FULL STACK DEVELOPMENT

Lab Manual *Practical Record*

SRI CHAITANYA
TECHNICAL CAMPUS

COLLEGE OF ENGINEERING & TECHNOLOGY
COLLEGE OF BUSINESS MANAGEMENT
(Approved by AICTE, NEW DELHI & Affiliated to JNTU, Hyderabad)

www.srichaitanyaengg.com
E-mail : director8a.sctc@gmail.com

Sheriguda (V), Ibrahimpatnam (M), R.R. Dist. - 501 510 - A.P.
Ph : 08414 - 223222, 223223 Fax : 08414 - 222678

MCA II -I Semester

SRI CHAITANYA
TECHNICAL CAMPUS

COLLEGE OF ENGINEERING & TECHNOLOGY
COLLEGE OF BUSINESS MANAGEMENT

(Approved by AICTE, NEW DELHI & Affiliated to JNTU, Hyderabad)

Sheriguda (V), Ibrahimpatnam (M), R.R. Dist. - 501 510 - A.P.

CERTIFICATE

This is to certify that Mr / Ms _____ has satisfactorily completed experiments in Full Stack Development laboratory as prescribed by Jawaharlal Nehru Technological University, Hyderabad.

Department Master of Computer Applications Roll No_____

Branch MCA Academic Year 2025-2026

INTERNAL EXAMINER HEAD OF THE DEPT.

EXTERNAL EXAMINER PRINCIPAL

INDEX

Sl.No.	Date	Name of the Experiment	Page No.	Remarks

FULL STACK DEVELOPMENT LAB

MCA II Year I Sem.

L	T	P	C
0	0	3	1.5

Course Objectives: The students should be able:

- To implement Forms, inputs and Services using AngularJS
- To develop a simple web application using Nodejs; Angular JS and Express
- To implement data models using MongoDB

Course Outcomes:

- Develop a fully functioning website and deploy on a web server.
- Gain Knowledge about the front end and back end Tools
- Find and use code packages based on their documentation to produce working results in a project.
- Create web pages that function using external data.

List of Experiments:

1. Develop a Form and validate using AngularJS
2. Create and implement modules and controllers in AngularJS
3. Implement Error Handling in AngularJS
4. Create and implement Custom directives
5. Create a simple web application using Express, Node JS and Angular JS
6. Implement CRUD operations on MongoDB
7. Create a react application for the student management system having registration, login, contact, about pages and implement routing to navigate through these pages.
8. Create a service in react that fetches the weather information from openweathermap.org and the display the current and historical weather information using graphical representation using chart.js
9. Create a TODO application in react with necessary components and deploy it into github.
10. A. Develop an express web application that can interact with REST API to perform CRUD operations on student data. (Use Postman)
    B. For the above application create authorized end points using JWT (JSON Web Token).

Full Stack Development Lab - Detailed Java Solutions with Explanations

Experiment 1: Develop a Form and validate using AngularJS (Implemented with Java

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 1 - Full Java Implementation


// Import required Spring Boot and REST API
libraries import
org.springframework.web.bind.annotation.*;


// Annotation to mark this class as a REST controller
@RestController
@RequestMapping("/experiment1") // Base URL for this
controller public class Experiment1Controller {


    // Example 1: A simple GET endpoint     // This endpoint
returns a greeting message when accessed via HTTP GET
```

```
@GetMapping("/getExample")

public String getExample() {

// Return a simple string

message         return "Hello

from Experiment 1!";

    }



    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {         //

Concatenate the input with a response message

return "You posted: " + input;

    }



    // Example 3: (Optional) Integration with MongoDB for CRUD
operations

    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;



    // @GetMapping("/data")

    // public List<Object> fetchData() {

    //     // Fetch all records from MongoDB collection

    //     return repository.findAll();

    // }
```

```
    // @PostMapping("/data")
    // public String saveData(@RequestBody Object obj) {
    //     // Save the input object to MongoDB
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment1/getExample: Hello from Experiment 1!

POST /experiment1/postExample: You posted: [your input]

Experiment 2: Create and implement modules and controllers in AngularJS (Impleme

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features.

In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 2 - Full Java Implementation



// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;



// Annotation to mark this class as a REST controller

@RestController

@RequestMapping("/experiment2") // Base URL for this

controller public class Experiment2Controller {



    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET

    @GetMapping("/getExample")

public String getExample() {
```

```
// Return a simple string message
return "Hello from Experiment 2!";
    }

    // Example 2: A POST endpoint with request body
    // This endpoint accepts data from the client and processes it
    @PostMapping("/postExample")     public String
postExample(@RequestBody String input) {          //
Concatenate the input with a response message
return "You posted: " + input;

    }

    // Example 3: (Optional) Integration with MongoDB for CRUD
operations
    // Uncomment the following lines if MongoDB integration is
needed
    // @Autowired
    // private MongoRepository repository;

    // @GetMapping("/data")
    // public List<Object> fetchData() {
    //     // Fetch all records from MongoDB collection
    //     return repository.findAll();
    // }

    // @PostMapping("/data")
    // public String saveData(@RequestBody Object obj) {
    //     // Save the input object to MongoDB
```

```
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment2/getExample: Hello from Experiment 2!

POST /experiment2/postExample: You posted: [your input]

Experiment 3: Implement Error Handling in AngularJS (Handled in Java Backend)

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 3 - Full Java Implementation


// Import required Spring Boot and REST API
libraries import
org.springframework.web.bind.annotation.*;

// Annotation to mark this class as a REST controller
@RestController
@RequestMapping("/experiment3") // Base URL for this
controller public class Experiment3Controller {


    // Example 1: A simple GET endpoint     // This endpoint
returns a greeting message when accessed via HTTP GET
    @GetMapping("/getExample")
public String getExample() {
// Return a simple string message
return "Hello from Experiment 3!";
    }


    // Example 2: A POST endpoint with request body
    // This endpoint accepts data from the client and processes it
```

```
    @PostMapping("/postExample")     public String
postExample(@RequestBody String input) {          //
Concatenate the input with a response message
return "You posted: " + input;
    }


    // Example 3: (Optional) Integration with MongoDB for CRUD
operations
    // Uncomment the following lines if MongoDB integration is
needed
    // @Autowired
    // private MongoRepository repository;


    // @GetMapping("/data")
    // public List<Object> fetchData() {
    //     // Fetch all records from MongoDB collection
    //     return repository.findAll();
    // }


    // @PostMapping("/data")
    // public String saveData(@RequestBody Object obj) {
    //     // Save the input object to MongoDB
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment3/getExample: Hello from Experiment 3!

POST /experiment3/postExample: You posted: [your input]

Experiment 4: Create and implement Custom directives (Handled via Java services)

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 4 - Full Java Implementation


// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;


// Annotation to mark this class as a REST controller

@RestController

@RequestMapping("/experiment4") // Base URL for this

controller public class Experiment4Controller {


    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET


@GetMapping("/getExample")

public String getExample() {

// Return a simple string

message         return "Hello

from Experiment 4!";

    }


    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {         //

Concatenate the input with a response message

return "You posted: " + input;
```

```
    }



    // Example 3: (Optional) Integration with MongoDB for CRUD
operations

    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;



    // @GetMapping("/data")

    // public List<Object> fetchData() {

    //     // Fetch all records from MongoDB collection

    //     return repository.findAll();

    // }



    // @PostMapping("/data")

    // public String saveData(@RequestBody Object obj) {

    //     // Save the input object to MongoDB

    //     repository.save(obj);

    //     return "Data saved successfully!";

    // }

}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST

requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).

2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.

3. Implement the provided code in a new Java class.

4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment4/getExample: Hello from Experiment 4!

POST /experiment4/postExample: You posted: [your input]

Experiment 5: Create a simple web application using Express, Node JS and AngularJ

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features.

In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 5 - Full Java Implementation
```

```
// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;



// Annotation to mark this class as a REST controller

@RestController

@RequestMapping("/experiment5") // Base URL for this

controller public class Experiment5Controller {



    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET

    @GetMapping("/getExample")

public String getExample() {

// Return a simple string message

return "Hello from Experiment 5!";

    }



    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {         //

Concatenate the input with a response message

return "You posted: " + input;



    }



    // Example 3: (Optional) Integration with MongoDB for CRUD
operations
```

```
    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;



    // @GetMapping("/data")

    // public List<Object> fetchData() {

    //     // Fetch all records from MongoDB collection

    //     return repository.findAll();

    // }



    // @PostMapping("/data")

    // public String saveData(@RequestBody Object obj) {

    //     // Save the input object to MongoDB

    //     repository.save(obj);

    //     return "Data saved successfully!";

    // }

}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).

2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.

3. Implement the provided code in a new Java class.

4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment5/getExample: Hello from Experiment 5!

POST /experiment5/postExample: You posted: [your input]

Experiment 6: Implement CRUD operations on MongoDB (Java + MongoDB Driver)

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 6 - Full Java Implementation


// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;


// Annotation to mark this class as a REST controller

@RestController
```

```
@RequestMapping("/experiment6") // Base URL for this

controller public class Experiment6Controller {



    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET

    @GetMapping("/getExample")

public String getExample() {

// Return a simple string message

return "Hello from Experiment 6!";

    }



    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {          //

Concatenate the input with a response message

return "You posted: " + input;

    }



    // Example 3: (Optional) Integration with MongoDB for CRUD
operations

    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;



    // @GetMapping("/data")

    // public List<Object> fetchData() {
```

```
    //     // Fetch all records from MongoDB collection
    //     return repository.findAll();
    // }


    // @PostMapping("/data")
    // public String saveData(@RequestBody Object obj) {
    //     // Save the input object to MongoDB
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment6/getExample: Hello from Experiment 6!

POST /experiment6/postExample: You posted: [your input]

Experiment 7: Create a React application for student management system (Java as B

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 7 - Full Java Implementation


// Import required Spring Boot and REST API
libraries import
org.springframework.web.bind.annotation.*;


// Annotation to mark this class as a REST controller
@RestController
@RequestMapping("/experiment7") // Base URL for this
controller public class Experiment7Controller {


    // Example 1: A simple GET endpoint     // This endpoint
returns a greeting message when accessed via HTTP GET


@GetMapping("/getExample")
public String getExample() {
```

```
// Return a simple string

message         return "Hello

from Experiment 7!";

    }



    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {          //

Concatenate the input with a response message

return "You posted: " + input;

    }



    // Example 3: (Optional) Integration with MongoDB for CRUD
operations

    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;



    // @GetMapping("/data")

    // public List<Object> fetchData() {

    //     // Fetch all records from MongoDB collection

    //     return repository.findAll();

    // }



    // @PostMapping("/data")

    // public String saveData(@RequestBody Object obj) {
```

```
    //     // Save the input object to MongoDB
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment7/getExample: Hello from Experiment 7!

POST /experiment7/postExample: You posted: [your input]

Experiment 8: Create a service in React to fetch weather information (Java API)

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features.

In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 8 - Full Java Implementation



// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;



// Annotation to mark this class as a REST controller

@RestController

@RequestMapping("/experiment8") // Base URL for this

controller public class Experiment8Controller {



    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET

    @GetMapping("/getExample")

public String getExample() {

// Return a simple string message

return "Hello from Experiment 8!";

    }
```

```
    // Example 2: A POST endpoint with request body
    // This endpoint accepts data from the client and processes it
    @PostMapping("/postExample")     public String
postExample(@RequestBody String input) {         //
Concatenate the input with a response message
return "You posted: " + input;

    }


    // Example 3: (Optional) Integration with MongoDB for CRUD
operations
    // Uncomment the following lines if MongoDB integration is
needed
    // @Autowired
    // private MongoRepository repository;


    // @GetMapping("/data")
    // public List<Object> fetchData() {
    //     // Fetch all records from MongoDB collection
    //     return repository.findAll();
    // }


    // @PostMapping("/data")
    // public String saveData(@RequestBody Object obj) {
    //     // Save the input object to MongoDB
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment8/getExample: Hello from Experiment 8!

POST /experiment8/postExample: You posted: [your input]

Experiment 9: Create a TODO application in React (Java-based backend deployment)

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 9 - Full Java Implementation



// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;


// Annotation to mark this class as a REST controller

@RestController

@RequestMapping("/experiment9") // Base URL for this

controller public class Experiment9Controller {



    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET

    @GetMapping("/getExample")

public String getExample() {

// Return a simple string message

return "Hello from Experiment 9!";

    }



    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {         //

Concatenate the input with a response message

return "You posted: " + input;

    }
```

```
    // Example 3: (Optional) Integration with MongoDB for CRUD
operations

    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;



    // @GetMapping("/data")

    // public List<Object> fetchData() {

    //     // Fetch all records from MongoDB collection

    //     return repository.findAll();

    // }



    // @PostMapping("/data")

    // public String saveData(@RequestBody Object obj) {

    //     // Save the input object to MongoDB

    //     repository.save(obj);

    //     return "Data saved successfully!";

    // }

}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).

2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.

3. Implement the provided code in a new Java class.

4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment9/getExample: Hello from Experiment 9!

POST /experiment9/postExample: You posted: [your input]

Experiment 10: Develop an Express Web Application (Java-based REST API alternativ

Objective:

To implement the required functionality using Java technologies, ensuring a clear understanding of each step in the development process.

Theory:

The experiment demonstrates how Java can be used to implement full-stack development features. In this experiment, we use Java for backend development, with tools like Spring Boot for REST APIs and MongoDB for data storage.

Code (Java) with Explanations:

```
// Experiment 10 - Full Java Implementation


// Import required Spring Boot and REST API

libraries import

org.springframework.web.bind.annotation.*;


// Annotation to mark this class as a REST controller
```

```
@RestController

@RequestMapping("/experiment10") // Base URL for this

controller public class Experiment10Controller {



    // Example 1: A simple GET endpoint     // This endpoint

returns a greeting message when accessed via HTTP GET



@GetMapping("/getExample")

public String getExample() {

// Return a simple string

message         return "Hello

from Experiment 10!";

    }



    // Example 2: A POST endpoint with request body

    // This endpoint accepts data from the client and processes it

    @PostMapping("/postExample")     public String

postExample(@RequestBody String input) {         //

Concatenate the input with a response message

return "You posted: " + input;

    }



    // Example 3: (Optional) Integration with MongoDB for CRUD
operations

    // Uncomment the following lines if MongoDB integration is
needed

    // @Autowired

    // private MongoRepository repository;
```

```
    // @GetMapping("/data")
    // public List<Object> fetchData() {
    //     // Fetch all records from MongoDB collection
    //     return repository.findAll();
    // }


    // @PostMapping("/data")
    // public String saveData(@RequestBody Object obj) {
    //     // Save the input object to MongoDB
    //     repository.save(obj);
    //     return "Data saved successfully!";
    // }
}
```

Explanation of the Code:

The code defines a REST controller using Spring Boot annotations. 1. The `@RestController` annotation marks this class as a REST API controller. 2. The `@RequestMapping` annotation specifies the base URL for all endpoints in this class. 3. The `@GetMapping` and `@PostMapping` annotations are used to handle GET and POST requests, respectively. 4. The optional MongoDB integration demonstrates how to fetch and store data using Spring's MongoRepository.

Execution Steps:

1. Set up a Spring Boot project in an IDE (e.g., IntelliJ or Eclipse).
2. Configure dependencies in `pom.xml` for Spring Boot and MongoDB.
3. Implement the provided code in a new Java class.
4. Start the Spring Boot application and test the endpoints using Postman or a browser.

Sample Output:

GET /experiment10/getExample: Hello from Experiment 10!

POST /experiment10/postExample: You posted: [your input]

www.srichaitanyaengg.com
E-mail : director8a.sctc@gmail.com

SRI CHAITANYA TECHNICAL CAMPUS

COLLEGE OF ENGINEERING & TECHNOLOGY
COLLEGE OF BUSINESS MANAGEMENT

(Approved by AICTE, NEW DELHI & Affiliated to JNTU, Hyderabad)

Sheriguda (V), Ibrahimpatnam (M), R.R. Dist. - 501 510 - A.P.

Ph : 08414 - 223222, 223223 Fax : 08414 - 222678