**Logo

Description automatically generated**

Course: **Guided Research I**

Title: **Scalability experiment of microservice architecture on an online bookstore**

**application**

**Midterm Report**

Student: **Tural Mehtiyev**

Instructors: Dr. Stephen Kaisler, Dr. Jamal Hasanov

**Table of Contents**

[**Project Objectives** 3](#_Toc142330996)

[**Project Status** 3](#_Toc142330997)

[**Midterm Deliverables** 4](#_Toc142330998)

[**Application Idea & System Design** 4](#_Toc142330999)

[**Architecture Description Diagram** 4](#_Toc142331000)

[**Use Case Diagram** 4](#_Toc142331001)

[**Order Flow Diagram** 5](#_Toc142331002)

[**Application Development, Testing & Documentations** 6](#_Toc142331003)

[**Product Catalog Microservice** 6](#_Toc142331004)

[**Shopping Cart Microservice** 15](#_Toc142331005)

[**Order Management Microservice** 30](#_Toc142331006)

[**Formulation of Test Case Scenarios** 35](#_Toc142331007)

[**Data Collection** 38](#_Toc142331008)

[**Summary Report & Aggregate Report:** 39](#_Toc142331009)

[**Aggregate Graph:** 40](#_Toc142331010)

[**Response Time Graph:** 40](#_Toc142331011)

[**Visual Description | Effects of Ramp-Up Steps Count on Response Time** 41](#_Toc142331012)

[**Statistical Analysis | Effects of Threads Count on Response Rate** 43](#_Toc142331013)

[**Application of the Central Limit Theorem** 45](#_Toc142331014)

[**Hypothesis Testing:** 48](#_Toc142331015)

# **Project Objectives**

A screenshot of a computer

Description automatically generated

Figure 1

# **Project Status**

A diagram of a technical approach

Description automatically generated

Figure 2

# **Midterm Deliverables**

## **Application Idea & System Design**

### **Architecture Description Diagram**

A diagram of a computer

Description automatically generated

Figure 3

### **Use Case Diagram**

A diagram of a company

Description automatically generated

Figure 4

### **Order Flow Diagram**

A screenshot of a computer program

Description automatically generated

Figure 5

## **Application Development, Testing & Documentations**

### **Product Catalog Microservice**

A screenshot of a computer

Description automatically generated

Figure 6

Link to Swagger UI: <https://product_catalog-1-f3543029.deta.app/docs>

**Description:** This microservice is responsible for managing the product inventory, details, and availability.

**API Endpoints**

* GET /products: Get all products.

**Response**

A close-up of a computer screen

Description automatically generated

Figure 7

**A screenshot of a computer

Description automatically generated**

Figure 8

* GET /products/{product\_id}: Returns the product with the specified ID.

A screenshot of a computer

Description automatically generated

Figure 9

A screenshot of a computer

Description automatically generated

Figure 10

* POST /products: Creates a new product in the catalog.

A screenshot of a computer code

Description automatically generated

Figure 11

A screenshot of a computer

Description automatically generated

Figure 12

* PUT /products/{product\_id}: Updates the specified product in the catalog.

A screenshot of a computer

Description automatically generated

Figure 13

A screenshot of a computer

Description automatically generated

Figure 14

* DELETE /products/{product\_id}: Deletes the specified product from the catalog.

A computer screen shot of a computer code

Description automatically generated

Figure 15

**Dependencies:** This service is independent and does not have any dependencies.

**Database Connection:** This service uses PostgreSQL as its database which is deployed on Amazon RDS. It connects to the PostgreSQL instance using the psycopg2 library. The database connection parameters are specified within the application code, including the database name, user, password, host, and port.

Upon application start-up, the service automatically connects to the database using the provided credentials and host information. The code then checks for the existence of necessary tables (products) and creates them if they do not exist.

**Database Tables**

**A screenshot of a computer

Description automatically generated**

Figure 16

**Service Deployment**

This service is designed to be deployed to Deta Space, a cloud-based, scalable environment for running FastAPI applications. Step-by-step instructions on how to do this can be found on the read me file in GitHub repository:  
  
<https://github.com/ADA-GWU/guidedresearchproject-tmehtiyev2019/tree/main/app/product_catalog_microservice>

### **Shopping Cart Microservice**

**Description:** This microservice handles the management of customer shopping carts.

A screenshot of a computer

Description automatically generated

Figure 17

**API Endpoints**

* GET /products: Get all shopping carts.

A screenshot of a computer

Description automatically generated

Figure 18

A screenshot of a computer

Description automatically generated

Figure 19

* GET /cart\_items: Get all items in all shopping carts.

A screenshot of a computer

Description automatically generated

Figure 20

A screenshot of a computer

Description automatically generated

Figure 21

GET /carts/{customer\_id}: Returns the active cart for the customer with the specified ID.

A screenshot of a computer screen

Description automatically generated

Figure 22

A screenshot of a computer

Description automatically generated

Figure 23

* POST /carts/{customer\_id}: Adds an item to the active cart of the customer with the specified ID.

A screenshot of a computer

Description automatically generated

Figure 24

A screenshot of a computer

Description automatically generated

Figure 25

* DELETE /carts/{customer\_id}/{product\_id}: Deletes the specified product from the cart of the customer with the specified ID.

A screenshot of a computer

Description automatically generated

Figure 26

A screenshot of a computer

Description automatically generated

Figure 27

* PUT /carts/{customer\_id}/{product\_id}: Updates the quantity of a specific item in the active cart of the customer with the specified ID.

A screenshot of a computer

Description automatically generated

Figure 28

PUT /carts/{customer\_id}: Updates the status of the active cart of the customer with the specified ID.

A screenshot of a computer

Description automatically generated

Figure 29

A screenshot of a computer

Description automatically generated

Figure 30

**Dependencies**

This service is dependent on the Product Catalog Microservice for product data.

The Shopping Cart Microservice is dependent on the Product Catalog Microservice for information about the products. Here's what this dependency involves:

When a customer wants to add a product to their shopping cart, the Shopping Cart Microservice needs to check the Product Catalog Microservice to ensure the product exists and to get its details. This means there is a call from the Shopping Cart Microservice to the Product Catalog Microservice.

Similarly, when a customer updates the quantity of a product in their cart, the Shopping Cart Microservice has to ensure that enough quantity of the product is available. This information is retrieved from the Product Catalog Microservice.

These dependencies are managed through API calls between the two services as described in the codes above.

**Database Connection**

This service uses PostgreSQL as its database which is deployed on Amazon RDS. It connects to the PostgreSQL instance using the psycopg2 library. The database connection parameters are specified within the application code, including the database name, user, password, host, and port.

Upon application start-up, the service automatically connects to the database using the provided credentials and host information. The code then checks for the existence of necessary tables (carts and cart\_items) and creates them if they do not exist.

**Database Tables**

A screenshot of a computer program

Description automatically generated

Figure 31

**Service Deployment**

This service is designed to be deployed to Deta Space, a cloud-based, scalable environment for running FastAPI applications. The step-by-step instructions on how to do this are given in the Read Me file in GitHub repository:

<https://github.com/ADA-GWU/guidedresearchproject-tmehtiyev2019/tree/main/app/shopping_cart_microservice#readme>

### **Order Management Microservice**

Order Management Microservice

**Description:** This microservice manages the processing of customer orders.

A screenshot of a computer

Description automatically generated

Figure 32

**API Endpoints**

* GET /orders: Get all orders.

A white background with black text

Description automatically generated

Figure 33

A screenshot of a computer

Description automatically generated

* GET /orders/{customer\_id}: Fetch all orders for the specified customer.

A screenshot of a computer program

Description automatically generated

Figure 34

A screenshot of a computer

Description automatically generated

Figure 35

* POST /orders/{customer\_id}: Creates a new order for the specified customer.

A screenshot of a computer program

Description automatically generated

Figure 36

**Dependencies**

This service is dependent on two other services:

Product Catalog Service: This service is used to retrieve product details and update inventory levels when an order is placed. The order management service makes GET and PUT requests to this service.

Shopping Cart Service: This service is used to retrieve the customer's shopping cart when placing an order. The order management service makes GET and PUT requests to this service.

**Database Connection**

This service uses PostgreSQL as its database which is deployed on Amazon RDS. It connects to the PostgreSQL instance using the psycopg2 library. The database connection parameters are specified within the application code, including the database name, user, password, host, and port.

Upon application start-up, the service automatically connects to the database using the provided credentials and host information. The code then checks for the existence of necessary tables (orders and order\_items) and creates them if they do not exist.

**Database Tables**

There are two tables created within this service:

A screenshot of a computer

Description automatically generated

Figure 37

## **Formulation of Test Case Scenarios**

Utilizing Apache JMeter, I have designed test scenarios that would provide valuable insights into the **effects of concurrent users on the application's response rate**. The primary objective was to understand how the application behaves under different loads and the corresponding response times.

**Scenario 1:**

Objective: Understand the application's behavior with a light load.

Start Threads Count: **10 users**

This signifies that 10 users will start simultaneously to send requests to the application.

Initial Delay: 0 seconds

The test begins without any initial delay.

Startup Time: 10 seconds

This parameter ensures that all users become active over a span of 10 seconds.

A screenshot of a computer

Description automatically generated

Figure 38

**Scenario 2:**

Objective: Assess the application's capability to manage increased load.

Start Threads Count: **50 users**

Initial Delay: 0 seconds

Startup Time: 10 seconds

Hold Load Time: 60 seconds

Shutdown Time: 10 seconds

A screenshot of a computer

Description automatically generated

Figure 39

**Scenario 3:**

Objective: Examine the application's robustness under a heavy user load.

Start Threads Count: **100 users**

Initial Delay: 0 seconds

Startup Time: 10 seconds

Hold Load Time: 60 seconds

Shutdown Time: 10 seconds

A screenshot of a computer

Description automatically generated

Figure 40

## **Data Collection**

After the formulation and execution of the scenarios, Apache JMeter's listeners were employed to capture a wide array of data points. During my testing phase, I have employed many JMeter listeners that were instrumental in both data collection and visualization. Below I give descriptive information about some of them:  
  
**View Results Tree:**

Detailed view of all request-response pairs, essential for debugging purposes. I have collected several data (**Sample time, Latency, Connect Time, Bytes sent/received, Success/error status**) about each request.

A screenshot of a computer program

Description automatically generated

Figure 41

### **Summary Report & Aggregate Report:**

Tabulated comprehensive statistics, including Sample count, Average response time, Median response time, 90th percentile response time, Minimum and Maximum response times, Error percentage, Throughput (requests per second), Received KB/sec & Sent KB/sec, providing a holistic overview of performance.

A screenshot of a computer

Description automatically generated

Figure 42

### **Aggregate Graph:**

A visual representation of key metrics like Average, Median, 90th percentile, and Min/Max response times, Throughput, Standard deviation, providing graphical insights for quick trend analysis.

A screenshot of a graph

Description automatically generated

Figure 43

### **Response Time Graph:**

Graphical portrayal of Response time against time or sample number, Deviation, Throughput, Median, Average, and 90th percentile values, aiding in visual detection of patterns or anomalies in response time.

A screenshot of a graph

Description automatically generated

Figure 44

## **Visual Description | Effects of Ramp-Up Steps Count on Response Time**

A graph and chart with numbers

Description automatically generated with medium confidence

Figure 45

A graph and chart with numbers

Description automatically generated with medium confidence

Figure 46

A graph of steps and a line

Description automatically generated with medium confidence

Figure 47

## **Statistical Analysis | Effects of Threads Count on Response Rate**

With the refined data, I have applied various statistical techniques to extract meaning from the numbers. By leveraging the **Central Limit Theorem**, **hypothesis testing**, I have gained a deeper understanding of the application's performance metrics (mainly response rate) under different user loads.

Below are the details of the response rate distribution for each test scenario:

A graph showing different colored lines

Description automatically generated

Figure 48

A graph of a bar graph

Description automatically generated with medium confidence

Figure 49

A diagram of a box plot

Description automatically generated

Figure 50

### **Application of the Central Limit Theorem**

**The Central Limit Theorem (CLT):**

The Central Limit Theorem is a statistical principle stating that, given a sufficiently large sample size, the distribution of the sample means of independent and identically distributed random variables will be approximately normal, irrespective of the original distribution of the variables.

**Why Did I Apply the Central Limit Theorem?**

* **Normality Assumption**: Many statistical techniques and tests assume the data to be normally distributed. If the original data is not normal, applying the CLT helps meet this assumption by working with the distribution of sample means instead.
* **Statistical Robustness**: By working with a normally distributed dataset (the distribution of sample means), my hypothesis tests and statistical inferences become more reliable and robust.
* **Simplification**: The normal distribution is well-understood, and its properties are widely used in statistics. By ensuring that my dataset adheres to a normal distribution (via the CLT), analysis and interpretation become more straightforward.

**Details from the Application:**

Sample Size: In the sampling process, the sample size (number of observations in each sample) is set as 50.

Number of Samples: I've collected 1,000 samples from each scenario. This means that I've drawn 1,000 separate samples, each of 50 observations, and then calculated their means.

Random Sampling: I've drawn random samples from the original data, ensuring that each sample is drawn independently.

The application of the Central Limit Theorem allowed me to make statistical inferences using the normal distribution, which has desirable properties. By applying the CLT, I'm aiming to leverage these properties to make more reliable decisions and interpretations based on my JMeter test results.

Below you can find the distribution for each test scenario after the application of CLT.  
  
A diagram of a normal distribution

Description automatically generated

Figure 51

A graph of a person with red and green color

Description automatically generated

Figure 52

A diagram of a box diagram

Description automatically generated

Figure 53

**Hypothesis Testing:**  
  
A screenshot of a computer

Description automatically generated

Figure 54