**Full Stack Web Development 5 – 12 Code :**

**5) Practical – 5**

Here is a simple example of implementing MVC (Model-View-Controller) using AngularJS:

Model (book.js)

angular.module('bookModule', []).factory('Book', function() {

var books = [

{ id: 1, title: 'Book 1', author: 'Author 1' },

{ id: 2, title: 'Book 2', author: 'Author 2' },

{ id: 3, title: 'Book 3', author: 'Author 3' }

];

return {

getBooks: function() {

return books;

},

getBook: function(id) {

for (var i = 0; i < books.length; i++) {

if (books[i].id === id) {

return books[i];

}

}

return null;

}

};

});

Controller (bookController.js)

angular.module('bookModule', []).controller('BookController', function($scope, Book) {

$scope.books = Book.getBooks();

$scope.getBook = function(id) {

$scope.book = Book.getBook(id);

};

$scope.addBook = function() {

var newBook = {

id: $scope.books.length + 1,

title: $scope.title,

author: $scope.author

};

$scope.books.push(newBook);

$scope.title = '';

$scope.author = '';

};

$scope.deleteBook = function(id) {

for (var i = 0; i < $scope.books.length; i++) {

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

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

break;

}

}

};

});

View (index.html)

<!DOCTYPE html>

<html ng-app="bookModule">

<head>

<title>Book MVC</title>

<script src="(link unavailable)"></script>

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

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

</head>

<body ng-controller="BookController">

<h1>Book MVC</h1>

<ul>

<li ng-repeat="book in books">

{{ book.title }} ({{ book.author }})

<button ng-click="deleteBook((link unavailable))">Delete</button>

</li>

</ul>

<form>

<input type="text" ng-model="title" placeholder="Title">

<input type="text" ng-model="author" placeholder="Author">

<button ng-click="addBook()">Add Book</button>

</form>

<button ng-click="getBook(1)">Get Book</button>

<div ng-if="book">

<h2>{{ book.title }}</h2>

<p>{{ book.author }}</p>

</div>

</body>

</html>

Output

The output will be a simple web page displaying a list of books, with the ability to add, delete, and retrieve individual books.

**6) Practical – 6**

Here is a simple example of implementing data binding using AngularJS:

HTML (index.html)

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>Data Binding</title>

<script src="(link unavailable)"></script>

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

</head>

<body ng-controller="myController">

<h1>Data Binding</h1>

<input type="text" ng-model="name" placeholder="Enter your name">

<h2>Hello, {{ name }}!</h2>

</body>

</html>

JavaScript (script.js)

angular.module('myApp', []).controller('myController', function($scope) {

$scope.name = 'John Doe';

});

Output

The output will be a web page with an input field and a greeting message. When you type something in the input field, the greeting message will update automatically.

**Practical – 7**

Here is a simple example of implementing animation using AngularJS:

HTML (index.html) :

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>Animation</title>

<script src="(link unavailable)"></script>

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

<style>

.animate {

transition: all 1s;

}

.animate.ng-hide {

opacity: 0;

}

.animate.ng-show {

opacity: 1;

}

</style>

</head>

<body ng-controller="myController">

<h1>Animation</h1>

<button ng-click="toggle()">Toggle</button>

<div ng-show="visible" class="animate">

This is a animated div.

</div>

</body>

</html>

JavaScript (script.js)

angular.module('myApp', ['ngAnimate']).controller('myController', function($scope) {

$scope.visible = true;

$scope.toggle = function() {

$scope.visible = !$scope.visible;

};

});

Output

The output will be a web page with a button and a div. When you click the button, the div will fade in or out.

**Practical – 8**

Here is a simple example of implementing CRUD (Create, Read, Update, Delete) operations using AngularJS:

HTML (index.html) :

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>CRUD Operations</title>

<script src="(link unavailable)"></script>

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

</head>

<body ng-controller="myController">

<h1>CRUD Operations</h1>

<form>

<input type="text" ng-model="name" placeholder="Name">

<input type="text" ng-model="age" placeholder="Age">

<button ng-click="create()">Create</button>

</form>

<table>

<tr ng-repeat="person in people">

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

<td>{{ person.age }}</td>

<td>

<button ng-click="update(person)">Update</button>

<button ng-click="delete(person)">Delete</button>

</td>

</tr>

</table>

</body>

</html>

JavaScript (script.js) :

angular.module('myApp', []).controller('myController', function($scope) {

$scope.people = [

{ name: 'John Doe', age: 30 },

{ name: 'Jane Doe', age: 25 }

];

$scope.create = function() {

$scope.people.push({ name: $scope.name, age: $scope.age });

$scope.name = '';

$scope.age = '';

};

$scope.update = function(person) {

person.name = $scope.name;

person.age = $scope.age;

};

$scope.delete = function(person) {

$scope.people.splice($scope.people.indexOf(person), 1);

};

});

Output

The output will be a web page with a form to create new person, a table to display all people, and buttons to update and delete each person.

**Practical – 10**

Here is a simple example of implementing database connectivity using MongoDB in Java:

MongoDB Java Driver Dependency (pom.xml) :

<dependencies>

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>4.3.1</version>

</dependency>

</dependencies>

Java Code (MongoDBExample.java) :

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

public class MongoDBExample {

public static void main(String[] args) {

// Replace with your MongoDB connection string

String connectionString = "mongodb://localhost:27017";

// Create a MongoClient instance

MongoClient mongoClient = MongoClients.create(connectionString);

// Get a database instance

MongoDatabase database = mongoClient.getDatabase("mydatabase");

// Get a collection instance

MongoCollection<Document> collection = database.getCollection("mycollection");

// Insert a document

Document document = new Document("name", "John Doe").append("age", 30);

collection.insertOne(document);

// Find all documents

for (Document doc : collection.find()) {

System.out.println(doc.toJson());

}

// Close the MongoClient instance

mongoClient.close();

}

}

Output

The output will be the JSON representation of the documents in the collection.

{ "\_id" : { "$oid" : "631f14b8b1f25e77f2c4" }, "name" : "John Doe", "age" : 30 }

**Practical – 11**

Here is a simple example of implementing CRUD (Create, Read, Update, Delete) operations using MongoDB in Java:

MongoDB Java Driver Dependency (pom.xml) :

<dependencies>

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>4.3.1</version>

</dependency>

</dependencies>

Java Code (MongoDBExample.java) :

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.MongoCollection;

import org.bson.Document;

public class MongoDBExample {

public static void main(String[] args) {

// Replace with your MongoDB connection string

String connectionString = "mongodb://localhost:27017";

// Create a MongoClient instance

MongoClient mongoClient = MongoClients.create(connectionString);

// Get a database instance

MongoDatabase database = mongoClient.getDatabase("mydatabase");

// Get a collection instance

MongoCollection<Document> collection = database.getCollection("mycollection");

// Create

Document createDoc = new Document("name", "John Doe").append("age", 30);

collection.insertOne(createDoc);

// Read

for (Document doc : collection.find()) {

System.out.println(doc.toJson());

}

// Update

collection.updateOne(new Document("name", "John Doe"), new Document("$set", new Document("age", 31)));

// Delete

collection.deleteOne(new Document("name", "John Doe"));

// Close the MongoClient instance

mongoClient.close();

}

}

Output

The output will be the JSON representation of the documents in the collection after each operation.

{ "\_id" : { "$oid" : "631f14b8b1f25e77f2c4" }, "name" : "John Doe", "age" : 30 }

{ "\_id" : { "$oid" : "631f14b8b1f25e77f2c4" }, "name" : "John Doe", "age" : 31 }

**Practical – 12**

Here is a simple example of implementing Node.js with MongoDB:

Node.js Code (app.js) :

const express = require('express');

const app = express();

const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017';

const dbName = 'mydatabase';

app.use(express.json());

MongoClient.connect(url, function(err, client) {

if (err) {

console.log(err);

} else {

console.log('Connected to MongoDB');

const db = client.db(dbName);

const collection = db.collection('mycollection');

// Create

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

collection.insertOne(req.body, (err, result) => {

if (err) {

res.send(err);

} else {

res.send(result);

}

});

});

// Read

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

collection.find().toArray((err, result) => {

if (err) {

res.send(err);

} else {

res.send(result);

}

});

});

// Update

app.put('/update', (req, res) => {

collection.updateOne({ name: req.body.name }, { $set: { age: req.body.age } }, (err, result) => {

if (err) {

res.send(err);

} else {

res.send(result);

}

});

});

// Delete

app.delete('/delete', (req, res) => {

collection.deleteOne({ name: req.body.name }, (err, result) => {

if (err) {

res.send(err);

} else {

res.send(result);

}

});

});

}

});

app.listen(3000, () => {

console.log('Server started on port 3000');

});

Java Code (MongoDBExample.java) :

import java.io.IOException;

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

public class MongoDBExample {

public static void main(String[] args) throws IOException, InterruptedException {

HttpClient client = HttpClient.newHttpClient();

// Create

HttpRequest createRequest = HttpRequest.newBuilder()

.uri(URI.create("http://localhost:3000/create"))

.header("Content-Type", "application/json")

.POST(HttpRequest.BodyPublishers.ofString("{\"name\": \"John Doe\", \"age\": 30}"))

.build();

HttpResponse<String> createResponse = client.send(createRequest, HttpResponse.BodyHandlers.ofString());

System.out.println(createResponse.body());

// Read

HttpRequest readRequest = HttpRequest.newBuilder()

.uri(URI.create("http://localhost:3000/read"))

.GET()

.build();

HttpResponse<String> readResponse = client.send(readRequest, HttpResponse.BodyHandlers.ofString());

System.out.println(readResponse.body());

// Update

HttpRequest updateRequest = HttpRequest.newBuilder()

.uri(URI.create("http://localhost:3000/update"))

.header("Content-Type", "application/json")

.PUT(HttpRequest.BodyPublishers.ofString("{\"name\": \"John Doe\", \"age\": 31}"))

.build();

HttpResponse<String> updateResponse = client.send(updateRequest, HttpResponse.BodyHandlers.ofString());

System.out.println(updateResponse.body());

// Delete

HttpRequest deleteRequest = HttpRequest.newBuilder()

.uri(URI.create("http://localhost:3000/delete"))

.header("Content-Type", "application/json")

.DELETE(HttpRequest.BodyPublishers.ofString("{\"name\": \"John Doe\"}"))

.build();

HttpResponse<String> deleteResponse = client.send(deleteRequest, HttpResponse.BodyHandlers.ofString());

System.out.println(deleteResponse.body());

}

}

Output

The output will be the response from the Node.js server after each operation.

{ "\_id" : { "$oid" : "631f14b8b1f25e77f2c4" }, "name" : "John Doe", "age" : 30 }

[ { "\_id" : { "$oid" : "631f14b8b1f25e77f2c4" }, "name" : "John Doe", "age" : 30 } ]

{ "n" : 1, "nModified" : 1, "ok" : 1 }

{ "n" : 1, "ok" : 1 }