RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world®

DEPARTMENT OF

INFORMATION SCIENCE AND ENGINEERING

Innovative Experiment
Report On

"API for Resource Optimization"

By

1RV23SIT15	Sumanth S N
1RV23SIT17	Vinayak Kumar Kamble
1RV23SIT18	Yashvanth P S

Under the Guidance of

Prof. Rashmi R
Assistant Professor
Dept. of ISE
RV College of Engineering®

Course Name: API Development and Integration Lab
Course Code: MIT438L

SEPT 2023 - 24

Table of Contents

Resource Optimization

INTRODUCTION

In the context of server and application management, resource optimization refers to the strategic allocation and utilization of computing resources—such as CPU, memory, storage, and network bandwidth—to maximize performance and efficiency. As demands on applications grow and environments scale, effective optimization ensures that resources are neither over-allocated, which can lead to wasted costs, nor under-allocated, which can result in poor performance or downtime. Techniques like load balancing, autoscaling, and monitoring can help dynamically manage resources to maintain optimal performance under varying workloads, reducing operational costs and improving user experience.

Resource optimization is especially critical in cloud and containerized environments where infrastructure is elastic but costs can scale rapidly with inefficient usage. Techniques such as horizontal and vertical scaling, resource throttling, and efficient caching strategies play a key role in balancing load and improving response times. Furthermore, modern tools for real-time monitoring and predictive analytics allow administrators to proactively detect bottlenecks and allocate resources accordingly, preventing performance degradation before it impacts users. Ultimately, effective resource optimization not only enhances application stability and scalability but also drives significant cost savings, making it a cornerstone of efficient IT operations.

SOFTWARE REQUIREMENTS WITH VERSION, INSTALLATION PROCEDURES

Following is the list of software requirements specified in order of installation:

Software Name	**Version**	**Installation link**	**Purpose**
Java Development Kit (JDK)	**11 or higher**	**https://www.oracle.com/java/technologies/downloads/#java11**	**Development environment for Java applications.**
Spring Boot	**2.6.7 or compatible**	**https://start.spring.io/**	**Framework for building RESTful APIs and microservices.**
Eclipse IDE	**2023-03 or higher**		**Integrated development environment (IDE) for Java development.**
Maven	**3.8.6 or higher**	**https://maven.apache.org/download.cgi**	**Build tool for managing dependencies and project lifecycle.**
Postman	**10.1 or higher**		**Tool for testing and interacting with APIs.**

SOURCE CODE LINK (GITHUB):

https://github.com/Yash-Raj-96/Resource-optimization.git

OVERVIEW OF THE ESSENTIAL ASPECTS OF THE API

The tables below provides a clear and concise overview of the essential aspects of the created APIs, making it easy for users to understand and reuse:

Section	**Details**
API Overview	**Name: Resource Optimizer API** **Version: 1.0** **Base URL: http://localhost:9090/api**
Authentication	**No authentication required for this version.**
Endpoints	**GET /cpu-usage** **Description: Analyze the cpu-usage of the app** **Request Example: POST /optimize/cpu** **Response Example: { "memory is optimized" }**
Error Handling	**400 Bad Request:: If the app is stopped/not running**
Rate Limiting	**Limit: 1000 requests per hour** **Handling: Returns 429 status code if exceeded**

Usage Examples	**Python Example:**
	import requests
	response = requests.get('https://api.example.com/v1/users/123',
	print(response.json())

Additional Resources	**API Documentation: Full Documentation** **Support: Contact Support**

DESCRIPTION ABOUT EACH MODULE

1]. TargetHealthController Class (API Endpoint)

The TargetHealthController class is responsible for monitoring, controlling, and optimizing the allocation of system resources such as CPU, memory, storage, and bandwidth. The class would encapsulate the logic to ensure that resources are efficiently used, adjusting allocations dynamically based on current demand or predefined thresholds. This class might interact with other components like load balancers, autoscaling mechanisms, or monitoring tools to collect metrics, analyze resource usage patterns, and take corrective actions.

2]. MetricsAnalyzerService Class

The MetricsAnalyzerService class is designed to collect, analyze, and interpret performance metrics of an application or system to provide insights that guide optimization strategies. This class plays a crucial role in ensuring efficient resource usage and application performance by monitoring various metrics such as CPU utilization, memory consumption, disk I/O, network bandwidth, and response times. Based on the analyzed data, it can help detect anomalies, bottlenecks, or inefficiencies that may require immediate corrective action.

3]. HTML Frontend

An HTML file designed to display metrics and provide an option to optimize would serve as a user interface (UI) for monitoring system performance in real time and triggering resource optimization actions. This file would present key metrics like CPU usage, memory consumption, disk I/O, and network traffic in visual formats such as graphs, charts, or tables. Alongside these visualizations, there would be an "Optimize" button or dropdown menu that allows users to manually initiate optimization actions, such as autoscaling, resource reallocation, or balancing workloads. The HTML page would likely interact with backend services, like the MetricsAnalyzerService and ResourceOptimizeController, through APIs to fetch live data and execute optimization processes based on user input.

4]. pom.xml (Maven Project Object Model)

The pom.xml file is the Maven configuration file for the project, detailing the project's dependencies, build configurations, and plugins. It ensures that essential libraries and tools, such as Spring Boot is included and correctly managed. This configuration file enables project build & dependency management, development and execution of the application.

IMPLEMENTATION DETAILS WITH TOOLS USED

1. Spring Boot Framework

- Tool Used: Spring Boot
- Purpose: Spring Boot was chosen as the framework for developing the backend of the application due to its ease of use and ability to create stand-alone, production-grade Spring-based applications with minimal configuration.

Implementation Details:

- The Spring Boot Starter Web dependency is used for setting up the web server and handling HTTP requests.
- The main application class is OptimizerApplication, which is the entry point for the Spring Boot application, and it uses the @SpringBootApplication annotation to enable component scanning and auto-configuration.
- The TargetHealthController class is responsible for exposing RESTful API endpoints to handle file uploads and process image data.

2. REST API

- Tool Used: Spring MVC
- Purpose: Spring MVC (Model-View-Controller) is used to implement REST APIs that allow users to upload images and retrieve the extracted text.

Implementation Details:

- The API endpoint /api/cpu-usage is defined using the @GetMapping annotation in the TargetHealthController class.
- It accepts file uploads through the @RequestParam annotation with the parameter MultipartFile file.
- The file is saved to a temporary directory, pre-processed, and passed to the service for process.

3. **Frontend for Metrics Usage**

- Tool Used: HTML/CSS
- Purpose: A simple web page to display metrics of app.

Implementation Details:

- An HTML form with the enctype="multipart/form-data" attribute is used to display results.
- The web page contains memory, cpu usage & option to optimize the resource of the app.
- Basic CSS is used to style the form, providing a user-friendly interface.

4. **Maven for Dependency Management**

- Tool Used: Maven
- Purpose: Maven is used to manage the dependencies and build the project efficiently.

Implementation Details:

- Dependencies like spring-boot-starter-web (for REST APIs) and tess4j (for OCR functionality) are defined in the pom.xml file.
- The Spring Boot Maven Plugin is used to package the application as a JAR file, making it easy to deploy.

WORKING PROCEDURE

The Optimizer application is designed to optimize memory, cpu usage & clean temp files of the server/app. The following outlines the step-by-step working procedure of the application:

1. 1. Backend Setup (API/Service Layer)

a. Metrics Collection Service:

- Implement a backend service (e.g., using Node.js, Python, or Java) that collects system and application metrics like CPU usage, memory usage, disk space, and other performance data.
- Integrate with monitoring tools or OS-level commands to gather real-time metrics.
- This service should expose REST APIs to fetch the latest metrics data.

b. Resource Optimization Service:

- Implement a service or controller (e.g., ResourceOptimizeController) that provides endpoints to optimize resources like CPU and memory.
- Define methods to:
  - Adjust resource allocations (e.g., scale up/down memory, CPU usage).
  - Free up memory by terminating unnecessary processes or clearing caches.

c. Temp File Cleanup Service:

- Implement a service to clean temporary files from the system. This could involve removing files from temp directories, clearing cache files, or performing disk cleanup tasks.
- Expose an API endpoint for this cleanup action.

d. API Endpoints:

- Example API endpoints:
  - /all – Retrieves system metrics (CPU, memory, disk usage, etc.).
  - /optimize/cpu – Triggers CPU optimization.
  - /optimize/memory – Triggers memory optimization.
  - /optimize/temp – Cleans up temporary files.

3HTML Structure

a. Basic Layout

Create an HTML file with sections for displaying metrics and optimization options.

Use HTML elements such as <div>, <table>, and <button> to structure the content.

b. Metrics Display

- Use <div> or <section> elements to display metrics like CPU usage, memory usage, and disk space. These can be presented in simple text or styled with CSS for better visualization.

c. Optimization Controls

- Include buttons for optimization actions like "Optimize CPU," "Optimize Memory," and "Clean Temp Files." These buttons would be static, and their functionality would depend on server-side logic triggered by form submissions.

4. CSS Styling

a. Basic Styles

- Style the layout with CSS to make it visually appealing. Use properties like background-color, padding, margin, border, and font-family to enhance the appearance.

b. Layout and Spacing

- Ensure proper spacing between sections and elements using margin and padding. Align text and buttons using CSS flexbox or grid layout if needed.

c. Button Styles

- Style buttons to make them stand out and provide clear visual feedback.

5. Backend Interaction

- Form Submissions: The forms in the HTML submit POST requests to the respective backend endpoints for CPU optimization, memory optimization, and temp file cleanup.
- Static Data: Since there’s no JavaScript to fetch live data, you can rely on server-side code to inject static metrics values into the HTML, or use server-side templating to update metrics before rendering the page.

.

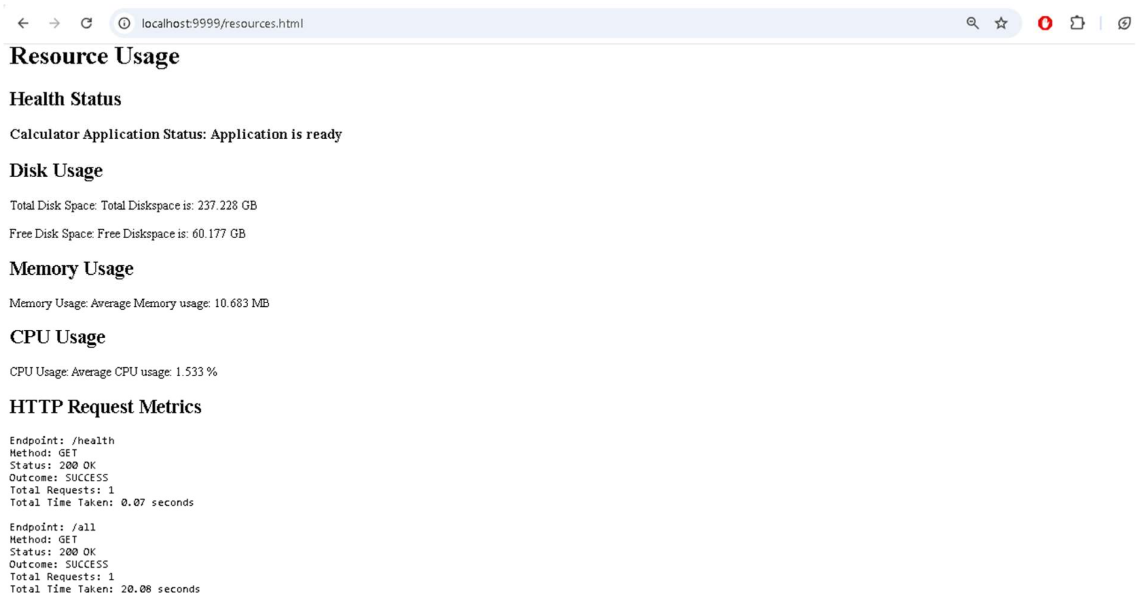
6. Backend Error Handling

a. Error Detection and Response

- On the server side, ensure that any operations (like metric retrieval, optimization actions, or file cleanup) include error handling logic.

- When an error occurs, return an appropriate HTTP status code and an error message in the response. For example, use status codes like 400 Bad Request or 500 Internal Server Error.
  b. Error Messages
- Include detailed error messages in the response body that can be displayed to the user. Ensure the messages are clear and helpful.

SCREENSHOTS


localhost:9999/resources.html

Resource Usage

Health Status

Calculator Application Status: Application is ready

Disk Usage

Total Disk Space: Total Diskspace is: 237.228 GB

Free Disk Space: Free Diskspace is: 60.177 GB

Memory Usage

Memory Usage: Average Memory usage: 10.683 MB

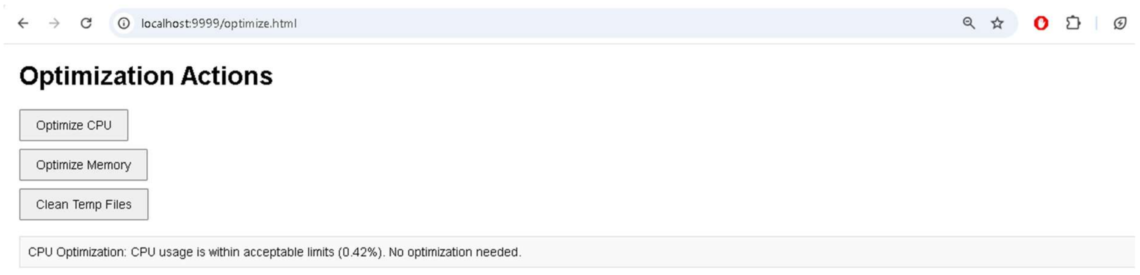
CPU Usage

CPU Usage: Average CPU usage: 1.533 %

HTTP Request Metrics

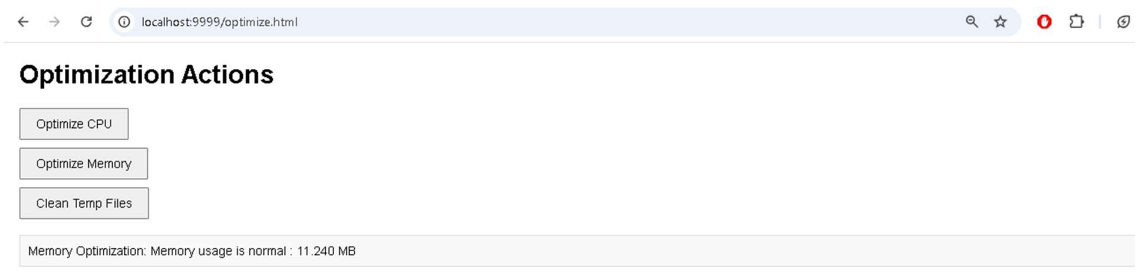
```
Endpoint: /health
Method: GET
Status: 200 OK
Outcome: SUCCESS
Total Requests: 1
Total Time Taken: 0.07 seconds

Endpoint: /all
Method: GET
Status: 200 OK
Outcome: SUCCESS
Total Requests: 1
Total Time Taken: 20.08 seconds
```

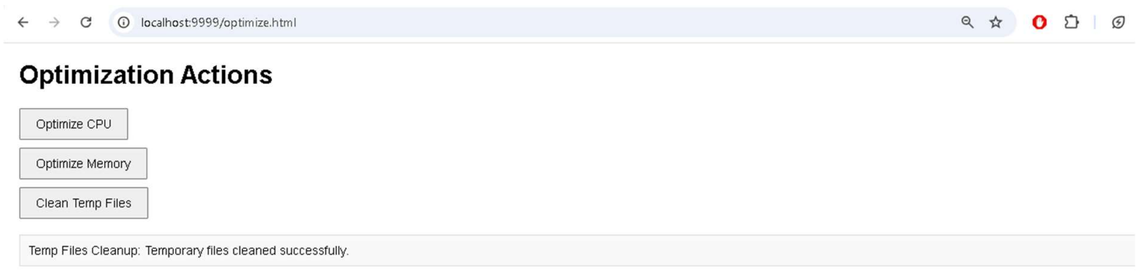

localhost:9999/optimize.html

Optimization Actions

CPU Optimization: CPU usage is within acceptable limits (0.42%). No optimization needed.



```
Memory Optimization: Memory usage is normal : 11.240 MB
```



```
Temp Files Cleanup: Temporary files cleaned successfully.
```

CONCLUSION

In conclusion, the Resource Optimizer project represents a critical advancement in managing and optimizing system resources efficiently. By integrating a comprehensive suite of tools and interfaces, this project addresses key challenges in resource management, including real-time monitoring, dynamic optimization, and effective cleanup of temporary files. The use of backend services to collect and analyze metrics, combined with a user-friendly front-end interface, empowers users to make informed decisions about resource allocation and system performance.

The project's architecture, featuring a blend of server-side logic and static HTML/CSS, ensures that users can monitor system metrics and perform optimization actions seamlessly. Although it operates without JavaScript, the solution remains robust by leveraging server-side rendering to display real-time data and handle error reporting. This approach not only simplifies the user experience but also enhances reliability by providing clear feedback and actionable insights directly through the web interface.

Overall, the Resource Optimizer project highlights the importance of effective resource management in today's dynamic computing environments. By focusing on optimizing CPU, memory, and disk usage, and offering tools for immediate action, the project delivers significant benefits in performance and cost efficiency. As system demands continue to evolve, this project lays a solid foundation for future enhancements, ensuring that resource optimization remains a pivotal component of modern IT operations.