**Week-1**

**BUSINESS PROCESS**

**Understanding the business process and workflow within a business entity**

### What is a Business Process?

A business process is a set of related activities or tasks performed by individuals or systems in a structured, repeatable sequence to achieve a specific organizational goal. These processes are fundamental to the operation of any business, as they outline how work is accomplished, ensure consistency, and facilitate the efficient use of resources.

### **Key Characteristics of a Business Process**

1. Structured Sequence: Business processes are composed of a series of steps or activities that must be followed in a specific order to achieve the desired outcome.
2. Goal-Oriented: Each business process is designed with a clear objective in mind, such as producing a product, delivering a service, processing a customer order, or managing payroll.
3. Repeatability: Business processes are repeatable and can be performed multiple times, often by different people or systems, under varying conditions.
4. Defined Inputs and Outputs: A business process typically begins with specific inputs (e.g., raw materials, customer information) and ends with defined outputs (e.g., finished products, reports, services delivered).
5. Cross-Functional: Business processes often span across various departments or functions within an organization, requiring collaboration and coordination among different teams.

### **Types of Business Processes**

1. Operational Processes:
   * These are the core activities that directly contribute to the creation of goods or services. They are essential to the primary operations of the business.
   * Examples:
     + Manufacturing products.
     + Processing customer orders.
     + Delivering services.
2. Supporting Processes:
   * Supporting processes facilitate the primary operational processes by providing necessary resources or support. They do not directly create products or services but are essential for the organization to function smoothly.
   * Examples:
     + Human resources management.
     + Accounting and finance.
     + IT support.
3. Management Processes:
   * These processes involve planning, monitoring, and controlling business activities to ensure that the organization meets its goals. They are typically related to strategic planning and decision-making.
   * Examples:
     + Strategic planning.
     + Performance management.
     + Compliance and risk management.

### **Importance of Business Processes**

1. Efficiency and Productivity: Well-designed business processes streamline operations, reduce redundancy, and improve efficiency, leading to higher productivity.
2. Consistency and Quality: By following standardized procedures, organizations can ensure consistent quality in the products and services they deliver.
3. Accountability: Business processes define roles and responsibilities, making it clear who is responsible for each task, which enhances accountability.
4. Scalability: Standardized processes make it easier to scale operations as the business grows, ensuring that the organization can handle increased volumes without sacrificing quality.
5. Continuous Improvement: Business processes can be monitored, analyzed, and optimized over time, allowing organizations to continuously improve their operations and adapt to changing circumstances.

**What is Workflow?**

A workflow is a series of steps that outline how to complete a task or process from start to finish. These are a fundamental concept in business and various other fields, such as project management, healthcare, manufacturing, and information technology. They provide structured and systematic way to manage and execute processes.

**Business Process and work flow within a business entity**

* 1. Define Process Objectives

For each key process, clearly define its objectives. What are the specific goals that this process is supposed to achieve? For example, in the sales process, the objective might to be increase revenue or acquire new customers

* 1. Map the process workflow

Create a visual representation of the workflow for each process. This can be done using flowcharts, process,diagrams, or workflow software. Identify the sequence of tasks and activities involved in each process, as well as the decision points and interactions between different teams or individuals.

* 1. Identify Roles and Responsibilities

Determine who is responsible for each step within the workflow. Clearly define roles and responsibilities for each employee or team involved in the process.

* 1. Document procedures and Guidelines

Document the standard operating procedures (SOPs) and guidelines for each process. This should include detailed instructions on how each task is performed, the tools and technology used, and any quality or performance standard.

* 1. Analyze Performance Metrics

Implement key performance indicators (KPI) to measure the performance of each process. These metrics could include things like cycle time, error rates, customer satisfaction, and cost effectiveness. Regularly track and analyse these metrics to identify areas for improvements.

* 1. Identify Bottlenecks and Inefficiencies

Analyse the workflow to identify bottlenecks or areas of inefficiency. These are points in the process where work slows down, errors occur, or resources are underutilized. Addressing these issues can lead to significant improvements.

* 1. Automation and Technology Integration

Consider how technology and automation can streamline and enhance the workflow. Automation can help to reduce manual task, minimize errors, and improve efficiency. Evaluate software and tools that can be integrated into workflow.

* 1. Continuous Improvement

Implement a culture of continuous improvement within the organization. Encourage employees to suggest and implement changes to optimize the workflow. Regularly review and update processes to adapt to changing business conditions.

* 1. Training and communication

Ensure that employees are well trained and understand their roles in the workflow. Effective communication and training are essential for successful process execution.

* 1. Testing And validation

Before implementing major changes to a workflow, conduct testing and validation to ensure that the changes will achieve the desired outcomes without introducing new problems.

* 1. Compliance and regulation

Ensure that the workflow complies with any relevant industry regulations, legal requirement, and quality standards.

* 1. Document and Monitoring changes

Maintain thorough documentation of all process changes and their impact. Continuously monitor the workflow to ensure that improvements are sustained over time.

* 1. Feedback and adaption

Encourage feedback from employees, customers, and stakeholders. Use this feedback to adapt and refine the workflow as needed.

**Week-2**

**Create and Manage product backlog using appropriate tool like Jira**

**STEP 1: REVIEW PRODUCT STRATEGY AND BUSINESS GOALS**

Its critical that your backlog is well-aligned with the road map. To do this, you need to review, the product strategy as well as the business goals that you want to achieve. Then ask yourself, what do you want to achieve in upcoming sprint? From there, draw out your initiatives and place them in backlog.

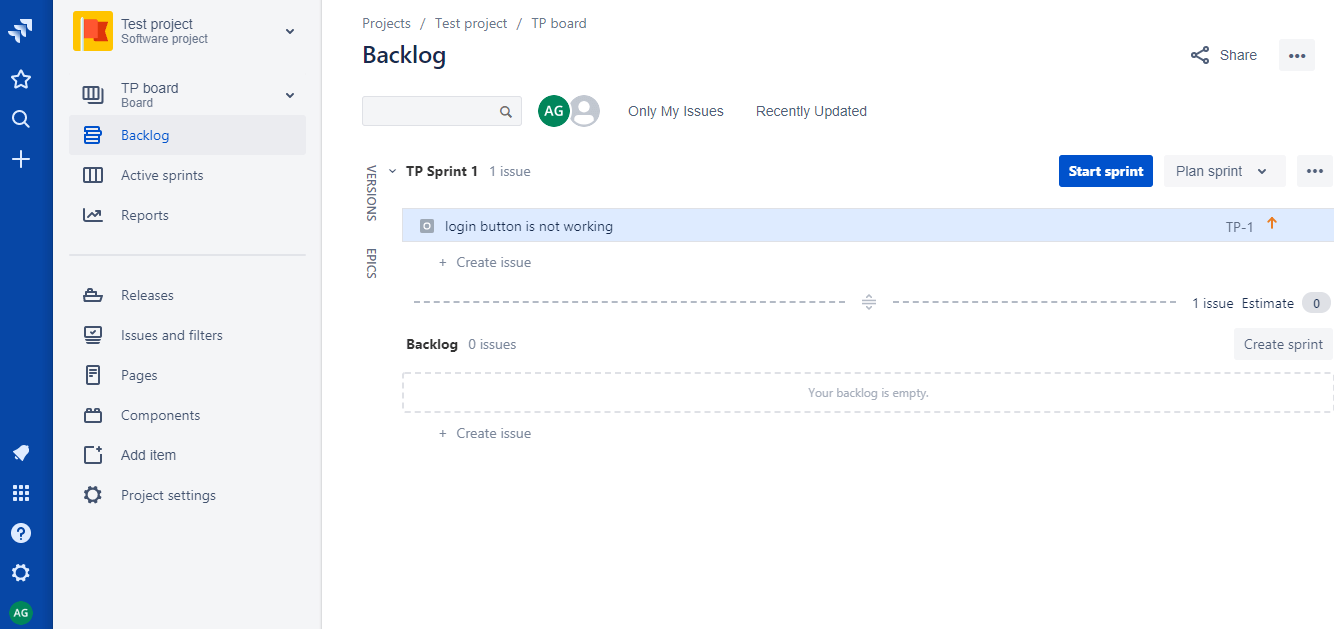
**STEP 2: BREAK DOWN BIG INITIATIVES INTO SMALLER UNITS OF WORK**

When it comes to agile product development. It’s key that you break down big initiatives into units of work small enough to be completed within a sprint. Let’s say your initiatives is to develop a content strategy for your upcoming marketing campaign. Breaking it down into smaller chunks of tasks will provide your team with a clear work breakdown structure (WBS) and offer transparency to your stakeholders. Simply create a parent issue (like an epic) and then support it with child issues made up of practical sized deliverable.



**STEP 3: SET PRIORITY LEVEL FOR ALL ISSUES TO REFLECT VALUE DELIVERY**

Consistent prioritization of product backlog issues is a must. The issues with the highest value delivery should be prioritized over the others. You can either drag and drop issues to arrange them prority or you can set priority manually by clicking on an issue and choosing the appropriate level. To learn how to define your own priority levels , read this.



**STEP 4: GROOM THE PRODUCT BACKLOG ON A REGULAR BASIS**

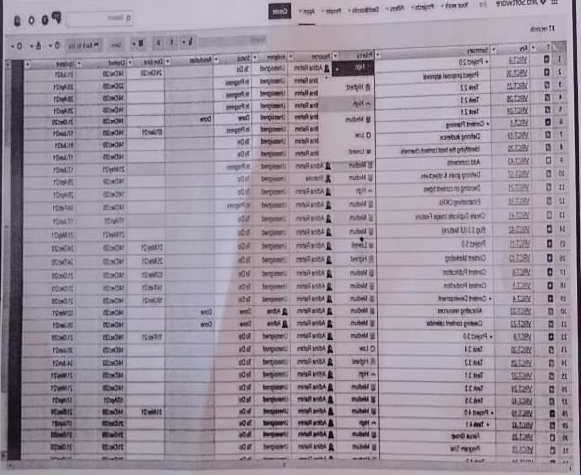
Backlog grooming, also known as backlog refinement, is a regular process that involves reviewing, prioritizing, and updating a product backlog. It's a key part of Agile development and is essential for keeping a project on track.

**STEP 5: LEVERAGE A BACKLOG MANAGEMENT TOOL TO ACCELERATE GROOMING**

Backlog management process is important as it helps teams define the scope of an iteration and stay on top of priorities.

But to do it effectively and efficiently each time, you need a boost in speed. Jira now offers a bulk edit feature where you can refine multiple issues at once, but with limitations - you have to go through multiple screens to bulk edit and the process is painfully inefficient.

A comprehensive, enterprise grade tool like Excel like issue editor Jira offers you the ability to bulk edit priorities, due dates, start dates, and summaries, for more than 10000 issues in one screen. To top it off, you can refine issues easily using the familiar Excel-like interface and features like Copy and Paste, Freeze Column, Sort and Filter.



**Week -3**

**Git, GitHub and Cloud basics**

**Experiment-1**

**Perform Local Git Operations**

Step 1: Install Git from the official website

Step 2: Launch the Git Bash in Windows

Step 3: Configure the name and email using following commands:

**git config --global user.name <USERNAME>**

**git config --global user.email <EMAIL>**

Step 4: Create folder with some files and launch a command prompt

Step 5: Navigate to the folder

Use the following commands to perform the basic Git operations:

**Command 1: Initialize the repository**

**git init**

**Command 2: View the status of the repository**

**git status**

**Command 3: Add the files to the stagging area**

**git add <filename>**

**Command 4: Commit changes to the repository**

**git commit -m “Message”**

**Command 5: View and Create branches**

**git branches**

**Command 6: View the commit histiry**

**git log**

**Command 7: switch between branches**

**git checkout branche name**

**Command 8: Merge two branches**

**git merge <branch name>**

**Command 9: git remote command to manage remote repositories**

**git remote add <git repository link>**

**Command 10 : To create a new branch and switch to it**

**git switch -c <New-Branch-name>**

**Experiment 2: Basics of Github operations such as creating an account, create push and pull the repository between GitHub and Git local repository.**

**Step 1: Open Github official website**

**Step 2: Create an account by clicking on signing up on Github**

**Step 3: Enter your email and click on continue**

**Step 4: Create a password for your account and click on continue**

**Step 5: Enter a username and click on continue**

**Step 6: Verify your account by solving captcha and click on create account**

**Step 7: Verify your email and you can sign in now with your email**

**Step 8: Now click on profile button to goto your repositories**

**Step 9: Click on New to create new repository**

**Step 10:Provide a name for your repository and check the ‘Add Readme file’**

**Step 11: Click on Create repository**

**Step 12: Goto desktop and create a folder with some dummy files.**

**Step 13: Launch command prompt and navigate to the created folder**

**Step 14: Use the following command to initialize repository**

**git init**

**Step 15: Now commit all the files using following command:**

**git commit -m “This is first commit”**

**Step 16: Now add the remote link of GitHub to the local git folder to connect the folder with the GitHub repository and push the repository**

**Step 17: The files will appear on the GitHub**

**Step 18: Click on Add New File -> Create new file -> provide a name to your file and write some content**

**Step 19: Add a commit message and click on commit**

**Step 20: To get the files on our local system we just added perform pull operation with following command**

**git pull**

**Experiment 3: Create repository name min-project-1. Push the same to GitHub**

Step 1: Open browser and login to your GitHub account

Step 2: Goto your repository and click on new to create a new one name ‘mini-project1’ by adding Readme file.

Step 3: Go to Desktop and create a folder with some dummy files

Step 4: Launch command prompt and navigate to the created folder

Step 5 : Use the following command to initialize repository

git init

Step 6:Now commit all the files using following command:

git commit -m “This is first miniproject”

Step 7: Now add the remote link of GitHub to the local git folder to connect the folder with the GitHub repository and push the repository

Step 8: The files will appear on the GitHub

Step 9: Now click on profile button goto your repositories

Step 10: Click on New to create a new repository

Step 11: Provide a name for your repository and check the ‘Add Readme file’

Step 12: Click on create repository

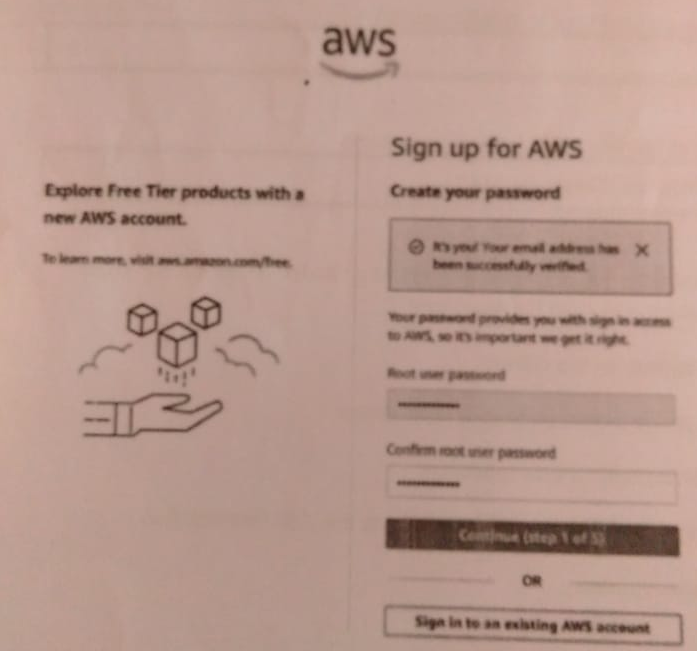
**Experiment 4: Create an account on Amazon Web Services (AWS)**

Step 1: To create an account on AWS or to go to AWS console click sign in button

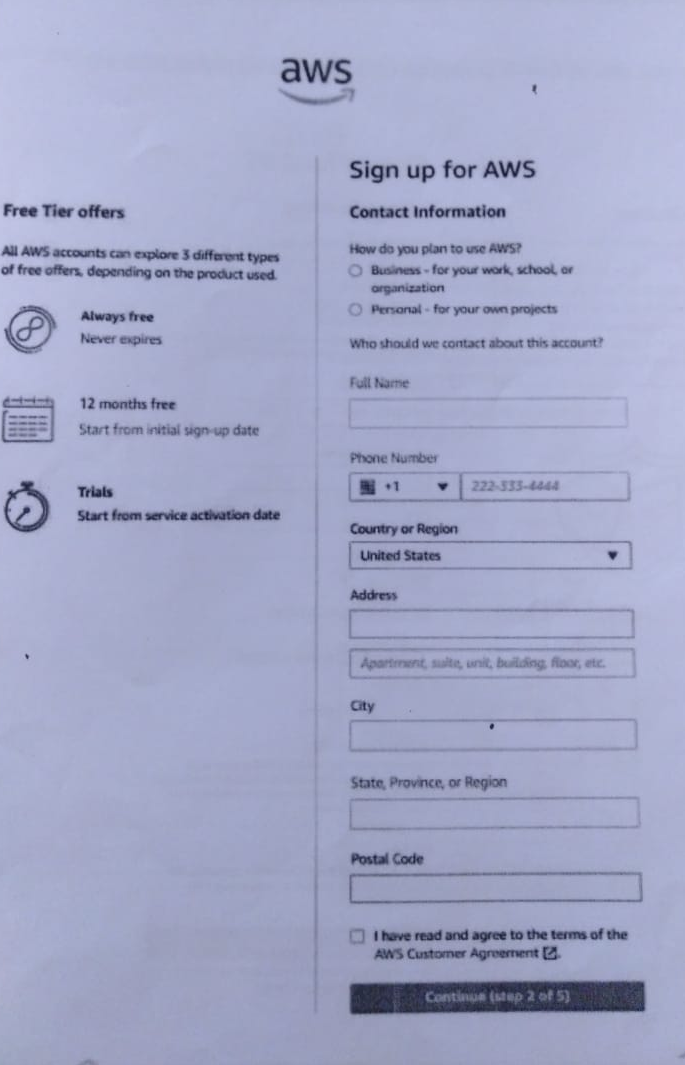
Step 2: Click on create new account

Step 3: Enter your email address and Account name. Verify your email address

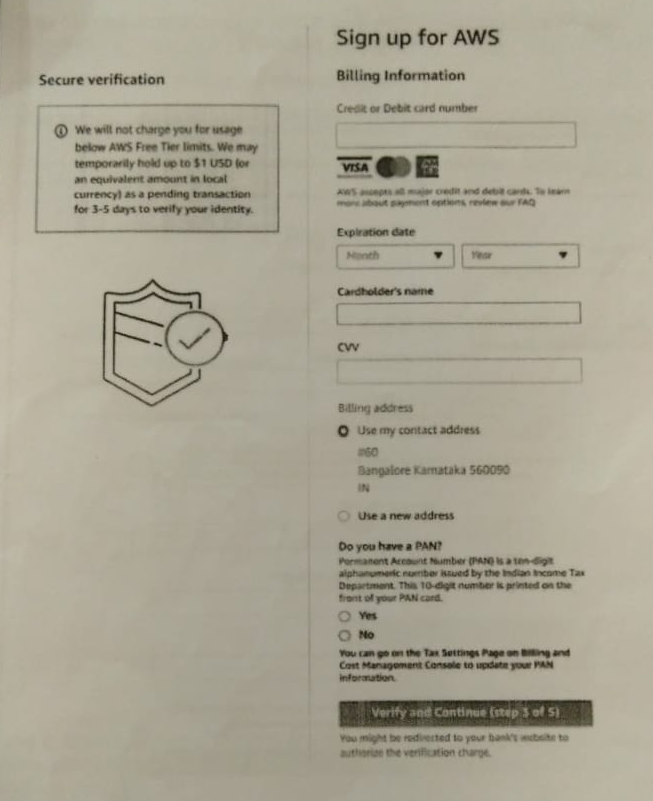
Step 4: After the verification click on next and enter the password for your AWS account and click on continue.



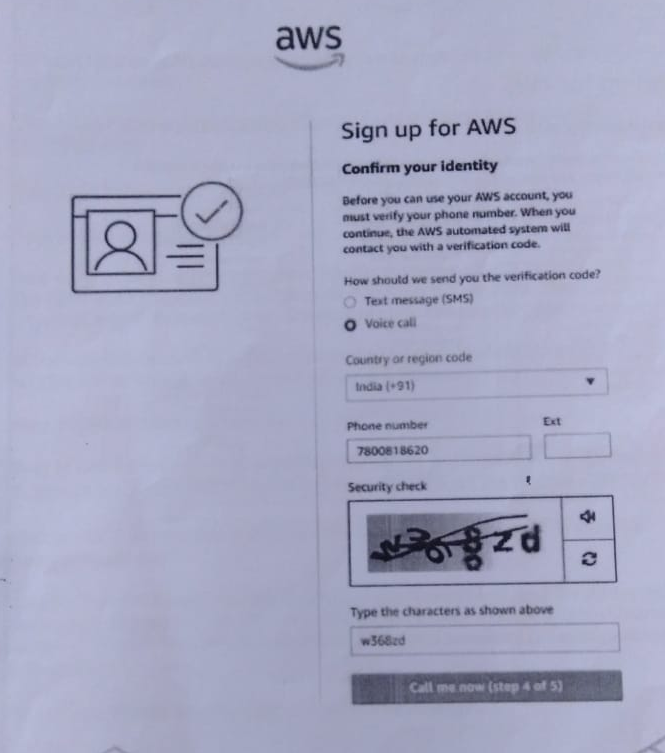
Step 5: In the next step we have to provide some information about us like fullname, mobile number, city..etc. For your verification purpose once you are done click on continue.



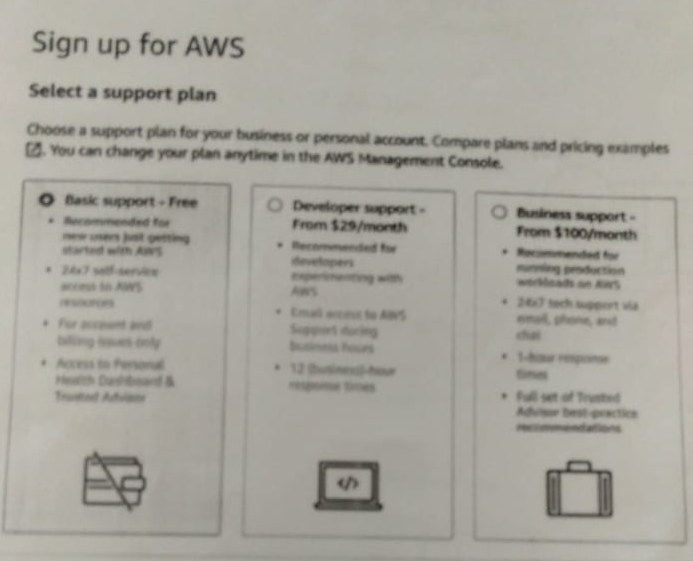
Step 6: In the next step we have to provide our card details for the payment and it is mandatory.



Step 7: After that you have to confirm your identity either via text or voice call.



Step 8: After confirming your identity, please go with the free tier plan and click on sign up button



**Experiment 5: Build a Basic web application on Amazon Web Service(AWS)**

We must have active AWS account to work on Amazon Web Services and deploy our web application there.

Step 1: Login to your AWS Console either by using User/Password or Root Email/Password

Step 2: **Goto Services > Compute > EC2 Virtual Server in the cloud.**

Step 3: Click on Launch Instances.

Step 4: An instance wizard will appear. Provide the name of EC2, in OS select Windows and for Image choose Microsoft Windows Server 2022. Select Instance type as t2.micro (free tier eligible). Create a key-pair and save it in your local machine.

In Network setting apart from default settings checkmark the both boxes saying “Allow HTTP/s traffic from Internet’. Click on Launch Instance.

Step 5: Click on view all instances

Step 6: Wait for few minutes. Select the instance you just created and click on connect button on top right window > **RDP Client > Download the remote desktop file**.

Click on Get Password and import the PEM file that you downloaded in step 4 > Decrypt Password

Step 7: Open the remote desktop file you just downloaded and copy the decrypted password. Allow the firewall and login wizard will appear prompting for username and password. Provide the username (Administrator) and password (Copied password). Click on OK.

Your created Virtual machine will boot and will be displayed.

**Experiment 6: To create a simple web application on AWS. Open an EC2 instance**

Step 1: Click on Start > Server Manager and open It.

Step 2: Once Server Manager is opened click on “Add roles and features” under configure this local server menu.

Step 3: Now wizard will be opened for installing web server to our EC2 Instance.

Step 4: Click Next to display the Select server roles page

Step 5: Select the Web Server (IIS) option. When a dialog opens, click Add Features

Step 6: Click Next to display the Select features page. Do not change default settings in the Features scroll box. ;

Step 7: Click Next two times to display the Select role services page. Click on Install and wait for few minutes and click on close.

Then you test the IIS installation by opening your web browser and enter IPv4 public IP of EC2 in your browser and press enter and you will see the default screen for IIS. Creating webpage and hosting it on EC2 Instance

Step 8: Now after installing IIS, open File Explorer and navigate to C:/netpub/wwwroot. The wwwroot folder is new in ASP.NET 5.0. All of the static files in your project go into

‘this folder.

Step 9: Create a index.html file with some data in HTML tag and save in the wwwroot

Step 10: Then in the new tab and enter ipv4 public IP address of your EC2 machine in your browser and press enter and you will see the HTML page you have created.

**Experiment 7: Create build pipeline for simple web application such as to do app, BMI Calculator, Number Convertor etc.**

**To create a javascript file on GitHub repository**

Step 1: Open or login to your GitHub account

Step 2: Create a repository

Step 3: Click on create a file

Step 4: Write the following code:

var name = “Raju";

console.log(`My name is ${name}`);

Step 5: Login to Jenkins and go to Dashboard. Click on create new project.

Step 6: Enter an item name and select pipeline then click OK

Step 7: A-configuration page will appear. Click on pipeline.

Step 8: Select Definition as Pipeline Script. Generate a Hello World Script and click on

pipe

Step 9: Click on pipeline syntax, a new tab will open and select sample step as ‘checkout: check out for version control’. SCM as Git Enter the repository URL which you want to build. Enter the credentials of your GitHub. Verify the branch name. And click on Generate pipeline script. Paste the generated in line 7 of your pipeline script (in the following code).

Do the same for sample step as ‘git: Git' and paste the generated pipeline syntax in line

12 in the following code.

pipeline {

agent any

stages {

stage('checkout') {

steps {

<Generated pipeline script for git: checkout>

}

stage('build') {

steps {

<Generated pipeline script for git:Git>

Bat label: ' ', script:'node <file\_name>.js'

}

}

Stage('test') {

steps {

echo 'Test was successful'

}

}

}

}

Step 10: Click save. Click on Build Now

Step 11: Click on the latest build in Build History and Click on Console

**Week 4**

**JSON**

JSON stands for JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. The JSON filename extension is .json

The most common use of JSON is to transmit data between server and client. It can also be used as a temporary storage.

For ex:

{

"emp": [

{

"empno": "01",

"empname": "Ganesh",

"Salary": 50000

},

{

"empno": "02",

"empname": "Sanjay",

"Salary": 60000

}

]

}

Object

* It is an unordered set of name/value pairs.
* Objects are enclosed in curly braces that is, it starts with '{' and ends with '}'.
* Each name is followed by ':'(colon) and the key/value pairs are separated by , (comma).
* The keys must be strings and should be different from each other.
* Objects should be used when the key names are arbitrary strings.

Syntax

{ string : value, .......}

Example

Example showing Object −

{

"id": "011A",

"language": "JAVA",

"price": 500,

}

**Program 1:**

<html>

<head>

<title>JSON example</title>

<script language = "javascript" >

var object1 = { "language" : "Java", "author" : "herbert schildt" };

document.write("<h1>JSON with JavaScript example</h1>");

document.write("<br>");

document.write("<h3>Language = " + object1.language+"</h3>");

document.write("<h3>Author = " + object1.author+"</h3>");

var object2 = { "language" : "C++", "author" : "E-Balagurusamy" };

document.write("<br>");

document.write("<h3>Language = " + object2.language+"</h3>");

document.write("<h3>Author = " + object2.author+"</h3>");

document.write("<hr />");

document.write(object2.language + " programming language can be studied " + "from book written by " + object2.author);

document.write("<hr />");

</script>

</head>

<body>

</body>

</html>

**Week 5**

JSX

**What is JSX?**

JSX stands for Javascript XML (Extensible Markup Language). JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React.

For Ex:

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

**Expressions in JSX**

In JSX, expressions are Javascript code snippets enclosed in curly braces {} that allow dynamic values or logic to be included within the markup.

For ex:

**import** React, { Component } from 'react';

**class** App **extends** Component{

   render(){

**return**(

         <div>

            <h1 className = "hello" >{25+20}</h1>

</div>

);

}

}

export **default** App;

**Specifying Attributes with JSX**

In JSX, you can specify attributes for elements similar to HTML. Attributes are defined using HTML-like syntax, and dynamic values or expressions can be included using curly braces {}.

For Ex:

const imgurl=’example.jpg’;

const element=<img src={imgurl} alt=”Example Image”>;

**Specifying Children with JSX**

In JSX, you can specify children elements within the opening and closing tags of component. Children can include text, other JSX elements, or combination

For ex:

const element={

<div>

<h1>Hello World</h1>

<p>This is</div>

A JSX Example </p>

};

**Props and state**

**State**

In React, “state” refers to an object that represents the internal data of a component. It allows a component to keep track of information that can change over time, often due to user interactions. When the state of a component change, React automatically re-renders the component to reflect the updated state.

For Example :

import React, {useState} from ‘react’;

const Counter={}=>{

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

Return(

<div>

<p>Count:{count}</p>

<button onClick={()=> setCount(count+1)}>

Increament

</button>

</div>

);

};

Export default counter;

**Props**

In React, “props” (short for properties) are a way to pass data from a parent component to a child component. They allow you to customize and configure child components by passing values as attributes.

For Ex:

import React from ‘react’;

const Greeting=(props)=>{

return <p> Hello, {props.name} !</p>;

};

const App=()=>{

return <Greeting name=”John”/>;

};

export default App;

**Components**

**What is components?**

In React, a component is a reusable and self contained piece of user interface. It can represent a part of a web-page, like a button or a form, or even a more complex UI element. Components make it easier to manage and organize the code in a React application.

**Function and Class components**

**Function components** : In React, a functional component is a type of component that is defined as a Javascript function, It is also sometimes referred to as a stateless component or a presentational component.

For Ex:

import React from ‘react’;

const Greeting=(props)=>{

return <p> Hello, {props.name}!</p>;

};

const App=()=>{

return <Greeting name=”John”/>;

};

export default App;

**Class components**

In React, a class components is a type of component that is defined using ES6 class syntax. Class components have additional features compared to functional components, such as the ability to manage state and use life cycle methods.

For ex:

import React, {Component} from ‘react’;

class Greeting extends Component{

render()

{

return <p> Hello, {this.props.name}!</p>;

}

}

class App extends Component

{

render()

{

return <Greeting name=”John”/>;

}

}

export default App;

**Rendering a Component**

It involves creating an instance of that component and placing it in the virtual DOM or the actual DOM. Components can be rendered within other components, creating a tree like structure.

import React, {Component} from ‘react’;

class Greeting extends Component{

render()

{

return <p> Hello, {this.props.name}!</p>;

}

}

class App extends Component

{

render()

{

return (

<div>

<h1>Welcome to my app</h1>

<Greeting name=”John”/>

</div>

);

}

}

export default App;

Creating first component

Creating a simple React component involves defining a javascript function or class that returns JSX, which represents the UI of the component.

import React from ‘react’;

const MyFirstComponent=()=>{

return <h1> Hello, React!</h1>;

};

export default MyFirstComponent;

**Component Life Cycle Methods**

**Mounting Phase**

Constructor(): The constructor is called before a component is mounted. It is often used for initializing state and binding methods.

For ex:

constructor (props)

{

super(props);

}

**Updating Phase**

**shouldComponentUpdate(nextProps,nextState):** This method is called before rendering when new props or state are received. It determines if the component should re-render by returning a boolean.

shouldComponentUpdate(nextProps,nextState){

//perform action after component is updated

}

**UnMounting Phase**

componentWillUnmount(): This method is called just before a component is unmounted and destroyed. It is often used for cleanup, such as cancelling network requests or clearing subscriptions.

componentWillUnmount()

{

//perform cleanup before component is unmounted

}

**Handling Events**

Conditional Rendering

Conditional rendering in React refers to the practice of rendering different content or components based on certain conditions or login. This can be achieved using javascript expressions within JSX.

import React from ‘react’;

const ConditionalComponent=({condition})=>{

if (condition)

{

return <p> This content is rendered conditionally</p>

}

else

{

return<p>This content is rendered when the condition is not met</p>

}

};

const App=()=>{

const IsConditionMet=true;

return <ConditionalComponent condition={IsConditionMet}/>;

};

export default App;

List And Keys

In React, lists are created by mapping over an array of data and rendering a component for each item in the array.

Keys are used to give each element in the list a unique identifier, helping React efficiently update and reconcile the DOM

For ex:

import React from ‘react’;

const MyList=()=>{

const items=[{id:1, text:’item 1’},{id:2, text:’item 2’},{id:3, text:’item 3’}];

return(

<ul>

{items.map(item=>(

<li key={item.id}>{item.text}</li>

))}

</ul>

};

};

export default MyList;

**Week 6**

**Context**

React context is a method to pass props from parent to child component(s), by storing the props in a store and using these props from the store by child component(s) without actually, passing them manually at each level of the component tree. Context provides a way to pass values between components without explicitly passing a prop through every level of the component tree.

**When to use context?**

Context is used to share the data which can be considered “global” for React components tree and use that data where needed, such as the current authenticated user, theme, etc.

Whenever you want a store to keep your states or variables in and use them elsewhere in your program, use context.

**Example:**

If we have three components in our App, A->B->C.

Where A is the parent of B and B is the parent of C. To change a state from C and pass it to A, keep the state of A in store, then extract the state from store and use it in C. This eliminates the necessity of the state to pass through B. So, the flow is like A->C.

**How to use Context?**

There are three main steps to use the React context into the React application:

1. Create React context
2. Setup a context provider and define the data which you want to store.
3. Use a context consumer whenever you need the data from the store.

Full Stack Development-20CS521 2023-24

Computer Science and Engineering Page 2

**React Context API**

The React Context API is a component structure, which allows us to share data across all levels of the application. The main of the Context API is to solve the problem of prop drilling (also called “Threading”). The Context API in React are given below:

1. React.createContext
2. Context.provider
3. Context.Consumer
4. Class.contextType

**REACT HOOKS**

**Introduction**

Hooks are the new feature introduced in the React 16.8 version. It allows us to use state and other React features without writing a class. Hooks are the functions which “hook into” React state and lifecycle features from function components. It does not work inside classes. Hooks are backward-compatible, which means it does not contain any breaking changes.

**When to use Hooks?**

Before Hooks whenever there is a requirement to add some state to function component, it would demand us to convert function component to a class component. With the introduction of Hook, this can be achieved by residing inside the existing function component itself.

**Rules of Hooks**

Hooks are like Javascript functions but need to follow these two rules while using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code.

**These rules are :**

1. Only call Hooks at the top level. Do not call Hooks inside the loops, condition, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a component render.
2. Only call Hooks from React functions. Hooks cannot be called from regular javascript functions. Instead, they are called from React function components. Hooks can also be called from custom Hooks.

**Week 7**

**Apache Maven**

**Why Maven?**

There are many problems that we face during the project development. They are discussed below:

**1) Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

**2) Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

**3) Building and Deploying the project:** We must have to build and deploy the project so that it may work.

**What is Maven?**

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project’s build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development team’s environment, Maven can set up the way to work as per standards in a very short time. As most of the project setups are simple and reusable. Maven makes life of developer easy while crating reports, checks, build and testing automation setups.

Maven simplifies the above mentioned problems. It does mainly following tasks.

1. It makes a project easy to build
2. It provides uniform build process (maven project can be shared by all the maven projects)
3. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
4. It is easy to migrate for new features of Maven

**Spring Framework**

It is one of the most popular JAVA EE framework. It is an open source lightweight framework that allows Java EE 7 developers to build simple, reliable, and scalable enterprise applications, This framework mainly focuses on providing various ways to help you manage your business objects. Spring Annotation are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program. So in this article , we are going to discuss what are the main types of annotations that are available in the spring framework with some examples.

Type of Spring Framework Annotations

There are 6 types of annotations

1. Spring core annotations
2. Spring web annotations
3. Spring Boot annotations
4. Spring scheduling annotations
5. Spring Data annotations
6. Spring Bean Annotaions

**Dependency Injection**

Dependency injection is the ability of an object to supply dependencies of another object, Dependency injection (DI) is a design pattern that implements the inversion of control for resolving the dependencies at run-time i.e. injecting the dependency object to the dependent object to use it at run time. The most important module in the Spring framework is core container and dependency injection (DI) acts as the heart of Spring’s core container.

In a conventional way, Developers will have control over the code in creating the object & injecting them at run time. Here the spring framework takes the control of doing the above mentioned activities at run time, that’s why the term is named as ‘Inversion of control’ (IOC) .i .e. the control is inverted. Dependency in programming is an approach where a class uses specific functionalities of another class.

So for example, if you consider two classes A and B, and say that class A uses functionalities of class B, then it is implied that class A has a dependency of class B. Now if you are coding in Java then you must know that you must create an instance of class B before the objects are being used by class A.

So the process of creating an object for some other class and let the class directly using the dependency is called Dependency Injection.

**Types of Dependency Injection**

1. Constructor Injection
2. Property Injection
3. Method Injection

**WEEK 8**

**Q 1. Create a REST full service with CRUD operations using Spring Data JPA and Spring Data REST ?**

1. **Create a spring Boot Project :**

**Start by creating a new Spring Boot project in your preferred development environment (e.g. Intellij IDEA, Eclipse)**

1. **Add Dependencles :**

**Make sure to include the necessary dependencies in your project’s “pom.xml” or “build.gradle” file.**

**You’ll need dependencies for Spring Boot ,Spring Data JPA & Spring Data REST. Here’s an example for Maven:**

**XML code:**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-data-jpa</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-data-rest</artifactId>**

**</dependency>**

1. **Configure Your Database:**

**Configure your database connection in the ‘application.Properties’ or ‘application.yml’ file. For example if you’re using H2,you can add the following configuration:**

**Yaml code:**

**application:**

**Configuration:**

**yaml code:**

**spring.datasource.driver-class-name: org.postgresql.Driver**

**spring.datasource.url: jdbc:postgresql://localhost:5432/your\_database\_name**

**spring.datasource.username: your\_username**

**spring.datasource.password: password**

**spring.jpa.hibernate.ddl-auto: update**

1. **Create an Entity: Define your JPA entity class, which represents the data you want to expose via the RESTful service. Annotate the class with ‘@Entity’, and define the fields and their mapping annotations.**

**For example :**

**Java code:**

**import javax.persistence.\*;**

**@Entity**

**public class Product {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**private String name;**

**private double price;**

**// Getters and setters**

**public Long getId() {**

**return id;**

**}**

**public void setId(Long id) {**

**this.id = id;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public double getPrice() {**

**return price;**

**}**

**public void setPrice(double price) {**

**this.price = price;**

**}**

**}**

1. **Create a Repository Interface:**

**Create a Repository interface that extends ‘JpaRepository’ to perform CRUD operations on your entity.**

**For example:**

**Java code:**

**interface ProdRepository extends JpaRepository<Product, Long> {**

**}**

1. **Run Your Application : Spring Boot will automatically create the necessary database tables and expose CRUD operation via Restful endpoints. You can run your application by adding a ‘main’ method in a class annotated with @SpringBootApplicarion.**
2. **Accessing the API: Once your application is running to the following endpoints (assuming you’re running on the default port 8080):**

* **‘GET/products’: List all products.**
* **‘POST/products: Create a new product.**
* **‘GET/products/{id}’: Retrieve a specific product by ID.**
* **‘PUT/products/{id}’: Update a specific products by ID.**
* **‘DELETE/products/{id}’: Delete a specific product by ID.**

Spring Data RE**ST automatically generates these endpoints for you , making your service RESTful with CRUD operations.**

**Make sure you have the necessary dependencies and configurations in place, & your project should work as a RESTful service with Spring Data JPA and Spring Data REST.**

**Transaction management and compliance to ACID principles:**

ACID (Atomicity, Consistency, Isolation,Durability) principles are essential for transaction management and compliance in database system. Here’s a brief overview of how they relate:

1. Atomicity: This principle ensure that a transaction is treated as a single, indivisible unit of work. It means that either all the changes made by a transaction are committed to the database, or none of them are. This property helps maintain the integrity of the database by preventing partial or incomplete updates.
2. Consistency: The consistency principle ensure that a transaction takes the database from one consistent to another. It enforces data integrity and business rules, ensuring that the database remains in a valid state even if a transaction fails.
3. Isolation: Isolation ensures that concurrent transactions do not interface with each other. It provides a mechanism to isolate one transaction is complete.
4. Durability: Durability guarantees that once a transaction is committed, its changes are permanent and will survive any system failures, including power outgates.

Securimg REST APIs with spring security with API security configuration:

Securing REST APIs with Spring Security involves configuring security setting to control access to your API endpoints. Here are the general steps to secure a REST API with Spring Security:

1. Add Spring Security Dependency: Make sure you have the Spring Security dependency in your project’s build file(e.g. Gradle or Maven).
2. Configure Security: Create aa configuration class that extends WebSecurity configure Adapter and override its methods to configure security to define which endpoints should be protected and how they should be secured.

Java code:

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/public/\*\*").permitAll() // Allow access to public resources

.anyRequest().authenticated() // All other requests require authentication

.and()

.httpBasic(); // Use HTTP Basic Authentication

}

}

Define User Authentication: You need to define how users are authenticated. You can use in-memory user, a database, or an external authentication provider like LDAP or OAuth. Here’s an example of in-memory authentication.

Java code:

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("user").password(passwordEncoder().encode("password")).roles("USER")

.and()

.withUser("admin").password(passwordEncoder().encode("adminpassword")).roles("USER", "ADMIN");

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

1. Handle Cross -Origin Requests: If your API is accessed from different origins (CORS), you might need to configure CORS settings to allow or restrict requests form specific domains.
2. Token Based Authentication: For more authentication, you can implement token-based authentication using JWT(JSON Web Tokens) or OAuth2. Spring Security provides support for these methods as well.
3. Test Your Configuration: Ensure that your API security is working as expected by testing different scenarios, including authorized and unauthorized access.
4. Customize Error Handling : Customize error responses for unauthorized access, such as returning 401 Unauthorized or 403 Forbidden status codes.

These are the basic steps to secure a REST API with Spring Security.

Creating unit test with spring boot: To create unit in a Spring Boot application, you can use the Spring Testing framework, which provides various tools and annotations for testing different components of your application. Here’s a basic example of how to create a unit test for a service lass using JUnit and the Spring Testing framework:

1. Add the necessary dependencies to your ‘pom.xml’ or ‘build.gradle’ file if they are not already included. You’ll typically need dependencies for JUnit and Spring Boot Test.
2. Create a unit test class for your service. This class should be located in the same package as your service implementation and should use the ‘@RunWith (SpringRunner.class)’ and ‘@SpringBootTest’ annotations. Here’s an example:

Java code:

@RunWith(MockitoJUnitRunner.class)

public class ServiceUnitTest {

@Mock

private MyService myService;

@Test

public void testSomeMethod() {

// Arrange

// Set up any necessary mocks or test data

// Act

// Call the method you want to test

// Assert

// Use JUnit assertions to verify the expected behavior

}

}

1. In your test methods, you can use JUnit assertions and Mockito(if necessary) to write test cases for your service methods.
2. Run your unit test using your IDE or a build tool (e.g. mvn test or gradle test).

Rember that unit tests are meant to test individual components in isolation so you should mock external dependencies using @MockBean when necessary to isolate the unit nyou’re testing. You can also use other annotations like @BeforeEach, @AfterEach and @BeforeAll if you need setup or cleanup code for your tests.

Additionally . you may use tools like Mokito for mocking dependencies and creating more complex test secnarios.

This is a basic example of creating unit tests in a Spring Boot application. Depending on your projects structure & requirement you may need to create tests for controllers, repositories, and other components as well.

**WEEK 9**

**No SQL Database**

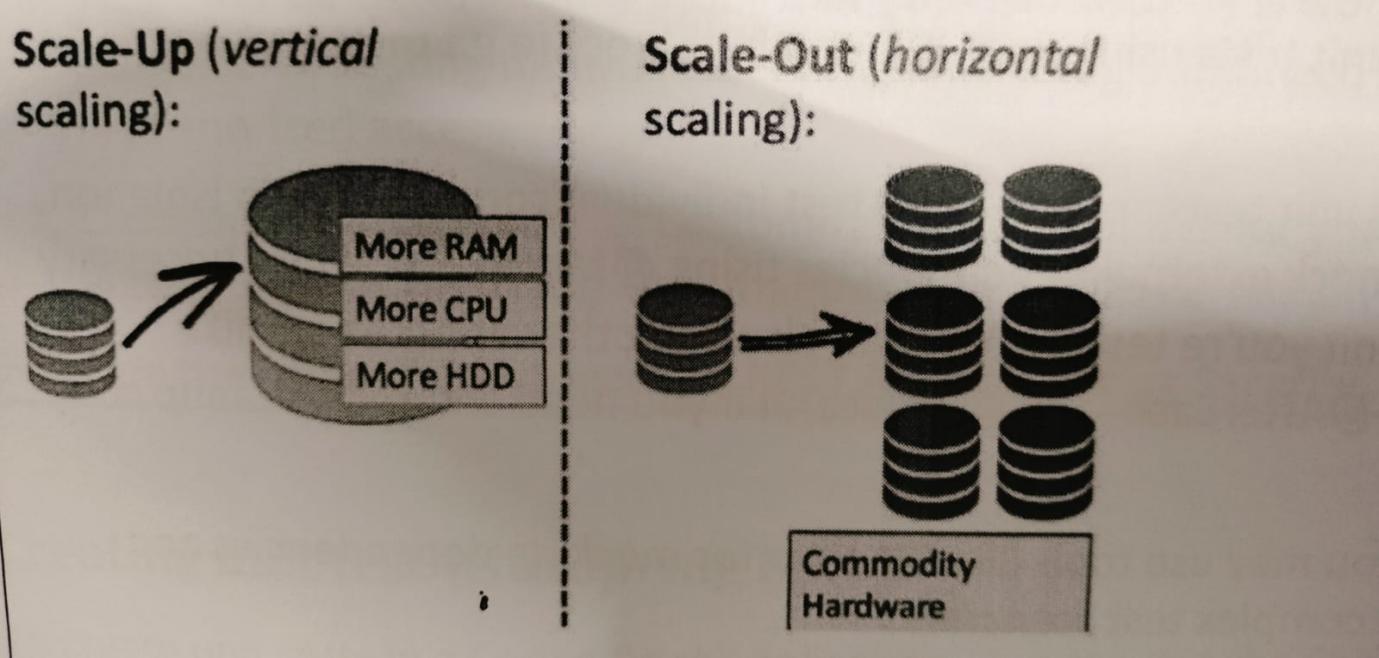
**No SQL Introduction**

What is NoSQL Database ?

NoSQL stands for Not only SQL . NoSQL Database are also known as non-relational database that don’t require a fixed schema . User can create documents with flexible schema and can scale up evenly without much of stretch.

Internet giants like Facebook, Google, Amazon use NoSQL database heavily to deal with Terabytes of data daily.

As NoSQL Database uses distributed storage, it is easy scale them up horizontally with commodity hardware.



**MongoDB Overview**

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability and easy scalability. MongoDB works on concept of collection and document.

**Database**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB sever typically has multiple database.

**Collection**

Collection is a group of MongoDB document. It is the equivalent of ana RDBMS table. A collection exists within a single database.

Here are some key features and concepts associated with MongoDB:

* **Document-Oriented:**  MongoDB is a document Database, which means it stores data in semi-structured BSON (Binary JSON) documents.
* **Collection:** Collection in MongoDB are analogous to tables in relational database. Each collection can contain multiple documents and documents within a collection do not have the same structure.
* **JSON Like Document:** MongoDB documents are represented ina JSON-like format, which makes it easy to work with data in many programming languages.
* **Schema Flexibility:** MongoDB offers schema flexibility, allowing you to add, remove, modify fields in a document without affecting other documents in the same collection.
* **Indexing:** MongoDB supports indexing, which can greatly improve query performance. You can create indexes on specific fields to speed up data retrival.
* **Query Language:** MongoDB uses a query language for querying data. You can preform complex queries filtering, storing, and aggregation operations using the MongoDB query languages.
* **Aggregation Transcation:** MongoDB provides a powerful aggregation framework for data analysis and transformation, allowing oyu to perform operations.
* **ACID Transaction:** MongoDB introduced support for multi-document ACID transaction in recent versions, ensuring data consistency and integrity for critical operations.
* **Geospatial Queries:** MongoDB supports geospatial queries to find data based on location.
* **Security:** MongoDB offers security feature such as authentication, role-based access control, encryption at rest, and auditing to protect your data.
* **Community and Enterprise Editions:** MongoDB is available in both community and free to use, while the enterprise edition offers additional features and support for large-scale deployments.

CURD operation in MongoDB to the basic for managing data in a MongoDB database. CURD stands for Create, Read, Update, Delete and these operation are fundamental for interaction with data in any database system, including MongoDB.

1. **Create(c) :**  To create a new document or record in a collection, you can use the insertOne or insertMany methods. For ex:

db.collection.inserrtOne({ key:”value”});

1. **Read(R):** Find” To retrive documents from collection, you can use the find method. For ex: db.collectionNme.find()

**Find One:**  To retrieve documents from a collection, you can use the findOnemethod. For ex: db.collectionNmae.findOne()

1. **Update(U):**  To modify existing document, you can use the updateOne or updateManymethods. For ex: db.collectionNmae.updateOne({ key:”value”},{$set:{newKey:”newValue”}});

**Repalce:** To replace an entire documents, you can use the replaceOne method. For ex: db.collectionName.replaceOne({ key:”value”},{newKey:”newValue”}});

1. **Delete(D):** To remove documents from a collection, you can use the deleteOne or deletemany methods. For ex: db.collectionName.deleteOne({ Key:”value”});

These are the basic CRUD operations in MongoDB uses a flexible schema called BSON, so each document in a collection can have

Different fields and data.

**MongoDB Example** The below example shows how a document can be modeled in MongoDB.

1. The\_id field is added by MongoDB to uniquely identify the document in the collection.

2. What you can note is that the Order Data (OrderID, Product, and Quantity) which in RDBMS normally be stored in a separate table, while in MongoDB it is actually stored as an ember

document in the collection itself. This is one of the key differences in how data is modele MongoDB.

{

\_id: <ObjectId>,

CustomerName: Guru99

Order:

{

Order ID: 111

Product: ProductA

Quantity: 5

१

}

}

Example of how data can be embedded in a document

MongoDB Shell (mongosh)

The MongoDB Shell, mongosh, is a fully functional JavaScript and Node.js 16.x REPL environ interacting with MongoDB deployments. You can use the MongoDB Shell to test queries and on directly with your database.

public Response Entity-Tutorial update Tutorial ( Path Variable("id") String il, @RequestBody Tutorial totocini)

1

@Delete Mapping("/tutorials/[id]")

public Response limity<HitpStatus> delete Tutorial(@PathVariable("id") String id) (

1

Delete Mapping("/tutorials")

pablic Response intity<HttpStatus>> deleteAllTutorials() (

1

GetMapping("/tutorials/published")

public Response EntityList<Tutorial>> findByPublished() [

@CrossOrigin is for configuring allowed ongins @RestController annotation is used to define a controller and to indicate that the return value of the methods should be be bound the web response body. @RequestMapping("/api") declares that all Apis url in the controller will start with/api We use @Autowired to inject TutorialRepository bean to local variable.

Now I will show you how to implement each controller's CRUD methods

Create Operation

We use @PostMapping annotation for handling A new Tutorial will be created by MongoRepository.save() method. POST HTTP requests.

PostMapping("/tutorials")

public Response Entity<Tutorial> create Tutorial(@RequestBody Tutorial tutorial) (

try { Tutorial tutorial tutorial Repository.save(new Tutorial (tutorial.getTitle(), tutorial.getDescription(). false));

return new ResponseEntity tutorial, HttpStatus.CREATED);

catch (Exception e) (

return new ResponseEntity(null, HttpStatus. INTERNAL SERVER ERROR);

Retrieve Operations

We use @GetMapping annotation. for handling HTTP GET requests, Repository's findAll(), findByTitleContaining(title), findByPublished() method to get the result. then

getAllTutorials(): returns List of Tutorials, if there is title parameter, it returns a List in that each Tutorial contains the title

get TutorialById(): returns Tutorial by given id

findByPublished(); return published Tutorials

@GetMapping("/tutorials")

public Response Entity<List<Tutorial>> getAllTutorials(@RequestParam(required = false) String title) (

try

List<Tutorial> tutorials = new ArrayList<Tutorial>();

if (title= null)

tutorial Repository.findAll().forEach(tutorials::add);

else

tutorialRepository.findByTitleContaining(title).forEach(tutorials::add);

if (tutorials.isEmpty()) (

return new Response Entity(HttpStatus.NO\_CONTENT);

return new Response Entity(tutorials, HttpStatus.OK); } catch (Exception e) {

return new Response Entity(null, HttpStatus.INTERNAL\_SERVER\_ERROR);

1

@GetMapping("/tutorials/(id)")..

public Response Entity<Tutorial> getTutorialById(@Path Variable("id") String id) {

Optional<Tutorial> tutorial Data = tutorialRepository.findById(id);

if (tutorialData.isPresent()) (

return new Response Entity(tutorialData.get(), HttpStatus.OK);

} else {

return new ResponseEntity>(HttpStatus.NOT\_FOUND);

@GetMapping("/tutorials/published")

public Response Entity<List<Tutorial>> findByPublished() { try {

List<Tutorial> tutorials tutorial Repository.findByPublished(true); if (tutorials.isEmpty()) (

return new Response Entity (HttpStatus.NO\_CONTENT);

return new Response Entity (tutorials, HttpStatus.OK); } catch (Exception e) {

return new Response Entity (HttpStatus INTERNAL SERVER\_ERROR);

}

Update Operation

@PutMapping will

help

us

handle

PUT

HTTP

requests.

- update Tutorial() receives id and from the id, we get Tutorial database payload. using findById() method. the Tutorial from

- then we use the payload and save() method for updating the Tutorial.

@PutMapping("/tutorials/(id)")

public Response Entity<Tutorial> updateTutorial(@Path Variable("id") String id, @RequestBody Tutorial tutorial) (

Optional<Tutorial> tutorialData = tutorialRepository findById(id);

if (tutorial Data. is Present())

Tutorial tutorial tutorialData.get(), tutorial.setTitle(tutorial.getTitle());

tutorial.setDescription(tutorial.getDescription());

tutorial.setPublished(tutorial.isPublished());

return new Response Entity(tutorialRepository.save(\_tutorial), HttpStatus.OK); | else |

return new Response Entity (HttpStatus.NOT\_FOUND);

1

Delete Operation

We use @DeleteMapping for There are 2 methods:

DELETE

HTTP

requests

delete Tutorial(): delete a Tutorial document with given id delete AllTutorials(); remove all documents in tutorials collection

The operations is done with the help of MongoRepository's deleteById() and deleteAll() method.

@DeleteMapping("/tutorials/{id}")

public Response Entity<HttpStatus> delete Tutorial(@PathVariable("id") String id) {

try {

tutorial Repository.deleteById(id);

return new ResponseEntity (HttpStatus.NO\_CONTENT);

catch (Exception e) {

return new Response Entity(HttpStatus. INTERNAL SERVER\_ERROR);

1

@DeleteMapping("/tutorials")

public Response Entity<HttpStatus> deleteAllTutorials() (

try (

tutorialRepository.deleteAll():

return new ResponseEntity (HttpStatus.NO\_CONTENT); } catch (Exception e) {

return new Response Entity(HttpStatus. INTERNAL SERVER\_ERROR);

}

}

**Week -10**

**Manual testing**

* Testing any software or an application according to the client needs without using any automation tool is known as manual testing. Manual testing is used to verify the behavior of an application or software in contradiction of requirements specification.
* We do not require any precise knowledge of any testing tool to execute the manual test cases. We can easily prepare the test document while performing manual testing on any application.

**Automation testing**

Automation testing is used with some specific tools to execute the test scripts without any human interference. In automation testing, the test automation engineer will write the test script or use the automation testing tools to execute the application. In test automation, the test engineer can execute repetitive task and other related task. In manual testing, it is a tedious process to implement the repetitive task again and again.

**Types of automation testing**

**Functional testing**

Functional testing assesses the software against the set functional requirement/specifications. It focuses on what the application does and mainly involves black box testing.

**Unit testing**

Unit testing involves running tests on individual components or functions in isolation to verify that they are working as required. It is typically done in the development phase of application and is therefore often the first type of automated testing done on application.

**Integration testing**

Integration testing involves testing all the various unit of the application in unity. It focuses on evaluating weather the system as a whole complies with the functional requirements set for it.

**Smoke testing**

Also known as “Build Verification testing “ and “confidence testing”. Smoke testing focuses on checking weather the major aspects of the application are the functional or not and is used to verify whether the application should be shelved due to lack of functionality or continued to the other phases of testing.

**Non-functional testing**

This testing encompasses testing all the various non functional elements of an application such as performance, reliability, usability, etc..,

**Performance testing**

Performance testing involves testing a software speed, stability, responsiveness under work load. The main purpose of performance testing is to identify any possible performance hurdles and have them removed to optimize software’s ability to deliver the best result to the end user.

**Regression testing**

Regression testing is sometimes called the testing type that fails under the maintenance testing category. Regression testing involves re-running of functional and non functional test to see if the software is performing the same way as before after some kind of code or program change has been made. If the performance is not the same, then the software is said to have regressed

**Week -11**

**Containers**

* Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a developer’s laptop to test environment, from a staging environment into production, and from a physical machine in a datacenter to a virtual machine in a private or public cloud.
* Containers are packages of software that contains all of the necessary elements to run in any environment. In this way, containers visualize the operating system and run anywhere, from a private data center to the public cloud or even on a developer personal laptop.
* A container is a standard unit of software. It packages up the code along with all the dependencies of the code. This packaging enables the application to run quickly and reliably from one computing element to another

Why containers?

* Problems arise when the supporting software environment is not identical, “you’re going to test using python 2.7, and then it’s going to run on python 3 in production and something weird will happen. Or you’ll rely on the behavior of a certain version of an SSL library and another one will be installed.
* You’ll run your tests on debain and production on red hat and all sorts of weird things happen. ”and it’s not just different software that can cause problems. ”the network topology might be different, or the security policies and storage might be different, be the software might be run on it.”
* A container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration flies needed to run it, bundled into one package. By containerizing the application platform and its dependencies, differences in OS distributions and underlying infrastructure are abstracted away.

**What is a Docker?**

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

**Components of Docker/how docker works?**

1. Docker container
2. Docker client
3. Docker daemon
4. Docker image
5. Docker registry

**Week-12**

**Container orchestration:**

Definition: Orchestration is the coordination and arrangement of different components or services to achieve a unified workflow or goal.

1. In Computing: It involves managing and automating the interactions and dependencies between various software components in a system.

2. Workflow Control: Orchestration ensures that tasks are executed in a specific order to achieve a desired outcome.

3. Complex Systems: Particularly relevant in distributed systems, cloud computing, and microservices architectures.

Orchestration Engine:

Definition: An orchestration engine is a software component responsible for coordinating and managing the execution of tasks in an orchestrated system.

1. Dependency Management: Manages dependencies between tasks to avoid race conditions and ensure proper order of execution.

2. State Management: Keeps track of the state of the orchestrated system and manages transitions between states.

3. Error Handling: Handles exceptions and errors gracefully, allowing for proper

recovery and fault tolerance.

4. Scalability: Should be capable of scaling with the complexity and size of orchestrated systems.

5. Example: Apache Airflow, Kubernetes (as a container orchestration engine), Docker Compose.

Orchestration Tools:

1. Kubernetes:

Widely used for container orchestration, managing the deployment, scaling, and operation of application containers.

Uses a declarative configuration to define the desired state of the system.

2. Docker Swarm:

Docker's native orchestration tool for managing a cluster of Docker hosts.

Simplicity in setup and integration with Docker.

3. Apache Airflow:

Designed for orchestrating complex workflows and data processing pipelines.

Utilizes Directed Acyclic Graphs (DAGs) to represent and execute workflows.

Introduction to Kubernetes:

Definition: Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications.

Why Kubernetes:

1. Container Orchestration: Manages the complexity of deploying and scaling containerized applications.

2. Portability: Offers a consistent environment across different infrastructure providers and on-premises environments.

3. Scalability: Scales applications horizontally by adding or removing containers based on demand.

4. Fault Tolerance: Ensures high availability by automatically recovering from container or node failures.

5. Resource Efficiency: Optimizes resource utilization through efficient allocation and scaling.

Kubernetes Configuration:

Deployment:

1. Definition: Describes the desired state for a set of pods.

2. ReplicaSets: Ensures a specified number of pod replicas are running at all times.

3. Rolling Updates: Facilitates updating applications with minimal downtime.

**Service:**

1. Definition: Provides a stable endpoint for accessing a set of pods.

2. Types: ClusterIP (internal cluster-only), NodePort (exposes on each node's IP), Load Balancer (provisions an external load balancer), ExternalName (maps to external services).

\* Load Balancer/Ingress:

Load Balancer:

Definition: Distributes incoming network traffic across multiple nodes.

Integration: Often used in conjunction with a Service of type LoadBalancer.

Cloud Providers: Automatically provisions load balancers (e.g., AWS ELB, GCP Load Balancer).

Deployment Strategies:

Blue-Green Deployment:

Overview: Blue-Green Deployment is a strategy that involves maintaining two separate environments-blue and green-and switching traffic between them.

Workflow:

1. Initial State (Blue): Current live environment serving production traffic.

ii. Deployment (Green): New version deployed in the inactive environment.

III. Switch (Traffic Routing): Traffic redirected from the blue environment to the green one.

iv. Validation: New version validated with live traffic in the green environment.

v. Rollback: If issues are detected, rollback involves switching traffic back to the blue environment.

Benefits:

Zero Downtime: Users experience no downtime during deployment

Quick Rollback: Instantly revert to the previous version in case of problems.

Safe Testing: Allows thorough testing with live data before full release.

Canary Deployment:

Overview: Canary Deployment is a gradual release strategy where a new version is rolled out to a subset of users before a full release.

Workflow:

1. Initial State (Baseline): Current live environment serving all users.

II. Canary Release (Subset): New version deployed to a small subset of users.

iii. Monitoring: Monitor the canary release for errors, performance, and other issues. iv. Expansion: If successful, gradually expand the release to a larger audience.

v. Full Rollout: Continue until the new version is deployed to all users.

Benefits:

LRisk Mitigation: Umits the impact of potential issues to a subset of users initially.

il. Real-Time Feedback: Gathers real-time feedback from actual users during the release.

iii. Gradual Rollout: Minimizes the risk of widespread problems affecting all users.

Disaster Recovery:

Definition: Disaster recovery involves planning and implementing strategies to restore and continue essential business operations after a disruptive event.

Types:

Data Backup and Restore: Regularly backing up data and restoring it in case of data loss or system failure.

Cloud-Based Disaster Recovery: Replicating data and applications to a cloud environment for rapid recovery.

Cold, Warm, Hot Sites: Varying levels of infrastructure preparedness for recovery (Cold: minimal, Warm: partial, Hot: fully operational),

Disaster Recovery Planc

1. Identify Critical Systems: Determine which systems and data are crucial for business continuity.

2. Risk Assessment: Analyze potential risks and their Impact on business. operations.

3. Set Objectives: Define Recovery Time Objective (RTO) and Recovery Paint Objective (RPO).

4. Backup Strategy: Establish regular backup schedules, including offsite storage.

5. Communication Plan: Develop a clear communication plan for internal and external stakeholders.

**Week 13**

a) IRCTC-A Train ticket booking application that helps the users to book a train ticket for travelling across India

User Story: Registration/Sign-up: As a new user I want to sign up for the application through a sign up furm, so that I can access the train booking app.

Acceptance Criteria:

1)While signing up, enter Username, Email, Password & Confirm Password, Security question and Address,

2)If user sign up with an incorrect detail, user receives an error message for incorrect information.

3)if user tries to sign up with an existing email address, existing mobile number, user receives error messages saying "email exists", "moolle registered already".

4] if sign up is successful, a confirmation emall is sent to user for mobile & email verification. After successful verification user can login Into app with credentials.

•User Story: Login As a registered/authorized user. I want to login into IRCTC application so that I can access variety of service provide by the app.

Acceptance Criteria:

1) Username, password and captcha are required for user login.

2) If we are trying to login with incorrect username or password, then error message will be displayed as "bad credentia's

3) After successful log in, home page is displayed

User Story: Search Train As an authorized user, I want to search traîn so that i can book ticket for my planned trip.

Acceptance Criteria:

1) User has to enter valid from(source), toldestination) train stations, valid date.

2)A valid search displays list of trains with schedule

3)User can sort, fliter and modify the search results.

4) User can view available tickets in each travel class like AC, Sleeper etc.

User Story: Book Train As an authorized user. I want to book train so that can make reservation for particular place, date and time.

Acceptance Criteria:

1)User has to choose available train & travel class for ticket booking.

2) Passenger's details like name, age, gender, berth preference need to be entered.

3) Valid contact information, travel insurance option & payment mode are required.

4) After successful payment, user should get the cooking detalls to registered mobile number and E-mall id.

•User Stories: Logout As a logged in user, I want to log out of IRCTC app so that I can prevent unauthorized access of my profile.

Acceptance Criteria:

1)When user Ings out of his account by clicking log out button, logged out message should appear and app has to redirected to the log-in page.

2) If user session expires due to internet fallure or system crash, then user has to be logged out the application,

\* test cases for the above application.

Test Cases for the Registration Page:

1) Verify that the registration page is accessible from the website's homepage and louds comectly for desktop 7 of 23mins.

2) Check that the system Validates the user Information such as email address, phone number, security questions, address and password complexity,

3)Ensure that the system does not allow c'uplicate email addresses or phone numbers.

Test Cases for the Login Page:

1) Verify that the login page loads correctly and is accessible from the website's homepage.

2)Check that the login credentials are case sensitive and the appropriate message is displayed If the user enters incorrect information.

3)Verify that the "Forgot Password" option works as intended, allowing users to reset their password in case they forget it

4) Ensure that the system limits the number of unsuccessful logins attempt to prevent brute-force attacks

Test Cases for the Train Search:

1) Ensure that the train search page displays list of trains corresponding to entered source and destination station for the specific date

2)Verify whether user is able to apply filter, sort trains and modify existing search.

3) Verify whether user is able to select the required train, check for availability of tickets.

Test Cases for the Train Booking:

1) Ensure whether user is able to add passenger detalls for selected train.

2) Verify the upper limit for number of passengers booked per train.

3)Verify that the system displays the total cost of the ticket purchase as per booking details, including any taxes and fees.

4) Verify whether user is able to cancel the ticket or partially cancel the tickets or not.

5)Verify whether booking confirmation is received by user or not.

Test Cases for the Payment Gateway:

1) Verify that the payment gateway is secure & encrypts user information to prevent fraud.

2) Ensure that the system accepts multiple payment options, such as credit/debit cards, GPay. PhonePay and mobile wallets.

3) Ensure that the payment gateway sends a confirmation email or SMS to the user after the successful transaction.

\*project:

Creating a project on IRCTC (Indian Railway Catering and Tourism Corporation) would typically involve multiple components, such as user authentication, ticket booking, payment processing, and possibly even additional features like seat selection, cancellations, and user account management Below is a simplified outline of a potential project structure:

Project Structure: User Authentication:

Allow users to create accounts and log in securely.

Implement password recovery mechanisms.

Train Search and Display:

Fetch and display a list of trains based on source, destination, date, and other preferences.

Implement filters and sorting options for user convenience.

Ticket Booking:

Allow users to select a train and book tickets for a specific date and class Implement logic to calculate and display the total fare.

Provide confirmation with a unique booking reference number.

Payment Gateway Integration: Integrate a secure payment gateway for online transactions

Handle payment failures and provide appropriate feedback to the user.

**E-ticket Generation:**

Generate an electronic ticket (PDF format) with all the relevant details

Provide a download link to the user.

Booking History:

Maintain a record of the user's booking history.

Allow users to view and print previous bookings.

Seat Selection (Optional): If

applicable, implement a feature for users to select their preferred seats. Ensure that seat selection integrates seamlessly with the booking process.

Cancellation:

Allow users to cancel their booked tickets.

Implement a refund mechanism if applicable.

User Profile Management:

Allow users to update their profiles, including contact information and preferences.

Implement security measures to protect user data.

Notification System:

Implement a notification system for booking confirmations, cancellations, and other

relevant updates.

Create a dashboard for administrators to manage trains, bookings, and user accounts.

Admin Dashboard:

implement data analytics for insights into user behavior and popular routes.

Technologies:

Frontend:

HTML, CSS, JavaScript

Frontend framework/library (eg, React, Angular, Vue)

Backend:

Server-side scripting (eg, Node.js, Django, Flask)

Database (e.g., MySQL, PostgreSQL) Authentication:

JWT (JSON Web Tokens), OAuth

Payment Gateway:

Integration with a payment gateway (eg, Stripe, PayPal)

PDF Generation:

Libraries like jsPDF, Puppeteer (Node.js]

Version Control:

Git

Deployment:

Cloud platform (e.g. AWS. Heroku)

Testing:

Test individual functions and components

Unit Testing:

Integration Testing:

Ensure that different modules work seamlessly together.

End-to-End Testing:

Simulate user interactions to test the entire booking flow.

Security Testing:

Identify and address potential vulnerabilities

Project Management: Agile Methodology

Divide the project into sprints with specific goals

Issue Tracking:

Use tools like flex or GitHub Issues for tracking tasks and imnes.

Conclusion:

This is a high-level outline, and the actual impimnentation may vary based on specific requirements, budget, and time constraints. Additionally, it's crucial to adhere to security best practites and comply with relevant regulations, rapecially when dealing with sensitive user information and online trarcactiona

A simple flowchart: