

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

### **Creating project plan**

- ✓ With project planning and project management running smoothly, IT departments are able to deliver more projects for the same budgets, and the majority of properly managed projects are completed by their due time/date.
- ✓ Planning for every project is very individual and specific. However, several major steps are going to be similar for many projects of the businesses.

### **Step #1: Write down project scope of work**

- ✓ The first step to start with is to craft a scope statement and a general outline for the project. Creating the outline and the scope is usually the first step towards writing technical documentation and specs.
- ✓ Certainly, documentation is going to be changed in the scope of developing the project but initially, it is created by stakeholders and a project manager.
- ✓ The first step also contains developing the purpose and objectives of the project as well as the business justification for the project and is practically a basis for the entire future work.

### **Step #2: Organize deliverables into a work breakdown structure (WBS)**

- ✓ When it comes to implementing project ideas effectively and promptly, there is a need to prepare deliverables.
- ✓ In turn, making deliverables is a part of the WBS that is the work breakdown structure for the project.
- ✓ A manager is the one to create the WBS according to the requirements of the customer, which need to include project scope, project tasks, and tasks split into simpler subtasks.
- ✓ Thus, the WBS is necessary as a project roadmap.

### STEP #3: Get to know your dependencies

- ✓ When you already have the parts of your project to do with the WBS, it is time to create a to-do list for every subtask.
- ✓ At this point, many managers often ignore dependencies and this might create bottlenecks in the project.
- ✓ To avoid bottlenecks, there is a need to plan tasks, taking in mind their dependencies on other tasks and subtasks
- ✓ **STEP #4: Create a timeline**
- ✓ Gantt chart is a very convenient tool to use when it comes to planning the milestones and the dependencies.
- ✓ With a Gantt chart, you can plan time and tasks in a way that at any given moment you can see the status your project is in.
- ✓ The example of a Gantt chart is represented in the picture below.

### Free Gantt Chart Template



### STEP #5 Plan budget

- ✓ It goes without saying, before starting the project, there is a need to estimate the total cost that you can afford.
- ✓ Ideally, there should be several estimations, depending on the timeline and features added to the project.

- ✓ For example, you might want to have the lowest and the highest total cost of your project at hand so you can estimate the project from several points of view.
- ✓ Potentially, this helps to be more realistic with the budget, and determine the features that are a must for the project and the ones that are auxiliary but desired.
- ✓ This kind of estimate might look like the following:

Estimator		Project Cost Estimation				Date			
Project Name						Project Number			
Client Name						Start Date			
Project Manager						End Date			
Remarks						Total #Deliverable:			
	Project Phase	Estimated Hours	Developers		Analysts		Others Cost	Total Cost	Status
			Req.	Avg. Cost	Req.	Avg. Cost			
1	Phase 1	12	3	\$90	6	\$90	\$90	\$900	
1.1	Phase 1.1	5	1	\$30	1	\$30	\$30	\$330	
1.2	Phase 1.2	5	1	\$30	0	\$30	\$30	\$180	
1.3	Phase 1.3	2	1	\$30	5	\$30	\$30	\$390	
2	Phase 2	20	13	\$90	6	\$90	\$90	\$4,740	
2.1	Phase 2.1	5	1	\$30	1	\$30	\$30	\$330	
2.2	Phase 2.2	5	5	\$30	0	\$30	\$30	\$780	
2.3	Phase 2.3	10	7	\$30	5	\$30	\$30	\$3,630	
3	Phase 2	6	7	\$90	18	\$150	\$150	\$1,020	
3.1	Phase 3.1	1	2	\$30	3	\$30	\$30	\$180	
3.2	Phase 3.2	1	2	\$30	3	\$30	\$30	\$180	
3.3	Phase 3.3	2	1	\$30	3	\$30	\$30	\$270	
3.4	Phase 3.4	1	1	\$30	4	\$30	\$30	\$180	
3.5	Phase 3.5	1	1	\$30	5	\$30	\$30	\$210	
Total		38	23		30		\$330	\$6,660	
Estimated Hours		38							
FTE		53							
Total Cost		\$6,660							

Project Cost Estimate Template By: [www.analysistabs.com](http://www.analysistabs.com)

Project Cost Estimate Template By: [www.analysisstabs.com](http://www.analysisstabs.com)

## Creating product Backlog

According to the official Scrum Guide, the product backlog is “...an ordered list of everything that is known to be needed in the product.”

The product backlog sits outside of the sprint loop (meaning it contains work that will not be completed during the current sprint) but informs how your sprint will be planned. The product backlog is composed of feedback from:

- The development team
- Customer
- Stakeholders

**Follow these steps to start your scrum product backlog.**

### 1. Add ideas to the backlog

Stakeholders will typically be approaching you with ideas for product improvements

## 2. Get clarification

Once you're approached by a stakeholder with a product addition or fix, make sure you understand:

- The reason behind the addition or fix
- The amount of value it contributes to the product as a whole
- The specifications of the item

## 3. Prioritize

The backlog should have clearly defined, high-priority items at the top and vague items that are not a priority at the bottom. If an item has no value, it should not be added to the backlog.

## 4. Update the backlog regularly

The backlog is a living document; make sure you're constantly prioritizing, refining, and keeping the backlog up to date.

You may have hundreds of items in your backlog as ideas for product improvements are suggested. Some of these items may be discarded, but many of them will begin making their way up the backlog for further refinement and, ultimately, development.

## Creating User Stories

A user story is a small unit of work in an **Agile workflow**. It is a short, written explanation of a particular user's need and how it can be fulfilled.

User stories follow a simple template. The chosen user story format will outline the "who," "what," and "why" of a particular requirement.

- **Who** wants something?
- **What** do they want?
- **Why** do they want it?

The following template is one of the most common:

**"As [persona], I want to [action], so that I can [benefit]."**

**Step 1: Outline acceptance criteria**

The definition of done is the set of criteria that needs to be fulfilled for your user story to be considered complete. Define the specific acceptance criteria for each user story and use it as a checklist.

**Step 2: Decide on user personas**

Conduct extensive user research by creating surveys, hosting focus groups, and reading user forums. Analyze your data and search for patterns to identify your key personas.

**Step 3: Create tasks**

Break your user story down into numerous tasks to make it more manageable. If it is a complex requirement, you can also add subtasks. Include detailed descriptions, so your team is aligned on what each task requires.

**Step 4: Map stories**

Use story mapping to structure work in a large process. In this case, your user stories will take the form of ordered steps.

**Step 5: Request feedback**

Speak to users and potential customers to find out what they want. Ask them for their opinions on existing products or if they have suggestions for new features. Incorporate this feedback into your user story.

## **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**

##### **Scenario1:** 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

##### **Scenario2:** 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 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**

**Scenario1:** Customers can successful check availability of hall in their location

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

**Scenario2:** 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 they 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 booking details page
- “And” I fill the details
- “And” I have also blocked the hall
- “Then” I will successfully get the booking details

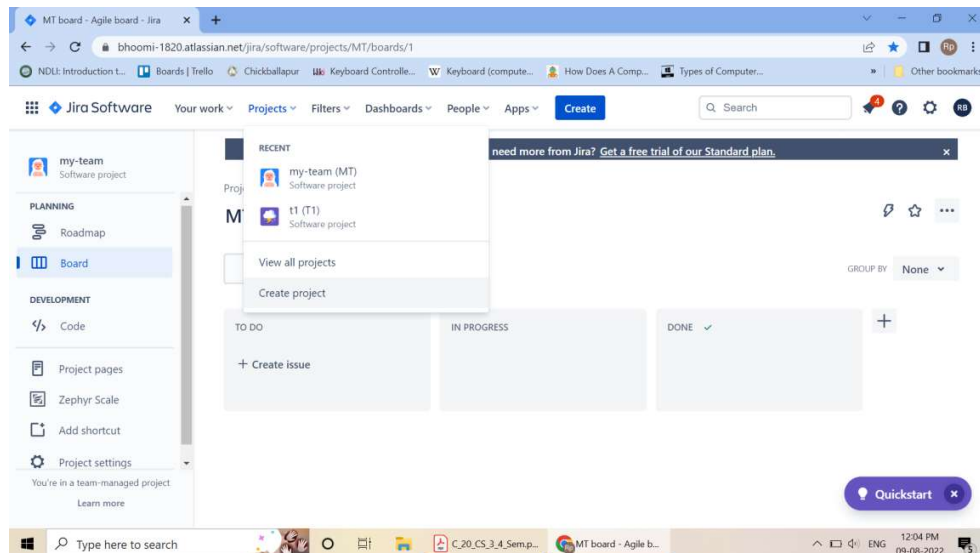
**Scenario 2:** customer will not get the booking details

- “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

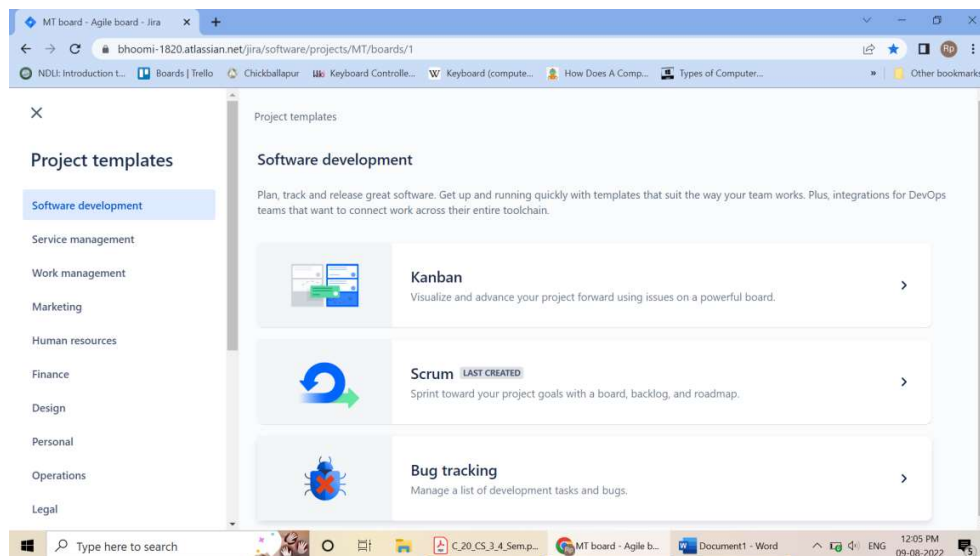


### 3. Create Sprint 1 with required user stories

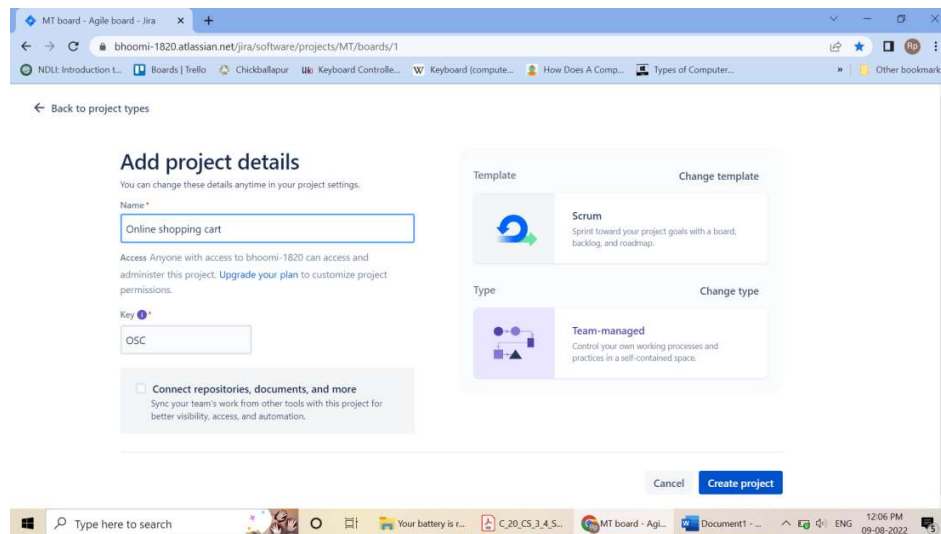
1. Login to your Jira account and click on create project to create a new one.



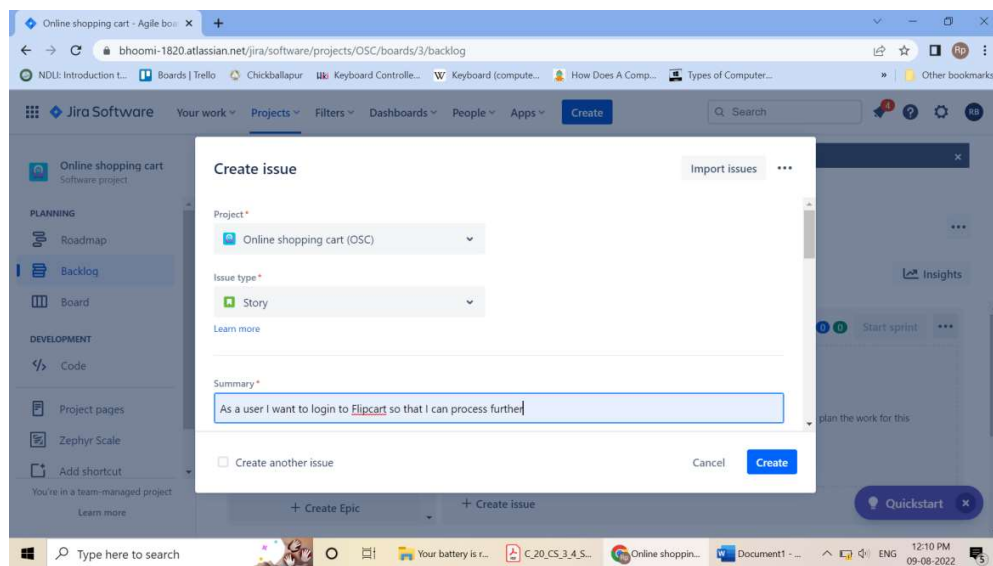
2. Choose your template.



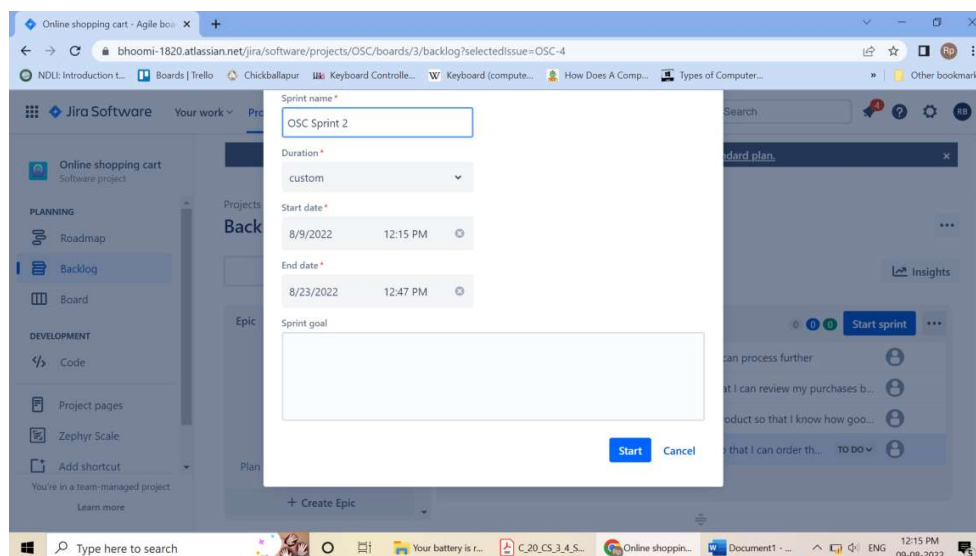
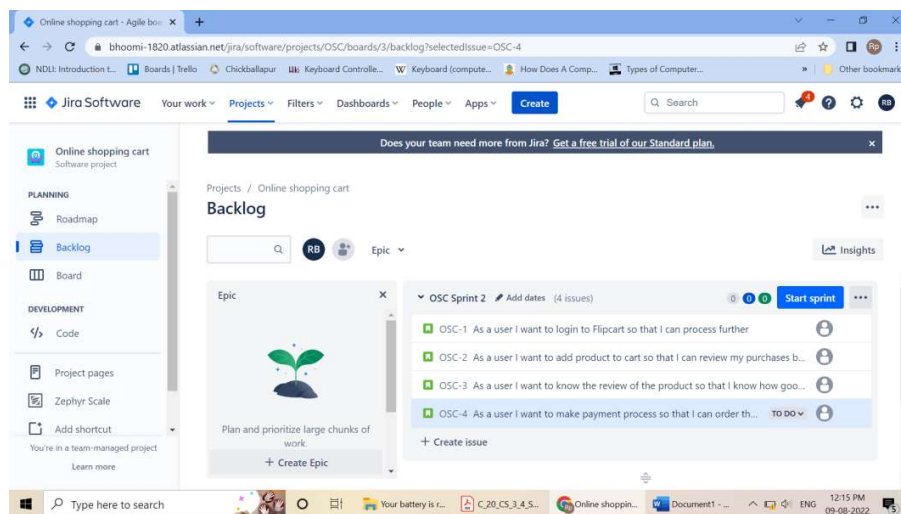
3. Give a name to your project. And then click on create project.



4. Now click on Create to create issue type of Story. And give the summary of your user story.



5. Create User Stories




### 1. Sprint1 is created on clicking on start

**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” textbutton 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-1Push 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

#### **Branching and merging**

- git branch newbranch -to create a new branch
- git checkout newbranch -to checkout from master branch to newbranch
- Make the required changes to the file commit it and then merge
- git checkout newbranch
- git mergenewbranch

## **6. Build a basic application on cloud**

### **Create a web application**

Use the program code (Forms – Use of HTML tags in forms select, input, file, etc.)

### **Deployment on cloud**

1. Open web browser and search for free cloud service (000webhost.com).
2. Proceed sign in process through the Google account.
3. Host free website and click on Manage Website.
4. Towards left column, tools>>file manager and select upload files.
5. Select public\_html and upload the existing html file.
6. After uploading, right click on the file and open.
7. Make sure the file has been uploaded.
8. Finally, we can perform the following operations
  - View the output of the webpageapplication
  - Download the webpage application
  - Share the webpage application through the given below
    - Webhost\_link + file\_name.extension
  - We can move from one directory to another
  - Either we can copy the content of the file or by copying the whole file

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

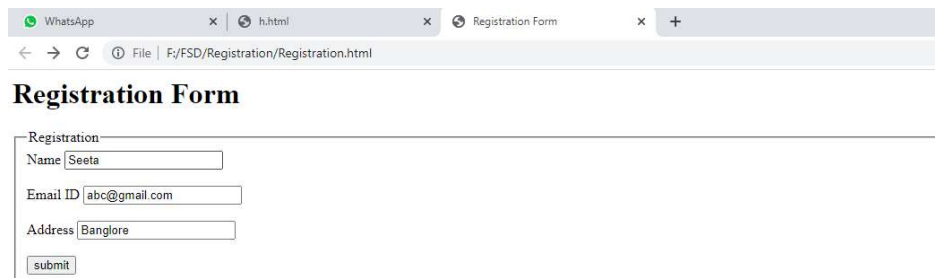
**Registration.html**

```
<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>Registrtrtion 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>
```



WhatsApp x h.html x Registration Form x +

← → ↻ File | F:/FSD/Registration/Registration.html

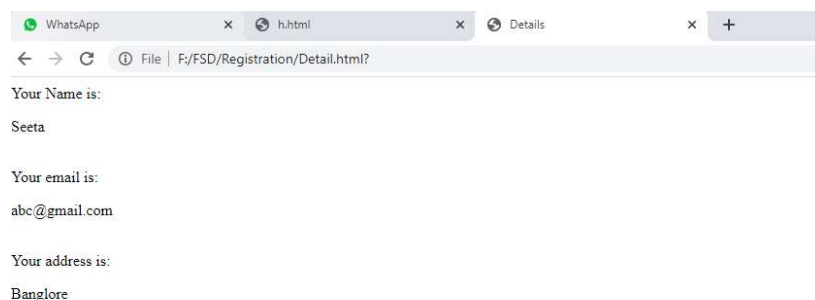
### Registration Form

Registration

Name

Email ID

Address



WhatsApp x h.html x Details x +

← → ↻ File | F:/FSD/Registration/Detail.html?

Your Name is:

Seeta

Your email is:

abc@gmail.com

Your address is:

Banglore



## 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.

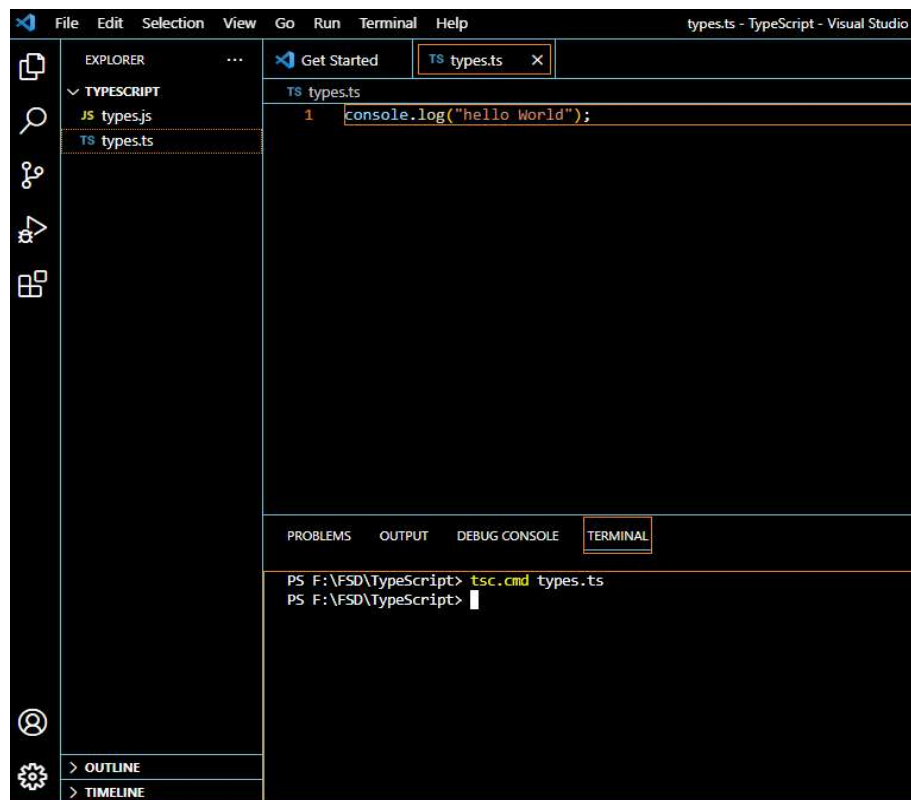
### Install Live server

`npm install -g live-server`

### 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` or `tsc.cmd types.ts`
- run the program
- `node types.js`
- Observe the output



The screenshot displays the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left shows a project named 'TYPESCRIPT' containing two files: 'types.js' and 'types.ts'. The 'types.ts' file is selected and open in the main editor. The code in the editor is as follows:

```
1 console.log("hello World");
```

Below the editor, the 'TERMINAL' tab is active, showing the following commands and output:

```
PS F:\FSD\TypeScript> tsc.cmd types.ts
PS F:\FSD\TypeScript> node types.js
hello World
PS F:\FSD\TypeScript> 
```

## 10.Demonstrate use of React Props

### App.js

```
import React from 'react';
import Classprops from "./Classprops";
import Functionprops from "./Functionprops";

class App extends React.Component{
  render() {
    return (
      <div>
        <Classprops name="Learner1" place="Bagepalii"></Classprops>
        <p>Child Component</p>
        <Classprops name="Learner2" place="Banglore"></Classprops>
        <button>Click</button>
        <Classprops name="Learner3" place="Mysore"></Classprops>
        <Functionprops name="Learner4" place="Madikeri"/>

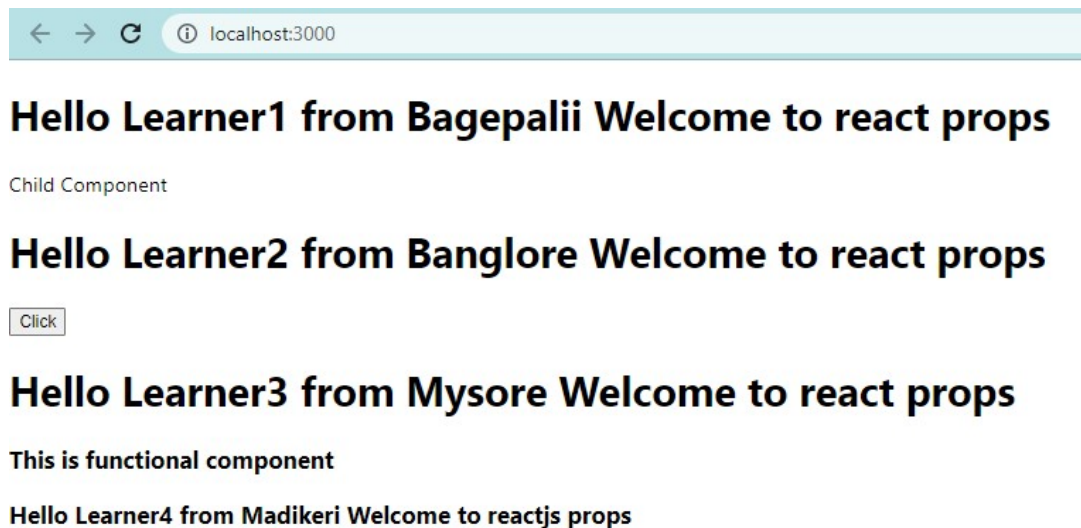
      </div>
    );
  }
}
export default App;
```

### Functionprops.js

```
import React from "react";
function Functionprops(props) {
  return (
    <div>
      <h3>This is functional component</h3>
      <h3>
        Hello {props.name} from {props.place} Welcome to reactjs props
      </h3>
    </div>
  );
}
export default Functionprops;
```

### Classprops.js

```
import React, { Component } from 'react';
class Classprops extends Component {
  render() {
    return (
      <div>
        <h1>Hello {this.props.name} from {this.props.place} Welcome to react props</h1>
        <p>{this.props.children}</p>
      </div>
    );
  }
}
export default Classprops;
```



## 11.Demonstrate Handling events in react

### Adding Events

- ✓ React events are written in camelCase syntax : onClick instead of onclick.
- ✓ React event handlers are written inside curly braces : onClick={shoot} instead of onClick="shoot()".

// Put the shoot function inside the Football component:

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

function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }
  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

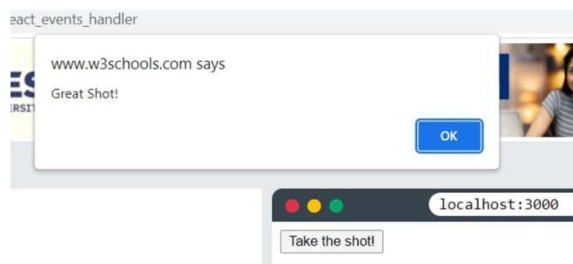
### Output

:

Before:



After:



### Passing Arguments

To pass an argument to an event handler, use an arrow function.

// Send "Goal!" as a parameter to the shoot function, using arrow function:

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

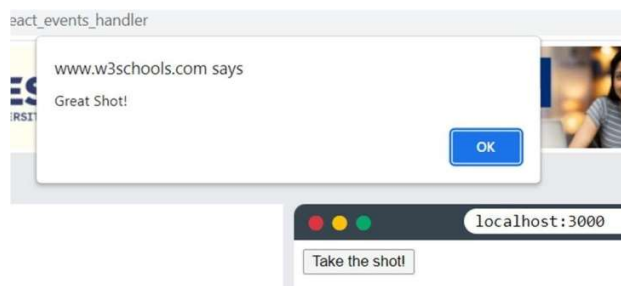
function Football() {
  const shoot = (a) => {
    alert(a);
  }
  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

### Output

Before:



After:





### Experiment 1.3

#### React Event Object

- ✓ Event handlers have access to the React event that triggered the function.
- ✓ In our example the event is the "click" event.

#### Example:

// Arrow Function: Sending the event object manually:

```
import React from 'react';
```

```
import ReactDOM from 'react-dom/client';
```

```
function Football() {
```

```
  const shoot = (a, b) => {
```

```
    alert(b.type);
```

```
    /*
```

```
    'b' represents the React event that triggered the function.
```

```
    In this case, the 'click' event
```

```
    */
```

```
  }
```

```
  return (
```

```
    <button onClick={{(event) => shoot("Goal!", event)}}>Take the shot!</button>
```

```
  );
```

```
}
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(<Football />);
```

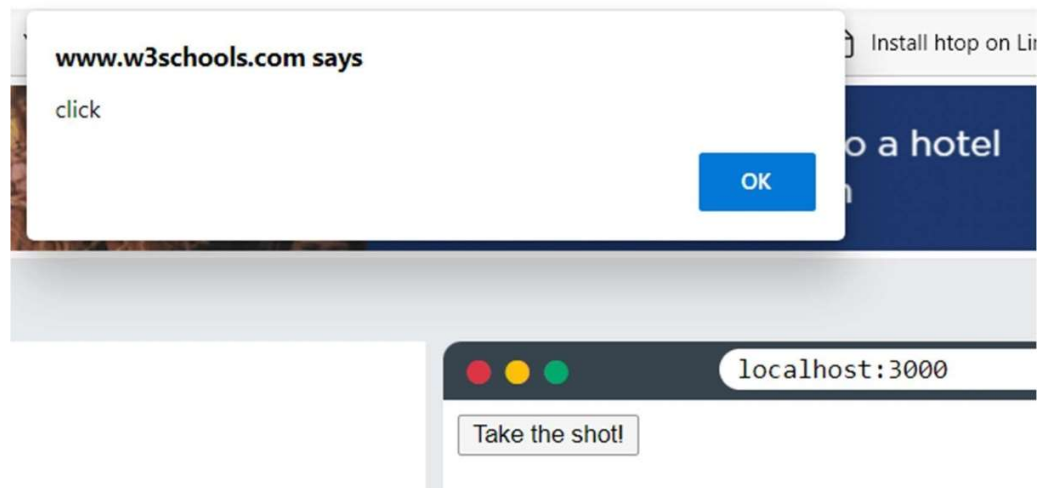
#### Output

Before:



After:

me=demo2\_react\_events\_event



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

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

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

#### Creating React.js app

**Step1:-** Install the npm using Command

**“npm insatall”**

**Step2:-** Create app using below command

**“npx create-react-app projectName”**

**Step3:-** After completed the installation go to the directory

**"cd projectName"**

**Step4:-** Create Hello.js file in src folder

#### Testing single page application

##### Hello.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> Login to GPT Bagepalli Website </h1>
    </div>
    { /* Calling to the methods */ }
    <div className="messages">
      {successMessage()}
    </div>

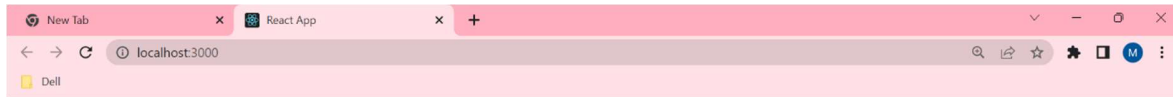
    <form>
      <fieldset>
        { /* Labels and inputs for form data */ }
        <label className="label">Name</label>
        <input onChange={handleName} className="input" value={name} type="text" /><br></br>
        <label className="label">Email</label>
        <input onChange={handleEmail} className="input" 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

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

## Index.js

```
import Home from './Home';  
<Home/>
```



## Login to GPT Bagepalli Website

Name

Email

Password

Submit

#### 14. Demonstaration of React-context.

1. Create a React app.
2. Create new folder components
3. Create new file ComponentA.js in components folder write following code and save it:

```
import React ,{Component} from 'react'
import ComponentB from './ComponentB'
class ComponentA extends Component{
  render(){
    return <Component B />
  }
}

export default ComponentA
```

4. Create new file ComponentB.js in components folder write following code and save it: import React ,{Component} from "react";

```
import ComponentC from './components/ComponentC'
class ComponentB extends Component{
  render(){
    return <ComponentC />
  }
}

export default ComponentB;
```

5. Create new file ComponentC.js in components folder write following code and save it:

```
import React ,{Component} from 'react'
class ComponentC extends Component{
  render(){
    return(
      <div>
        Component C
      </div>
    )
  }
}

export default ComponentC
```

6. Goto Visual Studio create a new file useContext.js in src folder and write following code and save it:

```
import React from 'react'
const UserContext=React.createContext() const
UserProvider=UserContext.Provider const
UserConsumer=UserContext.Consumer
export {UserProvider,UserConsumer }
```

7. Go to App.js and edit code as following and save it.

```
import React ,{Component} from "react"
import './App.css'
//import ComponentA from './components/ComponentA' import {
UserProvider } from "./components/UserContext";import
ComponentC from "./components/ComponentC"; class App
extends Component {
render(){
return(
<div className="App">
<UserProvider value="Irfan">
<ComponentC/>
</UserProvider>
</div>
)
}
}
export default App
```

8. Go to ComponentC.js and edit code as following and save it.



```
import React, {Component} from "react";

import { UserConsumer } from "./UserContext";

class ComponentC extends Component{ render(){

  return(
    <UserConsumer>
    (username) =>{
      return <div> Hello {username}</div>
    }
  )
}
}
export default ComponentC
```

8. Go to View in Visual Studio and Open terminal
9. Go into the folder using command `cd foldername`
10. To run type command `npm start`.

### Output



### Output With Default value



## 15. Implement navigation using reactrouter

### 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 />} />
      </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**

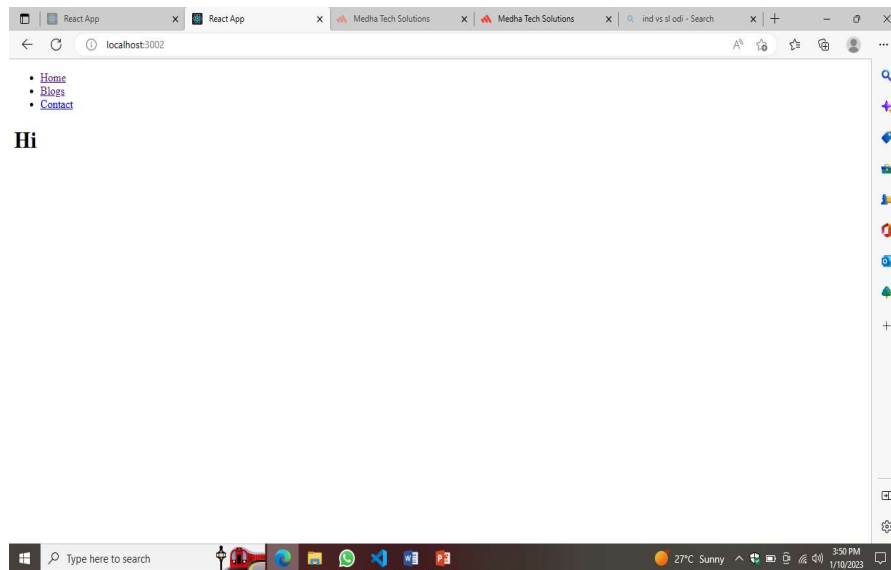
```
import { Outlet, Link } from "react-router-dom";  
const Layout = () => {  
  return (  
    <div>  
      <nav>  
        <ul>  
          <li>  
            <Link to="/">Home</Link>  
          </li>  
          <li>  
            <Link to="/blogs">Blogs</Link>  
          </li>  
          <li>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



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

### App.js

```
import Header from "./Header";

import './App.css'

import Products from "./Product";

import { useState } from "react";

import CartList from "./CartList";

function App() {

  const [product, setproduct] = useState([

    {

      url: 'https://th.bing.com/th/id/OIP.KJ863TFexg2ux2Lm6VxfdgHaHa?w=209&h=209&c=7&r=0&o=5&d

p

=1.3&pid=1.7',

      name: 'lenovoideapad Slim 3',

      category: 'Laptop',

      seller: 'Lenovo',

      price: 57000

    },

    {

      url: 'https://tse1.mm.bing.net/th?id=OIP.Wd5h_lUDFjl4tTtuIR_HjwHaHa&pid=Api&P=0&w=300&h=3

00',

      name: 'fastrack w98',

      category: 'Watch',

      seller: 'Fastrack',

      price: 1599

    },

    {

      url: 'https://i.ytimg.com/vi/KRxNI--IxrY/maxresdefault.jpg',
```

```
name: 'mi 12pro',
category: 'Mobile',
seller: 'Mi',
price: 20000
},
{
url: 'https://cdn1.smartprix.com/rx-iN1Shi5s0-w1200-h1200/boat-rockerz-450-pro.jpg',
name: 'boAt v20',
category: 'Headset',
seller: 'boAt',
price: 999
},
{
url: 'https://rukminim1.flixcart.com/image/1408/1408/washing-machine/z/c/d/ifb-digital8-kg-direct-drive-digital-8-kg-direct-drive-original-imad2zcwczwuzge4.jpeg?q=90',
name: 'IFB Washing Machine',
category: 'Electronics',
seller: 'Electro',
price: 2000
},
])
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;
```

### header.js

```
import React from 'react'  
  
import './App.css'  
  
function Header(props) {  
  return (  
    <div className='flex shopping-cart'>  
      <div onClick={()=>props.handleShow(false)}>ShoppingCart</div>  
      <div onClick={()=>props.handleShow(true)}> Cart  
      <sup>{props.count}</sup>  
    </div>  
  </div>  
)  
}
```



```
)  
}  
  
export default Header;
```

## Product.js

```
import React from 'react'  
  
function Products({product,addToCart} ){  
  
  return (  
  
    <div className='flex'>{  
  
      product.map((productitem,productIndex)=>{  
  
        return(  
  
          <div style={{ width:'50%'}} >  
  
            <div className='product-item'>  
  
              <imgsrc={productitem.url} width="40%"/>  
  
              <p>{productitem.name} | { productitem.category}</p>  
  
              <p>{productitem.seller}</p>  
  
              <p>Rs. {productitem.price}</p>  
  
              <button onClick={()=>addToCart(productitem)}>Add Cart</button>  
  
            </div>  
  
          </div>  
  
        )  
  
      })  
  
    }  
  
    </div>  
  
  )  
  
}  
  
export default Products;
```

## CartList.js

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

import './App.css'

function CartList({cart}) {

  const [CART,setCART]= useState([])

  useEffect(() => {

    setCART(cart)

  }, [cart])

  return (

    <div>

      {

        CART?.map((cartitem,cardindex)=>{

          return(

            <div>

              <imgsrc={cartitem.url}width={60}/>

              <span> {cartitem.name} </span>

              <button onClick={()=>{

                const _CART= CART.map((item,index)=>{

                  returncardindex ===index? {...item,quantity:item.quantity>0?item.quantity1:0}:item

                })

                setCART(_CART)

              }}>

                - </button>

              <span> {cartitem.quantity} </span>

              <button onClick={()=>{

                const _CART= CART.map((item,index)=>{
```

```
return cartIndex === 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;
```



Add Cart



fastrack w98 | Watch

Fastrack

Rs.1599

Add Cart



mi 12pro | Mobile

Mi

Rs.20000

Add Cart



boAt v20 | Headset

boAt

Rs.999

Add Cart



IFB Washing Machine | Electronics

Electro

Rs.2000

Add Cart

## 17. Spring Boot Application to demonstrate Autowiring, Qualifier and Bean Scope.

### Autowiring in Spring:

- ✓ Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
- ✓ Autowiring can't be used to inject primitive and string values. It works with reference only.

### Autowiring Modes

1. **No:** It is the default autowiring mode. It means no autowiring by default.
2. **byName:** The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
3. **byType:** The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.

```
package com.example.demo;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.support.ClassPathXmlApplicationContext;

@SpringBootApplication public class
NewApplication { public static void main(String[]
args) {

    System.out.println("hello");

    ClassPathXmlApplicationContext context=new ClassPathXmlApplicationContext("stud.xml");
    student s= (student) context.getBean("stud");
    System.out.println(s);

    }

}
```

## **Student.java**

```
package com.example.demo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class student {
    private String rollno;
    private String name;
    @Autowired
    private Address address;

    public student() {
    }
    public student(String rollno, String name, Address address) {
super();
        this.rollno = rollno;
        this.name = name;
        this.address = address;

        System.out.println("this is get and set level");
    }
    public String getRollno() {
return rollno;
    }
    public void setRollno(String rollno) {
this.rollno = rollno;
    }
    public String getName() {
return name;
    }
    public void setName(String name) {
this.name = name;
    }
    public Address getAddress() {
return address;
    }
}
```

```
public void setAddress(Address address) {
this.address = address;

} @Override
public String toString() {
return "student{" + "rollno=" + rollno + "\n" + ", name=" + name + "\n" + ", address=" + address + '}';
}
}
```

### **Address.java**

```
public class Address {
private String place;
private String city;    public
Address() {
}
public Address(String place, String city) {
this.place = place;
this.city = city; }

public String getPlace() {
return place;
}
public void setPlace(String place) {
this.place = place;
}
public String getCity() {
return city;
}
public void setCity(String city) {
this.city = city;    }
@Override
public String toString() {
return "Address{" +
        "place=" + place + "\n" +
        ", city=" + city + "\n" +
        '}';
}
}
```



✚No:

**Step1:** Go to resource> stud.xml in that add line **<context:annotation-config/>** before bean below code specified:

### Stud.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
">
<context:annotation-config/>
<bean class="com.example.demo.Address" name="address1">
<constructor-arg value="Holalkere"/>
<constructor-arg value="chitradurga"/>
</bean>
<bean class="com.example.demo.Address" name="address">
<constructor-arg value="Banglore"/>
<constructor-arg value="karnataka"/>
</bean>
<bean class="com.example.demo.student" name="stud" >
</bean>
</beans>
```

**Step 2:** after that Go to java>com.example>student in that write a annotation(**@Autowired**) in that file where you the property like this:

```
public class student {
    private String rollno;
```

```
private String name;  
@Autowired private Address  
address;
```

**Step 3:** To see Output run the Main class Output:



```
"C:\Program Files\Java\jdk-19\bin\java.exe" ...  
hello  
15:29:13.919 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refi  
15:29:14.196 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 8  
15:29:14.236 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.290 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.292 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.295 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.297 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.309 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.340 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
15:29:14.341 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Ci  
student{rollno='null', name='null', address=Address{place='Banglore', city='karnataka'}}  
  
Process finished with exit code 0
```

🚀By name:

**Step 1:** change only Stud .xml file like this `<bean class="com.example.demo.student" name="stud" autowire="byName">`and remove the annotation in Student class

### Stud.xml


```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns:context="http://www.springframework.org/schema/context"  
xmlns:p="http://www.springframework.org/schema/p"  
xmlns:tx="http://www.springframework.org/schema/tx"  
xsi:schemaLocation="http://www.springframework.org/schema/beans  
http://www.springframework.org/schema/beans/spring-beans.xsd  
http://www.springframework.org/schema/context  
http://www.springframework.org/schema/context/spring-context.xsd  
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
```

```
">
<bean class="com.example.demo.Address" name="address">
<constructor-arg value="Holalkere"/>
<constructor-arg value="chitradurga"/>
</bean>
<bean class="com.example.demo.Address" name="temp">
<constructor-arg value="Banglore"/>
<constructor-arg value="karnataka"/>
</bean>
<bean class="com.example.demo.student" name="stud" autowire="byName">
</bean>
</beans>
```

**Step 2:** To see Output run the Main class Output:



```
springbootapplication
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
hello
16:54:33.689 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.spr
16:54:34.028 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 8 bean definition
16:54:34.067 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.133 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.136 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.139 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.141 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.156 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.193 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
16:54:34.193 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared i
student{rollno='null', name='null', address=Address{place='Holalkere', city='chitradurga'}}
Process finished with exit code 0
```

 **By type:**

**Step1:** change only Stud .xml file like this `<bean class="com.example.demo.student" name="stud" autowire="byType">`

### Stud.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"
">

<bean class="com.example.demo.Address" name="temp">
```

```
<constructor-arg value="Banglore"/>

<constructor-arg value="karnataka"/>

</bean>

<bean class="com.example.demo.student" name="stud" autowire="byType">

</bean>

</beans>
```

**Step 2:** to see output run Main class

Output



```
SpringBootApplication
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
hello
16:56:20.212 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refr
16:56:20.469 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 2
16:56:20.526 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Cr
16:56:20.563 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Cr
student{rollno='null', name='null', address=Address{place='Banglore', city='karnataka'}}

Process finished with exit code 0
```

**Qualifier:** There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the **@Qualifier** annotation along with **@Autowired** to remove the confusion by specifying which exact bean will be wired. Following is an example to show the use of **@Qualifier** annotation.

**Step1:** change Stud.xml to like this

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
```

<http://www.springframework.org/schema/context/spring-context.xsd>

<http://www.springframework.org/schema/tx>    <http://www.springframework.org/schema/tx/spring-tx.xsd>

```
">
<context:annotation-config/>
<bean class="com.example.demo.Address" name="address">
<constructor-arg value="Holalkere"/>
<constructor-arg value="chitradurga"/>
</bean>
<bean class="com.example.demo.Address" name="temp">
<constructor-arg value="Banglore"/>
<constructor-arg value="karnataka"/>
</bean>
<bean class="com.example.demo.student" name="stud" >
</bean>
</beans>
```

**Step2:** Add annotations in student .java class like as below code:

### Student.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class student {
    private String rollno;
    private String name;

    @Autowired
    @Qualifier("temp")
    private Address address;

    public student() {
    }
    public student(String rollno, String name, Address address) {
        super();
        this.rollno = rollno;
        this.name = name;
        this.address = address;

        System.out.println("this is get and set level");
    }
    public String getRollno() {
        return rollno;
    }
}
```

```
    public void setRollno(String rollno) {
this.rollno = rollno;
    }
    public String getName() {
return name;
    }
    public void setName(String name) {
this.name = name;
    }
    public Address getAddress() {
return address
    }
    public void setAddress(Address address) {
this.address = address;
    }
    @Override    public String
toString() {    return
"student{" +
        "rollno=" + rollno + "\" +
        ", name=" + name + "\" +
        ", address=" + address +
        "\"";
    }
}
```

**Bean Scope:** When you start a Spring application, the Spring Framework creates beans for you. These Spring beans can be application beans that you have defined or beans that are part of the framework. When the Spring Framework creates a bean, it associates a scope with the bean. A scope defines the runtime context within which the bean instance is available.

In Spring, a bean can be associated with the following scopes:

- oSingleton
- oPrototype

Main class for Bean Scope: Main.java

```
import org.springframework.boot.autoconfigure.SpringBootApplication; import
org.springframework.context.support.ClassPathXmlApplicationContext;
@SpringBootApplication public class NewApplication {
    public static void main(String[] args) {
```

```
        System.out.println("hello");
ClassPathXmlApplicationContext context=new
ClassPathXmlApplicationContext("stud.xml");
student s= (student) context.getBean("stud");
        System.out.println(s);
        System.out.println("=====");
        System.out.println(s.hashCode());
        System.out.println("=====");
context.getBean("stud");
        System.out.println(s1.hashCode());
```

student s1= (student)

### 1. The Singleton Bean Scope:

**Step1:** change Stud.xml like as the below code:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
">
<context:annotation-config/>
<bean class="com.example.demo.Address" name="address">
<constructor-arg value="Holalkere"/>
<constructor-arg value="chitradurga"/>
</bean>
<bean class="com.example.demo.student" name="stud" >
</bean>
</beans>
```

**Step2:** To see out put Run the Main class

### Output:

```
18:00:11.512 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
18:00:11.515 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
18:00:11.532 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
18:00:11.570 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory
student{rollno='null', name='null', address=Address{place='Holalkere', city='chitradurga'}}
=====
1523457748
=====
1523457748

Process finished with exit code 0
```

## 2. The Prototype Bean Scope:

**Step1:** Stud.xml file add to been **scope="prototype"** like as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
">
<context:annotation-config/>
<bean class="com.example.demo.Address" name="address">
<constructor-arg value="Holalkere"/>
<constructor-arg value="chitradurga"/>
</bean>
<bean class="com.example.demo.student" name="stud" scope="prototype">
</bean>
</beans>
```

**Step2:** To see output run the Main class



**Output:**

```
18:01:29.066 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:01:29.070 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:01:29.072 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:01:29.094 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
student{rollno='null', name='null', address=Address{place='Holalkere', city='chitradurga'}}
=====
1523457748
=====
370370379

Process finished with exit code 0
```

## 18. Create REST controller for CRUD operations with MYSQL

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

Or Create a springboot project in spring.io with required dependencies

**Name:** week8 or any another name you want

**Dependencies:** Spring Web, Spring Data JPA, MySQL Driver

**Step 2:** Click on File ->Import ->Maven->Existing Maven project

And browse to your extracted project and click on create

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

**Step4:** Create Student.java class under entity package, Mycontroller.java under controller package and **StudentDao.java** interface under repository package

**Write the following Code**

**Student.java**

**package** com.example.week8.entity;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.Table;

@Entity

@Table(name="student")

**public class** Student {

    @Id

    @GeneratedValue(strategy=GenerationType.*AUTO*)

**int** regNo;

    String Name;

**public int** getRegNo() {

**return** regNo;

    }

**public void** setRegNo(**int** regNo) {

**this**.regNo = regNo;

    }

**public** String getName() {

**return** Name;

    }

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

        Name = name;

    }

**public** Student(**int** regNo, String name) {

**super**();

```
        this.regNo = regNo;
        Name = name;
    }
    public Student() {

    }
    @Override
    public String toString() {
        return "Student{" +
            "regno=" + regNo +
            ", name=" + Name + "\" +
            '}'";
    }
}
```

## StudentDao.java

```
package com.example.week8.repository;

import org.springframework.data.jpa.repository.JpaRepository;
public interface StudentDao extends JpaRepository<com.example.week8.entity.Student,Integer> {
}
```

## Mycontroller.java

```
package com.example.week8.controller;
import com.example.week8.entity.Student;
import com.example.week8.service.StudentService;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
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.RestController;

@RestController
public class Mycontroller {
    @Autowired
    private StudentService studentservice;
    @GetMapping("/")
    public String show1() {
        return("<h1>Welcome to GPT Bagepalli</h1>");
    }
}
```

```
}

@GetMapping("student")
private List<Student> show() {
    return this.studentservice.Display();
}
@PostMapping("student")
private Student show2( @RequestBody Student st)
{
    return this.studentservice.addStudent(st);
}
@PutMapping("student")
private Student show3(@RequestBody Student st) {
    return this.studentservice.updateStudent(st);
}
@DeleteMapping("student/{studentregno}")
private void show4(@PathVariable int studentregno){
    this.studentservice.deleteStudent(studentregno);
}
}
```

**Step5:** Create a Package with the name service com.example.week8.service

**Step4:** Create StudentService.java and StudentServiceImplement.java classes under service package,

**Write the following Code**

### **StudentService.java**

```
package com.example.week8.service;
import com.example.week8.entity.Student;

import java.util.List;
public interface StudentService {
    public List<Student> Display();
    public Student addStudent( Student st);
    public Student updateStudent(Student st);
    public void deleteStudent(int studentregno);
}
```

### **StudentServiceImplement.java**

```
package com.example.week8.service;

import com.example.week8.entity.Student;
//import com.example.week8.service.StudentService;
import com.example.week8.repository.StudentDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
//import java.util.ArrayList;
import java.util.List;
@Service
public class StudentServiceImplement implements StudentService {
    @Autowired
    private final StudentDao studentDao;
    StudentServiceImplement(StudentDao studentDao) {
        this.studentDao = studentDao;
    }
    @Override
    public List< Student>Display() {
        return studentDao.findAll();
    }
    @Override
    public Student addStudent(Student st){
        studentDao.save(st);
        return st;
    }
    @Override
    public Student updateStudent(Student st){
        studentDao.save(st);
        return st;
    }
    @Override
    public void deleteStudent(int regno){
        Student entity= studentDao.getReferenceById(regno);
        studentDao.delete(entity);
    }
}
```

### **application.properties in src/main/resources**

```
server.port=8085
spring.datasource.url=jdbc:MySql://localhost:3306/sys
spring.datasource.username=root
spring.datasource.password=cse@136
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

### **Test created APIs with the help of Postman**

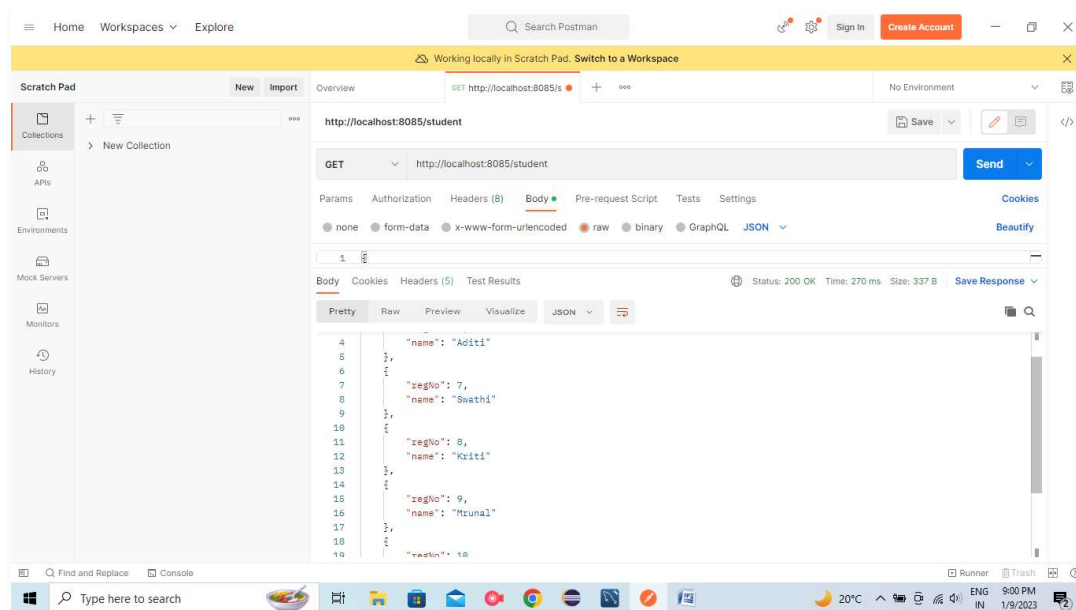
Step1: Download & Install postman from official website

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

Step2: Click on Collection and Create Collection → Add Request

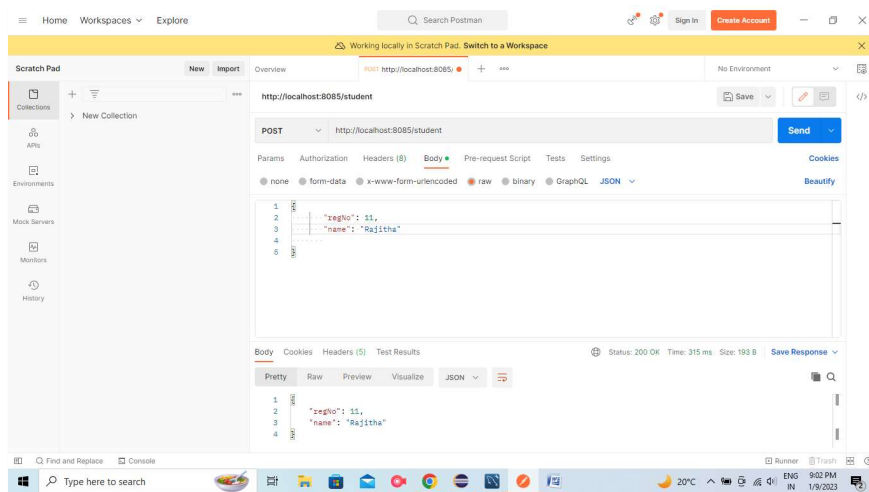
Step3: Demonstrate Get, Post, Put, Delete methods

Get: Select Get method from dropdown list and enter the URL [localhost:8085/student] → Send



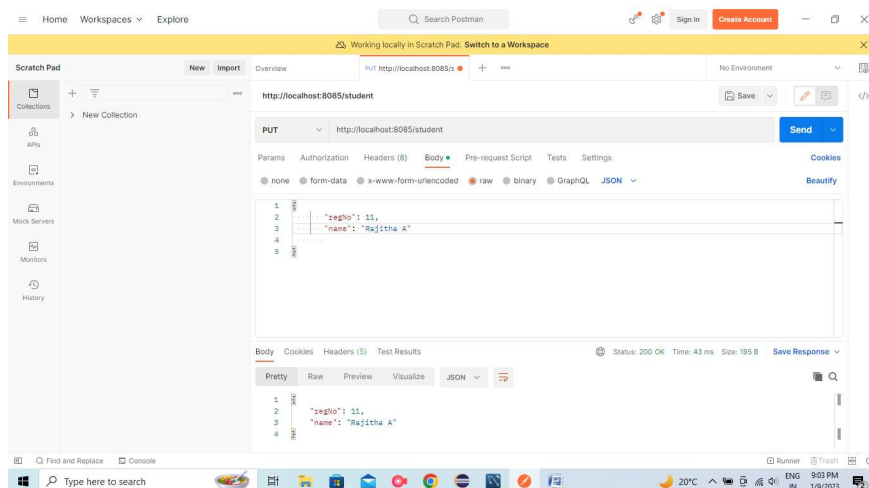
Post: Select Post method from dropdown list → Click on Body, choose raw and select JSON from dropdown list and enter the URL [localhost:8085/users] → Give the input in the form of JSON and Click on Send

## Full Stack Development- 20CS52I



Put: Select Put method from dropdown list and enter the URL [`localhost:8085/student`]

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/student/10`]

## 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 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>

## 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
```

## 22.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**

**publicvoid 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();
    }
}
```

## Week-11

- 1) Check the Docker version information

**docker -v    or    docker --version**

- 2) Check Docker images (List images)

**docker images**

- 3) Docker pull:    Pull an image or a repository from a registry

**docker pull Ubuntu**

**docker pull python**

**docker pull mysql**

**docker pull openjdk**

- 4) Docker ps    -List containers

**docker ps    or    docker ps -a**

- 5) Docker run    -Run a command in a new container

**docker run Ubuntu**

- 6) Docker exec    -Run a command in a running container

**docker exec -it gptpython python**

- 7) Docker build    -Build an image from a Dockerfile

**docker build -t stud1ex**

### How to Run ubuntu using Docker commands?

open command prompt or powershell

>> docker pull Ubuntu

>> docker ps -a

>> docker run -it Ubuntu



```
# ls
# pwd
```

### **How to Run python program using Docker commands?**

open command prompt or powershell

```
>> docker pull python
>> docker images
>> docker run --name gptpython -it -d python
>> docker exec -it gptpython python
>>> print("Hello FSD Lab ")
```

### **How to Run java program using Docker commands?**

Open command prompt or powershell

```
>> docker pull openjdk
>> docker images
>> docker run --name gptjava -it -d openjdk
>> docker ps
>> docker exec -it gptjava jshell
>>> System.out.println("Hello FSD Lab ");
```

### **How to Run MYSQL using Docker commands?**

open command prompt or powershell

```
>> docker pull mysql
>> docker images
>> docker run --name gptmysql -e MYSQL_ROOT_PASSWORD=user -d mysql
>> docker ps
>> docker exec -it gptmysql bash
Bash#>> mysql -p
Password:
mysql > show databases
mysql > use sys
mysql > show tables
```