

Exp No	Experiment Name
01.	Create a user story for Hospital management
02	Create a Local repository, clone repository using GitHub and Explain about Jenkins
03	Create a code using HTML, React, JS(Registration Page, Login Page, Feedback Form)
04	Create code using JavaScript a Shopping Cart .
05	Create a code using React Static and Dynamic, and Create a code using React counter(Hook concept )
06	Database connectivity ( Eclipse )
07	a. Create a REST API with Spring security using spring boot b. Create Hello World Program using Spring
08	Write the code for Constructor and setter injection( in spring )
09	Create REST controller for CRUD operations and test with Postman (MongoDB)
10	Create Data base by name college and a collection by name student
11	Demonstrate commands to create a docker file, build docker image with docker file, create docker container from docker image and run the docker container.

# 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 multifactor 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.

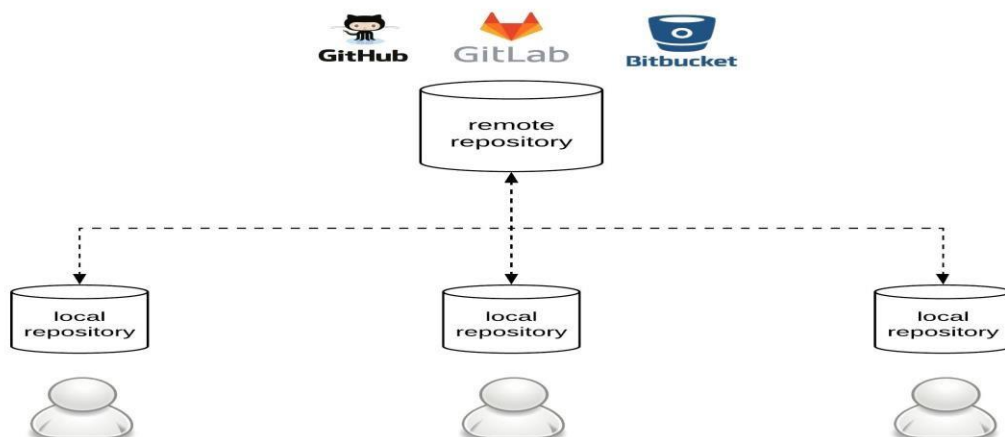


Fig : GitHub Local Repository

## 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>

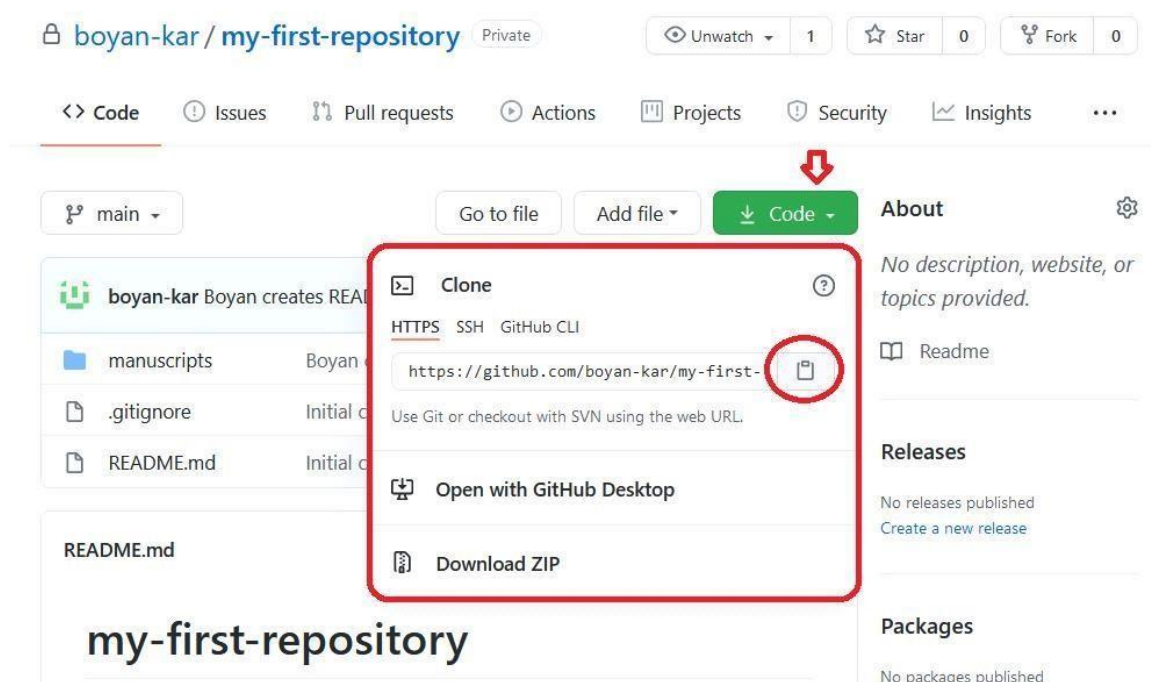


Fig : GitHub Clone Repository

## >> 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 DomainSpecific 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';

function RegistrationPage({ onSwitch }) {
  const Register = () => {
    alert('Registration Successful!');
    onSwitch('login');
  };

  return (
    <div className="form-container">
      <h2>Registration</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><br />
      </form>
    </div>
  );
}
```

```
function LoginPage({ onSwitch }) {  
  const Login = () => {  
    alert('Login Successful!');  
    onSwitch('feedback');  
  };  
  
  return (  
    <div className="form-container">  
      <h2>Login</h2>  
      <form onSubmit={Login}>  
        <input type="email" placeholder="Email" required /><br />  
        <input type="password" placeholder="Password" required /><br />  
        <button type="submit">Login</button><br />  
      </form>  
    </div>  
  );  
}
```

```
function FeedbackForm({ onSwitch }) {  
  const Feedback = () =>{  
    alert('Feedback submitted. 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 />
    <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;  
}
```

```
.link {  
color: blue;  
cursor: pointer;  
text-decoration: underline;  
}
```

### Registration

nagalakshmi

kg

nagalakshmi@gmail.com

...

Register

### Login

nagalakshmi@gmail.com

...

Login

### Feedback Form

nagalakshmi

nagalakshmi@gmail.com

good

Submit

## 4.Create code using JavaScript a Shopping Cart .

### App.js

```
import './App.css';

import Header from './component/Header';
import ProductList from './component/ProductList';
import CartList from './component/Cart';
import { useState } from 'react';

function App() {
  const [products] = useState([
    {
      url: "https://5.imimg.com/data5/KC/PC/MY-38629861/dummy-chronograph-
watch-500x500.jpg",
      name: "LOREMWatchBlack",
      category: "Watches",
      seller: "Watch Ltd Siyana",
      price: 2599,
    },
  ]);

  const [cart, setCart] = useState([]);
  const [showCart, setShowCart] = useState(false);

  const addToCart = (item) => {
```

```

    setCart([
      ...cart,
      { ...item, quantity: 1 }
    ]);
  };

  return (
    <div className="App">
      <Header />
      <ProductList products={products} addToCart={addToCart} />
      {showCart && <CartList cart={cart} />}
    </div>
  );
}

export default app.js;

```

### **header.js**

```

function Header({ handleShow, count }) {
  return (
    <div className='flex shopping-card'>
      <div onClick={() => handleShow(false)}>
        Shopping Cart App
      </div>
      <div onClick={() => handleShow(true)}>
        Cart <sup>{count}</sup>
      </div>
    </div>
  );
}

```

```
}  
  
export default headerlist;
```

## **product.js**

```
import './App.css';  
function ProductList({ product, addToCart }) {  
  return (  
    <div className="flex">  
      {product.map((item, index) => (  
        <div key={index} className="product-item" style={{ width: '33%' }}>  
          <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>  
  );  
}
```

```
export default ProductList;
```

## **cart.cs**

```
import { useEffect, useState } from 'react';  
import './App.css'; // Removed the space before the path  
function CartList({ cart }) {
```

```

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

useEffect(() => {
  setCART(cart);
}, [cart]);

const updateQuantity = (index, delta) => {
  const updatedCart = CART.map((item, i) =>
    i === index
    ? { ...item, quantity: Math.max(0, item.quantity + delta) }
    : item
  );
  setCART(updatedCart);
};

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

return (
  <div>
    {CART.map((item, index) => (
      <div key={index} style={{ marginBottom: '10px' }}>
        <img src={item.url} alt={item.name} width={40} /> { /* Fixed typo "imgsrc" */ }
        <span style={{ margin: '0 10px' }}>{item.name}</span>
        <button onClick={() => updateQuantity(index, -1)}>-</button>
        <span style={{ margin: '0 10px' }}>{item.quantity}</span>
        <button onClick={() => updateQuantity(index, 1)}>+</button>
        <span style={{ marginLeft: '10px' }}>

```

```
        Rs. {item.price * item.quantity}
      </span>
    </div>
  )}
  <p>Total: Rs. {total}</p>
</div>
);
}
```

```
export default CartList;
```

### **app.cs**

```
display: flex;flex{
flex-wrap: wrap;
}
```

```
.shopping-card {
display: flex;
justify-content: space-between;
background-color: #61dafb;
padding: 20px 30px;
}
```

```
.product-item {
padding: 20px;
}
```

```
Export default app.css;
```





LOREM Watch Black | Watches

Watch Ltd Siyana

Rs. 2599/-

## Hello World( TypeScript ):-

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

```
console.log(greeting);
```

### Output:-

```
TS hello.ts  X
TS hello.ts > ...
1  const greeting: string = "Hello, World!";
2  console.log(greeting);

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER

● PS C:\Users\2006r\OneDrive\Desktop\type script> tsc hello.ts
● PS C:\Users\2006r\OneDrive\Desktop\type script> node hello.js
❖ Hello, World!
○ PS C:\Users\2006r\OneDrive\Desktop\type script> █
```

## 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;

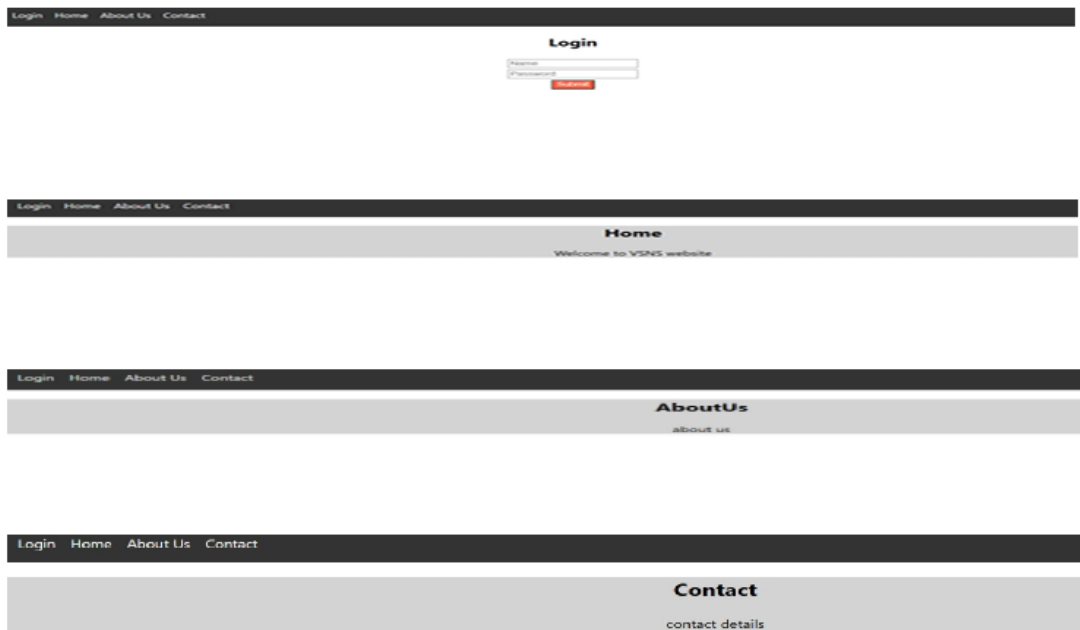
```

```

}
.App-nav a:hover { text-decoration: underline;
}

```

## Output:-



## >> 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);

```

```
};

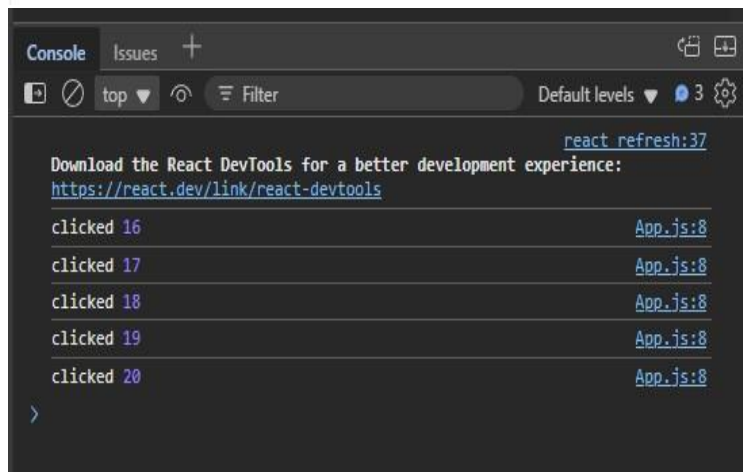
return (
  <div>
    <h1 id="count-display">Count: {count}</h1>
    <button onClick={addValue}>Increase Count</button>
  </div>
);
}

export default App;
```

## Output:-

**Count: 19**

Increase Count



## >> 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 connect My SQL Connector and Eclipse

### >>>Steps to Connect JDBC and 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

```
mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age  |
+-----+-----+-----+
| nanditha | 13    | 18   |
| anu      | 2     | 19   |
| bhoomi   | 3     | 17   |
| appi     | 4     | 16   |
| paddu    | 5     | 15   |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

## 6. Database connectivity ( Eclipse ):-

```
package nagalakshmi;

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/nandu","root","nagud");
        Statement stat = con.createStatement();
        String S1 = "delete from student1 where name='paddu'";
        String S2 = "update anu SET name='nagu' where rollno='13'";
        String S3 = "insert into anu values('nandu',20,20)";

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

**Output: -**

## \*insert

```
mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age   |
+-----+-----+-----+
| nagu   | 13     | 18    |
| anu    | 2      | 19    |
| bhoomi | 3      | 17    |
| appi   | 4      | 16    |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age   |
+-----+-----+-----+
| nagu   | 13     | 18    |
| anu    | 2      | 19    |
| bhoomi | 3      | 17    |
| appi   | 4      | 16    |
| nandu  | 20     | 20    |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

## \*Update

```
mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age   |
+-----+-----+-----+
| nanditha | 13     | 18    |
| anu    | 2      | 19    |
| bhoomi | 3      | 17    |
| appi   | 4      | 16    |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age   |
+-----+-----+-----+
| nagu   | 13     | 18    |
| anu    | 2      | 19    |
| bhoomi | 3      | 17    |
| appi   | 4      | 16    |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

## **\*delete**

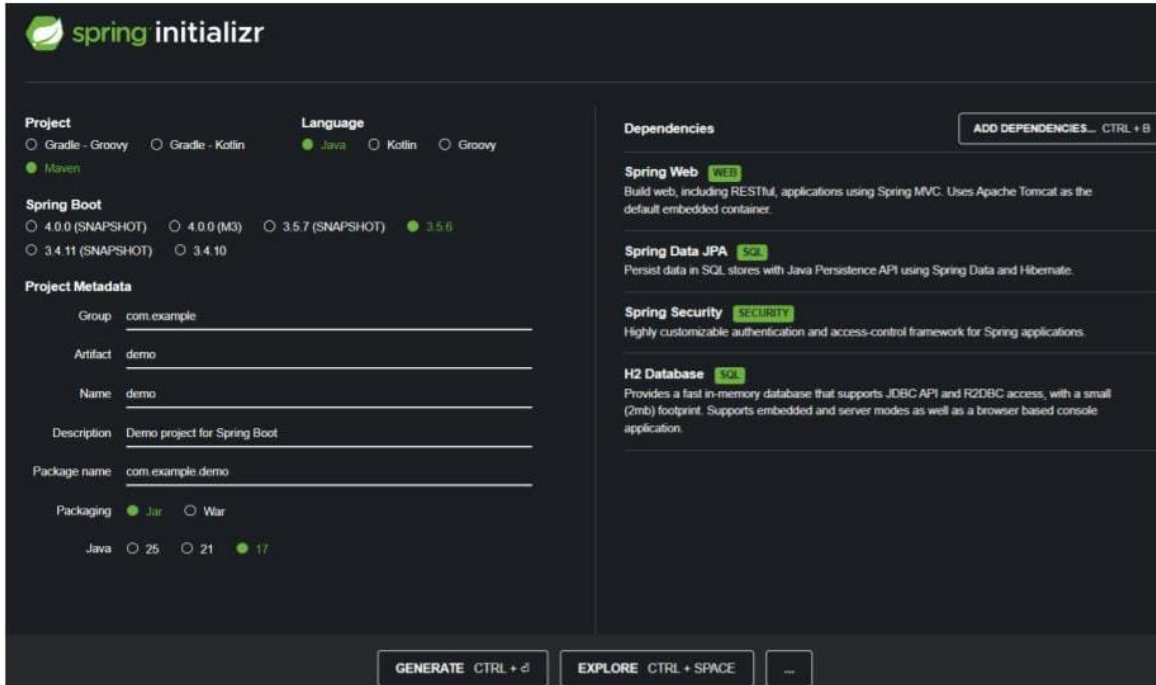
```
mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age  |
+-----+-----+-----+
| nanditha | 13    | 18   |
| anu     | 2     | 19   |
| bhoomi  | 3     | 17   |
| appi    | 4     | 16   |
| paddu   | 5     | 15   |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select *from student1;
+-----+-----+-----+
| name   | rollno | age  |
+-----+-----+-----+
| nanditha | 13    | 18   |
| anu     | 2     | 19   |
| bhoomi  | 3     | 17   |
| appi    | 4     | 16   |
+-----+-----+-----+
4 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. **Configure 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 :-



The image shows the Spring Initializr web application interface. It is a dark-themed form for generating a Spring Boot project. The 'Project' section has radio buttons for 'Gradle - Groovy' (selected), 'Gradle - Kotlin', and 'Maven'. The 'Language' section has radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has radio buttons for versions: '4.0.0 (SNAPSHOT)', '4.0.0 (M3)', '3.5.7 (SNAPSHOT)', '3.5.6' (selected), '3.4.11 (SNAPSHOT)', and '3.4.10'. The 'Project Metadata' section includes text input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo). There are also radio buttons for 'Packaging' ('Jar' selected, 'War' unselected) and 'Java' version ('25', '21', '17' selected). On the right, the 'Dependencies' section lists 'Spring Web' (WEB), 'Spring Data JPA' (SQL), 'Spring Security' (SECURITY), and 'H2 Database' (SQL). At the bottom, there are buttons for 'GENERATE' (CTRL + G), 'EXPLORE' (CTRL + SPACE), and a menu icon.

### >>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:-**

Please sign in

nagalakshm

...

Sign in



## 7 b. Create Hello World Program using Spring :-

### >>>HelloWorldContoller.java :-

```
package com.example.demo; 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.example.demo;

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 :-**

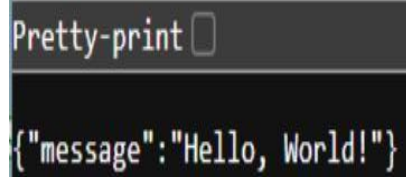
```
package com.example.demo; import
org.springframework.boot.SpringApplication; import
org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication public
class DemApplication {

    public static void main(String[] args) {
```

```
        SpringApplication.run(DemApplication.class, args);  
    }  
  
}
```

### **Output :-**



```
Pretty-print   
{ "message": "Hello, World!" }
```

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

>> Constructor Injection Code ( Eclipse ):-

>>>Engine.java

```
Package com.example; public
class Engine {
public void start() {
    System.out.println("Engine started!");
}
}
```

>>>Car.java :

```
package com.example; public
class Car { private 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; import  
  
org.springframework.context.ApplicationContext;  
  
Import  
org.springframework.context.support.ClassPathXmlApplicationContext;  
public class Main { public static void main(String[] args) {  
    // Load the constructor injection configuration  
  
    ApplicationContext context = new  
ClassPathXmlApplicationContext("spring-constructor-injection.xml");  
  
    Car car = (Car) context.getBean("car"); car.drive();  
  
    }  
}
```

## >>>Spring-constructor-injection.xml :-

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

```
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans.xsd">
```

```
<!-- Define the Engine bean -->
```

```
<bean id="engine" class="com.example.Engine" />
```

```
<!-- Define the Car bean using setter injection -->
```

```
<bean id="car" class="com.example.Car">
```

```
    <constructor-arg ref="engine" />
```

```
</bean>
```

```
</beans>
```

### Output: -



## >>Setter Injection code ( Eclipse ):-

### >>>Engine.java:

```
package com.example; public
class Engine {
public void start() {
        System.out.println("Engine started!");
    }
}
```

### >>>Car.java :

```
package com.example; public class Car { private
Engine engine;
// Setter Injection
Public void setEngine(Engine engine)
this.engine = engine;
    }
    public void drive() {
        engine.start();
        System.out.println("Car is driving!");
    }
}
```

### >>>Main.java :-

```
package com.example; import
org.springframework.context.ApplicationContext; import
org.springframework.context.support.ClassPathXmlApplicationContext
; public class Main { public static void main(String[] args) {
    .-// Load the setter injection configuration
    ApplicationContext context = new
ClassPathXmlApplicationContext("spring-setter-injection.xml");
    Car car = (Car) context.getBean("car"); car.drive();
}
}
```

### >>>Spring-setter-injection.xml :-

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

    <!-- Define the Engine bean -->

    <bean id="engine" class="com.example.Engine" />
```



```
<!-- Define the Car bean using setter injection -->
```

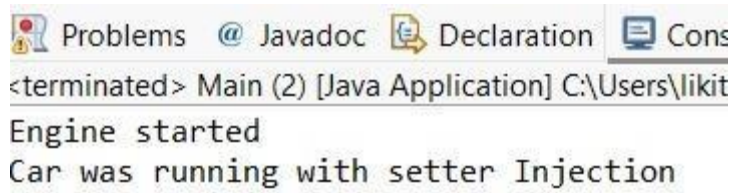
```
<bean id="car" class="com.example.Car">
```

```
    <constructor-arg ref="engine" />
```

```
</bean>
```

```
</beans>
```

## Output:-



```
<terminated> Main (2) [Java Application] C:\Users\likit  
Engine started  
Car was running with setter Injection
```

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

>>>index.js:-

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

const app = express();

app.use(express.json());

mongoose.connect('mongodb://localhost:27017/mongodb'
, useUrlParser: 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. POST – Send data to server (create)

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/users
- Body:** raw (JSON)
- Request Body:**

```
1 {
2   "name": "nagalakshmi",
3   "email": "nagalakshmi@gmail.com",
4   "age": "18",
5   "gender": "female"
6 }
```
- Response:** 201 Created (11.59 s, 363 B)
- Response Body:**

```
1 {
2   "name": "nagalakshmi",
3   "email": "nagalakshmi@gmail.com",
4   "age": 18,
5   "gender": "female",
6   "_id": "68d26519f6cdeb5720c9f816",
7   "__v": 0
8 }
```

## 2. GET – Retrieve data from server

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method **GET**, URL `http://localhost:3000/users/68d25519f6cdeb5720c9f816`, and a **Send** button.
- Tabs:** Params, Authorization, Headers (6), **Body** (selected), Scripts, Tests, Settings, and Cookies.
- Body Tab:** Shows the request body with radio buttons for `none`, `form-data`, `x-www-form-urlencoded`, **`raw`** (selected), `binary`, and `GraphQL`. A dropdown menu is set to **JSON** with a **Beautify** link.
- Response Area:** A large text area containing the response body, which is currently empty except for a placeholder text: `1 Ctrl+Alt+P to Ask AI`.
- Status Bar:** Displays **200 OK**, `486 ms`, `358 B`, and a **Save Response** button.
- Response Viewer:** A tabbed interface with **JSON** (selected), **Preview**, and **Visualize**. It shows the following JSON response:

```
1 {
2   "_id": "68d25519f6cdeb5720c9f816",
3   "name": "nagalakshmi",
4   "email": "nagalakshmi@gmail.com",
5   "age": 18,
6   "gender": "female",
7   "__v": 0
8 }
```

### 3 PUT – Update data

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/users/68d25519f6cdeb5720c9f816`. The request body is a JSON object: `{ "_id": "68d25519f6cdeb5720c9f816", "name": "nagalakshmikg", "email": "nagalakshmikg@gmail.com", "age": 18, "gender": "female", "__v": 0 }`. The response status is `200 OK` with a response time of 557 ms and a body size of 362 B. The response body is also a JSON object: `{ "_id": "68d25519f6cdeb5720c9f816", "name": "nagalakshmikg", "email": "nagalakshmikg@gmail.com", "age": 18, "gender": "female", "__v": 0 }`. The interface includes tabs for Params, Authorization, Headers (8), Body, Scripts, Tests, and Settings. The Body tab is active, showing the JSON body. The response status bar indicates `200 OK` and includes a 'Save Response' button. The response body is displayed in a JSON viewer with options for Preview, Visualize, and Debug with AI.

PUT `http://localhost:3000/users/68d25519f6cdeb5720c9f816` Send

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Schema Beautify

```
1 {
2   "_id": "68d25519f6cdeb5720c9f816",
3   "name": "nagalakshmikg",
4   "email": "nagalakshmikg@gmail.com",
5   "age": 18,
6   "gender": "female",
7   "__v": 0
8 }
```

Body Cookies Headers (7) Test Results 200 OK • 557 ms • 362 B Save Response

{ JSON Preview Visualize

```
1 {
2   "_id": "68d25519f6cdeb5720c9f816",
3   "name": "nagalakshmikg",
4   "email": "nagalakshmikg@gmail.com",
5   "age": 18,
6   "gender": "female",
7   "__v": 0
8 }
```

Activate Windows

### 4.DEL – Remove data

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/users/68d25519f6cdeb5720c9f816`. The response status is `404 Not Found` with a response time of 17 ms and a body size of 248 B. The response body is `User not found`. The interface includes tabs for Params, Authorization, Headers (6), Body, Scripts, Tests, and Settings. The Body tab is active, showing the response body. The response status bar indicates `404 Not Found` and includes a 'Save Response' button. The response body is displayed in a JSON viewer with options for Preview, Debug with AI, and HTML.

GET `http://localhost:3000/users/68d25519f6cdeb5720c9f816` Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Schema Beautify

```
1 Ctrl+Alt+P to Ask AI
```

Body Cookies Headers (7) Test Results 404 Not Found • 17 ms • 248 B Save Response

HTML Preview Debug with AI

```
1 User not found
```

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

A. Insert any five documents into above collection.

```
> use college
< switched to db college
> db.student.insertMany([
  { name: "appi", rollno: 12, branch: "ME" },
  { name: "nagu", rollno: 5, branch: "EC" },
  { name: "bhoomi", rollno: 1, branch: "CS" }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68d0f85de15b6e7f21ddc3d7'),
    '1': ObjectId('68d0f85de15b6e7f21ddc3d8'),
    '2': ObjectId('68d0f85de15b6e7f21ddc3d9')
  }
}
```



B. Display all the documents present in collection student.

```
> db.student.find()
< {
  _id: ObjectId('68d0d1824f80160a3ccb3509'),
  name: 'nanditha',
  rollno: 5,
  branch: 'cs'
}
{
  _id: ObjectId('68d0d1824f80160a3ccb350a'),
  name: 'anu',
  rollno: 2,
  branch: 'is'
}
{
  _id: ObjectId('68d0d1824f80160a3ccb350b'),
  name: 'bhoomi',
  rollno: 3,
  branch: 'me'
}
```

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

```
> db.student.find({}, {branch:1, _id:0})
< {
  branch: 'cs'
}
{
  branch: 'is'
}
{
  branch: 'me'
}
```

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

```
> db.student.updateOne({branch:"CS"},{$set:{branch:"EC"}})
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.student.find()
< {
  _id: ObjectId('68d0d1824f80160a3ccb3509'),
  name: 'nanditha',
  rollno: 5,
  branch: 'cs'
}
{
  _id: ObjectId('68d0d1824f80160a3ccb350a'),
  name: 'anu',
  rollno: 2,
  branch: 'is'
}
{
  _id: ObjectId('68d0d1824f80160a3ccb350b'),
  name: 'bhoomi',
  rollno: 3,
  branch: 'me'
}
```

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

```
> db.student.deleteMany({branch:"EC"})  
< {  
  acknowledged: true,  
  deletedCount: 7  
}
```

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

### \*Dockerfile ( 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 .

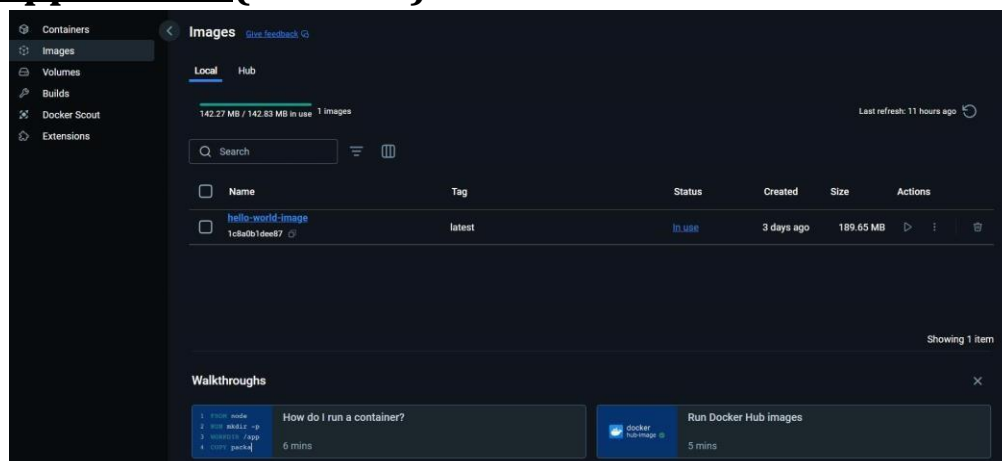
Step 2 : docker images.

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 :-

