1.Describe full stack development with industrial perspective. identify key trends in FSD

Full Stack Development (FSD) from an industrial perspective involves the ability to work across the entire technology stack of an application. This means handling both front-end (client-side) and back-end (server-side) development. Full stack developers are expected to have expertise in:

- **Front-end** technologies such as HTML, CSS, JavaScript, and frameworks like React.js, Angular, or Vue.js.
- **Back-end** technologies such as Node.js, Python, Ruby, Java, or PHP, along with databases like MySQL, MongoDB, PostgreSQL, etc.
- **Version control**, deployment, security practices, and DevOps tools.

Industrial Expectations from Full Stack Developers:

1. **End-to-End Development**: Full stack developers in the industry are expected to design, develop, and maintain all parts of an application, from user interfaces to databases and servers, offering complete product solutions.
2. **Agile and DevOps Practices**: Developers are often part of agile teams, where they contribute to rapid iterations, continuous integration, and continuous deployment (CI/CD) pipelines. Full stack developers often work closely with cross-functional teams, focusing on seamless collaboration between development, operations, and testing.
3. **Problem-Solving and Adaptability**: Full stack developers are valued for their ability to tackle diverse problems in different parts of the stack, quickly adapting to new challenges.
4. **Security Awareness**: Given the growing number of security threats, full stack developers must be aware of both front-end vulnerabilities (like XSS and CSRF) and back-end issues (like SQL injections, secure APIs, and data protection).
5. **Scalability and Performance Optimization**: As companies grow, they expect full stack developers to build scalable solutions and optimize application performance across various layers.

Key Trends in Full Stack Development:

1. **JavaScript Dominance**: With the rise of frameworks like React.js, Vue.js, and Node.js, JavaScript has solidified its role as the go-to language for both front-end and back-end development. Full stack JavaScript (MERN or MEAN stack) continues to be highly popular.
2. **Serverless Architectures**: Companies are increasingly adopting serverless computing (e.g., AWS Lambda, Azure Functions), allowing full stack developers to focus on application logic rather than server management. This trend aligns with a need for more scalable and cost-effective solutions.
3. **Microservices and API-First Approach**: Monolithic applications are being replaced by microservices and API-first development. Full stack developers must be adept at creating decoupled, independently deployable services, with RESTful or GraphQL APIs.
4. **DevOps Integration**: Full stack developers are increasingly expected to handle aspects of infrastructure management, CI/CD pipelines, and containerization (e.g., Docker, Kubernetes). This trend is driving the convergence of development and operations roles.
5. **Low-Code and No-Code Platforms**: While still emerging, low-code/no-code platforms are enabling quicker prototyping and development. Full stack developers might be involved in customizing and integrating these solutions, but it also suggests that simpler tasks are being automated.
6. **AI and Machine Learning Integration**: Full stack developers are increasingly required to integrate AI and ML into their applications. This involves embedding AI-driven features like chatbots, recommendation engines, and analytics into the product.
7. **Focus on Web Performance**: With more applications being used on mobile devices, there's a growing focus on optimizing web performance through techniques like lazy loading, PWA (Progressive Web Apps), and server-side rendering (SSR).

8. **Cloud-Native Development**: Full stack development is shifting to cloud-native approaches, where developers need to work with cloud platforms (e.g., AWS, Google Cloud, Azure) for hosting, scaling, and managing applications.
9. **Real-Time Web Applications**: The demand for real-time, interactive applications (e.g., messaging apps, live streaming) is growing. Full stack developers increasingly rely on WebSockets, event-driven architectures, and tools like Firebase or Socket.IO to support these features.
10. **Blockchain and Decentralized Apps (DApps)**: Full stack developers are exploring blockchain technologies and decentralized applications as industries look for more secure, transparent, and distributed solutions.

Conclusion:

In today's industrial setting, full stack developers are expected to be versatile problem solvers who can work on both front-end and back-end technologies, collaborate in agile and DevOps teams, and keep pace with emerging trends like serverless computing, microservices, and cloud-native development.

2.Explain in details in beloe:

1.Bootstrap 2.Node.js 3.Django 4.Angular.js 5.React.js 6.Docker

1. Bootstrap:

Definition: Bootstrap is a free, open-source front-end framework for creating responsive and mobile-first websites. It includes design templates based on HTML, CSS, and JavaScript to speed up the development of web interfaces.

Key Features:

- **Responsive Grid System**: Bootstrap uses a flexible, responsive grid system that adjusts layouts according to the screen size.
- **Pre-built Components**: Components like navigation bars, buttons, modals, and carousels are ready to be used out of the box.
- **Customizable**: Bootstrap allows developers to customize styles easily using Sass or CSS to match their branding or design requirements.
- **JavaScript Plugins**: JavaScript/jQuery plugins are provided for enhancing interactivity, such as modals, carousels, and tooltips.
- **Cross-Browser Compatibility**: Bootstrap ensures that the UI works well across different browsers and platforms.

Use Cases:

Bootstrap is widely used for building responsive websites, especially when rapid prototyping is required. It's suitable for both small websites and large-scale web applications.

---

2. Node.js:

Definition: Node.js is a JavaScript runtime built on Chrome's V8 engine, designed for building scalable network applications. It allows developers to use JavaScript on the server-side to build full-stack applications.

Key Features:

- **Non-blocking I/O**: Its event-driven architecture and non-blocking I/O make Node.js highly efficient and scalable.
- **Single-threaded but Asynchronous**: Node.js operates on a single thread but handles concurrent connections efficiently using asynchronous programming techniques.
- **NPM (Node Package Manager)**: NPM is the world's largest package repository, with thousands of modules and libraries to extend the functionality of Node.js.
- **Microservices and APIs**: Node.js is widely used to build microservices and APIs due to its lightweight and fast nature.
- **Cross-Platform**: It supports multiple platforms, such as Windows, Linux, and macOS.

Use Cases:

Node.js is ideal for developing real-time applications (e.g., chat apps), API services, and scalable web servers, such as those used by companies like Netflix and LinkedIn.

---

3. Django:

Definition: Django is a high-level Python-based web framework that promotes rapid development and clean, pragmatic design. It follows the Model-View-Template (MVT) architectural pattern.

Key Features:

- **Batteries-Included**: Django comes with pre-built features like authentication, URL routing, ORM, and more, reducing the need for third-party packages.
- **ORM (Object-Relational Mapper)**: It allows developers to interact with databases using Python objects, eliminating the need to write SQL queries manually.
- **Security**: Django has built-in protections against common web vulnerabilities such as SQL injection, XSS, and CSRF.
- **Admin Panel**: An automatically generated admin interface is included for managing the application's data.
- **Scalable**: Django's modular structure makes it suitable for scaling large web applications.

Use Cases:

Django is perfect for building complex, data-driven web applications, such as e-commerce platforms, social networks, or content management systems.

---

4. AngularJS:

Definition: AngularJS is a JavaScript-based open-source front-end web framework, primarily used for developing dynamic, single-page applications (SPAs). It was developed by Google and follows the Model-View-Controller (MVC) architecture.

Key Features:

- **Two-Way Data Binding**: Changes in the model are reflected in the view and vice versa, enabling dynamic updates in the UI without manually manipulating the DOM.

- **Directives**: AngularJS extends HTML with custom tags and attributes called directives, allowing you to create reusable components.
- **Dependency Injection**: This feature allows you to define dependencies that are automatically injected into components, improving code modularity and testability.
- **Templating**: AngularJS provides an expressive templating language that binds your view (HTML) with the model data.
- **Routing**: It includes a routing mechanism to handle navigation in SPAs.

Use Cases:

AngularJS is commonly used for building SPAs and real-time applications, where the data is frequently updated on the client side without reloading the entire page.

---

5. React.js:

Definition: React.js is a JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create reusable UI components and efficiently update and render them when data changes.

Key Features:

- **Component-Based Architecture**: React encourages the creation of reusable components that manage their own state, which can be combined to build complex UIs.
- **Virtual DOM**: React uses a virtual DOM, which makes updates to the real DOM efficient by only updating components that have changed.
- **Unidirectional Data Flow**: React follows a one-way data flow, making it easier to track the flow of data through an application and debug issues.
- **JSX (JavaScript XML)**: React uses JSX, a syntax extension that allows developers to write HTML-like syntax directly within JavaScript, enhancing code readability and structure.
- **React Hooks**: Hooks allow developers to use state and lifecycle features in functional components, making the code cleaner and more reusable.

Use Cases:

React.js is used to build dynamic user interfaces for single-page applications, social media platforms, dashboards, and e-commerce sites. It is used by companies like Facebook, Instagram, and Airbnb.

---

6. Docker:

Definition: Docker is a platform that allows developers to automate the deployment of applications inside lightweight, portable containers. These containers bundle the application code with all its dependencies, libraries, and configurations to ensure that the application runs consistently across different environments.

Key Features:

- **Containers**: Docker containers are lightweight, executable units that package application code along with its dependencies. Containers can run on any system that has Docker installed, ensuring consistency between development, staging, and production environments.

- **Docker Hub**: A vast library of pre-built container images available for use, including databases (MySQL, MongoDB), web servers (Nginx, Apache), and more.
- **Portability**: Containers can be deployed on any environment—be it a local machine, on-premise servers, or cloud services—without the need for reconfiguration.
- **Isolation and Security**: Containers run isolated from each other, ensuring that they don't interfere with other applications on the same host.
- **Orchestration**: Tools like Docker Swarm and Kubernetes are used to manage the deployment, scaling, and networking of containerized applications in production.

Use Cases:

Docker is widely used for application deployment, continuous integration/continuous deployment (CI/CD), microservices architectures, and creating reproducible development environments.

3.Describe Cloud computing service and cloud computing architecture.

Cloud Computing Services

Cloud computing provides on-demand access to computing resources—like servers, storage, databases, networking, software, and analytics—over the internet ("the cloud"). The key benefits of cloud computing include scalability, cost-efficiency, flexibility, and ease of access. There are three main types of cloud computing services:

1. **Infrastructure as a Service (IaaS)**:
    - **Description**: Provides virtualized computing resources over the internet, including virtual machines, storage, and networking.
    - **Examples**: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP).
    - **Use Case**: IaaS is ideal for businesses that need flexible, scalable infrastructure without owning or managing hardware.
2. **Platform as a Service (PaaS)**:
    - **Description**: Provides a platform allowing customers to develop, run, and manage applications without dealing with the infrastructure.
    - **Examples**: Heroku, Google App Engine, AWS Elastic Beanstalk.
    - **Use Case**: PaaS is used by developers to focus on coding and app development while the platform handles underlying infrastructure, databases, and operating systems.
3. **Software as a Service (SaaS)**:
    - **Description**: Delivers software applications over the internet, typically via a subscription model.
    - **Examples**: Google Workspace, Microsoft 365, Salesforce.
    - **Use Case**: SaaS allows businesses and individuals to use software without installing it on local devices. Ideal for CRM, collaboration, and email solutions.
4. **Function as a Service (FaaS)**:
    - **Description**: Also known as serverless computing, FaaS allows developers to run code in response to events without managing infrastructure.
    - **Examples**: AWS Lambda, Google Cloud Functions, Azure Functions.
    - **Use Case**: FaaS is used for running microservices, handling real-time data processing, or triggering functions without worrying about servers.

Cloud Computing Architecture

Cloud architecture refers to the structure and components that make up the cloud, ensuring smooth delivery of services. It consists of several layers and components:

1. **Front-End Layer**:
    - **Description**: This is the user-facing part of the cloud system. It includes the client-side devices, browsers, or apps that users interact with.
    - **Components**: Client devices, web browsers, or user interfaces (UIs).
2. **Back-End Layer**:
    - **Description**: This is the server side of the cloud architecture that manages all the resources and services. It contains the infrastructure and platforms that power the cloud.
    - **Components**:
        - **Servers**: Host applications, databases, and data.
        - **Storage**: Cloud storage services that store data securely (e.g., databases, file storage systems).
        - **Virtualization**: Abstracts the hardware layer to allow multiple virtual machines to run on a single physical machine.
        - **Networking**: Ensures secure and efficient data transmission between servers, storage, and users.
3. **Cloud Storage**:
    - **Description**: A critical component that stores data and files. It can be public, private, or hybrid.
    - **Examples**: AWS S3, Google Cloud Storage, Azure Blob Storage.
4. **Middleware**:
    - **Description**: Software that connects and integrates different components of the cloud. It manages communication and data management between front-end and back-end layers.
    - **Examples**: Application servers, message queues, and databases.
5. **Management and Security**:
    - **Description**: The cloud's management services handle the monitoring, logging, and governance of cloud resources. Security involves identity management, data encryption, and firewall protection.
    - **Components**: IAM (Identity and Access Management), encryption services, and monitoring tools.

Cloud Deployment Models

- **Public Cloud**: Resources are owned and operated by third-party cloud service providers, and shared among multiple clients.
- **Private Cloud**: Resources are used exclusively by one organization. It can be hosted on-premises or by a third-party provider.
- **Hybrid Cloud**: Combines public and private clouds, allowing data and applications to be shared between them.
- **Multi-Cloud**: Uses multiple cloud providers to avoid vendor lock-in and increase redundancy.

These architectures and services provide businesses the ability to scale operations, reduce costs, and innovate rapidly.

4. a) create a simple webpage using html to take input value of name and password and submit button. explain the code step by step.

Html code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Form</title>
</head>
<body>

    <h2>Login Form</h2>

    <form action="/submit" method="POST">
        <!-- Name input field -->
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br><br>

        <!-- Password input field -->
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
        <br><br>

        <!-- Submit button -->
        <button type="submit">Submit</button>
    </form>

</body>
</html>
```

Explanation:

1. **DOCTYPE and HTML Tag**:
    - `<!DOCTYPE html>` declares the document type as HTML5.
    - `<html>` is the root element of the webpage.
2. **Head Section**:
    - `<head>` contains meta information about the webpage.
    - `<meta charset="UTF-8">` ensures the page uses UTF-8 character encoding, which is standard for text and supports most characters.
    - `<meta name="viewport" content="width=device-width, initial-scale=1.0">` ensures the webpage is responsive, adjusting for different screen sizes, especially mobile.
    - `<title>Simple Form</title>` defines the title that appears in the browser tab.
3. **Body Section**:
    - The main content is placed inside the `<body>` tag.
4. **Form Element**:
    - `<form>`: This defines a form that will be used to collect user inputs.
        - `action="/submit"`: Specifies where the form data will be sent when the form is submitted. In this case, it's set to "/submit" (this should be replaced with the actual processing URL).
        - `method="POST"`: Specifies the HTTP method to use when submitting the form. `POST` is used when sending sensitive data (like passwords), as it does not expose it in the URL.
5. **Name Input Field**:
    - `<label for="name">Name:</label>`: The label for the name input field, which associates with the input using the `for` attribute.
    - `<input type="text" id="name" name="name" required>`: An input field of type "text" for entering the name. The `id` is used to link it with the label, and `name="name"` defines the

variable that will be sent in the form submission. The `required` attribute ensures the user can't submit the form without filling in the name.

6. **Password Input Field**:
    - `<label for="password">Password:</label>`: The label for the password field.
    - `<input type="password" id="password" name="password" required>`: An input field of type "password" for entering the password. The `password` type hides the actual characters entered for security. The `required` attribute ensures this field must be filled before submission.
7. **Submit Button**:
    - `<button type="submit">Submit</button>`: This is a button of type "submit". When clicked, the form is submitted and the data (name and password) is sent to the server endpoint specified in the `action`.

4. b) describe cascading style sheets css .explain the 3 types with code and write any one example with output.

Cascading Style Sheets (CSS)

CSS (Cascading Style Sheets) is a language used to style and format the presentation of web pages. It describes how HTML elements should be displayed on screen, paper, or in other media. CSS allows you to control the layout of multiple web pages all at once, making it easier to maintain and update the design of a website.

Key Features of CSS:

1. **Separation of content and presentation**: HTML is used for content, and CSS is used for styling.
2. **Reusable styles**: You can apply the same style to multiple HTML elements.
3. **Cascading**: Styles are applied in a cascading manner. This means that more specific rules override more general ones.

Types of CSS

There are three main ways to include CSS in a webpage:

1. **Inline CSS**: Used to apply styles directly to an HTML element.
2. **Internal CSS**: Defined inside a `<style>` tag within the `<head>` section of the HTML document.
3. **External CSS**: Written in a separate file with a `.css` extension and linked to the HTML document.

---

1. Inline CSS

This method applies CSS directly inside an HTML tag using the `style` attribute. It is used for quick styling but should be avoided for larger projects because it mixes structure with presentation.

Example:

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Inline CSS Example</title>
</head>
<body>
```

```
    <h1 style="color: blue;">Hello, World!</h1>
    <p style="font-size: 18px;">This is an example of inline CSS.</p>

</body>
</html>
```

Explanation:

- The `style` attribute is used within the HTML elements (`<h1>` and `<p>`) to apply the CSS directly to them.

---

2. Internal CSS

Internal CSS is defined within the `<style>` tag inside the `<head>` section of the HTML document. This method is useful when you want to apply styles to a single page.

Example:

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Internal CSS Example</title>
    <style>
        h1 {
            color: green;
        }
        p {
            font-size: 20px;
            color: darkgray;
        }
    </style>
</head>
<body>

    <h1>Hello, World!</h1>
    <p>This is an example of internal CSS.</p>

</body>
</html>
```

Explanation:

- The `<style>` tag contains CSS that applies to the entire page. All `<h1>` elements are styled with `color: green`, and `<p>` elements have a specific font size and color.

---

3. External CSS

External CSS is written in a separate file with a `.css` extension and linked to the HTML document using the `<link>` tag. This is the most common and efficient method because it allows you to apply styles to multiple HTML documents at once.

HTML File (index.html):

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <title>External CSS Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>

    <h1>Hello, World!</h1>
    <p>This is an example of external CSS.</p>

</body>
</html>
```

CSS File (styles.css):

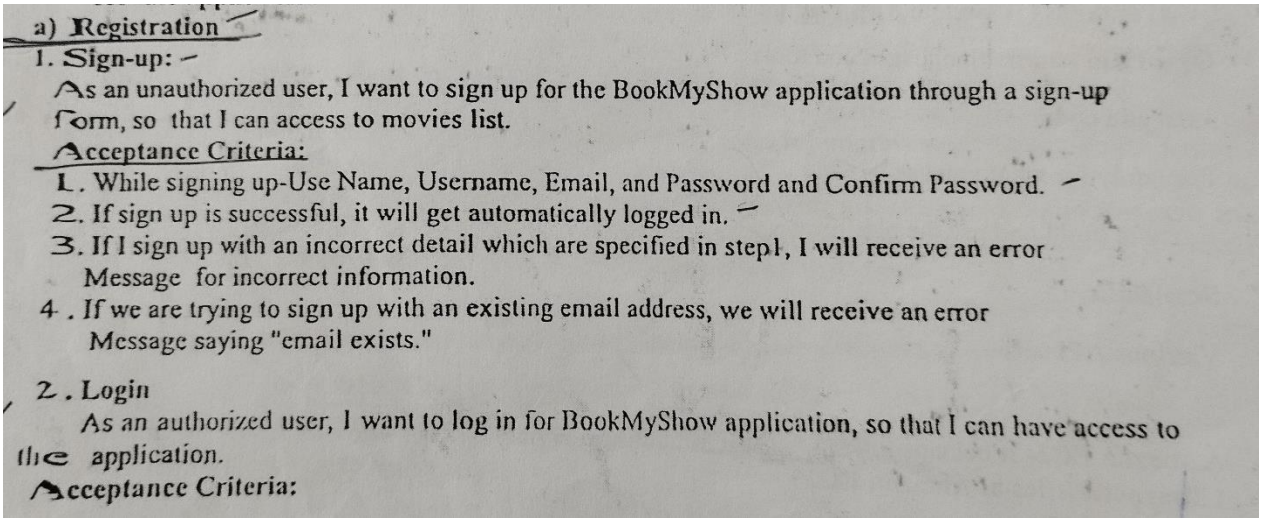
```
css
Copy code
h1 {
    color: purple;
    text-align: center;
}

p {
    font-size: 18px;
    color: black;
}
```

Explanation:

- The `<link rel="stylesheet" href="styles.css">` in the HTML file links to the external CSS file.
- The CSS file contains styles for the `<h1>` and `<p>` elements, which are applied across any HTML file that references this stylesheet.

5. 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 confirmed the details are sent to user. Identify and write the user stories for this application and also write Test cases for above application.



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.

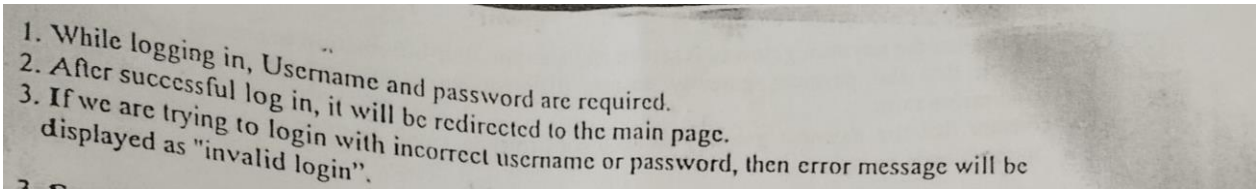
Acceptance Criteria:

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.

5. b) Swiggy is an online food ordering application that helps its users to buy a 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 is made, go to the payment page and make payment.
Write the user stories for this application. And Write the test cases for this application.

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. —8—

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.

5. **c) IRCTC - A Train ticket booking application that helps the users to book a train ticket for traveling across India. This application allows users to log in for booking tickets. Users can search for trains between source and destination places for a particular day. Once found, users can check the availability of tickets for different classes like general, sleeper, AC, etc. Users can book a particular train ticket on the required date. Once a ticket is booked after making the required amount through the online payment mode, the user can get the booking details.**
Identify and write the user stories for this application. And also write Test cases

User Story 1:
As a traveler, I want to log in to the IRCTC train ticket booking application, so that I can access and manage my train bookings seamlessly.
Acceptance Criteria:
- The user should be able to enter valid credentials (username and password) to log in.
- Upon successful login, the user should have access to their booking history and personal information.

User Story 2:
As a user planning a journey, I want to search for trains between source and destination places for a specific date, so that I can find suitable options for my travel.
Acceptance Criteria:
- The user should be able to input the source and destination locations.
- The user should be able to select a specific date for travel.
- The system should display a list of available trains with relevant details such as departure time, arrival time, and train type.

User Story 3:
As a traveler, I want to check the availability of tickets for different classes (general, sleeper, AC, etc.) on a particular train, so that I can choose the most suitable class for my journey.
Acceptance Criteria:
- The user should be able to view the availability of seats for various classes on a selected train.
- The system should display information on seat availability for the chosen date and class.

User Story 4:
As a user, I want to book a train ticket for a specific train on a chosen date, so that I can secure my travel arrangements.
Acceptance Criteria:
- The user should be able to select a train from the search results.
- The user should input the number of passengers, preferred class, and other necessary details.
- The system should provide a summary of the booking before confirming the reservation.

User Story 5:
As a traveler, I want to make an online payment for my booked train ticket, so that I can complete the booking process conveniently.
Acceptance Criteria:
- The user should be able to choose from various online payment modes (credit card, debit card, net banking, etc.).
- The payment process should be secure and prompt.
- Upon successful payment, the user should receive a confirmation of the booked ticket.

User Story 6:
As a user, I want to view and download the details of my booked train ticket, so that I have a record of my travel information.
Acceptance Criteria:
- The user should be able to access a digital copy of the booked ticket with details like PNR, train details, and passenger information.
- The system should provide an option to download or email the ticket details for offline reference.

These user stories cover the key functionalities of the IRCTC train ticket booking application, including login, searching for trains, checking availability, booking tickets, making payments, and accessing booking details.

Write test cases for the above application.

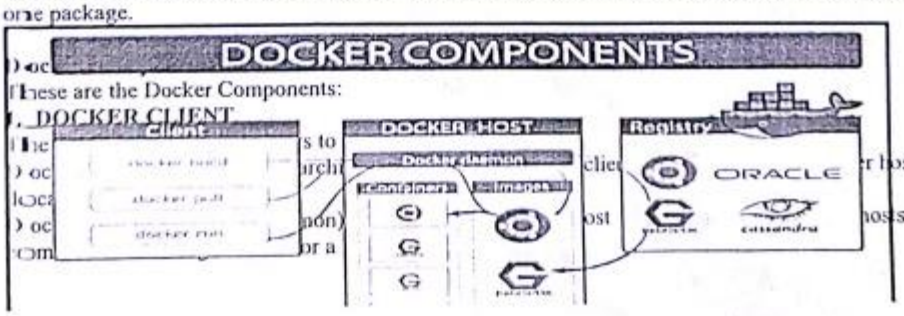
Test Cases for IRCTC Train Ticket Booking Application:

1. **Login Functionality:**
   - Verify that the user can log in with valid credentials.
   - Ensure that the user cannot log in with incorrect credentials.
   - Check for the visibility of the user's booking history after successful login.
2. **Train Search:**
   - Verify that the user can search for trains by providing valid source and destination locations.
   - Ensure that the user can select a specific date for the journey.
   - Check that the search results display relevant information such as train names, departure times, and arrival times.
3. **Availability Check:**
   - Verify that the user can check the availability of seats for different classes (general, sleeper, AC, etc.) on a selected train.
   - Ensure that the availability information is accurate and updated.
4. **Ticket Booking:**
   - Verify that the user can book a train ticket by selecting a specific train and providing necessary details (number of passengers, class, etc.).
   - Check that the system provides a summary of the booking for confirmation.
   - Ensure that the user cannot proceed with the booking if essential information is missing.
5. **Online Payment:**
   - Verify that the user can choose from various online payment modes (credit card, debit card, net banking, etc.).
   - Check the security of the payment process.
   - Ensure that the payment is processed successfully and the user receives a booking confirmation.
6. **Ticket Details Viewing:**
   - Verify that the user can view the details of the booked train ticket after successful payment.
   - Check for the accuracy of information such as PNR, train details, and passenger information.
   - Ensure that the user can download or email the ticket details for offline reference.
7. **Logout Functionality:**
   - Verify that the user can log out of the application.
   - Check that after logout, the user is redirected to the login screen.
8. **Error Handling:**
   - Verify that appropriate error messages are displayed when the user enters invalid information during login, search, or booking.
   - Ensure that error messages guide the user to correct their input.
9. **Navigation:**
   - Verify that the navigation through different stages of the booking process is smooth and intuitive.
   - Check that the user can easily go back to the previous steps if needed.
10. **Performance Testing:**
    - Verify the application's performance by simulating concurrent users searching for trains and booking tickets.
    - Ensure that the application handles simultaneous requests without significant degradation in performance.

These test cases cover a range of scenarios, including user authentication, search functionality, booking process, payment, and overall application usability.

6. Discuss the components of docker container.

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.



These are the Docker Components:

1. DOCKER CLIENT

The Docker client is the primary way that many Docker users interact with Docker. When we use commands such as **docker run**, the client sends these commands to **docker daemon**, which carries them out. The **docker** command uses the Docker API. The Docker 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

7. Analyze & Explain in detail about react concepts? write a simple program to create for diploma students using react.

React is a popular JavaScript library developed by Facebook for building user interfaces, particularly single-page applications (SPAs) where the content changes dynamically without needing a full page reload. Here are some key concepts:

1. Components

- **Components** are the building blocks of a React application. Each component is like a JavaScript function or class that can take input (called **props**) and return React elements that describe what should appear on the screen.
- Components can be functional (stateless) or class-based (stateful).

Example of a simple functional component:

```
function Greeting() {
  return <h1>Hello, Diploma Students!</h1>;
}
```

2. JSX (JavaScript XML)

- **JSX** allows us to write HTML inside JavaScript. It is a syntax extension that looks similar to XML or HTML. React uses JSX to describe the UI, and then Babel (a transpiler) converts it to pure JavaScript.

Example:

```
const element = <h1>Hello, world!</h1>;
```

3. State and Props

- **Props**: These are inputs to a component. Props are passed to components as function arguments and are **immutable**, meaning they cannot be changed by the component.
- **State**: This refers to data that changes over time and affects how the component behaves. State is managed within the component and is **mutable** (can be updated).

Example:

```
function Student(props) {
  return <h1>Hello, {props.name}!</h1>;
}

function App() {
  return (
    <div>
      <Student name="Ghanshyam" />
      <Student name="Aryan" />
    </div>
  );
}
```

4. Lifecycle Methods (For Class Components)

- **Lifecycle methods** are hooks in class components that allow you to control what happens at different stages of a component's life (Mounting, Updating, Unmounting).
- Some common methods include:
    - `componentDidMount()`: Called after the component is rendered the first time.

    - `componentDidUpdate()`: Called when the component is updated.
    - `componentWillUnmount()`: Called before the component is removed from the DOM.

5. Hooks (For Functional Components)

- **Hooks** are functions that let you use state and other React features in functional components.
- The most common hooks are:
    - `useState`: Allows you to add state to a functional component.
    - `useEffect`: Allows you to perform side effects in functional components (e.g., fetching data or updating the DOM).

Example using `useState` and `useEffect`:

```
import React, { useState, useEffect } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

6. Virtual DOM

- React uses a **Virtual DOM**, which is a lightweight copy of the actual DOM. When the state of an object changes, React updates the Virtual DOM first, compares it with the real DOM, and only updates the part of the real DOM that changed. This process is called **Reconciliation**, and it makes React very fast.

7. Conditional Rendering

- You can render different elements or components based on certain conditions using simple JavaScript control structures like `if` or the ternary operator.

Example:

```
function Welcome(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please log in.</h1>;
}
```

8. Handling Events

- Events in React are handled similarly to DOM events but with some differences in naming and syntax. In React, you pass a function as an event handler, rather than a string.

Example:

```
function handleClick() {
  alert('Button clicked!');
}

function App() {
  return <button onClick={handleClick}>Click Me</button>;
}
```

9. Forms and Controlled Components

- In React, forms are usually **controlled components**, meaning that the input elements are controlled by the state.

Example:

```
import React, { useState } from 'react';

function Form() {
  const [name, setName] = useState("");

  function handleSubmit(event) {
    event.preventDefault();
    alert(`Hello, ${name}`);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

Simple React Program for Diploma Students

Here's a basic program using React that displays a list of diploma students with their details:

```
import React from 'react';
import ReactDOM from 'react-dom';

// Student Component to display student information
function Student(props) {
  return (
    <div style={{ border: '1px solid black', padding: '10px', margin: '10px' }}>
      <h2>{props.name}</h2>
      <p>Course: {props.course}</p>
      <p>Year: {props.year}</p>
    </div>
  );
}

// Main App Component
function App() {
  const students = [
    { name: 'Ghanshyam', course: 'Computer Engineering', year: '2nd Year' },
    { name: 'Aryan', course: 'Mechanical Engineering', year: '3rd Year' },
    { name: 'Neha', course: 'Civil Engineering', year: '1st Year' },
  ];
```

```
  return (
    <div>
      <h1>Diploma Students</h1>
      {students.map((student, index) => (
              <Student key={index} name={student.name} course={student.course}
      year={student.year} />
      ))}
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

Explanation:

- **Student Component**: A reusable component that takes `name`, `course`, and `year` as props and displays the details of a student.
- **App Component**: The main component that renders a list of students using the `Student` component. The students' data is stored in an array, and `.map()` is used to render each student dynamically.

Steps to Run the Program:

1. Install Node.js and npm if you haven't already.
2. Use `npx create-react-app diploma-students` to set up a new React project.
3. Replace the code in `src/App.js` with the provided code.
4. Run `npm start` to view the application in the browser.

This simple program introduces diploma students and demonstrates how React can dynamically render components based on an array of data.

8. CRUD operation describe the convention a user-interface that let user view, search & modify parts of the Database. Explain how to create operation is used for insert new documents in the MongoDB database.

CRUD Operations Overview

CRUD stands for **Create, Read, Update, and Delete**, which are the four basic operations that can be performed on a database. These operations allow users to interact with the data stored in databases through a user interface or API.

1. **Create**: Insert new data into the database.
2. **Read**: Retrieve and display data from the database.
3. **Update**: Modify existing data in the database.
4. **Delete**: Remove data from the database.

Each of these operations can be used to manage the records (documents in MongoDB) effectively.

MongoDB and CRUD Operations

MongoDB, a popular NoSQL database, is document-oriented and stores data in flexible, JSON-like documents. It allows for easy and scalable database operations.

Create Operation in MongoDB

The **Create** operation in MongoDB is used to insert new documents into a collection. MongoDB provides various methods to insert documents, the most common of which are:

- `insertOne()`: To insert a single document.
- `insertMany()`: To insert multiple documents at once.

Each document is stored as a JSON-like structure (BSON) in MongoDB.

How to Insert New Documents in MongoDB (Create Operation)

Let's break down how to perform the **Create** operation in MongoDB.

Prerequisites

1. **MongoDB Server**: You need a MongoDB server running locally or in the cloud (like MongoDB Atlas).
2. **MongoDB Driver**: If using Node.js, you will need the `mongodb` package installed (`npm install mongodb`).
3. **Connection**: You'll need to connect to the MongoDB database using the connection string.

Example of Inserting a New Document

Step-by-Step Process:

1. **Connect to the MongoDB Database:**
   - Use the `mongodb` driver to establish a connection to the MongoDB database.
   - Use the `MongoClient` from the `mongodb` package to connect to a MongoDB instance.
2. **Insert a Document**:
   - Use `insertOne()` to insert a new document.

Here's a simple example in **Node.js**:

```
// Import MongoClient from the mongodb package
const { MongoClient } = require('mongodb');

// Connection URL to MongoDB server (local or cloud-based)
const url = 'mongodb://localhost:27017';

// Database Name
const dbName = 'school';

// Create a new MongoClient instance
const client = new MongoClient(url);

// Async function to perform the insert operation
async function insertStudent() {
  try {
    // Connect to the MongoDB server
    await client.connect();
    console.log('Connected successfully to MongoDB server');

    // Select the database by name
    const db = client.db(dbName);

    // Select the collection (equivalent to table in relational databases)
    const studentsCollection = db.collection('students');

    // Define a new document (student data) to insert
    const newStudent = {
      name: 'Ghanshyam',
      age: 19,
```

```
      course: 'Diploma in Computer Engineering',
      year: 2,
      email: 'ghanshyam@example.com',
    };

    // Insert the document using insertOne()
    const result = await studentsCollection.insertOne(newStudent);

    // Log the result of the insertion
    console.log(`New document inserted with _id: ${result.insertedId}`);
  } catch (err) {
    console.error('An error occurred while inserting the document:', err);
  } finally {
    // Close the connection to MongoDB server
    await client.close();
  }
}

// Call the insert function
insertStudent();
```

Explanation of the Code:

1. **MongoClient**: We create a MongoDB client instance using the connection URL to connect to the server.
2. **Database and Collection**: We select the database (`school`) and the collection (`students`) in which we want to insert the new document.
3. **Document**: We define a new student document that contains data like name, age, course, year, and email.
4. **insertOne()**: The method used to insert the document into the collection.
5. **Result**: After a successful insertion, MongoDB automatically generates a unique `_id` field for the new document. We can log this `_id` to verify the insertion.

Example Document After Insertion:

Once inserted, the document will look like this in the MongoDB `students` collection:

```
{
  "_id": ObjectId("someUniqueId"),
  "name": "Ghanshyam",
  "age": 19,
  "course": "Diploma in Computer Engineering",
  "year": 2,
  "email": "ghanshyam@example.com"
}
```

`insertMany()` Method (For Multiple Documents)

You can also insert multiple documents at once using `insertMany()`. Here's an example:

```
async function insertMultipleStudents() {
  try {
    await client.connect();
    const db = client.db(dbName);
    const studentsCollection = db.collection('students');

    // Define an array of new students
    const newStudents = [
      { name: 'Aryan', age: 21, course: 'Mechanical Engineering', year: 3 },
      { name: 'Neha', age: 20, course: 'Civil Engineering', year: 1 },
    ];

    // Insert multiple documents using insertMany()
    const result = await studentsCollection.insertMany(newStudents);
```

```
    console.log(`${result.insertedCount} new documents inserted with IDs:
${result.insertedIds}`);
  } finally {
    await client.close();
  }
}

insertMultipleStudents();
```

Summary of the Create Operation:

- The **Create** operation in MongoDB allows inserting new documents into collections.
- It can be done using `insertOne()` for single documents or `insertMany()` for multiple documents.
- MongoDB auto-generates a unique `_id` for each document if not provided.

This is how you can perform the Create operation to insert new records (documents) into a MongoDB collection.

9. Explain the JavaScript in detail. using Java Script language write simple code to create gaming application in the website.

JavaScript is a versatile, high-level programming language primarily used for creating interactive and dynamic web pages. It runs on the client side in the browser, making websites responsive to user interactions. It can also be used for server-side development (with Node.js), mobile app development (using frameworks like React Native), and more.

Key Concepts in JavaScript:

1. **Variables**: Containers that store data values. They can be declared using `var`, `let`, or `const`.
    - `var`: Global or function-scoped.
    - `let`: Block-scoped, can be updated.
    - `const`: Block-scoped, cannot be reassigned.

   Example:

   ```
   let playerName = 'John';
   const maxScore = 100;
   ```

2. **Functions**: Blocks of code designed to perform particular tasks. They can accept parameters and return values.
    - Example of a function declaration:

   ```
   function add(a, b) {
     return a + b;
   }
   console.log(add(5, 10)); // Output: 15
   ```

3. **Conditionals**: Used for decision-making. The `if`, `else if`, and `else` statements execute different blocks of code based on certain conditions.
    - Example:

   ```
   if (score > 50) {
     console.log('You won!');
   } else {
     console.log('Try again!');
   }
   ```

4. **Loops**: Execute a block of code repeatedly. Common loops include `for`, `while`, and `do...while`.
    - Example:

```
for (let i = 0; i < 5; i++) {
  console.log('Iteration: ' + i);
}
```

5. **Events**: JavaScript can detect events like clicks, key presses, etc., and run specific code in response.
    - Example:

```
<button onclick="startGame()">Start Game</button>
<script>
  function startGame() {
    console.log('Game started!');
  }
</script>
```

6. **Objects**: JavaScript is an object-oriented language. Objects are collections of key-value pairs.
    - Example:

```
let player = {
  name: 'John',
  score: 0,
  increaseScore: function(points) {
    this.score += points;
  }
};
player.increaseScore(10);
console.log(player.score); // Output: 10
```

create gaming application in the website.

HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>simon game</title>
    <link rel="stylesheet" href="simon.css">
</head>
<body>
    <h1>Simon Game</h1>
    <h2>press any key to start the game</h2>
    <div class="btn-container">
        <div class="line1">
            <div class="btn red" type="button" id="red">1</div>
            <div class="btn yellow" type="button" id="yellow">2</div>
        </div>
        <div class="line2">
            <div class="btn blue" type="button" id="blue">3</div>
            <div class="btn green" type="button" id="green">4</div>
        </div>
    </div>
    <script src="simon.js"></script>
</body>
</html>
```

CSS Code

```
body{
    text-align: center;
    background: skyblue;
}
h1{
    color:rgb(240, 9, 86);
}
.btn{
    width: 200px;
    height: 200px;
    border: 10px black solid;
    border-radius:20% ;
    margin: 0.5rem;
}
.btn-container{
    justify-content: center;
    display: flex;
}
.yellow{
    background: rgb(212, 129, 34);
}
.red{
    background: rgb(172, 27, 82);
}
.green{
    background: rgb(47, 209, 69);
}
.blue{
    background: rgb(106, 106, 226);
}
.flash{
    background: white;
}
.userFlash{
    background: green;
}
```

JS Cide

```
let gameseq = [];
let userseq =[];

let btns =["red","yellow","green","blue"]

let started = false;
let level =0;

let h2 = document.querySelector("h2");

document.addEventListener("keypress",function(){
    if(started == false){
      console.log("Game is started");
      started = true;
      levelUp();
```

```
    }
});


function gameflash(btn){
    btn.classList.add("flash");
    setTimeout(function () {
        btn.classList.remove("flash");
    },250);
}

function userFlash(btn){
    // console.log(btn);

    btn.classList.add("userFlash");
    setTimeout(function () {
        btn.classList.remove("userFlash");
    },250);
}


function levelUp(){
    userseq =[];
    level++;
    h2.innerText = level ${level};

    let randIdx = Math.floor(Math.random()*3);
    let randcolor = btns[randIdx];
    let randBtn = document.querySelector(.${randcolor});

//random btn choose
    gameseq.push(randcolor);
    gameflash(randBtn);
}

function checkAns(idx) {
    if (userseq[idx] == gameseq[idx]) {
        if (userseq.length == gameseq.length) {
            setTimeout(levelUp, 500);
        }
    }else{
        h2.innerHTML = game over!Your score was <b>${level}</b> <br>
press any key to start;
        document.body.style.backgroundColor = "red";
        setTimeout(function() {
            document.querySelector("body").style.backgroundColor =
"white"
        },150);
        reset();
    }
}

function btnpress(){
    let btn = this;
    userFlash(btn);
```

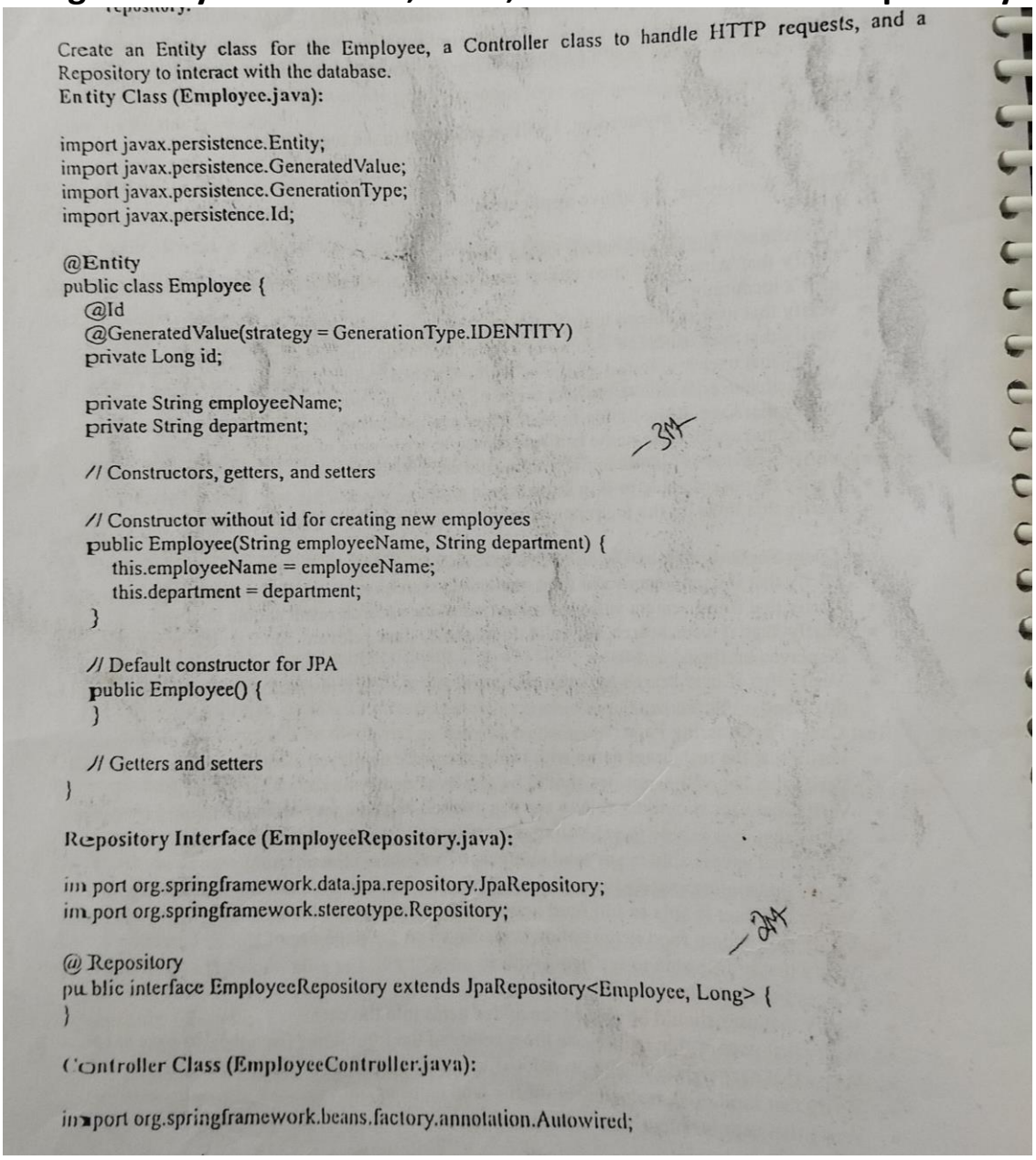
```
    usercolor = btn.getAttribute("id");
    userseq.push(usercolor);

    checkAns(userseq.length-1);
}

let allBtns = document.querySelectorAll(".btn");
for(btn of allBtns) {
    btn.addEventListener("click",btnpress);
}

function reset() {
    started = false;
    gameseq = [];
    userseq = [];
    level = 0;
}
```

10. Create a springboot application to maintain Employee details such as Employee id, Employee name, department mail ids & phone number, design Entity or Data JPA, Class, controller class and repository.



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);
    }
}
```

11. Explain in detail about enterprise and their types and steps.

What is an Enterprise?

- Enterprise refers to a for-profit business started and run by an entrepreneur. And we will often say that people running such businesses are enterprising.
- The roots of the word lie in the French word *entreprendre (from prendre),* meaning 'to undertake', which in turn comes from the Latin "*inter prehendere*" (seize with the hand).
- Entrepreneurs usually start an enterprise – with the associated risks – to make a profit, and for one of several reasons:

- **Problem-solving.** They see a particular issue that they feel they can solve.
- **Exploit ideas.** They have a **new idea or product** they believe will be successful.
- **Filling a gap.** They see a gap in the market they believe they can fill.
- **Competitive pricing.** They believe they can produce something on the market cheaper and offer it at a lower price.
- **Knowledge-based.** Where they believe they can supply specialist knowledge that customers will pay for.

Types of Enterprise

Various types of commercial enterprise exist within the UK. The main differences between them lie in how they are structured and legally owned.

1. Sole Proprietorship

Although often the smallest of companies, these represent the foundation of the UK's market economy. These can include 'trade' business, such as painters and decorators, or the owners of a single retail unit. And, in the modern era, many online businesses can fall into this category, from smaller enterprises selling products via Etsy or similar platforms to larger ones with a website and app.

2. Partnership

Partnerships usually consist of a small number of individuals who share ownership and decision-making (as well as profits). In some cases, such as legal firms, each partner may bring a particular speciality to the business to expand the overall services. In some cases, there may be a type of hierarchy where there are senior and junior partners.

3. Private Limited Companies (Ltd.)

- This sort of free enterprise has been legally incorporated and will have its own legal identity.
- It will have a set of shareholders who shoulder a limited amount of liability for any debts the enterprise incurs.
- Those shareholders will appoint directors to oversee overall operations and decisions of the business, though the relevant managers will oversee the day to day operations.

4. Public Limited Companies (PLC)

- Often confused with private limited companies, PLCs differ in that shares in the enterprise can be sold to the general public.
- To do this, they have to meet certain regulatory and legal criteria regarding the business's financial health, transparency of their accounts, how long they have been trading, and more.
- Being able to sell public shares can be useful in raising funds for things like expansion.

The Process of Organizing an Enterprise (With Diagram)

The five main steps involved in the process of organizing an enterprise.

The steps are: 1. Determining Activities 2. Grouping of Activities 3. Assigning Duties 4. Delegating Authority 5. Coordinating Activities.

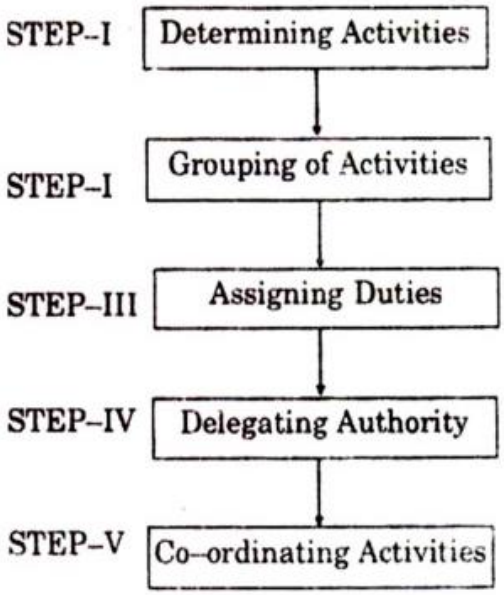


Fig. 6.2. Process of orgaisation.

Organising an Enterprise Step # 1. Determining Activities:

- The first step in organising is to identify and enumerate the activities required to achieve the objectives of the enterprise.
- The activities will depend upon the nature and size of the enterprise.
- For instance, a manufacturing concern will have production, marketing and other.

Organising an Enterprise Step # 2. Grouping of Activities:

- The various activities are then classified into appropriate departments and divisions on the basis of functions, products, territories, customers etc. Similar and related activities may be grouped together under one department or division.
- Grouping of activities helps to secure specialisation. Each department may be further sub divided into sections and groups. Grouping of activities should not only allow specialisation but keep in view the human factor, nature of activities and the needs of the organisation and the people.

Organising an Enterprise Step # 3. Assigning Duties:

- The individual groups of activities are then allotted to different individuals on the basis of their ability and aptitude.
- The responsibility of every individual should be defined clearly to avoid duplication of work and overlapping of effort. Each person is given a specific job best suited to him and he is made responsible for its execution.

Organising an Enterprise Step # 4. Delegating Authority:

- Every individual is given the authority necessary to perform the assigned task effectively. Authority delegated to a person should be commensurate with his responsibility.

- Through successive delegations a clear hierarchy of authority or chain of command running from the top to bottom of the structure is established. An individual cannot perform his job without the necessary authority or power.

Organising an Enterprise Step # 5. Coordinating Activities:

- The activities and efforts of different individuals are then synchronized. Such co-ordination is necessary to ensure effective performance of specialized functions.
- Interrelationships between different jobs and individuals are clearly defined so that everybody knows from whom he has to take orders and to whom he is answerable.

12. Explain how digital transformation bring the revolution in banking process.

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.