PRACTICAL-1

JavaScript implementation of MongoDB data models:

1. Project Setup

- **Install MongoDB:** Follow the instructions on the official website.
- **Install Node.js and npm:** Download and install Node.js from the official website. This includes npm (Node Package Manager).
- **Create a Project Directory:** Make a new directory for your project.
- **Initialize npm:** In your project directory, run npm init -y to create a package.json file.
- **Install MongoDB Driver:** Install the MongoDB driver for Node.js: npm install mongodb

2. Define Data Models (using JavaScript objects)

JavaScript

```
const studentSchema = {
  _id: { type: String, required: true }, // Using String for simplicity
  name: { type: String, required: true },
  age: { type: Number, min: 0 },
  grades: { type: Array, of: Number },
  courses: { type: Array, of: String }
};

const courseSchema = {
  _id: { type: String, required: true },
  name: { type: String, required: true },
  description: { type: String },
  instructor: { type: String }
};
```

3. Connect to MongoDB

JavaScript

```
const { MongoClient } = require('mongodb');

const uri = "mongodb://localhost:27017/"; // Replace with your connection string

const client = new MongoClient(uri);

async function run() {
  try {
    await client.connect();
```

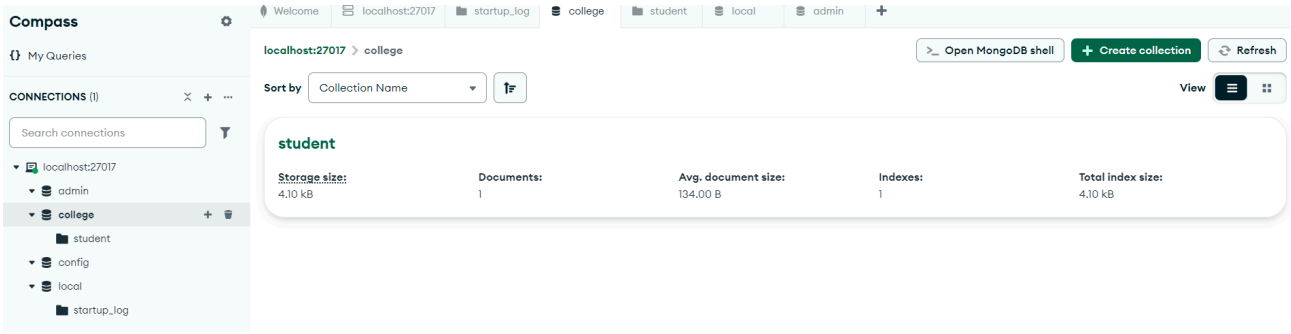
```
    console.log('Connected to MongoDB');

    // ... your database operations here ...

  } finally {
    // Ensures that the client will close when you finish/error
    await client.close();
  }
}

run().catch(console.dir);
```

```
C:\Users\shakshi kanojiya\Documents>node aad1.js
Connected to MongoDB
```

4. Create Collections

JavaScript

```
const db = client.db('your_database_name');
const studentsCollection = db.collection('students');
const coursesCollection = db.collection('courses');
```



5. Insert Data

JavaScript

```
const newStudent = {
  _id: '1',
  name: 'John Doe',
  age: 20,
  grades: [90, 85, 92],
  courses: ['Math', 'Science', 'History']
};

const result = await studentsCollection.insertOne(newStudent);
```

```
console.log('Inserted Document:', result);
```

```
C:\Users\shakshi kanojiya\Documents>node aad1.js
Connected to MongoDB
Inserted Document: { acknowledged: true, insertedId: '1' }
```

6. Read Data

JavaScript

```
const query = { age: { $gte: 20 } };
const cursor = studentsCollection.find(query);
const results = await cursor.toArray();
console.log('Found Documents:', results);
```

```
C:\Users\shakshi kanojiya\Documents>node aad1.js
Connected to MongoDB
Found Documents: [
  {
    _id: '1',
    name: 'John Doe',
    age: 20,
    grades: [ 90, 85, 92 ],
    courses: [ 'Math', 'Science', 'History' ]
  }
]
```

7. Update Data

JavaScript

```
const filter = { name: 'John Doe' };
const updateDoc = {
  $set: { age: 21 }
};

const result = await studentsCollection.updateOne(filter, updateDoc);
console.log('Updated Document:', result);
```

```
C:\Users\shakshi kanojiya\Documents>node aad1.js
Connected to MongoDB
Updated Document: {
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

8. Delete Data

JavaScript

```
const query = { age: { $lt: 18 } };
const result = await studentsCollection.deleteMany(query);
console.log('Deleted Documents:', result);
```

```
C:\Users\shakshi kanojiya\Documents>node aad1.js
Connected to MongoDB
Deleted Documents: { acknowledged: true, deletedCount: 0 }
```

9. Close Connection

- The connection is closed automatically in the finally block.

Tools and Notes for Students

- **MongoDB Compass:** Use the GUI for visualizing data and performing basic operations.
- **Node.js REPL:** Use the interactive console to experiment with code snippets.
- **Focus on:**
    - Data modeling principles.
    - CRUD operations.
    - Basic query operators (e.g., $gt, $lt, $in, $regex).
    - Data validation and sanitation.

Key Considerations

- **Error Handling:** Implement proper error handling to catch potential issues (e.g., connection errors, invalid data).
- **Asynchronous Operations:** Use async/await or promises to handle asynchronous operations effectively.
- **Security:** Always use appropriate security measures (e.g., authentication, authorization) to protect your MongoDB data.

Practical 2

Aim: Write a program to implement MongoDB CRUD Operations.

1. Create Operation

```
> use Aman
< switched to db Aman
> db["Staff"].find()
<
> db.Staff.insertOne({
  fname: "Rohan",
  mname: "Singh",
  lname: "Rathour"
  })
< {
    acknowledged: true,
    insertedId: ObjectId('67d8371475b1e2561f75ad81')
  }
```

```
> db.Students.insertMany([
  {
  fname: "Hitesh",
  mname: "Malviya",
```

2.Read Operation

```
> db.Students.find()
< {
    _id: ObjectId('67d8385d75b1e2561f75ad83'),
    fname: 'Vikas',
    mname: 'Makwanaa',
    lname: 'RajBhar',
    age: 30
  }
> db.Staff.find()
< {
    _id: ObjectId('67d8371475b1e2561f75ad81'),
    fname: 'Devi',
    mname: 'Singh',
    lname: 'Rathour',
    age: 23
  }
> db.Students.findOne({ fname: "Chappri" })
< null
```

3. Update Operation

```
> db.Staff.updateOne({fname: "Rohan"},{$set:{fname:"Kamal"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.Staff.updateOne({mname: "Singh"},{$set:{fname:"Devi"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

4. Delete Operation

```
> db.Students.deleteOne({mname: "Malviya"})
< {
    acknowledged: true,
    deletedCount: 1
  }
```

Practical 3

Aim: Write a program to perform validation of a form using Angular JS

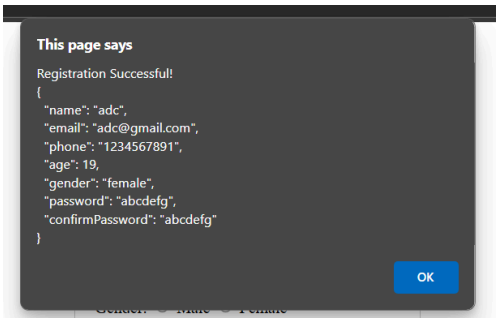
Code:

```
<!DOCTYPE html>
<html ng-app="registrationApp">
<head>
  <title>User Registration Form</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <style>
    .error { color: red; }
    form { width: 300px; margin: 20px auto; padding: 20px; border: 1px solid #ccc; }
.form-group { margin-bottom: 15px; }
  </style>
</head>
<body>
  <div ng-controller="RegistrationController">
    <form name="registrationForm" ng-submit="submitForm()" novalidate>
      <h2>User Registration Form</h2>

      <!-- Name -->
      <div class="form-group">
        <label>Name:</label>
        <input type="text" name="name" ng-model="user.name" required>
        <span class="error" ng-show="registrationForm.name.$touched &&
registrationForm.name.$invalid">
          Name is required
        </span>
      </div>

      <!-- Email -->
      <div class="form-group">
        <label>Email:</label>
        <input type="email" name="email" ng-model="user.email" required>
    Valid email is required
  </span>
      </div>

      <!-- Phone -->
      <div class="form-group">
        <label>Phone:</label>
        <input type="tel" name="phone" ng-model="user.phone" pattern="\d{10}" required>
        <span class="error" ng-show="registrationForm.phone.$touched &&
registrationForm.phone.$invalid">
```

```
            10-digit phone number required
          </span>
        </div>

        <!-- Age -->
        <div class="form-group">
          <label>Age:</label>
          <input type="number" name="age" ng-model="user.age" min="1" max="120"
required>
          <span class="error" ng-show="registrationForm.age.$touched &&
registrationForm.age.$invalid">
            Valid age (1-120) required
          </span>
        </div>

        <!-- Gender -->
        <div class="form-group">
          <label>Gender:</label>
          <input type="radio" name="gender" ng-model="user.gender" value="male" required>
Male
          <input type="radio" name="gender" ng-model="user.gender" value="female"> Female
          <span class="error" ng-show="registrationForm.gender.$touched &&
registrationForm.gender.$invalid">
            Gender is required
          </span>
    </div>

        <!-- Password -->
        <div class="form-group">
          <label>Password:</label>
          <input type="password" name="password" ng-model="user.password" required>
          <span class="error" ng-show="registrationForm.password.$touched &&
registrationForm.password.$invalid">
            Password required
          </span>
        </div>

        <!-- Confirm Password -->
        <div class="form-group">
          <label>Confirm Password:</label>
          <input type="password" name="confirmPassword" ng-model="user.confirmPassword"
required>
          <span class="error" ng-show="user.password != user.confirmPassword">
```

```
            Passwords must match
          </span>
        </div>

        <button type="submit" ng-disabled="registrationForm.$invalid || user.password !=
user.confirmPassword">
          Submit
        </button>
      </form>
    </div>

    <script>       angular.module('registrationApp', [])
      .controller('RegistrationController', ['$scope', function($scope) {
        $scope.submitForm = function() {
    if($scope.registrationForm.$valid) {
      alert('Registration Successful!\n' + JSON.stringify($scope.user, null, 2));
          }
        };
      }]);
    </script>
</body>
</html>
```

Output:


This page says
Registration Successful!
{
"name": "adc",
"email": "adc@gmail.com",
"phone": "1234567891",
"age": 19,
"gender": "female",
"password": "abcdefg",
"confirmPassword": "abcdefg"
}
OK

Practical 4

Aim: write a program to perform to create and implement and controllers in Angular JS

Code:

```
<!DOCTYPE html>
<html lang="en" ng-app="mainApp">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AngularJS Modules Demo</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <style>
    .section { margin: 20px; padding: 20px; border: 1px solid #ddd; }
    table { width: 100%; margin-top: 10px; border-collapse: collapse; }
    th, td { padding: 8px; text-align: left; border: 1px solid #ddd; }
    .selected { background-color: #e0f0ff; }
  </style>
</head>
<body>

  <!-- Main Application Module -->
  <div ng-controller="MainController as mainCtrl">
    <h1>AngularJS 1.x Modules and Controllers Demo</h1>

    <!-- User Management Module -->
    <div ng-controller="UserController as userCtrl" class="section">
      <h2>User Management</h2>

      <!-- Add User Form -->
      <div>
        <input type="text" ng-model="userCtrl.newUser.name" placeholder="Name">
        <input type="email" ng-model="userCtrl.newUser.email" placeholder="Email">
        <select ng-model="userCtrl.newUser.role">
          <option value="">Select a role</option>
          <option value="Admin">Admin</option>
          <option value="User">User</option>
          <option value="Guest">Guest</option>
        </select>
        <button ng-click="userCtrl.addUser()">Add User</button>
      </div>

      <!-- User List -->
      <div>
        <input type="text" ng-model="userCtrl.searchText" placeholder="Filter users...">
        <table>
```

```
            <thead>
              <tr>
                <th>Name</th>
                <th>Email</th>
                <th>Role</th>
                <th>Actions</th>
              </tr>
            </thead>
            <tbody>
              <tr ng-repeat="user in userCtrl.users | filter:userCtrl.searchText" ng-class="{selected:
user.selected}">
                <td>{{ user.name }}</td>
                <td>{{ user.email }}</td>
                <td>{{ user.role }}</td>
                <td>
                  <button ng-click="userCtrl.selectUser(user)">Select</button>
                  <button ng-click="userCtrl.removeUser($index)">Remove</button>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>

      <!-- Product Management Module -->
      <div ng-controller="ProductController as productCtrl" class="section">
        <h2>Product Management</h2>

        <!-- Add Product Form -->
        <div>
          <input type="text" ng-model="productCtrl.newProduct.name" placeholder="Product Name">
          <input type="text" ng-model="productCtrl.newProduct.category" placeholder="Category">
          <input type="number" ng-model="productCtrl.newProduct.price" placeholder="Price">
          <button ng-click="productCtrl.addProduct()">Add Product</button>
        </div>

        <!-- Product List -->
        <table>
          <thead>
            <tr>
              <th>Product Name</th>
              <th>Category</th>
              <th>Price</th>
              <th>Actions</th>
```

```
            </tr>
          </thead>
          <tbody>
            <tr ng-repeat="product in productCtrl.products">
              <td>{{ product.name }}</td>
              <td>{{ product.category }}</td>
              <td>{{ product.price | currency }}</td>
              <td>
                <button ng-click="productCtrl.selectProduct(product)">Select</button>
                <button ng-click="productCtrl.removeProduct($index)">Remove</button>
              </td>
            </tr>
          </tbody>
        </table>
      </div>

    </div>

    <script>
      // Create main application module
      angular.module('mainApp', []);

      // User Management Controller
      angular.module('mainApp').controller('UserController', function() {
        var vm = this;
        vm.users = [
          { name: 'John Doe', email: 'john@example.com', role: 'Admin' },
          { name: 'Jane Smith', email: 'jane@example.com', role: 'User' },
          { name: 'Mike Johnson', email: 'mike@example.com', role: 'Guest' }
        ];

        vm.newUser = {};

        vm.addUser = function() {
          if (vm.newUser.name && vm.newUser.email && vm.newUser.role) {
            vm.users.push(angular.copy(vm.newUser));
            vm.newUser = {};
          }
        };

        vm.removeUser = function(index) {
          vm.users.splice(index, 1);
        };
```

```
        vm.selectUser = function(user) {
            vm.users.forEach(u => u.selected = false);
            user.selected = true;
        };
    });

    // Product Management Controller
    angular.module('mainApp').controller('ProductController', function() {
        var vm = this;
        vm.products = [];

        vm.newProduct = {};

        vm.addProduct = function() {
            if (vm.newProduct.name && vm.newProduct.category && vm.newProduct.price) {
                vm.products.push(angular.copy(vm.newProduct));
                vm.newProduct = {};
            }
        };

        vm.removeProduct = function(index) {
            vm.products.splice(index, 1);
        };

        vm.selectProduct = function(product) {
            vm.products.forEach(p => p.selected = false);
            product.selected = true;
        };
    });

    // Main Controller
    angular.module('mainApp').controller('MainController', function() {
        // Placeholder for potential future logic
    });
  </script>

</body>
</html>
```

Output:

AngularJS 1.x Modules and Controllers Demo

User Management

Name | Email | Select a role | Add User

Filter users...

Name	Email	Role	Actions
Anushka Singh	anushka@gmail.com	User	Select Remove

Product Management

Product Name | Category | Price | Add Product

Product Name	Category	Price	Actions
Sasta Sundar Tikao	Clothes	$999.00	Select Remove

PRACTICAL-5

Aim: Demonstrating Errors With Angular js

Html code:

HTML CODE:

```
<!DOCTYPE html>
<html lang="en" ng-app="errorHandlingApp">

<head>
    <meta charset="UTF-8">
    <title>Error Handling in AngularJS</title>

    <!-- Correct p5.js Script -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.4.0/p5.js"></script>

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

    <style>
        body {
            background-color: navajowhite;
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }

        button {
            padding: 10px 20px;
            font-size: 16px;
            background-color: #008cba;
            color: white;
```

```
            border: none;
            cursor: pointer;
            margin-top: 20px;
        }

        button:hover {
            background-color: #005f75;
        }

        p {
            font-size: 18px;
            color: darkgreen;
        }

        .error-message {
            color: red;
            font-size: 18px;
            margin-top: 20px;
        }
    </style>
</head>

<body ng-controller="MainController as ctrl">

    <h1>AngularJS Error Handling Example</h1>

    <button ng-click="ctrl.fetchData()">Fetch Data</button>
    <p>{{ ctrl.data }}</p>

    <div ng-if="ctrl.errorMessage" class="error-message">
```

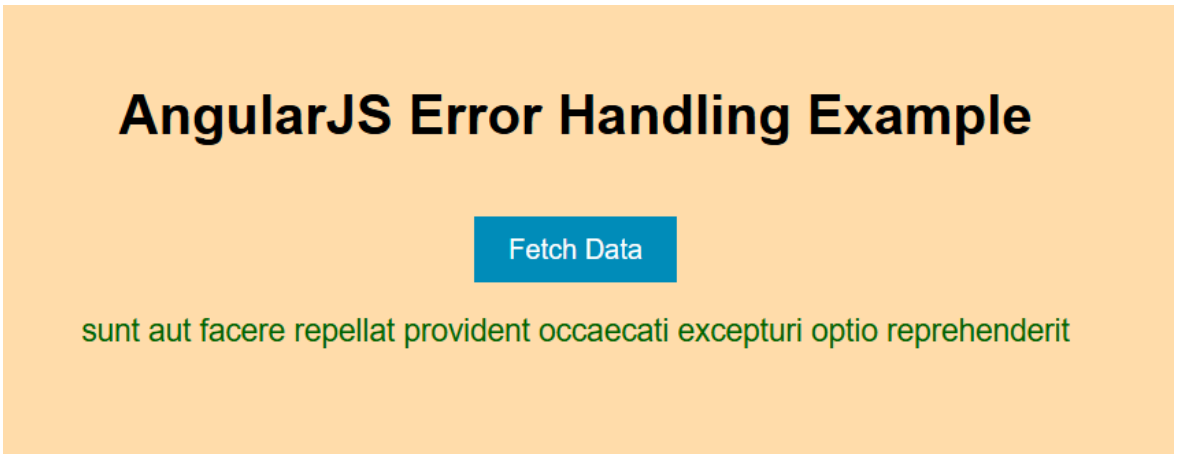
```
    <strong>Error:</strong> {{ ctrl.errorMessage }}
</div>

<!-- AngularJS Controller Script -->
<script>
    // Define AngularJS app and controller
    angular.module('errorHandlingApp', [])
        .controller('MainController', function ($http) {
            var ctrl = this;

            // Initial values for data and error message
            ctrl.data = '';
            ctrl.errorMessage = '';

            // Fetch data function
            ctrl.fetchData = function () {
                // Simulating an API call
                $http.get('https://jsonplaceholder.typicode.com/posts/1')
                    .then(function (response) {
                        // On success, update the data
                        ctrl.data = response.data.title;
                    })
                    .catch(function (error) {
                        // On error, show the error message
                        ctrl.errorMessage = 'An error occurred while fetching data.';
                    });
            };
        });
</script>
```

```
</body>


</html>
```

JS Code:

```
angular.module('errorHandlingApp', [])
  .controller('MainController', ['$http', function ($http) {
    var vm = this;
    vm.data = '';
    vm.errorMessage = '';

    vm.fetchData = function () {
      $http.get('https://jsonplaceholder.typicode.com/invalid-url')
        .then(function (response) {
          vm.data = response.data;
          vm.errorMessage = '';
        })
        .catch(function (error) { // Proper error handling
          vm.errorMessage = 'Failed to fetch data. Please try again later.';
          console.error('Error:', error);
        });
    };
  }]);
```

Output:



Practical 6

AIM: Build student/customer records management system using angular JS

Code:

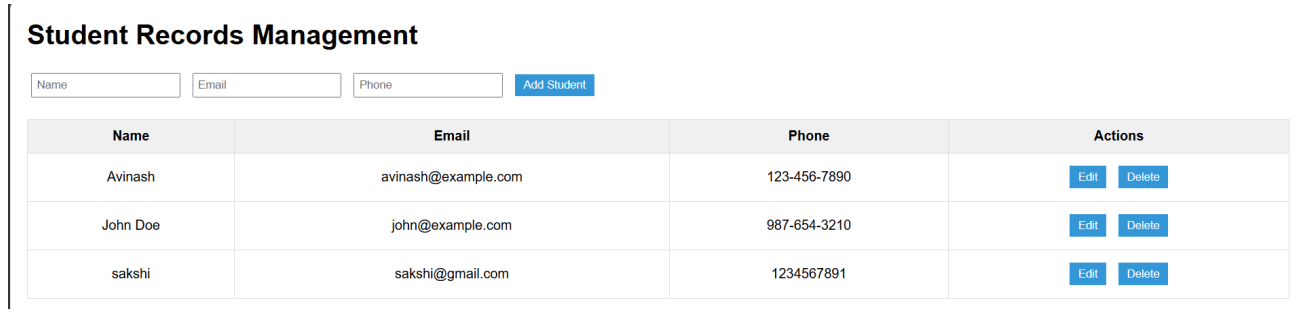
```
<!DOCTYPE html>
<html lang="en" ng-app="studentApp">

<head>
  <meta charset="UTF-8">
  <title>Student Records Management</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }

    input {
      margin: 5px;
      padding: 5px;
    }

    button {
```

```
            padding: 5px 10px;
            margin: 5px;
            background-color: #3498db;
            color: white;
            border: none;
            cursor: pointer;
        }

        button:hover {
            background-color: #2c3e50;
        }

        table {
            width: 100%;
            margin-top: 20px;
            border-collapse: collapse;
        }

        th, td {
            border: 1px solid #ddd;
            padding: 10px;
            text-align: center;
        }

        th {
            background-color: #f4f4f4;
        }
    </style>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>

<body ng-controller="StudentController as ctrl">
    <h1>Student Records Management</h1>

    <!-- Add/Edit Student Form -->
    <div>
        <input type="text" ng-model="ctrl.newStudent.name" placeholder="Name">
        <input type="email" ng-model="ctrl.newStudent.email" placeholder="Email">
        <input type="text" ng-model="ctrl.newStudent.phone" placeholder="Phone">
        <button ng-click="ctrl.addOrUpdateStudent()">
            {{ ctrl.isEditing ? 'Update Student' : 'Add Student' }}
        </button>
```

```
</div>

<!-- Students Table -->
<table>
  <tr>
    <th>Name</th>
    <th>Email</th>
    <th>Phone</th>
    <th>Actions</th>
  </tr>
  <tr ng-repeat="student in ctrl.students">
    <td>{{ student.name }}</td>
    <td>{{ student.email }}</td>
    <td>{{ student.phone }}</td>
    <td>
      <button ng-click="ctrl.editStudent($index)">Edit</button>
      <button ng-click="ctrl.deleteStudent($index)">Delete</button>
    </td>
  </tr>
</table>

<script>
  // AngularJS Module
  angular.module('studentApp', [])
    .controller('StudentController', function () {
      var vm = this;

      // Initial Student Data
      vm.students = [
        { name: 'Avinash', email: 'avinash@example.com', phone: '123-456-7890' },
        { name: 'John Doe', email: 'john@example.com', phone: '987-654-3210' }
      ];

      // Initialize newStudent object
      vm.newStudent = {};
      vm.isEditing = false;
      vm.editIndex = -1;

      // Add or Update Student Function
      vm.addOrUpdateStudent = function () {
        if (vm.newStudent.name && vm.newStudent.email && vm.newStudent.phone) {
          if (vm.isEditing) {
            vm.students[vm.editIndex] = angular.copy(vm.newStudent);
```

```
            vm.isEditing = false;
            vm.editIndex = -1;
          } else {
            vm.students.push(angular.copy(vm.newStudent));
          }
          vm.newStudent = {}; // Clear form
        }
      };

      // Edit Student Function
      vm.editStudent = function (index) {
        vm.newStudent = angular.copy(vm.students[index]);
        vm.isEditing = true;
        vm.editIndex = index;
      };

      // Delete Student Function
      vm.deleteStudent = function (index) {
        if (confirm('Are you sure you want to delete this student?')) {
          vm.students.splice(index, 1);
        }
      };
    });
  </script>
</body>

</html>
```

Output:



Practical 7

AIM: Write a program to create a simple web application using express, nodejs, angular js

index.html

```
<!DOCTYPE html>
<html lang="en" ng-app="StudentApp">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Management System</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
  <script src="app.js"></script>
  <style>
    body { font-family: Arial, sans-serif; }
    input, button { margin: 5px; padding: 8px; }
    table { width: 100%; border-collapse: collapse; margin-top: 10px; }
    th, td { border: 1px solid black; padding: 10px; text-align: center; }
    th { background-color: #f2f2f2; }
    button { cursor: pointer; }
  </style>
</head>
<body ng-controller="StudentController">
  <h1>Student Management System</h1>

  <input type="text" placeholder="Name" ng-model="newName" required>
  <input type="number" placeholder="Age" ng-model="newAge" required>
  <input type="text" placeholder="Course" ng-model="newCourse" required>
  <button ng-click="addStudent()">Add Student</button>

  <table>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Age</th>
      <th>Course</th>
      <th>Action</th>
    </tr>
```

```
      <tr ng-repeat="student in students">
        <td>{{ student.id }}</td>
        <td>{{ student.name }}</td>
        <td>{{ student.age }}</td>
        <td>{{ student.course }}</td>
        <td>
          <button ng-click="deleteStudent(student.id)">Delete</button>
        </td>
      </tr>
    </table>
</body>
</html>
```

server.js

```
const express = require("express");
const cors = require("cors");
const bodyParser = require("body-parser");
const path = require("path");

const app = express();
const PORT = 3000;

// Enable CORS for all requests
app.use(cors());

// Parse JSON request bodies
app.use(bodyParser.json());

// Serve static files (Frontend)
app.use(express.static(path.join(__dirname, "public")));
```

```
let students = [];
let idCounter = 1;

// Get all students
app.get("/students", (req, res) => {
   res.json(students);
});

// Add a new student
app.post("/students", (req, res) => {
   if (!req.body.name || !req.body.age || !req.body.course) {
      return res.status(400).json({ error: "All fields are required" });
   }

   const newStudent = {
      id: idCounter++,
      name: req.body.name,
      age: req.body.age,
      course: req.body.course
   };
   students.push(newStudent);
   res.json(newStudent); // Return the added student
});

// Delete a student
```

```
app.delete("/students/:id", (req, res) => {
   const studentId = parseInt(req.params.id);
   students = students.filter(student => student.id !== studentId);
   res.json({ message: "Student deleted successfully" });
});

// Start the server
app.listen(PORT, () => {
   console.log(`Server running on http://localhost:${PORT}`);
});
```

app.js

```
var app = angular.module("StudentApp", []);

app.controller("StudentController", function($scope, $http) {
   $scope.students = [];

   // Fetch students from server
   function fetchStudents() {
      $http.get("/students").then(function(response) {
         $scope.students = response.data;
      });
   }

   // Call fetch on page load
   fetchStudents();

   // Add a new student
   $scope.addStudent = function() {
      if (!$scope.newName || !$scope.newAge || !$scope.newCourse) {
         alert("Please fill all fields");
         return;
```

```
        }

        var newStudent = {
            name: $scope.newName,
            age: $scope.newAge,
            course: $scope.newCourse
        };

        $http.post("/students", newStudent).then(function(response) {
            $scope.students.push(response.data); // Update UI instantly
            $scope.newName = "";
            $scope.newAge = "";
            $scope.newCourse = "";
        });
    };

    // Delete a student
    $scope.deleteStudent = function(id) {
        $http.delete("/students/" + id).then(function(response) {
            fetchStudents(); // Refresh list
        });
    };
});
```

OUTPUT:

Student Management System

Name

Age

Course

Add Student

ID	Name	Age	Course	Action

Practical 8

Using SQLite for Local Authentication

Step 1: Add Dependencies

Modify pubspec.yaml:

```
dependencies:
  flutter:
    sdk: flutter sqflite: latest_version

  path_provider: latest_version
```

Run: flutter pub get

Step 2: Create Database Helper (database_helper.dart)

```
import 'package:sqflite/sqflite.dart'; import 'package:path/path.dart'; class DatabaseHelper { static

Database? _database; static final DatabaseHelper instance = DatabaseHelper._privateConstructor();

  DatabaseHelper._privateConstructor();
  Future<Database> get database async { if (_database !=

    null) return _database!; _database = await

    _initDatabase(); return _database!;

  }
  Future<Database> _initDatabase() async {
    String path = join(await getDatabasesPath(), 'user_database.db'); return await

    openDatabase( path, version: 1,

      onCreate: (db, version) async { await

        db.execute('''

          CREATE TABLE users ( id INTEGER PRIMARY KEY

            AUTOINCREMENT, email TEXT UNIQUE,

            password TEXT

          )
```

```
''');
        },
      );
  }
  Future<int> registerUser(String email, String password) async { final db = await database;

    return await db.insert('users', {'email': email, 'password': password});


  }
  Future<Map<String, dynamic>?> getUser(String email, String password) async { final db = await

    database;

    List<Map<String, dynamic>> result = await db.query(
      'users', where: 'email = ? AND password = ?',

      whereArgs: [email, password],

    ); return result.isNotEmpty ? result.first : null;

  }
}
```

Step 3: Create a Register Screen (register_screen.dart)

```
import 'package:flutter/material.dart'; import

'database_helper.dart'; class RegisterScreen extends

StatefulWidget {

  @override

  _RegisterScreenState createState() => _RegisterScreenState();
} class _RegisterScreenState extends State<RegisterScreen> {

  final TextEditingController emailController = TextEditingController(); final

  TextEditingController passwordController = TextEditingController(); final DatabaseHelper

  dbHelper = DatabaseHelper.instance;
```

```
  Future<void> register() async { try { await

dbHelper.registerUser(emailController.text, passwordController.text);

        Navigator.pop(context);
      } catch (e) { print("Registration Failed: $e");

      }
  }
  @override
  Widget build(BuildContext context) { return Scaffold(

      appBar: AppBar(title: Text("Register")), body: Padding(

      padding: const EdgeInsets.all(16.0), child: Column(

      children: [

                          TextField(controller: emailController, decoration:
InputDecoration(labelText: "Email")),
                  TextField(controller: passwordController, decoration:
 InputDecoration(labelText: "Password"), obscureText: true),
                                      ElevatedButton(onPressed: register, child: Text("Register")),
              ],
            ),
        ),
      );
  }
}
```

Step 4: Create a Login Screen (login_screen.dart)

```
import 'package:flutter/material.dart'; import

'database_helper.dart'; import 'home_screen.dart';

import 'register_screen.dart'; class LoginScreen

extends StatefulWidget { @override

  _LoginScreenState createState() => _LoginScreenState();
```

```
} class _LoginScreenState extends State<LoginScreen> { final TextEditingController
emailController = TextEditingController(); final TextEditingController passwordController =
TextEditingController(); final DatabaseHelper dbHelper = DatabaseHelper.instance;
  Future<void> login() async { final user = await
dbHelper.getUser(emailController.text, passwordController.text); if (user !=
null) {
        Navigator.pushReplacement( context,
          MaterialPageRoute(builder: (context) => HomeScreen()),
        );
      } else { print("Invalid Credentials");
      }
  }
  @override
  Widget build(BuildContext context) { return Scaffold(
      appBar: AppBar(title: Text("Login")), body: Padding(
      padding: const EdgeInsets.all(16.0), child: Column(
      children: [
                TextField(controller: emailController, decoration:
InputDecoration(labelText: "Email")),
                TextField(controller: passwordController, decoration:
InputDecoration(labelText: "Password"), obscureText: true),
                      ElevatedButton(onPressed: login, child: Text("Login")),
                TextButton( onPressed: () {
                                    Navigator.push(context, MaterialPageRoute(builder: (context) =>
RegisterScreen())); }, child: Text("Don't have an account? Register"),
                ),
              ],
            ),
        ),
      );
  }
```

```
}
```

Step 5: Create Home Screen (home_screen.dart)

```
import 'package:flutter/material.dart'; import 'login_screen.dart';

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) { return Scaffold( appBar: AppBar(title: Text("Home")), body: Center( child: Column( mainAxisAlignment: MainAxisAlignment.center, children: [
            Text("Welcome to Home Screen!"),
            ElevatedButton( onPressed: () {
              Navigator.pushReplacement( context,
                MaterialPageRoute(builder: (context) => LoginScreen()),
              ); }, child: Text("Logout"),
            ),
          ],
        ),
      ),
    );
  }
}
```

爵 **Project File Structure**

```
auth_app/
├── android/              # Android-specific files
├── ios/                  # iOS-specific files
├── lib/                  # Main Flutter code
│   ├── main.dart         # Main entry point of the app
│   ├── screens/          # UI Screens
│   │   ├── login_screen.dart      # Login screen UI
│   │   ├── register_screen.dart   # Registration screen UI
│   │   ├── home_screen.dart       # Home screen UI (after login)
│   ├── database/         # Database-related files
```

```
│   │   │── database_helper.dart   # SQLite database helper
│   │── services/              # Business logic and authentication
│   │   │── auth_service.dart      # Authentication logic (optional for
SharedPreferences)
│── pubspec.yaml
│── README.md

# Dependencies and project settings
# Documentation
```

1. main.dart (Entry Point)

. This file initializes the app and sets the **first screen** to LoginScreen().

2. screens/ (User Interface)

. **login_screen.dart** → Login form where users enter email & password. .

register_screen.dart → Registration form for new users.

. **home_screen.dart** → Home screen shown after successful login.

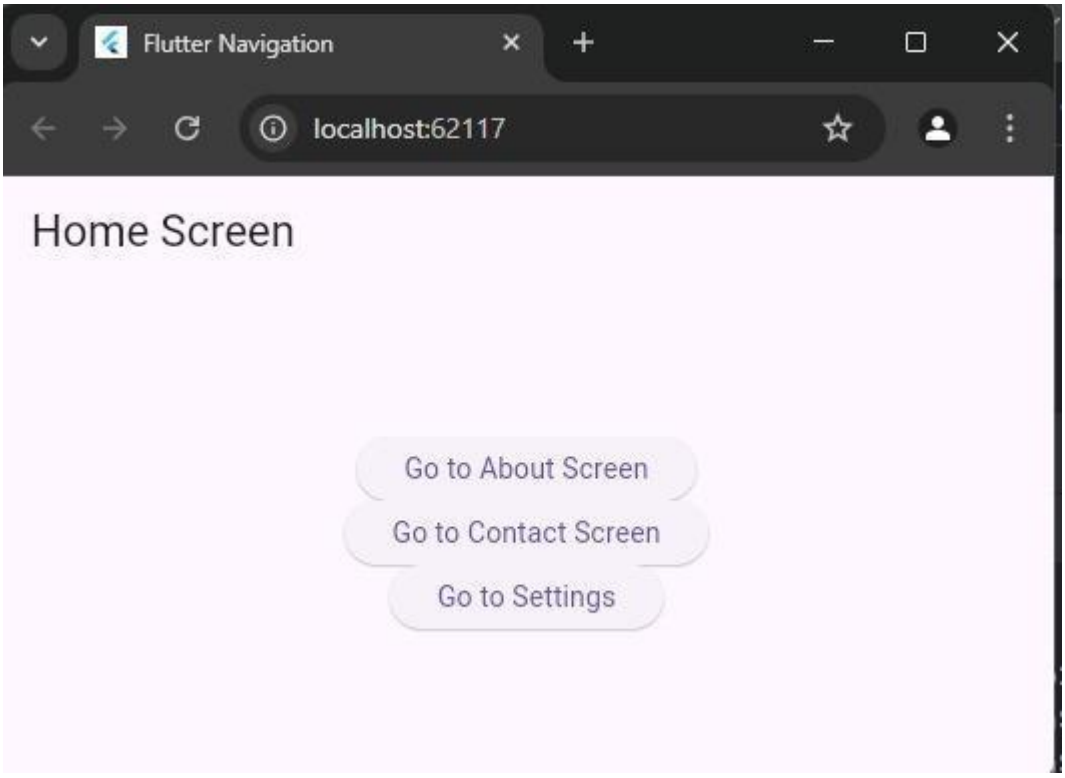
3. database/ (Database Management)

. **database_helper.dart** → Handles SQLite storage for user credentials.

4. services/ (Business Logic)

. **auth_service.dart** → Handles authentication logic using SharedPreferences (only needed if not using SQLite).

1. **Navigate to your Flutter project folder:** cd auth_app
2. **Run the app:**
   flutter run


Flutter Navigation
localhost:62117
Home Screen
Go to About Screen
Go to Contact Screen
Go to Settings

Practical 9

Flutter Navigation App (Demonstrating Navigation Between Screens) 🚀

This Flutter app will showcase **navigation** using **Navigator.push** and

Navigator.pushReplacement to switch between multiple screens.

爵 **Project Structure**

```
navigation_app/
├── lib/
│   ├── main.dart                    # Main entry point
│   ├── screens/                     # Screens folder
│   │   ├── home_screen.dart         # Home Screen
│   │   ├── about_screen.dart        # About Screen
│   │   ├── contact_screen.dart      # Contact Screen
│   │   ├── settings_screen.dart     # Settings Screen
├── pubspec.yaml                     # Dependencies and configurations
```

Step 1: Create a Flutter Project

```
flutter create navigation_app cd navigation_app
```

Step 2: Define main.dart (Entry Point)

```
import 'package:flutter/material.dart'; import 'screens/home_screen.dart'; void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) { return MaterialApp(
    debugShowCheckedModeBanner: false, title: 'Flutter Navigation', theme: ThemeData(primarySwatch: Colors.blue), home: HomeScreen(), // Start at the HomeScreen
    );
  }
}
```

Step 3: Create Screens

1 **home_screen.dart (HomePage)**

```
import 'package:flutter/material.dart'; import 'about_screen.dart';
```

```
import 'contact_screen.dart'; import 'settings_screen.dart'; class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) { return Scaffold( appBar: AppBar(title: Text("Home Screen")), body: Center( child: Column( mainAxisAlignment: MainAxisAlignment.center, children: [
            ElevatedButton(
              onPressed: () {
                Navigator.push( context,
                    MaterialPageRoute(builder: (context) => AboutScreen()),
                ); }, child: Text("Go to About Screen"),
            ),
            ElevatedButton(
              onPressed: () {
                Navigator.push( context,
                    MaterialPageRoute(builder: (context) => ContactScreen()),
                );
                                                                                    },
child: Text("Go to Contact Screen"),
            ),
            ElevatedButton(
              onPressed: () {
                Navigator.push( context,
                    MaterialPageRoute(builder: (context) => SettingsScreen()),
                ); }, child: Text("Go to Settings"),
            ),
          ],
        ),
      ),
    );
  }
}
```

2 ☐ **about_screen.dart (About Page)**

```
import 'package:flutter/material.dart'; class AboutScreen extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) { return Scaffold( appBar:
    AppBar(title: Text("About Screen")), body: Center( child:
    Column( mainAxisAlignment: MainAxisAlignment.center,
    children: [
                                  Text("This is the About Screen", style: TextStyle(fontSize: 20)),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  Navigator.pop(context); // Go back to Home
                }, child: Text("Back to Home"),
              ),
            ],
),
        ),
    );
  }
}
```

3 ☐ **contact_screen.dart (Contact Page)** import 'package:flutter/material.dart'; class ContactScreen extends StatelessWidget {

```
  @override
  Widget build(BuildContext context) { return Scaffold( appBar: AppBar(title: Text("Contact Screen")), body: Center( child: Column( mainAxisAlignment: MainAxisAlignment.center,

children: [
              Text("This is the Contact Screen", style: TextStyle(fontSize: 20)),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  Navigator.pop(context); // Go back
                }, child: Text("Back to Home"),
              ),
            ],
          ),
        ),
    );
  }
}
```

4 ☐ **settings_screen.dart (Settings Page)** import 'package:flutter/material.dart'; class SettingsScreen extends StatelessWidget {

```
  @override
  Widget build(BuildContext context) { return Scaffold( appBar:
    AppBar(title: Text("Settings Screen")), body: Center( child:
    Column( mainAxisAlignment: MainAxisAlignment.center,
    children: [
                                  Text("This is the Settings Screen", style: TextStyle(fontSize: 20)),
            SizedBox(height: 20),
            ElevatedButton( onPressed: () {

                Navigator.pop(context); // Go back
              }, child: Text("Back to Home"),
            ),
          ],
        ),
      ),
    );
  }
}
```

Step 4: Run the App

flutter run

Navigation Methods Used

Method	Purpose
Navigator.push(context, MaterialPageRoute(builder: (context) => ScreenName()))	Navigate to a new screen
Navigator.pop(context)	Go back to the previous screen
Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) => ScreenName()))	Replace current screen (prevents going back)

Output: