

|  |
| --- |
| **BBQ Ordering App (BBQ-On)** |

|  |
| --- |
| **IS213 Enterprise Solution Development**  **G7-T4**  **Assignment**  **Team Members:**  **| Eddie Leow | Fakhrul Fais | Hsieh Shin Min |**  **| Jasky Ong | Md Yazid | Teo Rue Quan |** |

## 

## **Introduction**

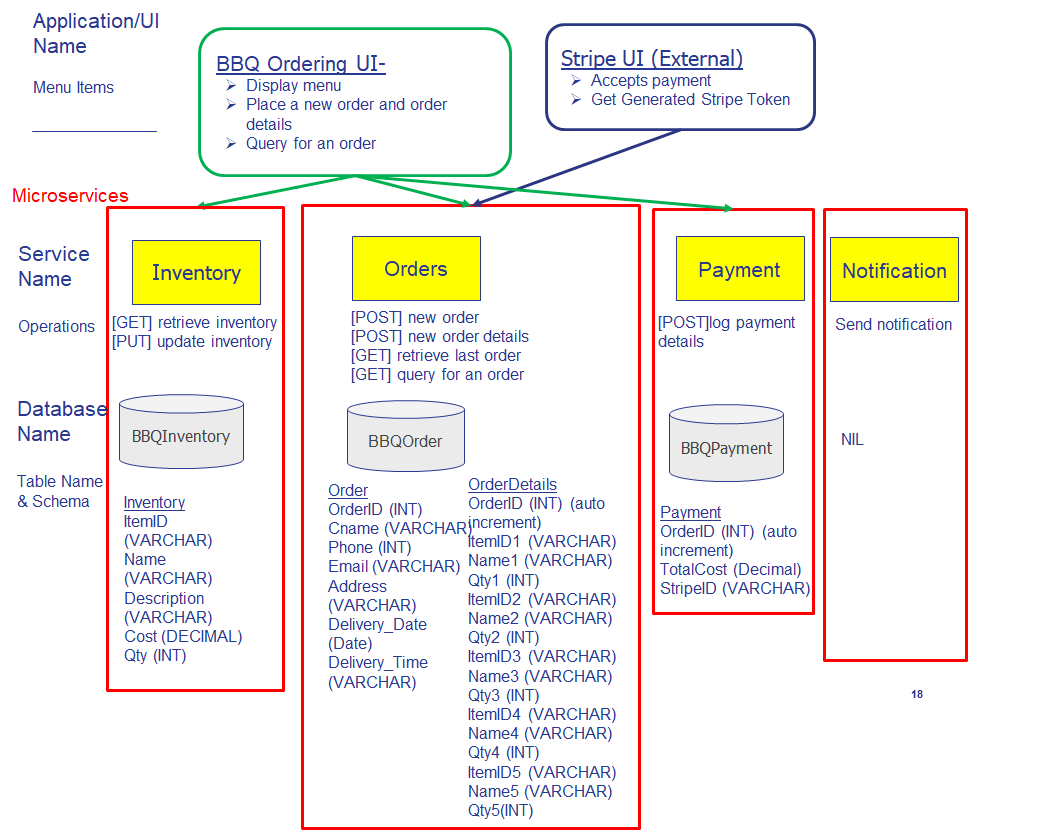
BBQ-On is a BBQ ordering app that pledges to offer an easy ordering of high-quality BBQ food items for an event of any scale. BBQ-On works based on inventory availability basis where we would offer customers a variety of BBQ food item that we have on hand. This is to ensure that we offer the freshest item with optimal quality to all our customers. Our BBQ-On ordering app is designed to help customers in ease of placing and checking of orders. The customer could also check on the BBQ food item (Menu) we offer and available at that point in time. Thus our user scenarios are as follows:

User Scenarios:

1. **Customer check on site to see what is available** *(Refer to Appendix Diagram 1c for the Scenario Overview diagram)*
2. **The customer creates order and makes payment** *(Refer to Appendix Diagram 2c for the Scenario Overview diagram)*
3. **Customer check order made/placed** *(Refer to Appendix Diagram 3c for the Scenario Overview diagram)*

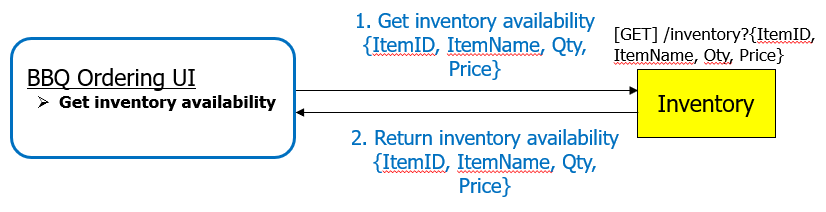
## **Technical Overview Diagram**

Our application consists four microservices which are Inventory, Orders, Payment and Notification.

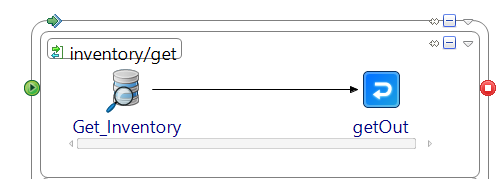


## **User Scenarios**

User Scenario 1 - Customer check the menu to see what is available on the BBQ Ordering site



*Diagram 1a: User Scenario 1 Diagram*

**

*Diagram 1b: Process Definition Diagram of Inventory Service*

*(\*Qty is the stock count of the items and is not displayed on the UI)*

Customer will enter the website and view what is available to order on the menu:

1. BBQ Ordering UI invokes Inventory service via GET request to inventory operation.

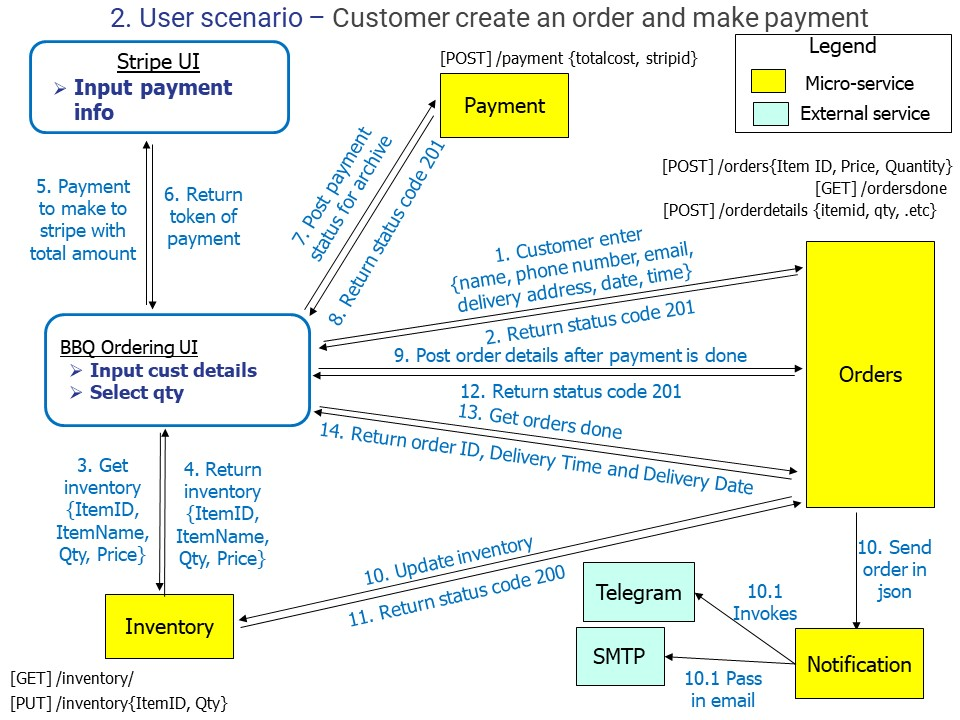
2. The operation does a JDBC get to the bbqinventory db, inventory table via the JDBC Query activity.

3. The service returns the itemid, name, description, cost and qty via the getOut activity.

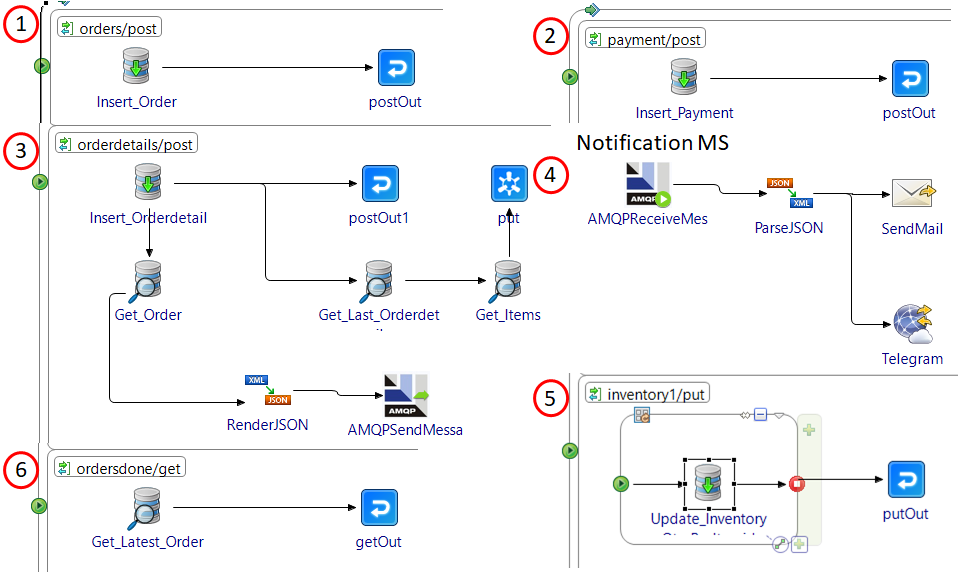
4. UI displays the Item ID, Item Name, Description and Price on the website

5. Customer will be able to click on the” Make An Order With Us” button to create an order

User Scenario 2 - Customer creates an order and make payment via credit card



*Diagram 2a: User Scenario 2 Diagram*



*Diagram 2b: Process Definition Diagram(s) of Orders, Payment and Notification Services*

Customer creates an order request for the BBQ food items through the UI:

1. Customer enters name, phone number, email, delivery address, delivery date and time and click on the “Submit” button on the UI. The UI then invokes Orders service via POST to the orders operation by passing in *cname, phone, email, address, delivery\_date and delivery\_time (orderid is auto-increment)*. The operation does a JDBC insert into the bbqorder database, orders table via the JDBC Update activity. The service then returns the status code of 201 via the postOut activity upon successful insert. *(Refer to Diagram 2b #1)*

2. UI then invokes Inventory service via GET to inventory operation and display the items to order on the UI. *(Refer to User Scenario 1)*

3. Customer enters the quantity of each items they wish to order in the UI and click on the “Proceed with Payment button”. Stripe Payment UI will be invoked, a window will pop out for the customer to enter his/her credit card information for payment. After successful payment, the Stripe UI will return a payment token to the UI.

4. UI invokes Payment service via POST to payment operation by passing in totalcost and stripeid *(orderid is auto-increment)*. The operation does a JDBC insert into the bbqpayment database, payment table via the JDBC Update activity. The service then returns status code of 201 with Payment Created message via the postOut activity. *(Refer to Diagram 2b #2)*

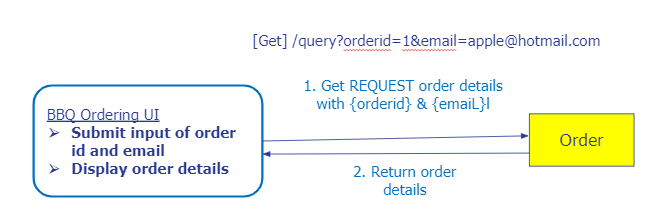
5. Once payment is completed, UI invokes Orders service via POST to orderdetails operation by passing in itemid1, name1, qty1…itemid5, name5 and qty5 *(orderid is auto-increment)*. The operation does a JDBC insert into the bbqorder database, orderdetails table via the JDBC Update activity. The service then returns status code of 201 with Orderdetails Created message via the postOut activity. *(Refer to Diagram 2b #3)*

6. Upon successful insertion of orderdetails, Orders service (Publisher) performs **asynchronous communications (event)** by sending a JSON file to RabbitMQ (Message Broker) via AMQPSendMessage activity to Notification service (Subscriber). In Notification service, AMQPReceiveMessage activity will receive the message, and will invoke the Telegram API and Send Mail SMTP via InvokeRESTAPI activities. SendMail activity will retrieve the customer’s email using ParseJSON activity. BBQ-On owner will be notified via Telegram message for a new order created and customer will be notified via email for a successful order placed and their order id. *(Refer to Diagram 2b #3 & #4)*

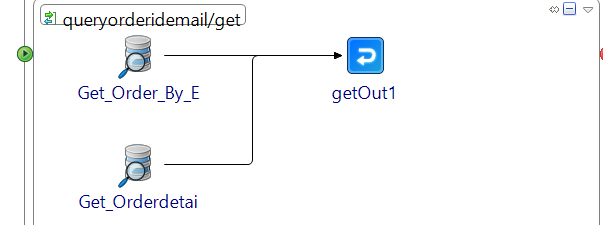
7. Meanwhile, Orders service (Consumer) performs a **synchronous communication (command)** to the Inventory service (Provider) via PUT request. Updated quantity per item is passed to Inventory microservice via PUT endpoint to inventory1 operation which does a JDBC update into the bbqinventory database, inventory table via the JDBC Update activity. The service then returns status code of 200 with Inventory Ok message via the putOut activity. *(Refer to Diagram 2b #5)*

8. Lastly, UI will be redirected to order\_success.html, which invokes Orders service via GET to ordersdone operation. The operation does a JDBC get to the bbqorder db, orders table via the JDBC Query activity and returns the last inserted entry to the UI via a getOut activity. The UI will display the *orderid, delivery\_time and delievery\_date. (Refer to Diagram 2b #6)*

User Scenario 3 - Customer check on his/her order



*Diagram 3a: User Scenario 3 Diagram*



*Diagram 3b: Process Definition Diagram of Orders Service*

Customer will enter the website and check on their order made:

1. The customer enters their Order ID and Email and click on the “Submit” button on the BBQ Ordering UI.

2. UI invokes the Orders service via GET to the queryorderidemail operation by passing the orderid and email. The operation does a JDBC get to the bbqorder database, orders table via JDBC Query activity.

3. The service then returns the matching orders and ordersdetails entry via the getOut activity.

4. UI will display all the orders and ordersdetails in a table format.

## **Web Services**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Service** | **Operation** | **Description** | **Input** | **Output** |
| *1* | *Inventory Service* | *Get all inventory* | ***GET /inventory***  *Returns a JSON file Inventory {itemid, name, description, cost, qty}*  *This service allow customers to access the menu to check what is available on the BBQ webpage* | *nil* | *itemid*  *name*  *description*  *cost*  *qty* |
| *Update inventory quantity upon successful order created* | ***PUT /inventory1***  *After a successful order and order details are created, this operation is invoked by a synchronous communication- command from Orders Microservice.*  *Quantity ordered by customer will be minus from the inventory quantity.*  *SQL statement: UPDATE INVENTORY*  *SET QTY = (QTY - ?)*  *WHERE ITEMID = ?* | *itemid, qty* | *qty* |
| *2* | *Orders service* | *Create an order* | ***POST /orders***  *This operation allows customer to create an order with his particular* | *cname, phone, email, address, delivery\_date, delivery\_time* | *orderid, cname, phone, email, address, delivery\_date, delivery\_time* |
| *Create order details* | ***POST /orderdetails***  *This operation allows customer to create order\_details with his particular.*  *It invokes a command to Inventory MS for updating quantity.*  *It also sends the last created order to message broker, RabbitMQ ,for notification MS.* | *itemid1, name1, qty1, itemid2, name2, qty2, itemid3, name3, qty3, itemid4, name4, qty4, itemid5, name5, qty5* | *orderid, itemid1, name1, qty1, itemid2, name2, qty2, itemid3, name3, qty3, itemid4, name4, qty4, itemid5, name5, qty5* |
| *Order done message* | ***GET /ordersdone***  *This operation returns the last order entry inserted into the database and return orderid, delivery\_time, delivery\_date.*  *i.e. (SELECT \* from orders order by orderid desc limit 1)* | *nil* | *orderid, cname, phone, email, address, delivery\_date, delivery\_time* |
| *Query an order* | ***GET /query?{orderid}& {email}***  *This operation returns an order and orderdetails by taking in 2 parameters, {orderid} and {email}* | orderid, email | *orders parameters,*  *orderdetails parameters* |
| *Payment Service Stripe API* | *Complete a payment* | ***Stripe UI Javascript***  *This service allows the customers to complete their payment* | *totalcost, orderid* | *Stripe Token*  *(stripid)* |
| *3* | *Payment Service* | *Logs a successful payment entry* | ***POST /payment***  *Upon successful payment in orders\_details.html, an entry is logged into the payment microservice with the payment token from Stripe API* | *orderid, totalcost, stripid* | *payment* |
| *4* | *Notification Services* | *Send Telegram message* | *Telegram api:* [*https://api.telegram.org/bot854460202:AAE0EiDN1jcsBysC5tXnt6M-4NgIbAUJ33E/sendMessage?chat\_id&text*](https://api.telegram.org/bot854460202:AAE0EiDN1jcsBysC5tXnt6M-4NgIbAUJ33E/sendMessage?chat_id&text)*(Chat\_id is the BBQ app’s owner, hardcoded)*  *This operation receives an order as JSON from from RabbitMQ (message broker). It then* ***notify*** *business owner of new order creation via telegram* | *Nil* | *Telegram message to owner* |
| *Send Email via*  *Simple Mail Transfer Protocol* | *Smtp.smu.edu.sg - Port 25*  *This operation* ***notify*** *customer of order success with orderid. Orderid is passed in as a parameter and concatenated into body message.*  *(\*Note: this web service only works when users are connected to SMU Wifi/ Internet access.)* | *orderid* | *Email to customer* |

# **Usage of XML schema**

|  |  |
| --- | --- |
|  |  |

The above screenshot shows the XML Schema for Inventory. The XML Schema is used for User Scenario 1 & User Scenario 2. User Scenario 1 used the following XML Schema to retrieve all the inventories from the database and produce the itemid, name, description, cost and qty as an output in JSON Object. On the other hand, User Scenario 2 used the following XML Schema to update each inventory input received synchronously from orders module and produce status code of “200” upon success. Once orderdetails from orders module received the status code of “200”, it will continue to other process.

## **Graphical User Interface:**

HTML with Javascript and CSS implementation from Bootstrap.

Consists of the Navigation Bar, Table Inventory and “Order, Payment & Submit” Button for Interactivity.

## **Beyond the Labs**

1. *Asynchronous communication using external services -Telegram API* 
   * *Function: Notify the owner of the BBQ-On of a new order created*

* Create a telegram bot using BotFather
* Get the token to access http API
* Create an invokeRESTAPI in tibco with the http and token
* Retrieve unique telegram ID and enter into Tibco required parameter field
* Set a default notification message to be sent whenever an order is created

1. *Asynchronous communication using external services - SMTP*
   * *Function: Notify the customer that the order is successful*

* Set up SMTP connection with smtp.smu.edu.sg, port 25
* Retrieve customer’s email from the latest order using JDBC query activity
* Pass in parameters to SendMail, email into “to”, orderid into “bodyText” and send the email

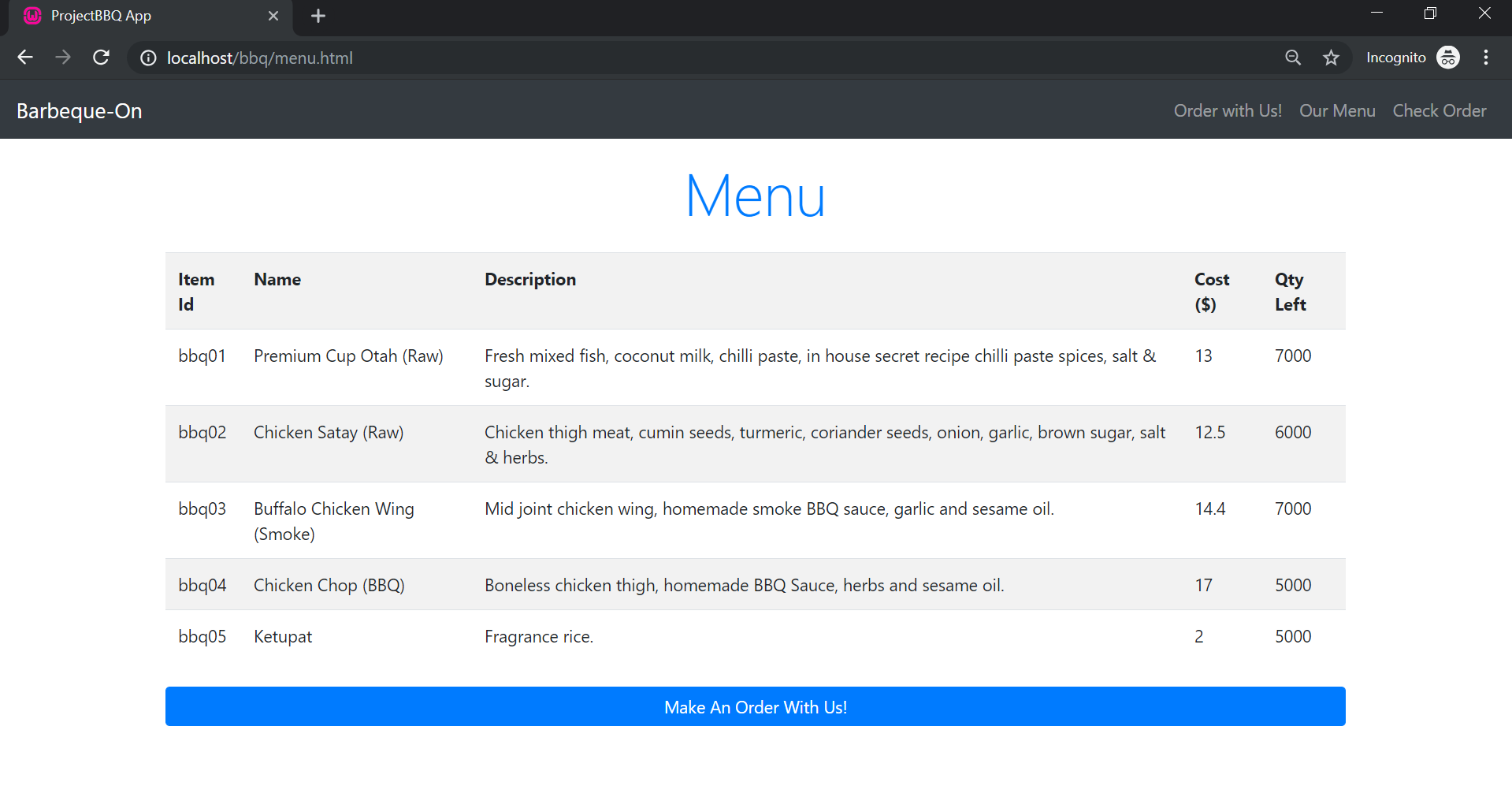
1. *Synchronous communication - Command. PUT service request from Orders Microservice to Inventory Microservice*
   * *Function: Update inventory database after an order is created*

* A PUT request endpoint take in value (qty) to be deducted from the inventory based on the item id. *(Refer to User Scenario 2 for a detailed description of PUT request)*

## **Scenario Walkthrough**

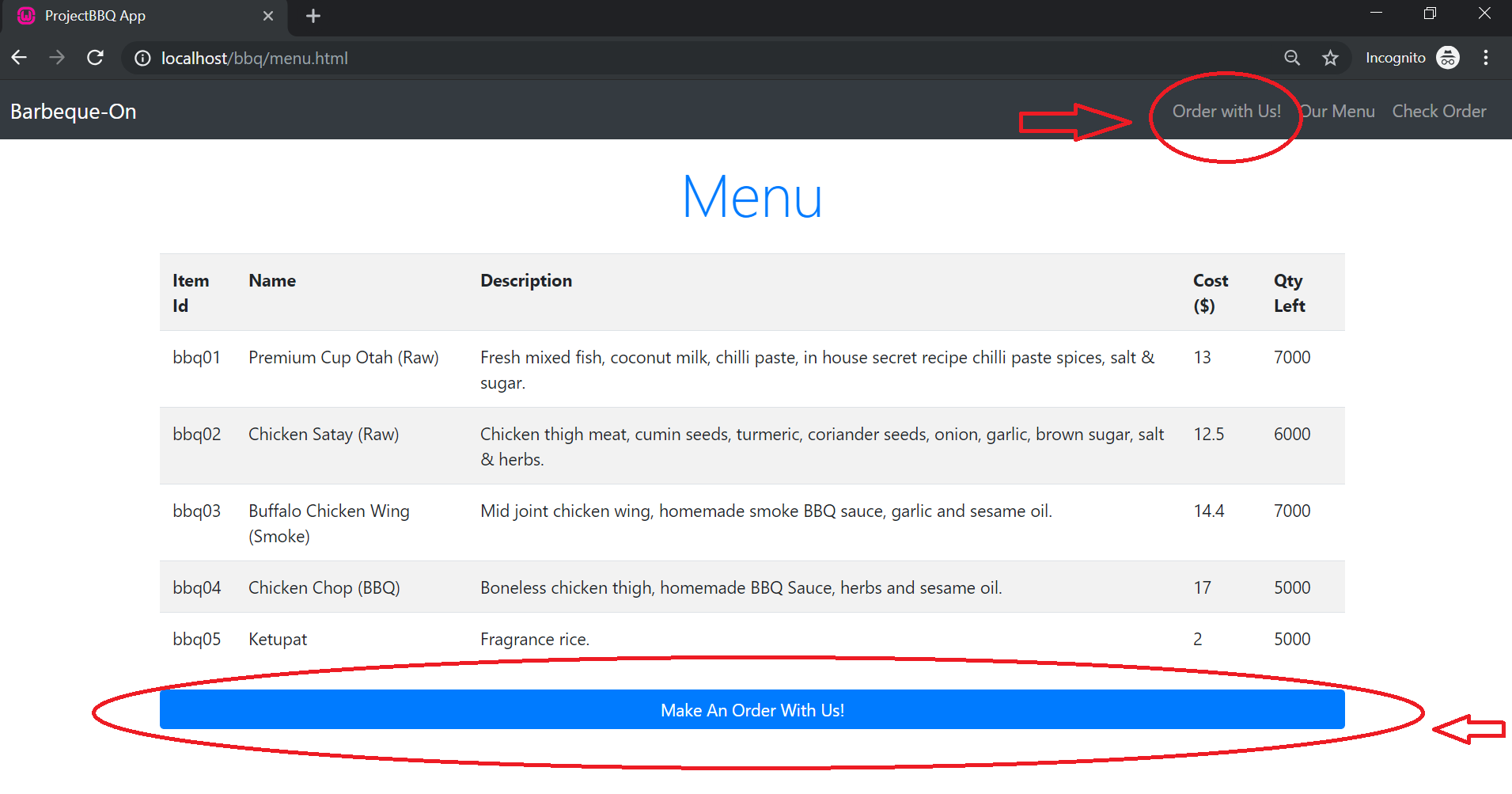
Scenario 1: Customer check on site what is available

1. Navigate to menu.html. UI will display items available for order

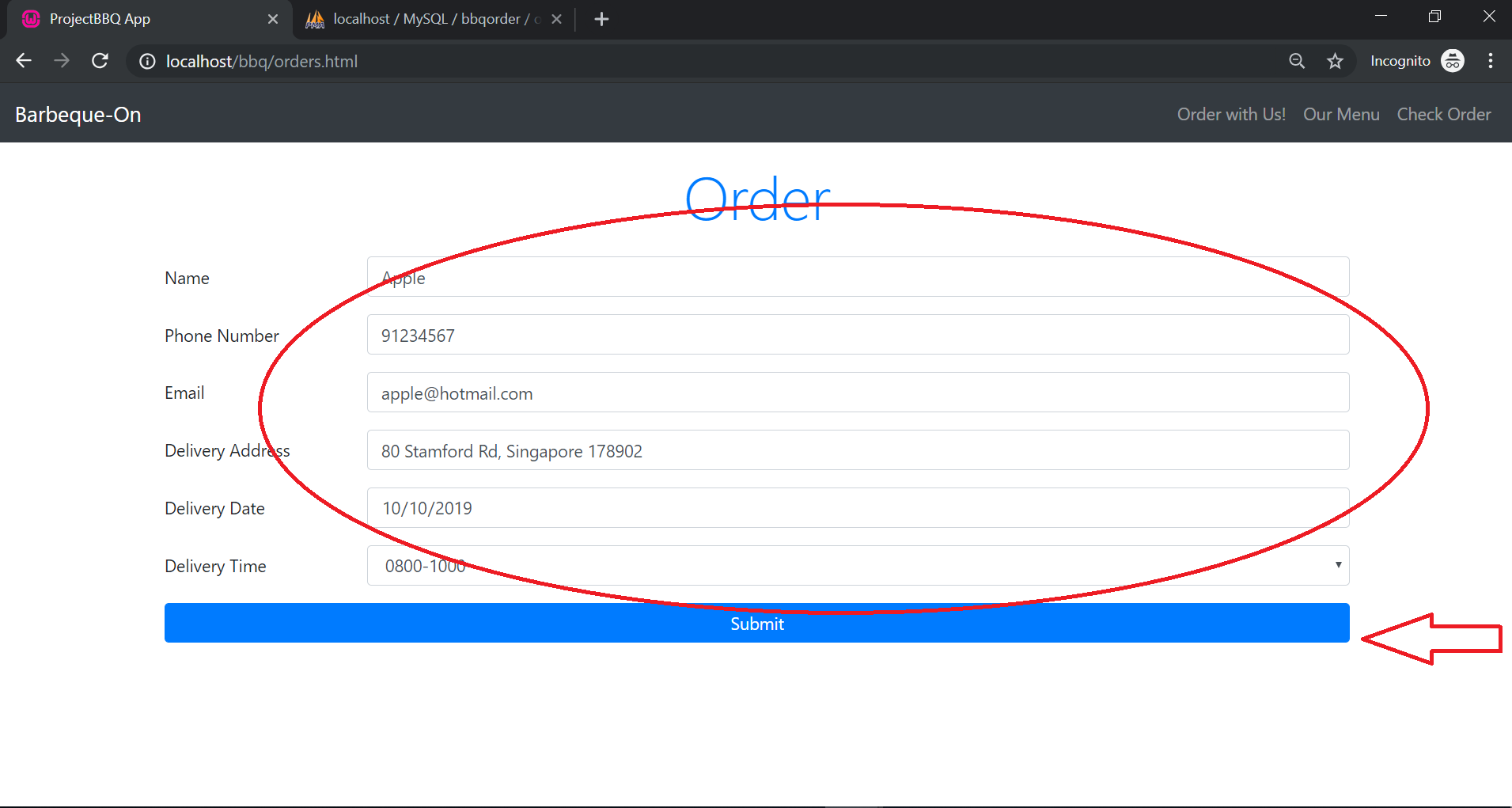


Scenario 2: Customer create order and make payment

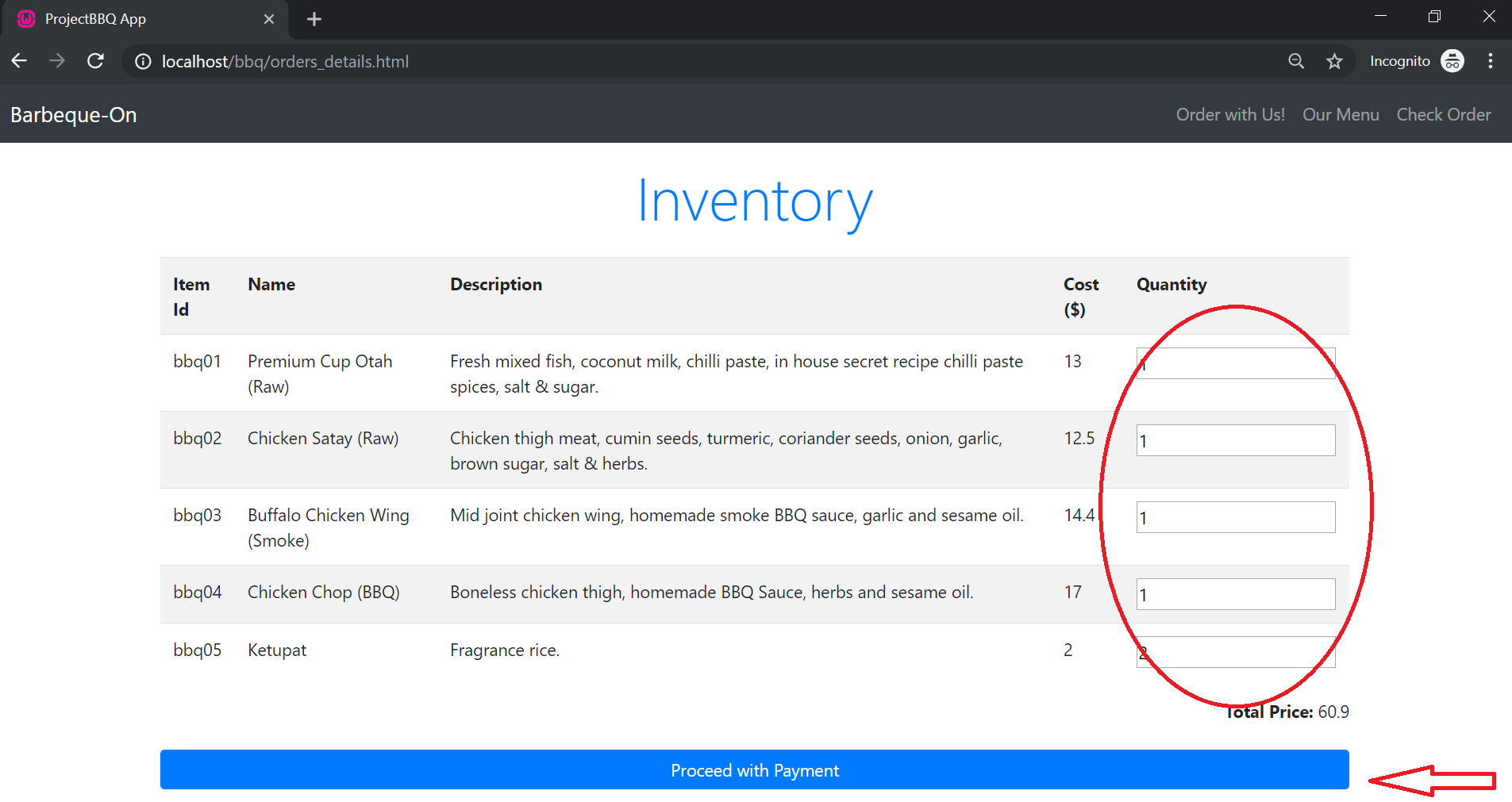
1. Click on “Make An Order With Us!” or “Order with Us!” on the navigation bar



2. Customer will be directed to orders.html, enter your particulars and click on submit

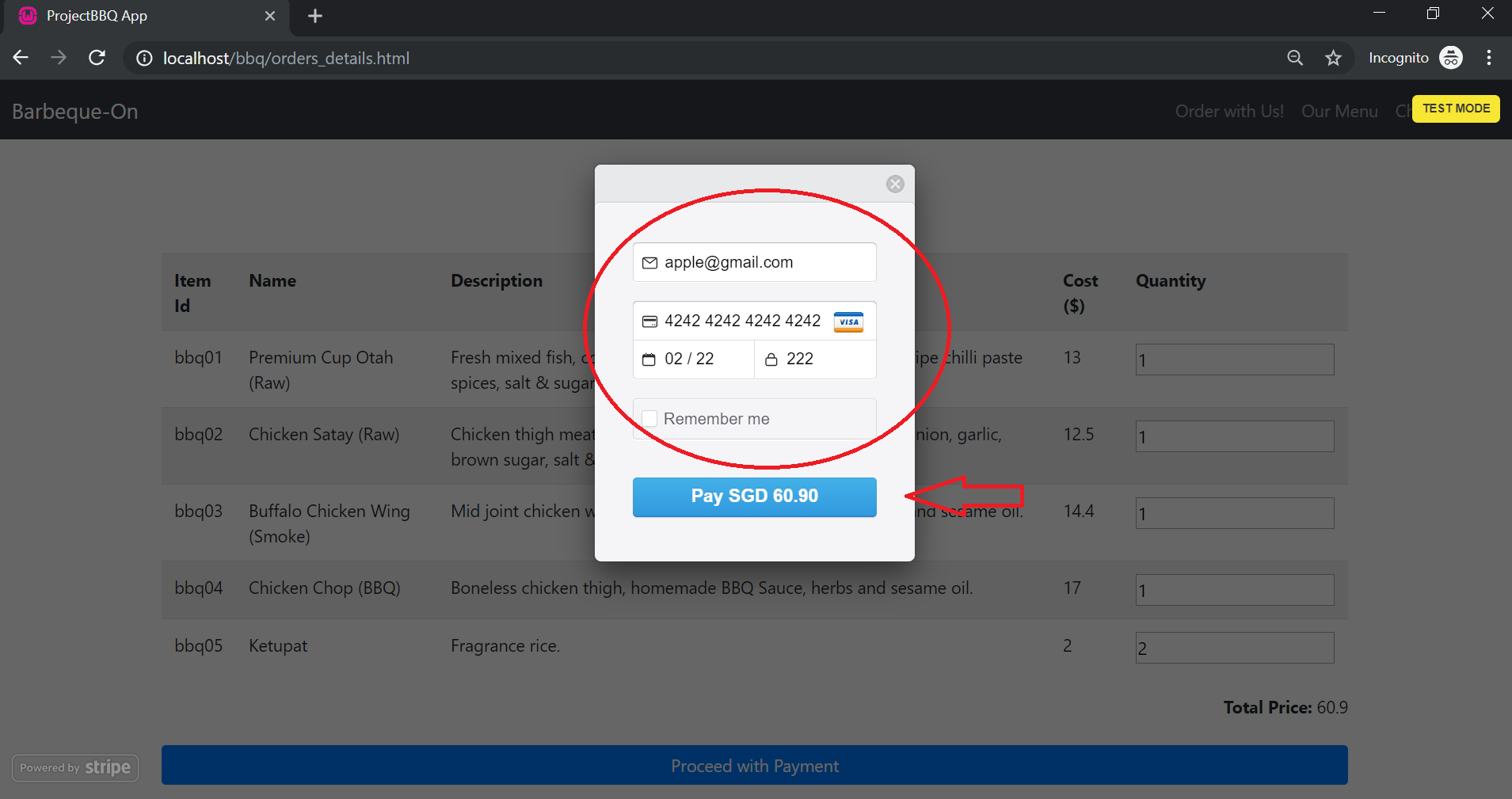


3. Customer will be directed to orders\_details.html. Enter quantity to buy. Click on “proceed with payment”

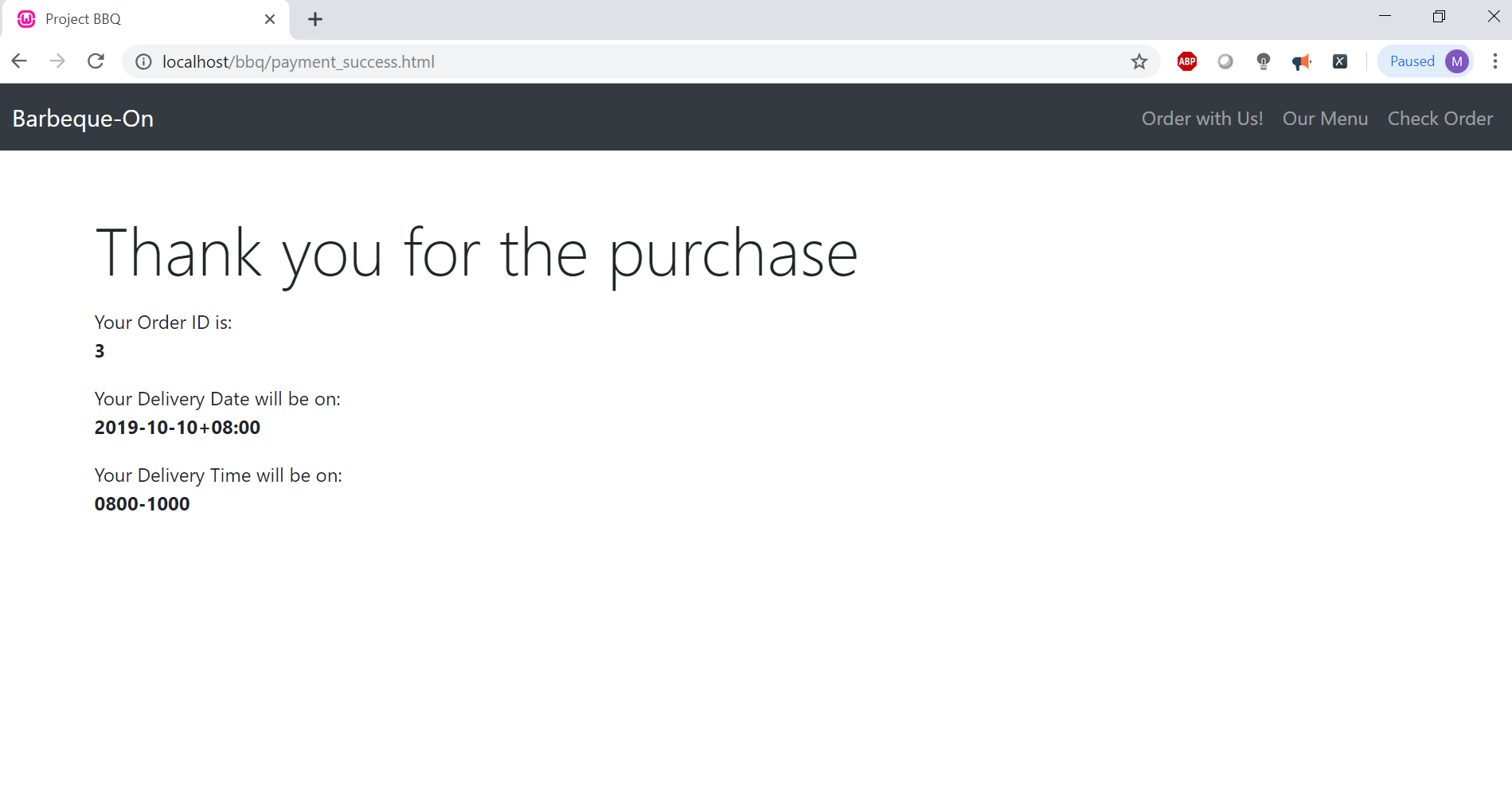


4. Enter payment details in the Stripe Payment UI and click on Pay

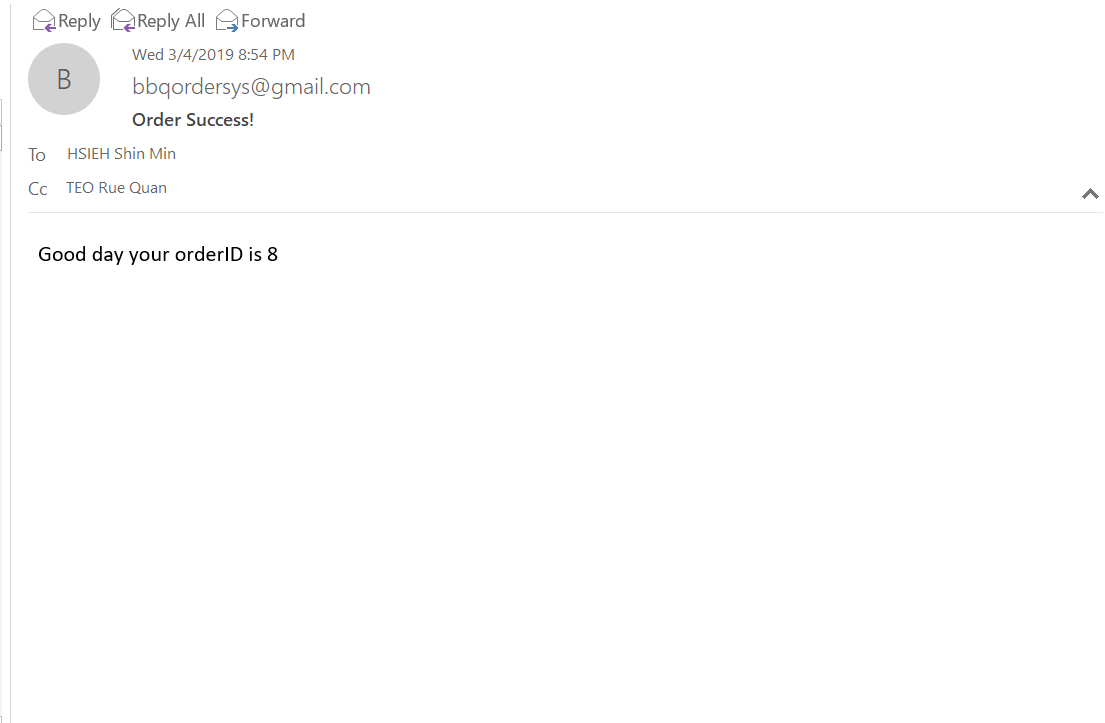
(Note: this is in test mode. Please enter “4242 4242 4242 4242” for card number, “02/22”for expiry date. “222” for CVC)



5. Customer will be directed to payment\_success.html. Displaying your orderid, delivery date and delivery time.



6. Customer will receive an email notification, indicating the orderid.

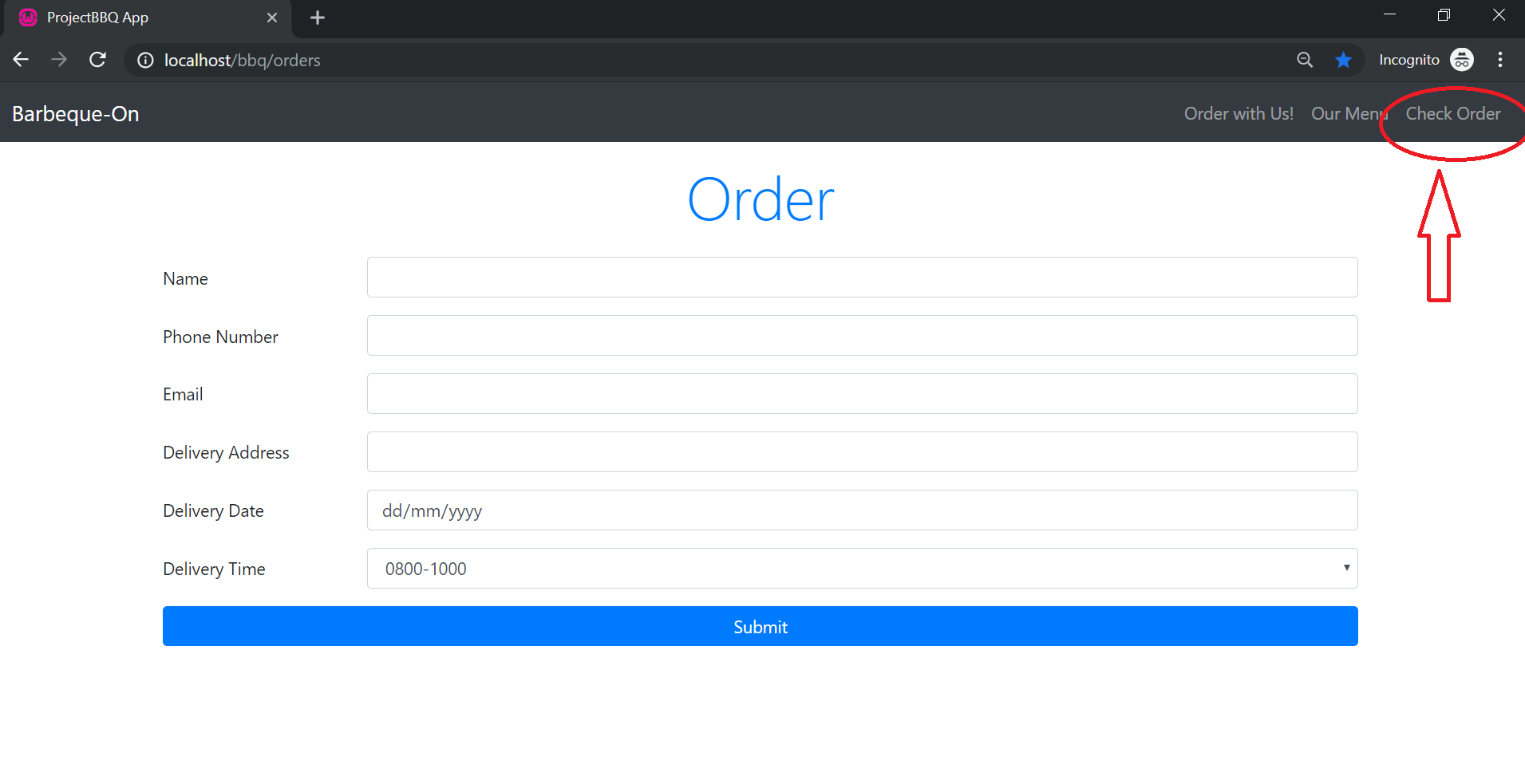


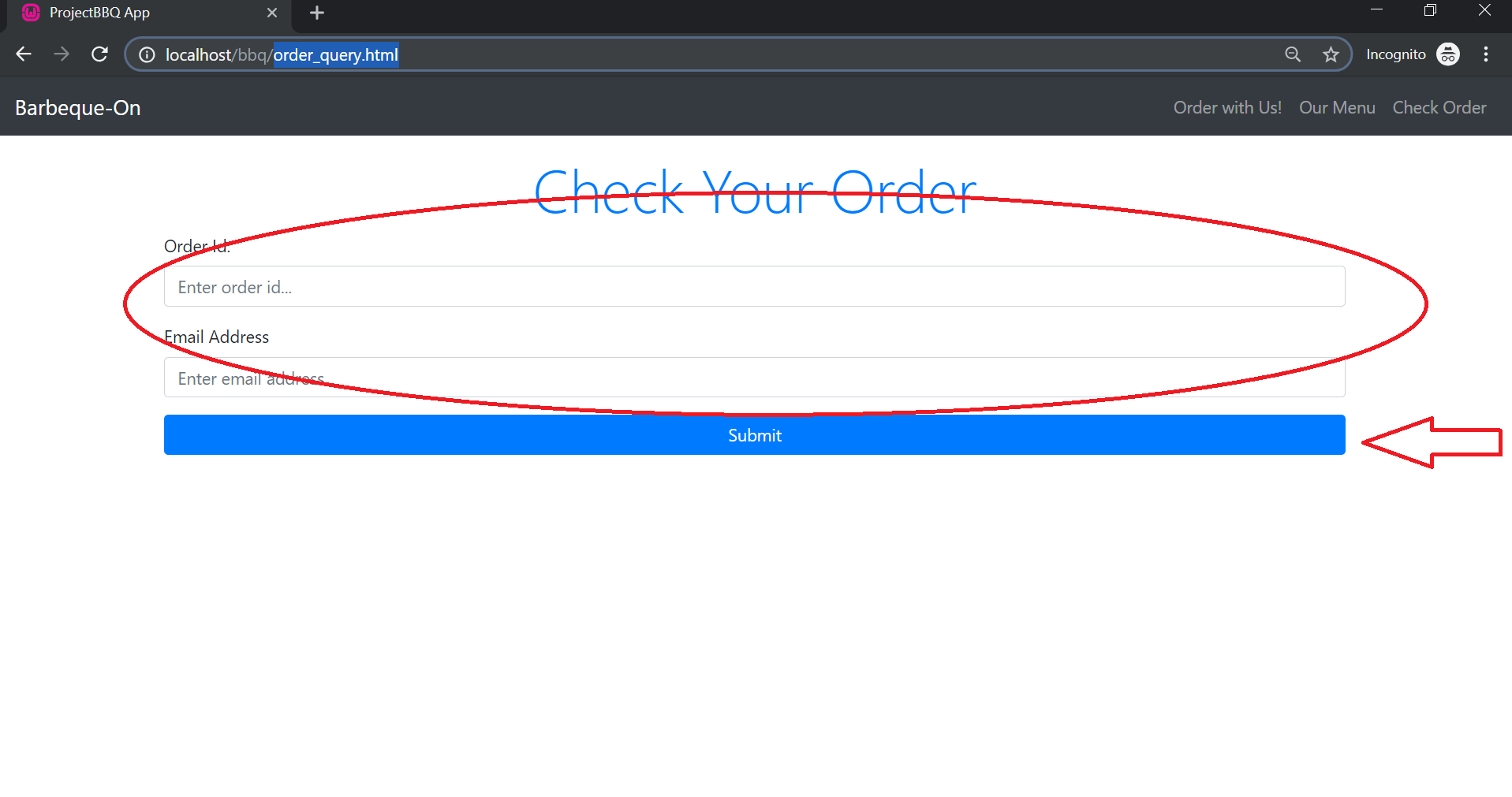
6.1 Meanwhile, owner of the BBQApp website will receive a telegram notification.



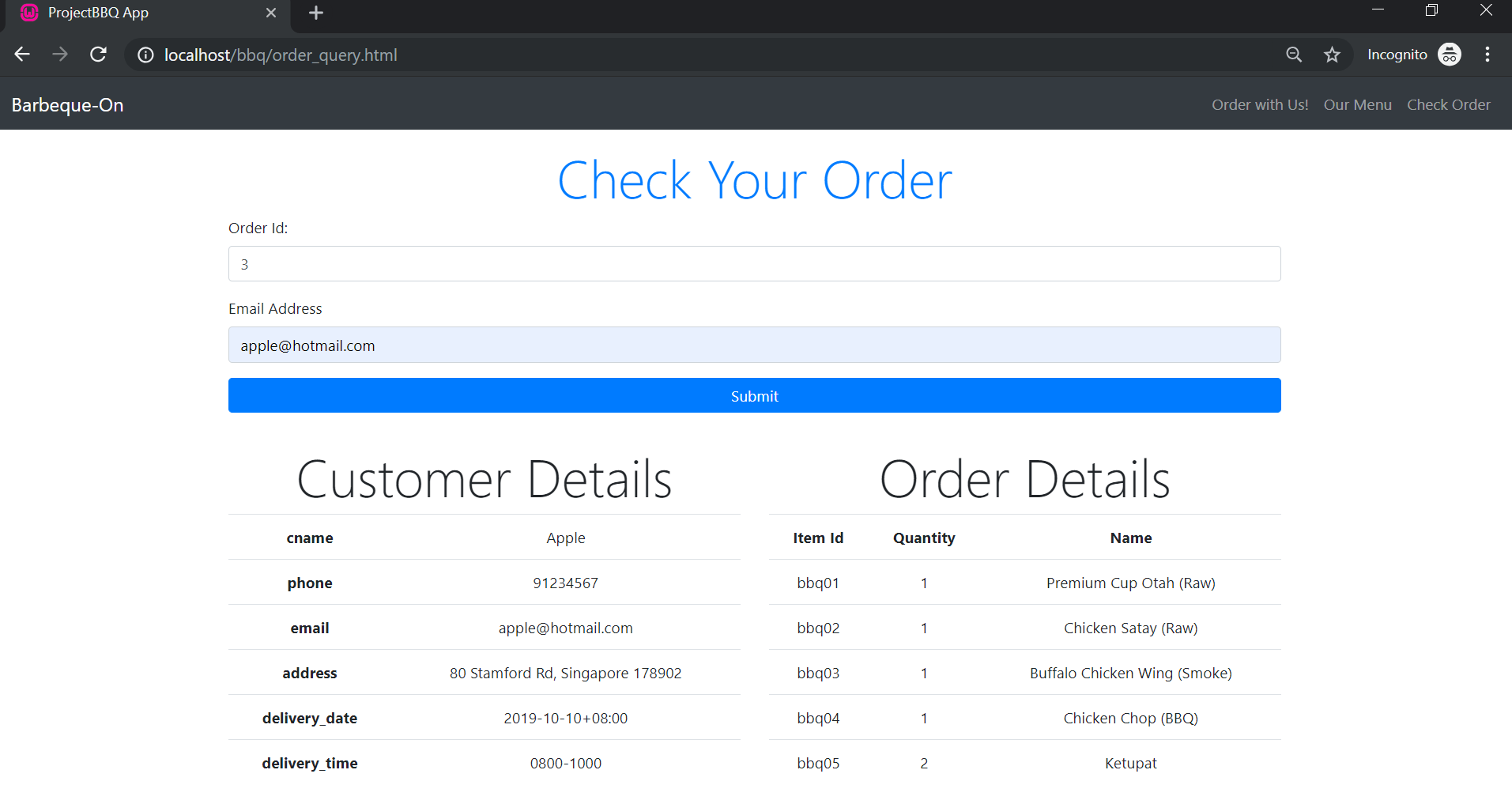
Scenario 3: Customer check order made

1. Clock on “Check Order”



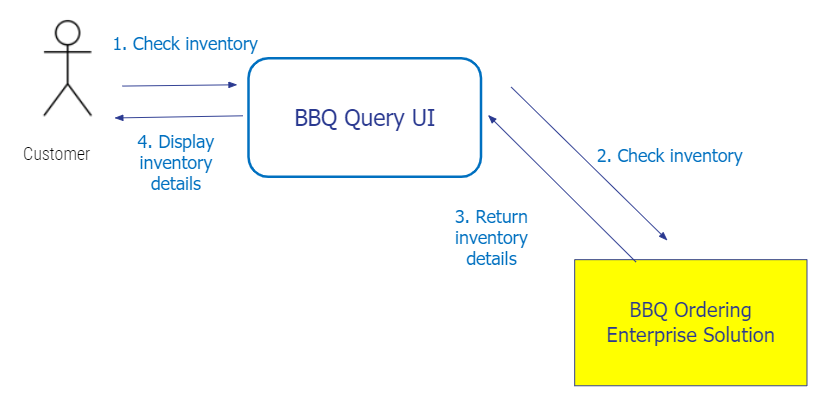
2. Customer will be directed to order\_query.html. Enter your orderid and email and click on “Submit

3. Your order and order details will be displayed.



**Appendix**

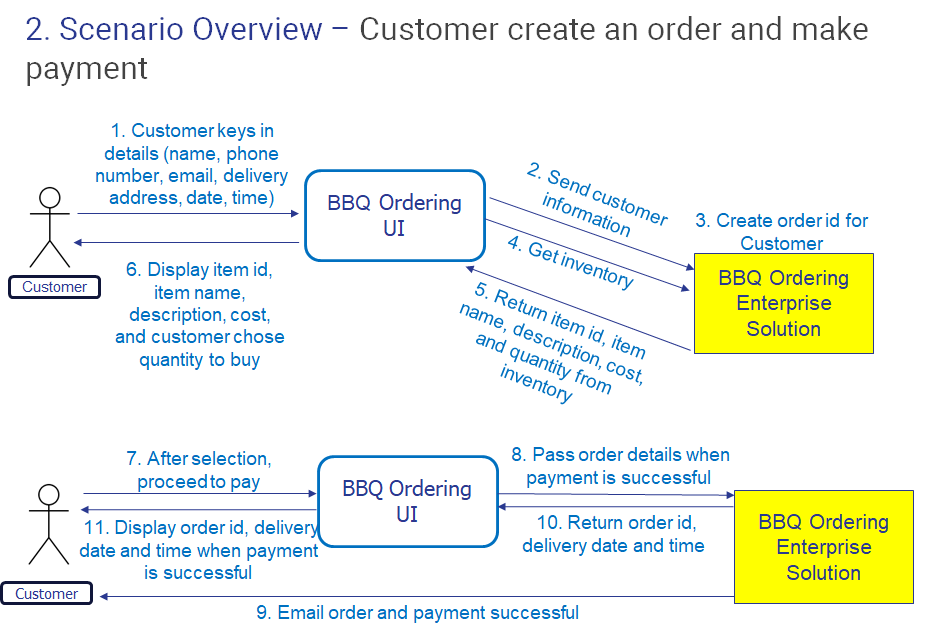
Scenario Overview 1 - Customer check on site what is available



**Diagram 1c: Scenario Overview for User Scenario 1**

The customer would be able to check the menu available at the point in time through the BBQ-On Ordering UI. BBQ-On ordering UI will check and get the updated inventory details from the BBQ-On ordering Enterprise Solution. The BBQ-On Ordering Enterprise Solution will then return the inventory details to the BBQ-On ordering UI to display all the details for the customer to view.

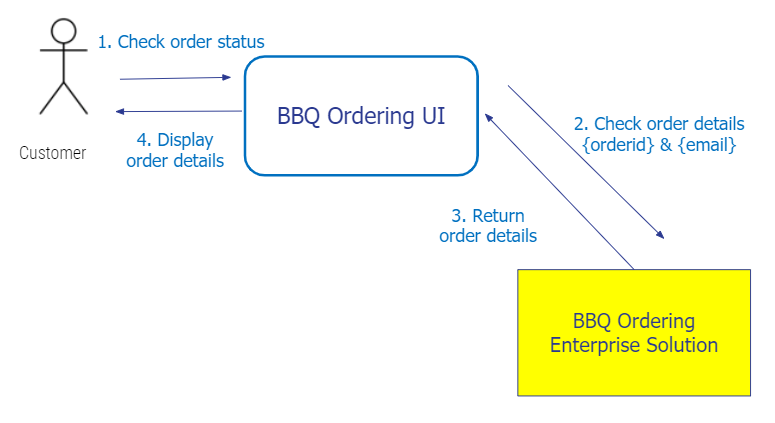
Scenario Overview 2 - Customer create order and make payment



**Diagram 2c: Scenario Overview 2 for User Scenario 2**

When the customer would like to place an order, the BBQ-On Ordering UI will display a page whereby the customer has to enter their details and particulars. Upon completion and submit, the BBQ-On UI will send the information to BBQ-On Ordering Enterprise Solution where the Order ID will be created. The BBQ-On Ordering UI will then retrieve the available inventory from BBQ-On Ordering Enterprise Solution and display the name description of the inventory for the customer to select the quantity they would like to order. After the selection is made, the customer would proceed by clicking make a payment on the BBQ-On UI. After payment is made through the UI, the BBQ-On UI will pass the order and payment detail to the BBQ-On Ordering Enterprise System. The system would then relay back payment success to the UI for it to be displayed for the customer to know that payment is success and order is created. Also at the same time, the system would send an Email t the customer on the success payment and order placed.

Scenario Overview 3 - Customer check order made



**Diagram 3c: Scenario Overview 3 for User Scenario 3**

The customer could check their order through the BBQ-On Ordering UI. The could enter their Order ID and Email to check on their order. The UI will then pass the details to the Enterprise system where the order would be retrieved and displayed on the UI for the customer to view.