

COLLEGE CODE :9222 COLLEGE NAME:

THENI KAMMAVAR SANGAM COLLEGE OF TECHNOLOGY DEPARTMENT: CSE

STUDENT NM ID : AA5E56A059C99C0AA0444E1DF554BB49

ROLL NO:23CS053/922223104052

DATE:26-10-2025

COMPLETED THE PROJECT NAMED AS PHASE 5 TECHNOLOGY PROJECT NAME:

JAVA REST CLIENT INTEGRATION SUBMITTED BY,

NAME: Veera Bhawineshwar M

MOBILE NO: 9789639346

# Project Demonstration & Documentation

## Project Title: Java Rest Client Integration

1. FINAL DEMO WALKTHROUGH SCRIPT

Good [morning/afternoon], everyone. Today, I’m going to walk you through our **Java REST Client Integration**, which demonstrates how our application communicates with an external RESTful API.

The main goal of this integration is to enable our Java application to **send and receive data** over HTTP using standard methods like **GET, POST, PUT, and DELETE**. We started by setting up our project with the necessary dependencies in pom.xml, including **Spring Boot Web** for REST operations and **Jackson** for JSON serialization and deserialization.

Next, we created a dedicated **REST Client Service** class that handles all HTTP communication. In this class, we use RestTemplate (or HttpClient) to build requests, set headers such as ContentType and Authorization, and process responses from the API.

During the demo, I will first show a **GET request**, which retrieves data from the API and maps it into Java objects. Then, I’ll demonstrate a **POST request**, where our Java client sends JSON data to the API to create a new record. Similarly, we can use **PUT** to update existing data and **DELETE** to remove records from the server.

We also implemented proper **error handling** to manage failed requests gracefully, including logging errors and validating API responses. Finally, I will run a few test requests to show real-time data flow between our Java client and the API, confirming that the integration works seamlessly.

In conclusion, this integration demonstrates how Java applications can efficiently interact with RESTful services, ensuring **scalability,**

**modularity, and robust communication** in modern software systems.

1. PROJECT REPORT & ARCHITECTURAL OVERVIEW
2. Project Title:

### Java REST Client Integration for [Specify Service/Use Case]

1. **Objective:**

The primary objective of this project is to develop a Javabased client application capable of consuming RESTful web services. The project aims to:

* + Demonstrate the integration of Java applications with REST APIs.
  + Handle HTTP requests (GET, POST, PUT, DELETE) programmatically.
  + Process JSON/XML responses and map them to Java objects.
  + Implement error handling, logging, and retry mechanisms for robust integration.

1. Scope:

The project will focus on:

* + Integration with a specific REST API (e.g., Weather API, Payment Gateway, Employee Management System).
  + Performing CRUD operations via REST endpoints.
  + Ensuring proper data serialization/deserialization using libraries like Jackson or Gson. Building a reusable Java client module for future integrations.

Technology stack

|  |  |  |
| --- | --- | --- |
| REST  Client Module | Jersey Client / Apache HttpClient / Spring WebClient | Jersey Client / Apache HttpClient / Spring WebClient |
| Data Model Layer | Jackson / Gson / JAXB | Acts as the bridge between the REST API and the Java application |
| Service Layer | SLF4J + Logback / java.util.logging | Provides reusable helper functions to other layers of the application |

# API Documentations & Proof Points

|  |  |  |
| --- | --- | --- |
| Category | Resource / Tool | Description |
| Client Type | RestClient | Modern, fluent API  for synchronous REST calls |
|  | RestTemplate | Legacy client, still |
|  |  | supported  for simple use cases |
|  | WebClient | Reactive, non-blocking client  for high concurrency |
|  | HTTP Interface | Declarative REST client using  annotated interfaces |

Tutorials

Baeldung

Comparison and usage of RestClient

vs RestTemplate

|  |  |  |
| --- | --- | --- |
|  | GeeksforGeeks | Overview and setup of RestClient |
| Testing | CodingTechRoom | REST API integration testing  in Java |
| API  Documentation | Swagger / OpenAPI | Auto-generates interactive API docs  from annotations |
|  | Springdoc OpenAPI | Integrates Swagger with Spring  Boot |

# Challenges and Solutions

|  |  |  |
| --- | --- | --- |
| Challenge | Description | Solution |
| Client Selection Confusion | Choosing between RestClient, RestTemplate,  WebClient, or HTTP Interface | Use RestClient for modern blocking calls; WebClient for reactive apps |
| Version Compatibility | RestClient requires Spring Framework  6.1+ | Upgrade to Spring Boot  3.2+ and Java 17 or higher |
| Compatibility | Spring Framework  6.1+ and Java 17+ | Upgrade to Spring Boot  3.2+ and Java 17 or higher |
| Error Handling | Poorly handled  HTTP errors can crash the app | Use .onStatus() or  .exchange() to handle errors gracefully |
| Testing REST Clients | Difficulty mocking external APIs | Use MockRestServiceServer  or WireMock for integration tests |

1. GITHUB README & SETUP GUIDE

Project Overview:

The **Java REST Client Integration Project** demonstrates how to connect a Java-based application with external RESTful APIs. It handles **HTTP communication**, **JSON**

### serialization/deserialization, authentication, and error management

in a modular and reusable way.

The client can perform CRUD operations (Create, Read, Update, Delete) on resources like users, products, or orders through REST endpoints.

It is designed with clean architecture principles — separating the **Client Layer**, **Data Model Layer**, **Service Layer**, and **Utility Layer** for better maintainability and scalability.

## Source code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Java REST Client Frontend</title>

<link rel="icon" href="img/data.png">

<style> body {

background-color: pink; font- family: 'Segoe UI', sans-serif;

display: flex; justify- content: center; align-items: center; flex-direction: column; padding: 2rem;

}

.container { background-color: #fff; padding: 2rem; border-radius: 10px; box-shadow: 0 0 10px rgba(0,0,0,0.1); max-width: 600px; width: 100%;

text-align: center;

}

h1, h2 { color: #333;

margin-bottom: 1rem;

}

form {

margin-bottom: 2rem;

}

input, textarea {

width: 100%; padding: 0.75rem; margin- bottom: 1rem; border: 1px solid #ccc;

border-radius: 5px;

}

button { padding: 0.75rem 1.5rem; background-color: #d63384;

color: white; border: none; border-radius: 5px; cursor: pointer;

}

.post {

border: 1px solid #ddd; padding: 1rem; margin-bottom: 1rem; border-radius: 5px; background-color: #ffe6f0;

text-align: left;

}

</style>

</head>

<body>

<div class="container">

<h1>Posts</h1>

<form id="postForm">

<h2>Create a New Post</h2>

<input type="text" id="title" placeholder="Title" required />

<textarea id="body" placeholder="Body" required></textarea>

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

</form>

<button onclick="loadPosts()">Load Posts</button>

<div id="posts"></div>

</div>

<script>

const API\_BASE = ['http://localhost:8080/api/posts';](http://localhost:8080/api/posts%27%3B)

async function loadPosts() { try {

const response = await fetch(API\_BASE);

if (!response.ok) throw new Error('Failed to fetch posts'); const posts = await response.json();

const container = document.getElementById('posts'); container.innerHTML = ''; posts.forEach(post => {

const div = document.createElement('div');

div.className = 'post'; div.innerHTML =

`<h3>${post.title}</h3><p>${post.body}</p>`; container.appendChild(div);

});

} catch (error) { console.error(error);

document.getElementById('posts').innerHTML = '<p>Error loading posts.</p>';

}

}

document.getElementById('postForm').addEventListener('submit', async (e) => {

e.preventDefault(); const title = document.getElementById('title').value.trim(); const body = document.getElementById('body').value.trim();

if (!title || !body) {

alert('Please fill in both fields.'); return;

}

try {

const response = await fetch(API\_BASE, { method: 'POST',

headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ title, body })

});

if (!response.ok) throw new Error('Failed to create post'); alert('Post created successfully!'); loadPosts(); } catch (error) { console.error(error);

alert('Error creating post.');

}

});

</script>

</body>

</html>

## FINAL SUBMISSION CHECKLIST

The following links represent the final deliverables for this project.

GitHub Repository Link:

<https://github.com/maheveera860-cloud/Veera-Bhawineshwar-M.git>

Deployed Website Link (Netlify): [https://java-rest-clien-tintegration.netlify.app](https://java-rest-clien-tintegration.netlify.app/)