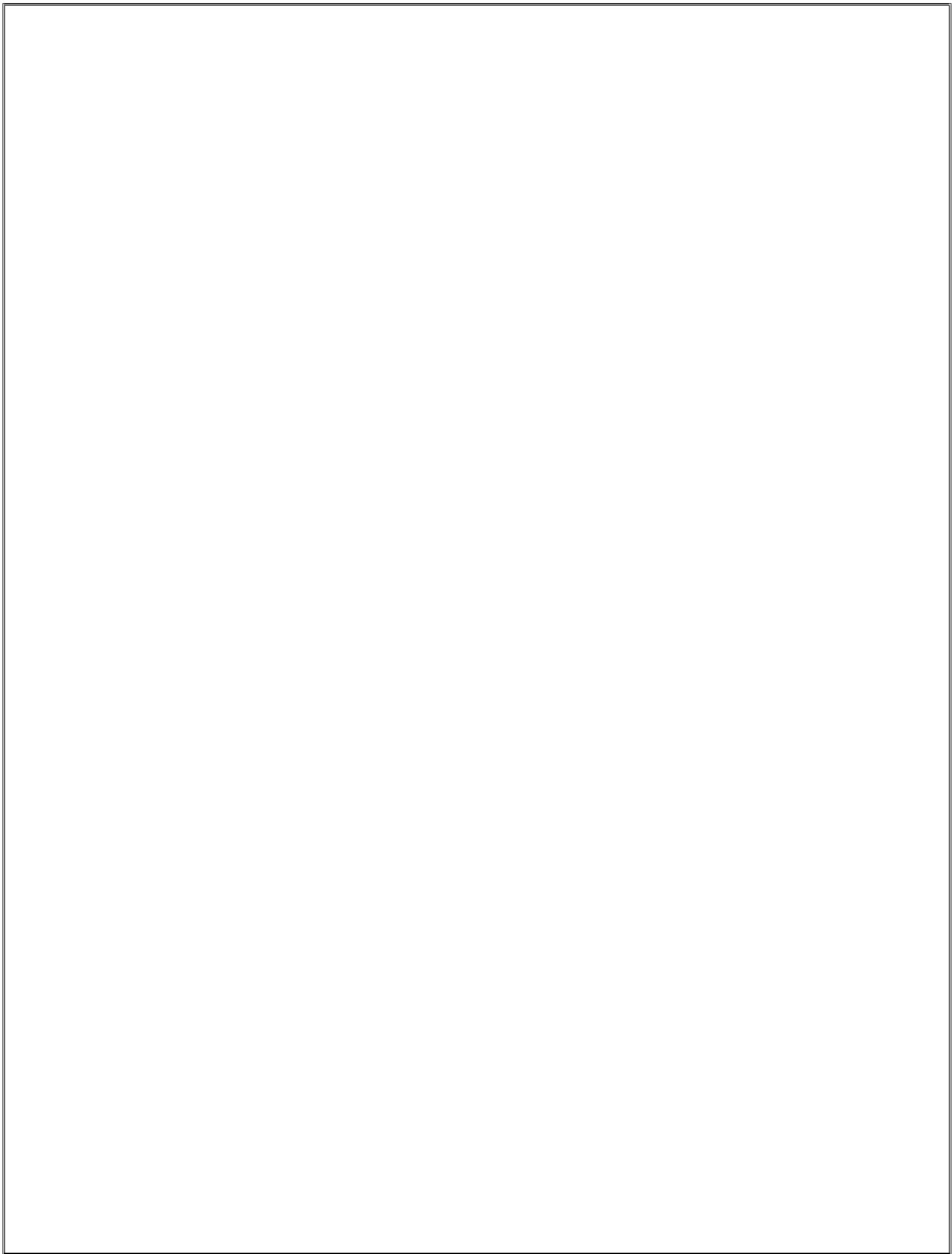


Table of Content

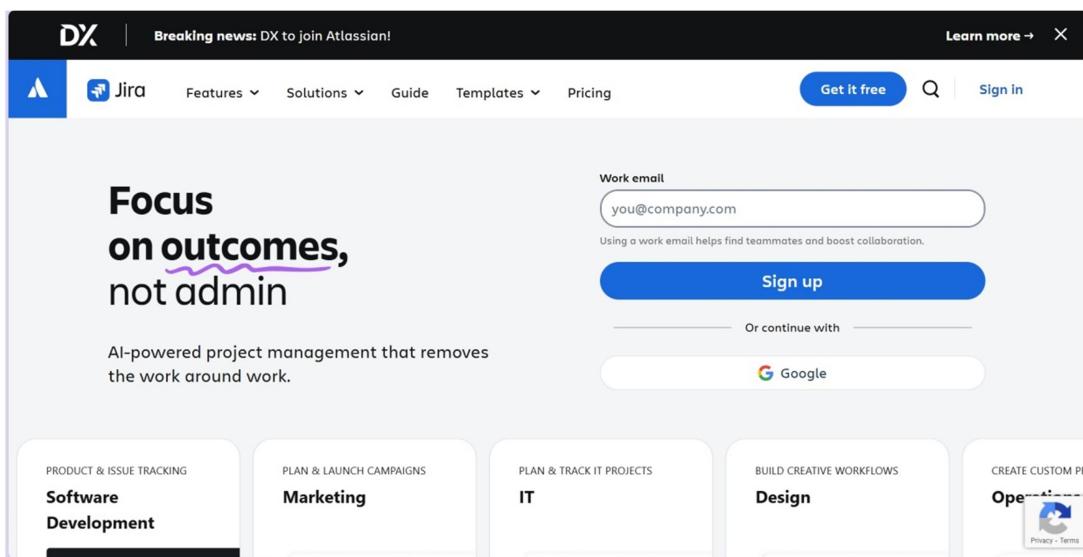
<u>Sl. No.</u>	<u>Experiments Name</u>	<u>Page No.</u>
1.	Create a Project Plan and Project Backlog and User Stories for an application using Jira Software	1
2.	Create a UI/UX Design for an application using Figma	12
3.	Preform Git and Github Operations using Cloud Version or Normal Application	19
4.	Design a Registration Page and preform Validation using Java Script	29
5.	Design a Registration page using React-JS	33
6.	Create a Product catalog like Product name, Product ID, Product Price and Product Image using React-JS	39
7.	Create a Navigation Page. Using a React-Router-Dom with multiple React pages.	44
8.	Preform CRUD Operations without using API with Spring Boot application	50
9.	Preform a CRUD Operations using API with Spring Boot application	63
10.	Preform A) Create and Drop Database, B) Create and Drop Collection, C) Insert one Insert many, D) Delete one and Delete many operations using Mongo DB shell	73
11.	Preform CRUD Operations using Mongo DB with Spring Boot application	77
12.	Create Junit test cases for CRUD operation and execute the test cases	89



1. Create a Project Plan and Project Backlog for a Project and User Stories using Jira Software

Step 1 : Login to Jira

- Open your browser and search for “Jira Login”.
- Login with you Gmail Account



Step 2: Create a new Project

- In the side menu look for Project and select “Create Project”

The screenshot shows the Jira 'For you' dashboard. On the left, a sidebar has 'For you' selected, with options: Recent, Starred, Apps, Plans, Projects (which is highlighted with a red box), Teams, and More. The main area shows 'Recent projects' with a card for 'Chatbot for Customer Su...' and a 'View all projects' link. Below this are sections for 'Worked on', 'Viewed', 'Assigned to me', 'Starred', and 'Boards'. Under 'TODAY', there are cards for '(Sample) Payment Processing Integration' and '(Sample) Update User Subscription', each with a 'Created' timestamp and user icon. At the bottom, there are links for 'Give feedback on the n...', 'Generate Monthly Invoices', and 'Subscription Management System'.

- Select “Software Development” and in that use the “Scrum” Template to start your Project

The screenshot shows the 'Project templates' section. On the left, a sidebar lists categories: 'Made for you', 'Custom templates [ENTERPRISE]', 'Software development' (which is selected and highlighted in blue), 'Service management', 'Work management', 'Product management', 'Marketing', 'Human resources', 'Finance', and 'Design'. The main area displays four template cards:

- Kanban** [LAST CREATED] (Jira): Visualize and advance your project forward using work items on a powerful board.
- Scrum** (Jira): Sprint toward your project goals with a board, backlog, and timeline.
- Top-level planning** (Jira PREMIUM): Monitor work from multiple projects, and create an easy-to-share plan for stakeholders.
- Cross-team planning**

- Select “Team-Managed Project” as the project type
- Give your project a name, and click next to Proceed

The screenshot shows the 'Add project details' form. At the top, there's a link to 'Back to project types'. The form fields include:

- Name ***: DeliveryApp
- Key * ⓘ**: DEL
- Access ***: Private
- Template**: Scrum (Jira) - Sprint toward your project goals with a board, backlog, and timeline.
- Type**: Team-managed - Control your own working processes and practices in a self-contained space.

At the bottom right are 'Cancel' and 'Create project' buttons.

Step 3: Create your Project Backlogs

- Come to the “Backlog” tab in your project
- Enable “Epic” by selecting “Show epic panel”

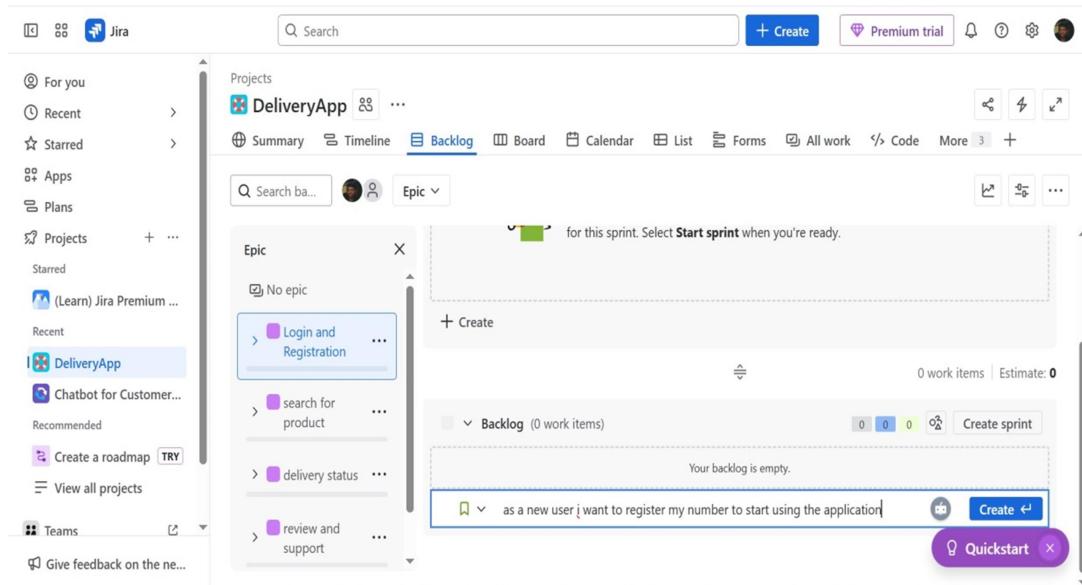
The screenshot shows the Jira software interface with the 'Backlog' tab selected in the sidebar. A tooltip for the 'Epic' button displays the message 'Your project has no Epic'. The main area shows a placeholder for an epic with a small plant icon and instructions to plan and prioritize large chunks of work.

- In the Epic column create different Epics for different Issues Ex- “Login and Registration”.

The screenshot shows the Jira software interface with the 'Backlog' tab selected. The 'Epic' column lists several epics, including 'No epic', 'Login and Registration', 'search for product', 'delivery status', and 'review and support'. The 'Backlog' section shows 0 work items.

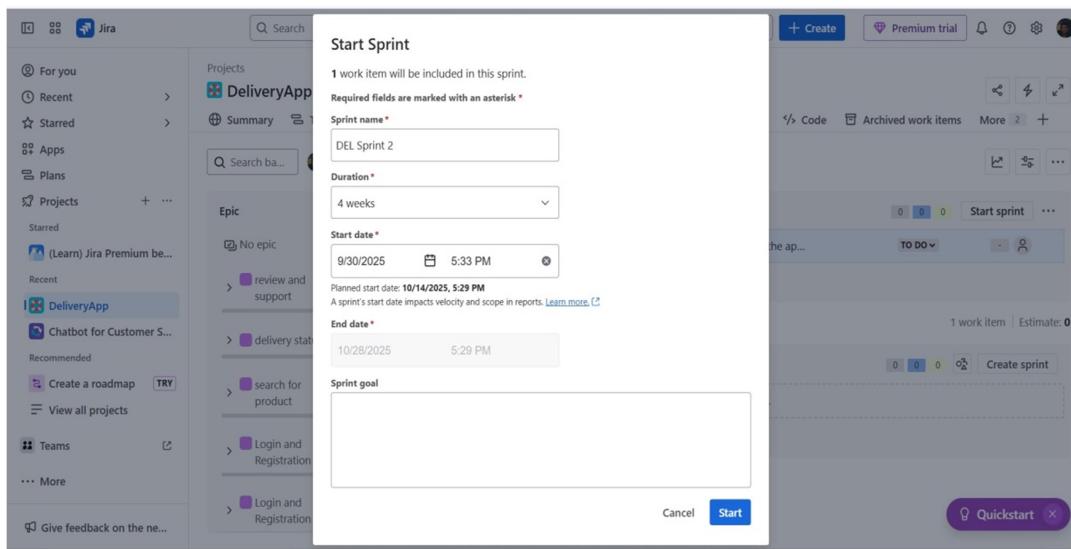
Step 4: Create User Stories

- For each epic, create related issues such as User Story, Acceptance Criteria, and Tasks by selecting appropriate issue types.
- Select a epic and create a user story related to that
- For each user story use appropriate acceptance criteria and task



Step 5: Start a Sprint

- Once issues are created, drag and drop in the sprint above.
- After dropping click on “Create a Sprint”.
- Give the sprint details like name, duration, and start date.
- Click on “Start”.

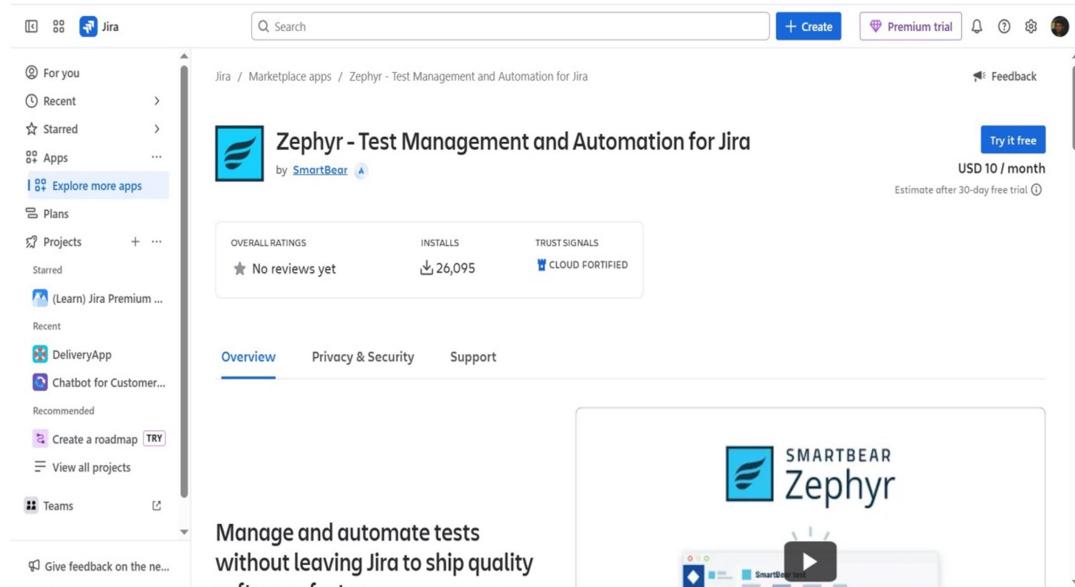


Step 6: Check the tasks status

- Come to the “Boards” tab and check the status of tasks of your current spring
- Move them “In Progress” and “Done” based on the task completion

Step 7: Add Test Cases

- In the menu bar go to “Apps” and “Explore Apps”.
- Search for “Zephyr”.
- Select “Zephyr-Test Management and Automation for Jira”
- And click on “Try for Free”.



Step 8: Add Test Cases.

- Come to “Backlogs”.
- Select your User Story.
- Select the gear symbol and add Zephyr Scale tool.
- Now it will appear in the gear menu.

The screenshot shows the Jira Backlog interface for the 'DeliveryApp' project. On the left, there's a sidebar with links like 'For you', 'Recent', 'Starred', 'Apps', 'Zephyr', 'Circle CI (CI/CD)', 'Timesheet tracking', 'Explore more apps', 'Plans', 'Projects', 'Starred', '(Learn) Jira Premium...', 'Recent', and 'Give feedback on the n...'. The main area shows an 'Epic' named 'DEL-8 as a new user i want to register my p...' under 'DEL Sprint 2 30 Sep - 28 Oct (1 work item)'. Below it is a 'Backlog' section with a message 'Your backlog is empty.' and a '+ Create' button. To the right, there's a 'Jira work item' panel for 'DEL-8' with sections for 'Description' (containing 'Zephyr' and acceptance criteria), 'Acceptance Criteria' (with bullet points for valid phone number and new number), 'Confluence content' (with a link to 'User Phone Num...'), and 'Pinned fields'.

Step 9: Create Test Cases using Zephyr Scale.

- Select Zephyr in your user story as shown in the above picture
- Click on “Create a new Test Case”.
- Give test Details like name, state, priority.

The 'Create Test Case' dialog box is open. It has the following fields:

- Name:** registration and login
- Objective:** (Rich text editor)
- Precondition:** (Rich text editor)
- Details:**
 - Status: Approved
 - Priority: Normal
 - Component: None
 - Owner: Honvith Nayaka, B.
 - Estimated run time: (dropdown menu)
- Buttons:** 'Create another test case' (link), 'Cancel', and 'Create and edit' (button)

- Click on “Create and add”.

- It automatically directs you to a new page.

The screenshot shows the Jira Test Script interface for a project named 'DeliveryApp'. The 'Test Script' tab is selected. A single step is defined: 'registration and login'. The 'Type' is set to 'Step by Step'. The 'Data type' is 'None'. To the right, there is a visual representation of the test flow with nodes and connections. The left sidebar shows various Jira navigation options like 'For you', 'Recent', 'Starred', etc.

- Click on “Add step”.
- Create a step for that particular test.
- Ex- Step = Registration, Test data = Correct phone number, Expected Result = Successful Registration.
- Create multiple steps for both positive and negative test data.

The screenshot shows the Jira Test Script interface with a completed step. The step is labeled 'registration' under 'TEST DATA' and 'correct phone number' under 'EXPECTED RESULT'. The interface includes a toolbar for editing and a bell icon for notifications. The left sidebar remains the same as the previous screenshot.

- Now come to backlogs and your test case for the selected user story will be added

- Click on the “Start button” to start testing

The screenshot shows the Jira Backlog interface for the 'G-Pay Software project'. On the left, there's a sidebar with options like Planning, Backlog, Board, etc. The main area shows a backlog for 'GP Sprint 1' (Nov 1 - Nov 29). There are two tasks listed: 'GP-2 As a new user, I want to sign up for the GPay app so that I can...' and 'GP-6 As a registered user, I want to link my bank account so that I ca...'. Both tasks are marked as 'DONE'. A note says 'Your backlog is empty.' on the right.

- Now demonstrating the completing of tasks by right clicking on tasks.

The screenshot shows the Jira Test Case Details interface for the 'DeliveryApp' project. The left sidebar shows various projects and recent activity. The main page displays a test case titled 'registration and login (DEL-T1)'. The 'Test Script' section contains a single step: 'registration' with 'TEST DATA' 'correct phone number' and 'EXPECTED RESULT' 'successful registration'. The status of the test step is 'PSS' (Pass). Other sections like 'Comment', 'Issues(0)', and 'Attachments' are also visible.

- Green color in user story indicates the completion of all tests for that user story.

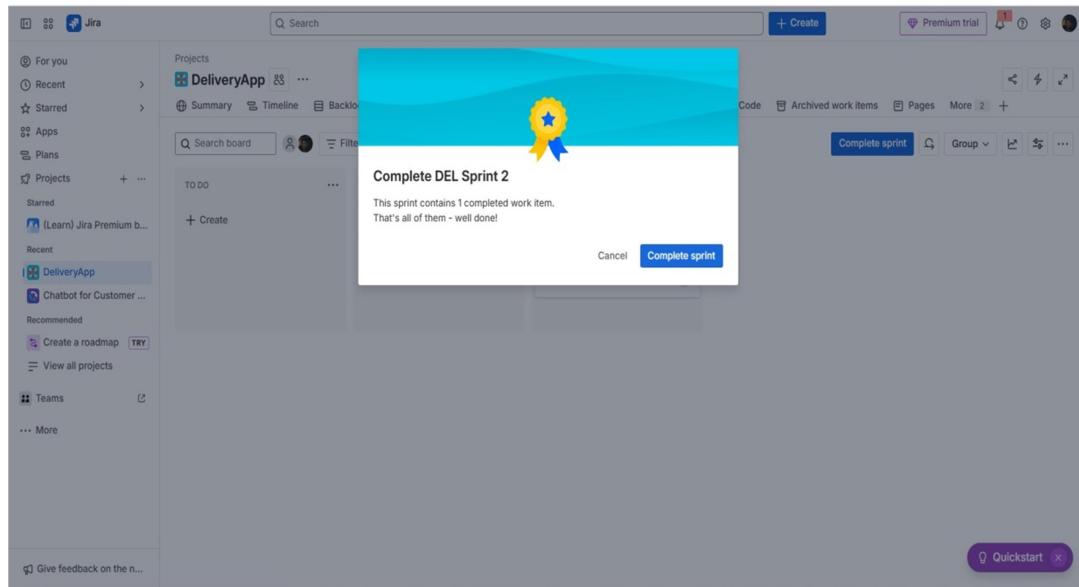
The screenshot shows the Jira Backlog screen for the 'DeliveryApp' project. On the left, the sidebar includes links for 'For you', 'Recent', 'Starred', 'Apps', 'Plans', 'Projects', 'Teams', and 'More'. The main area displays an 'Epic' section with a card for 'No epic' and several backlog items under 'review and support', 'delivery status', 'search for product', and 'Login and Registration'. A 'Backlog' section shows '0 work items' with a note 'Your backlog is empty.' To the right, there are panels for 'Acceptance Criteria' (with items like 'use a valid phone number'), 'Confluence content' (with a link to 'User Phone Nu...'), 'Zephyr' (with a test case 'DEL-T1 (1.0) registration... APPROVED'), and 'Pinned fields' (with 'Assignee' set to 'Unassigned'). A 'Details' panel shows 'Assignee' as 'Unassigned'.

Step 10: Completion of sprint

- Go to the boards tab and move your user story from “To Do” to “Done” this shows the task has been completed

The screenshot shows the Jira Board screen for the 'DeliveryApp' project. The sidebar is identical to the previous screenshot. The main board view has three columns: 'TO DO', 'IN PROGRESS', and 'DONE'. A user story card for 'as a new user I want to register my phone number to start using the application' is moved from 'TO DO' to 'DONE'. The 'DONE' column has a count of 1. A 'Complete sprint' button is visible at the top right of the board area. A 'Quickstart' button is located at the bottom right.

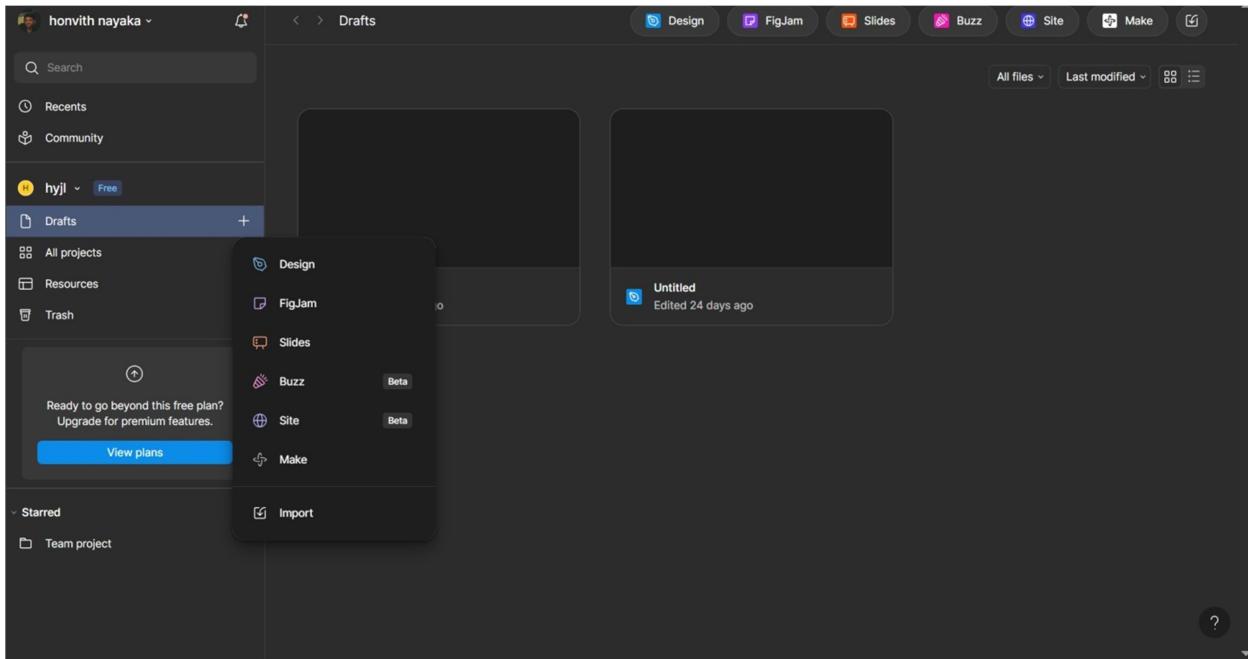
- Click on “Complete Sprint” to finish you Sprint



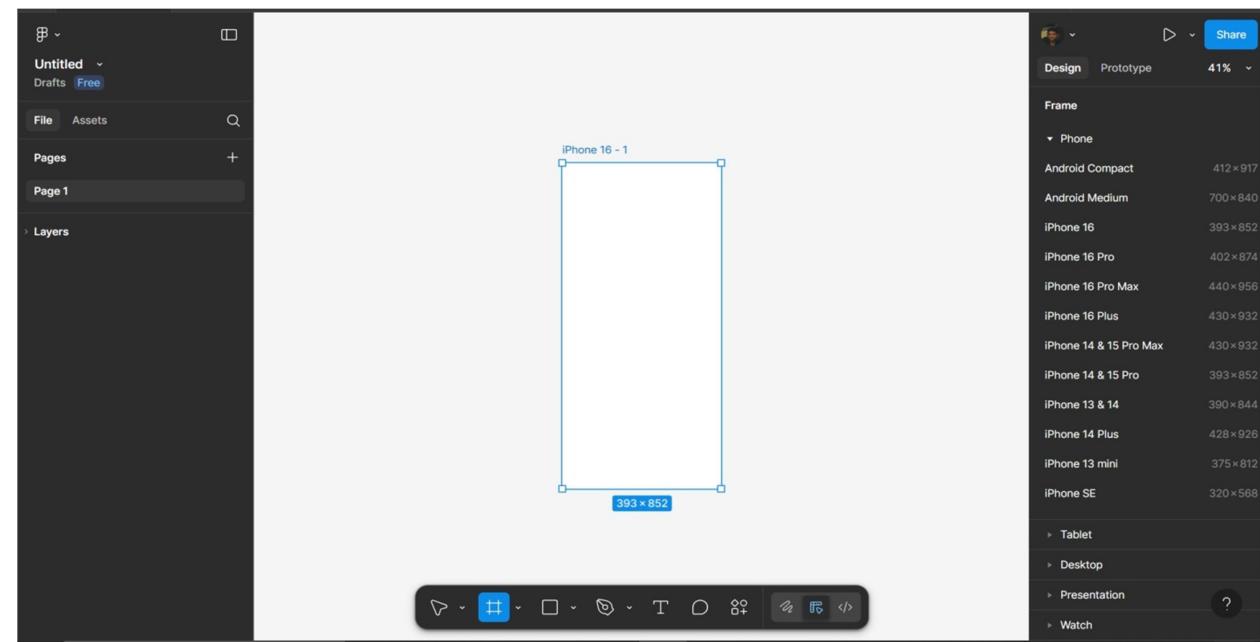
2. Create a UI/UX Design for an application using Figma

Step 1: Set up your frame

- Come to figma and in the side bar select “Draft” and “Design”

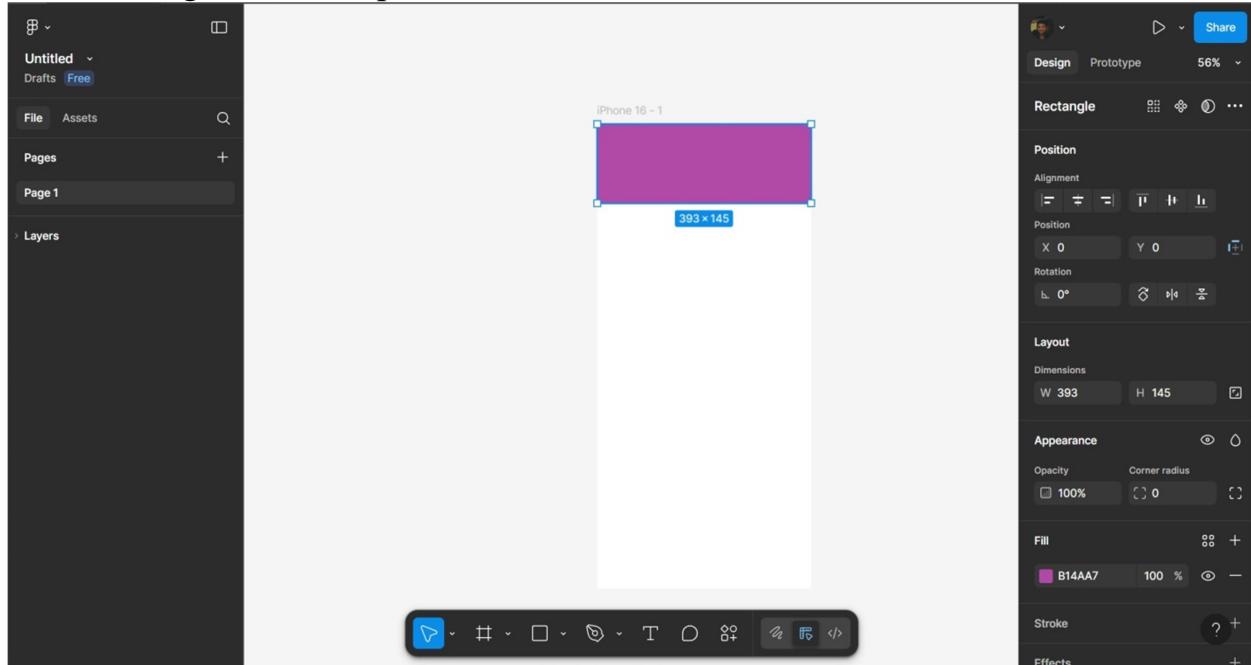


- Select “Frames” in the bottom menu and choose “iPhone 16”.

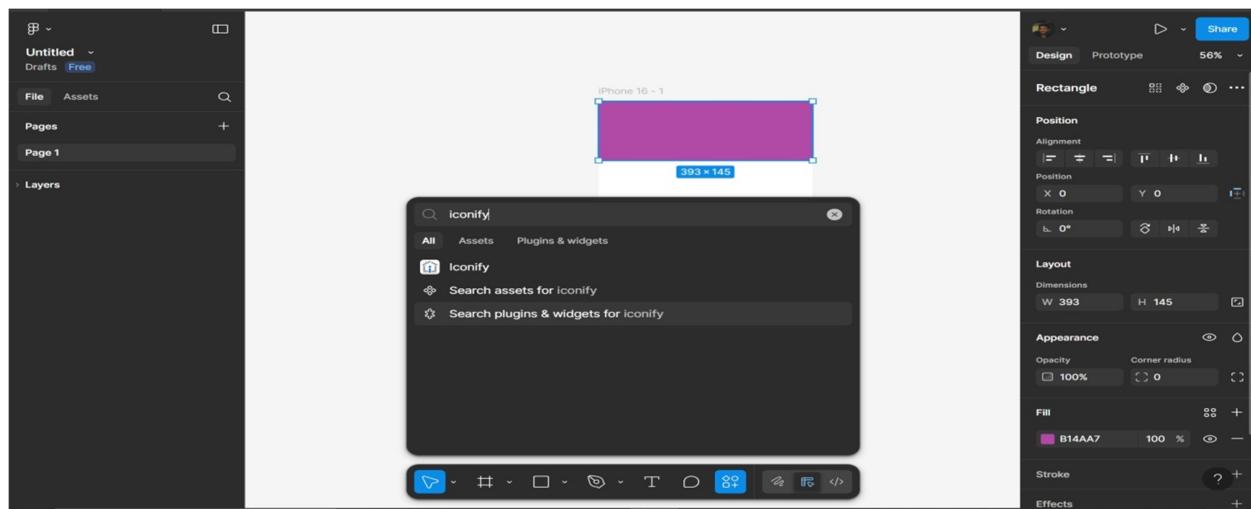


Step 2: Profile and Search Bar Section

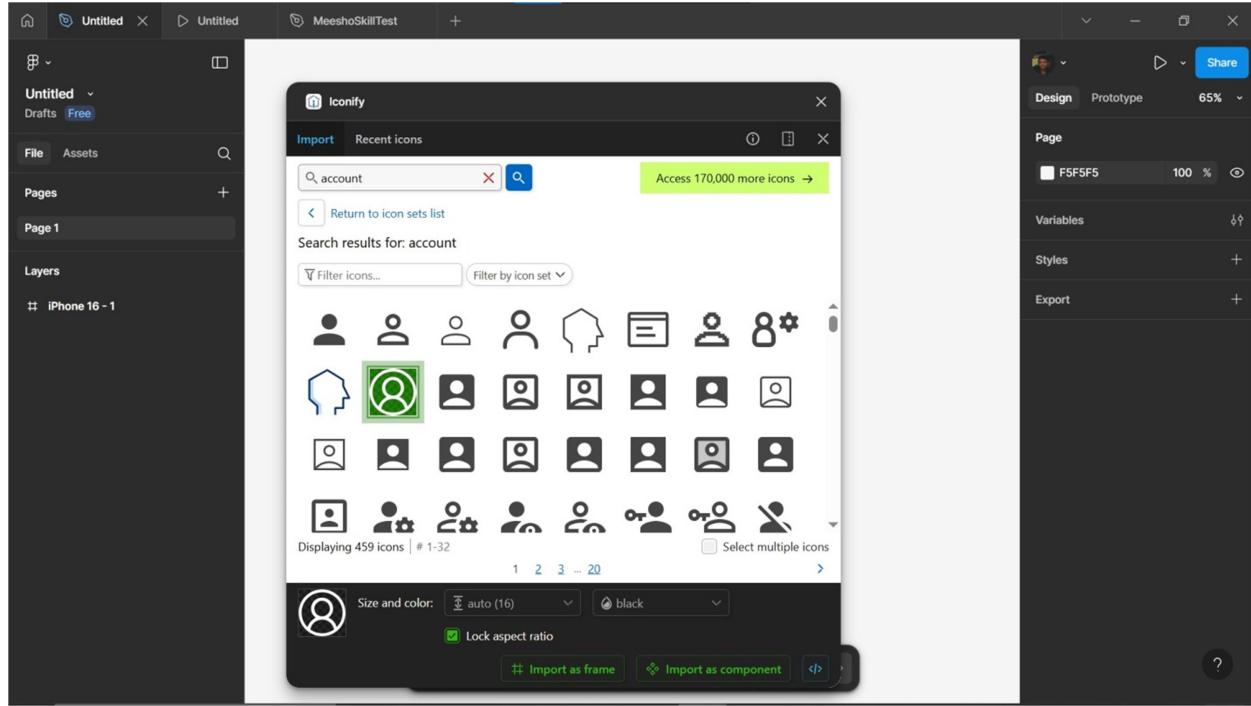
- Add a Rectangle using the “Shapes” Given in the bottom menu
- Place it on top of the Frame
- Using the “Fill” option in the side bar, add a different color



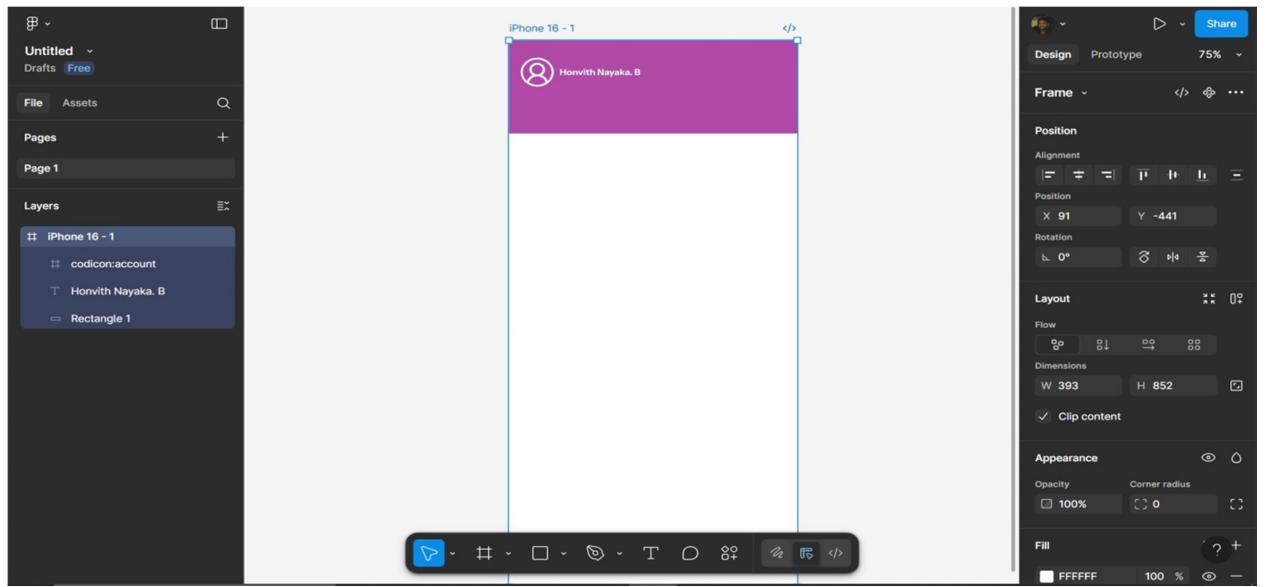
- Come to “Action” in the bottom menu and search for “Iconify” this gives you custom pre-made icons to use



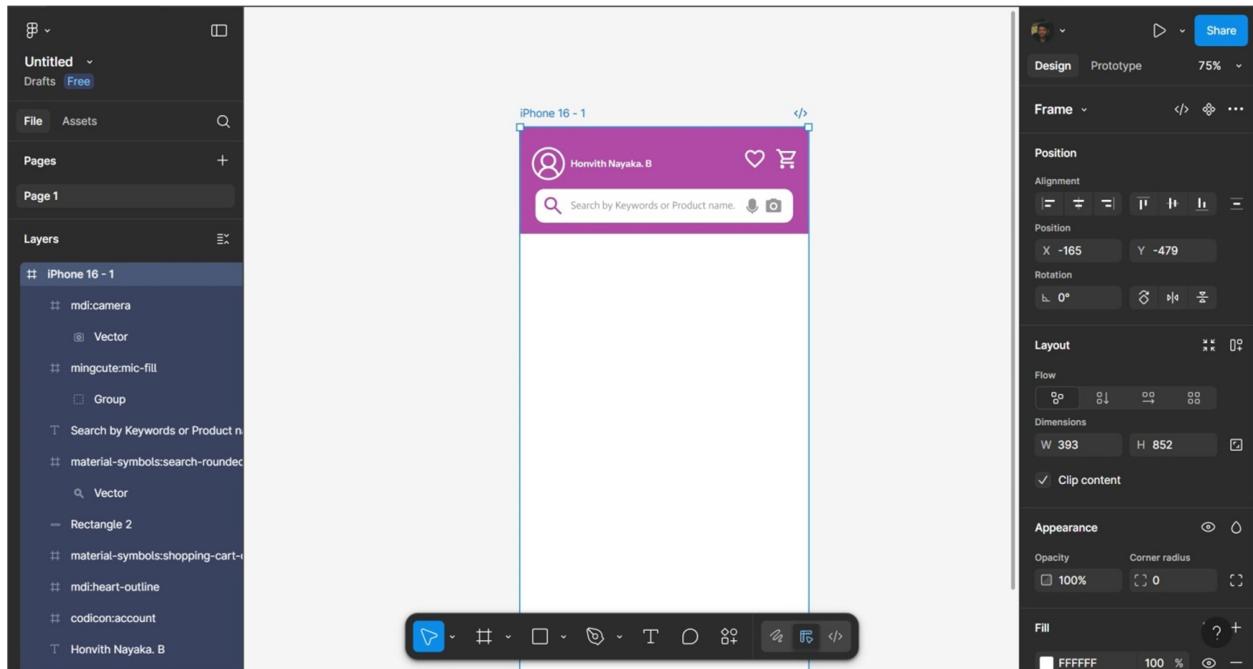
- Inside it search for “Account” and select a icon of your choice.
- And click on “Import as Frame”



- Adjust the size of and place it on top right corner of the frame
- Using the “Text” in the bottom menu add the username
- Adjust the test style in “Typography” Section given in the side bar

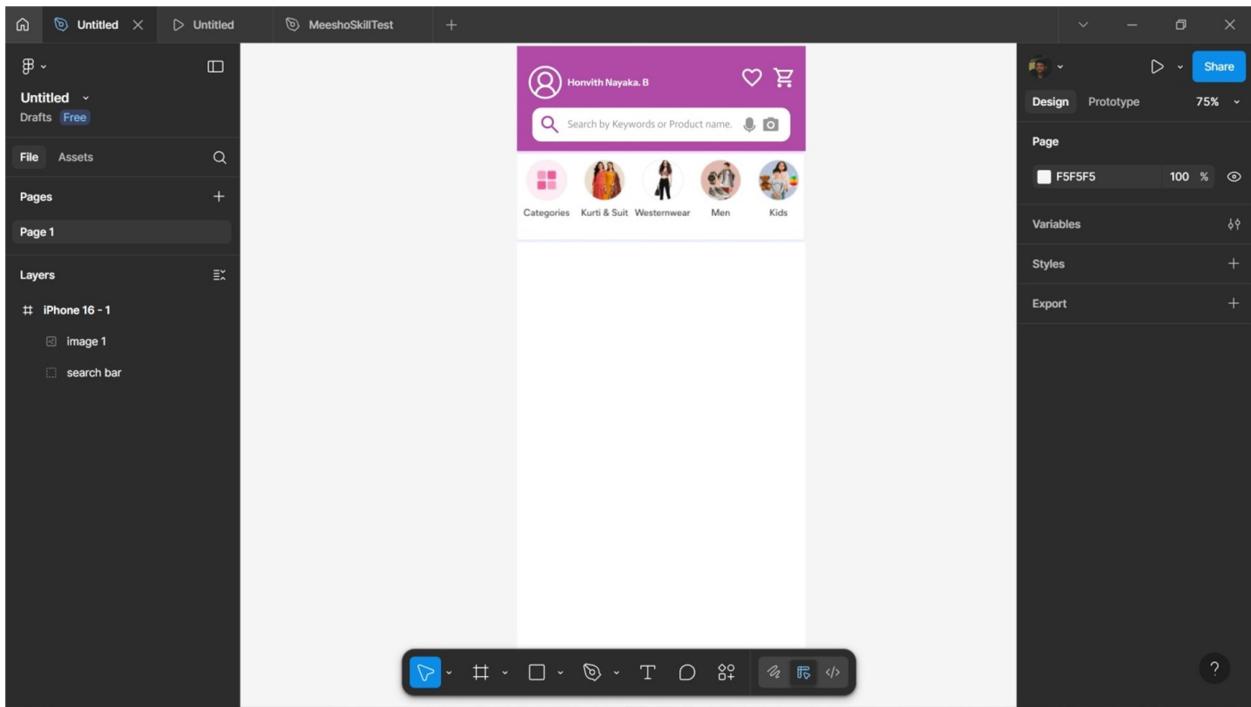


- Add a “Heart” and “Shopping cart” icons using Iconify
- Adding a search bar Draw a rectangle Below and Round its corners using the “Corner radius” in the “Appearance” section at side menu
- Add a search icon using Iconify inside the search bar
- Add a text saying “search by Keywords or Product name” inside the search bar
- Using Iconify add icons of a “mice” and “camera” also inside the search bar indicating the features



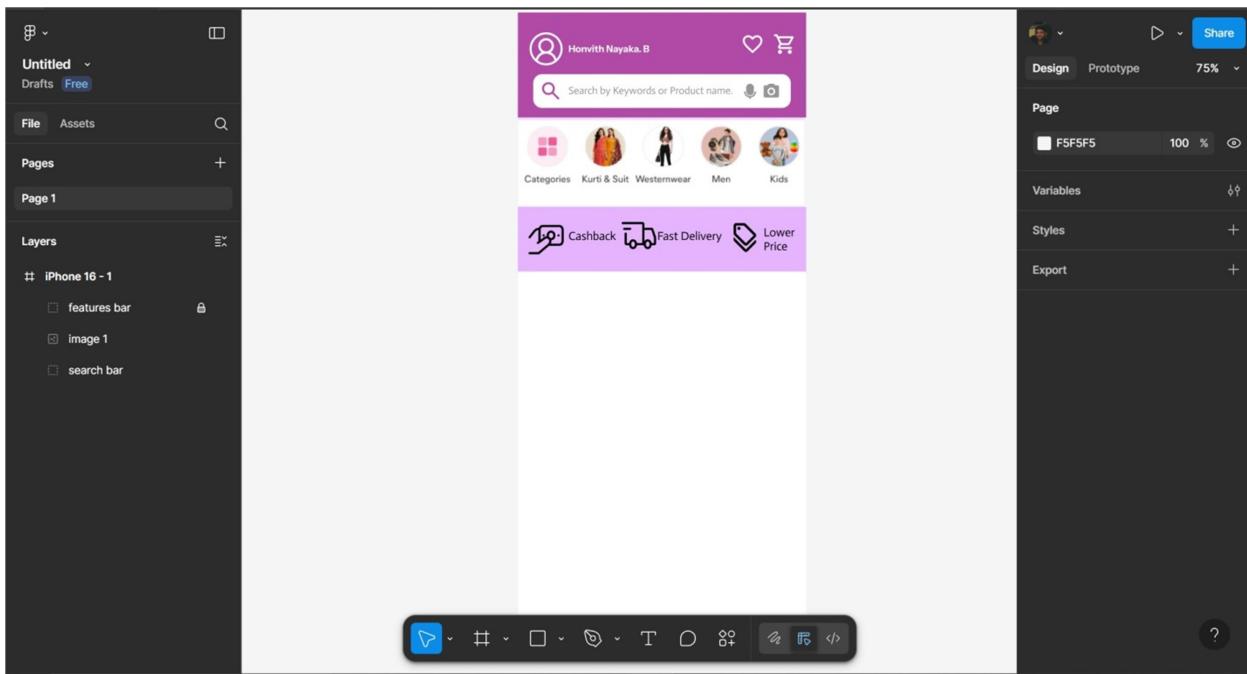
Step 3: Add a category section

- Take a images off different products from you browser and past them into your project arrange them to your liking creating the appearance of different category and name them appropriately



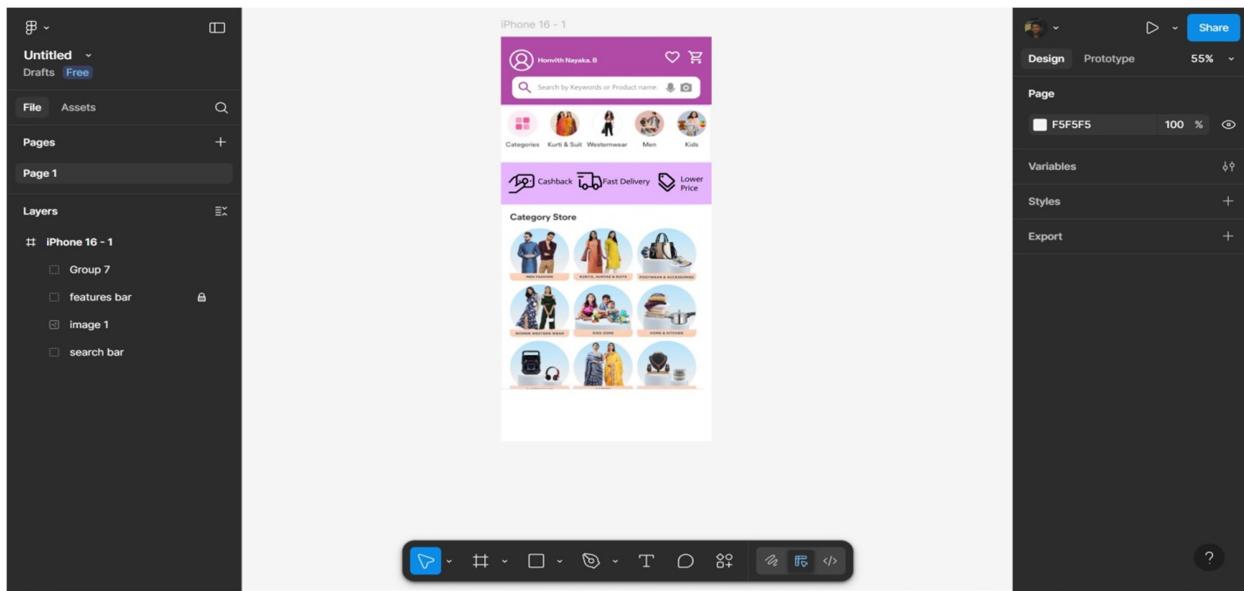
Step 4: Features Section

- Add a rectangle using the shapes in the bottom menu
- Change its color
- Add icons related to “Cash Backs”, “Fast Delivery”, and “Lowest price” and use the text tool to describe the features



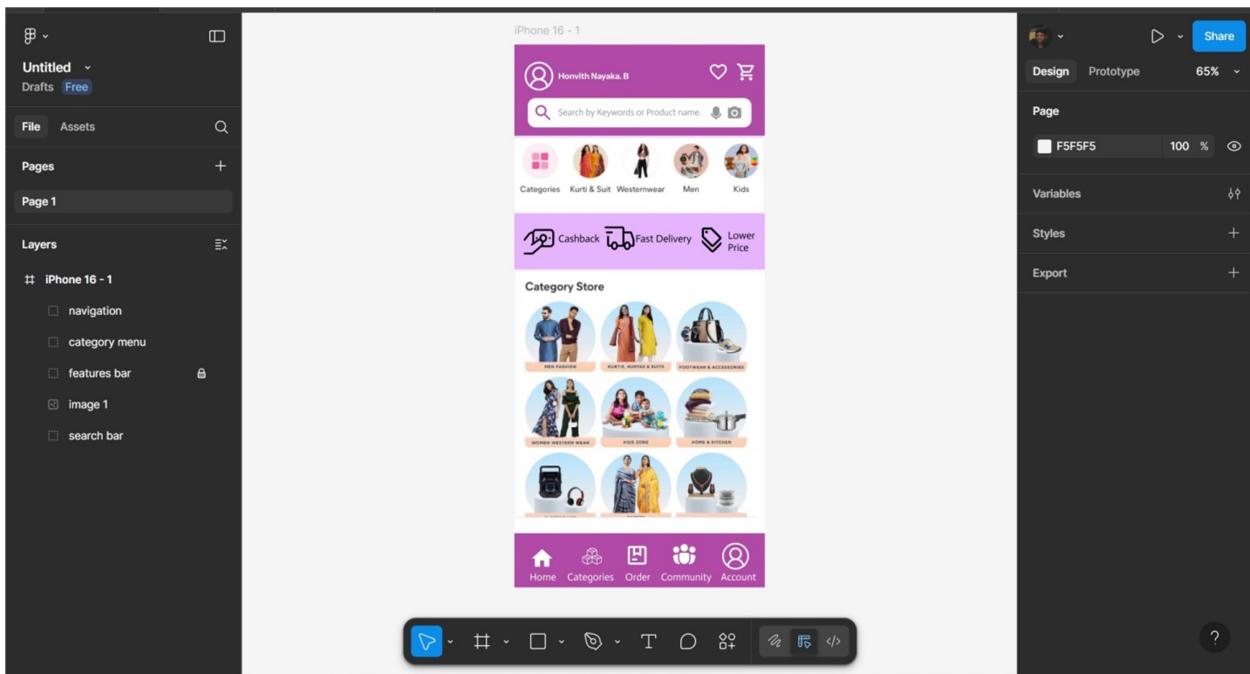
Step 5: Product Section

- Copy paste images of different products from the browser and arrange them in the format given in the picture below.
- Add related product details like “product name” and “Product price” etc.



Step 6: Navigation Bar

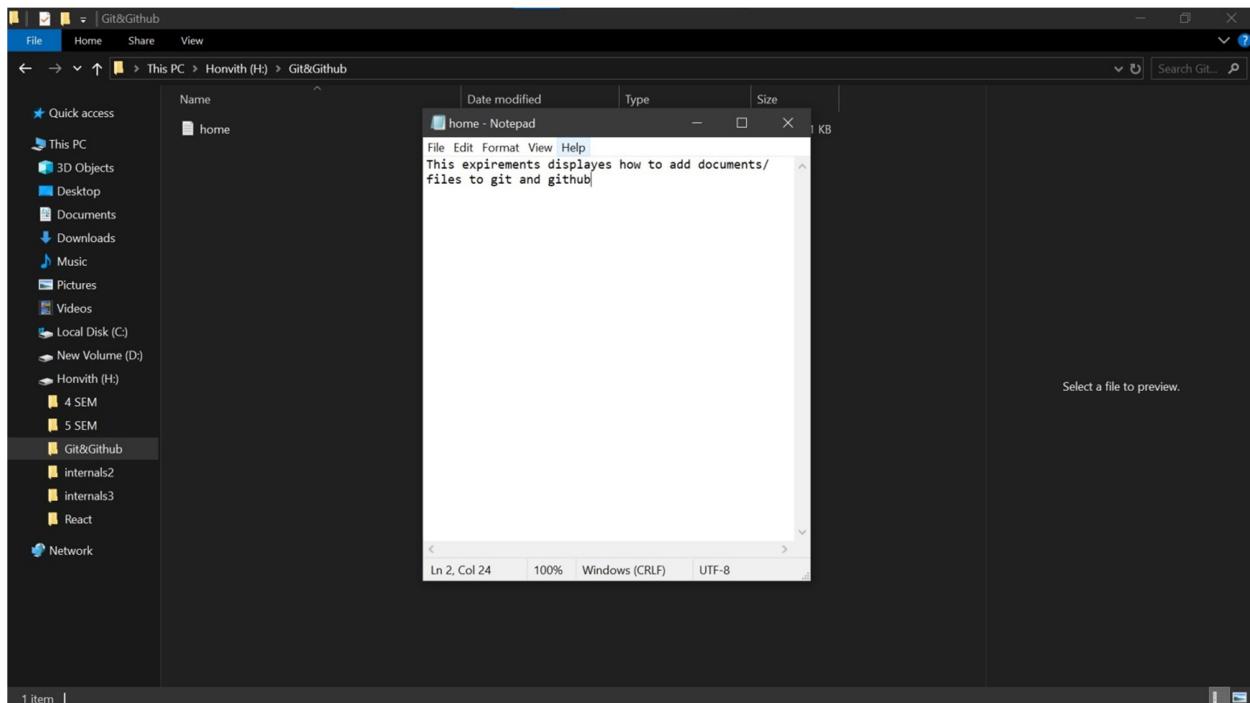
- Add a rectangle using the shapes in the bottom menu
- Change its color
- Add icons related to “Home”, “Categories”, “Orders”, ”Community”, and “Amount” and use the text tool to describe the features



3. Perform Git and Github operations using Cloud Version or Normal Application

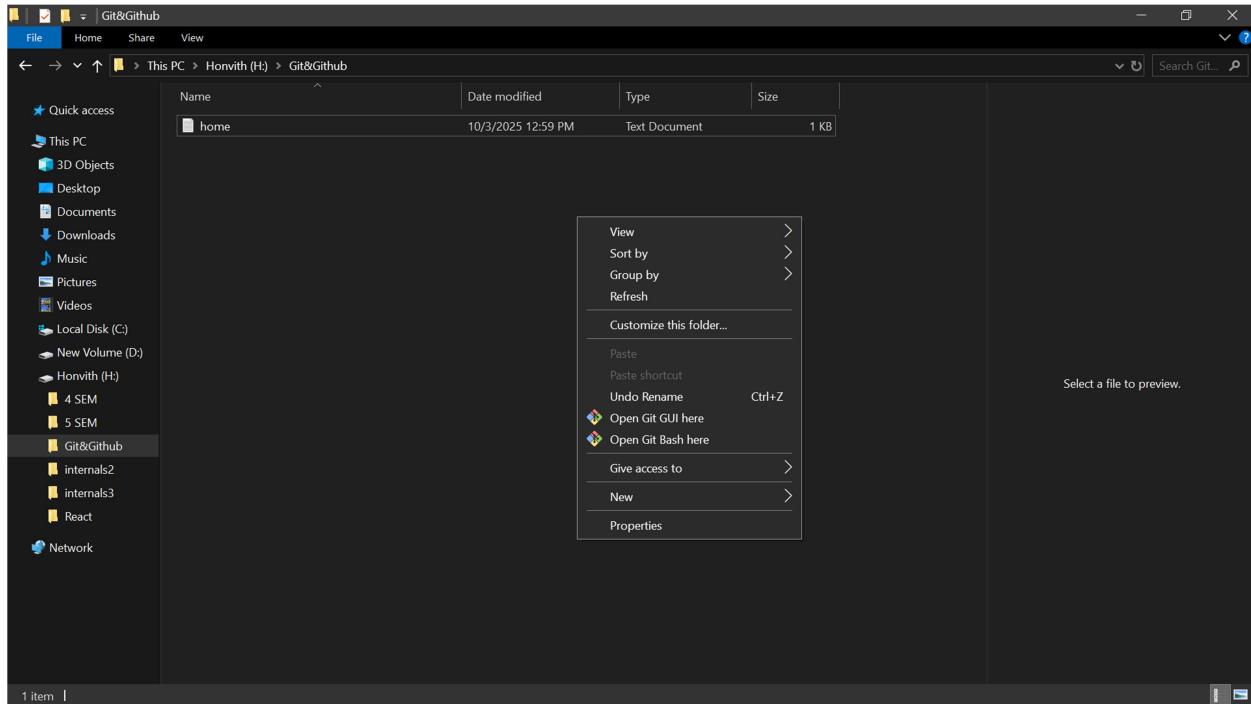
Step 1: Create your Document

- Create a folder named “Git&Github”
- Inside it create a text file named “Home”
- Add text inside the file.



Step 2: Enter the Git CLI

- Right click inside the folder and select “Open Git Bash Here”
- Execute the Commands



1. Configure User Identity

Command :

Set your name globally

git config --global user.name "Your Full Name"

Set your email globally

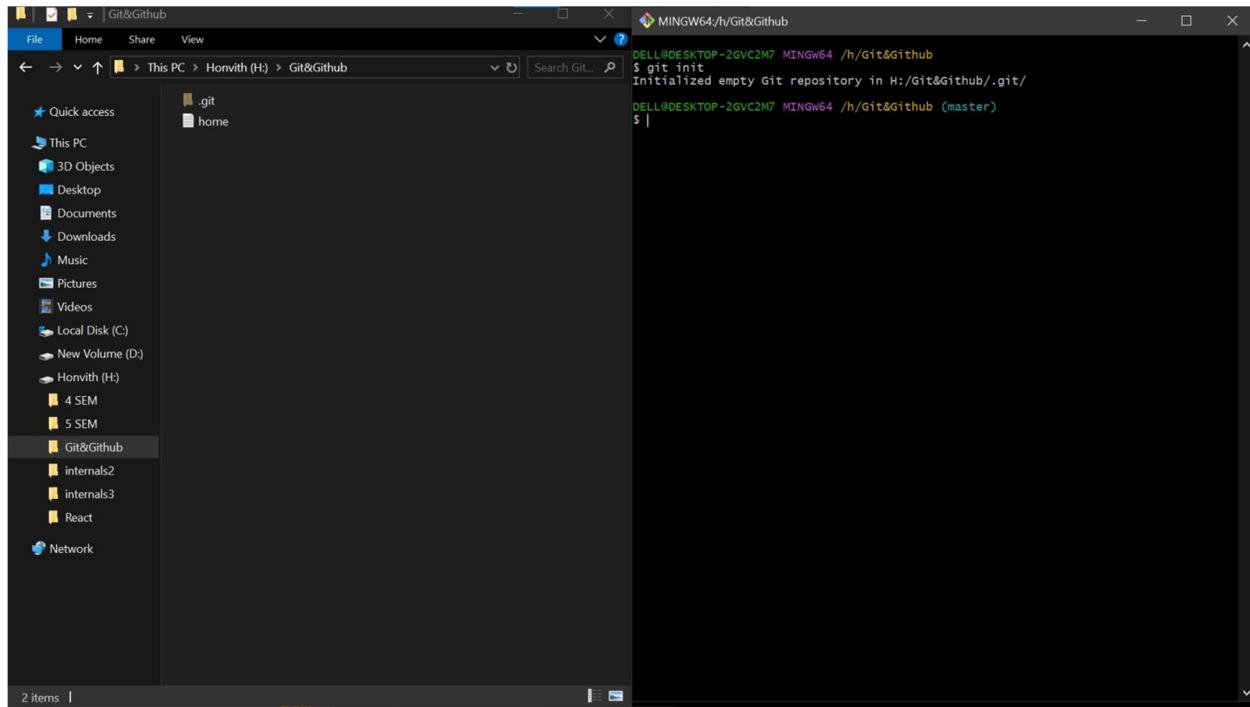
git config --global user.email "your.email@example.com"

What it Does: Every Git commit needs an author identity attached to it.

2. Initialize Git Repository

Command : git init

What it Does : Creates a new Git repository in your current directory by adding a hidden .git folder that tracks all version control information



3. Check Repository Status

Command: git status

What it does: Displays the current state of your working directory and staging area, showing which files are tracked, untracked, modified, or staged for commit.

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    home.txt

nothing added to commit but untracked files present (use "git add" to track)
```

4. Stage File for Commit

Command: git add home.txt

What it does: Adds the specified file (home.txt) to the staging area, preparing it for the next commit.

5. Verify Staging Status

Command: git status (Note: you wrote "get status" but meant "git status")

What it does: Confirms that home.txt is now staged and ready for commit.

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git add home.txt

DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   home.txt
```

6. Commit Staged Changes

Command: git commit -m "add text file"

What it does: Creates a permanent snapshot of the staged changes with a descriptive commit message.

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git commit -m "added a text file"
[master (root-commit) f2b8420] added a text file
 1 file changed, 2 insertions(+)
 create mode 100644 home.txt
```

7. View Commit History

Command: git log

What it does: Displays the commit history, showing commit hashes, author information, timestamps, and commit messages.

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git log
commit f2b84202f0e491ae09502dc2f22a65f34cc4edd4 (HEAD -> master)
Author: HonvithNayakaB <honvithnayaka.b@gmail.com>
Date:   Fri Oct 3 13:11:41 2025 +0530

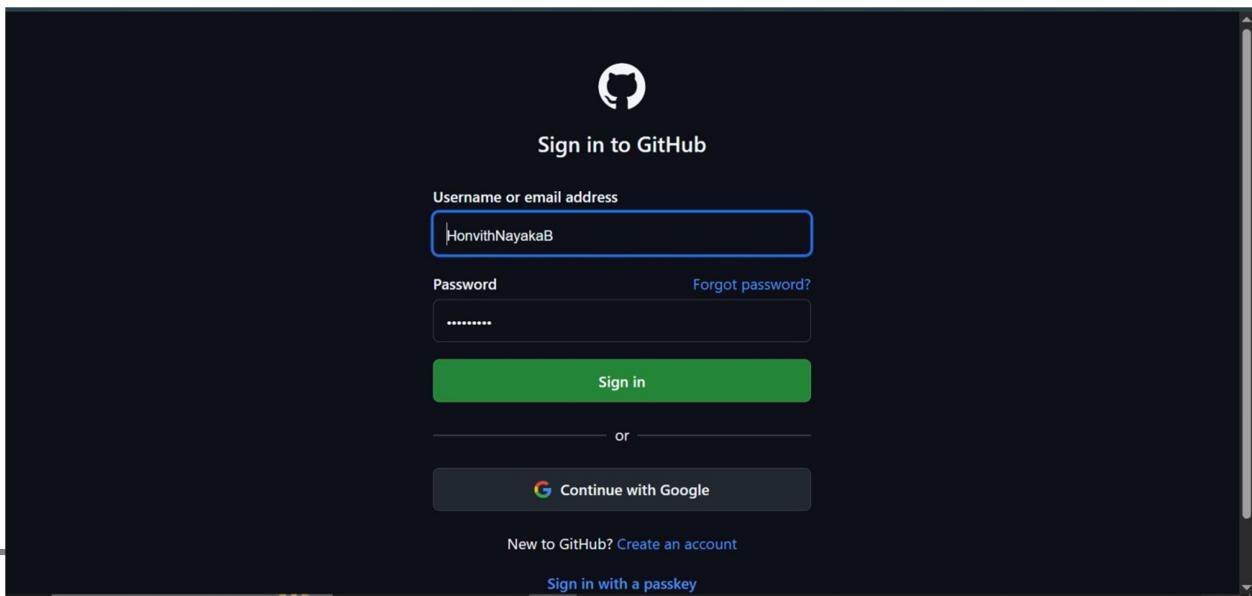
    added a text file
```

8. Add Remote Repository

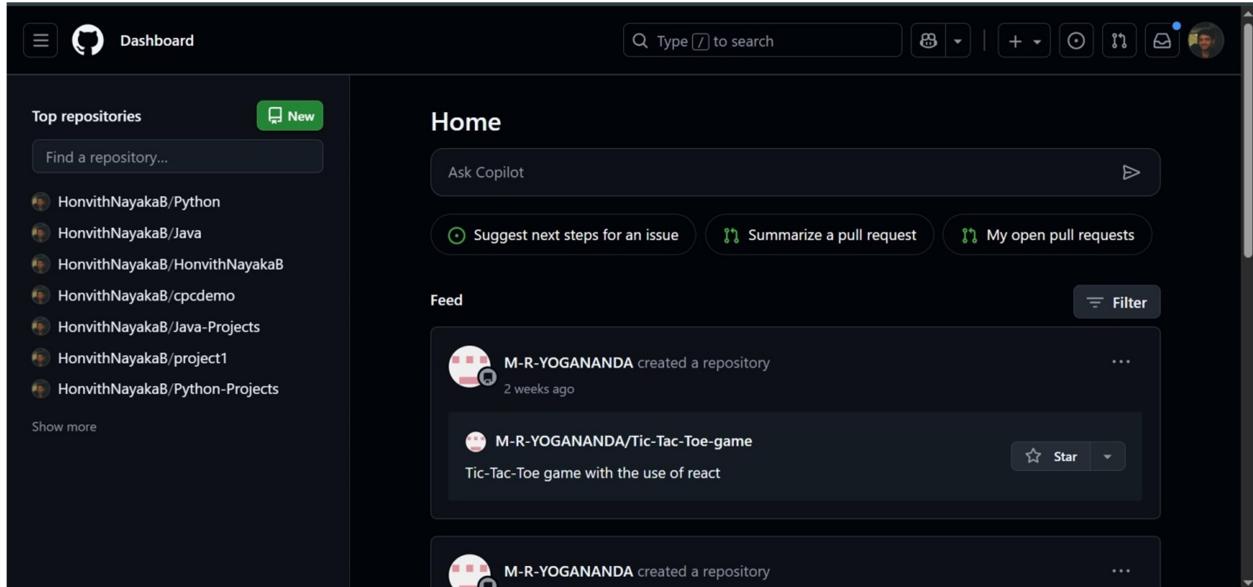
Command: git remote add origin “Github link to repository”

What it does: Links your local repository to a remote Github repository, establishing “origin” as the default remote name.

- Go to Github
- Login/register using your Gmail.



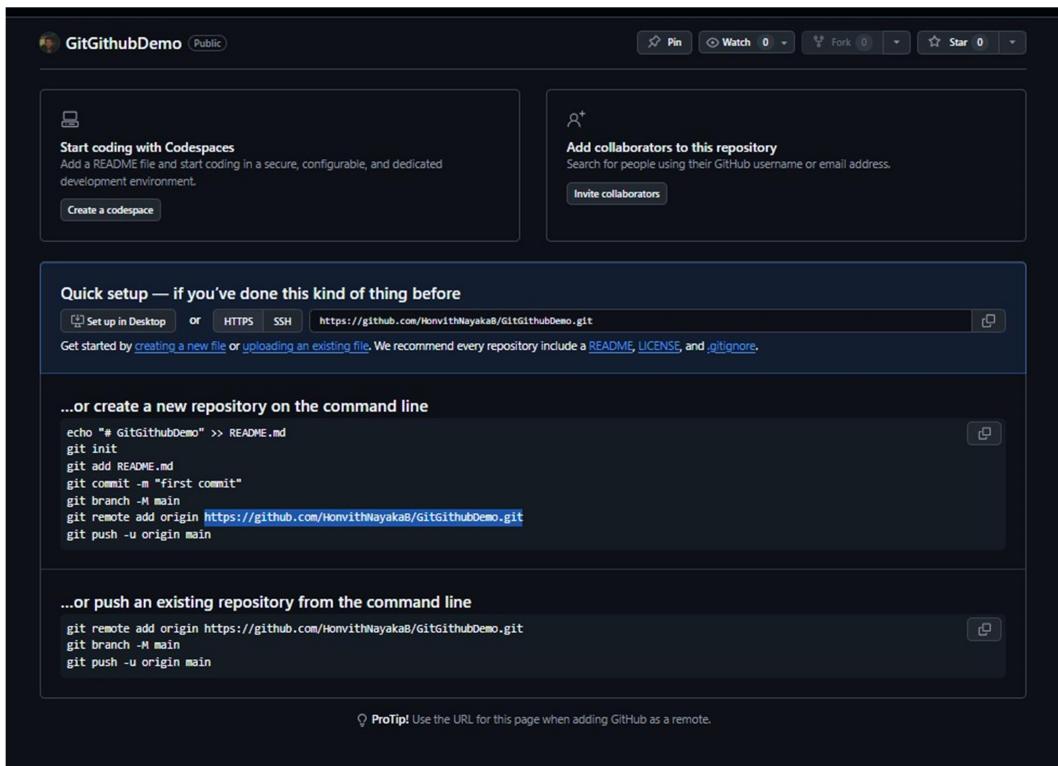
- Create a new repository by clicking on “New”



- Give the repository a name and click on “Create Repository”

The screenshot shows the "Create a new repository" form. It has two main sections: "General" (Step 1) and "Configuration" (Step 2). In the General section, the "Owner" is set to "HonvithNayakaB" and the "Repository name" is "GitHubDemo", which is noted as available. The Configuration section includes options for visibility ("Public"), README ("Off"), .gitignore ("No .gitignore"), and license ("No license"). At the bottom is a large green "Create repository" button.

- Copy the link Highlighted below and paste it in the command given below



- Come to the CLI and type = git remote add origin “link”

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git remote add origin https://github.com/HonvithNayakaB/GitGitHubDemo.git
```

9. Check Branch Information

Command : git branch

What it does : Lists all local branches and highlights the currently active branch with an asterisk.

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git branch
* master
```

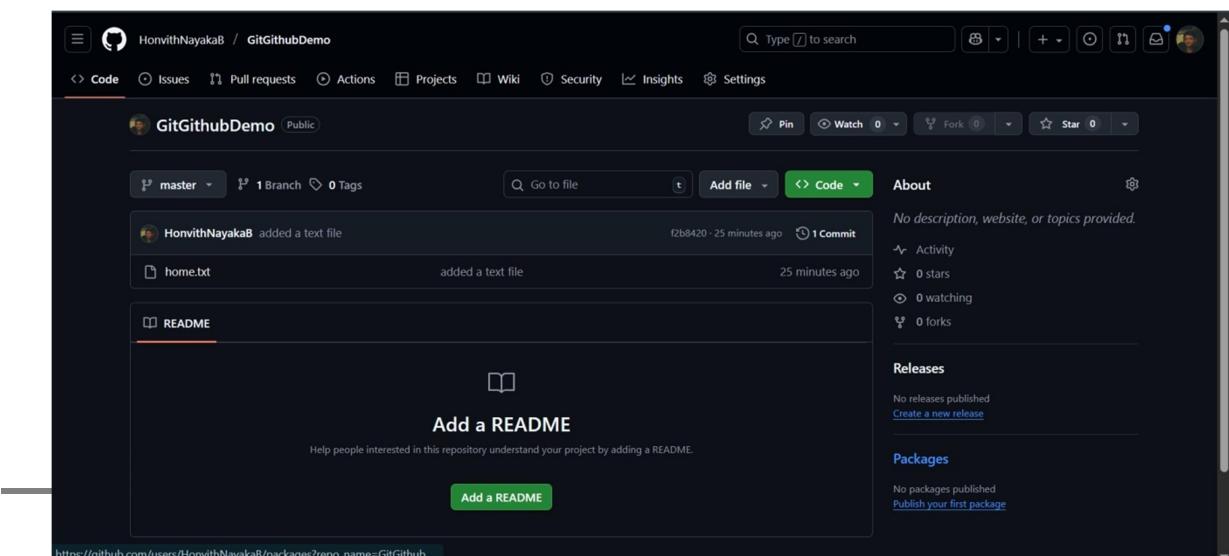
10. push to Remote Repository

Command : git push -u origin master

What it does:

1. Uploads your local main branch to the remote repository
2. -u flag sets up tracking between local main and remote origin/main
3. Future pushes can use just git push

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 143.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/HonvithNayakaB/GitGithubDemo.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

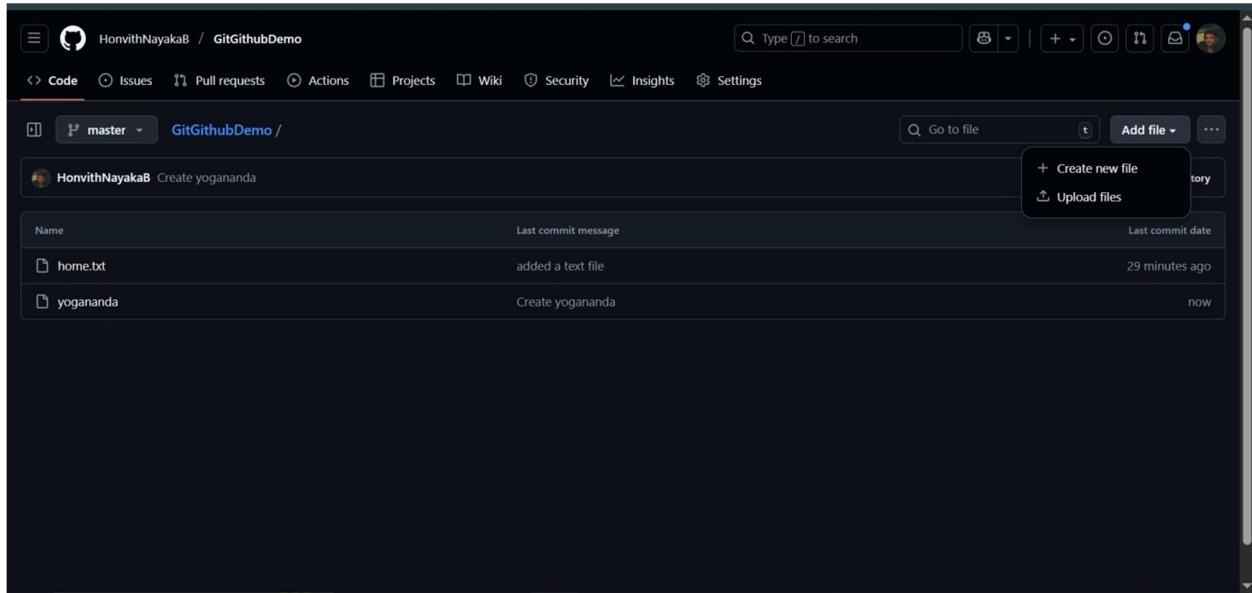


11. pull from Remote Repository

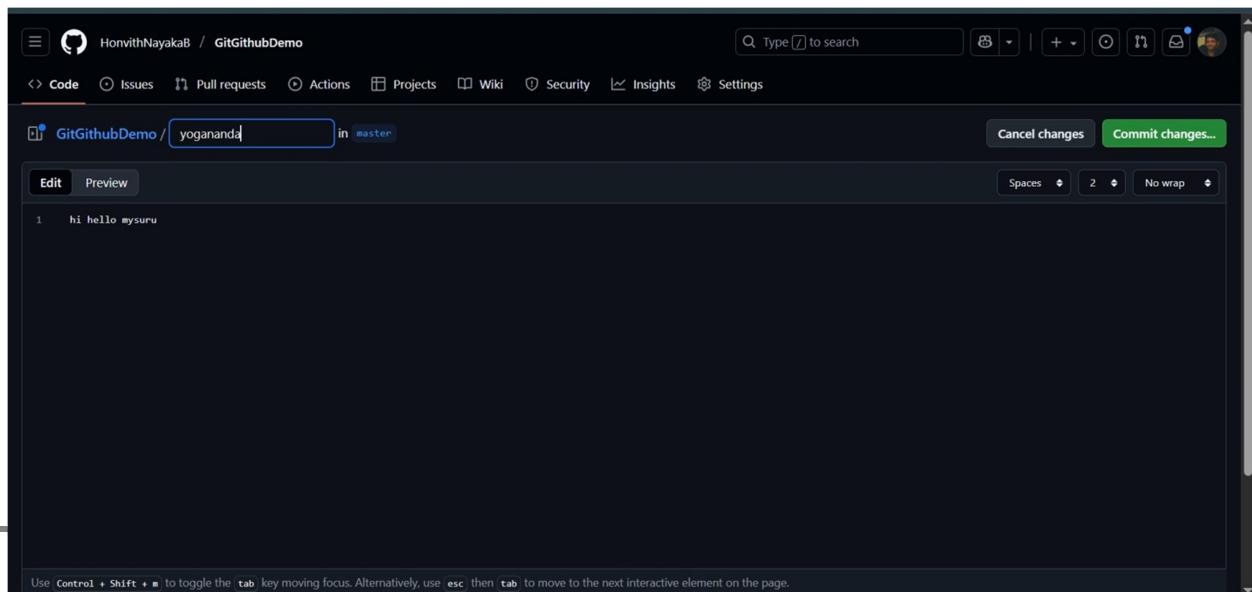
Command: git pull origin master

What it does: pull from specific remote branch

- In Github Repository click on “add file” and “+ Create new file”

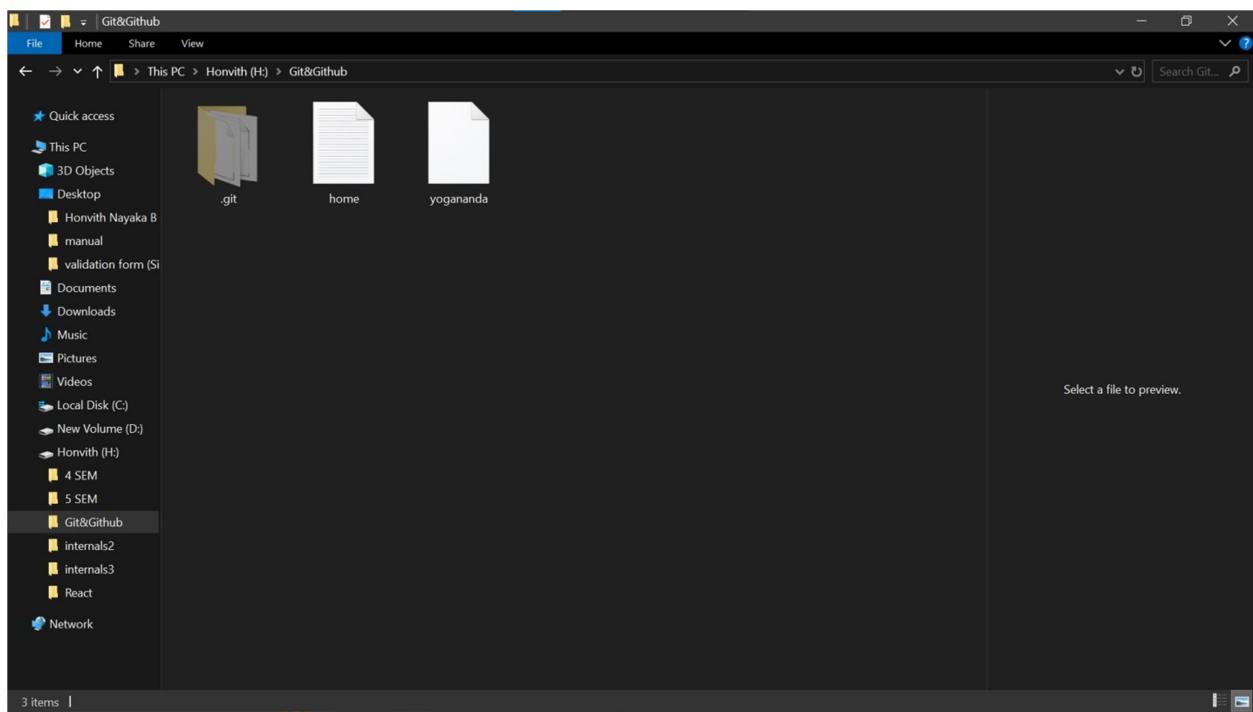


- Add file content and name, click on “commit changes”



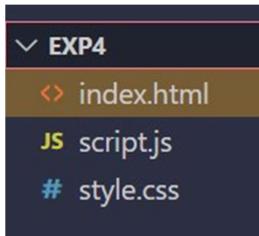
- Now in CLI type the command = git pull origin main

```
DELL@DESKTOP-2GVC2M7 MINGW64 /h/Git&Github (master)
$ git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 937 bytes | 27.00 KiB/s, done.
From https://github.com/HonvithNayakaB/GitGithubDemo
 * branch      master      -> FETCH_HEAD
   f2b8420..16d51a3  master      -> origin/master
Updating f2b8420..16d51a3
Fast-forward
  yogananda | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 yogananda
```



4. Design a Registration Page and perform Validation using JavaScript

Project Structure



Step 1: index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Sign Up Form</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <div class="container">
    <h2>Sign Up</h2>
    <form onsubmit="return validatePasswords()">
      <label>Full Name:</label>
      <input type="text" name="fullname" required pattern="[A-Za-z\s]{3,}" title="Only letters and spaces, min 3 characters">

      <label>Email:</label>
      <input type="email" name="email" required title="Enter a valid email address">

      <label>Username:</label>
      <input type="text" name="username" required minlength="4" maxlength="12" title="4 to 12 characters allowed">

      <label>Password:</label>
      <input type="password" id="password" required minlength="6" title="Minimum 6 characters">

      <label>Confirm Password:</label>
```

```

<input type="password" id="confirmPassword" required title="Re-enter the
password">

<input type="submit" value="Sign Up">
</form>

<div class="success" id="successMessage"></div>
</div>

<script src="script.js"></script>
</body>
</html>

```

Step 2: script.js

```

function validatePasswords() {
    const pwd = document.getElementById("password").value;
    const confirmPassword = document.getElementById("confirmPassword").value;

    if (pwd !== confirmPassword) {
        alert("Passwords do not match!");
        return false;
    }

    document.getElementById("successMessage").innerText = "Sign-up successful!";
    return true;
}

```

Step 3: style.css

```

body {
    font-family: Arial, sans-serif;
    background: linear-gradient(to right, #ff9a9e, #fad0c4);
    padding: 50px;
}

.container {
    max-width: 400px;
    margin: auto;
    background-color: white;
}

```

```
padding: 30px;
border-radius: 15px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
}

h2 {
    text-align: center;
    color: #e91e63;
}

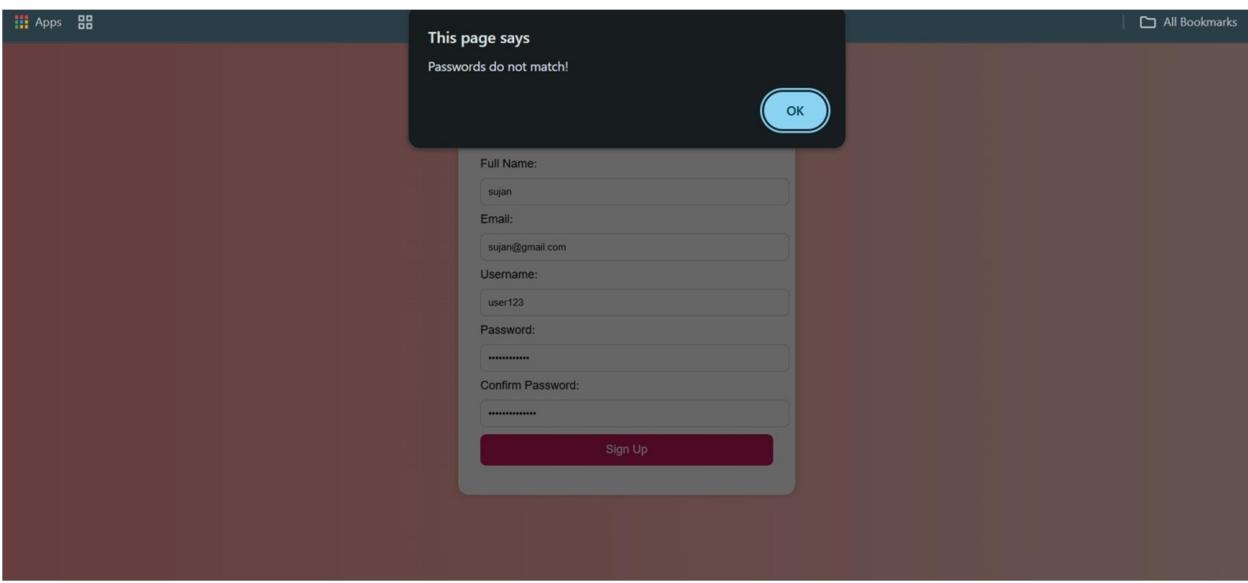
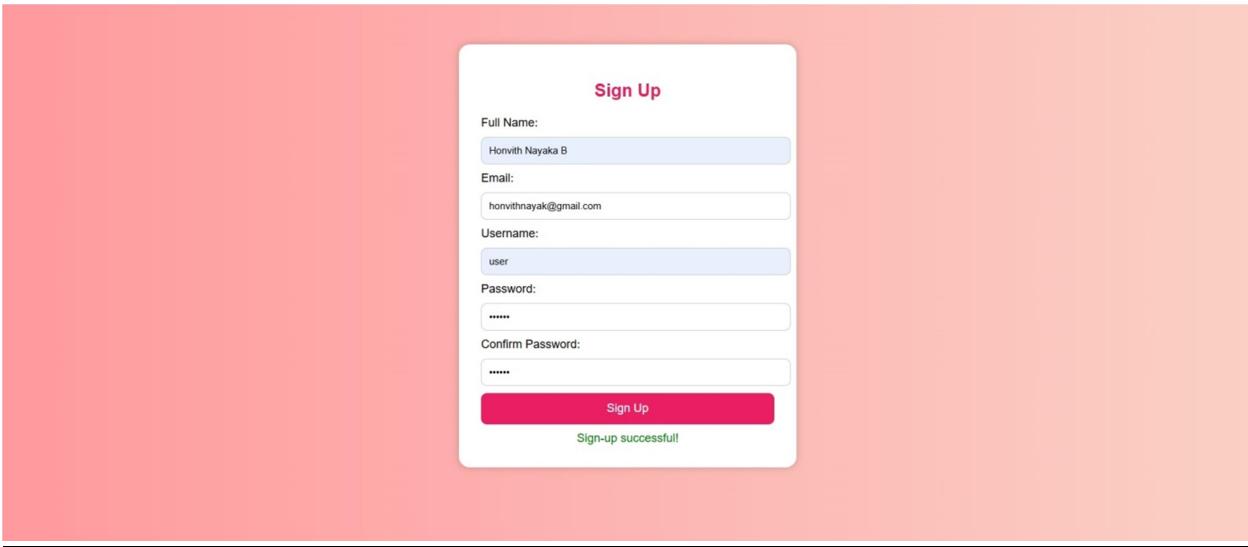
input[type="text"],
input[type="email"],
input[type="password"] {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 8px;
}

input[type="submit"] {
    background-color: #e91e63;
    color: white;
    padding: 12px;
    width: 100%;
    border: none;
    border-radius: 8px;
    font-size: 16px;
    cursor: pointer;
}

input[type="submit"]:hover {
    background-color: #c2185b;
}

.success {
    text-align: center;
    color: green;
    margin-top: 10px;
}
```

Output:



5. Create a Registration Page using React-JS

Step 1: Create a react project

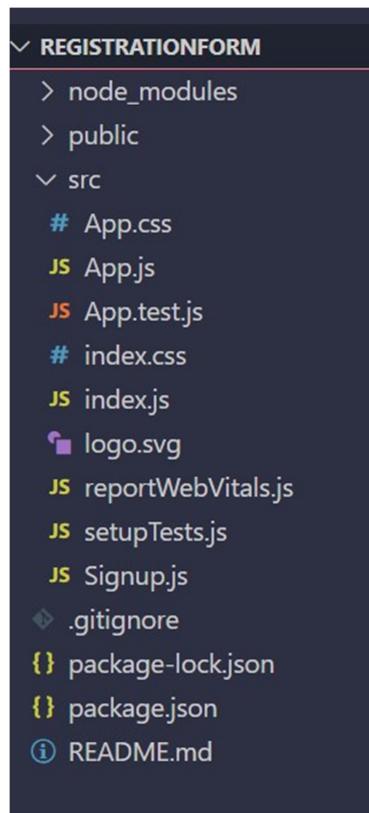
1. Create a new React project by running following command in “Command Prompt”:

- npx create-react-app registration-form

Step 2: Create the Signup.js Component

1. In the src folder, create a file named Signup.js.

File System:



3. Signup.js

```
import React, { useState } from "react";
import "./App.css";

function Signup() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
```

```
const [password, setPassword] = useState("");
const [submitted, setSubmitted] = useState(false);

function handleSubmit(e) {
  e.preventDefault();
  // show success message
  setSubmitted(true);

  // clear inputs after submit
  setName("");
  setEmail("");
  setPassword("");
}

return (
  <div className="signup-container">
    <h2>Registration Form</h2>
    <form onSubmit={handleSubmit} className="signup-form">
      <label>Full Name</label>
      <input
        type="text"
        placeholder="Enter your name"
        value={name}
        onChange={(e) => setName(e.target.value)}
        required
      />

      <label>Email</label>
      <input
        type="email"
        placeholder="Enter your email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        required
      />

      <label>Password</label>
      <input
        type="password"
        placeholder="Enter password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        required
      />
    </form>
  </div>
)
```

```
        <button type="submit">Register</button>
    </form>

    {/* success message */}
    {submitted && (
        <p className="success-message">Registered Successfully!</p>
    )}
    </div>
);
}

export default Signup;
```

Step 4: App.js

```
.signup-container {
    max-width: 400px;
    margin: 60px auto;
    padding: 25px;
    background-color: #f9f9f9;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}

.signup-form {
    display: flex;
    flex-direction: column;
    gap: 15px;
}

.signup-form label {
    text-align: left;
    font-weight: 500;
    color: #333;
}

.signup-form input {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
```

```
    font-size: 14px;
}

.signup-form input:focus {
    border-color: #007bff;
    outline: none;
    box-shadow: 0 0 4px #007bff33;
}

.signup-form button {
    padding: 10px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 6px;
    cursor: pointer;
    font-size: 15px;
    transition: 0.2s;
}

.signup-form button:hover {
    background-color: #0056b3;
}

form {
    display: flex;
    flex-direction: column;
    gap: 10px;
}

input, button {
    padding: 8px;
    font-size: 14px;
}

button {
    background-color: #4CAF50;
    color: white;
    border: none;
}

.success-message {
    margin-top: 15px;
    color: green;
    font-weight: bold;
    animation: fadeIn 0.5s ease;
```

```
}
```



```
@keyframes fadeIn {
  from { opacity: 0; }
  to { opacity: 1; }
}
```

Step 5: App.js

```
import React from "react";
import Signup from "./Signup";

function App() {
  return (
    <div>
      <Signup />
    </div>
  );
}

export default App;
```

Step 6: Run the Application

1. In the terminal, run the following command to start the application:

- **npm start**

2. Open your browser and go to <http://localhost:3000> to view your register form.

Output:

Registration Form

Full Name

Email

Password

Register

Registration Form

Full Name

Email

Password

Register

Registered Successfully!

6. Create a product catalog like Product name, Product ID, Product price and Product image using React-JS

Step 1: Create React App

```
npx create-react-app my-app
```

```
cd my-app
```

Step 2: Create products.json

Create a JSON file inside src folder with sample products:

```
[  
 {  
   "id": 1,  
   "name": "Apple iPhone 12",  
   "description": "The Apple iPhone 12 features a 6.1-inch Super Retina XDR display and A14 Bionic chip.",  
   "price": 999.00,  
   "category": "Electronics",  
   "image": "https://rukminim2.flixcart.com/image/704/844/kg8avm80/mobile/y/7/n/apple-iphone-12-dummyapplefsn-original-imafwg8dpyjvgg3j.jpeg?q=90&crop=false"  
 }]
```

Step 3: Create Viewcart Component

Create Viewcart.js in src/pages/:

```
import { useState } from "react";  
import data from "../products.json";
```

```

const Viewcart =()=>{
  const [products] = useState(data);

  return(
    <>
    <div className="header">
      <div className="logo">

        </div>
    </div>
    <div className="banner">

      </div>
    <div className="productlist">
      {products.map((product)=>(
        <div Key={product.id}>
          <div className="img">
            <img src={product.image} alt={product.name}/>
          </div>
          <div className="details">

            <h3>{product.name}</h3>

            <p>price Rs :{product.price}</p>
            <button>Add to Cart</button>
          </div>
        </div>
      )));
    </div>
    <div className="footer">
      @copywrite CPC Polytechnic, Mysore
    </div>

    </>
  )
}

export default Viewcart;

```

Step 4: Add CSS (App.css)

```
.header{  
    background-color: #f18317;  
    padding: 5px 30px;  
}  
  
.logo{  
    background: url(/src/logo.jpeg);  
    width: 300px;  
    height: 75px;  
}  
  
.banner{  
    background: url(/src/banner.jpeg);  
    width: 100%;  
    height: 400px;  
    background-size: cover;  
}  
  
.productlist{  
    display: flex;  
    flex-wrap: wrap;  
    gap: 40px;  
    justify-content: center;  
    padding: 30px 0px;  
}  
  
.productlist .img{  
    width: 100%;  
    height: 150px;  
}  
  
.productlist .img img{  
    width: 100%;  
    height: 100%;  
    object-fit: cover;  
}  
  
.footer{  
    background-color: #f9ad1c;  
    padding: 15px 30px;  
}  
  
.details button{
```

```
background-color: #f84E4F;  
padding: 5px;  
border: none;  
border-radius: 3px;  
color: white;  
font-size: bolder;  
}  
.details button:hover{  
background-color: rgb(196, 89, 89);  
}
```

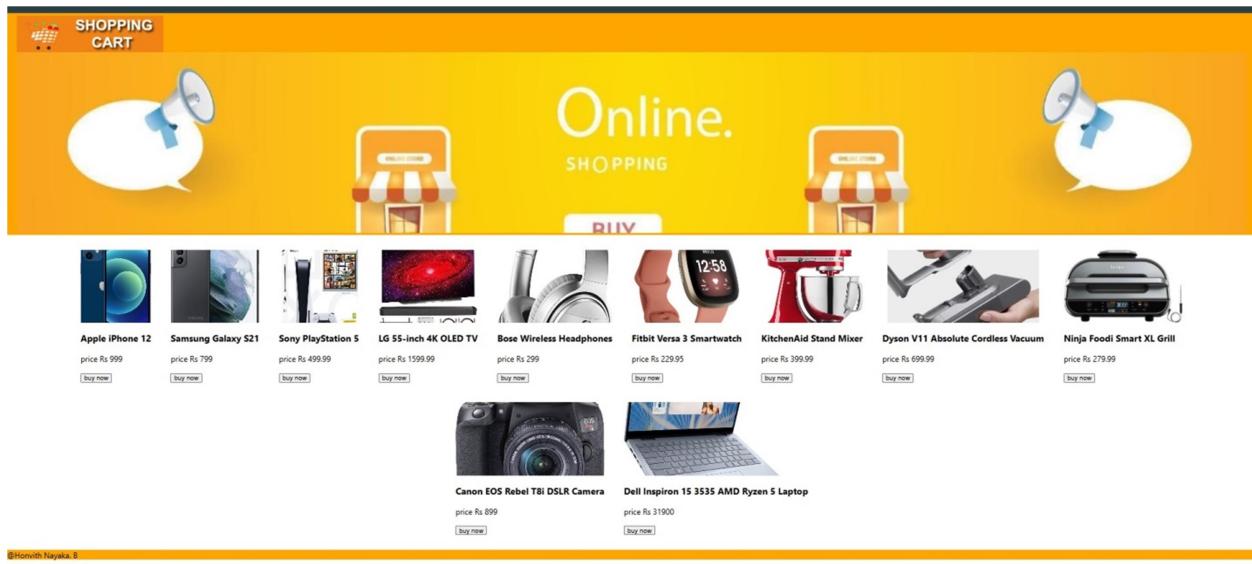
Step 5: Update App.js

```
import './App.css';  
import Viewcart from './pages/Viewcart';  
  
function App0 {  
  return (  
    <Viewcart/>  
  );  
}  
  
export default App;
```

Step 6: Run the Application:

npm start

Output:



7. Create a Navigation Page using React-Router-Dom with multiple react pages.

Step 1: Create React Project

```
npx create-react-app rdom
```

```
cd rdom
```

Step 2: Install React Router DOM

```
npm install react-router-dom
```

Step 3: Create Components

Navigation.js (inside src/components/)

This file contains the navigation menu for switching between pages.

```
1 import { Link } from "react-router-dom";
2 const Navigation = ()=>{
3   return(
4     <>
5       <Link to="/" className="my-link">Home</Link>
6       <Link to="/about" className="my-link">About </Link>
7       <Link to="/contact" className="my-link">Contact </Link>
8     </>
9   )
10 }
11 )
12 }
13 }
14 export default Navigation;
15
16
```

Home.js (inside src/pages/)

```
1 import { startTransition } from "react";
2 import { Link } from "react-router-dom";
3 import Navigation from "../components/Navigation";
4 const Home =()=>{
5
6     return(
7         <>
8             <Navigation/>
9             <h1>Home page</h1>
10
11
12     </>
13 )
14
15 }
16
17 export default Home;
18
```

About.js (inside src/pages/)

```
1 import { Link } from "react-router-dom";
2 import Navigation from "../components/Navigation";
3 const About =()=>{
4
5     return(
6         <>
7             <Navigation/>
8             <h1>About page</h1>
9
10
11     </>
12 )
13 }
14
15 export default About;
16
```

Contact.js (inside src/pages/)

```
1 import { Link } from "react-router-dom";
2 import Navigation from "../components/Navigation";
3 const Contact = ()=>{
4
5     return(
6         <>
7             <Navigation/>
8             <h1>Contact page</h1>
9
10        </>
11    )
12
13 }
14
15 export default Contact;
16
```

Step 4: Setup Routing in App.js

```
1 import { RouterProvider,createBrowserRouter } from 'react-router-dom';
2 import './App.css';
3 import Home from './pages/Home';
4 import About from './pages/About';
5 import Contact from './pages/Contact';
6
7 const router = createBrowserRouter([
8     {path:"/",element:<Home/>},
9     {path:"/about",element:<About/>},
10    {path:"/contact",element:<Contact/>}
11 ]
12 )
13
14 function App() {
15     return (
16         <RouterProvider router={router}>/>
17     );
18 }
19
20 export default App;
21
```

Step 5: Add Styling (App.css)

```
1  body{  
2    background-color: #cadetblue;  
3    display: flex;  
4    justify-content: center;  
5    align-items: center;  
6    min-height: 80vh;  
7  }  
8  
9  h1{  
10   font-size: 100px;  
11 }  
12  
13 .my-link{  
14   padding: 10px;  
15   font-size: 19px;  
16   color: #white;  
17   text-decoration: none;  
18 }  
19 .my-link:hover{  
20   color: #darkblue;  
21   text-decoration: underline;  
22 }  
23  
24  
25
```

Output:

[Home](#) [About](#) [Contact](#)

Home page

localhost:3000

[Home](#) [About](#) [Contact](#)

About page

localhost:3000/about

[Home](#) [About](#) [Contact](#)

Contact page

localhost:3000/contact

8. Perform CRUD Operations without using API with Spring Boot application

CRUD Operation:

- CRUD stands for Create, Read, Update, Delete
- It represents the four basic functions used to manage data in databases or applications. Almost every app you use (Instagram, Flipkart, Banking apps, etc.) is built around CRUD operations.

Create (C) → Adding new data (e.g., inserting a new user into a database).

- **Example:** `INSERT INTO users (name, email) VALUES ('user', 'abc@mail.com');`

Read (R) → Retrieving or viewing existing data without modifying it.

- **Example:** `SELECT * FROM users WHERE id = 1;`

Update (U) → Modifying existing data.

- **Example:** `UPDATE users SET email = 'new@mail.com' WHERE id = 1;`

Delete (D) → Removing data.

- **Example:** `DELETE FROM users WHERE id = 1;`

Prerequisite's :

Node.js

Eclipse

Web browser

Steps to Create a Spring Project :

Step 1 : Go to Google and Search for Spring initializer .

Step 2 : Select

Project : Maven

Language : Java

Spring Boot Version : 3.5.6 (By default)

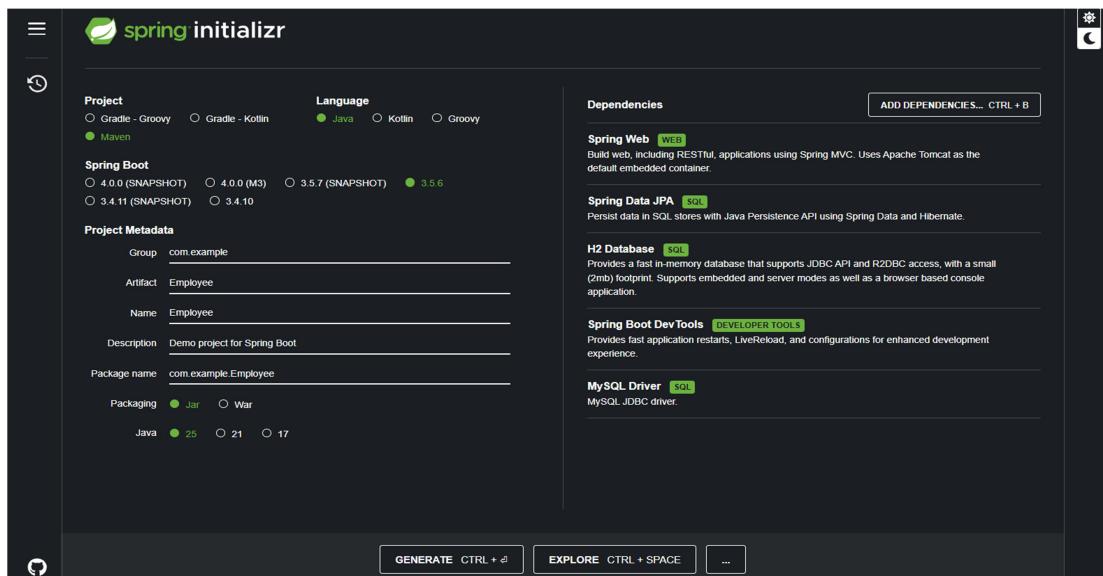
Step 3 : Enter your project Meta Data

Step 4 : Packaging : Jar , Java Version : Based on your System Configuration

Step 5 : Select your Required Dependencies

Uses of Dependencies :

- **Spring Web** → Builds REST APIs and web applications using Spring MVC and embedded Tomcat.
- **Spring Data JPA** → Simplifies database operations using JPA and Hibernate.
- **H2 Database** → Lightweight in-memory database for testing and development.
- **Spring Boot DevTools** → Provides auto-restart and developer productivity tools.
- **MySQL Driver** → Enables Spring Boot to connect with MySQL database.



Annotations and their use :

- **@Entity** → Marks a class as a JPA entity mapped to a database table.
- **@Id** → Specifies the primary key of the entity.
- **@Controller** → Marks the class as a Spring MVC controller that handles web requests.
- **@Autowired** → Automatically injects (wires) the required bean dependency.
- **@RequestMapping** → Maps a web request (URL) to a specific method or controller.
- **@RequestParam** → Binds a request parameter (like form input) to a method parameter

Step 1: Create the Entity class (Model)

Method to create Repository: src/main/java => right click on default package => New => Package => Model

Method to Create File : src/main/java => right click on Model Package => New => class =>Student

Create a new file:

src/main/java/com/cpc/crudcpc/model/Student.java

```
package com.cpc.crudcpc.model;
```

```
import jakarta.persistence.Entity;
```

```
import jakarta.persistence.Id;
```

```
@Entity
```

```
public class Student {
```

```
    @Id
```

```
    private int sid;
```

```
    private String sname;
```

```
    private String branch;
```

```
    private String college;
```

```
    public int getSid() {
```

```
        return sid;
```

```
    }
```

```
    public void setSid(int sid) {
```

```
        this.sid = sid;
```

```
    }
```

```
    public String getSname() {
```

```
        return sname;
```

```
    }
```

```
    public void setSname(String sname) {
```

```
        this.sname = sname;
```

```

    }

    public String getBranch() {
        return branch;
    }

    public void setBranch(String branch) {
        this.branch = branch;
    }

    public String getCollege() {
        return college;
    }

    public void setCollege(String college) {
        this.college = college;
    }

    @Override

    public String toString() {
        return "Student [sid=" + sid + ", sname=" + sname + ", branch=" + branch + ",
college=" + college + "]";
    }
}

```

Step 2: Repository Interface

Method to create Repository: src/main/java => right click on default package => New => Package => Repository

Method to Create File : src/main/java => right click on Repository Package => New => class => StudentRepo

Create this file:

src/main/java/com/cpc/crudcpc/repository/StudentRepo.java

package com.cpc.crudcpc.repository;

import org.springframework.data.repository.CrudRepository;

```
import com.cpc.crudcpc.model.Student;
public interface StudentRepo extends CrudRepository<Student, Integer> {
}
```

This UserRepository automatically gives you CRUD methods like:

- save(user) → insert or update
- findAll() → get all users
- findById(id) → get one user
- deleteById(id) → delete user

Step 3: Add Dependencies in pom.xml

Make sure your pom.xml has these (you already added tomcat-jasper for JSP):

```
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
</dependency>
```

```
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <scope>provided</scope>
</dependency>
```

Step 4: Controller (API Layer)

Method to create Repository: src/main/java => right click on default package => New => Package => Controller

Method to Create File : Method : src/main/java => right click on Controller Package => New => class => StudentController

Create:

src/main/java/com/cpc/crudcpc/controller/UserController.java

```
package com.cpc.crudcpc.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.cpc.crudcpc.model.Student;
import com.cpc.crudcpc.repository.StudentRepo;

@Controller
public class StudentController {

    @Autowired
    StudentRepo studentrepo;

    @RequestMapping("index")
    public String index() {
        return "index.jsp";
    }

    @RequestMapping("addStu")
    public String addStu(Student student) {
        studentrepo.save(student);
        return "index.jsp";
    }

    @RequestMapping("getStu")
    public ModelAndView getStu(@RequestParam int sid) {
        ModelAndView mv = new ModelAndView("display.jsp");
        Student student = studentrepo.findById(sid).orElse(new Student());
        mv.addObject("student", student);
        return mv;
    }
}
```

```
mv.addObject(student);

return mv;

}

@RequestMapping("delStu")
public ModelAndView delStu(@RequestParam int sid) {

    ModelAndView mv = new ModelAndView("delete.jsp");

    Student student = studentrepo.findById(sid).orElse(new Student());
    studentrepo.deleteById(sid);

    mv.addObject(student);

    return mv;
}

@RequestMapping("updStu")
public ModelAndView updStu(Student student) {

    ModelAndView mv = new ModelAndView("update.jsp");

    student = studentrepo.findById(student.getId()).orElse(new Student());
    mv.addObject(student);

    return mv;
}

}
```

Step 5: Database Configuration

In your project you already have `src/main/resources/application.properties`. Add this:

```
# Database Configuration (MySQL Example)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/studentcpcdb
```

```

spring.datasource.username=root
spring.datasource.password=yourpassword
# Hibernate Config
spring.jpa.hibernate.ddl-auto=update
# Server port
server.port=8080

```

- Create database in MySQL:

```
CREATE DATABASE cruddb;
```

Step 6: Run the Application

- In Eclipse → Right click project → **Run As** → **Spring Boot App**
- Server starts on: <http://localhost:8080>

Step 7: JSP Pages

01. Index.jsp

Method to create jsp file : src => main => right click => new => file => index.jsp

```

<html>
<head>
</head>
<body>
<div Style="text-align:center">
<h1>CRUD Operation by using JSP and Mysql(Without API)</h1>
</div>
<div Style="padding-left:50px">
<h2>Add Student</h2>
<form action="addStu">
Student Id:<input type="text" name="sid"/> <br/>
Student Name:<input type="text" name="sname"/> <br/>

```

```
Branch:<input type="text" name="branch"/> <br/>
College:<input type="text" name="college"/> <br/>
<input type="submit"/>
</form>
```

```
<h2>Get Student</h2>
<form action="getStu">
Student Id:<input type="text" name="sid"/> <br/>
<input type="submit"/>
</form>
```

```
<h2>Delete Student</h2>
<form action="delStu">
Student Id:<input type="text" name="sid"/> <br/>
<input type="submit"/>
</form>
```

```
<h2>Update Student</h2>
<form action="updStu">
Student Id:<input type="text" name="sid"/> <br/>
<input type="submit"/>
</form>
```

```
</div>
```

```
</body>
</html>
```

02. Update.jsp

Method to create jsp file : src => main => right click => new => file => Update.jsp

```
<html>
<head>
</head>
<body>
<div Style="text-align:center">
<h1>CRUD Operation by using JSP and Mysql(Without API)</h1>
</div>
<div Style="padding-left:50px">
<h2>Add Student</h2>
<form action="addStu">
Student Id:<input type="text" name="sid" value="${student.sid }> <br/>
Student Name:<input type="text" name="sname" value="${student.sname }> <br/>
Branch:<input type="text" name="branch" value="${student.branch }> <br/>
College:<input type="text" name="college" value="${student.college }> <br/>
<input type="submit"/>
</form>
</div>
</body>
</html>
```

03. Display.jsp

Method to create jsp file : src => main => right click => new => file => Display.jsp

```
Student Id:${student.sid}<br/>
Student Name:${student.sname}<br/>
Branch:${student.branch}<br/>
College:${student.college}<br/>
```

04. Delete.jsp

Method to create jsp file : src => main => right click => new => file => delete.jsp

`${student.sid} Record is Deleted Sucessfully.....`

Now when you run CrudcpcApplication:

- `http://localhost:8080/` → opens index.jsp
- Add/Get/Delete/Update → handled by JSPs (display.jsp, update.jsp, delete.jsp)

Output :

Crud Operation using JSP and Mysql (Without API)

Add Student

Student Id:
Student Name:
Branch:
College:

Get Student

Student Id:

Delete Student

Student Id:

Update Student

Student Id:

Insert :

Crud Operation using JSP and Mysql (Without API)

Add Student

Student Id:

Student Name:

Branch:

College:

➤ Database Record :

← T →	▼	sid	branch	college	sname
<input type="checkbox"/>	 Edit	 Copy	 Delete	1 CS	CPC Shakhi shree B

Update :

Update Student

Student Id:

➤ Database Record :

← T →	▼	sid	branch	college	sname
<input type="checkbox"/>	 Edit	 Copy	 Delete	1 CS	CPC Manasa

View :

Get Student

Student Id:

➤ Database Record :

Student Id : 1

Student Name : Manasa

Branch:CS

College:CPC

Delete :

Delete Student

Student Id:

Manasa deleted sucessfully.....

➤ Database Record :

	sid	branch	college	sname
--	------------	---------------	----------------	--------------

9.Perform CRUD Operations using API with Spring Boot application

CRUD Operation with API:

API:

API stands for Application Programming Interface.

It is a set of rules and methods that allows one software application to communicate with another.

Types of APIs:

- REST API → most common, uses HTTP (e.g., GET, POST, PUT, DELETE).
- GraphQL API → lets you request only the data you need.
- SOAP API → older, uses XML.
- Web APIs → used in browsers and web apps (like Google Maps API).

CRUD:

CRUD stands for the four basic operations you can perform on data in a database or an API:

1. C → Create → Add new data
2. R → Read → Get/retrieve data
3. U → Update → Modify existing data
4. D → Delete → Remove data.

Prerequisite's :

Node.js

Eclipse

Postman tool

Web browser

Steps to Create a Spring Project :

Step 1 : Go to Google and Search for Spring initializer .

Step 2 : Select

Project : Maven

Language : Java

Spring Boot Version : 3.5.6 (By default)

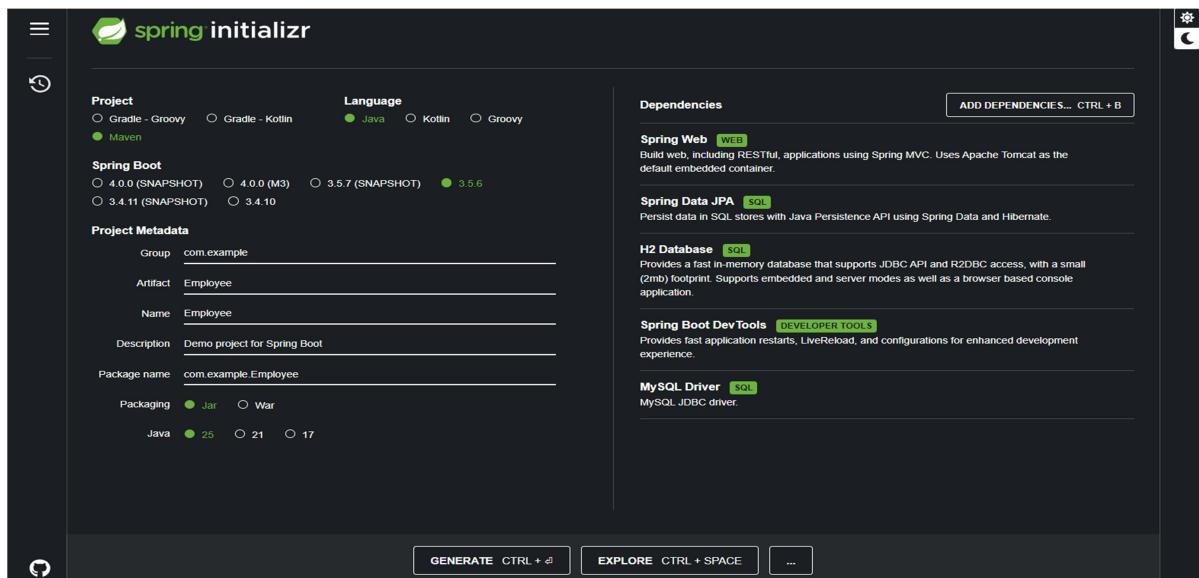
Step 3 : Enter your project Meta Data

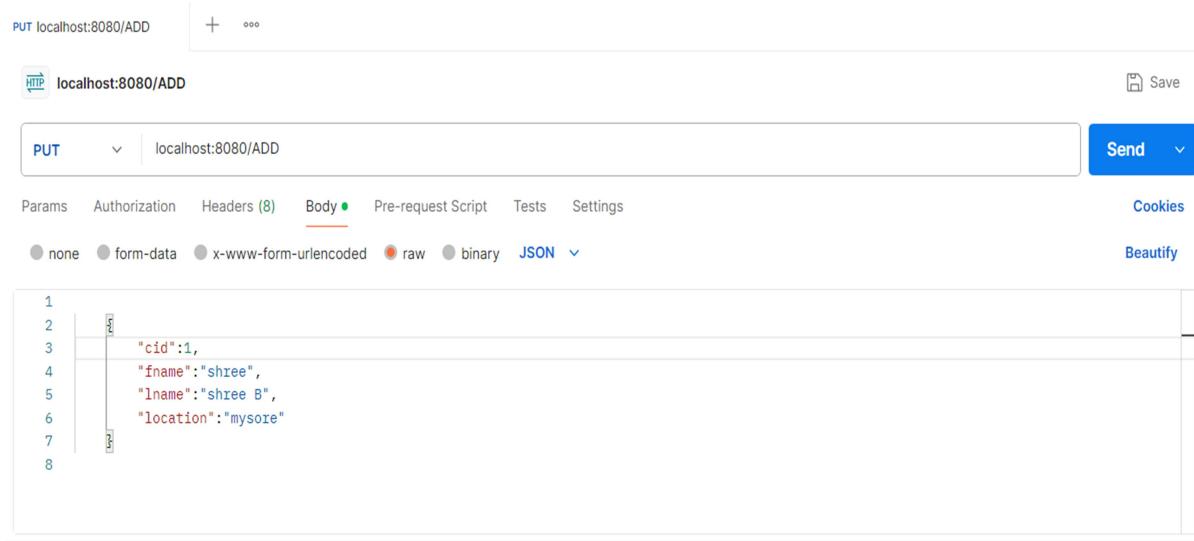
Step 4 : Packaging : Jar , Java Version : Based on your System Configuration

Step 5 : Select your Required Dependencies

Uses of Dependencies :

- **Spring Web** → Builds REST APIs and web applications using Spring MVC and embedded Tomcat.
- **Spring Data JPA** → Simplifies database operations using JPA and Hibernate.
- **H2 Database** → Lightweight in-memory database for testing and development.
- **Spring Boot DevTools** → Provides auto-restart and developer productivity tools.
- **MySQL Driver** → Enables Spring Boot to connect with MySQL database.





Annotations and their use:

- **@SpringBootApplication** → Marks the main class as a Spring Boot application with auto-configuration.
- **@RestController** → Creates REST APIs by combining **@Controller** and **@ResponseBody**.
- **@Autowired** → Injects required dependencies automatically.
- **@PostMapping** → Handles HTTP POST requests for creating data.
- **@GetMapping** → Handles HTTP GET requests for fetching data.
- **@DeleteMapping** → Handles HTTP DELETE requests for deleting data.
- **@PutMapping** → Handles HTTP PUT requests for updating data.
- **@RequestBody** → Maps the request JSON body to a Java object.
- **@PathVariable** → Maps a URL path variable to a method parameter.
- **@Entity** → Declares a class as a JPA entity mapped to a database table.
- **@Id** → Marks a field as the primary key of the entity.

CrudapiApplication.java : Default Package

```
package com.cpcapi.crudapi;
```

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CrudapiApplication {

    public static void main (String [] args) {
        SpringApplication.run (CrudapiApplication.class, args);
    }

}

```

3. EmpController.java:

Method to create Repository: src/main/java => right click on default package => New => Package => Controller

Method to Create File : src/main/java => right click on Controller Package => New => class =>EmpController

```

package com.cpcapi.crudapi.Controller;

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;

import com.cpcapi.crudapi.model.Employee;
import com.cpcapi.crudapi.repository.EmpRepo;

@RestController
public class EmpController {

    @Autowired
    EmpRepo emprepo;

    @PostMapping("/addEmp")
    public String addEmp(@RequestBody Employee employee) {

```

```

emprepo.save(employee);
return "Record Inserted Sucessfully";
}
@GetMapping("/getEmp")
public List<Employee> getEmp() {
    return(List<Employee>) emprepo.findAll();

}

@DeleteMapping("/delEmp/{eid}")
public String delEmp(@PathVariable int eid) {
    emprepo.deleteById(eid);
    return "Record Delete Sucessfully";
}

@PutMapping("/updEmp")
public String updEmp(@RequestBody Employee employee) {
    Employee empupdate =
    emprepo.findById(employee.getEid()).get();
    empupdate.setEname(employee.getEname());
    empupdate.setCompany(employee.getCompany());
    empupdate.setLocation(employee.getLocation());
    emprepo.save(empupdate);
    return "Record Updated Sucessfully";
}

}

```

4. Employee. Java:

Method to create Repository: src/main/java => right click on default package => New => Package => Model

Method to Create File : src/main/java => right click on Model Package => New => class => Employee.java

```

package com.cpcapi.crudapi.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

```

```
@Entity
public class Employee {

    @Id
    private int eid;
    private String ename;
    private String company;
    private String location;
    public int getEid() {
        return eid;
    }
    public void setEid(int eid) {
        this.eid = eid;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
        this.company = company;
    }
    public String getLocation() {
        return location;
    }
    public void setLocation(String location) {
        this.location = location;
    }
    @Override
    public String toString() {
        return "Employee [eid=" + eid + ", ename=" + ename + ",
company=" + company + ", location=" + location + "]";
    }
}
```

5. EmpRepo.java:

Method to create Repository: src/main/java => right click on default package => New => Package => Repository

Method to Create File : src/main/java => right click on Repository Package => New => Interface => EmpRepo

```
package com.cpcapi.crudapi.repository;

import
org.springframework.data.repository.CrudReposito
ry;

import com.cpcapi.crudapi.model.Employee;

public interface EmpRepo extends
CrudRepository<Employee, Integer> {

}
```

6.application.properties:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/crudapicpc_db
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto = update

server.port=8082
```

6. Add Dependencies in pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Output :

Insert Record :

The screenshot shows a Postman interface with the following details:

- URL:** `localhost:8080/addEmp`
- Method:** POST
- Body (JSON):**

```
1 {"eid": 1,
2 "ename": "Shakthi",
3 "company": "Infosys",
4 "location": "Mysore"}  
5
```
- Response Status:** 200 OK
- Response Body:** "Record Inserted Sucessfully"

View Record :

HTTP localhost:8080/getEmp

GET localhost:8080/getEmp

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
Key	Value	

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 15 ms Size: 301 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "eid": 1,
4     "ename": "Shakthi",
5     "company": "Infosys",
6     "location": "Mysore"
7   },

```

Update Record :

HTTP localhost:8080/getEmp

PUT localhost:8080/updEmp

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary JSON

Beautify

Key	Value
"eid": 1,	
"ename": "Manasa",	
"company": "Infosys",	
"location": "Mysore"	

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 29 ms Size: 190 B Save Response

Pretty Raw Preview Visualize Text

```

1 Record Updated Sucessfully

```

Updated Record :

The screenshot shows the Postman interface for a GET request to `localhost:8080/getEmp`. The request method is set to `GET`. The response status is `200 OK`, time is `10 ms`, and size is `300 B`. The response body is a JSON object:

```

1
2 {
3   "eid": 1,
4   "ename": "Manasa",
5   "company": "Infosys",
6   "location": "Mysore"
7 },

```

Delete Record :

The screenshot shows the Postman interface for a DELETE request to `localhost:8080/delEmp/1`. The request method is set to `DELETE`. The response status is `200 OK`, time is `44 ms`, and size is `190 B`. The response body is a simple message:

```

1 Record Deleted Sucessfully

```

10. Perform A) Create and Drop Database, B) Create and Drop Table. C) Insert one and Insert many. D) Delete one and Delete Many, operations using Mongo DB Shell

What is MongoDB?

- MongoDB is a **NoSQL database** that stores data in **documents** (similar to JSON format).
- Data is stored in **collections**, and each collection contains multiple **documents**.
- It is widely used because it is **flexible, fast, and easy to use**.

What is MongoDB Shell?

- MongoDB Shell (`mongosh`) is a **command-line interface** used to interact with MongoDB.
- You can use it to **create databases, insert data, query data, update, and delete documents**.

Pre-Requisites:

1. MongoDB must be installed on your system.
2. MongoDB server should be running.
3. Open **MongoDB Shell (`mongosh`)** from your terminal or command prompt.

How to Open MongoDB Shell:

1. Open **Command Prompt** (Windows) or **Terminal** (Mac/Linux).
2. Type:`mongosh`
3. Press **Enter** — it connects you to the local MongoDB server.
4. You'll see a prompt like:

```
test>
```

This means MongoDB Shell is ready to use.

Basic Commands:

1. Create Database

```
use studentdb
```

- Creates or switches to a database named studentdb.

```
test> use CollegeDB
switched to db CollegeDB

```

2. Create Collection

db.createCollection("student")

- Creates a collection named student.

```
CollegeDB> db.createCollection("Student")
{ ok: 1 }
```

3. Insert Documents (Create)

db.student.insertOne({ name: "Honvith Nayaka. B", age: "18", course: "CSE" })

- Inserts a single document into the student collection.

```
CollegeDB> db.Student.insertOne({name:"Honvith Nayaka. B", age:"18", course:"CSE"})
{
  acknowledged: true,
  insertedId: ObjectId('68e9f6644d150c8548cebea4')
}
```

db.student.insertMany([

```
{ name: "M R Yogananda", age: "20", course: "CSE" },
{ name: "Jamuna", age: "20", course: "CSE" }
```

)

- Inserts multiple documents at once.

```
CollegeDB> db.Student.insertMany([{name:"M R Yogananda", age:"18", course:"CSE"}, {name:"Jamuna", age:"18", course:"CSE"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68e9f70d4d150c8548cebea5'),
    '1': ObjectId('68e9f70d4d150c8548cebea6')
  }
}
```

4. Display Data (Read)

db.student.find()

- Displays all documents in the student collection.

`db.student.find().pretty()`

- Displays the data in a readable (formatted) way.

```
CollegeDB> db.Student.find().pretty()
[
  {
    _id: ObjectId('68e9f6644d150c8548cebea4'),
    name: 'Honvith Nayaka. B',
    age: '18',
    course: 'CSE'
  },
  {
    _id: ObjectId('68e9f70d4d150c8548cebea5'),
    name: 'M R Yogananda',
    age: '18',
    course: 'CSE'
  },
  {
    _id: ObjectId('68e9f70d4d150c8548cebea6'),
    name: 'Jamuna',
    age: '18',
    course: 'CSE'
  }
]
```

5. Delete Data (Delete)

`db.student.deleteOne({ name: "Honvith Nayaka. B" })`

- Deletes one document matching the condition.

```
CollegeDB> db.Student.deleteOne({name:"Honvith Nayaka. B"})
{ acknowledged: true, deletedCount: 1 }
```

`db.student.deleteMany({ age: "18" })`

- Deletes all documents where age is “18”.

```
CollegeDB> db.Student.deleteMany({age:"18"})
{ acknowledged: true, deletedCount: 2 }
```

6 . Show database

- Show dbs

```
>_MONGOSH
> show dbs
< Shakthi   112.00 KiB
  admin     192.00 KiB
  config      48.00 KiB
  cruddbcpc 108.00 KiB
  local      80.00 KiB
test >
```

7. Drop Collection or Database

`db.student.drop()`

- Deletes the student collection.

```
CollegeDB> db.student.drop()
true
```

`db.dropDatabase()`

- Deletes the current database.

```
CollegeDB> db.dropDatabase()
{ ok: 1, dropped: 'CollegeDB' }
```

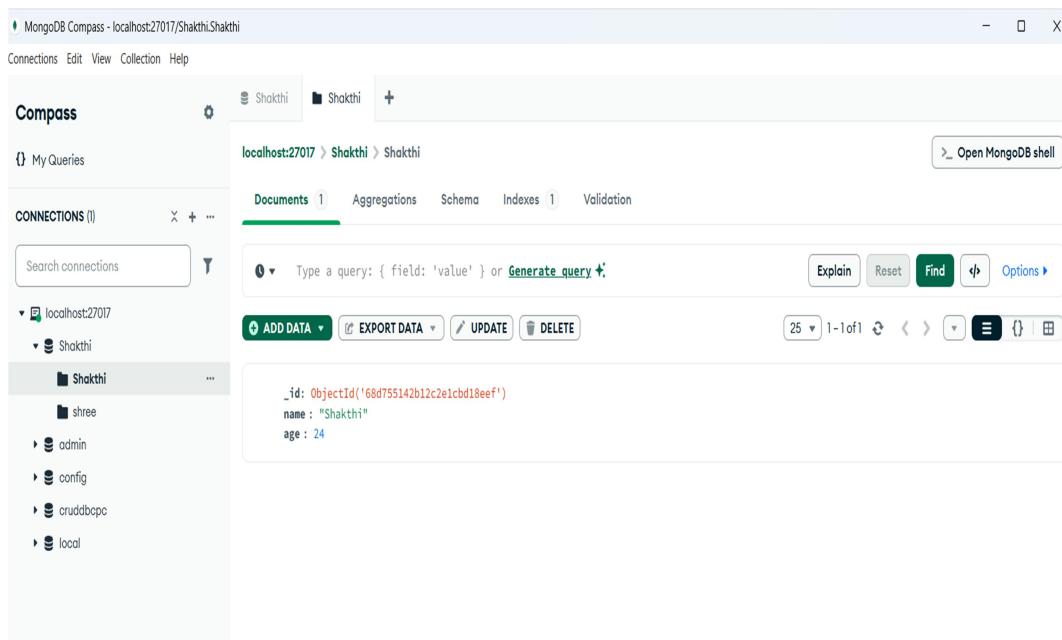
11. Perform CRUD Operations using Mongo DB Compass with Spring Boot application

What is MongoDB?

- MongoDB is a **NoSQL database** used to store data in a **document (JSON-like)** format.
- It is **non-relational, fast, and flexible** for handling large amounts of data.
- Data is stored in **collections** (similar to tables) and **documents** (similar to rows).

What is MongoDB Compass?

- MongoDB Compass is a **Graphical User Interface (GUI)** for MongoDB.
- It allows users to **view, insert, update, and delete** documents easily without writing code.
- It helps to visualize database structure and run queries interactively.



CRUD Operations:

Operation Description

Create Add new documents

Operation Description

Read Retrieve data from collection

Update Modify existing data

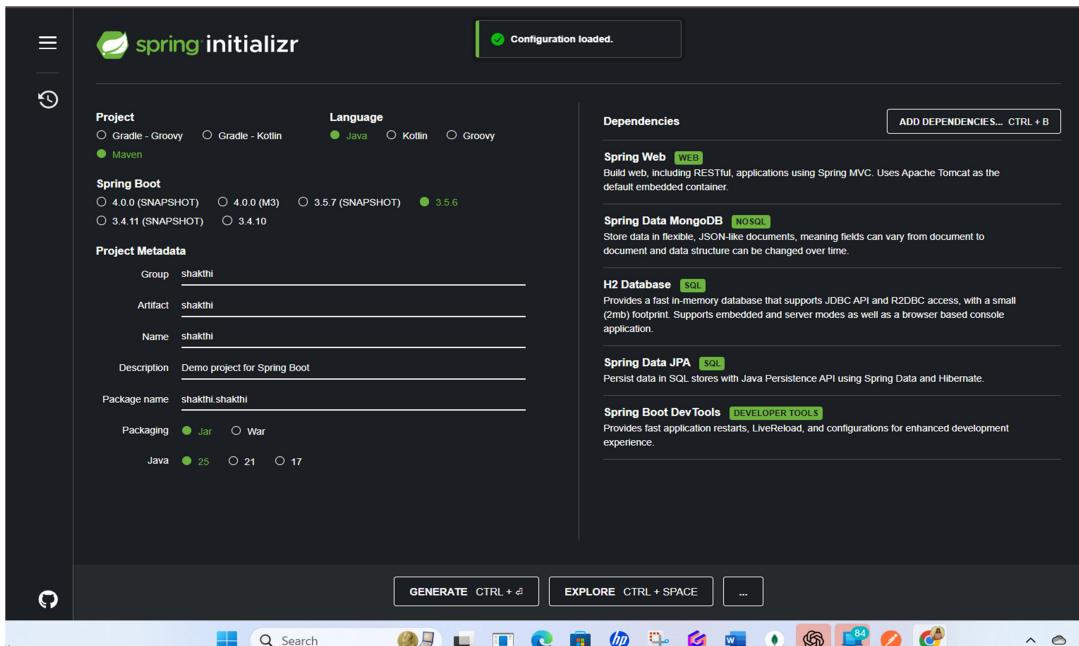
Delete Remove data from collection

Pre-Requisites:

- MongoDB Server installed and running
- MongoDB Compass installed
- Eclipse IDE installed
- MongoDB Java Driver added to the project.

Dependencies and their Uses :

- **Spring Web** – Used to build web applications and RESTful APIs using Spring MVC.
- **Spring Data MongoDB** – Enables connection and CRUD operations with MongoDB (NoSQL database).
- **H2 Database** – Provides an in-memory database for testing and quick data storage during development.
- **Spring Data JPA** – Simplifies working with relational (SQL) databases using the Java Persistence API.
- **Spring Boot DevTools** – Speeds up development with automatic restarts and live reload on code changes.



Annotations and their uses :

- **@SpringBootApplication** – Marks the main class of a Spring Boot project; enables auto-configuration and component scanning.
- **@RestController** – Used to define a RESTful web controller that returns data as JSON.
- **@Autowired** – Automatically injects required objects or dependencies into a class.
- **@RequestMapping** – Maps a specific URL or path to a method in the controller.
- **@GetMapping** – Handles HTTP GET requests to fetch data from the server.
- **@PathVariable** – Captures values from the URL and passes them as method parameters.
- **@RequestBody** – Binds JSON data from the request body to a Java object.
- **@Document** – Maps a Java class to a MongoDB collection.
- **@Id** – Marks a field as the unique identifier (primary key) in a MongoDB document.

crudmongo12.application => Default package

```
package com.cpcmongo.crudmongo12;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class Crudmongo12Application {  
  
    public static void main(String[] args) {  
        SpringApplication.run(Crudmongo12Application.class, args);  
    }  
  
}
```

Step 1: Controller (API Layer)

Method to create Repository: src/main/java => right click on default package => New => Package => Controller

Method to Create File : src/main/java => right click on Controller Package => New => class => StudentController

```
package com.cpcmongo.crudmongo12.controller;  
  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import com.cpcmongo.crudmongo12.model.Customer;  
import com.cpcmongo.crudmongo12.repository.CustRepo;
```

```
@RestController  
public class CustController {  
    @Autowired  
    CustRepo custrepo;  
    @RequestMapping("/add")  
    public String addcust(@RequestBody Customer customer) {  
        custrepo.save(customer);  
        return "Record Inserted Sucessfully";  
    }  
    @GetMapping("/get")  
    public List<Customer> getCust(){  
        return custrepo.findAll();  
    }  
    @RequestMapping("/del/{cid}")  
    public String delcust (@PathVariable int cid) {  
        custrepo.deleteById(cid);  
        return "Record Deleted Sucessfully";  
    }  
    @RequestMapping("/upd/{cid}")  
    public String updcust(@RequestBody Customer customer) {  
        Customer newCust=custrepo.findById(customer.getCid()).get();  
        newCust.setCname(customer.getCname());  
        newCust.setFname(customer.getFname());  
        newCust.setLname(customer.getLname());  
        newCust.setLocation(customer.getLocation());  
        custrepo.save(newCust);  
        return "Record Updated Sucessfully";  
    }  
}
```

```
}
```

Step 2: Create the Entity class (Model)

Method to create Repository: src/main/java => right click on default package => New => Package => Model

Method to Create File : src/main/java => right click on Model Package => New => class =>Customer

```
package com.cpcmongo.crudmongo12.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

{@Document("tblcust")
public class Customer {

    @Id
    private int cid;
    private String cname;
    private String fname;
    private String lname;
    private String location;
    public int getCid() {
        return cid;
    }
    public void setCid(int cid) {
        this.cid = cid;
    }
    public String getCname() {
```

```
        return cname;
    }

public void setCname(String cname) {
    this.cname = cname;
}

public String getFname() {
    return fname;
}

public void setFname(String fname) {
    this.fname = fname;
}

public String getLname() {
    return lname;
}

public void setLname(String lname) {
    this.lname = lname;
}

public String getLocation() {
    return location;
}

public void setLocation(String location) {
    this.location = location;
}

@Override

public String toString() {
    return "Customer [cid=" + cid + ", cname=" + cname + ", fname=" + fname + ",
    lname=" + lname + ", location="
        + location + "]";
}
```

```

    }
}

```

Step 3 :Repository Interface

Method to create Repository: src/main/java => right click on default package => New => Package => Repository

Method to Create File : src/main/java => right click on Repository Package => New => Interface=>CustRepo

```

package com.cpcmongo.crudmongo12.repository;

import org.springframework.data.mongodb.repository.MongoRepository;

import com.cpcmongo.crudmongo12.model.Customer;

public interface CustRepo extends MongoRepository<Customer, Integer> {}

```

Output :

Insert a document :

➤ Postman :

The screenshot shows the Postman interface with the following details:

- Request URL:** `localhost:8080/getEmp` (for GET requests) and `localhost:8080/add` (for POST requests).
- Method:** POST
- Body:** JSON (selected)
- JSON Data:**

```

1 {
2     "cid":1,
3     "cname": "Infosys",
4     "fname": "Shakthi Shree",
5     "lname": "B",
6     "location": "mys"
7 }
```
- Response Status:** 200 OK
- Response Time:** 895 ms
- Response Size:** 191 B
- Response Message:** "Record Inserted Sucessfully"

➤ MongoDB Compass :

The screenshot shows the MongoDB Compass interface. The top navigation bar includes 'Welcome', 'tblcust', and a '+' button. Below it, the path 'localhost:27017 > cruddbcp > tblcust' is displayed. The main area has tabs for 'Documents' (0), 'Aggregations', 'Schema', 'Indexes' (2), and 'Validation'. A search bar says 'Type a query: { field: 'value' } or Generate query'. Below the search bar are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The results section shows a single document:

```

_id: 1
cname : "Infosys"
fname : "Shakthi Shree"
lname : "B"
location : "mys"
_class : "com.cpcmongo.crudmongo12.model.Customer"

```

Update a Document :

➤ Postman :

The screenshot shows the Postman application. The top header shows tabs for 'GET localhost:8080/getEmp', 'POST localhost:8080/addEmp', 'PUT localhost:8080/getEmp', and 'PUT localhost:8080/getEmp'. The selected tab is 'PUT localhost:8080/getEmp'. The request URL is 'localhost:8080/upd/1'. The 'Body' tab is selected, showing a JSON payload:

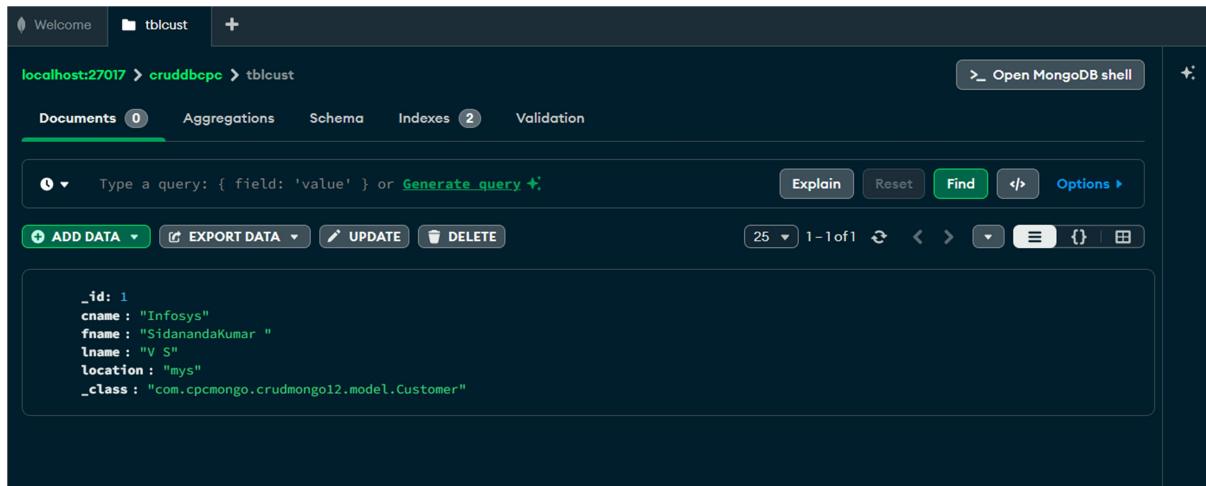
```

1 {
2   "cid":1,
3   "cname":"Infosys",
4   "fname": "SidanandaKumar",
5   "lname": "V S",
6   "location": "mys"
7 }

```

The response status is 'Status: 200 OK Time: 164 ms Size: 190 B'. The response body shows: '1 Record Updated Sucessfully'.

➤ MongoDB Compass :



View Document :

➤ Postman :

The screenshot shows the Postman application. At the top, there are tabs for 'GET localhost:8080/getEmp', 'POST localhost:8080/addEmp', 'PUT localhost:8080/getEmp', and 'GET localhost:8080/getEmp' (which is highlighted). Below the tabs, the URL 'localhost:8080/getEmp' is entered in the 'HTTP' field. The 'Body' tab is selected, showing a JSON payload:

```

1
2
3
4
5
6

```

At the bottom, the response is shown in 'Pretty' format:

```

1 [
2   {
3     "cid": 1,
4     "cname": "Infosys",
5     "fname": "SidanandaKumar ",
6     "lname": "V S",
7     "location": "mys"
8   }
9 ]

```

The status bar at the bottom indicates 'Status: 200 OK Time: 100 ms Size: 250 B Save Response'.

➤ MongoDB Compass :

The screenshot shows the MongoDB Compass interface. At the top, there are tabs for 'Welcome', 'tblcust' (selected), and a '+' button. Below the tabs, the path is shown as 'localhost:27017 > crudbcpc > tblcust'. The main area has tabs for 'Documents' (0), 'Aggregations', 'Schema', 'Indexes' (2), and 'Validation'. A search bar at the top says 'Type a query: { field: 'value' } or Generate query+'. Below it are buttons for 'Explain', 'Reset', 'Find', 'Options', 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. A status bar at the bottom indicates '25 1-1 of 1'.

```

_id: 1
cname : "Infosys"
fname : "SidanandaKumar "
lname : "V S"
location : "mys"
_class : "com.cpcmongo.crudmongo12.model.Customer"

```

Delete Document :

➤ Postman :

The screenshot shows the Postman application. At the top, there are tabs for 'GET localhost:8080/getEmp', 'POST localhost:8080/addEmp', 'PUT localhost:8080/getEmp', and 'DEL localhost:8080/getEmp'. Below the tabs, the URL is set to 'localhost:8080/getEmp'. A 'DELETE' method is selected in the dropdown. The 'Body' tab is active, showing the URL 'localhost:8080/del/1'. To the right, there are 'Send' and 'Save' buttons. Below the body, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body' (highlighted in green), 'Pre-request Script', 'Tests', and 'Settings'. Under 'Body', there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'JSON' (set to 'JSON'). The 'Body' section contains the number '1'. At the bottom, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The status bar at the bottom right shows 'Status: 200 OK Time: 50 ms Size: 190 B Save Response'.

➤ MongoDB Compass :

localhost:27017 > crudbopc > tblcust

Documents 1 Aggregations Schema Indexes 2 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#)

25 0 – 0 of 0

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import data

12. Create JUnit test cases for CRUD Operations and execute the test cases

Step 1: Create a Spring Boot Project

1. File → New → Spring Boot Project → Maven
2. Name the project: HotelJUnitProject
3. Select dependencies:
 - o Spring Data JPA
 - o H2 Database
 - o Spring Boot DevTools (optional)
4. Finish project creation.

Step 2: Create Package

- Right-click **src/main/java** → New → Package → Name:
com.cpcjunit.junittest

Step 3: Add Hotel Entity

Create **Hotel.java** inside the package.

```
package com.cpcjunit.junittest;

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

@Entity
@Table(name="tbl_hotel")
public class Hotel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int hid;
    private String hname;
    private String haddress;
    private String fitem;
    public Hotel(String hname, String haddress, String fitem) {
```

```

        this.hname = hname;
        this.haddress = haddress;
        this.fitem = fitem;
    }
    public Hotel() {

        // TODO Auto-generated constructor stub
    }
    public int getHid() {
        return hid;
    }
    public void setHid(int hid) {
        this.hid = hid;
    }
    public String getHname() {
        return hname;
    }
    public void setHname(String hname) {
        this.hname = hname;
    }
    public String getHaddress() {
        return haddress;
    }
    public void setHaddress(String haddress) {
        this.haddress = haddress;
    }
    public String getFitem() {
        return fitem;
    }
    public void setFitem(String fitem) {
        this.fitem = fitem;
    }
}

}

```

Step 4: Add Repository Interface

Create `HotelRepo.java`:

```

package com.cpcjunit.junittest;

import org.springframework.data.jpa.repository.JpaRepository;

public interface HotelRepo extends JpaRepository<Hotel, Integer>{
}

```

JUnit Test Class for CRUD Operations

Step 5: Create HotelRepoTest.java

```

package com.cpcjunit.junittest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.List;

import org.assertj.core.api.Assertions;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.OrderAnnotation;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.annotation.Rollback;

@DataJpaTest
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class HotelRepoTest {

    @Autowired
    private HotelRepo hotelrepo;

    @Test
    @Order(1)
    @Rollback(value=false)
    public void saveHotelTest() {
        Hotel hotel = new Hotel("radisonblue", "mysore", "dosa");
        hotelrepo.save(hotel);
        Assertions.assertThat(hotel.getHid()).isGreaterThan(0);

    }
    @Test
    @Order(2)
    public void getHotelTest() {
        List<Hotel> hotel = hotelrepo.findAll();
        Assertions.assertThat(hotel.size()).isGreaterThan(0);

    }
    @Test
    @Order(3)
    @Rollback(value=false)
    public void updHotelTest() {
        Hotel hotel = hotelrepo.findById(1).get();
        hotel.setFitem("biriyani");
    }
}

```

```

        Hotel newHotel = hotelrepo.save(hotel);
        Assertions.assertThat(newHotel.getFitem()).isEqualTo("biriyani");
    }

    @Test
    @Order(4)
    @Rollback(value=false)

    public void delHotel() {
        boolean beforeDelete = hotelrepo.findById(1).isPresent();
        hotelrepo.deleteById(1);
        boolean afterDelete = hotelrepo.findById(1).isPresent();
        assertTrue(beforeDelete);
        assertFalse(afterDelete);
    }

}

```

Run the JUnit Tests in Eclipse

Step 6: Run Test

1. Right-click on HotelRepoTest.java → Run As → JUnit Test.
2. Observe the **JUnit** view:
 - **Green bar** → All tests passed
 - **Red bar** → Test failed → check assertion or ID

Output:

```

1 package com.cpcjunit.junittest;
2
3 import static org.junit.jupiter.api.Assertions.assertFalse;
4 import static org.junit.jupiter.api.Assertions.assertTrue;
5
6 import java.util.List;
7
8 import org.assertj.core.api.Assertions;
9 import org.junit.jupiter.api.MethodOrderer;
10 import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;
11 import org.junit.jupiter.api.Order;
12 import org.junit.jupiter.api.Test;
13 import org.junit.jupiter.api.TestMethodOrder;
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
16 import org.springframework.test.annotation.Rollback;
17
18 @DataJpaTest
19 @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
20 public class HotelReposTest {
21
22     @Autowired
23     private HotelRepo hotelrepo;
24
25     @Test
26     @Order(1)
27     @Rollback(value=false)
28     public void saveHotelTest() {
29         Hotel hotel = new Hotel("radisonblue","mysore","dosa");
30         hotelrepo.save(hotel);
31         Assertions.assertThat(hotel.getId()).isGreaterThan(0);
32
33     }
34     @Test
35     @Order(2)
36     public void getHotelTest() {
37         List<Hotel> hotel = hotelrepo.findAll();
38         Assertions.assertThat(hotel.size()).isGreaterThan(0);
39
40     }
41     @Test
42     @Order(3)
43     @Rollback(value=false)
44     public void updHotelTest() {
45         Hotel hotel = hotelrepo.findById(1).get();
46         hotel.setFitem("biriyani");
47
48     }
49
50 }

```