GPU Warranty Claims Process

IS213 Enterprise Solution Development

[G6-T6]

Project

Gu Yingqi

Gui Jia Qing

Jonathan Hsienzheng I'anson-Holton

Tan Wei Jie

Tan Wen Hang Darrell

Wong Jia Ying

Introduction

Our team's overall business scenario revolves around managing Graphics Processing Unit (GPU) warranty claims. The process begins when users submit a warranty claim and ends with them either obtaining a new or repaired GPU in return or a refund.

The different scenarios covered are as follows:

1. Customer send warranty claim & warranty validation: This initiates the warranty claiming process with the customer filling in a form and their GPU going through checks to verify validation.
2. Shipping GPU to service centre (Claimee POV): This scenario occurs when the GPU is covered under warranty and the customer fills up a different form to enter their shipping address.
3. Servicer updates status (Received/Not Covered/Repaired): This occurs on the servicer side for them to update on the status of each request.
4. Not repairable, 1:1 replacement available: Users are updated via email and inventory is updated.
5. Not repairable, no 1:1 replacement available, no alternative available
6. Not repairable, no 1:1 replacement available, alternative available
7. Shipping GPU back to claimee

User Scenarios

[User Scenario 1 : Customer Send Warranty Claim & Warranty Validation]

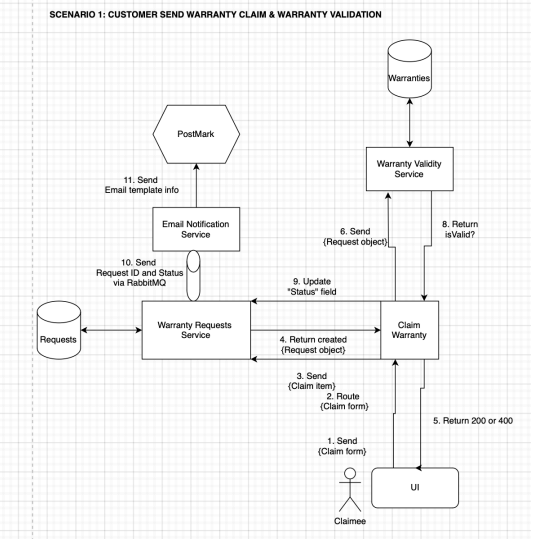


Figure 1: Warranty Claim & Validation

Figure 1 [Scenario 1]

1. A claimant submits a warranty claim form through a User Interface (UI) it routes to the Claim Warranty Complex microservice.
2. The Claim Warranty service receives the claim item data and sends it the warranty request
3. The Warranty Requests Service creates a request object based on the claim received and stores it in the Requests database.
4. The Claim Warranty service returns a response to the claimant, which could be a 200 HTTP status code for success or a 400 for failure.
5. The Warranty Validity Service retrieves warranty details from the Warranties database and checks the validity of the claim.

Warranty Validation:

6. The Claim Warranty Complex service sends a request object to the Warranty Validity Service to validate the claim.
7. If the claim is validated, the Warranty Requests Service updates the "Status" field of the request object in the Requests database.
8. It then sends the Object via AMQP to rabbitmq broker which will be consumed by the Email Microservice.
9. Depending on the “Status”, the Email Notification Service prepares an email to notify the claimant and is sent using PostMark.

External Services

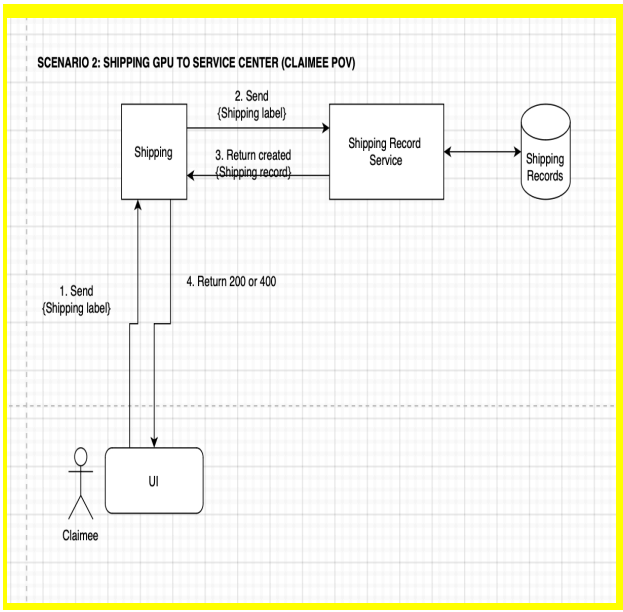
Service Name	Description of the functionality	Link to external web pages that describe the detailed inputs/output
PostMark	Email API allows Admin to send emails by making HTTP POST requests with JSON	*https://github.com/Stranger6667/postmarker/?tab=readme-ov-file*

	formatted messages to update claimee on the warranty request status	

Beyond the Labs

1. SpringBoot

[User Scenario 2: Claimee submit shipping label and Admin approve shipping label]
[User Scenario 7: Servicer sends GPU back to claimee]



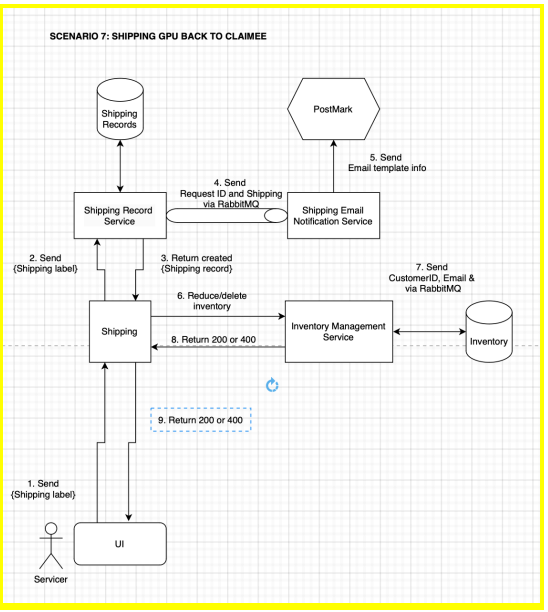


Figure 2: Shipping GPU to service centre

Figure 3: Shipping GPU back to Claimee

Claimee submits a shipping label to allow GPU to ship to him. Admin approves shipping and email will be sent to Claimee.

Figure 2 [Scenario 2]

1. Claimant UI triggers the Shipping GPU to Service Center process. The UI sends a Shipping label to initiate the shipping process.
2. Shipping label is routed to the Shipping microservice.
3. The Shipping microservice sends the Shipping label to the Shipping Record Service.
4. The Shipping Record Service creates a new shipping record and returns the created Shipping_record to the Shipping microservice.
5. Upon successfully creating the shipping record, the Shipping microservice sends a success response back 200 HTTP status code to the Claimant UI. If the process fails at any point, a 400 HTTP status code is returned to indicate an error.
6. The Claimant UI receives the HTTP status code, indicating whether the shipping record was successfully created (200) or if there was an error (400).

Figure 3 [Scenario 7]

1. Claimant UI triggers the Shipping GPU to Claimant process. The UI sends a request with shipping details to the Shipping Microservice to initiate the shipping process.
2. The Shipping microservice sends the Shipping Details to the Shipping Record Service. This includes information such as the claimant's address and the shipping label.
3. The Shipping Record Service creates a new shipping record and returns the created Shipping Record to the Shipping microservice. The record includes details like tracking number and expected delivery date.
4. Upon successfully creating the shipping record, the Shipping microservice sends a success response 200 HTTP status code to the Claimant UI. If the process fails at any point, a 400 HTTP status code is returned to indicate an error.
5. The Claimant UI receives the HTTP status code, indicating whether the shipping record was successfully created (200) or if there was an error (400).
6. Concurrently, the Shipping microservice sends a message with the Shipping Record ID and Status via RabbitMQ to the Email Notification Service. This message triggers the sending of a shipment confirmation to the claimant.

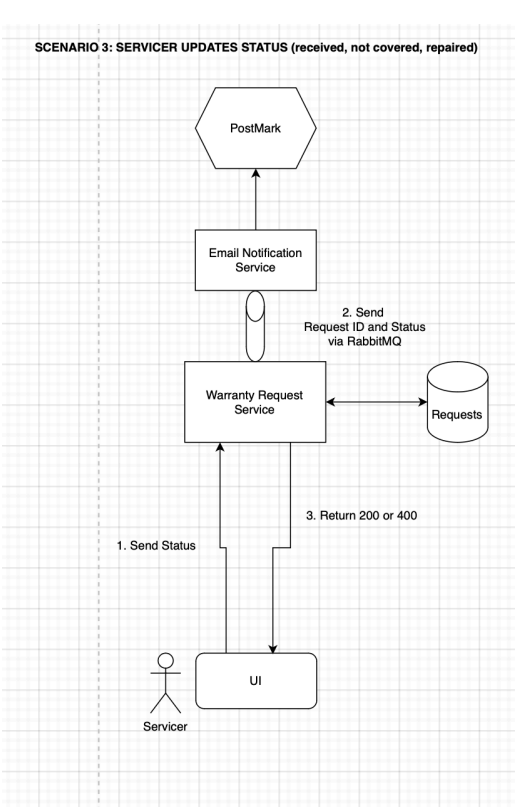
7. The Email Notification Service, upon receiving the message, uses the details to send an email through PostMark to the claimant, confirming the shipment has been processed and providing the tracking details.
8. Additionally, the Shipping microservice updates the Inventory Management Service, signalling that the GPU has been shipped and prompting an update in inventory, typically reducing the stock count.
9. The Inventory Management Service updates the inventory records accordingly and confirms the update, returning a 200 HTTP status code if successful or a 400 HTTP status code if there's an issue with updating the inventory.

External Services [Scenario 2 and 7]

Service Name	Description of the functionality	Link to external web pages that describe the detailed inputs/outputs
PostMark	Email API allows Admin to send emails by making HTTP POST requests with JSON formatted messages to inform Claimee that the GPU is being shipped out.	*https://github.com/Stranger6667/postmarker/?tab=readme-ov-file*

[User Scenario 3: Servicer updates status (Received, Not Covered, Repaired)]

[User Scenario 4: Not Repairable, 1:1 Identical Replacement Available]



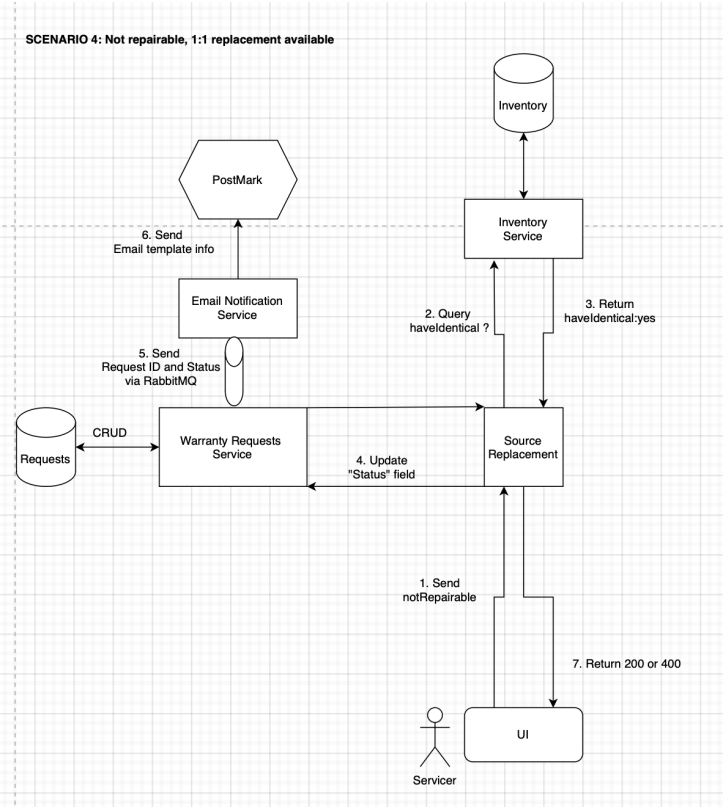


Figure 4: Servicer Updates Status (received, not covered, repaired) | **Figure 5: GPU not repairable, 1:1 identical replacement available**

Figure 4 [Scenario 3]

1. Servicer UI initiates a status update. The servicer updates the status of a warranty request (e.g., received, not covered, repaired) in the UI.
2. The UI sends the updated status to the Warranty Request Service.
3. The Warranty Request Service processes the status update. It updates the status of the warranty request in its database. The service then sends a confirmation back..
4. Response is then returned to the UI. If the Warranty Request Service successfully processes the status update, it returns a 200 HTTP status code. If there's an error, it returns a 400 HTTP status code.
5. The UI receives the response from the microservice. The servicer sees a confirmation message if the status was successfully updated or an error message if it was not.
6. Concurrently, the Warranty Request Service sends a message with the updated status. This message is sent to the Email Notification Service via RabbitMQ with the warranty request ID and the new status.

7. The Email Notification Service receives the message. Upon receiving the message, the service generates an email with the template information including the warranty request ID and the updated status.
8. The Email Notification Service sends the email through PostMark. The email is sent to the customer to notify them of the status update of their warranty request.

External Services

Service Name	Description of the functionality	Link to external web pages that describe the detailed inputs/output
PostMark	Email API allows Admin to send emails by making HTTP POST requests with JSON, keeping the user updated on the claim status	*https://github.com/Stranger6667/postmarker/?tab=readme-ov-file*

Beyond the Labs

N.A

Figure 5 [Scenario 4]

1. Servicer UI triggers the Replacement Process. The servicer determines that a GPU is not repairable and is eligible for a 1:1 replacement. The UI sends a replacement request with the GPU details..
2. The request is then sent to the Source Replacement microservice. The microservice is responsible for managing replacement logistics.
3. The Source Replacement microservice queries the Inventory Service to check if an identical GPU is available. This ensures that a 1:1 replacement can be sourced.
4. The Inventory Service checks the inventory and returns a response indicating if an identical item is available. If an identical GPU is available, the response will be affirmative.
5. Upon receiving confirmation, the Source Replacement microservice updates the "Status" field in the Warranty Requests Service. This update reflects that a replacement item has been sourced and is ready for dispatch.
6. The Warranty Requests Service sends the Request ID and updated Status via RabbitMQ to the Email Notification Service. The status includes details that a replacement GPU has been sourced.
7. The Email Notification Service, upon receiving the message, prepares an email to notify the claimant. The email includes information about the replacement availability and expected shipping details.
8. The Email Notification Service sends the email through PostMark to the claimant. The email confirms the replacement and may include a tracking number or additional instructions for receiving the replacement GPU.
9. The Servicer UI receives a response from the Source Replacement service. The response includes an HTTP status code: 200 if the process was successful and the replacement is confirmed, or 400 if there was an error or if a replacement could not be sourced.

External Services

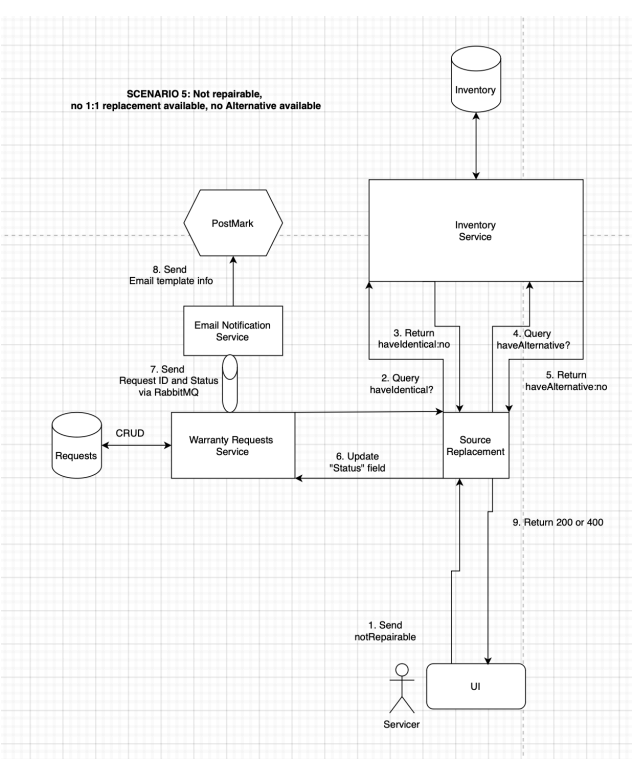
Service Name	Description of the functionality	Link to external web pages that describe the detailed inputs/output
PostMark	Email API allows Admin to send emails by making HTTP POST requests with JSON, keeping the user updated on the claim status	*https://github.com/Stranger6667/postmarker/?tab=readme-ov-file*

Beyond the Labs

1. SpringBoot

[User Scenario 5: Not Repairable, No 1:1 Replacement, No Alternative Available]

[User Scenario 6: Not Repairable, No 1:1 Replacement, Alternative Available]



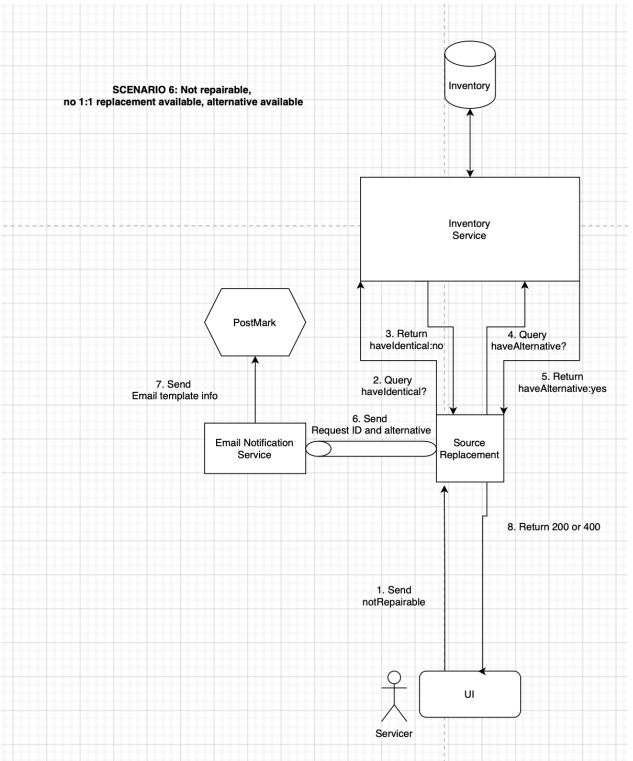


Figure 6: Not repairable, no 1:1 replacement, no alternative | Figure 7: Not repairable, no 1:1 replacement, alternative available

Figure 6 [Scenario 5]

1. Servicer UI initiates a non-refundable process. The servicer determines that the item is not repairable, no 1:1 replacement or alternative item is available. The UI sends this information to the Source Replacement Service.
2. The Source Replacement Service then routes the non-refundable item information to the Warranty Requests Service. The Warranty Requests Service is responsible for managing such warranty requests.
3. The Warranty Requests Service queries the Inventory Service to confirm if an identical item is available for replacement. The query checks for the availability of a 1:1 replacement.
4. The Inventory Service responds that no identical item is available. The response 'haveIdentical:no' indicates that there is no identical item in stock for a 1:1 replacement.
5. The Warranty Requests Service queries the Inventory Service for an alternative item. Since a 1:1 replacement isn't available, it checks for a possible alternative.
6. The Inventory Service responds that no alternative item is available. The response 'haveAlternative:no' indicates that there is no alternative available in the inventory.
7. The Warranty Requests Service updates the warranty request "Status" field. The status is updated to reflect that no replacement is available.
8. Upon updating the status, the Warranty Requests Service sends the Request ID and updated Status via RabbitMQ to the Email Notification Service. This includes the information that there is no replacement available.
9. The Email Notification Service receives the message and sends an email through PostMark to the customer. The email informs the customer that their item is not refundable, and no replacement or alternative is available.
10. A 200 HTTP status code to the Servicer UI, if the warranty status update and email notification were successfully processed, a 200 status code is sent. Otherwise, a 400 status code is sent to indicate an error.
11. The Servicer UI receives the HTTP status code. This lets the servicer know the outcome of the non-refundable process.

External Services

Service Name	Description of the functionality	Link to external web pages that describe the detailed inputs/output
PostMark	Email API allows Admin to send emails by making HTTP POST requests with JSON, keeping the user updated on the claim status	*https://github.com/Stranger6667/postmarker/?tab=readme-ov-file*

Beyond-The-Labs

Spring Boot

Figure 7 [Scenario 6]

1. Servicer UI initiates a non-refundable and replacement process. The servicer identifies the item is not repairable and sends a request to check for alternative replacements.
2. The request is routed to the Inventory Service. This is done through an HTTP request, specifying that the item is not repairable and a 1:1 replacement is not available.
3. The Inventory Service queries its records. It checks if an identical item is available, which is needed before considering alternatives.
4. The Inventory Service returns 'haveIdentical:no'. This indicates that no identical replacement is available in the inventory.
5. The Inventory Service then queries for an alternative item. It searches its records for a different item that could serve as a suitable replacement.
6. The Inventory Service returns 'haveAlternative:yes'. This confirms that while an identical replacement isn't available, there is an alternative that can be offered.
7. The Source Replacement Service receives the alternative item's details. It includes information necessary to propose the alternative to the customer, such as item specifications and availability.
8. The Source Replacement Service sends the Request ID and details of the alternative item via RabbitMQ to the Email Notification Service. This message triggers the process to inform the customer about the alternative.
9. The Email Notification Service receives the details. It prepares an email with information about the alternative item using an email template.
10. The Email Notification Service sends the email through PostMark to the customer. The email informs the customer about the non-refundable status of their original item and provides details of the alternative replacement available.
11. A response is then returned back to the UI. A 200 HTTP status code is returned if the alternative item process was successful. Otherwise, a 400 status code indicates there was an error in the process.
12. The Servicer UI receives the HTTP status code from the Source Replacement Microservice. This informs the servicer of the successful initiation of the alternative replacement process or if there was an error.

External Services

Service Name	Description of the functionality	Link to external web pages that describe the detailed inputs/output
PostMark	Email API allows Admin to send emails by making HTTP POST requests with JSON, keeping the user updated on the claim status	*https://github.com/Stranger6667/postmarker/?tab=readme-ov-file*

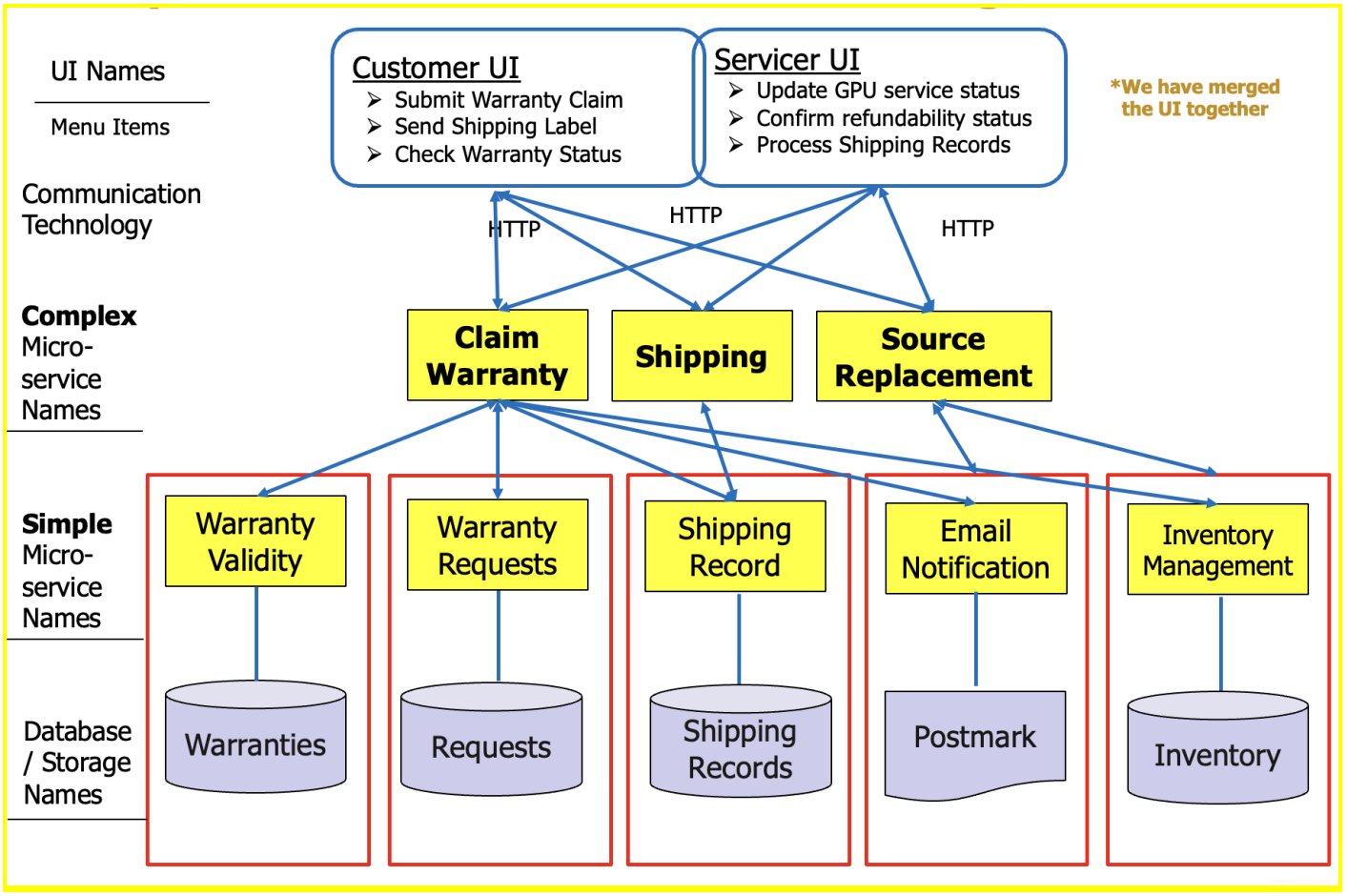
Beyond the Labs

SpringBoot Microservice : Warranty Request Service connecting to and processing data from requests db using Spring Data JPA

END OF REPORT

Appendix - Technical Overview Diagram(s) or SOA Layer Diagram(s)

SOA Layer Diagram: GPU Warranty Claims Process



Appendix - API Docs

(The appendix is not counted towards the page limit. It is not explicitly graded, but will be used to verify against your solution as described in the report if required.)

Source Replacement: HTTP-based API:

GET **/healthcheck** – Checks the health of the service to ensure it is running correctly.	
Parameters	None
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"message": "Service is running!"
	}

POST **/requests/{requestId}/status/notRepairable** - Processes a replacement request for a not repairable item by checking for an identical replacement or offering an alternative.

Parameters	requestId
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value if 1:1 available
	{
	"status": "success",
	"message": "Replacement request successfully processed."
	}
	
	Example Value if alternative offered
	{
	"status": "success",
	"message": "Alternative replacement offered."
	}
	
	Example Value if no replacement available
	{
	"status": "success",
	"message": "No alternative replacement available. Refund pending."
	}

Functionality – Send email to claimee	
Exchange Name	Warranty_service
Exchange Type	Topic
Queue Name	OfferAlternative
Binding Key	Offer.alternative

Body of incoming message	Code: 200 – Successful operation
	
	message = {
	"request_Id" : request_Id,
	"model_Id" : model_Id,
	"claimee" : claimee,
	"email" : email
	}

Warranty Request Service: HTTP-based API:

POST **/requests** – Creates a new request in the system	
Parameters	None
Responses	Content type – application/json

Request Body	{
	"model_Id": "string",
	"unit_Id": "string",
	"model_Type": "string",
	"claimee": "string",
	"email": "string",
	"description": "string"
	}

Code - 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"message": "Request added with ID: [requestId]"
	}

Code – 500	Description – Internal server error
	
	Example Value
	
	{
	"code": 500,
	"message": "Error while adding Request: [errorMessage]"
	}

PATCH **/requests/{requestId}/status** – Updates the status of an existing request.

Parameters	requestId
Responses	Content type – application/json
Code – 200	Description – successful operation Example Value { "code": 200, "message": "Request status updated successfully"

	}

Code – 500	Description – Internal server error
	
	Example Value
	
	{
	"code": 500,
	"message": "Error updating request status: [errorMessage]"
	}

GET **/requests/{requestId}** – Retrieves a specific request by its ID.	
Parameters	requestId
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"data": {
	"itemId": 1,
	"SerialNumber": "123",
	"Status": "Available"
	}
	}

Code – 500	Description – Internal server error

GET **/requests/all** - Retrieves a list of all requests in the system	
Parameters	None
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"data": [
	{
	"itemId": 1,
	"SerialNumber": "123",
	"Status": "Available"
	}
	]
	}

Code – 500	Description – Internal server error

DELETE **/requests/{requestId}** – Deletes a specific request by its ID.	
Parameters	requestId
Responses	Content type – application/json
Code – 204	Description – No Content
Code – 500	Description – Internal server error

Shipping: HTTP-based API:

GET **/api/shipping/shippingrecords/list** - Retrieves a paginated list of shipping records with optional filtering.	
Parameters	· page (query, integer): Page number of the records to retrieve. · per_page (query, integer): Number of records per page. · ShippingInID (query, integer): Identifier for filtering based on shipping in ID. · order_by (query, string): Attribute name to order the records by. · order_direction (query, string): Direction of sorting, either asc or desc.
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"data": [
	{
	"shippingRecordId": 1,
	"ShippingInID": "1",
	"ReceivedDateTime": "2023-03-25T14:30:00Z",
	"Remarks": "Delivered"
	}
	]
	}

Code – 404	Description – No shipping records found matching the criteria.
	
	Example Value
	
	{
	"code": 404,
	"message": "No shipping records found."
	}

GET **/api/shipping/shippingrecords/all** - Retrieves all shipping records.

Parameters	None
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"data": [
	{
	"shippingRecordId": 2,
	"ShippingInID": "2",
	"ReceivedDateTime": "2023-03-26T10:20:00Z",
	"Remarks": "1"
	}
	]
	}

Code – 404	Description – No shipping records available.
	
	Example Value
	
	{
	"code": 404,
	"message": "No shipping records available."
	}

Functionality – Send email to claimee	
Exchange Name	Warranty_service
Exchange Type	Topic
Queue Name	EMAIL
Binding Key	#

Body of incoming message	Code: 200 – Successful operation
	
	Example Value
	
	sendMqEmail({
	"message":{
	"sender":"darrell.tan.2022@scis.smu.edu.sg",
	"recipient":"weijie.tan.2022@scis.smu.edu.sg",
	"subject":"GPU Shipping Notification",
	"html_body":the_html_body
	}
	});

Inventory: HTTP-based API

GET **/api/inventory/inventory/all** - Retrieves all inventory items.	
Parameters	None
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"data": [
	{
	"itemId": 1,
	"SerialNumber": "123",
	"Status": "Available"
	}
	]
	}

Code – 404	Description – No inventory items found. Example Value { "code": 404, "message": "No inventory items found."

	}
GET **/api/inventory/inventory/list** - Retrieves a paginated list of inventory items with optional filtering.	
Parameters	page (query, integer): Page number of the items to retrieve. per_page (query, integer): Number of items per page. SerialNumber (query, string): Identifier for filtering based on serial number. order_by (query, string): Attribute name to order the items by. order_direction (query, string): Direction of sorting, either asc or desc.
Responses	Content type – application/json

Code – 200	Description – successful operation
	
	Example Value
	
	{
	"code": 200,
	"data": [
	{
	"itemId": 2,
	"SerialNumber": "124",
	"Status": "Checked Out"
	}
	]
	}

Code – 404	Description – No inventory items found matching the criteria.
	
	Example Value
	
	{
	"code": 404,
	"message": "No inventory items found matching the criteria."
	}

POST **/api/mq/send_message** - Sends a message to the configured message queue.

Parameters	body (body, raw JSON): Raw JSON payload to be sent.
Responses	Content type – application/json

Code – 200	Description – Message sent successfully
	
	Example Value
	
	{
	"code": 200,
	"message": "Message sent successfully"
	}

Code – 404	Description – Queue not found or unavailable. Example Value { "code": 404, "message": "Queue not found or unavailable."

	}
GET **/api/inventory/health** - Performs a health check on the inventory service.	
Parameters	None
Responses	Content type – application/json

Code – 200	Description – Service is healthy
	
	Example Value
	
	{
	"code": 200,
	"status": "Healthy"
	}

Code – 404	Description – Service unavailable or endpoint not found.
	
	Example Value
	
	{
	"code": 404,
	"message": "Service unavailable or endpoint not found."
	}

Warranty Validation: HTTP-based API

GET **/validate/<serial_number>/<claim_date> -** Return if the GPU model is valid or not

Parameters	claim_date claim_date (required): A datetime.date object indicating the date that the claim is submitted, has the format of %Y-%m-%d, e.g. 2023-01-01
Responses	Content type – application/json

Code – 200	- message (str): A message describing the status of the warranty.
	
	- status (str): A status indicating the status of the warranty.
	
	Example:
	
	`Request: GET /validate/<serial_number>?claim_date=2023-03-21`
	
	{
	
	"message": "Valid warranty",
	
	"status": "valid"
	
	}

Code – 400	Bad Request
	
	Example:
	
	`Request: GET /validate/<serial_number>?claim_date=2028-03-21`
	
	{
	
	"message": "Expired warranty",
	
	"status": "expired"
	
	}
	
	{
	
	"message": "Invalid serial number",
	
	"status": "invalid"
	
	}

: AMQP-based API

For the alternative design using AMQP request-and-reply

Email: AMQP-based API:

: AMQP-based API:

Appendix - Technical Contribution Table

E.g. coding of feature X, deployment, configure database, etc.

Name	**Contributions**
Jon	• Drawing of Scenario Microservice Interaction Diagram • Warranty Requests Microservice in Java Springboot using Spring JPA with MySQL db • Source replacement Orchestrator • Frontend-backend integration
Wei Jie	• Docker: Implemented Docker for consistent development and production environments, facilitating microservices deployment. • Microservices Architecture: Developed key microservices, with a primary focus on the Shipping Microservice for processing and tracking orders efficiently. Additional contributions included the Email Service for automated customer notifications and the Inventory Management Service for real-time inventory management. • Frontend and Backend Integration: Utilised JavaScript for dynamic frontend interactions. • Database Management: Configured and optimised a MySQL database, designing schemas for data storage and retrieval. • Messaging System with RabbitMQ: Integrated RabbitMQ using Pika in Python Flask for robust asynchronous messaging. • Testing and Documentation: Employed Postman for API testing to guarantee all microservices met functional and performance requirements.
Yingqi	• Warranty Validation microservice with MySQL db • Warranty claim orchestrator • Frontend-backend integration • Dockerised and Integrated entire project
Darrell Tan	• Setting up and testing of the Email Microservice using external application PostMark • Crafting Email template for the Email microservice • Setting up of RabbitMQ • Creating dockerfile for ease of containerisation • Assist with setting up Publisher for other Microservices so that the data is consistent for consumers
Wong Jia Ying	• Implementation of Front-end using HTML/CSS/Vue/Bootstrap • Integration of backend with frontend
Gui Jia Qing	• Setting up of the Refund Microservice

	● Creating dockerfile for containerisation ● Drawing of Scenario Microservice Interaction Diagram