V Semester Diploma Examination, June/July - 2023

FULL STACK DEVELOPMENT - 20CS52I

Question Paper

Instructions: i) Answer **one** full question from each section.
ii) Each section carries **20** Marks.

SECTION-I

1. a) Explain how Digital transformation can bring revolution in banking process. **-10**

b) Explain how design thinking has brought revolution in scanner experience that loved **-10**

2 .a) Discuss the different steps involved in organizing an enterprise **-10**

b) Identify the following cloud service types and list their characteristics and advantages.

Cisco WebEx, Google App Engine, Amazon EC2. **-10**

SECTION-II

3. a) BookMyShow is an online movie ticket booking application that helps its user to book movie tickets by logging in. Users can find their movie from the listings. Once found, User an check the availability of tickets for specific date and book ticket. After booking is confirm the details are sent to user. Identify and write the user stories for this application. **-12**

b) Write test cases for the above application. **-8**

4. a) Swiggy is an online food ordering application that helps its users to buy variety of authentic food items. This application allows users to log in for ordering food. Users can search for their favorite food based on rating or price. Users can select the items and add to the cart. Once the selection made go to payment page and make payment. Write the user stories for this application. **-12**

b) Write test cases for the above application. **-8**

SECTION-III

5. a) Create a spring Boot application to maintain employee details such as employee id, employee name, and department. Design entity or data JPA class, controller class and repository. **-12**

b) Compare spring and spring boot. **-8**

6. a) Create a login form using React JS **-10**

b) Compare the database through JDBC and ORM. **-10**

SECTION-IV

7. a) Discuss how REST API is best over SOAP API illustrate. **-10**

b) How to convert monolithic application to microservice architecture? Explain with an example. **-10**

8. a) Illustrate ACID transaction in MangoDB **- 10**

b) Create a REST controller class to perform CRUD operations on productand corresponding request and response DTOs. The product class should contain three data members

product name, product category, price. Use proper SpringBoot annotations **-10**

SECTION-V

9. a) Discuss the Components of Docker Container. **-10**

b) Discuss the difference in manual deployment and automated deployment **-10**

10 . a) Create a form to add a new product detail to the product catalogue using React. **-10**

b) Draw the CI/CD build process flow diagram for an online application and explain each component. **-10**

V Semester Diploma Examination, June/July - 2023

FULL STACK DEVELOPMENT - 20CS52I

Scheme of Valuation

Instructions: i) Answer **one** full question from each section.
ii) Each section carries **20** Marks.

Section-I

1. a) Explanation of Digital Transformation any 10 points related to banking Digital Transformation process. **10 Marks**

b) Explanation of Design Thinking **10 Marks**

2. a) Explanation of atleast 10 steps Each carries **1Marks=10*1=10Marks**

b) CiscoWebEx-Identification-**1Marks+**Characteristics **1Marks** + advantages-**1Marks = 3Marks**

GoogleAppEngine-Identification**1Marks+**Characteristics**1Marks**+advantages**1Marks = 3 Marks**

Amazon EC2 - Identification **1Marks +** Characteristics **2Marks +** advantages **1Marks = 4 Marks**
3+3+4=10Marks

Section-II

3. a) Identification of User Stories-**3Marks +** Writing of User Stories- **9Marks = 12Marks**

b) Writing of Test Cases **-8 Marks**

4 . a) Identification of User Stories-**3Marks +** Writing of User Stories **-9Marks= 12Marks**

b) Writing of Test Cases **- 8 Marks**

Section-III

5. a) Entity code **3Marks +** repository code **2Marks +**controller class code **7Marks -12 Marks**

b) Comparison **-8 Marks**

6. a) Login form creation code **-10 Marks**

b) comparison **-10 Marks**

Section-IV

7.a) Differences of REST API **-5Marks** + differences of SOAP API **-5 Marks = 10Marks**

b) Conversion steps Mentioned **2Marks** + steps explanation **6Marks** + example **2Marks** = **10Marks**

8 .a) Acid properties mentioned **2Marks +**Explanation of acid Properties **8Marks -10 Marks**

b) Product entity code **2Marks**+ request and response DTOs 2Marks each=**2*2M=4Marks**
+ controller class code **4Marks = 2+4+4Marks= 10Marks**

Section-V

9. a) Docker Client-**3Marks +** Docker Host-**5 Marks +** Docker Registries-**2Marks** = **10Marks**

b) Differences **-10 Marks**

10. a) Writing code to add new product. **-10 Marks**

b) Diagram **4Marks** + Components **6 Marks** **-10 Marks**

V Semester Diploma Examination, June/July - 2023

FULL STACK DEVELOPMENT - 20CS52I

Model Answers

Instructions: i) Answer **one** full question from each section.
ii) Each section carries **20** Marks.

Section-I

1. a) Explain how Digital transformation can bring revolution in banking process. -10

Digital transformation has the potential to revolutionize the banking process in numerous ways, leading to enhanced efficiency, improved customer experience, and increased innovation. Some key areas where digital transformation can bring about a revolution in the banking industry:

1.Digital Banking Services: Digital transformation enables banks to offer a wide range of digital banking services, including online banking, mobile banking apps, and virtual wallets. Customers can access their accounts, make transactions, pay bills, and manage finances conveniently from their devices, reducing the need for physical visits to the bank.

2. Personalized Customer Experience: With data analytics and artificial intelligence, banks can analyze customer behavior and preferences to offer personalized services and product recommendations. Digital channels allow for targeted marketing and tailored experiences, making customers feel valued and engaged.

3. Seamless On boarding and KYC: Digital transformation streamlines customer onboarding and Know Your Customer (KYC) processes. Customers can open accounts online with minimal paperwork, reducing the time and effort required for account setup and verification.

4. Enhanced Security: Digital transformation can strengthen security measures through biometric authentication, two-factor authentication, and robust encryption. This provides customers with a secure banking experience, protecting their financial data from cyber threats.

5. Automated Processes and AI-powered Chat bots: Banks can automate various processes, such as loan approvals and account verifications, using artificial intelligence. AI-powered chatbots provide 24/7 customer support, resolving queries and offering assistance in real-time.

6. Faster Transactions and Real-time Payments: Digital transformation enables real-time payments and faster transactions through technologies like Immediate Payment Service (IMPS), Unified Payments Interface (UPI), and block chain. This enhances the overall efficiency of the payment system.

7. Data-driven Insights: Digital transformation allows banks to leverage big data and analytics to gain insights into customer behavior, market trends, and risk management. These insights help banks make data-driven decisions and offer tailored financial products.

8. Open banking initiatives and APIs (Application Programming Interfaces) facilitate collaboration between banks and third-party fintech companies. This fosters innovation and the development of new financial services and products.

9. Remote Banking and Virtual Branches: Digital transformation enables remote banking services, reducing the need for physical branches. Virtual branches and remote customer support ensure seamless banking experiences for customers, regardless of their location.

10. Innovation and Agility: Digital transformation promotes a culture of innovation and agility within

banks. Embracing emerging technologies and adapting to market changes quickly allows banks to stay competitive and meet evolving customer demands.

11. Cost Optimization: By automating processes and reducing the reliance on physical infrastructure, banks can achieve cost optimization and operational efficiency.

1. b) Explain how design thinking has brought revolution in scanner experience that Children Loved. -10

Design thinking has brought a revolution in the scanner experience for children by putting their needs, preferences, and emotions at the center of the product design process. It has led to the creation of scanners that are not just functional but also engaging, enjoyable, and loved by children. This is how design thinking has influenced the scanner experience for children:

1.Empathizing with Children's Needs: Design thinking starts with empathizing with the end-users, in this case, children. Designers and developers spend time observing and talking to children to understand their preferences, interests, and pain points. By understanding children's unique perspective, design teams can identify specific features and elements that would appeal to them.

2. Child-Centric Design: Design thinking encourages child-centric design, where every aspect of the scanner is tailored to suit the cognitive abilities and emotional needs of children. The user interface is simplified, and the design incorporates vibrant colors, playful graphics, and interactive elements that resonate with children.

3. Gamification and Playfulness: Design thinking introduces elements of gamification and playfulness into the scanner experience. Scanning becomes an enjoyable activity through gamified feedback, visual rewards, and engaging animations. The goal is to make scanning an exciting and rewarding experience, akin to playing a game.

4. Storytelling and Imagination: Design thinking harnesses the power of storytelling and imagination. Scanners may incorporate characters or themes from children's favorite stories or media. By making the scanner experience immersive and storytelling-driven, children become more invested in the process.

5. Intuitive and Easy-to-Use: A key principle of design thinking is to create products that are intuitive and easy-to-use. For children, this means reducing complexity and making the scanning process straightforward and accessible. Large buttons, clear instructions, and minimal steps ensure children can interact with the scanner effortlessly.

6. Iterative Prototyping and Testing: Design thinking involves iterative prototyping and testing with real users, including children. This ensures that the scanner experience is continually refined based on direct feedback from the target audience. User testing helps identify any usability issues and allows designers to make improvements accordingly.

7.Inclusive Design: Design thinking promotes inclusive design, ensuring that the scanner experience is accessible to children with diverse abilities. For instance, designers may consider adding voice guidance for visually impaired children or making the scanner adaptable for children with motor skill challenges.

8. Emotional Connection: Design thinking emphasizes creating an emotional connection between the product and the user. Scanners that elicit positive emotions and joy during the scanning process foster a sense of attachment and fondness among children.

2. a) Discuss the different steps involved in organizing an enterprise. -10

Organizing an enterprise involves a series of steps that aim to structure the company's resource.

1. **Defining Goals and Objectives:** The first step in organizing an enterprise is to define its goals and objectives. This involves understanding the mission and vision of the organization, identifying key strategic objectives, and setting measurable targets for growth and success.

2. **Determining Organizational Structure:** Organizational structure refers to the way in which an enterprise is arranged and divided into various departments, divisions, and teams. Decisions about the structure are crucial as they define reporting lines, authority, and communication channels within the organization. Common types of organizational structures include functional, divisional, matrix, and flat structures.

3. **Job Design and Role Definition:** Job design involves creating and defining individual job roles within the organization. This includes determining job responsibilities, required skills and qualifications, reporting relationships, and the scope of authority for each position.

4. **Establishing Reporting Hierarchy:** Defining the reporting hierarchy is essential for clarifying decision-making authority and communication flow. It involves determining who reports to whom and establishing clear lines of authority from top-level management to lower-level employees.

5. **Delegation of Authority:** Delegation of authority is the process of distributing decision-making power and responsibilities throughout the organization. It ensures that each level of management has the necessary authority to carry out their roles effectively.

6. **Developing Policies and Procedures:** Organizations need to establish standard policies and procedures to guide employees in their day-to-day activities. These policies cover areas such as employee conduct, performance expectations, HR guidelines, financial management, and more.
7. **Forming Teams and Departments:** Based on the organizational structure, teams and departments are formed to carry out specific functions within the organization. Each team has a defined set of responsibilities and objectives aligned with the overall goals of the enterprise.
8. **Staffing and Recruitment:** Once the roles and departments are defined, the organization proceeds to hire and recruit employees. The recruitment process involves sourcing, interviewing, and selecting the best candidates who fit the required skills and qualifications.
9. **Training and Development:** To ensure that employees are equipped with the necessary skills and knowledge, training and development programs are conducted. This helps enhance their capabilities and productivity in their respective roles.
10. **Establishing Communication Channels:** Effective communication is essential for the smooth functioning of an organization. Organizations need to establish clear and efficient communication channels to facilitate the flow of information between employees, departments, and management.
11. **Implementing Technology and Systems:** In the modern business landscape, technology plays a crucial role in streamlining operations and improving efficiency. Organizations need to implement relevant technologies and systems to support their activities and processes.
12. **Performance Management and Feedback:** Regular performance evaluations and feedback sessions help monitor employee progress and identify areas for improvement. Constructive feedback motivates employees and promotes continuous growth.
13. **Continuous Evaluation and Improvement:** Organizing an enterprise is an ongoing process. Regular evaluation of the organizational structure, policies, and processes is necessary to identify areas that need improvement and to adapt to changing market dynamics.

2. b) Identify the following cloud service types and list their characteristics and advantage

Cisco WebEx, Google App Engine, Amazon EC2. -10Marks

Cisco WebEx -It is software-as-a-service (**SaaS**) type of cloud service.

Characteristics of Cisco WebEx

- High-Quality Video Conferencing
- Cross-Platform, Functional and Geographic Versatility
- File and Desktop Sharing

- Brainstorming via Whiteboarding
- AI Powered Functionality
- Security and privacy

Advantages Of Webex

- Easy Sharing Option
- Easy Invitation
- Difficult To Share Confidential Data
- HD Video And Audio
- Minimum Utilization Of Internet Data
- Accessible To Various Tools
- Sharing Task Is Easy
- Best Platform For Education Sector
- Stream Live Meetings On Social Media
- Reasonable Subscription Plans

Google App Engine- It is Platform as a Service **(PaaS)** type of cloud service.

Characteristics of Google App Engine

- Popular language.
- Open and flexible
- Fully managed
- Powerful application diagnostics
- Application versioning
- Application security
- Traffic splitting

Advantages Of Google App Engine

- Open and familiar languages and tools
- Just add code
- Pay only for what you use
- Easy to build and use the platform:
- Scalability
- Various API sets

Amazon EC2- It belongs to IaaS Cloud Computing Services

Characteristics of Amazon EC2

- Safe
- Dynamic Scalability
- Full Control of Instances
- Configuration Flexibility
- Integration with Other Amazon Web Services
- Optimal storage support
- Reliable and Resilient Performance Amazon Elastic Block Store (EBS)
- Support for Use in Geographically Disparate Locations
- Cost Effective
- Credible
- Flexibility of Tools
- Created for Amazon Web Services

Advantages of Amazon EC2

- Reliability
- Security
- Flexibility
- scalable
- Cost Savings
- Complete Computing Solution
- Elastic Web Computing
- Complete Controlled setup

Section-II

3. a) BookMyShow is an online movie ticket booking application that helps its user to book movie tickets by logging in. Users can find their movie from the listings. After booking is confirm the details are sent to user.. Identify and write the user stories for this application. -12

a) Registration

1. Sign-up:

As an unauthorized user, I want to sign up for the BookMyShow application through a sign-up form, so that I can access to movies list.

Acceptance Criteria:

1. While signing up-Use Name, Username, Email, and Password and Confirm Password.
2. If sign up is successful, it will get automatically logged in.
3. If I sign up with an incorrect detail which are specified in step1, I will receive an error Message for incorrect information.
4. If we are trying to sign up with an existing email address, we will receive an error Message saying "email exists."

2. Login

As an authorized user, I want to log in for BookMyShow application, so that I can have access to the application.

Acceptance Criteria:

1. While logging in, Username and password are required.
2. After successful log in, it will be redirected to the main page.
3. If we are trying to login with incorrect username or password, then error message will be displayed as "invalid login".

3. Searching a movie

As an authorized user, I want to search for a movie in BookMyShow application, so that I can book a movie ticket in a specific theater.

Acceptance Criteria:

1 .While searching, Valid theater should be specified.
2. Checking for availability of a movie ticket on specific date always should be current date and ahead of the current date.

4. Booking ticket

As an authorized user, I want to book a ticket in BookMyShow application, so that I can reserve the seat in a specific theater and date.

Acceptance Criteria:

1. While Booking, accommodation should be allotted according to the room size.
2. One should select the valid payment method based on the price of reserved room.
3. After successful payment one should get the booking details to registered mobile Number and E- mail id.

5. Logout

As an authorized user, I want to log out of application, so that I can prevent unauthorized access of my profile.

Acceptance Criteria: When I log out of my account, I will be redirected to the log-in page**.**

3. b) Write test cases for the above application. -8

Test Cases for the Login Page:

- Verify that the login page loads correctly and is accessible from the website's homepage.
- Check that the login credentials are case sensitive and the appropriate message is displayed if the user enters incorrect information.

- Verify that the "Forgot Password" option works as intended, allowing users to reset their password in case they forget it.
- Ensure that the system limits the number of unsuccessful login attempts to prevent brute-force attacks.

Test Cases for the Registration Page:

- Verify that the registration page is accessible from the website's homepage and loads correctly.
- Check that the system validates the user's information, such as email address, phone number, and password complexity.
- Ensure that the system does not allow duplicate email addresses or phone numbers.
- Verify that the user receives an email or SMS confirmation after registering.

Test Cases for the Ticket Booking Page:

- Ensure that the ticket booking page displays accurate information about the event, such as the date, time, and venue.
- Check that the system limits the number of tickets a user can purchase to prevent scalping.
- Verify that the system displays the total cost of the ticket purchase, including any taxes and fees.
- Ensure that the system accepts multiple payment options, such as credit/debit cards, PayPal, and mobile wallets.
- Test Cases for the Payment Gateway:
- Verify that the payment gateway is secure and encrypts user information to prevent fraud.
- Check that the payment gateway accepts different currencies and displays the correct conversion rates.
- Ensure that the payment gateway sends a confirmation email or SMS to the user after the transaction is complete.

By following these test cases, you can ensure that your online ticket booking system is reliable and user-friendly. Thorough testing will help you identify and fix any issues before your system goes live, ensuring a positive experience for your customers.

4. a) Swiggy is an online food ordering application that helps its users to buy variety of authentic food items. This application allows users to log in for ordering food. Users can search for their favorite food based on rating or price. Users can select the items and add to the cart. Once the selection made go to payment page and make payment. write the user stories for this application. -12

Registration

Sign-up:

As a foodie, I want to sign up for Swiggy application through a New user form, so that I can get access to order food of my favorite.

Acceptance Criteria:

- While signing up-Valid Phone Number/Email Id and OTP/Password.
- If sign up is successful, it will get automatically logged in.
- If I am trying to sign up with an invalid phone number/Email Id, I will receive an error message to enter a valid information.
- If we are trying to sign up with an existing phone number/Email Id, we will receive an error message saying "you are already registered."

Login

As an authorized customer, I want to login for application, so that I can have access to the application

for searching and ordering food.

Acceptance Criteria:

- While logging in, Phone number/Email Id and OTP/Password are required.
- After successful log in, it will be redirected to the main page.
- If we are trying to login with incorrect mobile number/Email Id or OTP/Password, then error message will be displayed as "invalid credentials".

Order Creation

As a customer, I should be able to browse through the menu and look at various food options and restaurants and along with their price.
As a customer, I should be able to select items from the menu and add them to cart.
As a customer, I should have cart containing all the choosen items .
As a customer, I should be able to remove items from my cart or increase item count.
As a customer, I should be able to cancel my entire order.
As a customer, I should be able to view the items bill for my oder along with price of each item.
As a customer, I should be able to see the listing of restaurants selling food items.

Acceptance Criteria:

- Categorized menu with prices is visible and enabled with selection choices as soon as the customer chooses items, the order is created in the database and is visible to the customer.
- See a thumbnail image for each product
- Click to view details for product
- Add to cart from detail page
- Search for a item
- View food item by category

Order completion

As a customer, I should be able to provide feedback for service and the food.

Acceptance Criteria

- All the feedbacks are recorded in database for further improvement.

Logout

As a customer, I want to log out of application, so that I can prevent unauthorized access of my profile.

Acceptance Criteria:

- When I log out of my account, I will be redirected to the log-in page.

4. b) Write test cases for the above application.

Test Cases for Swiggy application Login Page

- Verify that when user open online food ordering application then it should be asked for the user's location.
- Verify that user is able to login in the application without registration or not.
- Verify that user is able to sign up or login with mobile number or not.
- Verify that user is able to sign up or login with email address or not.
- Verify that user is able to redirect on home page screen without login or not.
- Verify that logo of the online food ordering application on the login screen.
- Verify that application name is displayed on the login page or not.
- Verify that user is able to login with invalid credentials or not.
- Verify that user is able to skip login screen or not.
- Verify that links on the login page should be working properly or not.

Test Cases For Online Food Ordering System Search Functionality

- Verify that if user enters valid food name, then search result should be displayed.
- Verify that if user enters valid restaurant name, then search result should be displayed.

- Verify that if user search by valid food name, then relevant food search result should be displayed on the screen.
- Verify that if user search by valid restaurant name, then relevant food search result should be displayed on the screen.

Test Cases For Ordering Page

- Verify that the restaurant name with rating should be displayed clearly.
- Verify that list of the cuisines should be displayed under the restaurant names.
- Verify that user should be able to see veg and non veg category on the ordering page or not.
- Verify that user is able to see billing discount on the order page or not.
- Verify that user is able to see total numbers of reviews on the ordering page or not.
- Verify that approximately time of the delivery food is displayed as per expected or not.
- Verify that user is able to add food item into the cart or not.
- Verify that add on food items option is displayed on the page or not.
- Verify that user is able to see items with its price or not.

Test Cases For Cart Checkout Page

- Verify that user should be able to see added items into the cart.
- Verify that user is able to increase the quantity of the food items from the cart page or not.
- Verify that user should be able to delete food items from the cart page.
- Verify that food price is displayed for the food items or not.
- Verify that user is able to edit delivery address or not.
- Verify that user is able to change delivery address or not.
- Verify that user is able to select payment method on cart checkout page.
- Verify that user is able to place order from the cart checkout page or not.

SECTION-III

5. a) Create a spring Boot application to maintain employee details such as employee id, employee name, and department. Design entity or data JPA class, controller class and repository. -12

Create an Entity class for the Employee, a Controller class to handle HTTP requests, and a Repository to interact with the database.

Entity Class (Employee.java):

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String employeeName;
    private String department;

    // Constructors, getters, and setters

    // Constructor without id for creating new employees
    public Employee(String employeeName, String department) {
        this.employeeName = employeeName;
        this.department = department;
    }
```

```
    // Default constructor for JPA
    public Employee() {
    }

    // Getters and setters
}
```

Repository Interface (EmployeeRepository.java):

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

Controller Class (EmployeeController.java):

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/employees")
public class EmployeeController {

    private final EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeController(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    // Endpoint to get all employees
    @GetMapping
    public ResponseEntity<List<Employee>> getAllEmployees() {
        List<Employee> employees = employeeRepository.findAll();
        return new ResponseEntity<>(employees, HttpStatus.OK);
    }

    // Endpoint to get an employee by id
    @GetMapping("/{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
        Employee employee = employeeRepository.findById(id).orElse(null);
        if (employee == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(employee, HttpStatus.OK);
    }

    // Endpoint to create a new employee
    @PostMapping
```

```
    public ResponseEntity<Employee> createEmployee(@RequestBody Employee employee) {
        Employee createdEmployee = employeeRepository.save(employee);
        return new ResponseEntity<>(createdEmployee, HttpStatus.CREATED);
    }

    // Endpoint to update an existing employee
    @PutMapping("/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody
Employee employee) {
        Employee existingEmployee = employeeRepository.findById(id).orElse(null);
        if (existingEmployee == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        existingEmployee.setEmployeeName(employee.getEmployeeName());
        existingEmployee.setDepartment(employee.getDepartment());
        employeeRepository.save(existingEmployee);
        return new ResponseEntity<>(existingEmployee, HttpStatus.OK);
    }

    // Endpoint to delete an employee by id
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {
        Employee employee = employeeRepository.findById(id).orElse(null);
        if (employee == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        employeeRepository.delete(employee);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
```

Spring Boot Application (EmployeeManagementApplication.java):

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmployeeManagementApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmployeeManagementApplication.class, args);
    }
}
```

5. b) Compare spring and spring boot. -10

Spring	**Spring Boot**
Spring Framework is a widely used Java EE framework for building applications.	Spring Boot Framework is widely used to develop REST APIs.
It simplify Java EE development that makes developers more productive.	It shorten the code length and provide the easiest way to develop Web Applications.

The primary feature of the Spring Frameworkis dependency injection.	The primary feature of Spring Boot is Autoconfiguration. It automatically configures the classes based on the requirement.
It helps to develop loosely coupled applications.	It helps to create a stand-alone application with less configuration.
The developer writes a lot of code (boilerplate code) to do the minimal task.	It reduces boilerplate code.
To test the Spring project, we need to set up the sever explicitly.	Spring Boot offers embedded server such as Jetty and Tomcat, etc.
It does not provide support for an in-memory database.	It offers several plugins for working with an embedded and in-memory database such as H2.
Developers manually define dependencies for the Spring project in pom.xml.	Spring Boot use the concept of starter in pom.xml file that internally takes care of downloading the dependencies JARs based on Spring Boot Requirement.

6. a) Create a login form using React JS . -10

```
import React, { useState } from "react";
import ReactDOM from "react-dom";

function App() {
  // React States
  const [errorMessages, setErrorMessages] = useState({});
  const [isSubmitted, setIsSubmitted] = useState(false);

  // User Login info
  const database = [
    {
      username: "user1",
      password: "pass1"
    },
    {
      username: "user2",
      password: "pass2"
    }
  ];

  const errors = {
    uname: "invalid username",
    pass: "invalid password"
  };

  const handleSubmit = (event) => {
    //Prevent page reload
    event.preventDefault();
```

```
    var { uname, pass } = document.forms[0];

    // Find user login info
    const userData = database.find((user) => user.username === uname.value);

    // Compare user info
    if (userData) {
      if (userData.password !== pass.value) {
        // Invalid password
        setErrorMessages({ name: "pass", message: errors.pass });
      } else {
        setIsSubmitted(true);
      }
    } else {
      // Username not found
      setErrorMessages({ name: "uname", message: errors.uname });
    }
  };

  // Generate JSX code for error message
  const renderErrorMessage = (name) =>
    name === errorMessages.name && (
      <div className="error">{errorMessages.message}</div>
    );

  // JSX code for login form
  const renderForm = (
    <div className="form">
      <form onSubmit={handleSubmit}>
        <div className="input-container">
          <label>Username </label>
          <input type="text" name="uname" required />
          {renderErrorMessage("uname")}
        </div>
        <div className="input-container">
          <label>Password </label>
          <input type="password" name="pass" required />
          {renderErrorMessage("pass")}
        </div>
        <div className="button-container">
          <input type="submit" />
        </div>
      </form>
    </div>
  );

  return (
    <div className="app">
      <div className="login-form">
        <div className="title">Sign In</div>
        {isSubmitted ? <div>User is successfully logged in</div> : renderForm}
      </div>
    </div>
  );
}
```

export default App;

6. b) Compare the database through JDBC and ORM. -10

ORM when compared to JDBC is easier to work with as it does all the work by itself. It maps Java classes to the database variables via XML. While working with domain-driven applications and in the case of complex object relationships, ORM is mostly preferred but when the application is simple enough then it is better to use JDBC.

Object Relational Mapping	**Java Database Connectivity**
Little slower than JDBC	It is faster compared to ORM
ORM frameworks handle database management automatically.	JDBC requires manual management of database connections and SQL statements
ORM frameworks handles data coversion automatically.	JDBC requires manual handling of data conversion between the database
ORM frameworks provide support for lazy loading and caching of data.	JDBC doesn't
ORM frameworks provide support for managing the object-relational mapping and relations between tables.	JDBC doesn't
ORM frameworks provide a more object-oriented and intuitive way to interact with the database.	while JDBC is more focused on SQL and relational databases
SQL queries requirement is comparatively quite less used in ORM	SQL queries are required here
The flow from Object/data to hibernate i.e. the frontend part is based on the ORM technique	Whereas when the data is stored in the database finally i.e., the backend part is still based on JDBCin
There are not many restrictions while dealing with data. Even a single database cell can be retrieved, changed, and saved.	JDBC comes with a lot of restrictions on extracting the result-set, processing it, and then committing it back to the database.

SECTION-IV

7. a) Discuss how REST API is best over SOAP API illustrate. -10

There is no direct comparison between SOAP and REST APIs. But there are some points to be listed below which makes you choose better between these two web services.

Differentiating between SOAP API and REST API

SOAP API	**REST API**

SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol	**REST** stands for **R**epresentational **S**tate **T**ransfer.
SOAP is a protocol, it follows a strict standard to allow communication between the client and the server	REST is an architectural style that doesn't follow any strict standard
SOAP uses only XML for exchanging information in its message format.	REST is not restricted to XML and its the choice to use like XML, JSON, Plain-text. Moreover, REST can use SOAP protocol but SOAP cannot use REST.
For services interfaces to business logic, SOAP uses **@WebService**	For services interfaces to business logic, REST uses **@path**
SOAP is difficult to implement and it requires more bandwidth.	REST is easy to implement and requires less bandwidth such as smart phones.
SOAP has ACID compliance transaction.	REST lacks ACID compliance transaction.
SOAP has SSL(**S**ecure **S**ocket **L**ayer) and WS-security	REST has SSL and HTTPS.
Because it is XML based and relies on SOAP, it works with WSDL	It works with GET, POST, PUT, DELETE
Highly structured/typed	Less structured -> less bulky data
Designed with large enterprise applications in mind	Designed with mobile devices in mind
SOAP cannot make use of REST since SOAP is a protocol without any architectural pattern. SOAP cannot make use of REST	REST can make use of SOAP because it is an architectural pattern having protocol.

7. b) How to convert monolithic application to microservice architecture? Explain with an example. - 10

A typical process to migrate from a monolithic system to a microservices based system involves the following steps:

1. Identify logical components.
2. Flatten and refactor components.
3. Identify component dependencies.
4. Identify component groups.
5. Create an API for remote user interface.

6. Migrate component groups to macroservices (move component groups to separate projects and make separate deployments).
7. Migrate macroservices to microservices.
8. Repeat steps 6-7 until complete.

1. Identify Logical Components

There are three main information components with the data used in the system:

- d
ata objects
- d
ata actions
- j
ob to perform and use cases

The data objects are the logical constructs representing the data being used. The data actions are the commands that are used on one or more data objects, possibly on different types of data, to perform a task. The job to perform represents the function the users are calling to fulfill their organizational roles. The jobs to perform may be captured as use cases, user stories, or other documentation involving user input.

Example: Movie application

Monolith Architecture

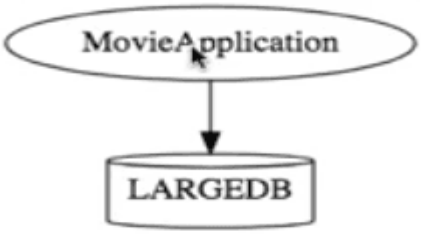
The salient features of monolith applications are:

- Released, or taken to production, once every few week or months or years
- Generally have a wide range of features and functionality
- Have a development team of over 50 people working on them
- Debugging problems that arise in them, is a huge challenge

It is almost impossible to bring in new technologies and technical processes, midway through the lifetime of such an application.

Monolith applications are typically huge, with them having a million lines of code on average.

A monolithic application looks as follows:

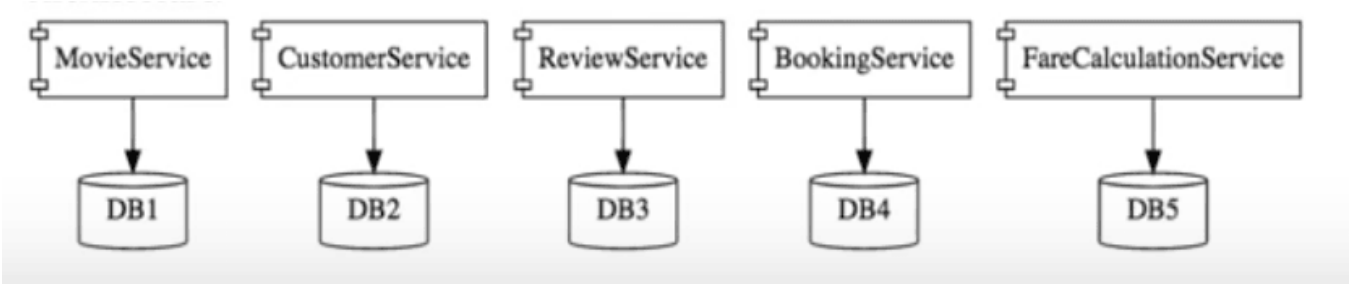


We have a large application talking to a large database.

Microservices Architecture

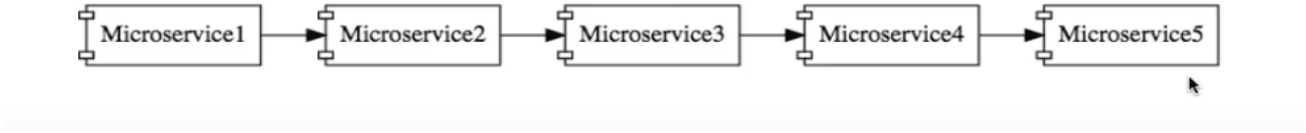
In microservices architectures, instead of building a large application, we build a number of smaller microservices.

This is how we would split up the monolith MovieApplication into microservice application



As we can see the databases are also separated out.

Microservices architecture involves a number of small, well-designed microservices, that exchange messages among themselves.



8 a) Illustrate ACID transaction in MongoDB - 10

ACID is an acronym that stands for atomicity, consistency, isolation, and durability.

Atomicity

Atomicity guarantees that all of the commands that make up a transaction are treated as a single unit and either succeed or fail together. This is important as in the case of an unwanted event, like a crash or power outage, we can be sure of the state of the database. The transaction would have either completed successfully or been rollbacked if any part of the transaction failed.

If we continue with the above example, money is deducted from the source and if any anomaly occurs, the changes are discarded and the transaction fails.

Consistency

Consistency guarantees that changes made within a transaction are consistent with database constraints. This includes all rules, constraints, and triggers. If the data gets into an illegal state, the whole transaction fails.

Going back to the money transfer example, let's say there is a constraint that the balance should be a positive integer. If we try to overdraw money, then the balance won't meet the constraint. Because of that, the consistency of the ACID transaction will be violated and the transaction will fail.

Isolation

Isolation ensures that all transactions run in an isolated environment. That enables running transactions concurrently because transactions don't interfere with each other.

For example, let's say that our account balance is $200. Two transactions for a $100 withdrawal start at the same time. The transactions run in isolation which guarantees that when they both complete, we'll have a balance of $0 instead of $100.

Durability

Durability guarantees that once the transaction completes and changes are written to the database, they are persisted. This ensures that data within the system will persist even in the case of system failures like crashes or power outages.

8. b) Create a REST controller class to perform CRUD operations on product and corresponding request and response DTOs. The product class should contain three data members product name, product category, price. Use proper SpringBoot annotations. -10

A REST controller class using Spring Boot to perform CRUD operations on a Product entity. I'll also provide the corresponding request and response DTOs (Data Transfer Objects) to handle the data sent to and received from the API.

Product entity

Product.java

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String productName;
    private String productCategory;
    private double price;
    // Constructors, getters, and setters
}
```

ProductRequestDTO:

```
// ProductRequestDTO.java

public class ProductRequestDTO {

    private String productName;
    private String productCategory;
    private double price;
    // Constructors, getters, and setters
}
```

ProductResponseDTO:

```
// ProductResponseDTO.java
public class ProductResponseDTO {
    private Long id;
    private String productName;
    private String productCategory;
    private double price;
    // Constructors, getters, and setters
}
```

ProductController:

```
// ProductController.java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/products")
public class ProductController {
    private final ProductService productService;
    @Autowired
    public ProductController(ProductService productService) {
        this.productService = productService;
    }
```

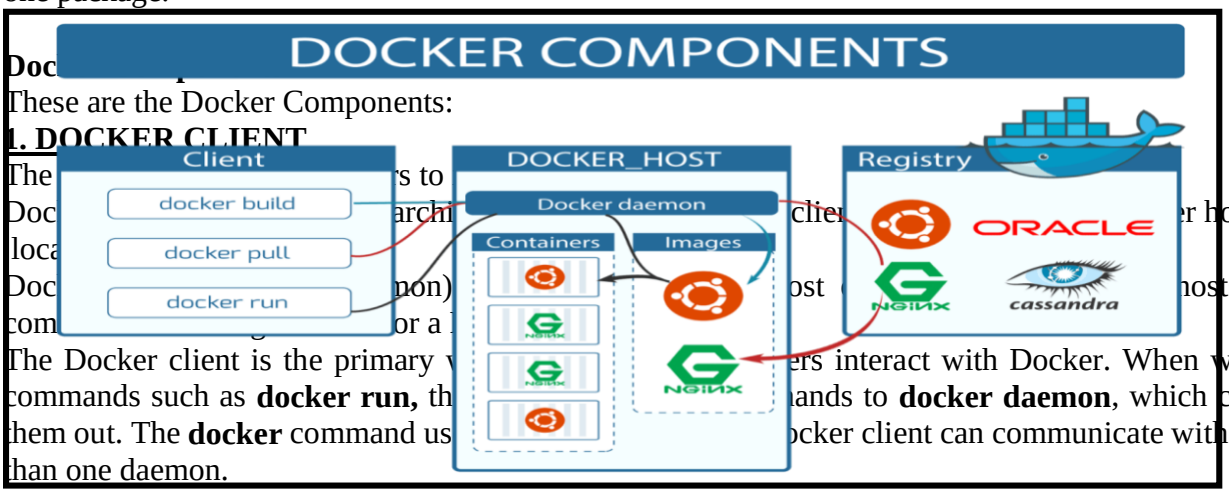
```
    // Endpoint to create a new product
    @PostMapping
    public ResponseEntity<ProductResponseDTO>
      createProduct(@RequestBody ProductRequestDTO requestDTO) {
        Product createdProduct = productService.createProduct(requestDTO);
        ProductResponseDTO responseDTO = convertToResponseDTO(createdProduct);
        return new ResponseEntity<>(responseDTO, HttpStatus.CREATED);
    }

    // Endpoint to get a product by its ID
    @GetMapping("/{id}")
    public ResponseEntity<ProductResponseDTO> getProduct(@PathVariable Long id)
{
        Product product = productService.getProductById(id);
        if (product == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        ProductResponseDTO responseDTO = convertToResponseDTO(product);
        return new ResponseEntity<>(responseDTO, HttpStatus.OK);
    }

    // Endpoint to get all products
    @GetMapping
    public ResponseEntity<List<ProductResponseDTO>> getAllProducts() {
        List<Product> products = productService.getAllProducts();
        List<ProductResponseDTO> responseDTOs = products.stream()
                .map(this::convertToResponseDTO)
                .collect(Collectors.toList());
        return new ResponseEntity<>(responseDTOs, HttpStatus.OK);
    }

    // Endpoint to update a product
    @PutMapping("/{id}")
    public ResponseEntity<ProductResponseDTO> updateProduct(@PathVariable Long id,
                                            @RequestBody ProductRequestDTO requestDTO) {
        Product updatedProduct = productService.updateProduct(id, requestDTO);
        if (updatedProduct == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        ProductResponseDTO responseDTO = convertToResponseDTO(updatedProduct);
        return new ResponseEntity<>(responseDTO, HttpStatus.OK);
    }

    // Endpoint to delete a product
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
        boolean deleted = productService.deleteProduct(id);
        if (!deleted) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);  }
        return new ResponseEntity<>(HttpStatus.NO_CONTENT); }

    // Helper method to convert Product entity to ProductResponseDTO
    private ProductResponseDTO convertToResponseDTO(Product product) {
        ProductResponseDTO responseDTO = new ProductResponseDTO();
        responseDTO.setId(product.getId());
```

```
responseDTO.setProductName(product.getProductName());
responseDTO.setProductCategory(product.getProductCategory());
responseDTO.setPrice(product.getPrice());
return responseDTO; } }
```

SECTION-V

9a)Discuss the Components of Docker Container. -10

Docker is an open-source software platform. It is designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts which are required, such as libraries and other dependencies and ship it all out as one package.



Doc

These are the Docker Components:

1. DOCKER CLIENT

The ...s to Doc...archi...clie...r host loca... Doc...mon)...ost ...hosts and com...or a ... The Docker client is the primary ...ers interact with Docker. When we use commands such as **docker run,** th...ands to **docker daemon**, which carries them out. The **docker** command us...ocker client can communicate with more than one daemon.

We can communicate with the docker client using the Docker CLI. We have some commands through which we can communicate the Docker client. Then the docker client passes those commands to the Docker daemon.

docker build ... docker run ... docker push ..etc.

2. DOCKER HOST

The Docker host provides a complete environment to execute and run applications. It includesDocker daemon, Images, Containers, Networks, and Storage.

a.Docker Daemon

Docker Daemon is a persistent background process that manages Docker images, containers, networks, and storage volumes. The Docker daemon constantly listens for Docker API requests and processes them.

b.Docker Images:

Docker-images are a read-only binary template used to build containers. Images also contain metadata that describe the container's capabilities and needs.

- Create a docker image using the docker build command.
- Run the docker images using the docker run command.
- Push the docker image to the public registry like DockerHub using the **docker push** command after pushed we can access these images from anywhere using docker pull command.
- An image can be used to build a container. Container images can be shared across teams within an enterprise using a private container registry, or shared with the world using a public registrylike Docker Hub.

c) Docker Containers:

A container is a runnable instance of an image. We can create, start, stop, move, or delete a container using the Docker API or CLI. We can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

d) Docker Networking

Through the docker networking, we can communicate from one container to other containers.

By default, we get three different networks on the installation of Docker – none, bridge, and host. The none and host networks are part of the network stack in Docker. The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing.

e) Docker Storage

A container is volatile it means whenever we remove or kill the container then all of its datawill be lost from it. If we want to persist the container data use the docker storage concept.

We can store data within the writable layer of a container but it requires a storage driver. Interms of persistent storage, Docker offers the following options:

- Data Volumes
- Data-Volume Container
- Bind Mounts

3. DOCKER REGISTRIES

Docker-registries are services that provide locations from where we can store and download images. A Docker registry contains repositories that host one or more Docker Images.Public Registries include Docker Hub and Docker Cloud and private Registries can also be used. We can also create our own private registry. Push or pull image from docker registry using the following commands docker push docker pulldocker run

9. b) Discuss the difference in manual deployment and automated deployment - 10

Manual deployment and automated deployment are two approaches used in software development to release and manage applications. They differ significantly in terms of the processes involved, the level of human intervention, and the benefits they offer.

The key differences between manual deployment and automated deployment are :

Characteristics	Manual Deployment	Automated Deployment
Process and Human Intervention	Manual Deployment approach, deployment tasks are performed manually by human operators. This involves activities like copying files, configuring settings, and executing scripts step by step. Each deployment action requires explicit human input and decision-making.	Automated deployment relies on software tools, scripts, and systems to carry out the deployment process. Developers and operations teams set up deployment pipelines that automatically execute predefined steps, reducing the need for direct human intervention.
Speed and Efficiency	Manual Deployment: Because manual deployment involves human operators performing tasks, it can be slow and prone to errors. The time taken to deploy applications may vary based on individual skill levels and the complexity of the application.	Automated Deployment: Automated deployment is generally much faster and more efficient. Once the deployment pipeline is set up, it can be executed repeatedly without human intervention. This consistency ensures that the deployment process is reliable and predictable.
Consistency and Reproducibility	Human-based deployment can lead to inconsistencies between different environments (e.g., development, testing, production). If the same steps are not followed accurately, issues may arise during deployment.	Automated deployment ensures consistency and reproducibility. The same deployment process is applied across all environments, reducing the risk of configuration drift and errors.
Risk and Error Minimization	Human error is a significant risk in manual deployment. Typos, mis-configurations, or missed steps can lead to deployment failures and downtime.	Automated systems can help minimize human errors as the deployment process is predefined and thoroughly tested. Automated rollback mechanisms can also be implemented to revert to a stable state in case of deployment issues.

Scalability and Complexity	Manual deployment becomes challenging and time-consuming as the application and infrastructure scale in size and complexity.	Automation can handle more complex deployments and scale easily as it is designed to handle a wide range of scenarios. It is well-suited for modern microservices-based architectures and cloud-native applications.
Continuous Deployment and Continuous Integration (CI/CD)	Implementing continuous deployment and continuous integration without automation is difficult and may not be practical.	Automated deployment is a key enabler for CI/CD workflows, where changes are automatically tested, integrated, and deployed to production, often multiple times a day.

10 a) Create a form to add a new product detail to the product catalogue using React. -10

React component code for Productform.js

```
import React, { useState } from 'react';
const ProductForm = () => {
  const [productName, setProductName] = useState('');
  const [description, setDescription] = useState('');
  const [price, setPrice] = useState('');
  const handleSubmit = (event) => {
    event.preventDefault();
    // Assuming you'll perform validation here before proceeding
    // For simplicity, we'll just log the product details to the console.
    console.log('Product Name:', productName);
    console.log('Description:', description);
    console.log('Price:', price);
    // Add code here to submit the data to the server or state management system.
    // You can also reset the form fields after submission if needed.
  };
  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="productName">Product Name:</label>
        <input
          type="text"
          id="productName"
          value={productName}
          onChange={(e) => setProductName(e.target.value)}
          required
        />
      </div>
      <div>
        <label htmlFor="description">Description:</label>
        <textarea
          id="description"
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          required
        />
      </div>
      <div>
        <label htmlFor="price">Price:</label>
        <input
```

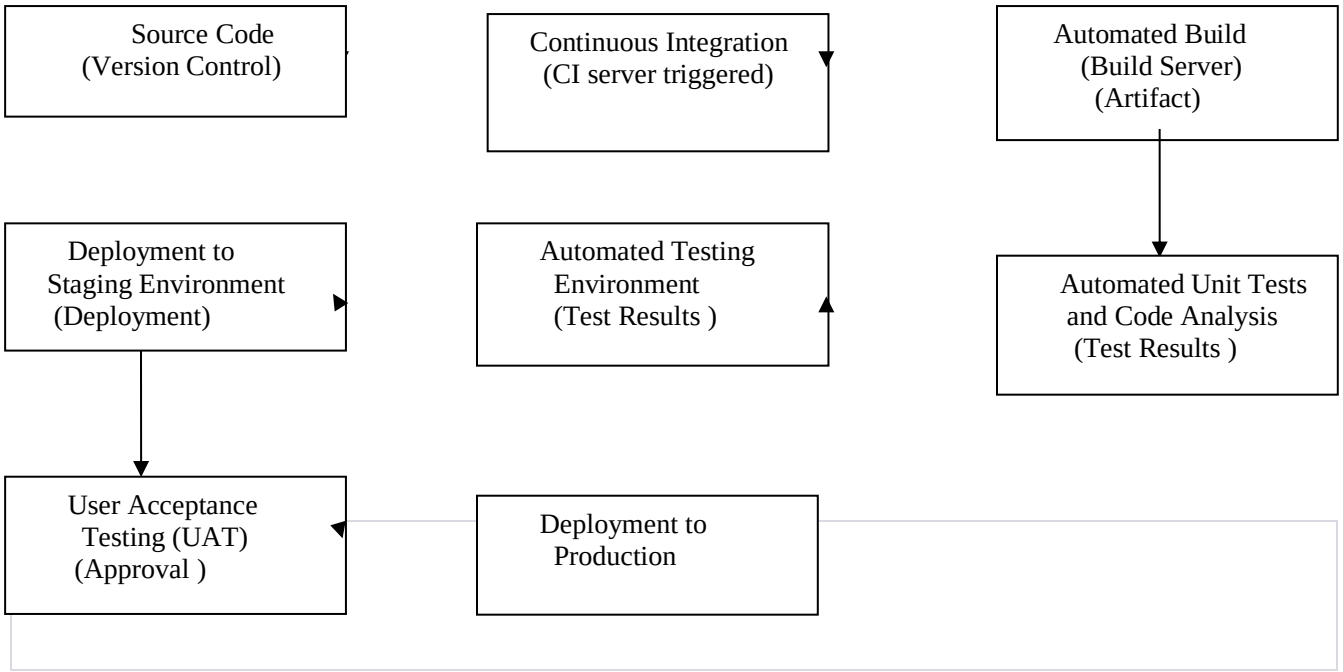
```
        type="number"
        id="price"
        value={price}
        onChange={(e) => setPrice(e.target.value)}
        required
       />
     </div>
     <button type="submit">Add Product</button>
   </form>
  );
};
export default Product Form;
```

10. b) Draw the CI/CD build process flow diagram for an online application and explain each component. -10

The CI/CD build process flow diagram for an online application typically involves multiple stages and components to automate the building, testing, and deployment of the application. Below is a simplified representation of the CI/CD process flow.

1. Source Code (Version Control): This is the central repository where developers store and manage the application's source code. Popular version control systems include Git, SVN, etc. Developers push code changes to this repository.

2. CI Server (Continuous Integration): The CI server monitors the version control system for code changes. Whenever a new commit is pushed or a pull request is submitted, the CI server is triggered. Its primary purpose is to automate the integration of code changes into a shared repository and perform various automated tasks.



3. Automated Build (Build Server): Upon triggering, the CI server initiates an automated build process. It compiles the source code, gathers dependencies, and generates a build artifact (e.g., executable, binary, or container image). This artifact represents the built application.

4. Automated Unit Tests and Code Analysis: After the build, the CI server runs automated unit tests to check the functionality and correctness of the application. Additionally, it may perform static code analysis to identify potential issues, bugs, or code style violations.

5. Automated Testing Environment: This is an isolated environment where the application is deployed for automated testing. It simulates the production environment but may have fewer resources. Automated integration tests, regression tests, and other tests are conducted here.

6. Deployment to Staging Environment: If all the previous stages (build and automated tests) are successful, the application is deployed to a staging environment. The staging environment is a near-

production replica where final testing is conducted before going live.
7. User Acceptance Testing (UAT): In this phase, the application is tested by actual users (typically non-technical stakeholders) to ensure it meets business requirements and user expectations.
Deployment to Production: Upon successful UAT and approval, the application is deployed to the production environment, making it available to end-users.

.

CERTIFICATE

Certified that, as per the guidelines the question paper and the model answers are prepared and typed by me for the course **Full Stack Development-20CS52I** and they are found correct according to my knowledge.

THARA B S
Lecturer in CS& E,Gpt, Tumakuru