Assignment Objective:

To simulate the real-world software development lifecycle by collaboratively designing, developing, and deploying an object-oriented application using advanced OOP concepts, popular design patterns, software testing, and GitHub for version control. The project will also involve creating a Software Requirements Specification (SRS) documentation and UI/UX prototyping using Figma.

Task Overview:

Step	What to Do
SRS documentation	You can follow the SRS template for documentation and list out all user stories related to your selected project.
Make a UI/UX design and prototyping of your software	Design Landing Page / Dashboard and other Functional UI components from your SRS documentation.
Map Features->Problem->Patterns	Use the design pattern matrix of the following for mapping your functional development
Select at least 7 Design Patterns for implementing backend	Pick ones that make sense for their solution
Connect Patterns with OOP and implementation of OOP in the backend	Shows how OOP and Design Patterns work together in the code
Synchronous Frontend design with backend	Implement a frontend that communicates with the backend.
Software Functional Testing for backend coding	Unit testing of backend logic. Tools: Mocha, Chai.
Sofware Testing through POSTMAN	• Build a **Postman Collection,** Test CRUD APIs (create job, list jobs, get status, retry job, etc.). • Export collection for submission.
Team Collaboration via GitHub	Create a GitHub repository for the project and do teamwork for pushing, pulling, branching, and resolving conflicts.
Deploy your project in AWS	CI/CD Pipeline
Document Usage	Make a complete report based on provided template

Task 1: Software Requirements Specification (SRS)

- Write a **Software Requirements Specification (SRS)** document that includes:
    - **Purpose** – Why the system is being developed.

- Problem Statement
- **Scope** – Overview of the product, its goals, and benefits.
- **User Characteristics** – Target users and their needs.
- **Constraints** – Limitations like regulatory, hardware, or technology restrictions.
- **Functional Requirements** – Detailed description of system functions (e.g., “The system shall allow users to log in with email and password”).
- **Non-Functional Requirements (NFRs)** – Performance, reliability, security, usability, scalability.
- User interface mockups/wireframes (Low Fidelity Design)
- Complete System Diagram
- Safety Considerations.
- Risk Management

Task 2: UI/UX Design and Prototyping Using Figma - 8

- Based on the SRS and Low Fidelity Design, design the **User Interface (UI)** in Figma.
- Share the link with your peers and collaborate (we will check individual contributions).
- Do **prototyping** on your design and **share the production level URL in the report**
- Share the link to the Figma project in report.

Task 3: Implementation (Coding) Using Design Pattern and OOP Principles Using the following Matrix. (You must extend two more epics in your group project)

3.1 Design Pattern Selection Matrix (Example usages)

Example Project Feature	Suggested Design Pattern	How to Use It
Integrating multiple sandbox payment gateways (e.g., Stripe, PayPal)	Adapter	Create a common interface to interact with different external services
Allowing users to customize dashboards with widgets	Decorator	Wrap base dashboard objects with additional features like graphs, alerts, etc.
Hiding system complexity and giving a clean interface (e.g., project management module)	Facade	Combine submodules like TaskManager, Calendar, and Notifier behind a single interface
Dynamically creating objects for users (Admin, Member, Guest)	Factory	Return specific user class based on role or credentials

Handling request pipeline (e.g., request logging, validation, authentication)	Middleware / Chain of Responsibility	Each middleware processes the request and passes it to the next handler
Broadcasting updates (e.g., task assigned, message sent)	Observer	Notify all observers (e.g., team members) when a change occurs
Duplicating object configurations (e.g., template settings, profiles)	Prototype	Clone an existing object with the same settings instead of rebuilding from scratch
Restricting access to sensitive data (e.g., only Admins can view financials)	Proxy	Use a proxy to control access to sensitive operations based on role
Application configuration or logging instance	Singleton	One instance of configuration or logger throughout the project
Switching between strategies at runtime (e.g., sorting by date, priority, or status)	Strategy	Define interchangeable sorting or filtering strategies and switch them as needed

3.2 Implementation Using OOP Principles

Step	What to Do	Why It Matters
Apply OOP Concepts	Use these basic concepts: • **Classes & Objects** – to represent things like User, Project, Task • **Inheritance** – for code reuse (e.g., Admin and Member both inherit from User) • **Encapsulation** – keep internal logic hidden • **Polymorphism** – same method, different behavior (Use of method overloading and overriding)	Makes your code clean, indented, reusable, and easy to maintain.

You Must include:

- At least 5 interacting classes with the implementation of OOP principles (inherence, polymorphism, encapsulation, abstraction). Each class should implement at least one OOP principals.
- Use of at least 7 design patterns
- In report, you must explain the following:

OOP Explanation:

- -Why you defined each class
- -Where inheritance is used
- -How encapsulation is applied
- -Where polymorphism appears

Design Pattern Explanation:

- Which 7 patterns were used
- Where they are used in your code
- Why each pattern fits your problem

Task 4: Team Collaboration via GitHub

- Create a new GitHub repository for the project
- Share your project to your team members
- Use branches for feature development
- Use pulls requests and code reviews for other members of your project
- Practice merging conflicts and mention it in the report (Minimum 2)
- Maintain a **README.md** with setup instructions
- We will check all commits which are initiated by different team members.

Task 5: Functional Testing (only unit testing)

- Test each of your backend functionality (such as create task, update task, etc.) using **unit testing**

Task 6: API Testing using Postman

- Test the endpoint of your backends functionality using **Postman**
- Screenshots of Postman test cases or **Swagger documentation (If we cover)**

Example Endpoints to Test (It must be based on your project):

- POST /login
- GET /users
- POST /notifications
- PUT /settings/{id}
- DELETE /account/{id}
- And other functionality as well

Task 7: CI/CD Pipeline

Follow the submission template for this section.

Task 8: Report

- Live demo of the product Including Figma Design
- Submit a **final report** with the template provided.
- Project GitHub Link and public IP address must be included in the report.