agile Approach**

To ensure efficient development, we followed asprint-based approach, spanning from September until completion. Our sprint structure aligned with Agile methodologies and the software development lifecycle:

**Phase 1 (September - October): System Analysis and UML Design**

* Conducted requirement analysis based on the Cedar Woods Accommodation System specifications. (Beril)
* Created Use Case Diagram to capture system functionalities. (Beril & Nathan)
* Developed a Class Diagram to represent object-oriented relationships between entities. (Beril & Nathan)
* Designed two Sequence Diagrams to illustrate interactions for key use cases (e.g., Guest Check-in and Check-out). (Beril & Nathan)
* Started drafting the group report structure. (Beril)

**Phase 2 (November - December): Core Feature Development**

* Implemented Java classes based on UML diagrams (Guest, Accommodation, Booking, Cleaning, etc.). (Nathan & Beril)
* Developed Check-in & Check-out functionalities with validation checks. (Nathan)
* Created initial JavaFX GUI wireframe with interactive components (buttons, text fields, tables). (Nathan)
* Expanded the group report by documenting design choices, UML analysis and team workflow. (Beril)

**Phase 3 (January - February): Feature Expansion & Testing**

* Improved the GUI design and implemented area-based accommodation status updates. (Nathan)
* Integrated the cleaning status module, allowing cleaning staff to update room conditions. (Nathan)
* Added a statistics module to calculate daily breakfast counts and accommodations requiring cleaning. (Nathan)
* Conducted unit testing to validate system functionality. (Defyn)
* Revised the group report with detailed implementation descriptions. (Beril)

**Phase 4 (March - Final Submission): Finalization & System Testing**

* Conducted final integration of all components. (Beril & Nathan)
* Performed rigorous unit testing and system testing to verify accuracy. (Beril & Nathan)
* Debugged and refined the software for usability and efficiency. (Defyn)
* Completed the final version of the group report. (Beril)
* Prepared for the final demonstration session. (Beril & Nathan)

**3. UML Diagrams & Design Decisions**

As part of the Object-Oriented Analysis and Design (OOAD) approach, we created several UML diagrams to model the system structure and interactions:

* Use Case Diagram: Captured functionalities such as guest check-in, check-out, room status updates, and booking.
* Class Diagram: Defined relationships between entities such as Guest, Accommodation, Booking, and CleaningStaff.
* Sequence Diagrams: Illustrated guest check-in and cleaning status updates to visualize system interactions.

**Design Principles and Patterns Applied**

To ensure maintainability and scalability, we followed standard object-oriented design principles:

* **Single Responsibility Principle (SRP)** – Each class had a specific function (e.g., Guest for guest details, Accommodation for booking management).
* **Open/Closed Principle (OCP)** – Our system allowed for new accommodation types (Cabins, Yurts, etc.) without modifying existing code.
* **Liskov Substitution Principle (LSP)** – The Accommodation class was extendable to multiple types (Cabins, Yurts, etc.), ensuring compatibility.
* **Model-View-Controller (MVC) Pattern** – We structured the system into Model (business logic), View (JavaFX GUI), and Controller (handles interactions).
* **Singleton Pattern** – We applied the **Singleton** pattern to ensure that certain system-wide components, such as the Manager, maintain a **single instance** throughout the application's runtime. This prevents redundant object creation and ensures that the same instance is shared across different parts of the application, improving efficiency and consistency. By applying these principles, we ensured our design was maintainable.

**4. Implementation & Testing Strategy**

Our implementation was carried out using Java and JavaFX, with a focus on usability and maintainability. Key components included:

* **Graphical User Interface (GUI)**:
  + Used a JavaFX template to structure the layout.
  + Added table columns to populate with accommodation details.
  + Implemented occupancy and availability tracking using using hasguestbooking.
  + Designed a cleaning status choice box under the Room Maintenance header.
  + Integrated a breakfast checkbox at check-in, updating the table with "Yes" when selected.
* **Business Logic Enhancements**:
  + Created an enum for cleaning status and integrated it into the system.
  + Implemented automatic updates to cleaning status (e.g., check-out sets status to "Dirty").
  + Fixed pricing calculations by completing the TODO for price logic.
  + Ensured that the guest count in the table updates correctly upon booking.
  + Enabled the check-out button to remove guests and reset the room status.
* **Testing Approach**:
  + Created test cases to validate core functionalities, such as check-in, check-out, and cleaning updates.
  + Tested boundary conditions, ensuring guests couldn't book unavailable rooms.
  + Verified GUI responsiveness and data accuracy.

By systematically implementing and testing the system, we ensured functionality and reliability.

**5. Conclusion**

By following Agile development, we successfully designed and implemented a functional Cedar Woods Accommodation System that met the project specifications. Our structured UML modeling, Java-based implementation, and testing approach ensured a scalable and maintainable system.

Using object-oriented principles and design patterns, we built a modular system that supports future enhancements. Our GUI provided an intuitive user experience, enabling staff to manage accommodations efficiently.

Future improvements could include database integration for persistent storage and cloud-based accessibility to enhance usability beyond the desktop environment. This project effectively demonstrated object-oriented programming, Agile workflows, and software engineering best practices.