**Practical no. 1**

**Aim:** Write a program to implement MongoDB data models.

**Date:**12/12/2023 **Roll no.:** 03 **sign:**

In MongoDB, data has a flexible schema. It is totally different from SQL database where you had to determine and declare a table's schema before inserting data. MongoDB collections do not enforce document structure.

The main challenge in data modeling is balancing the need of the application, the performance characteristics of the database engine, and the data retrieval patterns.

**What is data modeling?**

**Data modeling** is the process of creating a clean data model of how you will store data in a database. Data models also describe how the data is related. The goal of data modeling is to identify all the data components of a system, how they are connected, and what are the best ways to represent these relationships.

Data modeling is done at the application level. Data models consist of the following components:

* **Entity**—an independent object that is also a logical component in the system. Entities can be categorized into tangible and intangible. Tangible entities, such as books, exist in the real world.
* **Entity types**—the categories used to group entities. For example, the book entity with the title “Alice in Wonderland” belongs to the entity type “book.”
* **Attributes**—the characteristics of an entity. For example, the entity “book” has the attributes [ISBN](https://en.wikipedia.org/wiki/ISBN) (String) and title (String).
* **Relationships**—define the connections between the entities. For example, one user can borrow many books at a time. The relationship between the entities "users" and "books" is one to many.

**What are the advantages of data modelling**

* Ensures better database planning, design, and implementation, leading to improved application performance.
* Promotes faster application development through easier object mapping.

**Mongooese:**

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node. js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB

**How to import mongoose in NodeJs?**

Any package can be imported in a NodeJs file by using the require keyword and passing the package name in single quotes. var mongoose = require("mongoose"); This is how mongoose is imported in a NodeJs file.

**Syntax**

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

**Program:**

**Index.js**

const mongoose = require("mongoose")

mongoose.set('strictQuery',true);

mongoose.connect("mongodb://127.0.0.1:27017",console.log('Connected to database'));

const studentSchema = new mongoose.Schema({

name:String,rollNo:Number,class:String,age:Number,email:String

})

const student = mongoose.model('Student',studentSchema);

student.createCollection().then(function(){

console.log("Collection is created");

})

**Model.js**

const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

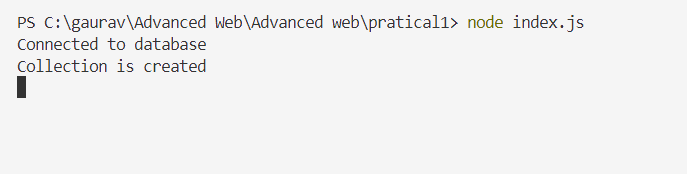
age:Number,

email:String

},{collection:"students"});

module.exports = mongoose.model("student",studentSchema)

**Output:**

****

**Practical no. 2**

**Aim:** Write a program to implement CRUD operations on MongoDB.

**Date:**19/12/2023 **Roll no.:** 03 **sign:**

**What is CRUD in MongoDB?**

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database. CRUD is data-oriented, and it’s standardized according to HTTP action verbs.

**When it comes to the individual CRUD operations:**

* The Create operation is used to insert new documents in the MongoDB database.
* The Read operation is used to query a document in the database.
* The Update operation is used to modify existing documents in the database.
* The Delete operation is used to remove documents in the database.

**Create Operations**

For MongoDB CRUD, if the specified collection doesn’t exist, the [create](https://docs.mongodb.com/manual/tutorial/insert-documents/) operation will create the collection when it’s executed

MongoDB provides two different create operations that you can use to insert documents into a collection:

* [db.collection.insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/#db.collection.insertOne)
* [db.collection.insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/#db.collection.insertMany)

**Read Operations**

The [read](https://docs.mongodb.com/manual/tutorial/query-documents/) operations allow you to supply special query filters and criteria that let you specify which documents you want.

MongoDB has two methods of reading documents from a collection:

* [db.collection.find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find)
* [db.collection.findOne()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find)

**Update Operations**

Like create operations, [update](https://docs.mongodb.com/manual/tutorial/update-documents/) operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

For MongoDB CRUD, there are three different methods of updating documents:

* [db.collection.updateOne()](https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne)
* [db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany)
* [db.collection.replaceOne()](https://docs.mongodb.com/manual/reference/method/db.collection.replaceOne/#db.collection.replaceOne)

**Delete Operations**

[Delete](https://docs.mongodb.com/manual/tutorial/remove-documents/) operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.

MongoDB has two different methods of deleting records from a collection:

* [db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/#db.collection.deleteOne)
* [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/#db.collection.deleteMany)

**Program:**

**CreateCollection.js**

const mongoose = require('mongoose');

mongoose.set('strictQuery',true);

mongoose.connect('mongodb://127.0.0.1:27017',{

dbName:"StudentDb"},console.log("Connected to database"))

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

)

const Student = mongoose.model("Student",studentSchema);

var db = mongoose.connection;

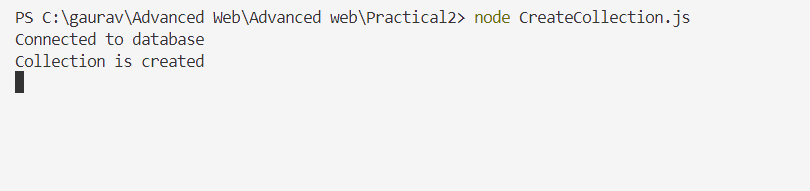
Student.createCollection().then(function(){

console.log('Collection is created');

});

db.on('error',console.error.bind(console,'connection error'));

**Output:**

****

**insertOne.js**

const mongoose = require('mongoose');

mongoose.set('strictQuery',true);

mongoose.connect('mongodb://127.0.0.1:27017',{

dbName:"StudentDb"},console.log("Connected to database"))

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

)

const Student = mongoose.model("Student",studentSchema);

var db = mongoose.connection;

var student1 = new Student({name:"Gaurav",

rollNo:3,

class:"SyCs",

age:19,

email:"gauravchavan101004@gmail.com"});

student1.save(function(err,result){

if(err){

console.log(err);

}else{

console.log(result);

}

})

console.log('insert');

**Output:**

**insertMany.js**

const mongoose = require('mongoose');

mongoose.set('strictQuery',true);

mongoose.connect('mongodb://127.0.0.1:27017',{

dbName:"StudentDb"},console.log("Connected to database"))

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

)

var db = mongoose.connection;

const Student = mongoose.model("Student",studentSchema);

db.once('open',function(){

Student.insertMany([

{name:"Sahil Mestry",

rollNo:18,

class:"SYCS",

age:18,

email:"sahilmestry2006@gmail.com"

},

{name:"Pratik Vardekar",

rollNo:37,

class:"SYCS",

age:19,

email:"pratikVardekar2004@gmail.com"

},

{name:"Jayesh Pednekar",

rollNo:27,

class:"SYCS",

age:19,

email:"jayeshPednekar@gmail.com"

}]).then(function(){

console.log('data inserted');

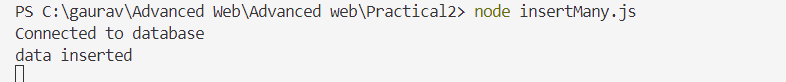
}).catch(function(err){

console.log(err);

})

})

**Output:**

****

**getData.js**

const mongoose = require('mongoose');

mongoose.set('strictQuery',true);

mongoose.connect('mongodb://127.0.0.1:27017',{

dbName:"StudentDb"

},console.log("Connected to database"))

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

)

const Student = mongoose.model("Student",studentSchema);

var db = mongoose.connection;

Student.find({}).then(data=>{

console.log('Data : ');

console.log(data);

}).catch(error=>{

console.log(error);

})

**Output:**

****

**deleteOne.js**

const mongoose = require('mongoose');

mongoose.set('strictQuery',true);

mongoose.connect('mongodb://127.0.0.1:27017',{

dbName:"StudentDb"},console.log("Connected to database"))

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

)

const Student = mongoose.model("Student",studentSchema);

var db = mongoose.connection;

Student.deleteOne({rollNo:3}).then(function(){

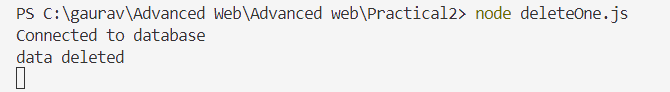
console.log('data deleted');

}).catch(function(error){

console.log(error);

})

**Output:**

****

**updateOne.js**

const mongoose = require('mongoose');

mongoose.set('strictQuery',true);

mongoose.connect('mongodb://127.0.0.1:27017',{

dbName:"test",useUnifiedTopology:true,useNewUrlParser:true

},err=>err? console.log(err):console.log("Connected to database"))

const studentSchema = new mongoose.Schema(

{

name:String,

rollNo:Number,

class:String,

age:Number,

email:String

}

)

const Student = mongoose.model("Student",studentSchema);

var db = mongoose.connection;

var student1 = new Student({name:"Gaurav",

rollNo:3,

class:"SyCs",

age:19,

email:"gauravchavan101004@gmail.com"});

Student.updateOne({name:"Gaurav"},{name:"om"},function(err,result){

if(err){

console.log(err);

}else{

console.log(result);

}

}

);

**Output:**

**Practical no. 3**

**Aim:** Write a program to perform validation of a form using AngularJS

**Date:**09/01/2024 **Roll no.:** 03 **sign:**

AngularJS performs form validation on the client side. AngularJS monitors the state of the form and input fields (input, text-area, select), and notify the user about the current state. AngularJS also holds information about whether the input fields have been touched, modified, or not. Form input fields have the following states:

$untouched: It shows that field has not been touched yet.

$touched: It shows that field has been touched.

$pristine: It represents that the field has not been modified yet.

$dirty: It illustrates that the field has been modified.

$invalid: It specifies that the field content is not valid.

$valid: It specifies that the field content is valid.

These all are the properties of the input field which can be either true or false. Forms have the following states:

$pristine: It represents that the fields have not been modified yet.

$dirty: It illustrates that one or more fields have been modified.

$invalid: It specifies that the form content is not valid.

$valid: It specifies that the form content is valid.

$submitted: It specifies that the form is submitted.

These all are the properties of the form which can be either true or false. These states can be used to show meaningful messages to the user.

**Program:**

**Index.html**

<!DOCTYPE html>

<html>

<head>

<title>AngularJs Form Input Fields Validation Example</title>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<script>

var app = angular.module('formApp', []);

app.controller('formCtrl', function ($scope) {

$scope.sendForm = function () {

$scope.msg='Form Submited Successfully';

};

$scope.getClass = function (color) {

return color.toString();

}

});

</script>

<style>

.valid.false {

background: red;

}

.valid.true {

background: green;

}

.error {

color: red;

}

</style>

</head>

<body ng-app="formApp" ng-controller="formCtrl">

<h3>Form validation using AngularJs</h3>

<form name="personForm" ng-submit="sendForm()">

<label for="name">Name</label>

<input id="name" name="name" type="text" ng-model="person.name" required />

<span class="error" ng-show="personForm.name.$error.required"> Required! </span>

<br /><br />

<label for="adress">Adress</label>

<input id="address" name="address" type="text" ng-model="person.address" required />

<span class="error" ng-show="personForm.address.$error.required"> Required! </span>

<br /><br />

<label for="contact">Contact No</label>

<input id="mobile" name="mobile" type="number" ng-model="person.mobile" required />

<span class="error" ng-show="personForm.mobile.$error.required">Required number!</span>

<span class="error" ng-show="personForm.mobile.$error.mobile">Invalid mobile!</span>

<br /><br />

<label for="email">Email</label>

<input id="email" name="email" type="email" ng-model="person.email" required />

<span class="error" ng-show="personForm.email.$error.required">Required!</span>

<span class="error" ng-show="personForm.email.$error.email">Invalid Email!</span>

<br /><br />

<input type="checkbox" ng-model="terms" name="terms" id="terms" required />

<label for="terms">I Agree to the terms.</label>

<span class="error" ng-show="personForm.terms.$error.required">You must agree to the terms</span>

<br /><br />

<button type="submit">Submit Form</button>

<br /><br />

<span>{{msg}}</span>

</form>

</body>

</html>

**welcome.html**

<!DOCTYPE html>

<html>

<head>

<title>Welcome Page</title>

</head>

<body style="background-color: yellow;">

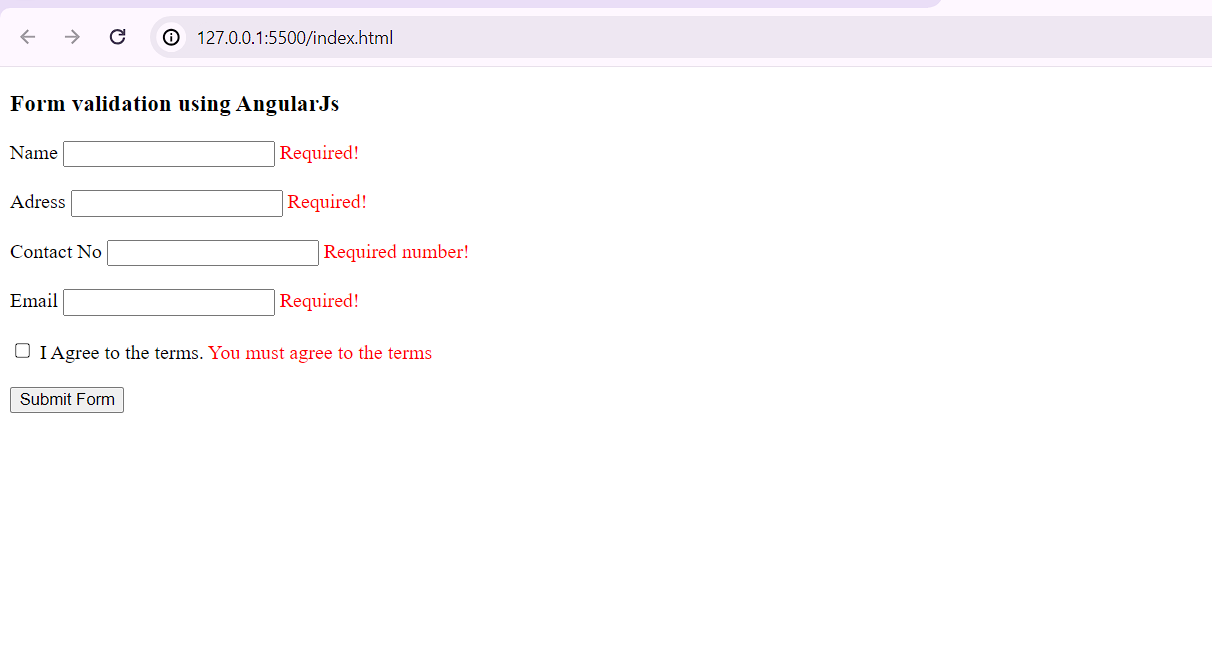
<h1>Record Successfully Submitted........</h1>

</body>

</html>

**Output:**

Before:



After:



**Practical no. 4**

**Aim:** Write a program to create and implement modules and controllers in Angular JS.

**Date:**16/01/2024 **Roll no.:** 03 **sign:**

­­­The AngularJS module defines the functionality of the application which is applied on the entire HTML page. It helps to link many components. So it is just a group of related components. It is a container that consists of different parts like controllers, services, and directives.

Note: These modules should be made in normal HTML files like index.html and no need to create a new project in VisualStudio for this section.

**Creating a Module in AngularJS:**

**Syntax :**

var app = angular.module("Module-name", []);

In this [], we can add a list of components needed but we are not including any components in this case. This created module is bound with any tag like div, body, etc by adding it to the list of modules.

**Example :**

Module-name: DemoApp.js:

var app = angular.module('DemoApp', []);

**Program:**

**module.html**

<!DOCTYPE html>

<html ng-app="app">

<head>

<meta charset="utf 8">

<title>Angular First App</title>

</head>

<body>

<h1 ng-controller="HelloWorldCtrl">Name:{{Name}}</h1>

<h1 ng-controller="HelloWorldCtrl">Website:{{Website}}</h1>

<script src="https://code.angularjs.org/1.6.9/angular.js"></script>

<script>

angular.module("app", []).controller("HelloWorldCtrl", function($scope) {

$scope.Name = " Kankavli College Kankavli ";

$scope.Website = "www.kckcollege.com.com";

} )

</script>

</body>

</html>

**Output:**

****

**Practical no. 5**

**Aim:** Write a program to implement Error Handling in Angular JS.

**Date:**23/01/2024 **Roll no.:** 03 **sign:**

**Error handling in Angular JS**

Error handling in AngularJS involves managing errors that occur during the execution of your AngularJS application and providing appropriate responses or feedback to the user. Here's a basic overview of error handling techniques in AngularJS:

* **Using try-catch blocks:** You can use traditional JavaScript try-catch blocks to handle errors within your AngularJS code. However, this approach is generally not recommended for handling Angular-specific errors or asynchronous operations
* **Using AngularJS $exceptionHandler service**: AngularJS provides a built-in service called **$exceptionHandler** that allows you to handle uncaught exceptions in your AngularJS application globally.
* **Error handling in promises**: If you're using promises for asynchronous operations (e.g., with **$http** service), you can handle errors using **.catch()** method of the promise object.
* **Using AngularJS Interceptors**: AngularJS interceptors (**$httpProvider.interceptors**) can be used to globally intercept and handle HTTP requests and responses, including errors.
* **Custom error handling**: You can define custom error handling mechanisms throughout your application as per your specific requirements. For instance, you might show error messages to users using AngularJS's data binding and templating system.

**Index.html**

<!DOCTYPE html>

<html>

<head>

<title>Exception Handling in AngularJS</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js"></script>

<script src="app.js"></script>

</head>

<body ng-app="myApp">

<h1>Error Handling</h1>

<div ng-controller="exceptionController">

<p>Enter a number: </p>

<input type="number" ng-model="num" />

<button ng-click="calculate()">Calculate</button>

<p>Sq. Root. Result: {{result}}</p>

<p>Any Exception: {{errorMessage}}</p>

</div>

</body>

</html>

**app.js**

angular.module('myApp', [])

.controller('exceptionController', function ($scope, $exceptionHandler) {

$scope.calculate = function () {

try {

// Ensure the input is treated as a number (supports integers and decimals)

var inputNumber = Number($scope.num);

// Check if the conversion results in a valid number

if (isNaN(inputNumber)) {

throw new Error("Invalid input. Please enter a valid number.");

}

// Check for decimal numbers

if (!Number.isInteger (inputNumber)) {

$scope.result = "Not Possible";

throw new Error("Cannot input decimal number.");

}

// Check for non-negative numbers

if (inputNumber < 0) { $scope.result = "Not Possible";

throw new Error("Number must be positive.");

}

// Calculate the square root of the input number

$scope.result = Math.sqrt(inputNumber);

$scope.errorMessage = "No Exceptions";

} catch (e) {

// Handle any exceptions that occur during calculation

$scope.errorMessage = e.message;

// Use AngularJS's $exceptionHandler service to log or handle the error as needed

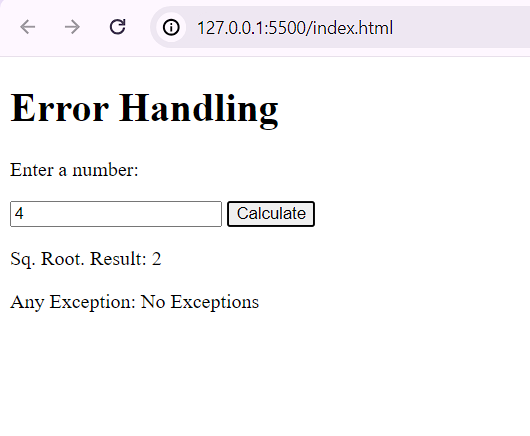
$exceptionHandler(e);

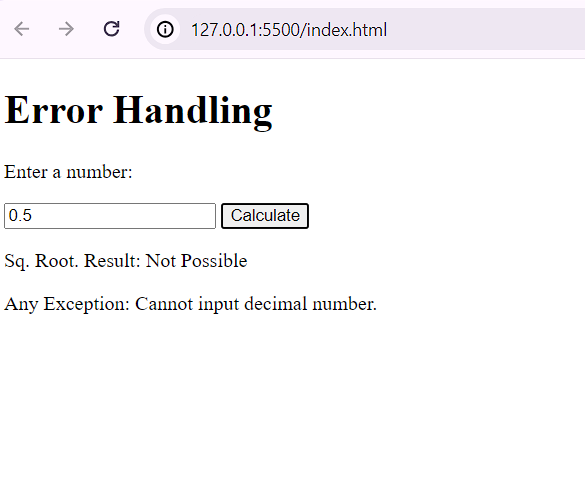
}

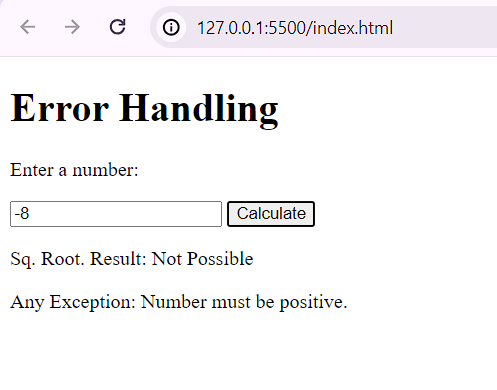
}

});

**Output:**

****

****

****

**Practical no. 6**

**Aim:** Create an application for Customer / Students records using AngularJS

**Date:**30/01/2024 **Roll no.:** 03 **sign:**

**AngularJS:**

AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions.

An AngularJS application consists of following three important parts −

* **ng-app** − This directive defines and links an AngularJS application to HTML.
* **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
* **ng-bind** − This directive binds the AngularJS Application data to HTML tags.

## How AngularJS Integrates with HTML

* The ng-app directive indicates the start of AngularJS application.
* The ng-model directive creates a model variable named name, which can be used with the HTML page and within the div having ng-app directive.
* The ng-bind then uses the name model to be displayed in the HTML <span> tag whenever user enters input in the text box.
* Closing</div> tag indicates the end of AngularJS application.

# AngularJS – Directives:

AngularJS directives are used to extend HTML. They are special attributes starting with **ng**-prefix.

* **ng-app** − This directive starts an AngularJS Application.
* **ng-init** − This directive initializes application data.
* **ng-model** − This directive defines the model that is variable to be used in AngularJS.
* **ng-repeat** − This directive repeats HTML elements for each item in a collection.

# AngularJS – Controllers:

# AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using *ng-controller* directive. A controller is a JavaScript object that contains attributes/properties, and functions. Each controller accepts $scope as a parameter, which refers to the application/module that the controller needs to handle.

**Syntax :**

<div ng-app = "" ng-controller = "studentController">

...

</div>

**Program:**

**Index.html**

<IDOCTYPE html>

<html>

<head>

<title>Student Management Portal</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<script src="app.js"></script>

</head>

<body>

<h1>Studet Database Portal</h1>

<div ng-app="myApp" ng-controller="studentController">

<label>Name</label>

<input type="text" name="name" ng-model="newStudent.name" />

<label>Address</label>

<input type="text" name="address" ng-model="newStudent.address" />

<label>Dept.</label>

<input type="text" name="dept" ng-model="newStudent.dept" />

<input type="hidden" ng-model="newStudent.id" />

<input type="button" value="Save" ng-click="saveRecord()" class="btn btn-primary" />

<br><br>

<table border="1" bordercolor="blue">

<tr style="color: blue">

<th style="width:150px">Name</th>

<th style="width:150px">Address</th>

<th style="width:150px">Dept</th>

<th>Action</th>

</tr>

<tr style="color: black" ng-repeat="student in students">

<td>{{ student.name }}</td>

<td>{{ student.address }}</td>

<td>{{ student.dept }}</td>

<td>

<a href="#" ng-click="edit(student.\_id)">edit</a> |

<a href="#" ng-click="delete(student.id)">delete</a>

</td>

</tr>

</table>

</div>

</body>

</html>

**app.js**

var app = angular.module('myApp', []);

app.controller('studentController', function ($scope) {

$scope.students = [];

var empid = 1;

// Save or update student record

$scope.saveRecord = function () {

// Check if the student is new (id is null)

if ($scope.newStudent.id == null) {

// Assign a unique ID to the new student

$scope.newStudent.id = empid++;

// Add the new student to the students array

$scope.students.push($scope.newStudent);

} else {

// If the student already exists (editing an existing student), find and update the student's record

for (i in $scope.students) {

if ($scope.students[i].id == $scope.newStudent.id) {

$scope.students[i] = $scope.newStudent;

}

}

}

// Reset the newStudent object to clear the form fields

$scope.newStudent = {};

};

// Function to delete a student record based on the provided ID

$scope.delete = function (id) {

for (i in $scope.students) {

if ($scope.students[i].id == id) {

// Remove the student from the students array

$scope.students.splice(i, 1);

// Reset the newStudent object

$scope.newStudent = {};

}

}

};

// Function to edit a student record. It loads the student's information into the form for editing

$scope.edit = function (id) {

for (i in $scope.students) {

if ($scope.students[i].id == id) {

// Create a copy of the student object to edit, to avoid direct modification of the array's object

$scope.newStudent = angular.copy($scope.students[i]);

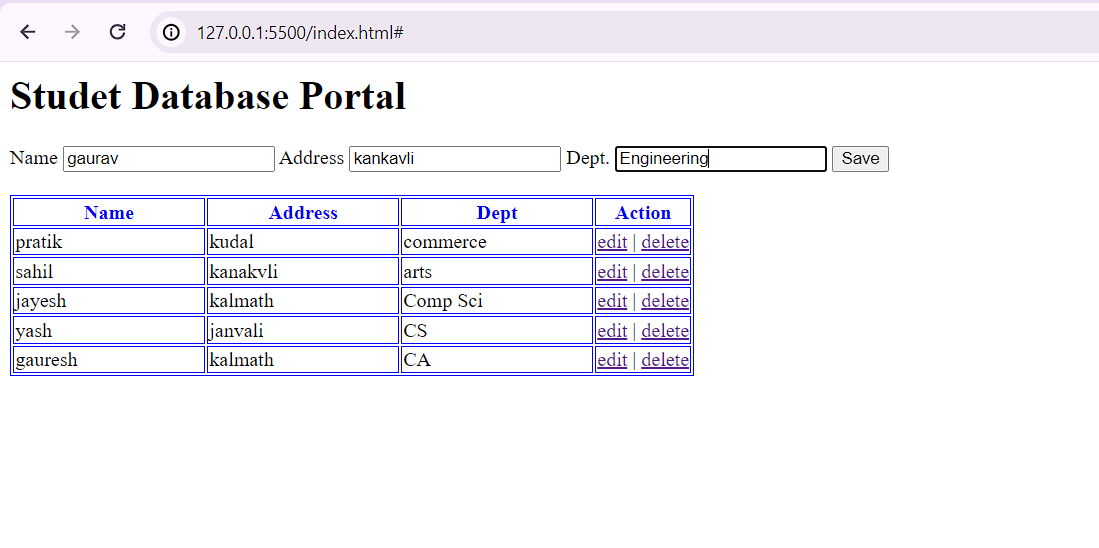
}

}

};

});

**Output:**



**Practical no. 7**

**Aim:** Write a program to create a simple web application using Express, Node JS and Angular JS

**Date:**06/02/2024 **Roll no.:** 03 **sign:**

**Node.js:**

* Node.js is a very powerful JavaScript-based platform built on Google Chrome's JavaScript V8 Engine.
* It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications.
* Node.js is open source, completely free, and used by thousands of developers around the world.

**Express.js:**

* Express is a popular web application framework for Node.js, a JavaScript runtime that allows developers to run JavaScript code on the server-side.
* Express is designed to make it easy to create and manage web applications and is often used to build the backend of web applications, or the server-side of a full-stack web application.

**Angular.js:**

* Angular is a popular JavaScript framework for building web applications, while Express is a minimal and flexible Node.js web application framework.
* Together, these two frameworks allow us to create powerful and dynamic web applications with a seamless development experience.

**Program:**

**Index.html**

<IDOCTYPE html>

<html>

<head>

<title>Student Management Portal</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

<script src="app.js"></script>

</head>

<body>

<h1>Studet Database Portal</h1>

<div ng-app="myApp" ng-controller="studentController">

<label>Name</label>

<input type="text" name="name" ng-model="newStudent.name" />

<label>Address</label>

<input type="text" name="address" ng-model="newStudent.address" />

<label>Dept.</label>

<input type="text" name="dept" ng-model="newStudent.dept" />

<input type="hidden" ng-model="newStudent.id" />

<input type="button" value="Save" ng-click="saveRecord()" class="btn btn-primary" />

<br><br>

<table border="1" bordercolor="blue">

<tr style="color: blue">

<th style="width:150px">Name</th>

<th style="width:150px">Address</th>

<th style="width:150px">Dept</th>

<th>Action</th>

</tr>

<tr style="color: black" ng-repeat="student in students">

<td>{{ student.name }}</td>

<td>{{ student.address }}</td>

<td>{{ student.dept }}</td>

<td>

<a href="#" ng-click="edit(student.\_id)">edit</a> |

<a href="#" ng-click="delete(student.id)">delete</a>

</td>

</tr>

</table>

</div>

</body>

</html>

**app.js**

var app = angular.module('myApp', []);

app.controller('studentController', function ($scope, $http) {

$scope.students = [];

var empid = 1;

// Load students from backend

$scope.loadStudents = function () {

$http.get('http://localhost:3000/students')

.then(function (response) {

$scope.students = response.data;

});

};

$scope.loadStudents(); // Initial load

// Save or update student record

$scope.saveRecord = function () {

if ($scope.newStudent.id == null) {

// Create new student

$http.post('http://localhost:3000/students', $scope.newStudent)

.then(function (response) {

$scope.students.push(response.data);

$scope.newStudent = {}; // Reset

});

} else {

// Update existing student

$http.put(`http://localhost:3000/students/${$scope.newStudent.id}`, $scope.newStudent)

.then(function (response) {

$scope.loadStudents(); // Reload the list

$scope.newStudent = {}; // Reset

});

}

};

// Delete student

$scope.delete = function(id) {

$http.delete('http://localhost:3000/students/' + id)

. then(function(response) {

// Handle success

$scope.loadStudents(); // Reload or remove the student from the list locally

}, function(error) {

console.error('Error deleting student:', error);

});

};

// Edit student

$scope.edit = function (id) {

// Find student in $scope.students

let student = $scope.students.find(student => student.\_id === id);

if (student) {

$scope.newStudent = angular.copy(student);

$scope.newStudent.id = student.\_id; // Use MongoDB's\_id

}

};

});

**server.js**

const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

const port = process.env.PORT || 3000;

// Enable Cross-Origin Resource Sharing (CORS) for all routes.

app.use(cors());

// Use the body-parser middleware to parse JSON-formatted request bodies

app.use(bodyParser.json());

app.use(express.static('public'));

mongoose.set('strictQuery', true);

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/studentDB', {

useNewUrlParser: true,

useUnifiedTopology: true

});

// Student model

const studentSchema = new mongoose.Schema({

name: String,

address: String,

dept: String

});

const Student = mongoose.model('Student', studentSchema);

app.listen(port, () => {

console.log('Server listening at http://localhost:%s', port);

});

// Routes

app.get('/students', async (req, res) => {

const students = await Student.find();

res.json(students);

});

app.post('/students', async (req, res) => {

const student = new Student(req.body);

await student.save();

res.status (201).json(student);

});

app.put('/students/:id', async (req, res) => {

const updatedStudent = await Student.findByIdAndUpdate (req.params.id, req.body, {new: true});

res.json(updatedStudent);

});

app.delete('/students/:id', async (req, res) => {

// Ensure `req.params.id is correctly obtained

try {

const result = await Student.findByIdAndDelete (req.params.id);

if (!result) {

return res.status(404).send({ message: 'Student not found' });

}

res.send({ message: 'Student deleted' });

} catch (error) {

res.status (500).send(error);

}

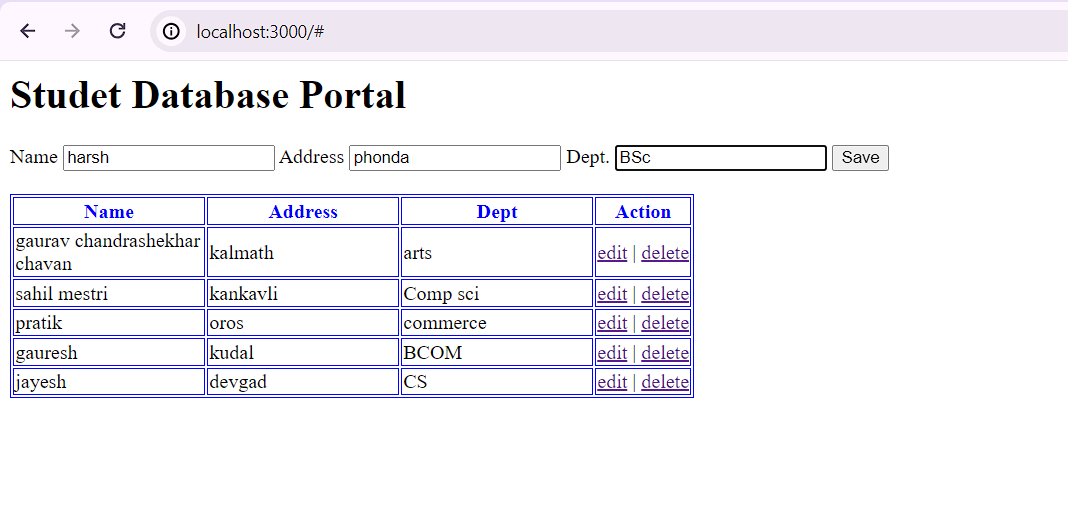
});

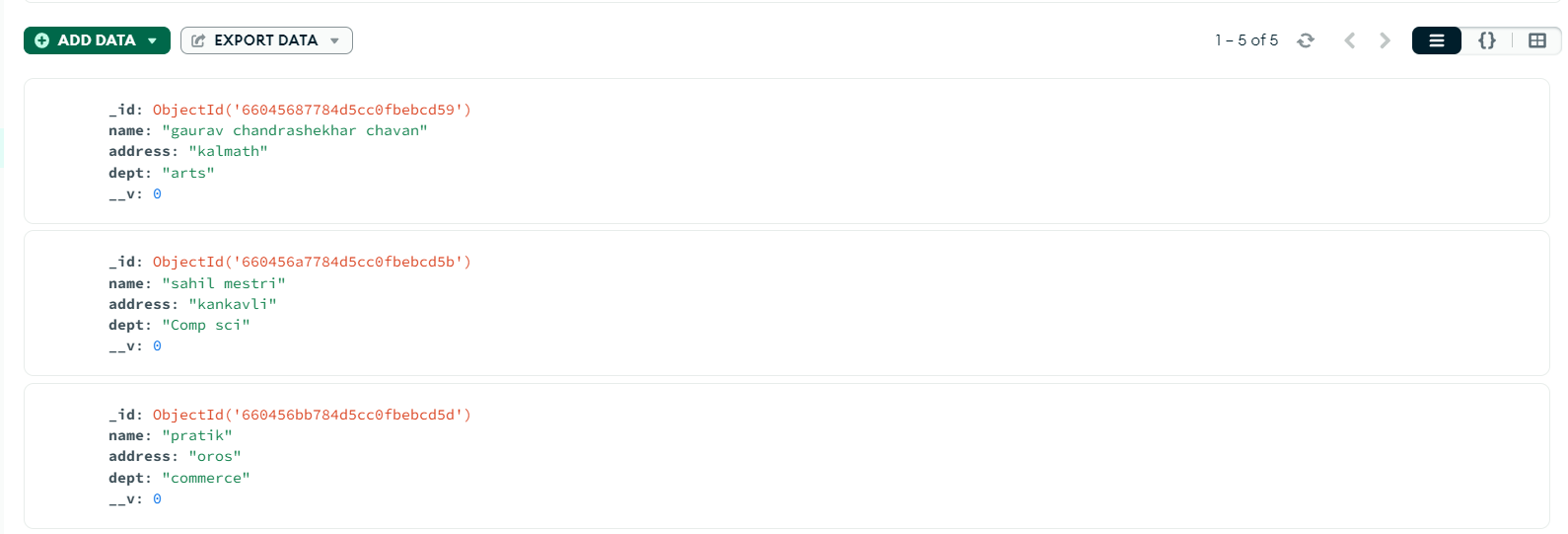
app.get('\*', (req, res) => {

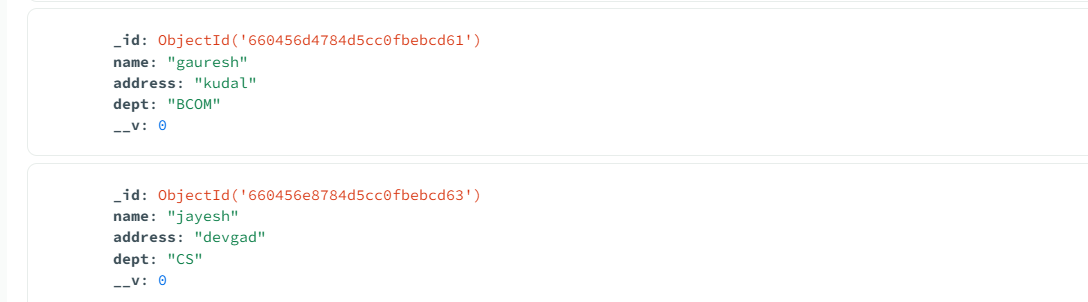
res.sendFile('index.html', { root: 'public' });

});

**Output:**

****

****



**Practical no. 8**

**Aim:** Create a simple HTML “Hello World” Project using AngularJS Framework and apply ng-controller, ng-model and expressions.

**Date:**13/02/2024 **Roll no.:** 03 **sign:**

AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

AngularJS extends HTML with **ng-directives**.

* The **ng-app** directive defines an AngularJS application.
* The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.
* The **ng-bind** directive binds application data to the HTML view.

**AngularJS Modules:**

An AngularJS module defines an application.

The module is a container for the different parts of an application.

The module is a container for the application controllers.

Controllers always belong to a module.

A module is created by using the AngularJS function angular.module.

**ng-controller:**

The ng-controller directive adds a controller to your application. In the controller you can write code, and make functions and variables, which will be parts of an object, available inside the current HTML element. In AngularJS this object is called a scope.

**ng-model:**

The**ng-model directive** is a directive that is used to bind the values of the HTML controls (input, select, and textarea) or any custom form controls, and stores the required user value in a variable and we can use that variable whenever we require that value. It also is used during form validations. The various form input types (text, checkbox, radio, number, email, URL, date, datetime-local time, month, week) can be used with the *ngModel directive*. This directive is supported by [<input>](https://www.geeksforgeeks.org/html-input-tag/), [<select>](https://www.geeksforgeeks.org/html-select-tag/) & [<textarea>](https://www.geeksforgeeks.org/html-textarea-tag/).

**Syntax:**

<element ng-model="">

Content...

</element>

**Program:**

<!DOCTYPE html>

<html>

<head>

<title>Hello World with AngularJS</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<script>

angular.module('myApp', [])

.controller('helloController', function ($scope) {

$scope.message = 'Hello World';

// Function to add an exclamation mark to the message

$scope.addExclamation = function() {

$scope.message += '!';

};

// Function to change the message to a different greeting

$scope.changeGreeting = function() {

$scope.message = 'Greetings, Universe!';

};

});

</script>

</head>

<body>

<div ng-app="myApp" ng-controller="helloController">

<input type="text" ng-model="message">

<p>{{message}}</p>

<button ng-click="addExclamation()">Add Exclamation</button>

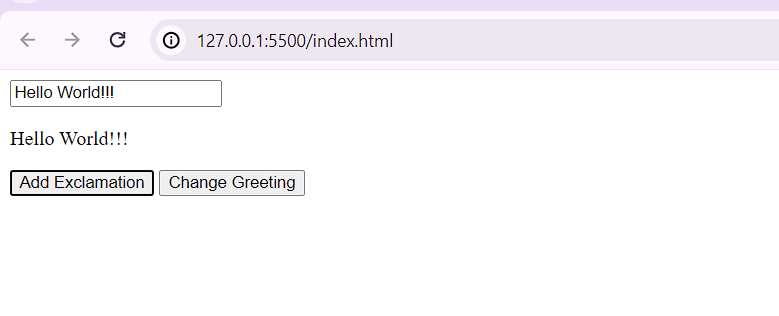
<button ng-click="changeGreeting()">Change Greeting</button>

</div>

</body>

</html>

**Output:**

****

**Practical no. 9**

**Aim:** Create an app using Flutter for User Authentication.

**Date:**20/02/2024 **Roll no.:** 03 **sign:**

**Flutter:**

Flutter is an open-source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.

Flutter gives easy and simple methods to start building beautiful mobile and desktop apps with a rich set of material design and widgets.

# To build a login page user interface:

Use the TextField widget, for user input as username and password. FlatButton widget, to show action. Also, use Image to set logo for login page.

**TextField widget is an input widget that helps you to take input from the user.**

1. The **Padding** widget helps you to give some space all around your TextField widget.
2. **obscureText** *true* property helps you to display special character in textfield instead of actual letters.
3. TextField widget has **decoration** property that takes **InputDecoration** widget to design textfield.
4. **border** property specifies **OutlineInputBorder** widget to give outline to textfield.
5. **labelText** property helps to write labels for textfield so that users can understand what he needs to input in that particular textfield.
6. **hintText** property helps to write specific hint about user input.
7. decoration property of container allows to design button, mention **color**(*Colors.blue*) and **borderRadius**(*BorderRadius.circular(20)*) property in decoration.

**Program:**

**main.dart**

import 'package:flutter/material.dart';

import 'HomePage.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

debugShowCheckedModeBanner: false,

home: LoginDemo(),

);

}

}

class LoginDemo extends StatefulWidget {

@override

\_LoginDemoState createState() => \_LoginDemoState();

}

class \_LoginDemoState extends State<LoginDemo> {

@override

Widget build(BuildContext context) {

return Scaffold(

backgroundColor: Colors.white,

appBar: AppBar(

title: Text("User Login Page"),

),

body: SingleChildScrollView(

child: Column(

children: <Widget>[

Padding(

padding: const EdgeInsets.only(top: 60.0),

child: Center(

child: Container(

width: 200,

height: 150,

child: Image.asset('assets/java.png'), // Fix image asset path

),

),

),

Padding(

padding: EdgeInsets.symmetric(horizontal: 15),

child: TextField(

decoration: InputDecoration(

border: OutlineInputBorder(),

labelText: 'Email',

hintText: 'Enter valid email id as abc@gmail.com',

),

),

),

Padding(

padding: const EdgeInsets.only(left: 15.0, right: 15.0, top: 15, bottom: 0),

child: TextField(

obscureText: true,

decoration: InputDecoration(

border: OutlineInputBorder(),

labelText: 'Password',

hintText: 'Enter secure password',

),

),

),

TextButton(

onPressed: (){

//TODO FORGOT PASSWORD SCREEN GOES HERE

},

child: Text(

'Forgot Password',

style: TextStyle(color: Colors.blue, fontSize: 15),

),

),

Container(

height: 50,

width: 250,

decoration: BoxDecoration(

color: Colors.blue,

borderRadius: BorderRadius.circular(20),

),

child: TextButton(

onPressed: () {

Navigator.push(

context, MaterialPageRoute(builder: (\_) => HomePage())

);

},

child: Text(

'Login',

style: TextStyle(color: Colors.white, fontSize: 25),

),

),

),

SizedBox(

height: 130,

),

Text('New User? Create Account')

],

),

),

);

}

}

**HomePage.dart**

import 'package:flutter/material.dart';

class HomePage extends StatefulWidget {

@override

\_HomePageState createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Home Page'),

),

body: Center(

child: Container(

height: 80,

width: 150,

decoration: BoxDecoration(

color: Colors.blue,

borderRadius: BorderRadius.circular(10),

),

child: TextButton(

onPressed: () {

Navigator.pop(context);

},

child: Text(

'Welcome',

style: TextStyle(color: Colors.white, fontSize: 25),

),

),

),

),

);

}

}

**pubspec.yaml**

name: flutter\_application\_2

description: "A new Flutter project."

publish\_to: 'none' # Remove this line if you wish to publish to pub.dev

version: 1.0.0+1

environment:

sdk: '>=3.2.6 <4.0.0'

dependencies:

flutter:

sdk: flutter

# The following adds the Cupertino Icons font to your application.

# Use with the CupertinoIcons class for iOS style icons.

cupertino\_icons: ^1.0.2

dev\_dependencies:

flutter\_test:

sdk: flutter

flutter\_lints: ^2.0.0

# The following section is specific to Flutter packages.

flutter:

uses-material-design: true

# To add assets to your application, add an assets section, like this:

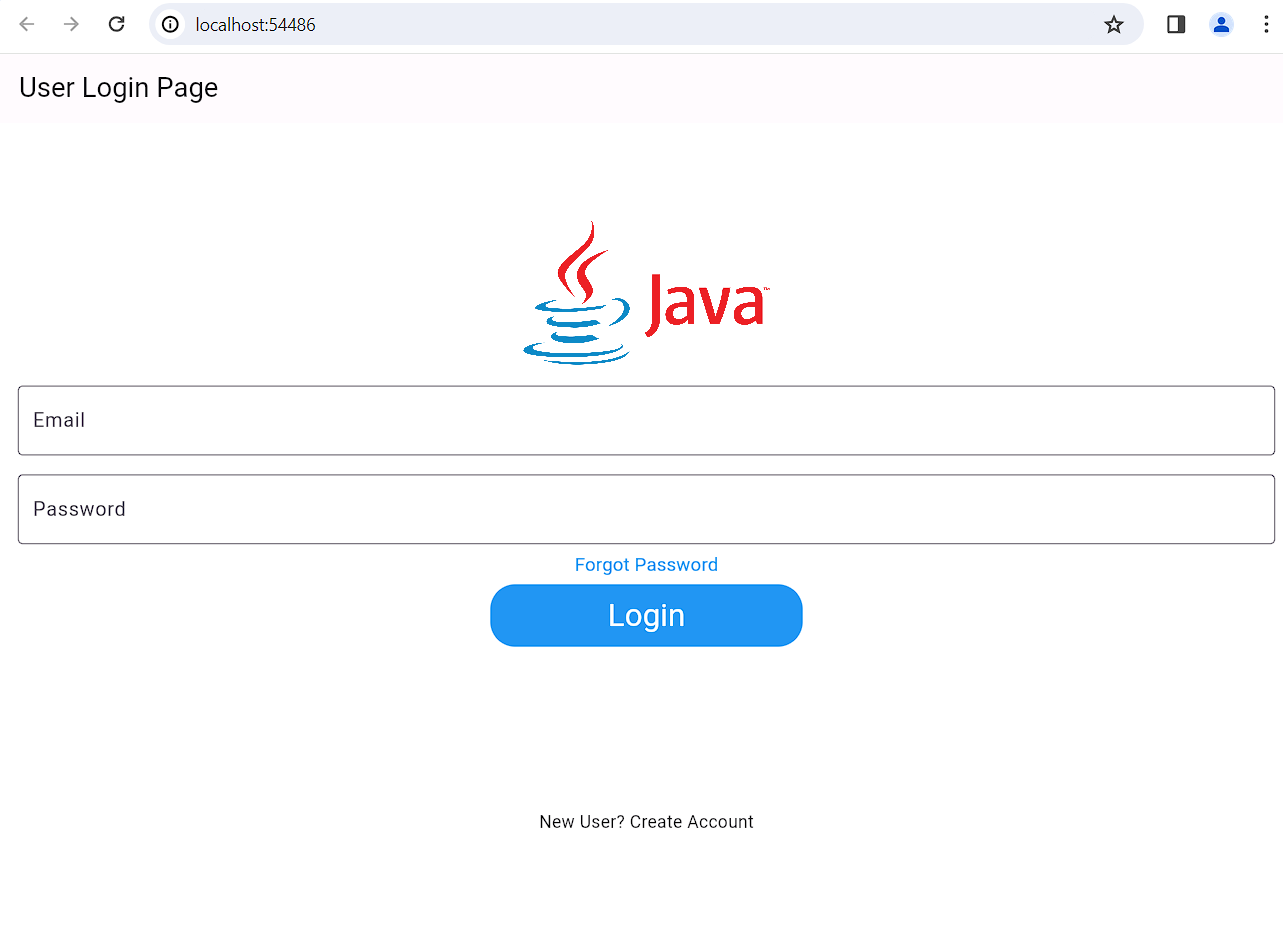
# assets:

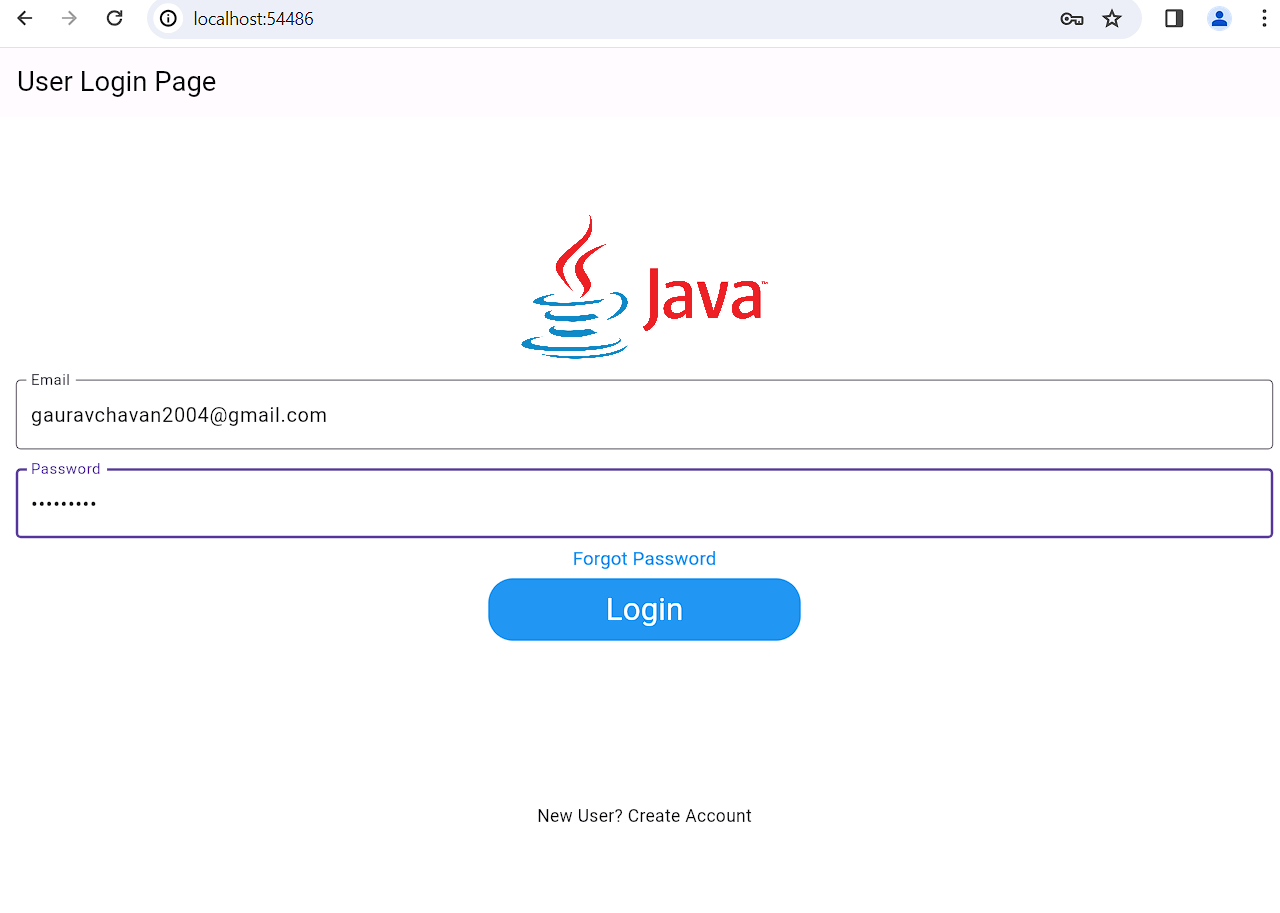
# -asset/images/java.png

assets:

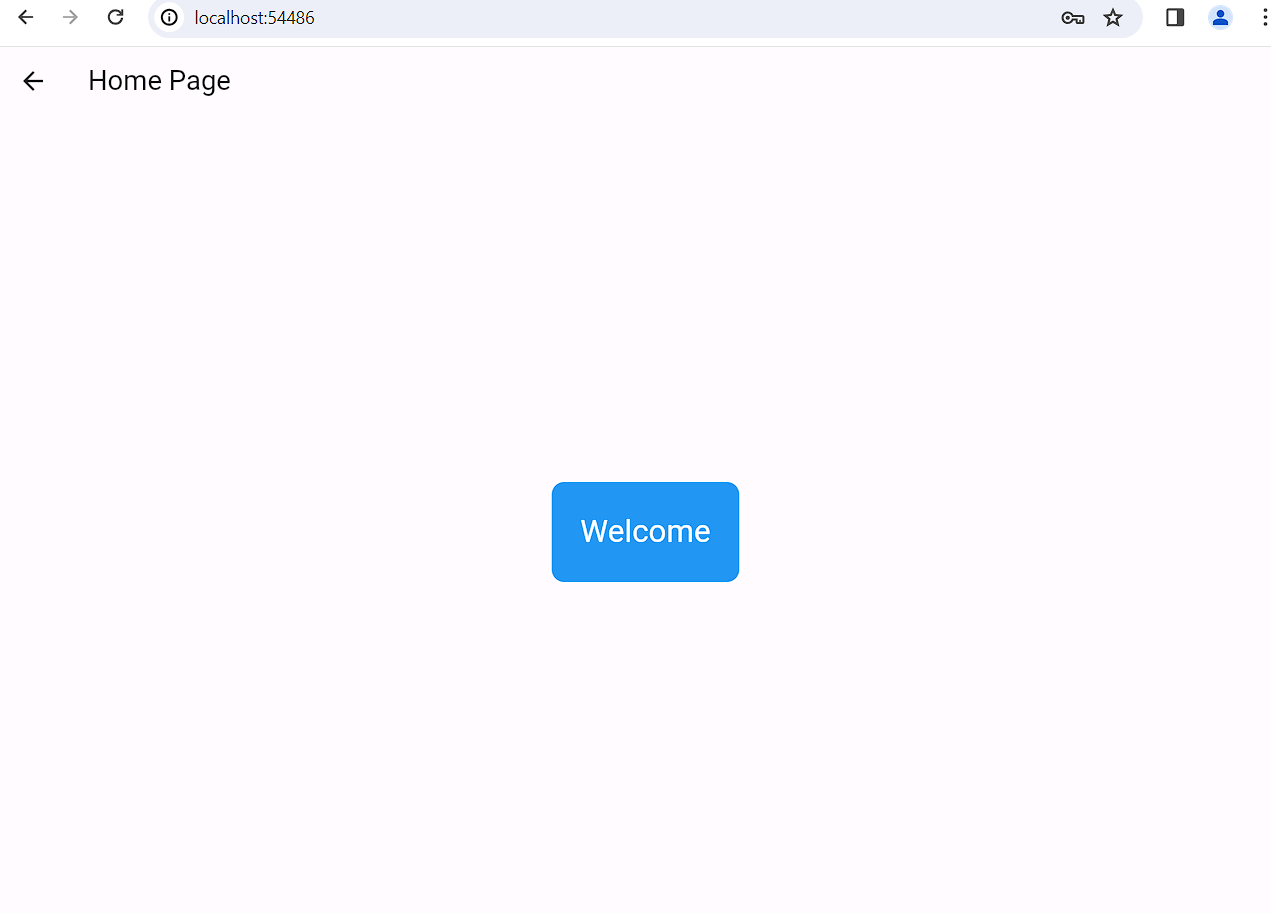
- assets/java.png

**Output:**

****

****

**After Click on Login Button:**

****

**Practical no. 10**

**Aim:** Create an app using Flutter to demonstrate navigation in an App.

**Date:** **Roll no.:** 03 **sign:**

**Flutter Navigation and Routing:**

Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages.

We know that every mobile application contains several screens for displaying different types of information.

**For example,**

an app can have a screen that contains various products. When the user taps

on that product, immediately it will display detailed information about that product.

* In Flutter, the screens and pages are known as **routes**, and these routes are just a **widget.**
* In Android, a route is similar to an Activity, whereas, in iOS, it is equivalent to a **ViewController.**
* In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as **routing.**
* Flutter provides a basic routing class **MaterialPageRoute** and

two methods:

**Navigator.push()** and **Navigator.pop()** that shows how to navigate between

two routes.

**Program:**

import 'package:flutter/material.dart';

void main() {

runApp(MaterialApp(

title: 'Flutter Navigation',

theme: ThemeData(

// This is the theme of your application.

primarySwatch: Colors.green,

),

home: FirstRoute(),

));

}

class FirstRoute extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('First Screen'),

),

body: Center(

child: ElevatedButton(

child: Text('Click Here'),

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(builder: (context) => SecondRoute()),

);

},

),

),

);

}

}

class SecondRoute extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text("Second Screen"),

),

body: Center(

child: ElevatedButton(

onPressed: () {

Navigator.pop(context);

},

child: Text('Go back'),

),

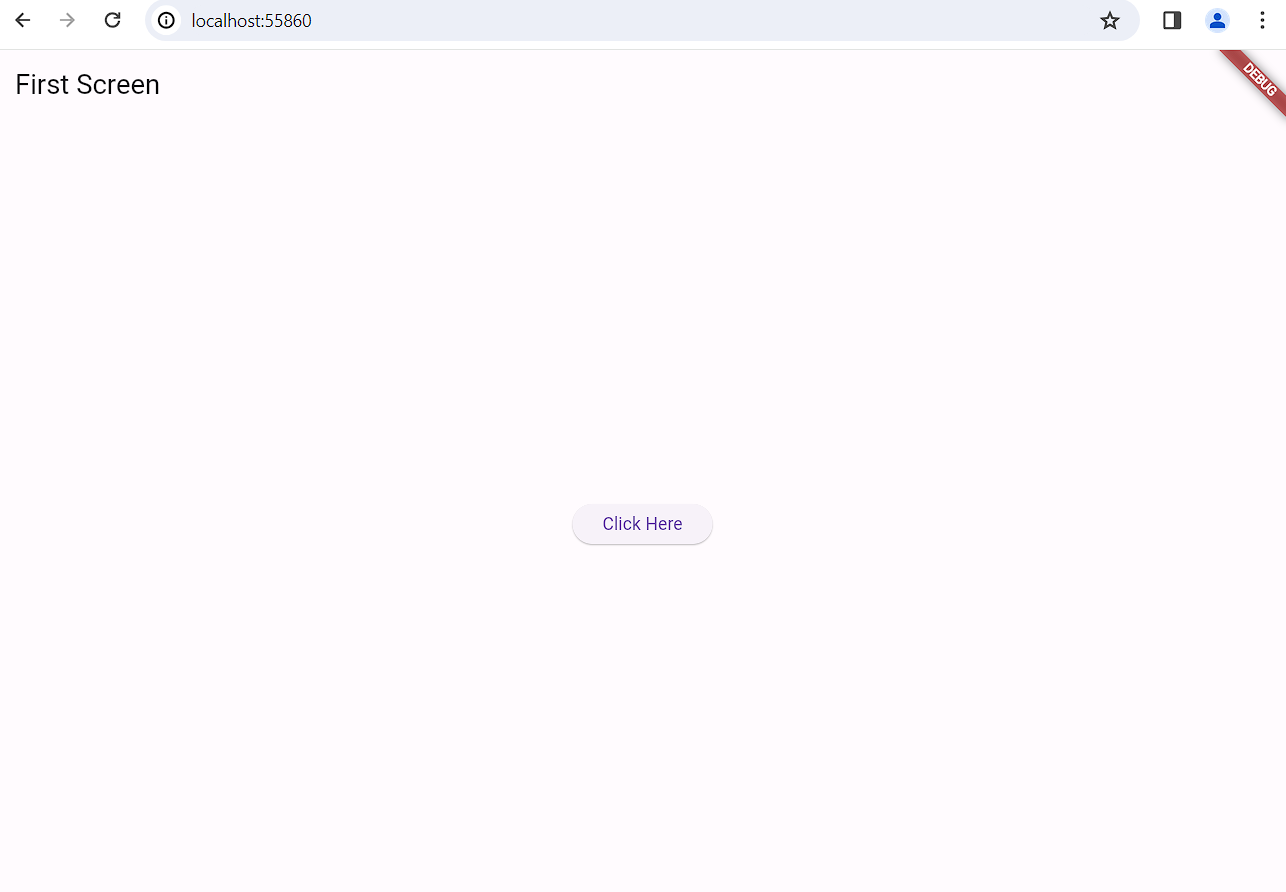
),

);

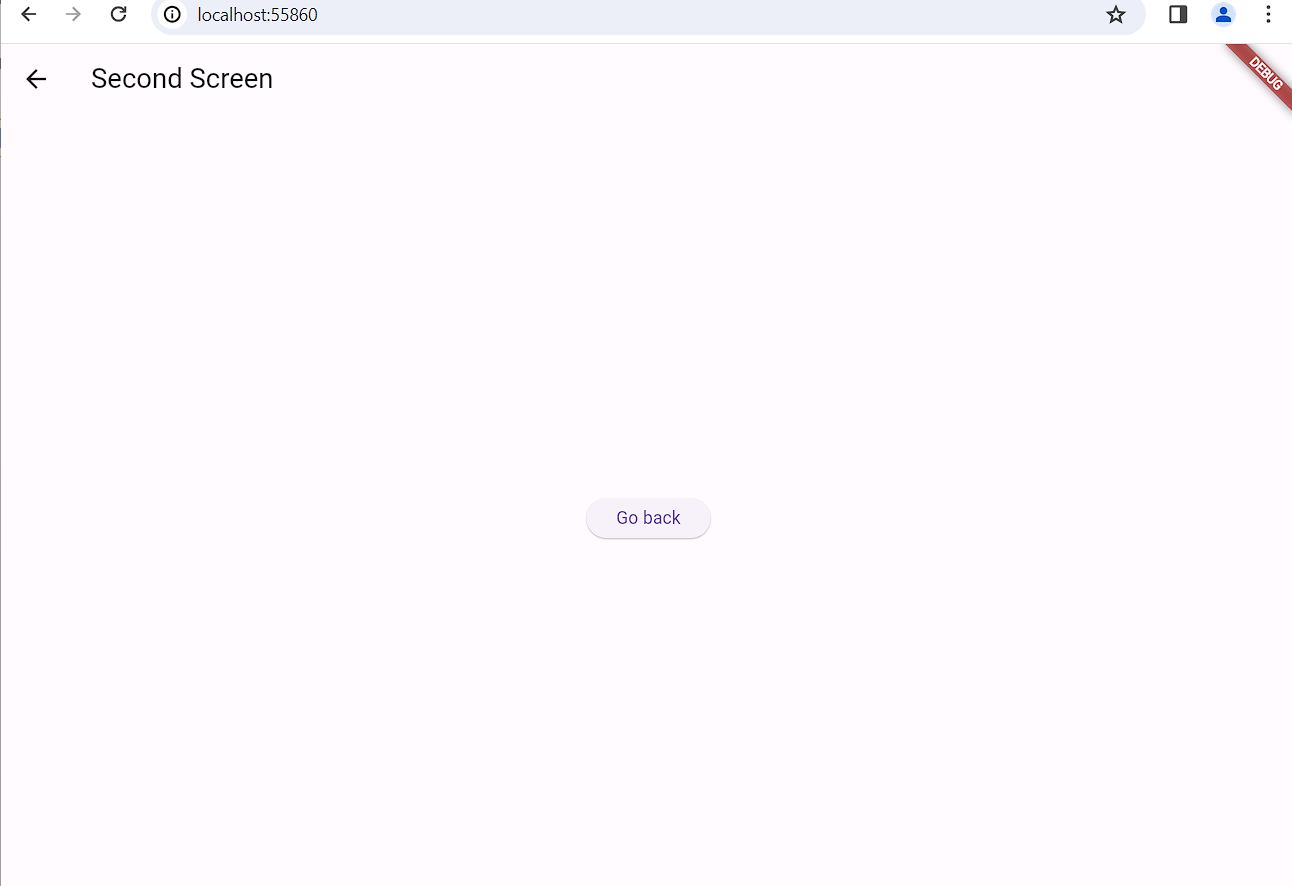
}

}

**Output:**

****

**After:**

****