

**PAWSystem**

**IS213 Enterprise Solution Development (AY2019/20 Term 2)**

***G7T3***

**Project Assignment Report**

*Agnes Lam*

*Chantel Er*

*Kong Yu Jie*

*Lee Pei Yi*

*Liu Ming Miao*

*Nicolas Wijaya*

Table of Contents

[**1.** **Introduction** 3](#_Toc37371428)

[**2.** **Technical Overview Diagram** 3](#_Toc37371429)

[**3.** **User Scenarios** 4](#_Toc37371430)

[**3.1.** **User Scenario 1 – Customer Manages Booking** 4](#_Toc37371431)

[**3.1.1.** **Steps Explanation for Scenario 1** 4](#_Toc37371432)

[**3.1.2.** **Microservices** 4](#_Toc37371433)

[**3.2.** **User Scenario 2 – Service Provider manage booking** 4](#_Toc37371434)

[**3.2.1.** **Steps Explanation for Scenario 2** 5](#_Toc37371435)

[**3.2.2.** **Microservices** 5](#_Toc37371436)

[**3.3.** **User Scenario 3 – Customer Creates Review** 5](#_Toc37371437)

[**3.3.1.** **Steps Explanation for Scenario 3** 5](#_Toc37371438)

[**3.3.2.** **Microservices** 6](#_Toc37371439)

[**3.4.** **User Scenario 4 – Customer makes Payment** 6](#_Toc37371440)

[**3.4.1.** **Steps Explanation for Scenario 4** 6](#_Toc37371441)

[**3.4.2.** **Microservices** 7](#_Toc37371442)

[**3.4.3.** **Beyond the Labs** 7](#_Toc37371443)

[**3.5.** **User Scenario 5 – Service Provider Acknowledges Payment** 7](#_Toc37371444)

[**3.5.1.** **Steps Explanation for Scenario 5** 7](#_Toc37371445)

[**3.5.2.** **Microservices** 8](#_Toc37371446)

[**4.** **References** 8](#_Toc37371447)

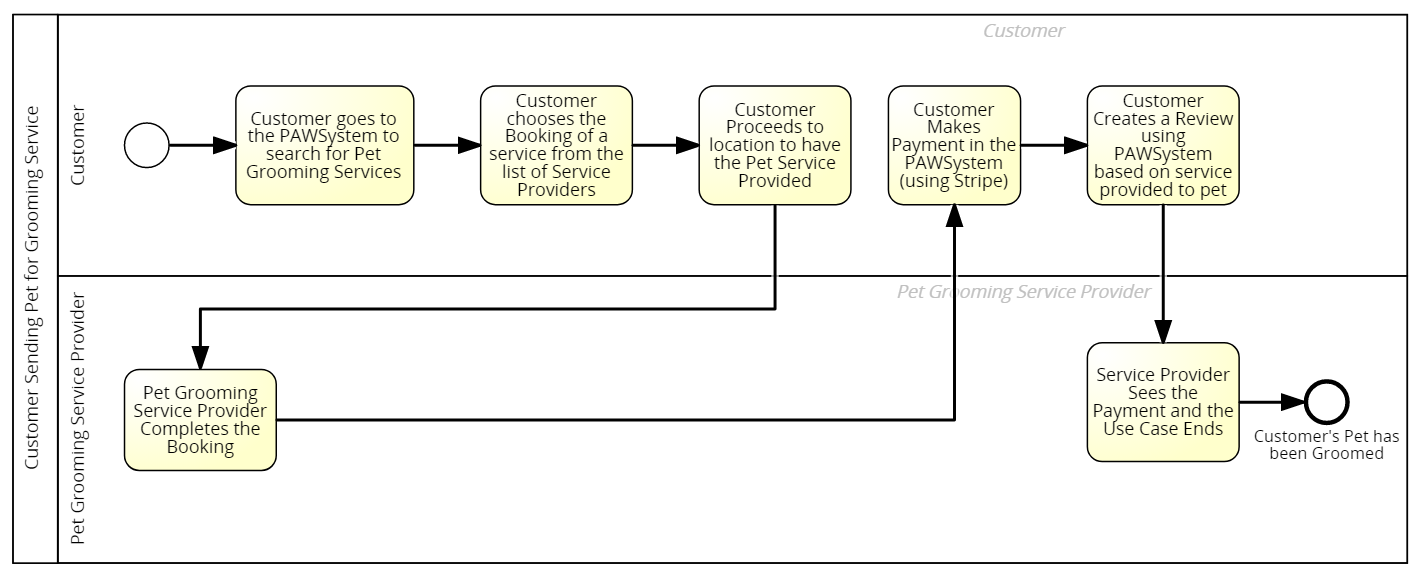
[**5.** **Appendices** 8](#_Toc37371448)

# **Introduction**

Our team has created a Pet Grooming System called the PAWSystem. PawS System aims to be a one-stop platform for all fragmented pet’s services app in the market. It provides the various functions not limited to the following.

1. Booking Services
2. Payment Service
3. Manage Provider Services

A customer’s journey to using this system to book and proceed to the grooming services will mostly be in 7 steps. It is documented into the workflow as shown below.



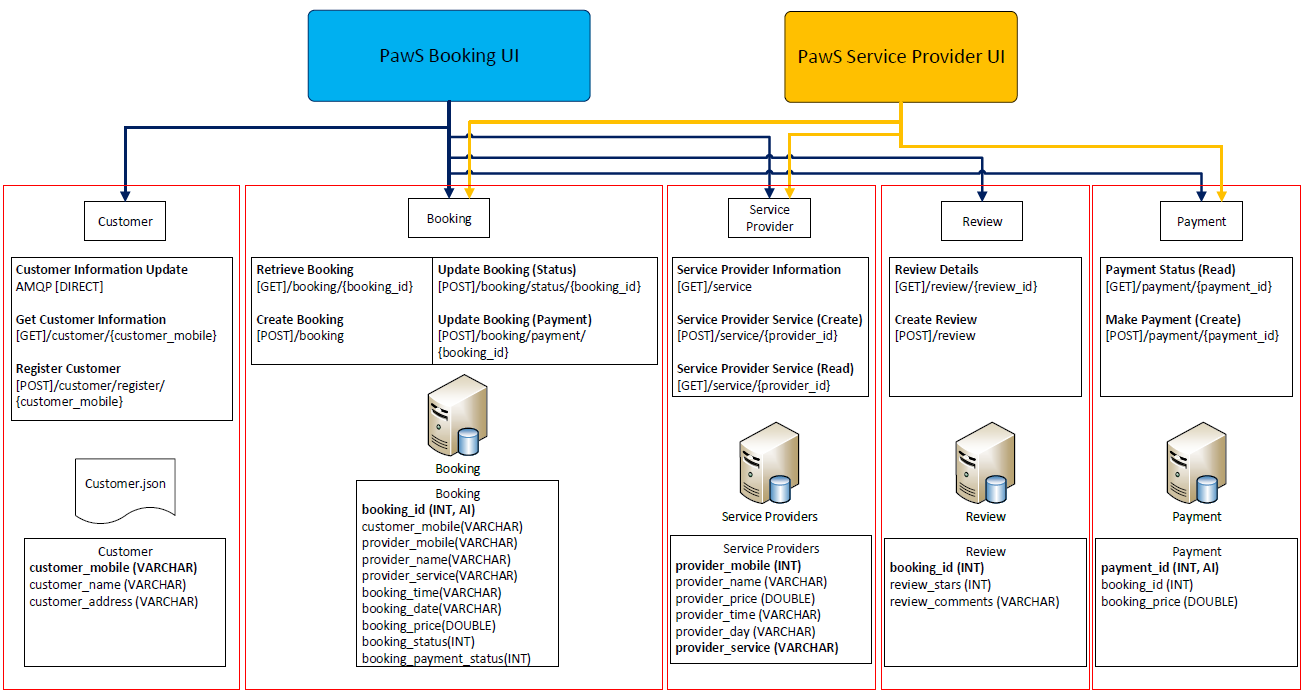
There are two main actors who will be using this scenario, which will then be separated to two User Interfaces.

1. Customer (with pet)
2. Service Provider

For the PAWSystem, our team has implemented 5 user scenarios which consists, but not limited to the following scenarios.

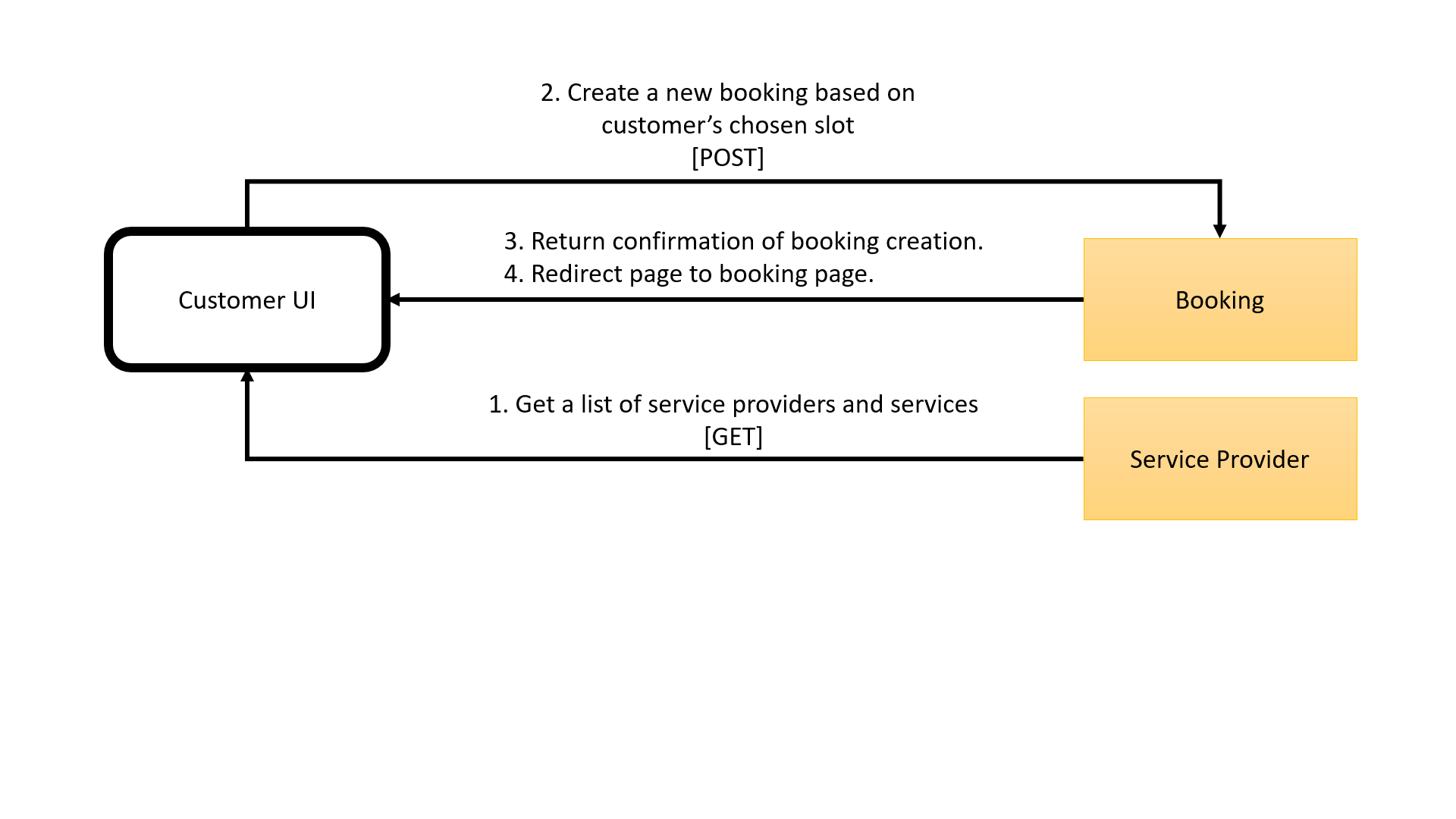
1. Customer manages Booking
2. Service Provider manages Booking
3. Customer creates Review
4. Customer makes Payment
5. Service Provider acknowledges Payment

# **Technical Overview Diagram**



# **User Scenarios**

## **User Scenario 1 – Customer Manages Booking**



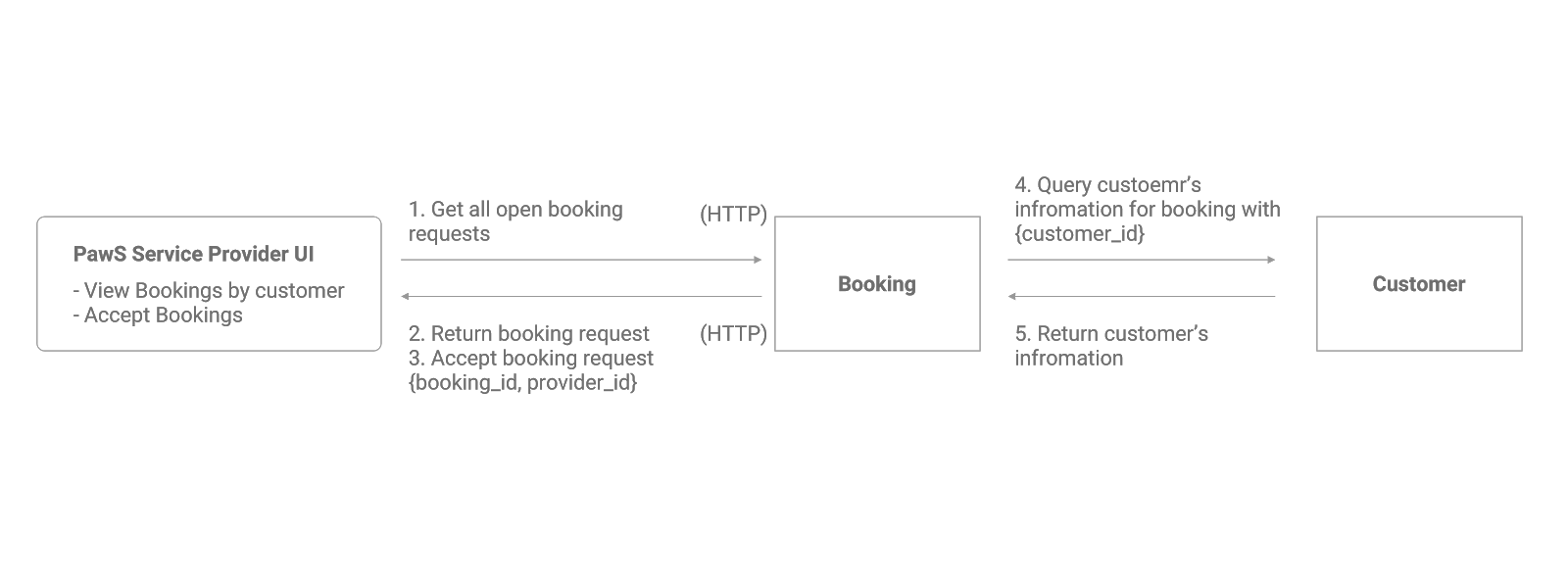
### **Steps Explanation for Scenario 1**

1. Customer will be able to access the list of service providers and their availability and chooses a slot.
2. Upon receiving this UI request, PawS UI then sends a [POST] request to PawS Enterprise Solution book the slot.
3. PawS Enterpise Solution then sends then returns the creation status to the PawS customer UI.
4. The PawS UI then displays the booking page with the new booking’s date and time availability of the service provider to the customer.

### **Microservices**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Service Name*** | ***Description of the functionality*** | ***Operational information*** | ***Input (if any)*** | ***Output (if any)*** |
| Service Provider | Get a list of service providers day/time and services provided | HTTP[GET] /serviceprovider | none | [{  provider\_day,  provider\_mobile,  provider\_name,  provider\_price,  provider\_service,  provider\_time  }] |
| Booking | Create new booking for customer | HTTP[POST]/  Booking | {  customer\_mobile,  provider\_id,  provider\_name,  provider\_service,  booking\_time,  booking\_day,  booking\_price,  booking\_status,  booking\_payment\_status  } | "Booking Creation Status" |

## **User Scenario 2 – Service Provider manage booking**



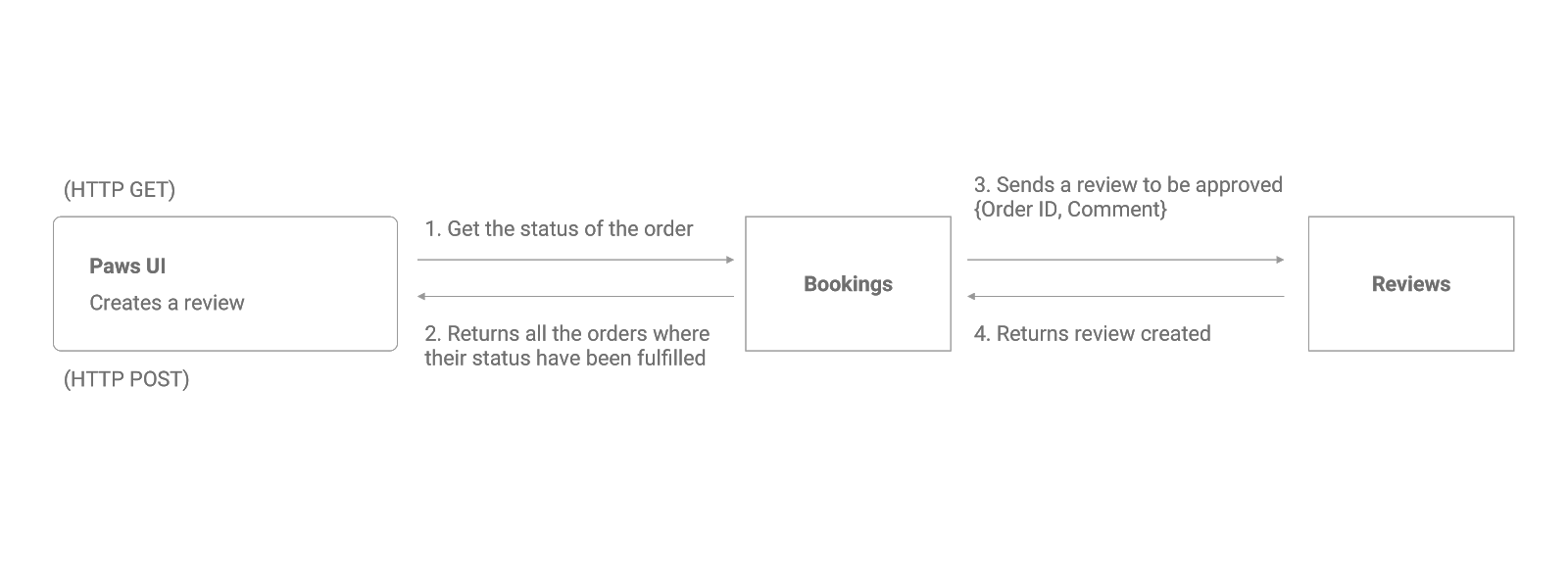
### **Steps Explanation for Scenario 2**

1. Customer starts the booking request by selecting his available day and time as shown on the PawS UI. Thereafter, the UI will show the passenger a list of available Service Providers he can choose from based on his availability. Passenger chooses amongst the list of Service Providers.
2. Upon receiving this UI request, PawS UI then sends a GET request to PawS Enterprise Solution to get the list of service providers as well as the available booking slots.
3. PawS Enterpise Solution then sends a reply message back to PawS UI containing the list of available slots of service provider.
4. The PawS UI then displays the booking date and time availability for service provider to the passenger.

### **Microservices**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Service Name*** | ***Description of the functionality*** | ***Operational information*** | ***Input (if any)*** | ***Output (if any)*** |
| **Booking** | Get a list of available = bookings | HTTP[GET]  /booking/  <string:provider\_id> | none | [{  customer\_mobile,  provider\_id,  provider\_name,  provider\_service,  booking\_time,  booking\_day,  booking\_price,  booking\_status,  booking\_payment\_status  }] |
| Update booking status | HTTP[POST]  /booking/status/  <string:booking\_id> | {  booking\_id, booking\_status  } | "Booking Status Change" |

## **User Scenario 3 – Customer Creates Review**



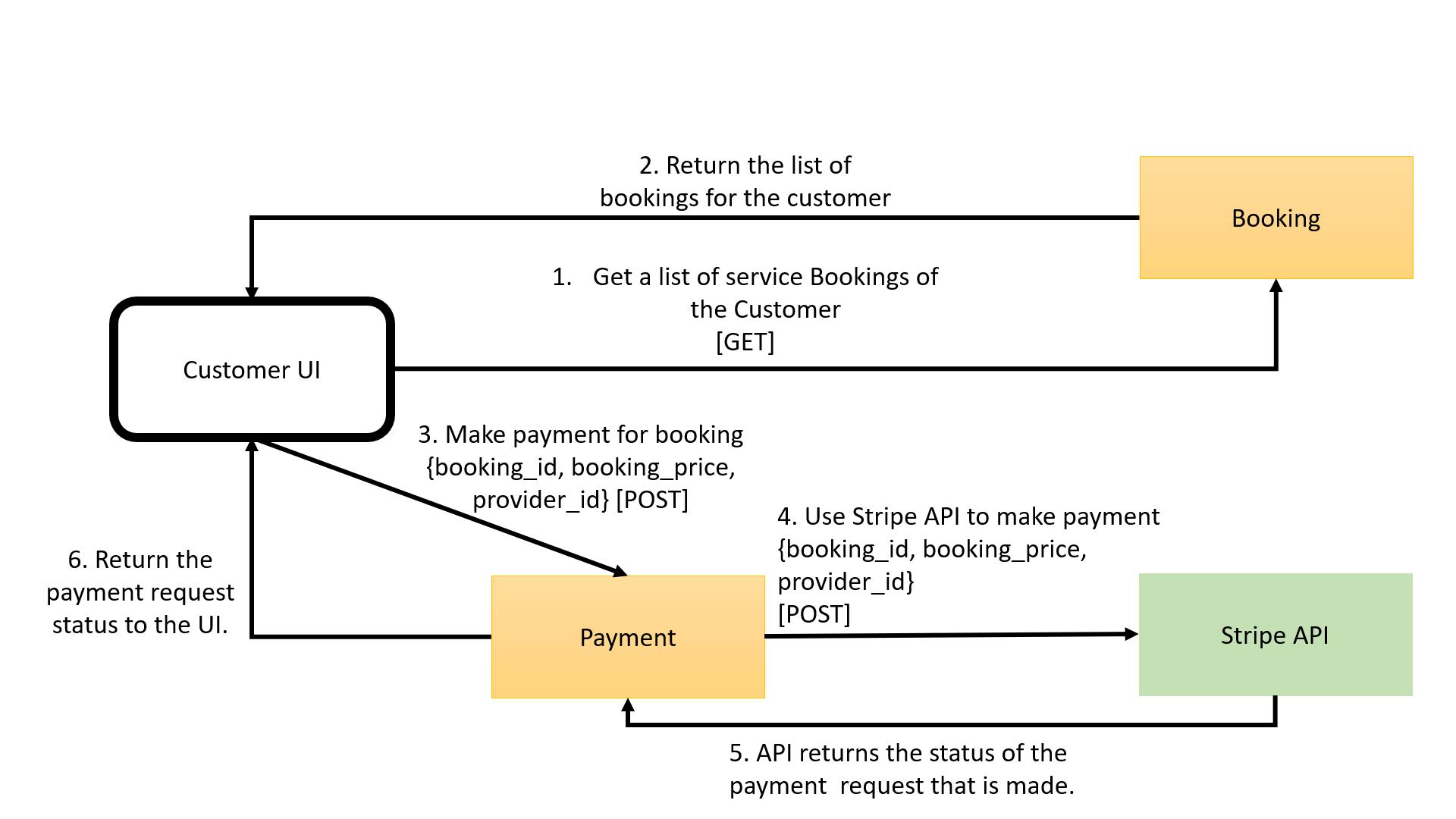
### **Steps Explanation for Scenario 3**

1. The customer checks the UI for the fulfilled order and submits a review to the service provider.
2. The PawS UI then gets the submitted review from the PawS Enterprise Solution. The PawS Enterprise Solution returns the review to the service provider.
3. The UI then shows the customer the submit status of the review. Successful reviews are shown for the service provider.

### **Microservices**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Service Name*** | ***Description of the functionality*** | ***Operational information*** | ***Input (if any)*** | ***Output (if any)*** |
| Review | Customer creates a review based on booking | HTTP[POST]/review/  <string:booking\_id> | {  booking\_id,  review\_star, review\_comment  } | "Review Submission Success" |
| Booking | Get list of booking made by customer | HTTP[GET]  /booking/  <string:customer\_mobile> | {  customer\_mobile  } | [{  customer\_mobile,  provider\_id,  provider\_name,  provider\_service,  booking\_time,  booking\_day,  booking\_price,  booking\_status,  booking\_payment\_status  }] |

## **User Scenario 4 – Customer makes Payment**



### **Steps Explanation for Scenario 4**

1. The customer UI sends a HTTP request message to the booking microservice to get the booking information for the customer’s booking. The customer checks the price of his booking on the Customer UI.
2. The booking microservice returns a reply message consisting of the customer’s booking information.
3. Next, the customer proceeds to make payment on the Customer UI. The Customer UI sends a request message to the external payment service informing the service about the customer’s payment.
4. The microservice sends a payment request using the Stripe API gateway using [POST].
5. Stripe API payment then sends back a reply message containing the payment success status.
6. The Customer UI displays the payment status to the UI updates payment status into the payment microservice.

### **Microservices**

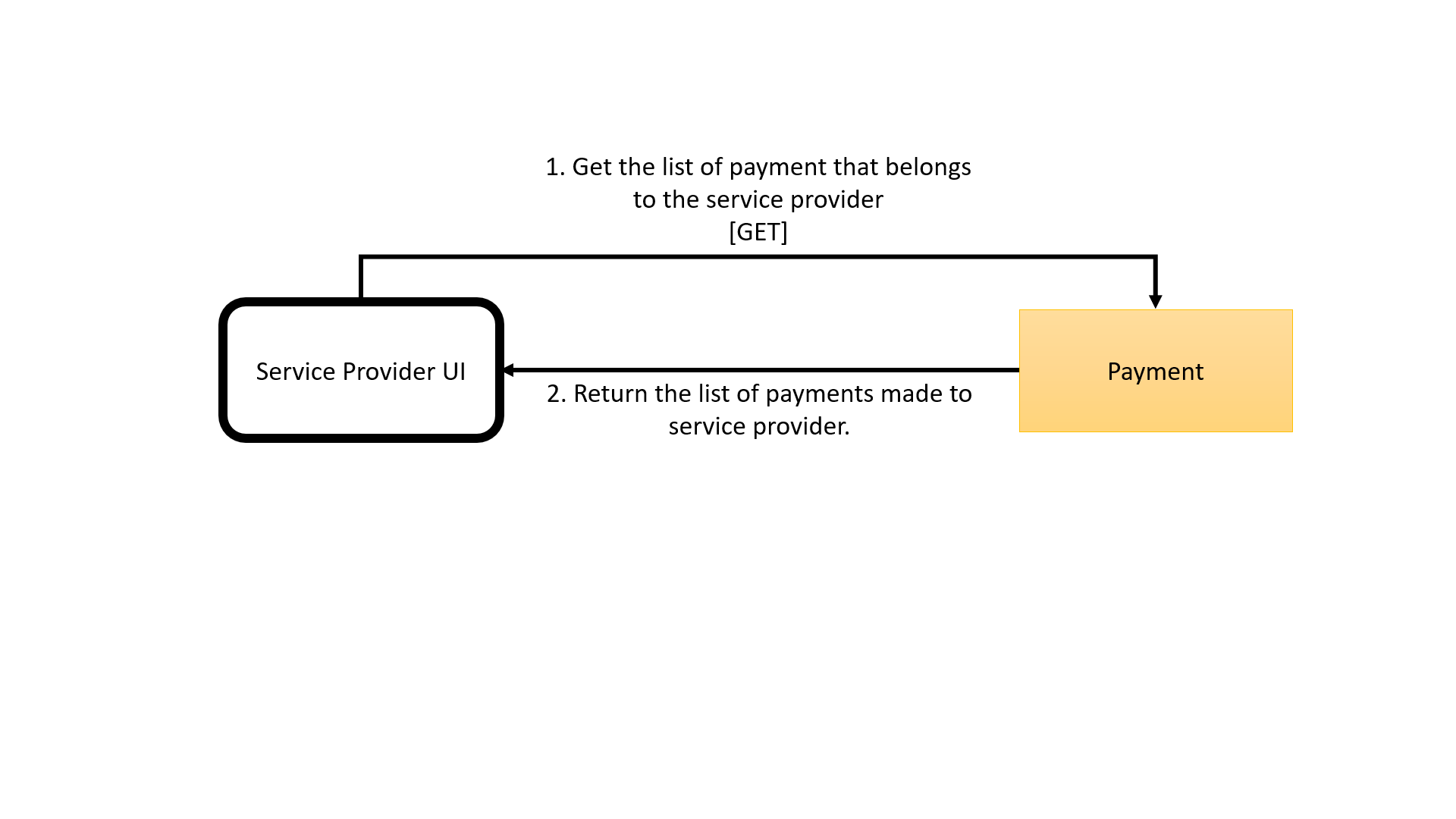
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Service Name*** | ***Description of the functionality*** | ***Operational information*** | ***Input (if any)*** | ***Output (if any)*** |
| *Payment* | *Update booking status* | [POST]/payment/  <string:payment\_id> | {  payment\_id,  booking\_id,  booking\_price,  provider\_id  } | "Update Status" |
| *Stripe Payment API* | *Validating payments* | Charge.php will [POST] to the Stripe’s DB | {  first\_name, last\_name,  card\_number,  payment\_amount  } | "Payment process status" |
| Booking | Get list of booking made by customer | HTTP[GET]/customer/  <string:customer\_mobile> | {  customer\_mobile  } | [{  customer\_mobile,  provider\_id,  provider\_name,  provider\_service,  booking\_time,  booking\_day,  booking\_price,  booking\_status,  booking\_payment\_status  }] |

### **Beyond the Labs**

* + - 1. **Stripe API**In this portion, the Stripe API has bee used to do the payment of each customer. The Stripe API is organized around REST. Upon user payment and confirmation, it sends the data from the webpage via the HTTP POST method and passes the information of the transaction in the JSON data format. The information is saved in the Stripe account holders’ database.

On top of that, this transaction is also captured into the payment microservice database table so that the merchant, in this case, the service provider can check whether the payment has been made for the service provided. This will be mentioned in part 3.5.

## **User Scenario 5 – Service Provider Acknowledges Payment**



### **Steps Explanation for Scenario 5**

1. The service provider logs in and the UI will invoke the payment microservice to retrieve all payments that have been successfully made for the service provider.
2. The microservice will retrieve the request and returns to the Service Provider UI to be reflected back to user.

### **Microservices**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Service Name*** | ***Description of the functionality*** | ***Operational information*** | ***Input (if any)*** | ***Output (if any)*** |
| Payment | Get list of payments made to the service provider | HTTP[GET]/payment/provider/  <string:provider\_id> | {provider\_id} | {  payment\_id,  booking\_id,  booking\_price, } |

# **References**

* 1. Stripe. (n.d.). Stripe API Reference. Retrieved from <https://stripe.com/docs/api>
  2. jQuery and Javascript Plugins. Retrieved from
     1. <https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css>
     2. <https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js>
     3. <https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js>
     4. <https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js>

# **Appendices**

* 1. Technical Overview Diagram
  2. Customer Sends Pet for Grooming Services Process