OODP Group Project Report guidelines

Project Title:

E.g.: "Smart Parking System" or "Student Course Registration Portal"

👥 **Group Members:**

Name	**Student ID**	**Role (Optional)**
Alice Tan	U1234567A	Project Manager
Bob Lee	U1234567B	Lead Developer
Chen Mei	U1234567C	UML Designer
David Koh	U1234567D	Tester/Documentation Lead

📝 🧠 Chapter 1: Requirement Analysis & Feature Selection

Guidelines for Students: Emphasizing Analytical Thinking

This chapter focuses on how your team **analyzed the BTO project specification** and decided on the **features and scope** of your project. It should showcase your **understanding of the problem**, as well as the **reasoning behind your decisions**.

1.1 Understanding the Problem and Requirements

Start by reflecting on how your team carefully read and dissected the Week 8 document.

🧩 **Your report should address:**

- How did you identify the **main problem domain**?
- What **explicit requirements** were given? (e.g., user login, admin functions)
- What **implicit expectations** did your team infer from the context?
- Were there any **ambiguous or missing parts**? How did you interpret or resolve them?

💬 **Prompt to write:**

"We began by reading through the BTO document line-by-line, highlighting all use cases and system requirements. Based on this, we created a list of essential features and identified user roles and system entities..."

1.2 Deciding on Features and Scope

Explain how your team used your analysis to decide which features would be implemented.

🧭 **Your report should include:**

- A full list of all potential features you identified.
- How your team **prioritized** them based on importance, feasibility, and timeline.
- A clear distinction between:
    - ✅ **Core features** (essential, must implement)
    - 🟡 **Optional or bonus features** (nice to have, only if time permits)
    - ❌ **Excluded features**, and why they were left out

💬 **Prompt to write:**

"We grouped features into three categories: core, optional, and excluded. Our goal was to ensure that the core features demonstrated strong object-oriented principles without making the system overly complex..."

🏗 Chapter 2: System Architecture & Structural Planning

Guidelines for Students: Emphasizing Design Thinking

This chapter focuses on **how you designed your system's structure** after finalizing the functional scope. It should demonstrate the thought process behind your **technical decisions**, especially how you applied **object-oriented thinking** in early planning stages.

2.1 Planning the System Structure

Explain how you designed the **overall system layout** before implementation.

🧱 **Your report should describe:**

- How you **broke down the system into logical components**.
- How you **modeled user flows** or **mapped use cases** to system structure before finalizing your class diagram.
- How you created early visual models such as Flowcharts to show step-by-step process logic

2.2 Reflection on Design Trade-offs

Briefly reflect on the **decisions and compromises** made during planning.

🧠 **Your report should answer:**

- What trade-offs did you consider (e.g., simplicity vs. extensibility)?
- Did your team debate certain approaches before agreeing on a design?

💬 **Prompt to write:**

“We considered combining the controller and logic layer to simplify the codebase, but ultimately separated them to promote maintainability and allow for better testing...”

✅ Summary Checklist for These Chapters

📍 **For Chapter 1:**

- Did you describe how you interpreted the BTO document?
- Did you explain your feature selection and prioritization process?
- Did you justify which features were excluded?

📍 **For Chapter 2:**

- Did you outline your system structure and planned architecture?
- Did you reflect on the design trade-offs or evolution of ideas?
- Did you focus on your thinking process, not just the final result?

🧩 3. Object-Oriented Design

3.1 Class Diagram (with Emphasis on Thinking Process):

This section is not just about drawing a UML diagram, but about clearly articulating how the team translated the **problem domain** into a **logical object-oriented model**.

📌 **Design Thinking Process:**

Before presenting the diagram, include a written reflection addressing the following:

- **How were the main classes identified?**
  E.g., Based on nouns in the problem description, user roles, entities in use cases.

- **What are the responsibilities of each class?**

- **How were relationships (inheritance, association, aggregation) determined?**
  Describe your reasoning when deciding:
  - "Should this be an attribute or its own class?"
  - "Is this an is-a or has-a relationship?"
- **What trade-offs were considered?**
  E.g., Simplicity vs. flexibility, abstraction vs. performance, etc.

🖼️ **Final Output:**

- Present the **UML Class Diagram** showing:
  - Class names
  - Key attributes and methods
  - Visibility indicators (+, -, #)
  - Relationships: inheritance, composition, aggregation, association
- Diagram must be created using a tool and submit the high-resolution file separately.

Include a UML class diagram showing main classes, attributes, methods, relationships (inheritance, composition, etc.)

3.2 Sequence Diagrams (with Emphasis on Thinking Process)

Sequence diagrams illustrate how objects interact over time to fulfill a particular use case. They are a vital tool to show how your system behaves dynamically and ensure that your class design supports real user interactions.

📌 Design Thinking Process: *Why These Diagrams?*

Before drawing the diagrams, the team should reflect and document their thinking process behind the selection:

- What are the most important or complex use cases in your system?
  (E.g., ones involving multiple objects, database access, conditional logic)
- Do these scenarios exercise the most critical parts of your system architecture?
  Your choices should not be arbitrary — the use cases selected should show:

    - How users interact with the core features
    - How your controller-service-repository layers interact
    - How different objects collaborate to fulfill tasks
- What types of patterns or logic are worth demonstrating?
  For example:
    - Authentication flow
    - Search and filter logic
    - Report generation with file output
    - Role-based access control

📑 Required: 2–3 Sequence Diagrams with Justification

Each diagram must include a short justification paragraph that answers:

"Why did we choose this scenario to represent? What does it help us validate or communicate about our design?"

3.3 Application of OOD Principles (SOLID) — *With Emphasis on Thinking Process*

Applying object-oriented principles is not about blindly following rules — it's about thinking critically to create a design that is clear, modular, maintainable, and extensible.

This section serves as a structured reflection. For each principle in SOLID, you are expected to:

- Choose one meaningful and context-relevant example from your own project.
- Explain why you chose to apply the principle there.
- Reflect on how applying the principle improved your design — or what trade-offs were involved.

You are not required to ensure that every single class follows every principle. Instead, your goal is to show that you understand each principle well enough to apply it intentionally and explain your reasoning.

💻 4. Implementation (Java)

4.1 Tools Used:

- Java 17
- IDE: IntelliJ / Eclipse
- Version control: GitHub

4.2 Sample Code Snippets:
Show parts that demonstrate:

- Encapsulation
- Inheritance
- Polymorphism
- Interface use
- Error handling

🧪 Chapter 5: Testing

Guidelines for Students: Demonstrating Functional Validation

Testing is essential to ensure your system works as intended. This section should document your team's **testing strategy**, and include a well-organized list of test cases that reflect the **actual functionality you implemented**.

5.1 Test Strategy

Briefly describe how your team approached testing the system.

🧪 Your report should mention:

- What type(s) of testing you used (e.g., unit testing, manual functional testing)
- Tools used (if any)

5.2 Test Case Table

You should include a complete table of **test cases**, with sufficient variety to cover your system's key functionalities. You are **encouraged to propose new or revised test cases** if:

- Your system includes additional or modified features not in the original Week 8 document
- You made changes to workflows or flows that affect system behavior

- You want to demonstrate deeper understanding by testing edge cases, failures, or alternative paths

✅ **Do not just copy and paste original test cases. Reflect what your system actually does.**

✅ **Summary Checklist for This Section**

- Did you describe your testing approach and tools used?
- Did you review and **update test cases** based on what your system actually implements?
- Did your test cases include a variety of scenarios (valid, invalid, edge cases)?
- Did your test cases demonstrate understanding of the feature behaviors?

📚 6. Documentation

6.1 Javadoc :

✅ All public classes/methods documented with Javadoc

✅ HTML Javadoc files generated and included

6.2 Developer Guide:

How to set up the environment and build the project.

🔍 7. Reflection & Challenges

- What went well
- What could be improved
- Individual contributions
- Lessons learned about OODP

📎 8. Appendix

- GitHub link (if any)
- References (if any external libraries/tools used)

Separate submissions (zip everything in a zip file):

- Full UML diagrams
- Source code