FULL STACK DEVELOPMENT

BRANCH:- PROFESSIONAL ELECTIVE

SEMESTER – FIFTH

These important questions have been prepared using your previous exam papers (PYQs), verified concepts, and additional reference from trusted online academic sources.
For deeper understanding, please refer to your class notes as well.

1️⃣ HIGH & LONG IMPORTANT QUESTIONS

Unit 1 & 2: Introduction and Enterprise Automation

1. **Define Business Process Automation (BPA).** Explain in detail **why** it is necessary to automate business processes in a modern enterprise.
2. **Explain the concept of Digital Transformation** through the convergence of **IT (Information Technology) and OT (Operational Technology)**. Provide a suitable example.

Unit 3: Design Principles and Architecture

3. **Explain the key Design Principles** for enterprise applications: **Availability, Performance, Consistency, Scalability, Manageability, and Cost**. Describe the significance of each.
4. **Compare and Contrast Monolithic Architecture and Microservice Architecture** in detail. Discuss the advantages and disadvantages of using a Microservice approach.

Unit 4: Security and DevOps Practices

5. **Explain different Design Methods for Application Security.** Describe the working and considerations for **Token-based, Cookie-based, and Multi-factor Authentication (MFA)**.
6. **What is Continuous Integration (CI)?** Explain the different **DevOps Engineering Practices** (Configuration Management, Continuous Integration,

Automated Testing, Continuous Delivery, Continuous Deployment, Continuous Monitoring) in the context of an enterprise application lifecycle.

Unit 5: Version Control and Cloud

7. **Explain the fundamentals of Git** as a Version Control System (VCS). Describe the basic local Git operations: **creating a repository, cloning, staging and committing changes, and viewing history**.
8. **Define Cloud Computing.** Explain the three main **Service Models (IaaS, PaaS, SaaS)** and the three main **Deployment Models (Public, Private, Hybrid)** in detail.

Unit 6: Front-End Basics (JS/TS)

9. **Describe the concept of JavaScript Objects,** including **Methods, Constructors, Data Properties, and Accessor Properties.** Explain the role of the **Prototype** in JavaScript.
10. **What is TypeScript?** Discuss the **advantages of using TypeScript** over plain JavaScript for large-scale application development.

Unit 7 & 8: React Framework

11. **Explain the concept of State and Props in React.** Describe how **communication between components** is achieved using Props.
12. **Describe the Component Life Cycle methods in React,** covering the **Mounting, Updating, and Unmounting phases.**

Unit 9: Spring Core and API

13. **Define Inversion of Control (IoC) and Dependency Injection (DI).** Explain the three main **Types of Dependency Injection (Constructor, Property, and Method Injection)** with suitable examples.
14. **What is a RESTful API?** Explain the key **REST Architectural Constraints** and the **Properties of a REST API**.

Unit 10: ORM, Transactions, and NoSQL

15. **Explain the need for Object Relational Mapping (ORM).** Describe the features and benefits of using JPA (Java Persistent API) in a Spring Boot application.
16. **Explain the ACID principles** in the context of database transactions. Describe **Transaction Management** in Spring.

17. **Differentiate between SQL (Structured) and NoSQL (Unstructured/Semi-structured) databases.** Explain the **CAP Theorem** and the **BASE principles** related to NoSQL.

Unit 11 & 12: MongoDB and Testing

18. **Explain the architecture and key components of MongoDB.** Describe the process of **Data Modelling** in a NoSQL environment.
19. **Describe the different types of Application Testing** (Functional Testing, Integration Testing, System Testing, Acceptance Testing). Explain the difference between **Manual and Automated Testing**.

Unit 13: Containerization and Orchestration

20. **What is Docker?** Explain the concept of **Containers** and the working of the main Docker Components (**Docker Image, Docker Container, Docker Client, Docker Daemon, Docker Registry**).

21. **Define Container Orchestration.** Explain the need for an orchestration engine and briefly describe the role of **Kubernetes** in managing containerized applications.

2️⃣ IMPORTANT & SHORT QUESTIONS

These questions cover basic definitions, comparisons, and core conceptual checks.

1. Differentiate between **Epic** and **User Story** in Agile/Scrum methodology.
2. Define **Backlog Refinement** and **Sprint Retrospective**.
3. List the different **Architectural Patterns** mentioned in the syllabus (Monolithic, Layered, SOA, Microservice).
4. Briefly explain the security concepts of **OpenID** and **SAML**.
5. What is **Infrastructure as Code (IaC)**?
6. List and explain the purpose of the three stages of a Git operation: **Working Directory, Staging Area, and Local Repository.**
7. Define **Virtualization** in the context of cloud computing.
8. Write a short note on **JSON** structure and its usage in APIs.
9. Explain the purpose of the **Spread Operator** in ES6.
10. What is the role of the **Virtual DOM** in React?
11. Define **JSX** and state its purpose in React development.

12. What are **Controlled Components** and **Uncontrolled Components** in React Forms?
13. Explain the term **Lifting State Up** in React.
14. What is the difference between a **Function Component** and a **Class Component** in React?
15. Define **Spring Framework** and **SpringBoot**. Why is SpringBoot preferred?
16. List and briefly explain the **HTTP Methods (GET, POST, PUT, DELETE)** used in REST APIs.
17. Write a short note on the purpose of **Spring Data JPA**.
18. Differentiate between **Collection** and **Document** in MongoDB.
19. Explain the concept of **Indexing** in MongoDB.
20. What is a **CI/CD Pipeline**?
21. Briefly explain **Blue-Green Deployment** as a deployment strategy.

3️⃣ "AA BHI SAKTA HAI" QUESTIONS (20–30% probability)

These questions cover new, peripheral, or conceptual extensions.

1. Explain the role of **Design Thinking** in the context of software development.
2. Describe the importance of **cost estimation** and **risk management** in project planning.
3. What is **Component Scanning** in the Spring IoC container?
4. Explain the usage of **React Hooks** like useState and useEffect.
5. What are **Higher-Order Components (HOCs)** in React?
6. Briefly explain the **Controller, Service, and Repository layers** in a typical Spring REST architecture.
7. Explain the function of an **API Gateway**.
8. Define and explain **Static Code Analysis** as a part of the quality assurance process.
9. What are the key elements of a **Disaster Recovery Plan**?
10. Explain the importance of **Load Balancing** in application deployment.

QUICK REVISE

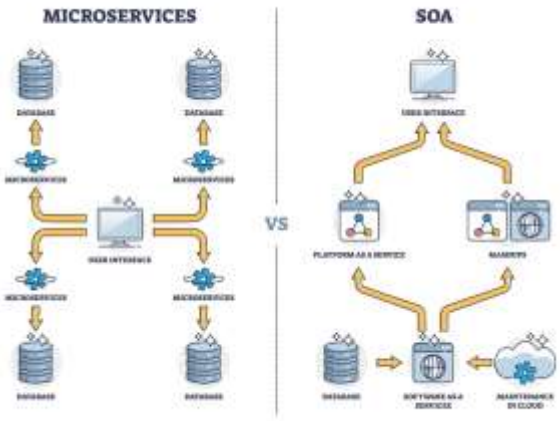
💻 1. Enterprise, Architecture, and DevOps (Units 1-4)

Enterprise and Automation

- **Business Process Automation (BPA):** Using technology to execute recurring tasks/processes in an organization.
    - **Need:** Increases efficiency, reduces errors, saves costs, and improves consistency.
- **Digital Transformation:** Convergence of **IT (Information Technology)** and **OT (Operational Technology)** to fundamentally change how a business operates.
- **Design Thinking:** A non-linear, iterative process used to understand users, challenge assumptions, and redefine problems to create innovative solutions (e.g., in software development).
- **Agile/Scrum Basics:**
    - **Epic:** A large body of work that can be broken down into smaller tasks (user stories).
    - **User Story:** A short, simple description of a feature told from the perspective of the user.
    - **Sprint:** A fixed-length period (usually 2-4 weeks) during which a development team works to complete a set of user stories.

Application Architecture

- **Monolithic Architecture:** A single, unified, indivisible unit containing the entire application code (UI, business logic, data access).
    - *Drawback:* Difficult to scale, deploy, and maintain as it grows.
- **Microservice Architecture:** An application structured as a collection of smaller, independent services, communicating via APIs.
    - *Advantage:* Independent deployment, better scalability, high fault isolation.



- **Design Principles (The "ilities"):**
  - **Scalability:** Ability to handle increased load/users.
  - **Availability:** Application is accessible when needed (uptime).
  - **Performance:** Responsiveness under a given workload (speed).
  - **Consistency:** Data views are the same across the system.
  - **Manageability:** Ease of operating, maintaining, and monitoring the system.

Security and DevOps

- **Authentication:** Verifying who the user is (e.g., username/password).
- **Authorization:** Determining what the user is allowed to do.
  - **Token-based Auth:** Client sends a unique, self-contained token (like JWT) with each request after initial login. **Stateless** (server doesn't need to save session info).
- **DevOps Practices:** Culture and practices that automate and integrate the processes between software development and IT teams.
  - **Continuous Integration (CI):** Developers merge code into a central repository frequently, after which automated builds and tests are run.
  - **Continuous Delivery (CD):** Code changes are automatically built, tested, and prepared for release to production.
  - **Continuous Deployment (CD):** Automatically deploys the code to production if all tests pass.
  - **Infrastructure as Code (IaC):** Managing and provisioning infrastructure through code instead of manual processes.

⚙️ 2. Version Control and Cloud (Unit 5)

Git and Version Control

- **Version Control System (VCS):** A system that records changes to a file or set of files over time.
- **Git Local Workflow (Three States):**
    1. **Working Directory:** Files being modified.
    2. **Staging Area (Index):** Marks changes to be included in the next commit.
    3. **Local Repository:** Contains committed history (.git folder).
- **Basic Commands:** git clone, git add (to stage), git commit (to record changes), git push (to send to remote).

Cloud Fundamentals

- **Virtualization:** Creating a virtual version of a resource (like an OS, server, storage device, etc.).
- **Service Models:**
    - **IaaS (Infrastructure as a Service):** Provides raw computing infrastructure (VMs, storage, networks). *You manage OS, applications.* (e.g., AWS EC2)
    - **PaaS (Platform as a Service):** Provides a platform for developing, running, and managing applications. *You manage only code/data.* (e.g., Google App Engine)
    - **SaaS (Software as a Service):** Provides ready-to-use software over the internet. *You manage nothing*. (e.g., Gmail, Office 365)
- **Deployment Models:**
    - **Public Cloud:** Services offered over the public internet by third-party providers.
    - **Private Cloud:** Services used exclusively by one organization, either on-premise or hosted.
    - **Hybrid Cloud:** A mix of public and private cloud environments.

🌐 3. Front-End Development (Units 6-8)

JavaScript (JS) and TypeScript (TS)

- **JS Objects:** Collections of key-value pairs (properties) and functions (methods).
- **Prototype:** Mechanism by which JS objects inherit features from one another.
- **ES6 Features: Arrow Functions**, **Template Strings**, **let/const**, **Spread Operator (...)**, **Map/Set**.

- **TypeScript (TS):** A superset of JavaScript that adds **static typing** (checking types during development).
    - *Advantage:* Catches errors early, improves code readability and maintainability, especially for large applications.

React Core Concepts

- **React:** A JavaScript library for building user interfaces based on components.
- **JSX:** A syntax extension for JavaScript (looks like HTML) used to describe what the UI should look like.
- **Virtual DOM (VDOM):** An in-memory representation of the real DOM. React compares the VDOM to find minimum necessary changes before updating the real DOM, improving **performance**.
- **Component:** A reusable, independent piece of UI (can be **Function** or **Class** based).
- **Props (Properties):** Used for **read-only data transfer** from a **parent component to a child component** (Downstream communication).
- **State:** A data structure that holds data specific to a component and can **change over time** (mutable). Changes trigger re-rendering.
- **Component Life Cycle (Class):**
    - **Mounting:** constructor(), render(), componentDidMount().
    - **Updating:** shouldComponentUpdate(), render(), componentDidUpdate().
    - **Unmounting:** componentWillUnmount().
- **React Hooks:** Functions that let you "hook into" React state and lifecycle features from function components.
    - useState: To add state to function components.
    - useEffect: To handle side effects (like data fetching, manual DOM changes, etc.).
- **Controlled Components:** Form elements (like <input>) whose value is controlled by React **state**.
- **Uncontrolled Components:** Form elements whose data is handled by the DOM itself (using **Refs**).
- **Lifting State Up:** The process of moving state from a child component to its nearest common ancestor component, allowing siblings to communicate/sync.

🛠️ 4. Back-End (Spring/API) and Databases (Units 9-12)

Spring Framework and Core

- **Spring Framework:** Provides comprehensive infrastructure support for developing robust Java applications.
- **SpringBoot:** An extension of Spring that simplifies development by providing **auto-configuration** and a pre-configured, production-ready environment.
- **Inversion of Control (IoC):** The framework controls the flow of program execution, not the developer (the framework "calls you").
- **Dependency Injection (DI):** A design pattern where an object receives its dependencies from an external source (IoC Container) rather than creating them itself.
    - **Types: Constructor Injection**, **Setter Injection**, **Field/Property Injection**.
- **Spring Bean:** An object managed by the Spring IoC Container.

REST API

- **API (Application Programming Interface):** A set of rules that defines how different software components should interact.
- **REST (Representational State Transfer):** An architectural style for designing networked applications.
- **Key REST Constraints/Properties: Stateless** (server has no session info), **Client-Server Separation**, **Uniform Interface**.
- **HTTP Methods (CRUD Mapping):**
    - **GET:** Read/Retrieve resource.
    - **POST:** Create a new resource.
    - **PUT:** Update/Replace an existing resource (full update).
    - **DELETE:** Remove a resource.
- **Layered Architecture (Spring REST):**

    1. **Controller Layer:** Handles HTTP requests and responses (API Endpoints).
    2. **Service Layer:** Contains the main business logic.
    3. **Repository Layer:** Communicates with the database.

Data Management

- **ORM (Object Relational Mapping):** A technique that converts data between incompatible type systems using an object-oriented programming language (Java) and a relational database (SQL).
    - **JPA (Java Persistence API):** The standard specification/interface for ORM in Java.
    - **Spring Data JPA:** Simplifies the implementation of JPA repositories.

- **ACID Properties (Transactions):** Ensure data integrity in databases.
    - **Atomicity:** Transaction is all-or-nothing.
    - **Consistency:** Transaction moves the database from one valid state to another.
    - **Isolation:** Concurrent transactions don't interfere with each other.
    - **Durability:** Changes are permanent once committed.
- **NoSQL Databases (MongoDB)**
    - **Features:** Schema-less, high scalability, flexible data model.
    - **Data Model:** Documents (JSON-like) stored in Collections.
    - **CAP Theorem:** A distributed system can only guarantee **two** of the three: **Consistency, Availability, Partition Tolerance.**
    - **BASE Principles (NoSQL): Basically Available, Soft State, Eventually Consistent.**

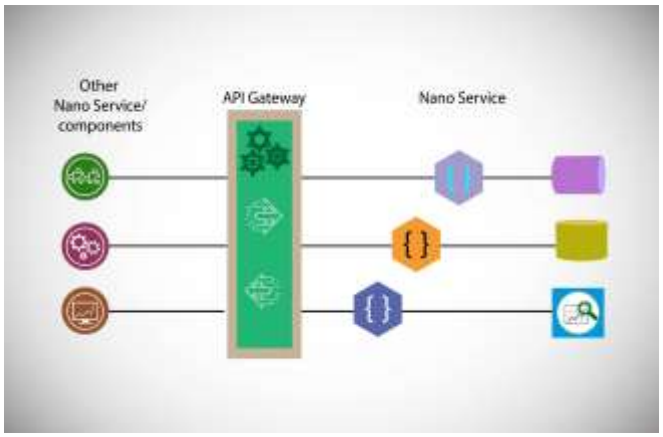
📦 5. Testing, Deployment, and Orchestration (Units 11-13)

Application Testing

- **Functional Testing:** Tests specific features/functions of the software.
- **Unit Testing:** Tests the smallest testable parts of an application (e.g., individual methods or classes).
- **Integration Testing:** Tests how modules/components interact with each other (e.g., Service Layer to Repository Layer).
- **System Testing:** Tests the entire system against the requirements.
- **Acceptance Testing (UAT):** Formal testing to verify if the system meets user requirements.

Deployment and Containers

- **Static Code Analysis:** Analyzing source code *without* executing it to detect errors, bugs, and security vulnerabilities.
- **Containers (Docker):** A lightweight, standalone, executable package of software that includes everything needed to run it (code, runtime, libraries, settings).
    - **Docker Image:** A read-only template with instructions for creating a Docker container.
    - **Docker Container:** A runnable instance of a Docker image.
- **Container Orchestration (Kubernetes - K8s):** Automation of deployment, scaling, and management of containerized applications.



- **Deployment Strategies:**
  - **Blue-Green Deployment:** Run two identical production environments ("Blue" is current, "Green" is new). Switch traffic from Blue to Green instantly after testing.
  - **Canary Deployment:** Gradually roll out a new version to a small subset of users before a full rollout.

System Management

- **Disaster Recovery (DR):** The process, policies, and procedures related to preparing for recovery or continuation of vital technology infrastructure after a natural or human-induced disaster.
- **Load Balancing:** Distributing network traffic across multiple servers to ensure no single server is overworked, improving availability and responsiveness.