

### 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 register

- “Given” I am on registration page
- “And” I give valid customer name and phone number
- “And” I click on sign in
- “Then” I will successfully register

**Scenario 2:** customer cannot successfully register

- “Given” I am on registration page
- “And” I give invalid customer name and phone number
- “Then” I will get an 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 a 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 (wire framing).

- Continue with your Gmail account or login to Figma.
- First create design file

- And adding elements to over design file from figma community
- Click on“ # ” button on the tool menu at the (Top left)
- Depends on which size you want to use choose the screen size from the right sidebar.
- Add background color to the frame by clicking it and add color from the “Fill” section in the (right panel).
- Create text button (click on “T” text button from the (Top left)
- Click on rectangle“□ ” button to select image from the popup menu at the (Top left)

### 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.

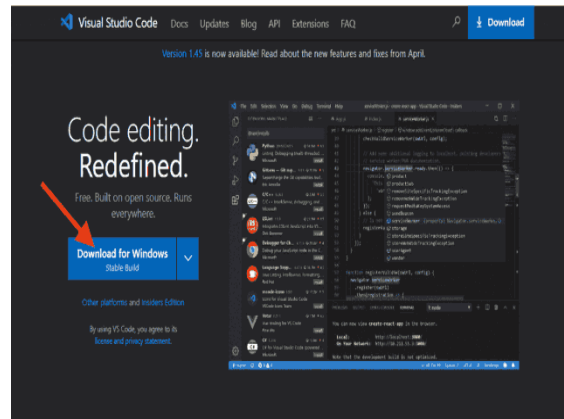
#### **Clone or pull operation:**

- Click on code dropdown button
- Click on Download Zip

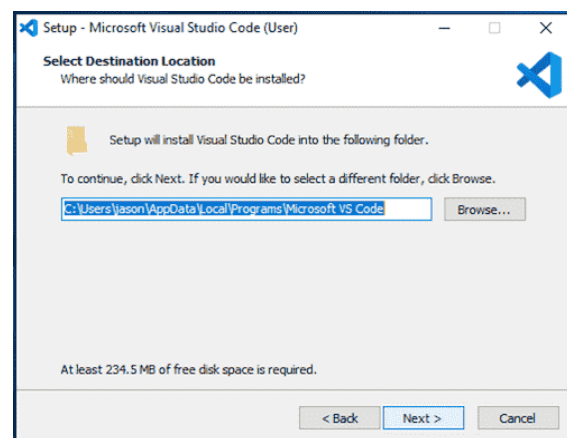
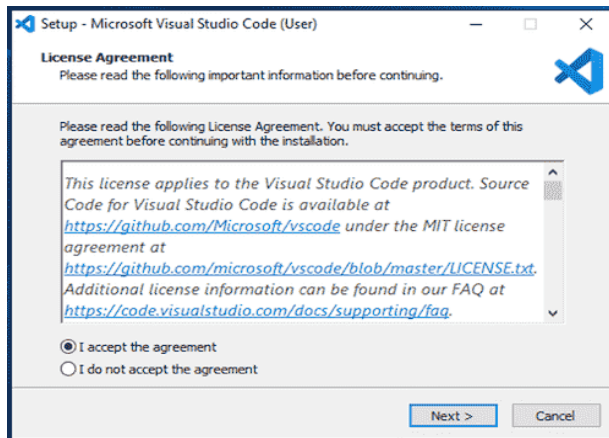
### 6. **Setting Up the Environment and Tools for front end development Installing VS Code**

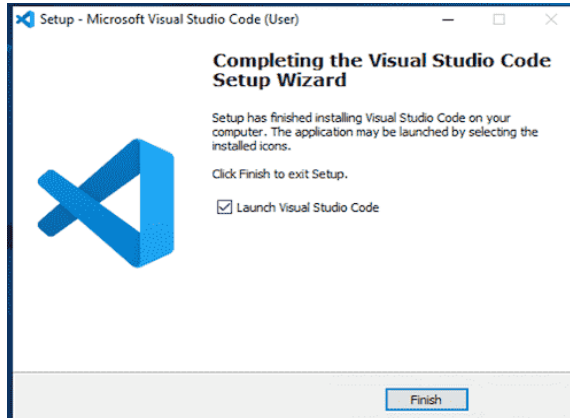
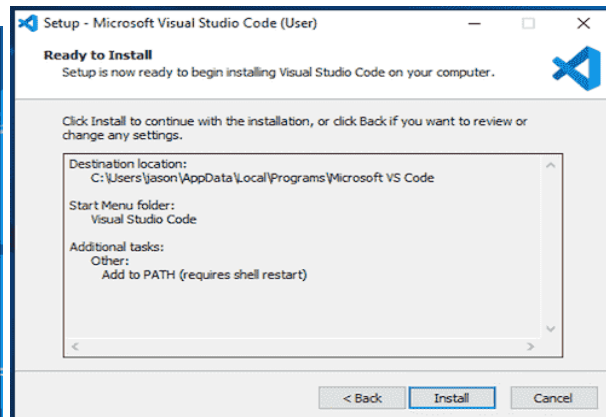
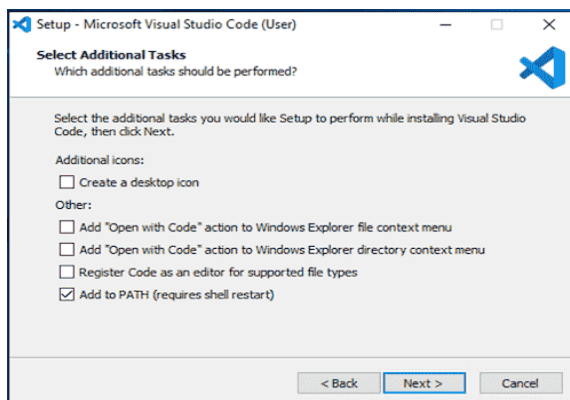
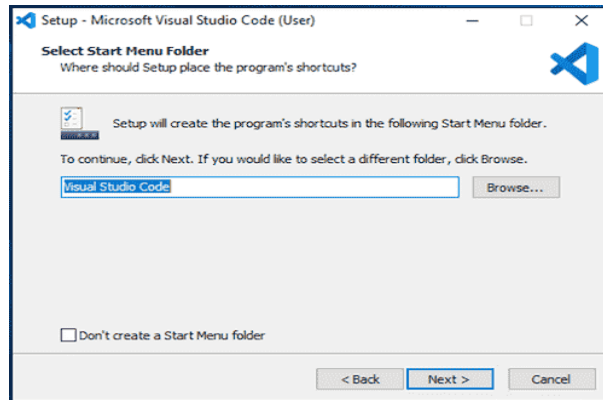
VS Code is a free code editor that runs on Windows, Mac and Linux.

Download VS Code from <https://code.visualstudio.com/>.



Install Visual Studio Code by opening the downloaded setup file and following the prompts.





7. 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>Registrtrion 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>
```

## 8. 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>

## **9. 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
```

**10. 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

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

```
<html>
<head>
<title>Form Elements</title>
</head>
<body>
<form>
<labe>Text Box</labe>
<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>

## 12. Testing single page application (Registration form) using React.

### Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import Home from './Home';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
```

```
    <Home />
  </React.StrictMode>
);
```

### **App.css**

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

### **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>
);
```

```
        </button>
      </fieldset>
    </form>
  </div>
);
}
```

### 13. 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.**

**Blogs.js**

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

**Contact.js**

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

**Home.js**

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

**Layout.js**

```
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;
```



```
    </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;  
}
```

#### 14. Build single page application (Add Product to Product List)

##### App.js

```
import { useState } from "react";  
import "./App.css";  
function App() {  
  const [list, setList] = useState([]);  
  const [value, setValue] = useState("");  
  const addToList = () => {  
    let tempArr = list;  
    tempArr.push(value);  
    setList(tempArr);  
    setValue("");  
  };  
  const deleteItem = (index) => {  
    let temp = list.filter((item, i) => i !== index);  
    setList(temp);  
  };  
  return (  
    <div className="App">  
      <fieldset>  
        <h>Add Product to List</h><br></br>  
        <input type="text" value={value} onChange={e => setValue(e.target.value)} />  
        <button onClick={addToList}> Click to Add </button><br></br><br></br>  
        <h>Product Catalog</h><br></br>  
        <ol>  
          {list.map((item, i) => <li onClick={() => deleteItem(i)}>{item} </li>)}  
        </ol>  
      </div>  
    )
```

```
    </ol>
    <h>Click on Product to Delete</h><br></br>
  </fieldset></div>
);
}export default App;
```

**index.js**

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App/>
  </React.StrictMode>
);
```

**15. Build simple page application like shopping cart using ReactJS.****App.js**

```
import Header from './Header';
import Products from './Product';
import { useState } from 'react';
import CartList from './CartList';
function App() {
  const [product, setproduct] = useState([
    {
      url:'imgs/lenovo.png',
      name: 'lenovo ideapad Slim 3',
      price: 57000
    },
    {
      url: 'imgs/watch.png',
```

```
name: 'fastrack w98',
price: 1500
},
])
const [cart, setCart] = useState([])
const [showCart, setShowCart] = useState(false)
const addToCart = (data) => {
  setCart([...cart, { ...data, quantity: 1 }])
}
const handleShow = (value) => {
  setShowCart(value)
}
return (
  <div >
    <Header count={cart.length} handleShow={handleShow} />
    {
      showCart ?
      <CartList cart={cart} /> :
      <Products product={product} addToCart={addToCart} />
    }
  </div>
)
}
export default App;
```

### **Product.js**

```
import React from 'react';
export default function Products({product,addToCart} ){
  return (
    <div className='flex'>{
      product.map((productitem,productIndex)=>{
        return(
          <div>
            <img src={productitem.url} width="20%" alt=""/>
            <p>{productitem.name}</p>
            <p>Rs. { productitem.price}</p>
            <button onClick={()=>addToCart(productitem)}>Add Cart</button>
          </div>
        )
      })
    }
  )
}
```

```
  })  
  }  
</div>  
)  
}
```

### **CarList.js**

```
import React,{useState,useEffect} from 'react';  
function CartList({ cart }) {  
  const [CART,setCART]= useState([])  
  useEffect(() => {  
    setCART(cart)  
  }, [cart])  
  return (  
    <div>  
      {  
        CART?.map((cartitem,cardindex)=>{  
          return(  
            <div>  
              <img src={cartitem.url} width={60} />  
              <span> {cartitem.name} </span>  
              <button onClick={()=>{  
                const _CART= CART.map((item,index)=>{  
                  return cardindex ===index? {...item,quantity:item.quantity>0?item.quantity-1:0}:item  
                })  
                setCART(_CART)  
              }}>  
                - </button>  
              <span> {cartitem.quantity} </span>  
              <button onClick={()=>{  
                const _CART= CART.map((item,index)=>{  
                  return cardindex ===index? {...item,quantity:item.quantity+1}:item  
                })  
                setCART(_CART)  
              }}>+ </button>  
              <span> Rs.{cartitem.price* cartitem.quantity} </span>  
            </div>  
          )  
        })  
      }  
    )  
  )  
}
```

```
}
<p>Total=<span>
</span>
{ CART.map(item=>item.price*item.quantity).reduce((total,value)=>total+value,0)}
</p>
</div>
)
}
export default CartList;
```

### Header.js

```
import React from 'react'
export default function Header(props) {
  return (
    <div>
      <div onClick={()=>props.handleShow(false)}>ShoppingCart</div>
      <div onClick={()=>props.handleShow(true)}> Cart
      <sup>{props.count}</sup>
    </div>
  </div>
  )
}
```

```
-----
spring.security.user.name=niru
spring.security.user.password=niru
server.port=8090
-----
```

## 16.Create Spring application with Spring Initializer using dependencies like Spring Web, 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

The screenshot shows the Spring Initializr web application in a browser. The URL is start.spring.io. The page is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a bottom bar with buttons. In the Project section, 'Maven' is selected. In the Language section, 'Java' is selected. In the Spring Boot section, '3.0.0' is selected. In the Project Metadata section, the Group is 'com.spring', Artifact is 'spring', Name is 'spring', Description is 'Demo project for Spring Boot', and Package name is 'com.spring.spring'. In the Dependencies section, 'Spring Web' and 'Spring Data JPA' are listed. The bottom bar has buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'. There is also a 'Activate Windows' watermark in the bottom right corner.

Step3: click on generate → goto download 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 → Right Click and Run as Spring Boot App

## 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 jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="user")
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstname;
    private String lastname;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getFirstname() {
        return firstname;
    }
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
    public String getLastname() {
```



```
        return lastname;
    }
    public void setLasttname(String lastname) {
        this.lastname = lastname;
    }
    public User(Long id, String firstname, String lastname) {
        super();
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
    }
    public User() {
    }
}
```

#### **UserRepository.java**

```
package com.example.demo.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.entity.User;
```

```
@Repository
```

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

```

#### **Usercontroller.java**

```
package com.example.demo.controller;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.entity.User;
import com.example.demo.exception.ResourceNotFoundException;
import com.example.demo.repository.UserRepository;
```

```
@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(() -> new
```

```
ResourceNotFoundException("UserNotFound"+userId));
```

```
}
```

@PostMapping

**public** User createUser(@RequestBody User user)

{

**return this**.userRepository.save(user);

}

@PutMapping("/{id}")

**public** User updateUser(@RequestBody User user, @PathVariable("id") **long** userId)

{

    //return this.userRepository.save(user);

    User ex=**this**.userRepository.findById(userId)

        .orElseThrow(()-> **new**

ResourceNotFoundException("UserNotFound"+userId));

    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();

}

}

## 17. Test created APIs with the help of Postman

**Step1:** Download & Install postman from official website

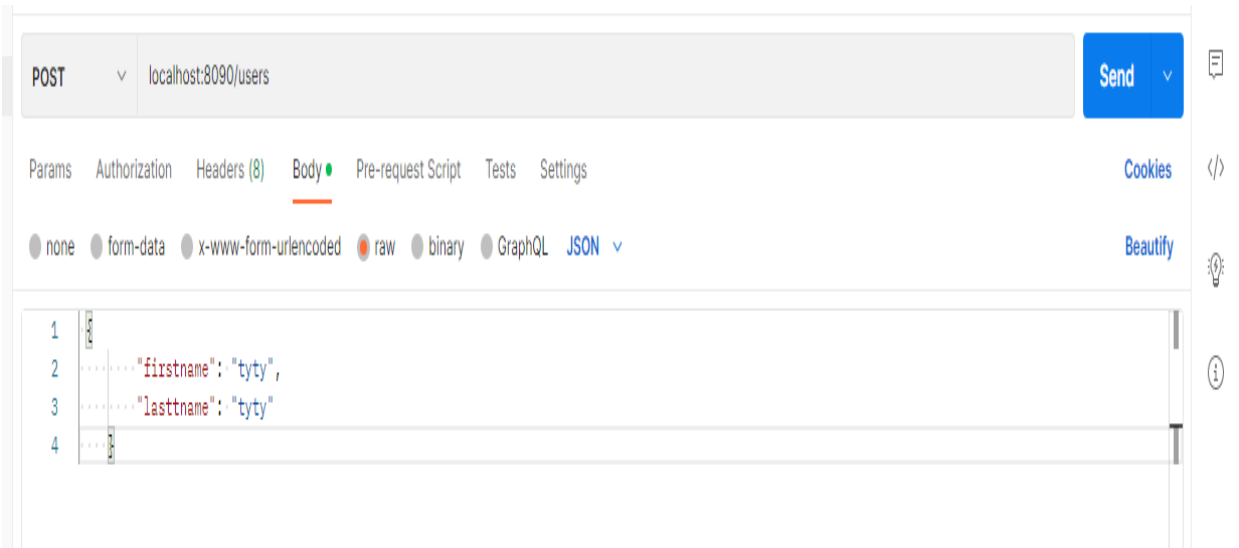
<https://www.postman.com/downloads/>

**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



**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]

### 18. Writing Junit test cases for CRUD operations

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

```
package com.example.demo;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertNotNull;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
```

```
@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();
}
}
```

**Note: Perform the junit test for Rest API CRUD operation**

## 19. 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"})
```

## **20. Perform CRUD Operations on MongoDB through REST API using Spring Boot**

### **Starter Data 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.

```
package com.example.security.entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Data
@NoArgsConstructor
@AllArgsConstructor
@Document(collection = "Book")
public class Book {
    @Id
    private int id;
    private String bookName;
    private String authorName;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getBookName() {
        return bookName;
    }
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }
    public String getAuthorName() {
```



```
        return authorName;
    }
    public void setAuthorName(String authorName) {
        this.authorName = authorName;
    }
}
```

**Step 5:** Inside the repository package

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

```
package com.example.security.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.security.entity.Book;
public interface BookRepo
    extends MongoRepository<Book, Integer> {
}
```

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

```
package com.example.security.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.example.security.entity.Book;
import com.example.security.repository.BookRepo;
import java.util.List;
@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>

Create and manage users and roles Migration to MongoDB

Integrate the work of each group and carry out integration testing

Bug tracking – using Jira or similar tools

Practice – code analysis using tools

Install docker on desktop and start the docker tool.

create docker container from docker image Run the docker container

### 21. 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=niranjana

spring.security.user.password=murthy

server.port=8090