**Week 10 Plan**

**Day 1: SOLID Principles + Design Patterns + Security Concepts**

**Theory**

**SOLID Principles**

* Introduction to SOLID
  + **S**ingle Responsibility Principle
  + **O**pen/Closed Principle
  + **L**iskov Substitution Principle
  + **I**nterface Segregation Principle
  + **D**ependency Inversion Principle
* Simple BFSI examples for each

**Design Patterns Overview**

* Classification of Design Patterns:
  + **Creational** (Singleton, Factory)
  + **Structural** (Adapter, Decorator)
  + **Behavioral** (Observer, Strategy)
* Real-world BFSI examples
  + Factory Pattern for Account creation
  + Observer Pattern for Transaction alerts
  + Strategy Pattern for Interest calculation

**Security – Authentication & Authorization**

* Basics of Authentication and Authorization
* Token-Based Authentication – Why and How?
* Introduction to JWT
  + Structure (Header, Payload, Signature)
  + Token Validation, Expiry
* OAuth 2.0 Concepts
* Role-based Access Control using JWT scopes and roles

**Assignments (Classroom Practical Session)**

**1. Factory Pattern for Account Management**

Build a simple Factory implementation that returns different account types (Savings, Fixed Deposit, Recurring Deposit) based on user input.

**2. JWT Token Generator + Validator (Simulated)**

Create a simple Java app that:

* Issues a JWT token after mock login
* Validates the token and restricts access to a "Customer Dashboard"
* Use hardcoded roles (e.g., customer, admin) and allow/deny access accordingly

**3. SOLID Violation & Refactor Task**

Given a poorly designed **BankAccountManager** class that violates multiple SOLID principles, identify and refactor it following SRP and OCP. (Starter code will be given.)

**Day 2: Automation Testing using Selenium**

**Theory**

**Selenium Basics**

* What is Selenium?
* WebDriver Architecture
* Setting up with Java/Python
* Locators and Element Interaction
* Handling Forms and Assertions
* Sample BFSI Scenarios: Loan form, RD/FD calculators

**Assignments (Classroom Practical Session)**

**1. Automate Fixed Deposit Form**

* Automate filling out a sample FD form: deposit amount, tenure, rate
* Submit the form and validate success message or result
* Optional: Capture and print result (like maturity amount)

**2. Recurring Deposit Setup Automation**

* Test an RD Setup form with monthly contribution input
* Validate if the Total Investment amount shown is correct based on tenure
* Add negative test case: what if user enters non-numeric data?

**3. Form Element Validation**

* Write test cases that:
  + Check presence of required fields
  + Ensure dropdowns have correct options (e.g., "Tenure" values)
  + Validate that "Submit" button is enabled only after valid input

**Day 3: Domain Training (NBFC)**

**Day 4: DevOps – Introduction to ESB, Kafka, WSO2 Overview + CI/CD Using Jenkins**

**Theory**

**Enterprise Messaging Concepts**

* What is ESB?
* BFSI Use Cases for Integration (Loan Approval Workflow, Notifications)
* Kafka Overview
  + Producer/Consumer Model
  + Message Queues in Action (e.g., Payment Status Updates)
* WSO2 as ESB

**CI/CD Basics**

* Jenkins Introduction
* Pipelines, Jobs, Triggers
* CI/CD in Banking Projects
* Hands-on Demo: Sample Java + React build with Jenkins

**Assignments (Classroom Practical Session)**

**1. Kafka Message Simulation**

Simulate a simple message flow using Kafka CLI:

* Producer sends loan application update
* Consumer receives update for processing

**2. Jenkins CI Pipeline for Loan Management App**

Create a basic Jenkins pipeline to:

* Pull source code from Git
* Build project (Maven/npm)
* Archive artifacts

**Day 4: DevOps – Docker**

**Theory**

* What is Docker and Why Docker?
* Images, Containers, Volumes, Networks
* Docker in BFSI DevOps – Use Cases
* Dockerfile Basics
* Hosting a Full Stack App with Docker
* Docker vs Traditional VM setup

**Assignments (Classroom Practical Session)**

**1. Dockerize a React + Spring Boot Loan App**

Create Dockerfiles for frontend and backend  
Run them as separate containers and connect

**2. Create Custom Image for FD Calculator**

Write a Dockerfile to run an FD calculator app (React or Angular based) inside a container