Index Sheet

FSD MANUAL (Practical)

1.Create a user story for Hospital management

Features	Description	Priority	Acceptance Criteria
Admin authentication	Secure login for admin & doctors	High	Admin can login using their name & password with multi-factor authentication as an option.
Patient Creation	Create new patient account details	High	Ability to enter personal details & initial deposit for patient.
Patient account update	Update customer details like address & contact information	Medium	Admin can modify the patient data, records.
Account management	View & manage patient accounts, including balances	High	Ability to view patient details, patient status.
Transaction management	Credit patient accounts, with transaction receipts	High	Performs transactions generate receipts, & update balance
Report Generation	Generate reports on patient details & activity	High	Admin can generate & export reports in various format.
Audit Log	Log all actions performed by the admins for accountability	High	System records every change or option with a timestamp,
Alert & notifications	Provide alerts for suspicions activity or important patient threshold	Medium	System sends notifications for activities like patient details.
Data security & Privacy compliance	Ensure all patient data in secure & complaints with hospital	High	Data encryption, secure storage & regular patient details.

2.Create a Local repository, clone repository using GitHub and Explain about Jenkins :

>> What is GitHub :

GitHub is a web-based platform that uses Git for version control. It allows developers to store and manage their code, track changes, collaborate with others, and work on multiple projects simultaneously. GitHub is popular for open-source projects, team collaboration, and code sharing.

Key Concepts:

Git: A distributed version control system that tracks changes in source code during software development.

Repository: A project or folder that contains all files and history related to the project. In GitHub, a repository can be hosted online (remote repository).

Local Repository :-

A local repository is a Git repository that exists on your local machine. It's where you can make changes, commit them, and manage your code without affecting the main (remote) repository until you push those changes.

Example :- Suppose you have a project on your local computer. You initialize it as a Git repository using the following command : ***git init*** .

Now, this folder is a local Git repository where you can track changes.

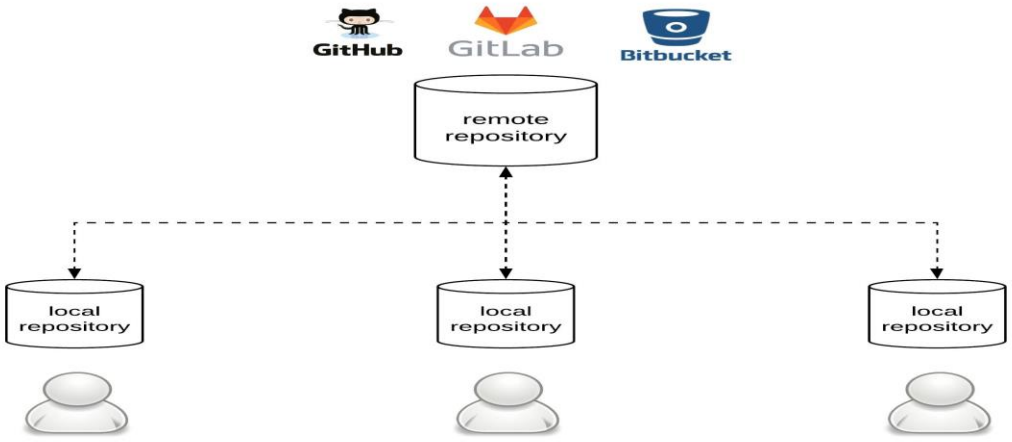


Fig : GitHub Local Repository

Clone Repository :-

Cloning a repository means creating a copy of a remote GitHub repository on your local machine. This allows you to work on the project locally, make changes, and then push those changes back to the remote repository on GitHub.

Example :- To clone a repository from GitHub, you would use:

git clone https://github.com/username/repository-name.git

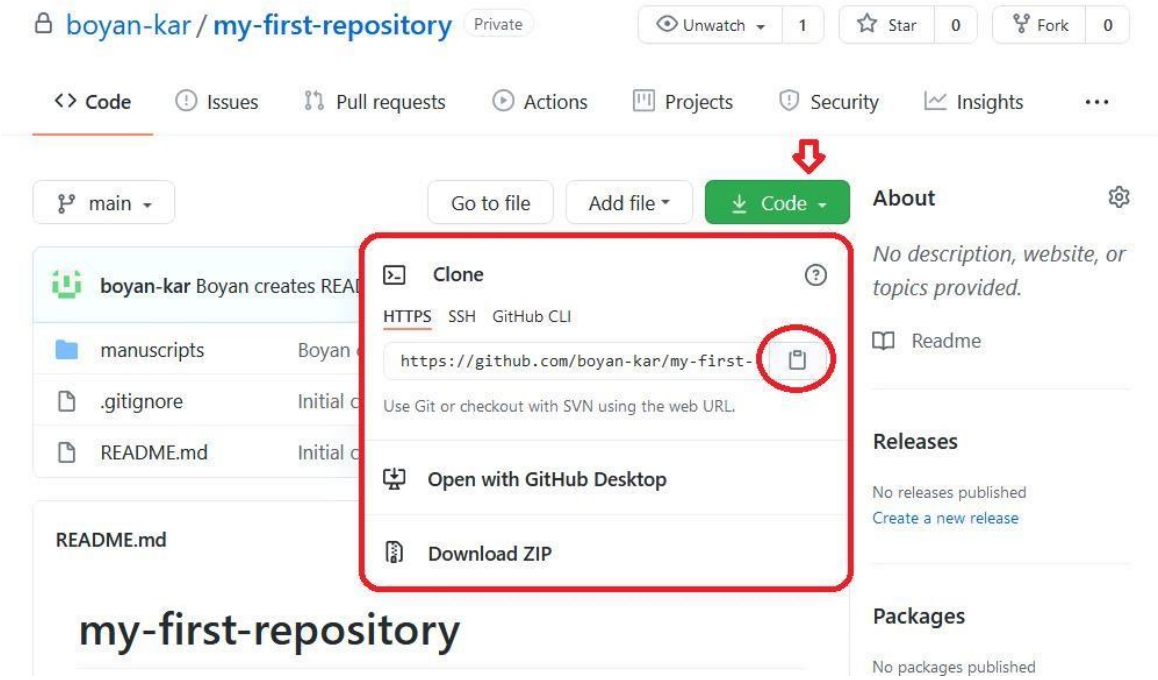


Fig : GitHub Clone Repository

>> What is Jenkins :-

Jenkins is an open-source automation server used to automate parts of the software development process, such as building, testing, and deploying applications. It's widely used for continuous integration (CI) and continuous delivery (CD) in software projects.

Key Features of Jenkins:

Automation: Jenkins automates repetitive tasks, like code integration, builds, and testing, making the development process more efficient.

Plugins: Jenkins has a vast ecosystem of plugins that extend its functionality. These plugins allow Jenkins to integrate with many other tools and technologies, such as GitHub, Docker, Maven, etc.

Continuous Integration (CI): Jenkins helps developers integrate their code changes into a shared repository frequently, automatically building and testing the code to ensure it works correctly.

Continuous Delivery (CD): After testing, Jenkins can automatically deploy the code to production or a staging environment, making the software delivery process faster and more reliable.

Pipeline as Code: Jenkins supports defining CI/CD pipelines as code using a Domain-Specific Language (DSL). This allows teams to version control their build pipelines along with their codebase.

Example Workflow with Jenkins:

Commit Code: A developer commits code to a version control system like GitHub.

Automated Build: Jenkins detects the commit and triggers an automated build of the code.

Testing: Jenkins runs automated tests to validate the code changes.

Deployment: If tests pass, Jenkins can automatically deploy the application to a staging or production environment.

Software / Tools Setup for React Experiments (Exp 3–5)

Software Required: -

1. Node.js (LTS Version)
2. npm (comes with Node.js)
3. Visual Studio Code

Installation Steps: -

Step 1 – Install Node.js

Download from https://nodejs.org

Verify installation in terminal/command prompt: **node -v**

Step 2 – Install Visual Studio Code

1. Download from https://code.visualstudio.com.
2. Install extensions (optional): *React Developer Tools*, *Prettier*.

Step 3 – Create a New React Project

Run the following commands in Command Prompt/Terminal:

1. **npx create-react-app my-app**
   - **npx** → Runs a package without installing it globally.
   - **create-react-app** → Generates a new React project with default setup.
   - **my-app** → Name of the project folder.
2. **cd my-app**
   - **cd** means Change Directory.
   - This command moves into the new **my-app** folder that was just created.
3. **npm start**
   - **npm** is Node Package Manager.
   - **start** runs the React development server.
   - The application will open in the browser at **http://localhost:3000**

Step 4 – Add Your Experiment Code

- Open the project in VS Code.
- Replace the contents of **src/App.js** and **src/App.css** with your experiment code.

3. Create a code using HTML, React, JS(Registration Page, Login Page, Feedback Form) :-

>>>App.js

```
import React, { useState } from 'react';
import './App.css';
//>>>Registration Page
function RegistrationPage({ onSwitch }) {
 const Register = () => {
  alert("Registration Successful");
  onSwitch("login");
 };
 return (
  <div className="form-container">
   <h2>Registration Page</h2>
   <form onSubmit={Register}>
    <input type="text" placeholder="First Name" required /><br />
    <input type="text" placeholder="Last Name" required /><br />
    <input type="email" placeholder="Email" required /><br />
    <input type="password" placeholder="Password" required /><br />
    <button type="submit">Register</button>
   </form>
  </div>
 );
```

```
}
//>>>Login page
function LoginPage({ onSwitch }) {
  const Login = () => {
    alert("Login Successful");
    onSwitch("feedback");
  };

  return (
    <div className="form-container">
      <h2>Login Page</h2>
      <form onSubmit={Login}>
        <input type="email" placeholder="Email" required /><br />
        <input type="password" placeholder="Password" required /><br />
        <button type="submit">Login</button>
      </form>
    </div>
  );
}
//>>>Feed Back Form page
function FeedbackForm({ onSwitch }) {
  const [rating, setRating] = useState(0);
```

```
const Feedback = () => {
  alert(`Feedback submitted. Rating: ${rating} star(s). Thank you!`);
};

return (
  <div className="form-container">
    <h2>Feedback Form</h2>
    <form onSubmit={Feedback}>
      <input type="text" placeholder="Your Name" required /><br />
      <input type="email" placeholder="Your Email" required /><br />

      <div className="stars">
        {[1, 2, 3, 4, 5].map((star) => (
          <span
            key={star}
            onClick={() => setRating(star)}
            style={{
              cursor: 'pointer',
              fontSize: '20px',
              color: star <= rating ? 'gold' : 'gray'
            }}
          >
            ★
```

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

      <textarea placeholder="Your Feedback" required></textarea><br />
      <button type="submit">Submit</button><br />
    </form>
  </div>
 );
}

function App() {
 const [page, setPage] = useState('register');

 return (
  <div className="App">
   {page === 'register' && <RegistrationPage onSwitch={setPage} />}
   {page === 'login' && <LoginPage onSwitch={setPage} />}
   {page === 'feedback' && <FeedbackForm onSwitch={setPage} />}
  </div>
 );
}
```

```
export default App;
```

>>>App.css

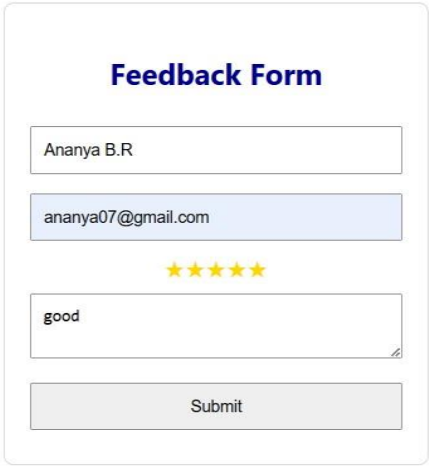
```
.form-container {
  width: 300px;
  margin: 50px auto;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 8px;
  text-align: center;
  color:darkblue;}
input, textarea, button {
  width: 100%;
  margin: 8px 0;
  padding: 10px;
  box-sizing: border-box;}
```

Output: -

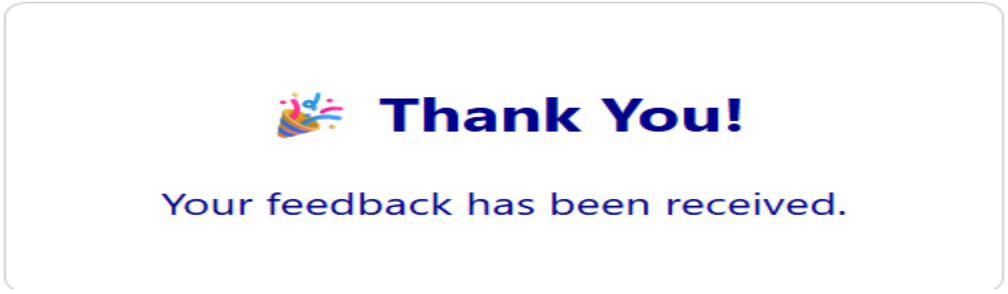
*Registration Page

*Login page

*Feed Back form



*Thank you page



4.Create code using JavaScript a Shopping Cart .

>>>App.js :-

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


const Header = ({ count, Show }) => (
 <div className="shopping-card">
  <div onClick={() => Show(false)}>ShoppingCart App</div>
  <div onClick={() => Show(true)}>Cart <sup>{count}</sup></div>
 </div>
);
//>>>Product list
const ProductList = ({ product, addToCart }) => (
 <div className="product-list">
  {product.map((item, i) => (
   <div key={i} className="product-item">
    <img src={item.url} alt={item.name} width="100%" />
    <p>{item.name} | {item.category}</p>
    <p>{item.seller}</p>
    <p>Rs. {item.price}</p>
    <button onClick={() => addToCart(item)}>Add To Cart</button>
   </div>
  ))}
 </div>
);
```

//>>>Cartlist

```
const CartList = ({ cart }) => {
  const [CART, setCART] = useState([]);
  useEffect(() => setCART(cart), [cart]);

  const total = CART.reduce((sum, item) => sum + item.price * item.quantity, 0);

  return (
    <div className="cart-list">
      {CART.map((item, i) => (
        <div className="cart-item">
          <img src={item.url} alt={item.name} width={40} />
          <span>{item.name}</span>
          <span>Rs. {item.price * item.quantity}</span>
        </div>
      ))}
      <p><strong>Total:</strong> Rs. {total}</p>
    </div>
  );
};

export default function App() {
  const [products] = useState([
    {
      url: "https://encrypted-
tbn3.gstatic.com/shopping?q=tbn:ANd9GcQhJV2cVOiGPfLP0Wb5mJtSlANZIT1cOCr
Zg68V9xjrh0-
GG85WycSVOj2gZXdh1xpD2rjm6v4Ko4zs6rN_BrKtd9V3uX251td9AqDxgLiDtQ4SS8
qIZo4I",
```

```
    name: "IPHONE 16PRO MAX",
    category: "IPHONE",
    seller: "APPLE",
    price: 70000,
  },
 ]);

 const [cart, setCart] = useState([]);
 const [showCart, setShowCart] = useState(false);
 const addToCart = item => setCart([...cart, { ...item, quantity: 1 }]);

 return (
  <div>
   <Header count={cart.length} Show={setShowCart} />
   {showCart ? <CartList cart={cart} /> : <ProductList product={products}
addToCart={addToCart} />}
  </div>
 );
}
```

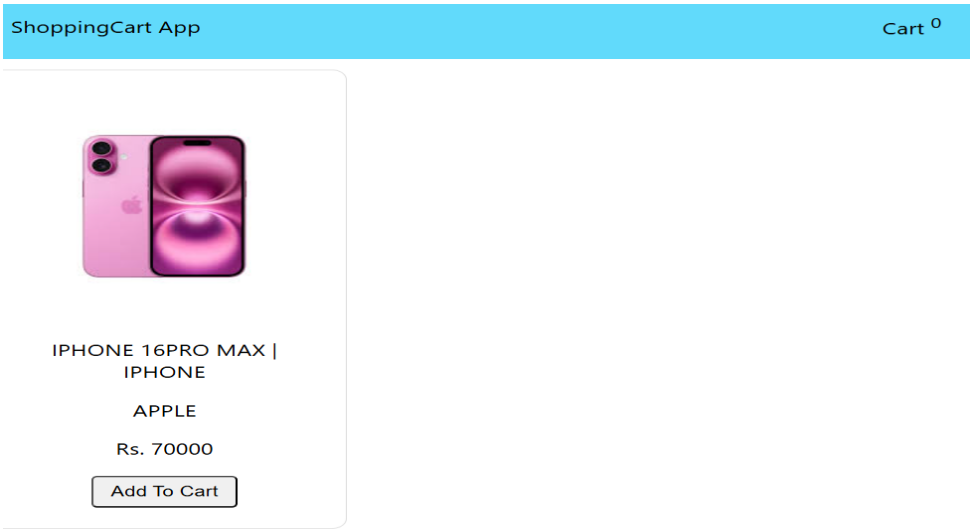
>>>App.css

```
.flex, .shopping-card, .cart-item { display: flex; }
.flex { flex-wrap: wrap; }
.shopping-card {
 justify-content: space-between;
 align-items: center;
 background: #61dafb;
 padding: 20px 30px;
 font: bold 20px;
```

```
  cursor: pointer;
}
.product-item, .cart-item {
  padding: 20px; margin: 10px; text-align: center;
  border-radius: 10px; border: 1px solid #ddd;
  align-items: center;
}
.product-item { width: 30%; }
.product-item img {
  max-height: 350px;
  margin-bottom: 10px;
}
.product-item button, .cart-item button {
  padding: 6px 12px; font-size: 14px;
  border-radius: 4px; cursor: pointer;
}
```

Output:-

*Shopping cart

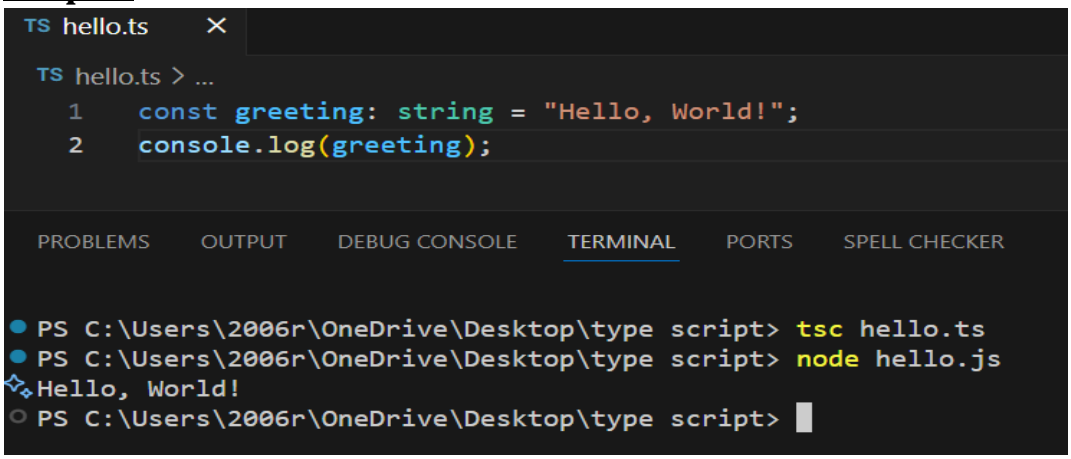


*Add to Cart

>>>Hello World(TypeScript):-

```
const greeting: string = "Hello, World!";

console.log(greeting);
```

Output:-



5.Create a code using React Static and Dynamic, and Create a code using React counter(Hook concept):-

>>>App.js:-

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes, Link, useNavigate }
from 'react-router-dom';
import './App.css';
//>>> Login Component
function Login() {
 const navigate = useNavigate();

 const Submit = () => {
  navigate('/');
 };

 return (
  <form onSubmit={Submit}>
   <center>
    <h2>Login</h2>
    <input type="text" placeholder="Name" required ><br/>
    <input type="password" placeholder="Password" required /><br
/>
    <button type="submit" style={{ backgroundColor: 'red',color:
'white' }}>
      Submit
    </button>
   </center>
  </form>
```

```
  );
}
// >>>Home Component
function Home() {
  return (
    <div style={{ backgroundColor: 'lightgrey' }}>
      <center>
        <h2>Home</h2>
        <p>Welcome to VSNS website</p>
      </center>
    </div>
  );
}
// >>>AboutUs Component
function AboutUs() {
  return (
    <center>
      <div style={{ backgroundColor: 'lightgrey' }}>
        <h2>About Us</h2>
        <p>About us</p>
      </div>
    </center>
  );
}
//>>> Contact Component
function Contact() {
  return (
    <div style={{ backgroundColor: 'lightgrey' }}>
```

```
    <center>
      <h2>Contact</h2>
      <p>Contact details</p>
    </center>
   </div>
  );
}
//>>> Main App Component
function App() {
  return (
   <Router>
    <div className="App">
     <nav className="App-nav">
      <ul>
       <li><Link to="/login">Login</Link></li>
       <li><Link to="/">Home</Link></li>
       <li><Link to="/about">About Us</Link></li>
       <li><Link to="/contact">Contact</Link></li>
      </ul>
     </nav>

     <Routes>
      <Route path="/login" element={<Login />} />
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<AboutUs />} />
      <Route path="/contact" element={<Contact />} />
     </Routes>
    </div>
```

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

>>>App.css:-

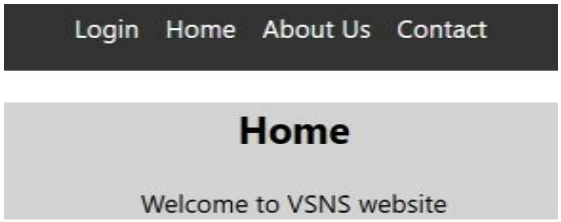
```
.App-nav {
  background: #333;
  color: #fff;
  padding: 1rem;
}
.App-nav ul {
  list-style: none;
  padding: 0;
  margin: 0;
  display: flex;
  justify-content: center;
  align-items: center;
}
.App-nav li {
  margin-right: 1rem;
}
.App-nav a {
  color: #fff;
  text-decoration: none;
}
.App-nav a:hover {
```

```
  text-decoration: underline;
}
```

Output:-

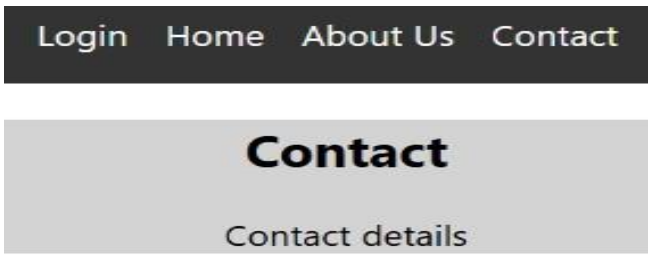
***Login page**

***Home page**



***About us page**

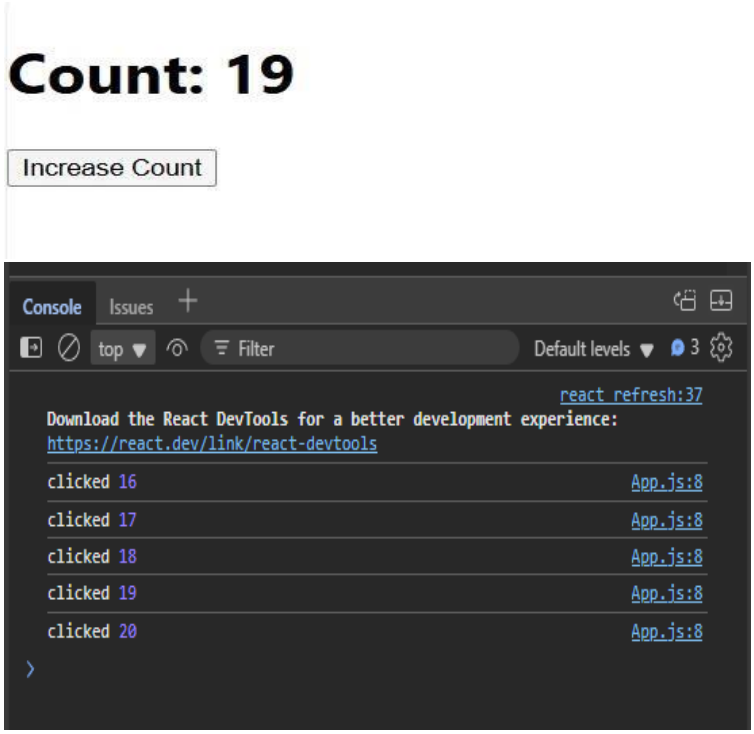
***Contact Page**



>> Counter (with out using Hook):-

```
import React from 'react'; function
App() {
    let count = 19;


    const addValue = () => {
        count += 1;
        console.log("clicked", Math.random()); console.log("clicked", count);
    };


    return (
        <div>
            <h1 id="count-display">Count: {count}</h1>
                        <button onClick={addValue}>Increase Count</button>
        </div>
    );
}


export default App;
```

Output:-



>> Counter(Hook concept):-

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


function Counter() {
    // Initialize the count state with a value of 0 const [count,
    setCount] = useState(0);


    // Function to increase the count const
    increaseCount = () => {
        setCount(count + 1);
    };


    return (
        <div>
```

```
            <h1>Count: {count}</h1>
                        <button onClick={increaseCount}>Increase</button>
        </div>
    ); }

export default Counter;
```

Output :-

6. Steps to add My SQL Connector to Eclipse

>>>Steps to add JDBC to Eclipse :-

Step 1 : Install My SQL connector from browser.

Step 2 : Unzip the file.

Step 3 : Copy the file Path.

Step 4 : Open the eclipse , right click on the eclipse go for the properties .

Step 5 : Choose java build path, go for library.

Step 6 : Paste the following path to the Module path.

Step 7: Apply and close.

Step 9: Open the my SQL command line and create a database

6. Database connectivity (Eclipse):-

```
package ananya;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class connectors {

public static void main(String[] args) throws SQLException {
Connection con = DriverManager.getConnection(
"jdbc: mysql://localhost:3306/ananya”,”root”,”ananya123");
Statement stat = con.createStatement();
String S1 = "delete from anu where rollno='26'";
String S2 = "update anu SET name='BerrySubbi’ where rollno='27'";
String S3 = "insert into boo values('05','rossie','15')";

stat.execute(S1);
stat.execute(S2);
stat.execute(S3);
System.out.println("success");
con.close();
}
}
```

Output: -

*Insert

```
+------+------------+--------+
| id   | name       | rollno |
+------+------------+--------+
| 01   | BerrySubbi | 27     |
| 07   | ananya r   | 21     |
+------+------------+--------+
2 rows in set (0.00 sec)

mysql> insert into anu values('05','rossie','15');
Query OK, 1 row affected (0.02 sec)

mysql> select *from anu;
+------+------------+--------+
| id   | name       | rollno |
+------+------------+--------+
| 01   | BerrySubbi | 27     |
| 07   | ananya r   | 21     |
| 05   | rossie     | 15     |
+------+------------+--------+
3 rows in set (0.00 sec)
```

*Update

```
+------+----------+--------+
| id   | name     | rollno |
+------+----------+--------+
| 01   | berry    | 27     |
| 07   | ananya r | 21     |
+------+----------+--------+
2 rows in set (0.00 sec)

mysql> update anu SET name='BerrySubbi' where rollno='27';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select *from anu;
+------+------------+--------+
| id   | name       | rollno |
+------+------------+--------+
| 01   | BerrySubbi | 27     |
| 07   | ananya r   | 21     |
+------+------------+--------+
2 rows in set (0.00 sec)
```

*Delete

```
mysql> select *from anu;
+------+------------+---------+
| id   | name       | rollno  |
+------+------------+---------+
| 01   | berry      | 27      |
| 07   | ananya r   | 21      |
| 08   | chaithanya | 26      |
+------+------------+---------+
3 rows in set (0.00 sec)

mysql> delete from anu where rollno='26'
    -> ;
Query OK, 1 row affected (0.02 sec)

mysql> select *from anu;
+------+----------+---------+
| id   | name     | rollno  |
+------+----------+---------+
| 01   | berry    | 27      |
| 07   | ananya r | 21      |
+------+----------+---------+
2 rows in set (0.00 sec)
```

Using Spring Initializer (Spring IO) from experiment (7-8)

1. **Go to Spring Initializer**: Open https://start.spring.io in your browser.
2. C**onfigure your Project**:

- **Project**: Select **Maven Project**
- **Language**: Choose **Java**.
- **Spring Boot Version**: Choose the Spring Boot version (3.55)
- **Project Metadata**:

- Group: e.g., com.example

- Artifact: e.g., demo

- Name: e.g., demo

· **Packaging:** Either Jar or War, depending on your requirements.

. Java Version: Set the appropriate Java version for your project **(e.g.,** 11 or 17).

3.Add Dependencies: In the **"Dependencies"** section, start typing to search for required dependencies. For example:

- **Spring Web**: For building web applications (REST APIs, MVC).
- **Spring Data JPA**: For interacting with databases using JPA.
- **Spring Boot DevTools**: For development tools (optional).
- **H2 Database**: If you need an in-memory database.
- **Spring Context:** For core Spring dependencies (like Dependency injection)
- **Spring Security:** For spring security

4.Generate the Project: Click **Generate** to download .zip file

5.Unzip and Import: Unzip the downloaded project and import it into your Eclipse

7 a. Create a REST API with Spring security using spring boot :-

This experiment demonstrates how to create a secure REST API using Spring Boot with Spring Security, H2 Database, and Spring Data JPA.

Step 1: Generate a Spring Boot Project

1. Open your browser and go to https://start.spring.io (Spring Initializr).
2. Configure the project as follows:
    - Project: Maven
    - Language: Java
    - Spring Boot Version: Latest stable version
    - Packaging: Jar
    - Java: 17
3. Add the following dependencies:
    - Spring Web
    - Spring Security
    - Spring Data JPA
    - H2 Database
4. Click Generate to download the project as a .zip file.
5. Extract the .zip file to a suitable location.

Step 2: Import Project into Eclipse

1. Open Eclipse IDE.
2. Go to File → Import → Existing Maven Project.
3. Browse and select the extracted project folder.
4. Click Finish.
5. Wait until Maven finishes downloading the required dependencies.

Step 3: Create Java Classes

Inside **src/main/java/com/example/demo/**, create the following files:

>>Spring Boot Security (Spring tool)

>>>MessageController.java :-

```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class MessageController {
@GetMapping("/")
public String getMessage() {
return "<h1>welcome to spring boot security</h1>";
}
}
```

>>>DemoApplication.java :-

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

@SpringBootApplication
@EnableWebSecurity
public class SpringSecurityApplication {
public static void main(String[] args) {
SpringApplication.run(SpringSecurityApplication.class,args);
}
```

}

>>>ApplicationProperties :-

spring.security.user.name=boo;
spring.security.user.password=2701;

Output: -

***Login with password**

***Then next output**

7 b. Create Hello World Program using REST FULL API Services: -

This experiment demonstrates how to create a simple RESTful web service using Spring Boot.

Step 1: Install Spring Tool Suite (STS)

1. Download Spring Tool Suite (STS) from:
   https://spring.io/tools
2. Extract the downloaded file.
3. Launch STS by running the executable file.

Step 2: Create a New Spring Starter Project

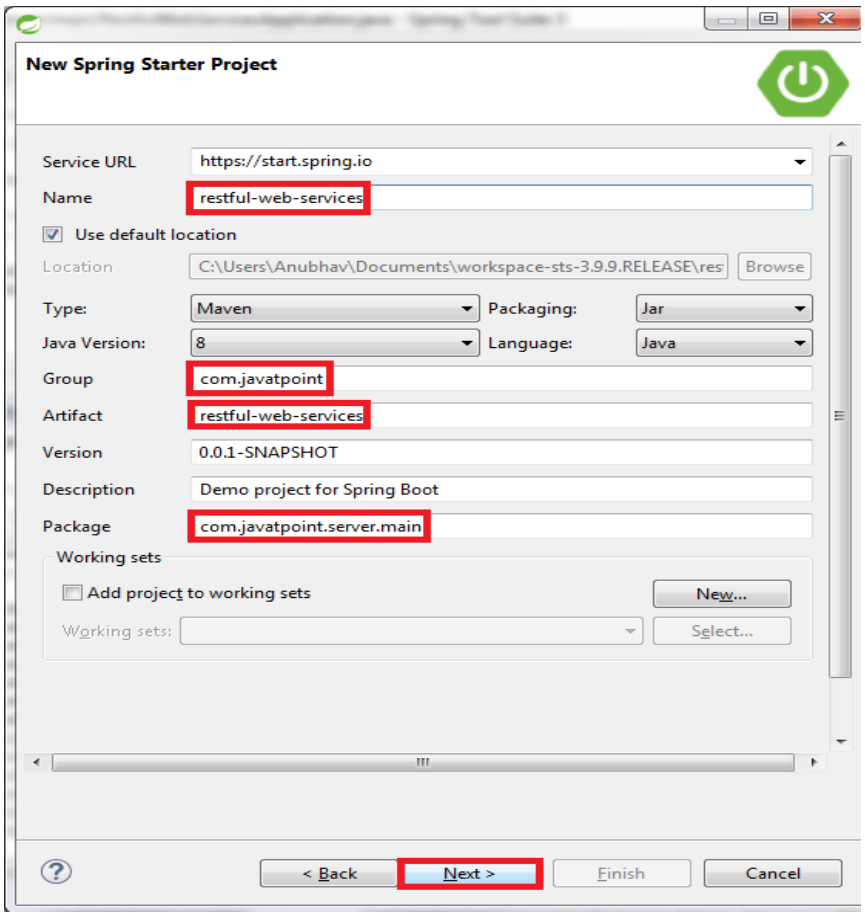
1. In STS, go to File → New → Spring Starter Project.
2. If not listed, click on Other..., type Spring Starter Project, select it, then click Next.

Step 3: Provide Project Information

Fill in the project details as follows:

- **Name:** restful-web-services
- **Group:** com.javatpoint
- **Package:** com.javatpoint.server.main

Click **Next**.

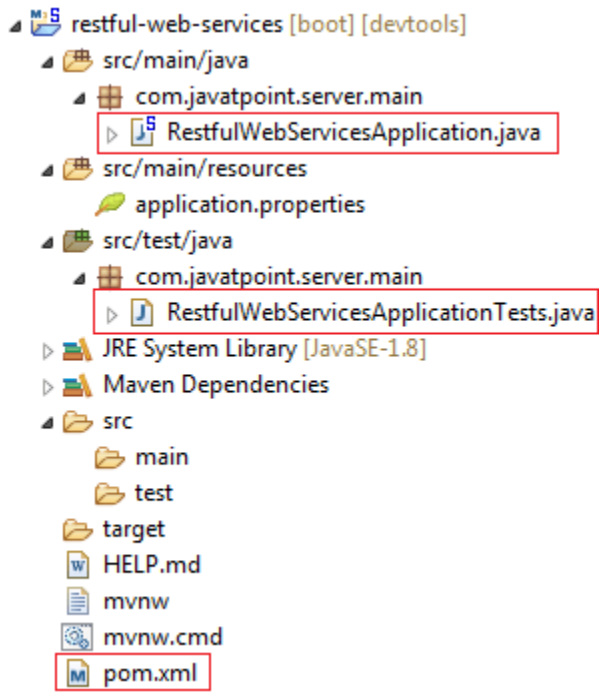


Step 4: Add Dependencies

Choose **Spring Boot Version 2.1.8** and add the following dependencies:

- Spring Web (Spring MVC)
- Spring Boot DevTools
- Spring Data JPA
- H2 Database

Step 5: We can see the project structure in the project explorer window.



Step 6: Create a REST Controller

Create a new class in the package com.javatpoint.server.main:

>>>HelloWorldContoller.java :-

```
package com.javapoint.server.main;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

//controller

@RestController

public class HelloWorldController

{

  @GetMapping("/hello")

  public String helloWorld()

  {


    return "Hello World";
```

```
  }
  @GetMapping("/helloworld").

  public HelloWorldBean helloWorldBean()
  {
     return new HelloWorldBean("Hello World");
  }
}
```

>>>HelloWorldBean.java :-

```
package com.javapoint.server.main;

public class HelloWorldBean {
   public String message;
   public HelloWorldBean(String message) {
      this.message=message;
   }
   //generating getter and setter
   public String getMessage() {
      return message;
   }
   public void setMessage(String message)
   {
      this.message=message;
   }}
```

>>>DemoApplicationjava :-

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class DemApplication {


public static void main(String[] args) {

        SpringApplication.run(DemApplication.class, args);

}


}
```

Output :-

***localhost:8080/helloworld**

8. Write the code for Constructor and setter injection(in spring) :-

>> Constructor Injection Code (Eclipse):-

>>>Engine.java

```
package com.example.demo;
import org.springframework.stereotype.Component;
@Component
public class Engine {
public void start() {
 System.out.println("Engine started...");
  }
}
```

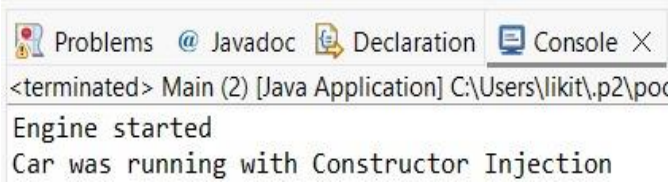
>>>Car.java

```
package com.example.demo;
import org.springframework.stereotype.Component;
@Component
public class Car {
private final Engine engine;
// Constructor Injection
public Car(Engine engine) {
 this.engine = engine;
  }
 public void drive() {
 engine.start();
System.out.println("Car is driving!");
 }}
```

>>>Main.java

```
package com.example.demo;


import org.springframework.stereotype.Component;


@Component
public class Car {
   private final Engine engine;


   // Constructor Injection
   public Car(Engine engine) {
      this.engine = engine;
   }


   public void drive() {
      engine.start();
      System.out.println("Car is driving!");
   }
}
```

Output: -



>>Setter Injection code (Eclipse):-

>>>Engine.java:

```
package com.example.demo;
import org.springframework.stereotype.Component;
@Component
public class Engine {
   public void start() {
      System.out.println("Engine started...");
   }
}
```

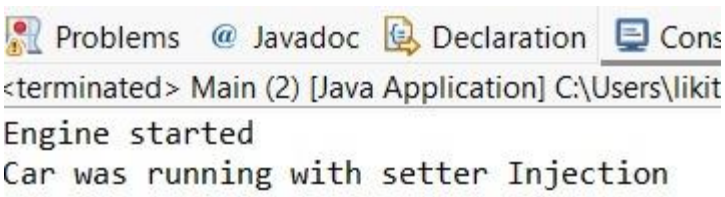
>>>Car.java

```
package com.example.demo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class Car {
   private Engine engine;
   // ☑ Setter Injection
   @Autowired
   public void setEngine(Engine engine) {
      this.engine = engine;
   }
   public void drive() {
      engine.start();
      System.out.println("Car is driving with setter injection...");}}
```

>>>Main.java :-

```
package com.example.demo;


import org.springframework.stereotype.Component;


@Component
public class Car {
   private final Engine engine;


   // Constructor Injection
   public Car(Engine engine) {
      this.engine = engine;
   }


   public void drive() {
      engine.start();
      System.out.println("Car is driving!");
   }
}
```

Output:-



9. Create REST controller for CRUD operations and test with PostMan (MongoDB) :-

1. MongoDB Installation Process

1.Go to MongoDB Community Server Download.

2.Download the installer for your OS.

3.Run the installer → choose Complete Setup.

4.After installation, start the MongoDB server.

2. Postman Installation Process

1.Download Postman from https://www.postman.com/downloads/.

2.Install and open it.

3.You will use Postman to send API requests (GET, POST, PUT, DELETE).

3. Project, Folder and File Creation in Eclipse

1.Open Eclipse IDE → File → New → Project → Crud(name of the project)

2.Inside project create a folder (models).

3.Inside it, create:

- index.js → main server file (Crud)
- models/user.js → /model/

4.Initialize npm command in terminal:

npm init -y

npm install express mongoose

>>>index.js :-

```
const express = require('express'); const
mongoose = require('mongoose');


const app = express();


app.use(express.json());


mongoose.connect('mongodb://localhost:27017/mongodb',
useNewUrlParser: true,
useUnifiedTopology: true,
})
.then(() => console.log('Connected to MongoDB'))
.catch(err => console.error('Could not connect to MongoDB', err));


 app.listen(3000, () => console.log('Server running on http://localhost:3000'));


const User = require('./models/user');


app.post('/users', async (req, res) => {
try {
                const user = new User(req.body); await
                user.save(); res.status(201).send(user);
        } catch (err) { res.status(400).send(err);
```

```
        }
});


app.get('/users', async (req, res) => { try {
        const users = await User.find();
        res.send(users);
        } catch (err) { res.status(500).send(err);
        }
});


app.get('/users/:id', async (req, res) => {
 try {
 const user = await User.findById(req.params.id);
 if (!user) return res.status(404).send('User not found');
res.send(user);
} catch (err) { res.status(500).send(err);
        }
});
app.put('/users/:id', async (req, res) => {
    try {
    const user = await User.findByIdAndUpdate(req.params.id,
req.body, { new: true, runValidators: true });
 if (!user) return res.status(404).send('User not found');
res.send(user);
```

```
    } catch (err) { res.status(400).send(err);
    }
});
app.delete('/users/:id', async (req, res) => {
    try { const user = await User.findByIdAndDelete(req.params.id);
    if (!user) return res.status(404).send('User not found');
    res.send('User deleted');
    } catch (err) { res.status(500).send(err);
    }
});
```

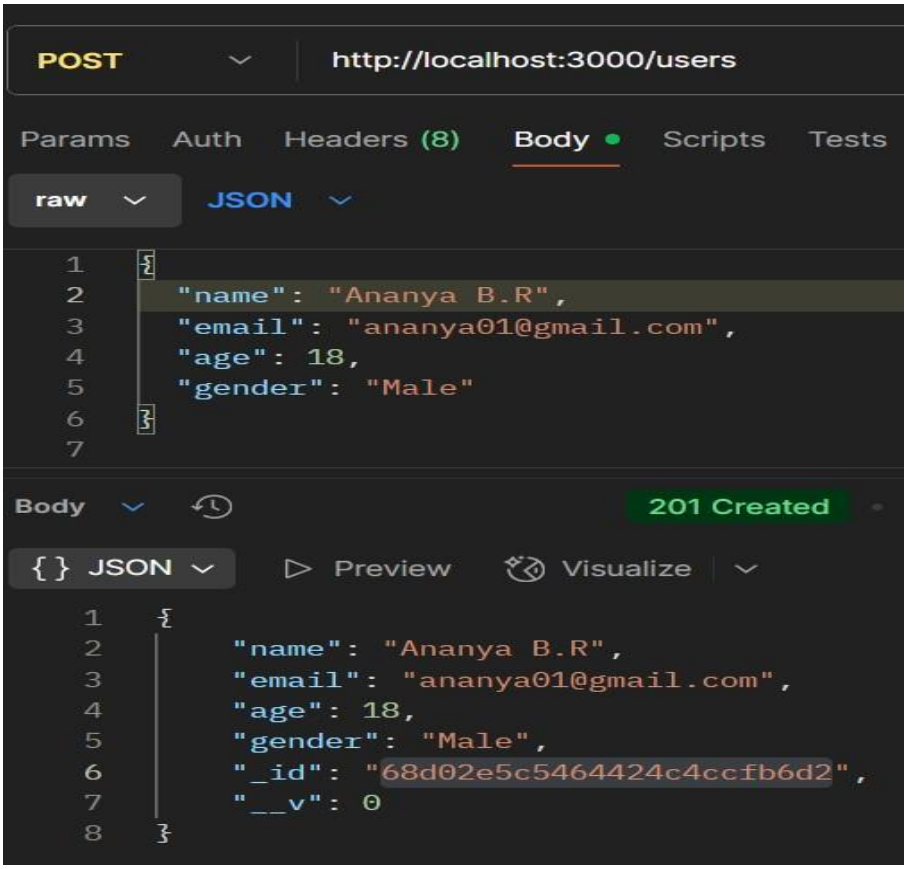
>>>User.js :

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
        name: {
                type: String,
                 required: true
        },
        email: { type: String,
                required: true,
                unique: true
        },
        age: { type: Number,
                required: true
```

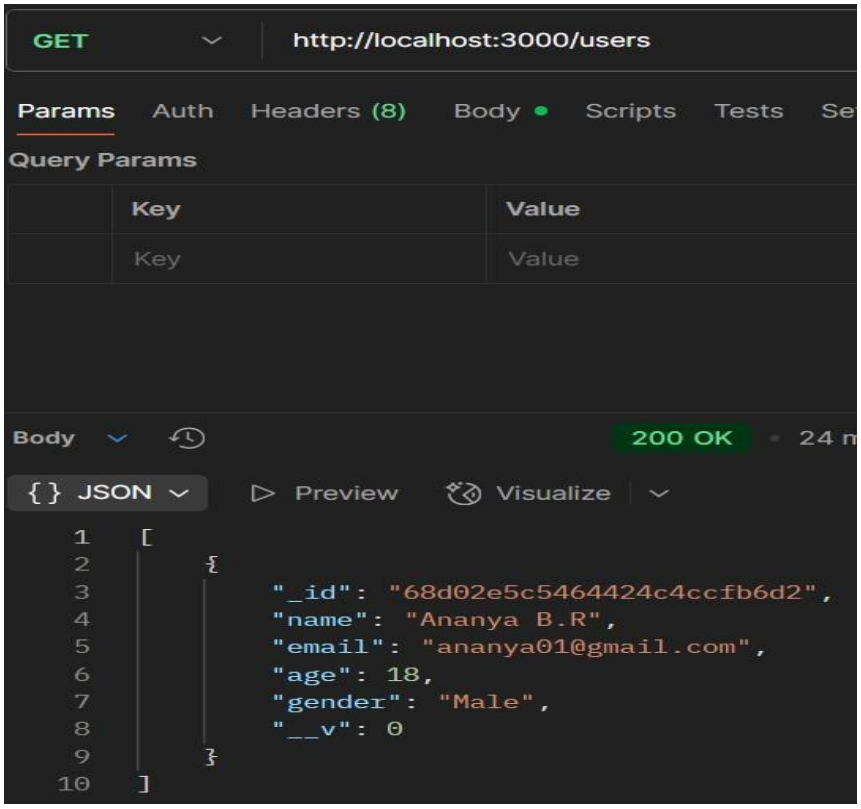
```
    },
    gender: { type: String,
        required: true
    }
});
const User = mongoose.model('User', userSchema);
module.exports = User;
```

Output:-

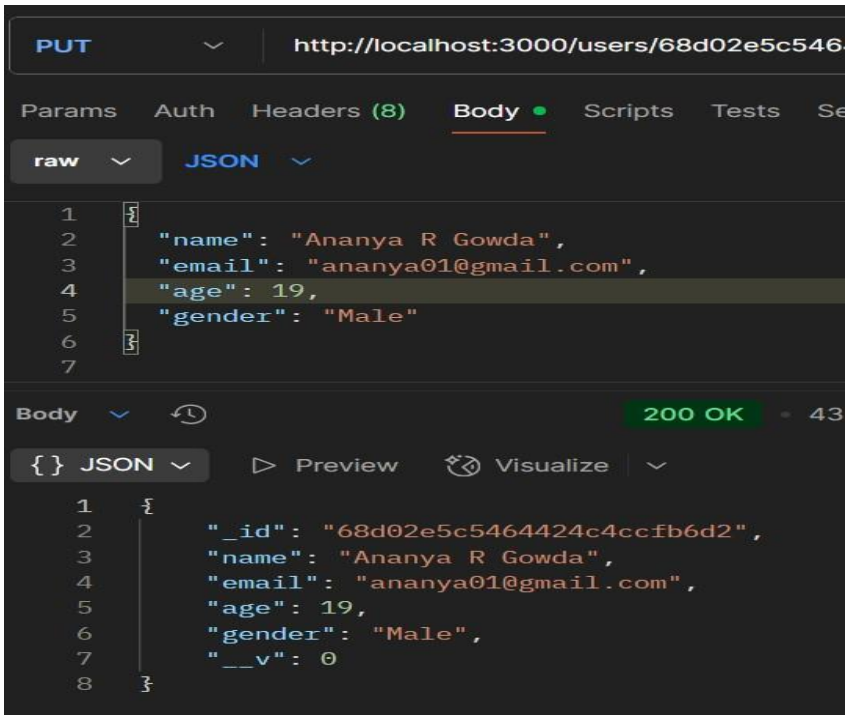
1. POSTMethod – Send data to server (create)



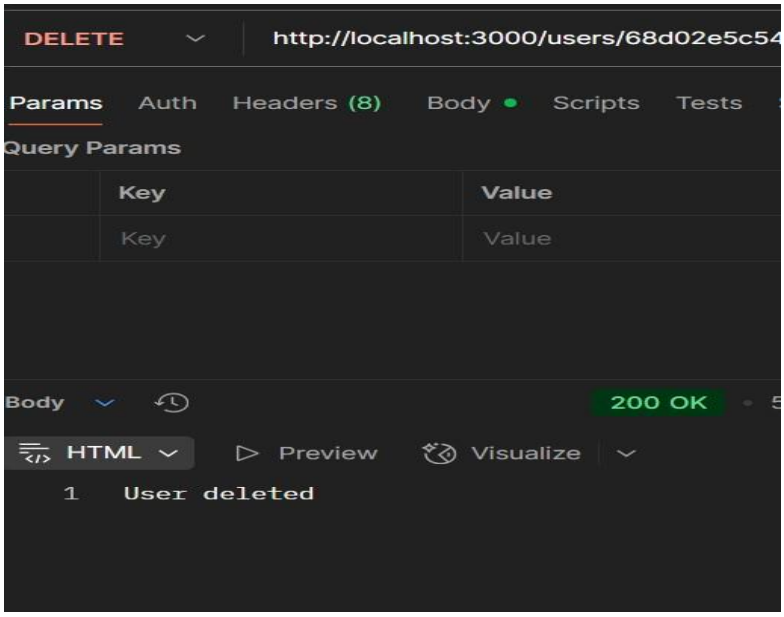
2. <u>GET Method</u> – Retrieve data from server



3. <u>PUT Method</u>– Update data



***Delete Method: -** Remove data



10.Create Data base by name college and a collection by name student :-

A. Insert any five documents into above collection.

```
test> use college
switched to db college
college> db.student.insertMany([
...    { name: "manu", rollno: 12, branch: "ME" },
...    { name: "mango", rollno: 5, branch: "EC" },
...    { name: "mam", rollno: 1, branch: "CS" },
...    { name: "jac", rollno: 11, branch: "EE" },
...    { name: "man", rollno: 12, branch: "EEE" }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68cfcc6f43cf9af4decebea4'),
    '1': ObjectId('68cfcc6f43cf9af4decebea5'),
    '2': ObjectId('68cfcc6f43cf9af4decebea6'),
    '3': ObjectId('68cfcc6f43cf9af4decebea7'),
    '4': ObjectId('68cfcc6f43cf9af4decebea8')
  }
}
```

B. Display all the documents present in collection student.

```
college> db.student.find()
[
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea4'),
    name: 'manu',
    rollno: 12,
    branch: 'ME'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea5'),
    name: 'mango',
    rollno: 5,
    branch: 'EC'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea6'),
    name: 'mam',
    rollno: 1,
    branch: 'CS'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea7'),
    name: 'jac',
    rollno: 11,
    branch: 'EE'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea8'),
    name: 'man',
    rollno: 12,
    branch: 'EEE'
  }
]
```

C. Display only name of branch of students from student documents.

```
college> db.student.find({}, { branch: 1, _id: 0 })
[
  { branch: 'ME' },
  { branch: 'EC' },
  { branch: 'CS' },
  { branch: 'EE' },
  { branch: 'EEE' }
]
```

D. Update the branch of student from ME to EC.

```
college> db.student.updateOne(
...     { branch: "ME" },
...     { $set: { branch: "EC" } }
... )
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
college> db.student.find()
[
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea4'),
    name: 'manu',
    rollno: 12,
    branch: 'EC'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea5'),
    name: 'mango',
    rollno: 5,
    branch: 'EC'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea6'),
    name: 'mam',
    rollno: 1,
    branch: 'CS'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea7'),
    name: 'jac',
    rollno: 11,
    branch: 'EE'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea8'),
    name: 'man',
    rollno: 12,
    branch: 'EEE'
  }
]
```

E. Delete all the documents whose branch=’EC’.

```
college> db.student.deleteMany({ branch: "EC" })
{ acknowledged: true, deletedCount: 2 }
college> db.student.find()
[
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea6'),
    name: 'mam',
    rollno: 1,
    branch: 'CS'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea7'),
    name: 'jac',
    rollno: 11,
    branch: 'EE'
  },
  {
    _id: ObjectId('68cfcc6f43cf9af4decebea8'),
    name: 'man',
    rollno: 12,
    branch: 'EEE'
  }
]
```

11. Demonstarte commands to create a docker file, build docker image with docker file, create docker container from docker image and run the docker container.

*Dokerfile (visual studio) :-

```
FROM python:3.9-slim

WORKDIR /app

COPY . /app

CMD ["python", "-c", "print('Hello World')"]
```

*Commands to explore Docker image (VS terminal) :-

Step 1 : docker build -t hello-world-image .

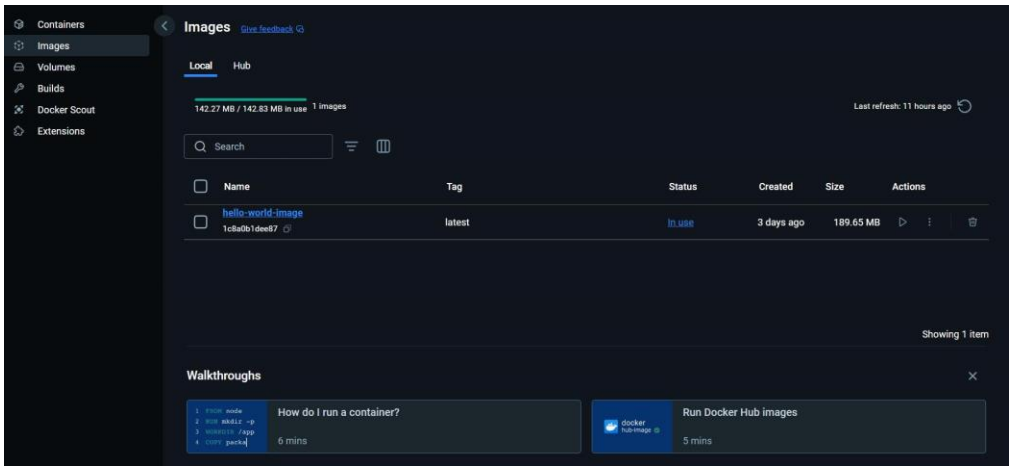
Step 2 : docker images.

Step 3 : docker create --name hello-world-container hello-world-image.

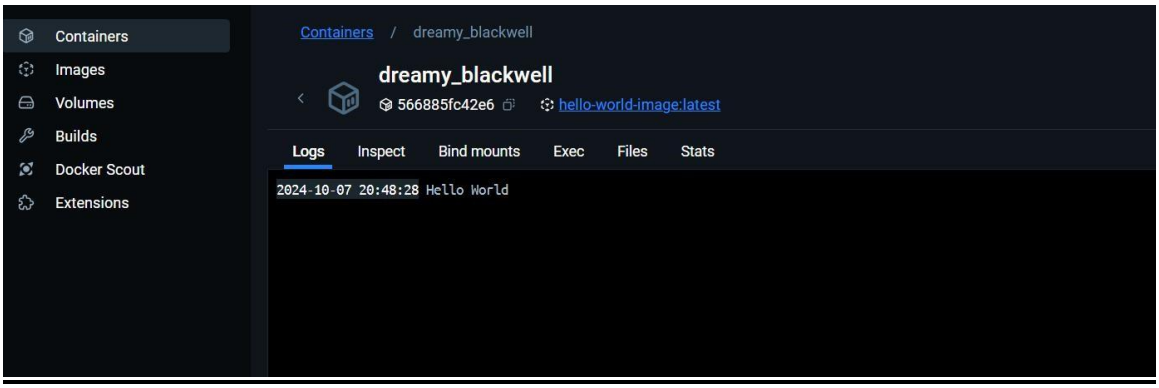
Step 4 : docker start hello-world-container .

Step 5 : docker logs hello-world-container.

*Run the Docker file in Dockerfile using Docker application (Docker) :-



Output :-


Containers
Images
Volumes
Builds
Docker Scout
Extensions
Containers / dreamy_blackwell
dreamy_blackwell
566885fc42e6
hello-world-image:latest
Logs
Inspect
Bind mounts
Exec
Files
Stats
2024-10-07 20:48:28 Hello World