SKILL WORKBOOK

24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

TEAM FSAD
K L UNIVERSITY | VADDESWARAM

SKILL WORKBOOK

24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L - FSAD

STUDENT NAME	
STUDENT ID	
YEAR	
SEMESTER	
SECTION	
FACULTY NAME	

DEPARTMENT VISION AND MISSION

Vision

To be a globally competent in computing education and research, fostering innovation, industry collaboration, and technological advancements for societal impact.

Mission

M1: To impart high-quality computing education that integrates theory, practice, and industry relevance.
M2: To advance research and innovation in emerging global technologies for real- world impact.
M3: To foster an entrepreneurial mindset, interdisciplinary collaboration, and lifelong learning.
M4: To promote responsible and ethical computing for the benefit of society.
M5: To prepare graduates for leadership in the global technology ecosystem.

Program Educational Objectives

1. Practice engineering in a broad range of industrial, societal and real-world applications.
2. Pursue advanced education, research and development, by adapting creative and innovative practices in their professional careers.
3. Conduct themselves in a responsible, professional, and ethical manner.
4. Participate as leaders in their fields of expertise and in activities that support service and economic development throughout the world.

PROGRAM OUTCOMES		
PO	**Graduate Attributes**	**Program Outcome Description**
1	**Engineering Knowledge**	To impart mathematics, science, & engineering knowledge to develop skills to solve complex engineering problems.
2	**Problem Analysis**	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	**Design/ development of solutions**	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4	**Conduct investigations of complex problems**	An ability to use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
5	**Modern tool usage**	Ability to create, select and apply appropriate techniques, resources and modern engineering activities, while understanding its limitations.
6	**The engineer and society**	Ability to apply reasoning and the contextual knowledge to assess social & health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practices.
7	**Environment and sustainability**	Ability to demonstrate the engineering knowledge to find solutions to contemporary issues by understanding their impact on societal and environmental contexts, towards sustainable development
8	**Ethics**	An ability to apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
9	**Individual and teamwork**	To inculcate abilities to be able to act as a leader as well as team player effectively in multi-disciplinary settings
10	**Communication**	To develop oral and written communication skills to articulate the complex engineering activities with the engineering community and society effectively through reports and design documentation, make effective presentations, and give and receive clear instructions.
11	**Project management and finance**	To develop working knowledge and understanding of the engineering and management principles to manage projects in multi-disciplinary environments.
12	**Lifelong learning**	To inculcate the habit of constant knowledge upgrading habit to meet the ever-changing technology and industry needs.
PROGRAM SPECIFIC OUTCOMES		
PSO1	An ability to design and develop software projects as well as to analyze and test user requirements.	
PSO2	Working knowledge on emerging technologies as per the industry requirements	

COURSE COORDINATOR

Mr. J. SURYA KIRAN

Team of Course Instructors

1. DR.G. DINESH KUMAR
2. DR. P. V. VARA PRASAD
3. DR. NICHENAMETLA RAJESH
4. DR. CH.M.H. SAI BABA
5. MR. S. SANDEEP KUMAR
6. DR. SK. MOHAMMAD GOUSE
7. DR. SRIDEVI SAKHAMURI
8. MR. JONNALAGADDA SURYA KIRAN
9. DR. K. ASHESH
10. DR. B. VENKATESWARLU
11. DR. T. PRAVEEN
12. DR. PADMANABAN K
13. DR. A K VELMURUGAN
14. DR. KUNDA VENKATA PRASAD
15. DR. GANGA RAMA KOTESWARA RAO
16. DR.G. VEERRAJU
17. MR. A. PAVAN KUMAR
18. DR. BALAJEE R M
19. MR. AREPALLI GOPI
20. DR.K. NAGARJUNA
21. DR. SUNEETHA BULLA
22. DR. SRITHAR S
23. MR. CHITTIBABU RAVELA
24. DR. SAJJA TULASI KRISHNA
25. MR. P. CHANDRA MOHAN RAI
26. MS. P. SUPRIYA
27. MR. G. DINESHNATH
28. DR. LAKSHMINARAYANA KODAVALI
29. MR. YENGALA AMARAIAH
30. DR. B. L. N. PHANEENDRA KUMAR
31. DR.K. VENKATA GURUNATHAM NAIDU
32. DR. P. PRAVEEN KUMAR
33. DR. K RAMA KRISHNA
34. MR. G TAGORE SAI PRASAD
35. MR. N MURALI KRISHNA
36. MR. CH. NAGARAJU

SKILL EXPERIMENTS

2025-26 EVEN SEMESTER FSAD SKILL CONTINUOUS EVALUATION

#Skill	Date of Evaluation	Implementation (20M)	Output (20M)	Viva Voce (5M)	Git Repo (5M)	Total (50M)	Faculty Signature
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 1 ➔ Git Version Control

Prerequisites:

- **Basic knowledge of Git commands**
- **Ability to create and edit project files**
- **Understanding of version control basics**

The development team wants to manage their project using **Git** to maintain a structured and collaborative workflow. In this lab, you will simulate a real-world version control process: creating project files, tracking and committing changes, using GitHub as remote storage, working with **multiple branches**, and handling merge conflicts. This helps you understand how developers work together efficiently in software projects.

Tasks:

1. Create a new project folder, initialize Git using **git init**, and configure Git with your **username** and **email**.
2. Create two files (example: **main.java**, **notes.txt**) with sample content, check the repository status, and add them to staging.
3. Commit the staged files with a meaningful commit message.
4. Create a new repository on GitHub, connect your local repo using the remote URL and push the committed changes.
5. Create the first branch (example: **feature-update**), make modifications, and **add + commit** the changes.
6. Create a second branch (example: **bug-fix**), apply changes, and **add + commit** them.
7. Switch back to the **main branch** and merge both branches one after the other.
8. If any **merge conflict** occurs, resolve it manually, commit the fix, and complete the merge.

VIVA QUESTIONS:

1. Why is Git used in development?
2. What information is stored in Git configuration?
3. What is Maven and why is it used?
4. What is the role of the POM.xml file?
5. Explain Maven Build Life Cycle.

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 2 ➔ Hibernate CRUD Operations

Prerequisites:

- **Basic knowledge of Java**
- **Understanding of classes, objects, and OOP**
- **Familiarity with SQL basics**

A retail inventory system needs to store product details such as **productId**, **name**, **description**, **price**, and **quantity**. The admin should be able to **add new products**, **retrieve product information**, **update price or quantity**, and **delete discontinued products**.

Your task is to **implement complete CRUD operations** using **Hibernate** with proper entity mapping using **JPA annotations**.

1. Create a **Product entity** with fields: **id (primary key), name, description, price, quantity**.
2. Configure **Hibernate** and map the Product entity to a table using **JPA annotations**.
3. Insert **multiple Product records** into the database.
4. Retrieve a product using its **id**.
5. Update the **price** or **quantity** of any selected product.
6. Delete a product record by **id** if it is discontinued.
7. **Note:**
    a. Implement Hibernate **ID generation strategies** (**AUTO, IDENTITY, SEQUENCE**) inside the entity to observe how primary key values differ.
    b. The **ID can be manual or auto generated**, based on your chosen strategy.
8. **Push the Hibernate project into a single GitHub repository.**

VIVA QUESTIONS:

1. What is ORM and why is it used in application development?
2. Which annotation is used to mark a class as a JPA entity?
3. Why must every Hibernate entity have a primary key?
4. How do you update an existing record using Hibernate?
5. What is the difference between get() and load() in Hibernate?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session: _____to_____**

#SKILL - 3 ➔ Working with HQL - Sorting, Pagination & Aggregates

Prerequisites:

- **Basic knowledge of Hibernate**
- **Basic idea of HQL syntax**

In this lab, you will extend the Hibernate module created in **Skill 2** and perform advanced data retrieval using **HQL**, including **sorting**, **pagination**, **filtering**, and **aggregate functions**. These operations help in retrieving structured and meaningful data from the database using Hibernate.

Tasks:

1. Use the **Product entity** created in Skill 2.
2. Add **5–8 additional Product records** into the database.
3. Write HQL queries to retrieve all products sorted by price:
   a. Ascending order
   b. Descending order
4. Write an HQL query to sort products by quantity (highest first).
5. Implement pagination using HQL to display:
   a. First 3 products
   b. Next 3 products
6. Write HQL queries for aggregate operations:
   a. Count total number of products
   b. Count products where quantity > 0
   c. Count products grouped by description
   d. Find minimum and maximum price
7. Write an HQL query using **GROUP BY** to group products by description.
8. Write an HQL query using **WHERE** to filter products within a price range.
9. Write HQL queries using **LIKE** such as:
   a. Names **starting with** certain letters
   b. Names **ending with** certain letters
   c. Names containing a pattern **anywhere** (substring)
   d. Names with an **exact character length**
10. **Push the Hibernate project to GitHub.**

VIVA QUESTIONS:

1. What is HQL and how does it differ from SQL?
2. Why do we use sorting in HQL?
3. What is pagination and where is it used?
4. Name any three aggregate functions supported in HQL.
5. What is the difference between HQL and HCQL (Hibernate Criteria Query Language)?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session: _____to_____**

#SKILL - 4 ➔ Spring Dependency Injection – Constructor & Setter Injection

Prerequisites:

- **Basic knowledge of Java**
- **General Idea on Spring Framework**
- **Basic Idea on DI & IoC**

A training institute wants to automate the management of student information in a Spring application. The system should inject details such as **studentId**, **name**, **course**, and **academic year** into objects without manually assigning values. This Skill demonstrates **Constructor Injection** and **Setter Injection** using both **XML** and **Annotation** configurations.

Tasks:

1. Create a Java class named **Student** with fields: studentId, name, course, year.
2. Create a **Constructor** that accepts all fields and assigns values.
3. Create **Setter methods** for at least two fields (example: course, year).
4. Configure the Student bean using:
   a. **XML Configuration** - The student should take:
      - a **POJO class**
      - an **XML configuration file**
      - a **main class** to load the container
   b. **Annotation Configuration** - The student should take:
      - a **POJO class**
      - a **Java class with necessary annotations**
      - a **main class** to load the container
5. Create a Spring configuration file (XML or Java-based).
6. Write a main class to load the Spring IoC container.
7. Retrieve the Student bean and print the injected values.
8. **Push the Spring (core) project to GitHub.**

VIVA QUESTIONS:

1. What is Dependency Injection (DI) in Spring?
2. What is the difference between Constructor DI and Setter DI?
3. What is the purpose of the IoC container?
4. Why is DI preferred over manual object creation?
5. Can Constructor Injection and Setter Injection be used together?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session: _____to_____**

#SKILL - 5 ➔ Spring Autowiring Demo using @Autowired

Prerequisites:

- **General Idea on Spring Framework**
- **Modules of Spring Framework**
- **Understanding of Autowiring concepts**

A college application maintains **student** information along with their **certification details**. The **Student** object contains a **Certification** object, and instead of creating this Certification object manually, Spring should automatically provide it using the **@Autowired** annotation. When the required classes are marked with **@Component**, Spring creates their objects and manages them inside the **IoC container**. **Using @Autowired**, the Certification object is supplied to the **Student** object by Spring without using the **new keyword**, allowing both objects to be linked through **annotation-based configuration.**

Tasks:

1. Create a **Certification** class with fields such as id, name, and dateOfCompletion.
2. Create a **Student** class with fields such as id, name, gender, and a Certification object.
3. Annotate both classes with **@Component** to enable Spring detection.
4. Use **@Autowired** on a field, constructor, or setter inside the Student class to inject the Certification object.
5. Configure component scanning using either:
   a. An XML configuration file
   b. A Java configuration class with @Configuration and @ComponentScan
6. Load the Spring IoC container using ApplicationContext.
7. Retrieve the Student bean and print all details, including the injected Certification object.
8. **Push the Spring project to GitHub.**

VIVA QUESTIONS:

1. What is autowiring in Spring?
2. What is the purpose of the @Autowired annotation?
3. How does Spring choose which bean to inject?
4. What is component scanning in Spring?
5. What happens if more than one bean matches the dependency type?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 6 ➔ Spring MVC Web Request Handling Demo

Prerequisites:

- **General Idea on Spring Web MVC and Form Handling**
- **Basic understanding of Spring Boot**
- **Understanding of HTTP methods (GET/POST)**

An online library system wants to allow users to explore books, view specific book details, submit feedback, and interact with the system through various types of requests. To support these features, the backend team needs to implement multiple **Spring MVC endpoints** demonstrating **request mappings**, **path variables**, **request parameters**, and **JSON data handling**. Your task is to create a **Spring Boot MVC controller** that exposes different demo operations using **@RestController**, **@GetMapping**, **@PostMapping**, and **@PathVariable**.

Tasks:

1. Create a controller class named **LibraryController**.
2. Create a method mapped to **/welcome** that returns a welcome message.
3. Create a method mapped to **/count** that returns an integer representing total books.
4. Create a method mapped to **/price** that returns a sample book price as **double**.
5. Create a method mapped to **/books** using **@GetMapping** that returns a list of book titles.
6. Create a method mapped to **/books/{id}** using **@GetMapping** and return book details using **@PathVariable**.
7. Create a method mapped to **/search** using **@GetMapping** that accepts a **request parameter** (title) and returns a confirmation message.
8. Create a method mapped to **/author/{name}** using **@GetMapping** that returns a formatted message with the **author's name**.
9. Create a method mapped to **/addbook** using **@PostMapping** that accepts a **Book object** from the request body and adds it to an **in-memory list**.
10. Create a method mapped to **/viewbooks** using **@GetMapping** that returns all added **Book objects**.
11. **Push the Spring Boot project to GitHub.**

VIVA QUESTIONS:

1. What is the role of @RestController in Spring MVC?
2. What is the difference between @GetMapping and @PostMapping?
3. How does @PathVariable help in handling dynamic URLs?
4. When do we use @RequestParam?
5. Can one controller contain multiple request-mapped methods?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 7 ➔ REST API - CRUD Operations using ResponseEntity

Prerequisites:

- **General Idea on Spring Boot MVC Architecture**
- **Knowledge of Service and Controller layers**

A university application needs backend services to manage **course details** such as **courseId**, **title**, **duration**, and **fee**. The admin should be able to **add**, **update**, **delete**, and **view** course information. The system must return proper **HTTP status codes** and **structured responses** for reliable communication between client and server.

Your task is to implement **REST endpoints** using **ResponseEntity** to perform complete CRUD operations on course data.

Tasks:

1. Create a **Course entity** with fields: **courseId**, **title**, **duration**, **fee**.
2. Create a **Service layer** that performs full CRUD operations.
3. Implement REST endpoints using **@PostMapping**, **@PutMapping**, **@DeleteMapping**, and **@GetMapping**.
4. Use **ResponseEntity** to return:
   a. **Success messages**
   b. **Error messages**
   c. **Status codes** such as **OK**, **CREATED**, **NOT_FOUND**, **BAD_REQUEST**
5. Add a search endpoint **/courses/search/{title}** to filter courses by title.
6. Test all endpoints (valid and invalid cases) **using Postman.**
7. **Push the Spring Boot project to GitHub.**

VIVA QUESTIONS:

1. What is ResponseEntity used for?
2. What is the difference between @PostMapping and @PutMapping?
3. Why are HTTP status codes important in REST APIs?
4. What happens if an invalid ID is passed to an endpoint?
5. Can ResponseEntity return both data and status codes together?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 8 ➔ Spring Boot – JPQL & Query Methods Module

Prerequisites:

- **Basic Idea on Spring Boot and Spring Data JPA**
- **Knowledge of writing derived query methods**

An e-commerce application needs a **product search feature** where users can view products by **category**, filter items within a **price range**, and sort products by **price**. The backend team must implement these operations using **Spring Data JPA**, including **derived query methods** and **JPQL queries**, to fetch data efficiently without writing long SQL statements.

Your task is to build a **product search module** that supports category-based search, price filtering, and sorting using Spring Data JPA.

Tasks:

1. Create a **Product entity** with the following fields:
   a. **id**
   b. **name**
   c. **category**
   d. **price**
2. Create a **ProductRepository** and add derived query methods:
   a. findByCategory(String category)
   b. findByPriceBetween(double min, double max)
3. Write **JPQL queries** using **@Query** for:
   a. Sorting products by price
   b. Fetching products above a price value
   c. Fetching products by category
4. Create **REST controller endpoints** to test query methods:
   a. **/products/category/{category}**
   b. **/products/filter?min=&max=**
   c. **/products/sorted**
   d. **/products/expensive/{price}**
5. Insert sample product records and test using **Postman**.
6. Verify output for different **categories**, **ranges**, and **price values**.
7. **Push the Spring Boot project to GitHub.**

VIVA QUESTIONS:

1. What is JPQL and how is it different from SQL?
2. What are derived query methods in Spring Data JPA?
3. Why is method naming important in Spring Data JPA?
4. Can JPQL return custom objects in a query?
5. When should we choose derived queries instead of JPQL?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 9 ➔ Global Exception Handling using @ControllerAdvice

Prerequisites:

- **Understanding of REST API exception flow**
- **Concept of custom exceptions**

A university application provides **student details** through REST APIs. When users enter an **invalid student ID**, the system shows technical error messages that are difficult for users to understand. To improve usability, the application should return **clear, readable, and user-friendly error responses** through **centralized exception handling**.

Your task is to implement **Global Exception Handling** using **@ControllerAdvice** and create custom exceptions for invalid student inputs.

Tasks:

1. Create a custom exception class named **StudentNotFoundException**.
2. Create a REST endpoint **/student/{id}** that retrieves student information and **throws the custom exception** when the given student ID is invalid.
3. Create a **GlobalExceptionHandler** class using **@ControllerAdvice** to handle exceptions across the entire application.
4. Add a method with **@ExceptionHandler(StudentNotFoundException.class)** to return a **meaningful error response** along with an appropriate HTTP status code.
5. **Add a second custom exception**, such as **InvalidInputException**, and handle it inside the same @ControllerAdvice class.
6. **Return a structured JSON response** (fields like timestamp, message, statusCode) from the exception handler to make the error readable and consistent.
7. Test the API with **valid and invalid student IDs** and observe the difference in responses.
8. Test an endpoint with **invalid input format** (e.g., sending text instead of a number) to trigger the second custom exception **using Postman**.
9. **Push the Spring Boot project to GitHub.**

VIVA QUESTIONS:

1. What is the purpose of @ControllerAdvice in Spring?
2. How does @ExceptionHandler work inside a global handler?
3. Why do applications use global exception handling?
4. Difference between custom exceptions and default exceptions?
5. Can a single @ControllerAdvice class handle multiple exceptions?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session: _____to_____**

#SKILL - 10 ➔ React State Management using useState Object

Prerequisites:

- **Basic Idea of JavaScript**
- **Knowledge of React fundamentals**

An online academic portal wants to maintain a list of **students** where the user can **add new students**, **display the list**, and **delete a student** instantly without refreshing the page. The application should use **React Hooks** to manage data at the component level so that the UI updates automatically whenever the state changes.

Your task is to implement a React component using **useState** to store an **object array**, add new items, display them, and delete them dynamically.

1. Create a new React component named **StudentManager**.
2. Inside the component, create an initial **students array** with at least 5 objects containing:
   a. **id**
   b. **name**
   c. **course**
3. Use **useState** to store:
   a. **students** (array of student objects)
   b. **newStudent** (object with fields like id, name, course)
4. Create **input boxes** to accept id, name, and course.
   i. Update the **newStudent object** using **onChange** events.
5. Create an **Add Student** button.
   i. On click, add the **newStudent object** to the students array.
   ii. After adding, **clear the input fields**.
6. Display the list of students below using the **students state**.
7. Each student row should have a **Delete button**.
   i. On clicking delete, remove the record from the students array.
   ii. The UI should update immediately using **setStudents**.
8. If the list becomes empty, display the message:
   "No students available"
9. Add **basic CSS styling** for a clean UI (margin, padding, table style, buttons).
10. **Push the React project to GitHub.**

VIVA QUESTIONS:

1. What is the purpose of the useState hook in React?
2. How does React update the UI when state changes?
3. Why do we store form inputs in state for controlled components?
4. What is the importance of using keys when rendering lists?
5. Can one component use multiple useState variables? Why?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 11 ➔ React API Integration - Fetching Data Using Fetch API, Axios & Local JSON

Prerequisites:

- **Basic understanding of React**
- **Knowledge of useEffect and useState hooks**
- **Understanding of API calls (fetch / axios)**

A news portal wants to fetch and display user and article information dynamically using **React**. Developers must learn how **useState**, **useEffect**, **fetch()**, **axios**, and **local JSON** work. This Skill demonstrates how frontend applications interact with **local data**, **public APIs**, and **fake APIs**, and how different components are navigated using hyperlinks.

Tasks:

Part A – Fetching from Local JSON

1. Create a **users.json** file inside the public folder with 5–6 user records, and build a **LocalUserList** component using **fetch()** to load and display the data with name, email, phone, and loading/error states.

Part B – Fetching from JSONPlaceholder

2. Create a **UserList** component using **fetch()** to call https://jsonplaceholder.typicode.com/users
   Display name, email, phone with loading and error handling.

Part C – Fetching from Fake API

3. Create a **FakePostList** component using **axios** to call **any ONE API**, such as:
   a. https://dummyjson.com/posts OR
   b. https://fakestoreapi.com/products

Display fields such as title, body, or description, and add a **Refresh** button to reload data.

Part D – Integration & Navigation

4. Create a **Dashboard/Home component** containing three hyperlinks/buttons:
   a. Local Users
   b. Users API
   c. Fake API Posts

Each link should open its respective component.

5. Import **Dashboard** into **App.jsx** so all components can be accessed through hyperlinks instead of writing them directly in App.js.
6. Add a **dropdown filter** in FakePostList (or PostList) to filter data based on API fields (userId or category).
7. Apply **basic CSS styling** to keep the UI neat and readable.
8. **Push the React project to GitHub.**

VIVA QUESTIONS:

1. What is the purpose of useState in React?
2. Why do we use useEffect for API calls?
3. Difference between fetch() and axios?
4. Why does React re-render the UI?
5. What happens if the dependency array is missing in useEffect?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 12 ➔ Full-Stack CRUD Application using React & Spring Boot

Prerequisites:

- **Basic Understanding of Spring Boot**
- **Basic understanding of React**
- **Ability to integrate frontend & backend**

A **student management system** needs full **CRUD operations** where the **React frontend** communicates with a **Spring Boot backend** through **REST APIs**. Users should be able to **add**, **view**, **update**, and **delete** student records through a simple UI, and all operations must update the data immediately without page reloads.

Your task is to build both the **backend (Spring Boot)** and **frontend (React)** parts and integrate them into a working full-stack application.

Tasks:

Backend – Spring Boot

1. Create **CRUD REST endpoints** in Spring Boot:
    a. **POST /students** → Add a new student
    b. **GET /students** → Retrieve all students
    c. **PUT /students/{id}** → Update an existing student
    d. **DELETE /students/{id}** → Delete a student by ID
    e. Implement a **Service layer** and **Repository layer**.
    f. Use **@RestController**, **@PostMapping**, **@GetMapping**, **@PutMapping**, **@DeleteMapping**.
    g. Return responses using **ResponseEntity**.

Frontend – React

2. Create a React component **StudentList**:
    a) Use **axios** to call **GET /students**.
    b) Display all students in a table/list.
    c) Include **Delete** and **Update** buttons for each row.
3. Create a React form **AddStudent**:
    a) Use **useState** to capture form inputs (name, email, course).

b) Submit data to backend using **axios POST**.
c) Clear form after successful submission.

4. Implement **Update and Delete features**:

   a. **Delete:** call axios DELETE and remove the student from UI.
   b. **Update:**
      i. Prefill form with selected student data
      ii. Call axios PUT to update the record

5. Ensure the **StudentList updates automatically** after:

   a) Adding a student
   b) Updating a student
   c) Deleting a student

This can be done using **state updates** or **refetching data** after each operation.

6. **Push the full-stack project into one GitHub repository. The repository must contain two folders with the following structure:**
   • frontend/ → React code
   • backend/ → Spring Boot code

VIVA QUESTIONS:

1. How does React connect with Spring Boot in a full-stack application?
2. What is Axios used for in React?
3. What is the difference between state and props?
4. Why do we use REST endpoints in backend development?
5. How is form data handled in React components?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session: _____to_____**

#SKILL - 13 ➔ Deployment of Full-Stack Application (Spring Boot + React)

Prerequisites:

- **Basic Idea on Spring Boot**
- **Knowledge of React build process**
- **Basic knowledge of hosting (Nginx/Apache)**

A company wants to deploy a **production-ready full-stack application**. Developers must prepare the **React frontend** and **Spring Boot backend** for deployment, configure **environment variables**, host the application, and verify functionality after deployment.

Tasks:

1. Generate the **React production build**.
2. Package the **Spring Boot backend** as a JAR file.
3. Configure **environment variables** in the React build.
4. Run the backend JAR and verify that all **APIs are available**.
5. Deploy the React build using **Nginx/Apache** or through the **Spring Boot static folder**.
6. Test the deployed application in a web browser to ensure frontend–backend integration works.

VIVA QUESTIONS:

1. Difference between development build and production build?
2. How is a Spring Boot JAR deployed?
3. Why are environment variables important in deployment?
4. What is an Nginx reverse proxy?
5. Why should we test the application after deployment?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session: _____to_____**

#SKILL - 14 ➔ User Authentication & Session Management using React

Prerequisites:

- **Basic Idea on React**
- **Basic idea of authentication workflow**
- **Ability to call backend APIs**

A web application requires a **user authentication system** where users can **register**, **log in**, and after successful login, they are redirected to a **Home page**. The application must store the logged-in user info in **localStorage or sessionStorage** and then use that stored information to **fetch the complete user profile from the backend database**.

The retrieved user details must be displayed on a **Profile page**, and users must be able to **log out**, which clears the session and redirects them to Login.

Tasks:

1. Create a **Register component** using useState and submit the form to the backend to save the user in the database, then redirect to the Login page.
2. Create a **Login component** that validates the user by calling the backend API and, if credentials are correct, stores the username or userId in localStorage/sessionStorage and redirects to Home.
3. Create a **Home component** that loads only when a logged-in user exists in storage.
4. Create a **Profile component** that reads the stored userId/username, calls the backend to fetch full user details, and displays the information.
5. Redirect the user to the Login page if no logged-in user is found in storage.
6. Implement a **Logout button** that clears the stored user data and redirects to Login.
7. Add simple navigation links for **Home**, **Profile**, and **Logout**.
8. Apply basic **CSS styling** to make the UI clean and readable.
9. **Push the full-stack project into one GitHub repository. The repository must contain two folders with the following structure**:
   • frontend/ → React code
   • backend/ → Spring Boot code

VIVA QUESTIONS:

1. What is the difference between localStorage and sessionStorage?
2. How do we fetch user details based on logged-in user info?
3. Why do we store login information in browser storage?
4. What is the purpose of protected pages in authentication?
5. How is logout implemented in React applications?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 15 ➔ Implementing JWT-Based Authentication & Role Authorization

Prerequisites:

- **Basic Idea on Spring Security**
- **Basic Idea on JWT Tokens & RBAC**

A corporate portal requires authentication using **JWT tokens** with **role-based access**. Only **ADMIN** users should manage employee records, while **EMPLOYEE** users can access profile-related data.Your task is to implement **JWT-based secure login** and restrict access to specific endpoints.

Tasks:

1. Create **User entity** with **username**, **password**, and **role**.
2. Implement **/login** to generate a **JWT token**.
3. Configure **Spring Security** with a **JWT filter**.
4. Create secured endpoints:
   a. **/admin/add**
   b. **/admin/delete**
   c. **/employee/profile**
5. Test **authentication** and **authorization** using **valid and invalid JWT tokens**.
6. Test all secured endpoints using **Postman** (with and without JWT tokens).
7. **Push the complete backend project into GitHub inside a single repository**.

VIVA QUESTIONS:

1. What is JWT?
2. How does Spring validate tokens?
3. Authentication vs Authorization?
4. Why are roles needed?
5. What happens when a token expires?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation:

DEPARTMENT OF CSE, CS&IT, AI&DS
COURSE CODE: 24SDCS02 / 24SDCS02E / 24SDCS02P / 24SDCS02L
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/____ **Time of The Session: _____to_____**

#SKILL - 16 ➔ API Documentation for Full-Stack Student CRUD Application using Swagger

Prerequisites:

- **Basic Idea on Spring Boot**
- **Concept of API documentation**

The student management system developed in **SKILL – 12** includes full **CRUD operations** using **Spring Boot REST APIs**. To help testers, developers, and frontend teams understand each endpoint clearly, the backend must include **Swagger/OpenAPI documentation**. Swagger UI must display all **Student CRUD endpoints**, response structures, validation messages, and example requests.

Your task is to integrate **Swagger/OpenAPI** into the existing Spring Boot project from SKILL 12 and document all the APIs.

Tasks:

Part–I: Swagger/OpenAPI Setup:

1. Add **Swagger/OpenAPI dependency** to the Spring Boot project (pom.xml).
2. Create a **SwaggerConfig** class (if needed) to enable OpenAPI documentation.
3. Start the application and open:
    a. **http://localhost:8080/swagger-ui.html** or
    b. **/swagger-ui/index.html**

Part–II: Documenting Student CRUD APIs

4. Ensure the following **Student CRUD endpoints** appear in Swagger UI:
    a. **POST /students** → Add a new student
    b. **GET /students** → Retrieve all students
    c. **GET /students/{id}** → Retrieve a student by ID
    d. **PUT /students/{id}** → Update a student
    e. **DELETE /students/{id}** → Delete a student
5. Enable automatic schema/model generation for the **Student entity**, including fields:
    a) **id**
    b) **name**

c) **email**
d) **course**

6. Write optional **@Operation** and **@ApiResponse** annotations to describe:

a) What each endpoint does
b) Expected success responses
c) Error responses

Part–III: Testing Through Swagger UI

7. Use Swagger UI to test the following:
   a. **Adding a new student**
   b. **Viewing all students**
   c. **Updating student details**
   d. **Deleting a student**
   e. **Handling invalid IDs** (example: 999 → Not Found)

8. Verify Swagger displays proper:

   a. **Schemas (Student model)**

   b. **Validation errors**

   c. **Example request bodies**

   d. **Response structure**

VIVA QUESTIONS:

1. What is Swagger/OpenAPI used for in backend applications?
2. How does Swagger help developers and testers?
3. What is the importance of schema/model documentation?
4. How can we test CRUD endpoints directly through Swagger UI?
5. Can Swagger UI show both error responses and success responses?

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of _____ EMP ID & Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation: