**Business Requirements:**

* Clearly define the problem the system aims to solve.
* Specify the functionalities the system needs to provide.
* Identify the target users and their needs.
* Outline any business goals the system should support

**Non-Functional Requirements:**

* Define performance requirements like scalability, response time, and throughput.
* Specify security requirements like authentication, authorization, and data encryption.
* Outline maintainability requirements like code modularity, documentation, and testing strategies.
* Indicate any other non-functional requirements relevant to the system's success

1. UML Use Case Diagram (10 points):
   * Create a visual representation of the system's actors (users and external systems) and their interactions with the system. Use Cases not complete, need for sequence diagram
2. UML Domain Model (10 points):
   * Identify key entities and their relationships within the problem domain, independent of any specific technology.
3. UML Class Diagram (10 points):
   * Translate the domain model into a set of classes, their attributes, and relationships, reflecting the system's functionality.  ***SUNDAY***
4. UML Sequence Diagrams (10 points):
   * Show the message flow between objects participating in specific use cases, depicting the interaction sequence ***SUNDAY***
5. UML State Diagram (10 points):
   * Illustrate the possible states and transitions an object can undergo throughout its lifecycle within the system. ***SUNDAY***
6. UML Activity Diagram (Swimlane Diagram) (10 points):
   * Visually represent the activities and flows within a specific process, highlighting the responsibility of different actors using swimlanes ***SUNDAY***
7. UML Component Diagram (10 points):
   * Depict the system's physical components and their dependencies, providing a high-level architectural view ***SUNDAY***
8. Cloud Deployment Diagram (10 points):
   * Illustrate the chosen cloud platform (e.g., AWS, Azure) and how the system's components will be deployed within it
9. Skeleton Classes and Tables Definition (10 points):
   * Provide basic outlines of the main classes involved, including their attributes and methods.
   * Define the structure of any database tables required to store system data
10. Design Patterns (10 points):
    * Explain any GRASP, SOLID, GOF, Microservices design patterns and best practices implemented in your design, justifying their use for specific scenarios.

Documentation and Explanation (40 points)

a. Document: Documenting Your Design Decisions

a. Compile all design work into a well-structured GitHub project with

folders for:

i. Images

ii. Scripts

iii. PowerPoint

iv. Word

b. Include a README.md file explaining:

i. Clear explanations for each UML diagram and design component.

c. Links to sample projects demonstrating similar design approaches:

i. <https://github.com/aws-samples>

ii. <https://github.com/Azure-Samples>

iii. <https://github.com/aws-samples/amazon-bedrock-kendra-lex-chatbot>

iv. <https://github.com/aws-samples/generative-ai-amazon-bedrock-langchain-agent-example>

v. <https://github.com/aws-samples/aws-refarch-wordpress>

d. Links to sample blogs demonstrating similar design implementations:

i. <https://aws.amazon.com/blogs/machine-learning/deploy-generative-ai-self-service-question-answering-using-the-qnabot-on-aws-solution-powered-by-amazon-lex-with-amazon-kendra-and-amazon-bedrock/>

ii. <https://aws.amazon.com/blogs/machine-learning/build-generative-ai-agents-with-amazon-bedrock-amazon-dynamodb-amazon-kendra-amazon-lex-and-langchain/>

b. Video Guide: Presenting Your Design Decisions

Objective:

• Create a clear and concise video (8-16 minutes) explaining your

software design decisions, learnings, and rationale.

Target Audience:

• Technical audience familiar with software design concepts.

Video Structure:

• Introduction (1-2 minutes):

o Briefly introduce yourself and the project.

o Briefly state the problem statement you addressed.

• Design Solution Overview (2-3 minutes):

o Briefly explain your high-level solution approach and its key components.

o Mention any relevant design patterns or principles applied.

• Detailed Explanation (5-8 minutes):

• Use visuals: Utilize diagrams (UML, architecture, etc.) throughout your explanation.

• Follow the recommended breakdown:

o Problem Statement: Briefly re-emphasize the problem the system aims to solve.

o Use Case Diagram: Explain the actors (users, systems) and their interactions with the system.

o Activity/Sequence Diagram: Illustrate the business flow or information flow between objects.

o Domain Model/Class Diagram: Explain key classes, their attributes, and relationships.

o State Chart/Machine Diagram: Explain object state transitions and subsequent actions.

o Component Diagram: Describe the system's components and their dependencies.

o Architecture Pattern: Explain the chosen architecture (e.g., Layered, Microservices, serverless, distributed) and its justification.

o Design Principles & Patterns: Explain how specific principles (SOLID, GRASP, etc.) or patterns (GOF, Cloud Native) influenced your design

• Conclusion and Next Steps (1-2 minutes):

o Briefly summarize the key takeaways from your design process.  
o Discuss how this project helped you acquire software design skills and how it might benefit your career