1. **How to create project plan and product backlog for project and User story creation.**

- Open browser, search for Jira Login.
- Continue with your Gmail account or login to Jira.
- Click on Jira software and select project from top menu bar then select create project from dropdown Menu.
- Select Scrum click on template and click on create.
- Give a name to your project and Give a Description if you want.
- Click on create.
- Select issues from top menu bar and select issue type. This will be default setting.
- Give a summary to your project.
- Now write a user story in Description box.
- Your story will then go into the backlog to be Assigned and auctioned by the project manager, product owner or other relevant stakeholders and click on start sprint
- Click on Board and select Insights
- Click on Insights and click “Sprint burn down”
- And click on Learn more.

2. Create and manage product backlog using appropriate tool like Jira

Summary: Customer registration functionality

Description

AS A customer

I WANT to have registration functionality

SO THAT I can successfully resist

Scope

- build a registration page
- customer validation
- customer should be able to change the phone number
- it should work in all the browser
- it should also work in mobile

Pre-condition

- customer should have email and phone number

Acceptance criteria

Scenario 1: customer can successfully resister

- “Given” I am on registration page
- “And” I give valid customer name and phone number
- “And” I check on sing in
- “Then” I will successfully resister

Scenario 2: customer cannot successfully resister

- “Given” I am on registration page
- “And” I give invalid customer name and phone number
- “Then” I will get a error message as “registration failed incorrect customer name”

Summary: Customer checking availability

Description

AS A customer

I WANT to have checking available of hall

SO THAT I can check the available halls

Scope

- build an available checking page
- it should be only inside the Karnataka
- customer should be able to check the available halls in their particular location

pre-condition

- customer should have nearest halls in their location

Acceptance criteria

Scenario 1: Customers can successfully check availability of hall in their location

- "Given " I am on check available of hall page
- "And" I give particular location and date

Scenario 2: customer can't successfully check availability of hall in their location

- "Given " I am on check available of hall page
- "And" I give wrong location
- "Then" I will get the error message as in valid location

Summary: Customer booking hall

Description

AS A customer

I WANT to booking hall

SO THAT i can book the hall

Scope

- build a booking hall page
- customer should be able to change the date and location

Pre-condition

- customer should be able to book the hall in their particular date

Acceptance criteria

Scenario 1: customer can successfully booking hall

- "Given" I am on booking page
- "And" I give available date time

- “And” I will book the hall
- “Then” I successfully booked the hall

Scenario 2: customer can’t successfully booking hall

- “Given” I am on booking page
- “And” I give invalid date and time
- “Then “I will get the error messages as their hall is already booked

Summary: Customer booking details

Description:

AS A customer

I WANT to block the hall

SO THAT I can get the booking details

Scope

- build a booking details page
- it should be able to see after the booking also
- customer should be able to change details if their want

Pre-condition

- customer have to fill the every information given in the booking details

Acceptance criteria

Scenario 1: customer can successfully get the booking details

- “Given” I am on the booing details page
- “And” I fill the details
- “And” I have also blocked the hall
- “Then” I will successfully get the booing details

Scenario 2: customer will not get the booking details

- “Given” I am on the booking details page
- “And” I will fill the details without blocking hall
- “Then” I will get a error message as the hall is not blocked yet

3. Create Sprint 1 with required user stories

Note: Create user story for required topic and follow the steps below.

- Give a summary to your project.
- Now write a user story in Description box.
- Your story will then go into the backlog to be Assigned and auctioned by the project manager, product owner or other relevant stakeholders and click on start sprint
- Click on Board and select Insights
- Click on Insights and click "Sprint burn down" And click on Learn more.

4. Create UI/UX design - for created user stories in JIRA.

- Understand User Stories
- Create wireframes to outline the basic structure and layout of the interface.
- Use tools like Sketch, Figma, Adobe XD, or even paper and pencil to sketch your initial ideas.
- Choose a color scheme and typography that aligns with your brand or project.
- Use tools like InVision, Marvel, or Figma to create clickable prototypes.
- Share your designs with stakeholders and team members.
- Collect feedback and iterate on the designs accordingly.
- Once the design is finalized, implement the changes in JIRA.
- Update the user stories with relevant design assets and information.

5. Create repository – named mini project-1 Push and pull operation in GitHub.

- Browse to the official Git website: https://git-scm.com/downloads
- Click the download link for Windows and allow the download to complete.
- Double-click the file to extract and launch the installer.

Git operations

- **Creating a repository**
- Open browser, search for GitHub Login.
- Sign in with your username and password
- In the upper-right corner, use the drop-down menu, and select **New repository**.
- Give a name for your repository. For example, "hello-world".
- Add a description of your repository. For example, "Mini Project I"
- Click **Create repository**.

Push Operation:

- Go to add files and select upload files.
- Choose your files then select a file or folder click on open.
- Click on commit changes.

Clone or pull operation:

- Click on code dropdown button
- Click on Download Zip

6. Create a form like registration form or feedback form, after submit hide create form and enable the display section using java script.

Registration.html

```
<html>
<head>
    <title> Registration Form</title>
    <script>
      function passvalues()
      {
            var name = document.getElementById("name").value;
              var email = document.getElementById("email").value;
              var address = document.getElementById("address").value;
              localStorage.setItem("name",name);
              localStorage.setItem("email",email);
              localStorage.setItem("address",address);
              return;
      }
    </script>
  </head>
  <body>
<h1>Registrtion Form</h1>
    <form action="Details.html">
<fieldset>
  <legend>Registration</legend>
 <label> Name </label>
    <input type="text" id="name"/><br><br>
 <label> Email ID </label>
    <input type="email" id="email"/><br><br>

 <label> Address </label>
    <input type="address" id="address"/><br><br>
    <input type="submit" value="submit" onclick="passvalues()"/>
</fieldset>
    </form>
  </body>
</html>
```

Details.html

```
<html>
    <head>
        <title> Details</title>
    </head>
    <body>
<form>
    Your Name is:<p id="name"></p><br>
        Your email is:<p id="email"></p><br>
        Your address is:<p id="address"></p>
<script>
      document.getElementById("name").innerHTML = localStorage.getItem("name");
      document.getElementById("email").innerHTML = localStorage.getItem("email");
      document.getElementById("address").innerHTML = localStorage.getItem("address");
        </script>
</form>
    </body>
</html>
```

7. Create form validation using JavaScript

Index.html

```
<html>
<body>
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;
if (name==null || name=="")
{
  alert("Name can't be blank");
  return false;
}
else if(password.length<6)
{
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
<form name="myform" method="post" action="valid.html" onsubmit="return
validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
</body>
</html>
```

valid.html

```
<html>
<body>
 <h1>Validation Successfull</h1>
</body>
</html>
```

8. Create and run simple program in TypeScript

Install TypeScript using Node.js Package Manager (npm)

Step-1 Install Node.js. It is used to setup TypeScript on our local computer.

To install Node.js on Windows, go to the following link: **https://www.javatpoint.com/install-nodejs**

Step-2 Install TypeScript. To install TypeScript, enter the following command in the Terminal Window.

- npm install typescript --save-dev //As dev dependency
- npm install typescript -g //Install as a global module

  or
- npm install -g typescript
- npm install typescript@latest -g //Install latest if you have an older version

Step-3 To verify the installation was successful, enter the command **$ tsc -v** in the Terminal Window.

Install Live server

```
npm install -g live-server
```

Create and run first program in TypeScript

- open command prompt
- go to d: drive(any drive)
- d:\>mkdir typescript
- d:\>cd typescript
- d:\typescript> npm install typescript --save-dev
- open visual studio code
- file-open folder-choose typescript folder from d:
- create new file- save it as types.ts(any name.ts)
- Write the below code and save it

- console.log("Hello World");
- go to command prompt and compile the program
- tsc types.ts
- run the program
- node types.js
- Observe the output

9. Forms - Use of HTML tags in forms like select, input, file, textarea, etc.

```
<html>
<head>
<title>Form Elements</title>
</head>
<body>
<form>
<lable>Text Box</lable>
<input type="text" id="t1" name="name" value=""/><br><br>

Radio Button: <br>
<input type="radio" id="r1" name="" value=""/>Male<br> <br>
<input type="radio" id="r1" name="" value=""/>FeMale<br><br>
Check Box:<input type="checkbox" id="c1" name="" value=""/><br><br>
File:<input type="file" id="e1" name="file" value=""/><br><br>

Select:<br>
<label>Sem</label>
<select name="sem" id="sem">
  <option value="1">1 Sem</option>
  <option value="2">2 Sem</option>
</select><br><br>

Text Area:<br>
<textarea id="ta1" name="textarea" rows="4" cols="50">
At w3schools.com you will learn how to make a website.
</textarea><br><br>



<fieldset>
    <legend>Personal Details:</legend>
    <label>First name:</label>
    <input type="text" id="fname" name="fname"><br><br>
    <label>Last name:</label>
    <input type="text" id="lname" name="lname"><br><br>
  </fieldset><br><br>
Button:<input type="button" id="t1" name="" value="Submit"/><br>
</form>
</body>
</html>
```

10. Testing single page application (Registration form) using React.

Note: Add Home.js file in index.js file

Index.js

```
   <Home />
```

Home.js

```
import { useState } from 'react';
import './App.css';
export default function Form()
{
// States for registration
const [name, setName] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [submitted, setSubmitted] = useState(false);

const handleName = (e) => {
   setName(e.target.value);
};

const handleEmail = (e) => {
   setEmail(e.target.value);
};

const handlePassword = (e) => {
   setPassword(e.target.value);
};

const handleSubmit = (e) => {
   e.preventDefault();
   if (name === '' || email === '' || password === '') {
   alert("Please enter all the fields");
   } else {
   setSubmitted(true);
   }
};
// Showing success message
const successMessage = () => {
   if(submitted)
   return (
   <div className="success" >
      <h1>User {name} successfully registered!!</h1>
   </div>
   );
};
return (
   <div className="form">
   <div>
      <h1>User Registration</h1>
   </div>
   {/* Calling to the methods */}
   <div className="messages">
      {successMessage()}
   </div>

   <form>
   <fieldset>
      {/* Labels and inputs for form data */}
      <label className="label">Name</label>
      <input onChange={handleName} className="input" value={name} type="text"
/><br></br>
      <label className="label">Email</label>
      <input onChange={handleEmail} className="input" value={email} type="email"
/><br></br>
      <label className="label">Password</label>
      <input onChange={handlePassword} className="input" value={password}
type="password" /><br></br>
      <button onClick={handleSubmit} className="btn" type="submit">
      Submit
      </button>
      </fieldset>
   </form>
   </div>
);
}
```

App.css

```css
.input {
  width: 30%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
```

11. Implement navigation using react router

Add React Router

- To add React Router in your application, run this in the terminal from the root directory of the application:

```
npm i -D react-router-dom
```

Index.js

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";
export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Create a folder name called pages. Within a pages create following files.

<u>Blogs.js</u>

```
const Blogs = () => {
    return <h1>Blog Articles</h1>;
  };
    export default Blogs;
```

<u>Contact.js</u>

```
const Contact = () => {
    return <h1>Contact Me</h1>;
  };
  export default Contact;
```

<u>Home.js</u>

```
const Home = () => {
    return <h1>Home</h1>;
  };
  export default Home;
```

<u>Layout.js</u>

```
import { Outlet, Link } from "react-router-dom";
const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>
      <Outlet />
    </>
  )
};export default Layout;
```

NoPage.js

```
const NoPage = () => {
    return <h1>404</h1>;
  };
    export default NoPage;
```

App.css

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #04AA6D;
}

li {
  float: left;
  border-right:1px solid #bbb;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover:not(.active) {
  background-color: #111;
}
```

12. Build single page application (Add Product to Product List)

Step 1 : Create React App

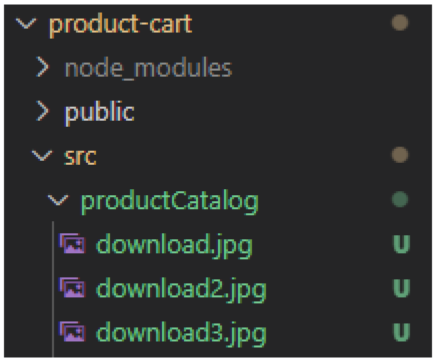
Run the command

- ➢ npx create-react-app **product-cart**

inside product-cart/src, create a new folder **productCatalog**

Step 2 : Download and import images.

Download any 3 images of your choice and name them as download.jpg, download2.jpg and download3.jpg and move/copy them to src/productCatalog

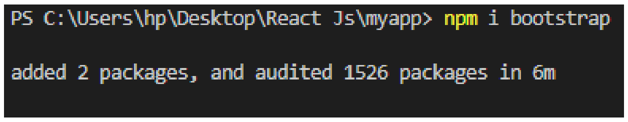


```
product-cart
  > node_modules
  > public
  src
    productCatalog
      download.jpg      U
      download2.jpg     U
      download3.jpg     U
```

Step 3: Install Bootstrap for styling

Run this command to download bootstrap library in your react application

- ➢ npm i bootstrap



```
PS C:\Users\hp\Desktop\React Js\myapp> npm i bootstrap

added 2 packages, and audited 1526 packages in 6m
```

Step 4: Create a file **Products.js** in the productCatalog folder

<u>Products.js</u>

```
import React, { useEffect, useState } from 'react'
import "../../node_modules/bootstrap/dist/css/bootstrap.min.css" import "./index.css"
import iphone from"./download.jpg" import
ipad from"./download2.jpg" import laptop
from"./download3.jpg"

const Products = () => { const
        productList = [
            {name:"Apple" , price:249 , img:iphone},
            {name:"Microsoft" , price:99, img:laptop},
            {name:"iPad" , price:125, img:ipad}
        ]
        const [cartList, setCartList] = useState([{name:"Apple" , price:249, img:iphone}])
        const [totalPrice, setTotalPrice] = useState(getPrice())

        function getPrice(){ let i =0 ;
            cartList.forEach((item) => { i = i +
                item.price

            })
            return i
        }

        useEffect(()=> { setTotalPrice(getPrice())
        }, [cartList])

        function addItem(index){
                let tempCartItem = productList.filter((item,i) => i === index ) setCartList(
                cartList.concat(tempCartItem))
        }


        function deleteItem(index){
                let newList = cartList.filter((item,i) => i !== index) setCartList(newList)
        }
    return (
        <div className='container conatiner-fluid border mt-5'>
            <h1 className='text-center'>Products</h1>
            <ol className='list-group m-3'>
                {
```

```
          productList.map((item,index)=> { return (
                    <li key={index}
                    className="list-group-item  d-flex"
                    >
                    <p>{item.name} &nbsp; ${item.price}</p>


                    <img src={item.img}/>


                    <button onClick={()=> {
                         addItem(index)
                    }}
                    className="btn  btn-success"
                    >+</button>
                    </li>
               )
          })
     }
   </ol>
```

```
            <h4>Your Cart</h4>
          <ul className='list-group m-2'>
            {
                cartList.map((item,index) => {
                    console.log(item)
                    return (
                        <li key={index}
                        className="list-group-item d-flex "
                        >
                        <p>{item.name} &nbsp; ${item.price} </p>
                        <img src={item.img}/>
                        <button
                        className='btn btn-danger' onClick={()=>
                        deleteItem(index)}
                        >-</button>
                        </li>
                    )
                })
            }
          </ul>

          <p> Total Price: <b>${totalPrice}</b></p>

          <button className='btn btn-warning' onClick={()=> {
            alert("Total Price is $" + totalPrice + " for " + cartList.length + "
items")
          }}><b>Proceed</b> </button>
        </div>
    )
}

export default Products
```

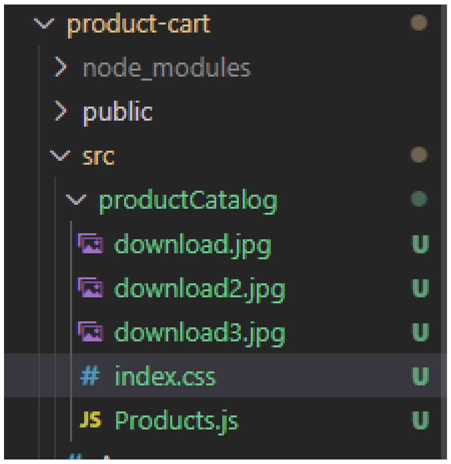
Step 5 : Additional Styling

Create **index.css** inside productCatalog and give some styling as below

index.css

```
li {
   justify-content: space-between;
}
li p {
   flex-grow: 1;
}
li img {
   margin-right: 15px;
   border-radius: 10px;
}
```

The folder will look something like this



```
product-cart
  node_modules
  public
  src
    productCatalog
      download.jpg       U
      download2.jpg      U
      download3.jpg      U
      index.css          U
      Products.js        U
```

Step 6: Render your Products Component

In **index.js** , import your Products.js and render it.

index.js

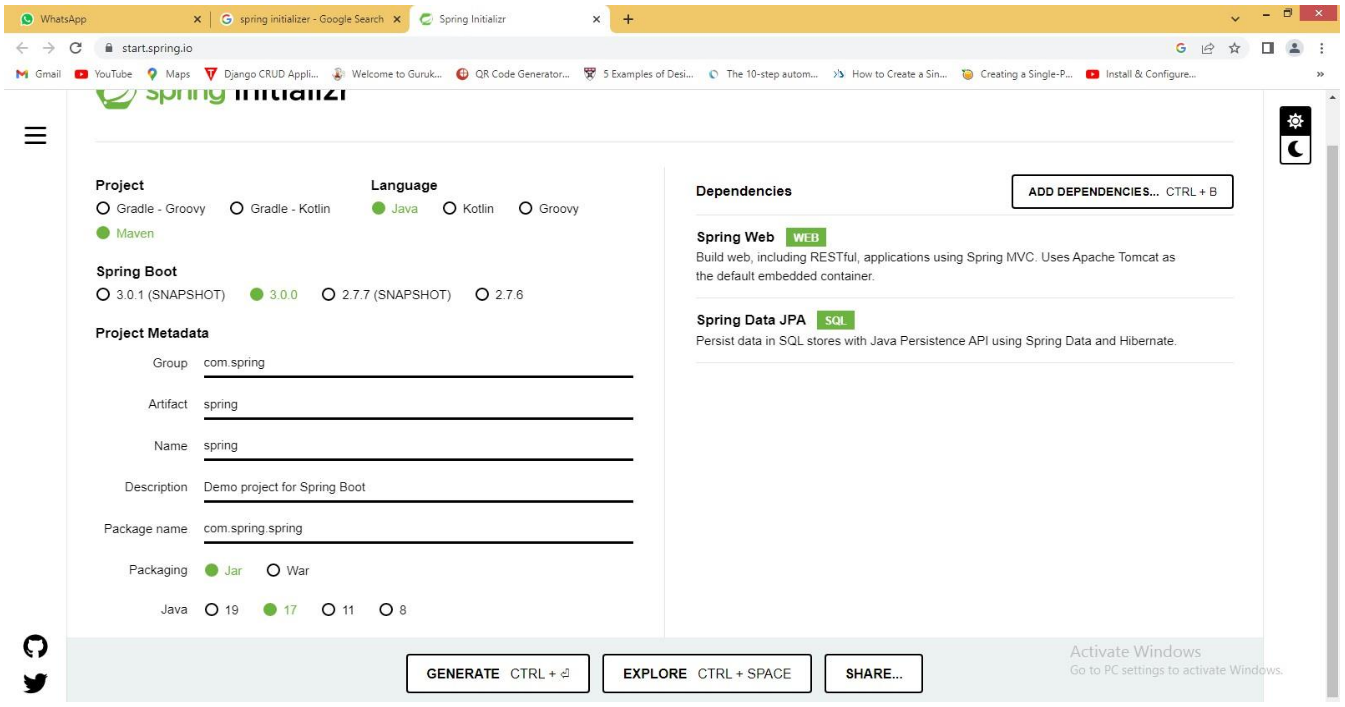
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Products from './productCatalog/Products';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Products />
);
```

13. Create Spring application with Spring Initializer using dependencies like SpringWeb, Spring Data JPA

Step1: goto google and search for spring initialize. Visit https://start.spring.io/ website

Step2: Choose project, language, spring Boot version. Add project metadata and dependencies as shown below



Project: Gradle - Groovy Gradle - Kotlin [✓ Maven]

Language: [✓ Java] Kotlin Groovy

Spring Boot: 3.0.1 (SNAPSHOT) [✓ 3.0.0] 2.7.7 (SNAPSHOT) 2.7.6

Project Metadata

Group	com.spring
Artifact	spring
Name	spring
Description	Demo project for Spring Boot
Package name	com.spring.spring
Packaging	[✓ Jar] War
Java	19 [✓ 17] 11 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Step3: click on generate→goto downloan and extract the zip file.

Step4: Open Eclipse →file→import→maven→existing maven project→next-→browse the extracted file→next→finish

Step5: Goto main Method→Add
System.out.println("Welcome to Spring Boot Application");
Right Click and Run as Spring Boot App

14. Create REST controller for CRUD operations

Step 1: Go to Eclipse→Help→Eclipse Marketplace→Find/Search for STS4(Spring Tool Suite4) and Install

Step 2: Click on **File -> New ->Project-> Spring Starter Project**

Name: Springboot-first-app

Dependencies: Spring Web, Spring Data JPA, MySQL Driver

Step3: Create 3 Packages with the following names entity, controller and repository

Step4: Create User.java class under entity package, Usercontroller.java under controller package and UserRepository.java **interface under repository package**

Step4: Write the following Code

User.java

```
package com.example.demo.entity;
// Import required packages and dependencies
@Entity
@Table(name="user")
public class User {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private Long id;
	private String firstname;
	private String lasttname;
//Add Getter & Setter
//Add Default and parameter constructor
Note: Right click → source → select getter& setter
}
```

UserRepository.java

```
// Import required packages and dependencies

@Repository
public interface UserRepository extends JpaRepository<User,Long>
{
}
```

Usercontroller.java

```
package com.example.demo.controller;
// Import required packages and dependencies

@RestController
@RequestMapping("/users")
public class Usercontroller {
@Autowired
private UserRepository userRepository;

@GetMapping
public List<User> getAllUser()
{
        return this.userRepository.findAll();
}

@GetMapping("/{id}")
public User getUserById(@PathVariable(value="id") long userId) {
        return this.userRepository.findById(userId).orElseThrow();
}
@PostMapping
public User createUser(@RequestBody User user)
{
        return this.userRepository.save(user);
}
@PutMapping("/{id}")
public User updateUser(@RequestBody User user,@PathVariable("id") long userId)
{
        User ex=this.userRepository.findById(userId).orElseThrow();
        ex.setFirstname(user.getFirstname());
        ex.setLasttname(user.getLasttname());
        return this.userRepository.save(ex);
}
@DeleteMapping("/{id}")
public ResponseEntity<User> deleteUser(@PathVariable("id") long userId)
{
        User ex=this.userRepository.findById(userId).orElseThrow();
        this.userRepository.delete(ex);
        return ResponseEntity.ok().build();
}
}
```

Application.property

```
        spring.datasource.url=jdbc:mysql://localhost:3306/emp
        spring.datasource.username=root
        spring.datasource.password=root
        spring.jpa.hibernate.ddl-auto = update
```

15. Test created APIs with the help of Postman

Note: Create crud operation to Test with Postman

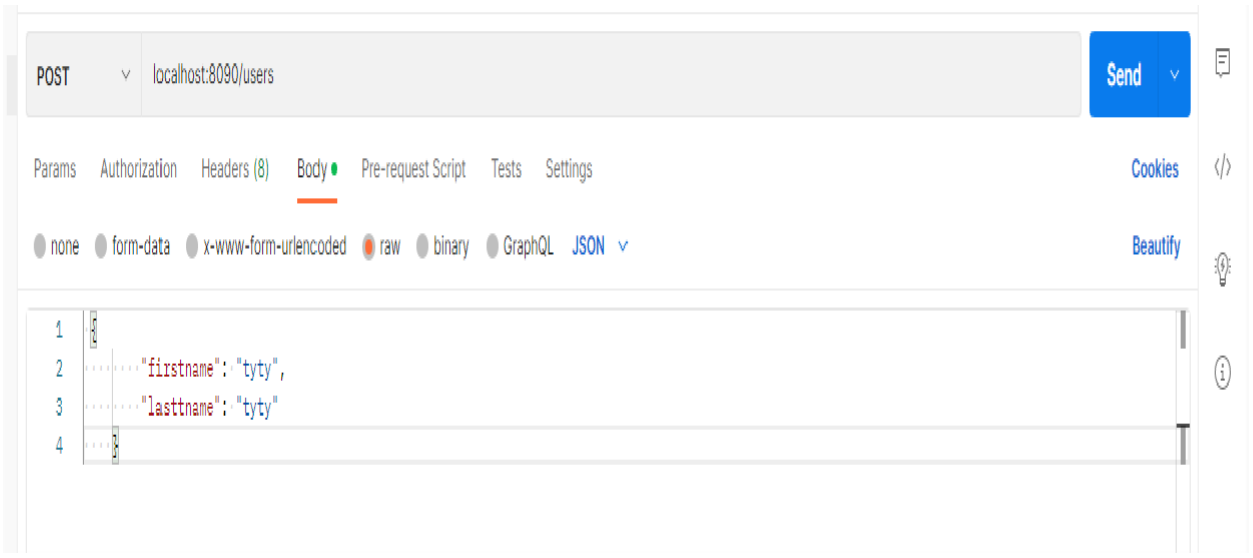
Step1: Download & Install postman from official website
https://www.postman.com/downloads/

Step2: Click on Collection and Create Collection → Add Request

Step3: Demonstrate Get, Post, Put, Delete methods

Get: Select Get method from dropdown list and enter the URL [localhost:8090/users] → Send

Post: Select Post method from dropdown list → Click on Body, choose raw and select JSON from dropdown list and enter the URL [localhost:8090/users] → Give the input in the form of JSON and Click on Send



localhost:8090/users

```
{
    "firstname": "tyty",
    "lastname": "tyty"
}
```

Put: Select Put method from dropdown list and enter the URL [localhost:8090/users/1] Update the existing data by using primary key and Click on Send

Delete: Select Delete method from dropdown list and enter the URL [localhost:8090/users/1]

16. Writing Junit test cases for CRUD operations

Note: Create crud operation to Test with Junit

Download JUnit from https://junit.org/junit4/

Goto download & install

Find Plain-old Jar & Download the following

- junit.jar
- hamcrest-core.jar
- Create a folder in any drive by giving relevant name, copy and paste both jar files to the folder.
- Create a project in eclipse
- Right click on project select build path, click on configure build path
- Select java build path, Click on Libraries and click on class path in libraries, go to Add External JAR's, select junit.jar and hamcrest-core.jar files, click on apply and then apply and close.
- Goto src/test/java folder find default package and Testclass
- Write the below code

```
// Import required packages and dependencies
@SpringBootTest
class SpringbootFirstAppApplicationTests {
        @Autowired
        UserRepository userRepo;
        @Test
        public void testCreate()
        {
                User u=new User();
                u.setId(3L);
                u.setFirstname("Kavya");
                u.setLasttname("shree");
```

```
            userRepo.save(u);
            assertNotNull(userRepo.findById(902L).get());
    }
    @Test
    public void testReadAll()
    {
            List<User> list=userRepo.findAll();
            assertThat(list).size().isGreaterThan(0);
    }
    @Test
    public void testUpdate()
    {
            User u=userRepo.findById(2L).get();
            u.setFirstname("Murthy");
            userRepo.save(u);
            assertNotEquals("Niranjan",userRepo.findById(902L).get().getFirstname());
    }
    @Test
    public void testDelete()
    {
            userRepo.deleteById(2L);
            assertThat(userRepo.existsById(852L)).isFalse();
    }
}
```

17. CRUD Operations on document using Mongo DB

Creating a Table.

```
db.createCollection("student")
{ ok: 1 }
show tables
student
```

insert() Method

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

Syntax: db.COLLECTION_NAME.insert(document)

```
db.student.insert({"id":1,"name":"chandru","mark":300})

db.student.insertMany([{"id":1,"name":"chandru","mark":300},
                                        {"id":2,"name":"suman","mark":290}])
```

View data from Table.

```
db.student.find({})
```

Update.

```
db.student.update({"name":"chandru"},{$set:{"name":"sekar",id:5}})
```

Delete only one data.

```
db.student.deleteOne({"name":"sekar"})
```

18. Perform CRUD Operations on MongoDB through REST API using Spring Boot StarterData MongoDB

Step 1: Create a Spring Boot project.

Step 2: Add the following dependency

- Spring Web
- MongoDB
- Lombok
- DevTools

Step 3: Create 3 packages and create some classes and interfaces inside these packages

- entity
- repository
- controller

Step 4: Inside the entity package create a Book.java file.

```
// Import required packages and dependencies
@Data
@NoArgsConstructor
@AllArgsConstructor
@Document(collection = "Book")
public class Book
{
    @Id
    private int id;
    private String bookName;
    private String authorName;
    //Call Getter & Setter
}
```

Step 5: Inside the repository package

Create a simple interface and name the interface as **BookRepo**. This interface is going to extend the **MongoRepository**

// Import required packages and dependencies

```
public interface BookRepo extends MongoRepository<Book, Integer> {
}
```

Step 6: Inside the controller package. Inside the package create one class named as **BookController**

// Import required packages and dependencies

```
@RestController
public class BookController {
    @Autowired
    private BookRepo repo;

@PostMapping("/addBook")
    public String saveBook(@RequestBody Book book){
            repo.save(book);
            return "Added Successfully";
    }
@GetMapping("/findAllBooks")
    public List<Book> getBooks() {
            return repo.findAll();
    }
@DeleteMapping("/delete/{id}")
    public String deleteBook(@PathVariable int id){
            repo.deleteById(id);
            return "Deleted Successfully";
    }
}
```

Step 7: Below is the code for the application.properties file

```
server.port:8989
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=jss
```

Step 8: Inside the MongoDB Compass

Go to your MongoDB Compass and create a Database named **BookStore** and inside the database create a collection named **Book**

Testing the Endpoint in Postman

POST – http://localhost:8989/addBook

GET – http://localhost:8989/findAllBooks

DELETE – http://localhost:8989/delete/1

19. Securing REST APIs with Spring Security

In order to add security to our Spring Boot application, we need to add the *security starter dependency*

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

This will also include the *SecurityAutoConfiguration* class containing the initial/default security configuration.

By default, the Authentication gets enabled for the Application. Also, content negotiation is used to determine if basic or formLogin should be used.

There are some predefined properties:

```
spring.security.user.name=root
spring.security.user.password=root
```

If we don't configure the password using the predefined property *spring.security.user.password* and start the application, a default password is randomly generated and printed in the console log:

```
Using default security password: c8be15de-4488-4490-9dc6-fab3f91435c6
```

File - new – Project - spring starter project
Name: spring-basic-security
Package: com.example.security
Click Next - Add Dependencies: Spring Web, Spring Security, Spring Boot Dev Tools….
Finish
Name: SpringBasicSecurityApplication
package com.example.security;

SecurityController.java

```
package com.example.security;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class SecurityController {
        @GetMapping("/")
        public String Welcome() {
                return ("<h1>Welcome to SpringBoot Security</h1>");
                }
        }
```

application.properties File

```
        spring.security.user.name=niranjan
        spring.security.user.password=murthy
        server.port=8090
```

20. Github Commands

Sl no.	Commands	Example	Description
1.	*git --version*		To display the version of the git downloaded onyour PC
2.	*git config --global user.name < "Username" >* *git config --global user.email < "valid-email" >*	git config --global user.name "riya123" git config –global user.email "riya2517@gmail.com"	GitHub uses the email address and username set in your local Git configuration to associate commits pushed from the command line with your account on GitHub.com.
3.	*git config --list*		To view what changes were made during the configuration.
4.	*git clone < link of a repository from Git hub >*	git clone https://github.com/riya123/project.git	To clone a repository from remote machine (Github) to local machine (PC)
5.	*git status*		To display the status of the code or a file • Untracked- new files that git hasn't trackedyet. • Modified- changed. • Staged- file is ready to be committed. • Unmodified- unchanged.
6.	*git init*		To initialize a new, empty repository.
7.	*git add < file name >* *or* *git add <.>*	git add f1.html git add .	To add a new or changed files in your workingdirectory to git staging area. (Use "." to add all the files that's in your working directory to the git staging area.)

8.	*git commit -m “message”*	git commit -m “this is my first commit”	To commit the changes that is made in a file. (-m is an option to specify the message that shouldbe displayed)
9.	*git push origin main*		To upload local repository (PC) content to remote repository (Git hub).
10.	*git remote add origin <link of the repository form git hub>*	git remote add origin https://github.com/riya123/repo.git	To add a new remote repository with the name oforigin.
11.	*git remote -v*		To verify that the remote repository actuallyexists.
12.	*git branch*		To check which branch we are on currently.
13.	*git branch -M <name>*	git branch -M main	To rename a branch
14.	*git push -u origin main*		-u : To set upstream, meaning that the next time you want to push something into git hub, you justhave to type “git push” instead of typing the full command every time. It specifies that you want towork on “origin main” for a long time.
15.	*git checkout <branch name>*	git checkout sub1	To navigate/get inside of a branch.
16.	*git checkout -b <new branch name>*	git checkout -b sub2	To create a new branch.
17.	*git branch -d <branch name>*	git branch -d sub1	To delete a branch
18.	*git diff <branch name>*	git diff sub2	To compare commits, files, branches and more
19.	*git pull origin main*		To fetch and download content from remote repository and update the local repository to matchthe content.
20.	*git merge <branch name>*	git merge sub3	To merge 2 branches together.
21.	*git reset <file name>*	git reset sub4	To undo the changes after adding the changes that were done to a file

22.	*git reset HEAD~1*		To undo the changes by 1 step/commit which has already been committed.
23.	*git log*		Shows the commit history of the current activebranch. (Commit hash can be copied from here)
24.	*git reset <commit hash>*	git reset 56142346434sdf64645sfd4	Undoing committed changes by many commits.Multiple commits can be undo-ed by this. (But the undone changes will be just in git but not visible in VS code)
25.	*git reset --hard <commit hash>*	git reset --hard 56142346434sdf64645sfd4	The undone changes will be visible in VS code

21. Docker Commands

Build Commands

docker build - Builds an image from a Dockerfile located in the current directory

docker build https://github.com/ docker/rootfs.git#container:docker - Builds an image from a remote GIT repository

docker build -t imagename/tag - Builds, and tags an image for easier tracking

Clean Up Commands

docker image prune – Clears up unused images

docker image prune -a – Clears all images that are not being used by containers

docker system prune - Removes all stopped containers, all networks not used by containers, all dangling images, and all build cache

docker image rm image - Removes an image

docker rm container - Removes a running container

docker kill $ (docker ps -q) - Stops all running containers

Container Interaction Commands

docker start container – Starts new container

docker stop container – Stops a container

docker pause container – Pauses a container

docker unpause container – Unpauses a container

docker wait container – Blocks a container

docker restart container – Restarts a container

docker create image – Creates a new container from image.

Container Inspection Commands

docker ps

Lists all running containers

docker -ps -a

Lists all containers

docker top container

Shows all running processes in an existing container

docker inspect container

Displays low-level information about a container

docker logs container

Gathers the logs for a container

docker stats container

Shows container resource usage statistics

Managing Image Commands

docker ps

Lists all running containers

docker -ps -a

Lists all containers

docker diff container

Inspects changes to directories and files in the

container filesystem

docker top container

Shows all running processes in an existing container

docker inspect container

Displays low-level information about a container

docker logs container

Gathers the logs for a container

docker stats container

Shows container resource usage statistics

Run Commands

Docker uses the run command to create containers from provided images. The default syntax for this command

docker run [options] image [command] [arg...]

Then you can use one of the following flags:

--detach-d

Runs a container in the background and id

--eny-e

Sets environment variables

--hostname-h

Sets a hostname to a container

--label-I

Creates a meta data label for a container

--name

--network

Connects a container to a network

--rm

Removes container when it stops

--read-only

Sets the container filesystem as read-only

--workdir -W

Sets a working directory in a container

<u>Registry Commands</u>

docker login

Log ins to a registry

docker logout

Logs out from a registry

docker pull mysql

Pulls an image from a registry

docker push repo/ rhel-httpd latest

Pushes an image to a registry

docker search term

Searches Required hub for images with specifics

Service Commands

docker service ls

Lists all services running in a swarm

docker stack services stackname

Lists all running services

docker service ps servicename

Lists the tasks of a service

docker service update servicename

Updates a service

docker service create image

Creates a new service

docker service scale servicename=10

Scales one or more replicated services

docker service logs stackname servicename

Lists all service logs