Ticketstar
A Movie Ticket Booking Website
Integration Tests

- Sahana Pandurangi Raghavendra
- Megana Reddy Boddam

Startup and Shutdown Application Logic Tier:

- We have deployed our web application on AWS.
- The APIs used in our project are deployed on the Amazon API Gateway. All our APIs shut down when we delete this deployment in the Amazon API Gateway service.
- A unique API key is required to access all our API endpoints which is present in our code files. The API endpoints used by our web application are as follows
    a. Search Movies Server
        - addNewUser API
        - getUser API
        - listOfCities API
        - listOfTheaters API
        - listOfMovies API
    b. Ticket Booking Server
        - listOfShowDays API
        - listOfShowTimes API
        - listOfShowSeats API
        - addSeats API
    c. Food Beverages Server
        - listOfFood API
    d. Payment Processing Server
        - bookingCost API
        - listOfTickets API

- Currently, we have kept our API endpoints deployed because Amazon allows us to throttle the rate and volume of API requests. Thus, preventing any denial-of-service attacks on our web application.

  The URL to access the APIs is as follows:
  "https://9qhi5gewy6.execute-api.us-east-1.amazonaws.com/prod/" + <api endpoint>
  Eg: The api request to display the list of cities will be as follows
  "https://9qhi5gewy6.execute-api.us-east-1.amazonaws.com/prod/listofcities"

- After an API request is sent to the Amazon API gateway (above mentioned url where we have hosted our APIs), the lambda function associated with the API request is started and executed which internally has business logic to query or modify our database according to the inputs provided in the API request and returns a response. These inputs and responses are all summarized for each API method in the following pages.
- Thus, our middle tier starts up when an API request is sent by the movie ticket booking client to the Amazon API gateway, the middle tier then executes the business logic and

interacts with the database and is finally shutdown (execution of appropriate lambda service) when an API response is received by the client.

- In our test cases, we have parsed the json response from the response to the API request and validated them against the expected response.
- The unittest framework provided by python (used to code integration tests in our code) helps us validate the actual response with the expected response using the assertEqual() method. It also summarizes the number of test cases that have passed and failed in the test suite [1].

Instructions on Implementing the Integration Tests:

Execution Environment:

- Download the "integration_tests.zip" file.
- Setup PyCharm. Open the integration_test folder in pycharm.
- Go to Run → Edit Configurations, click on the "+" icon a, choose Python and configure the settings as shown in **Fig 1** with appropriate script path and python interpreter (python 2.7). Configure only <filename>testsuite.py files because these are the files that have the main function from where execution starts.

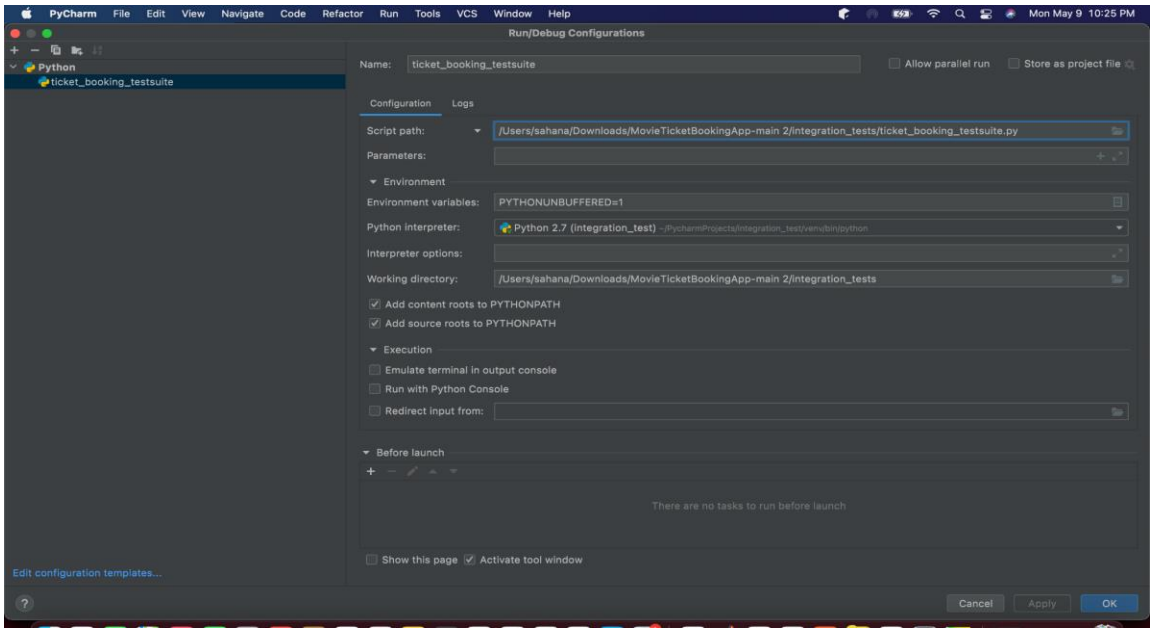

Fig 1: Test Suite Implementation Configurations

- Install the following libraries and run the following commands before running the tests.
  pip install urllib3
  pip install requests
- Navigate to File → New Project setup → preferences to new project. Click on python interpreter and add the requests library to the interpreter by clicking on the "+" icon→ search for requests package and install the package. The requests library must be a part of the packages installed on the interpreter as shown in Figure 2. Select the same interpreter in run configurations while running the python scripts.

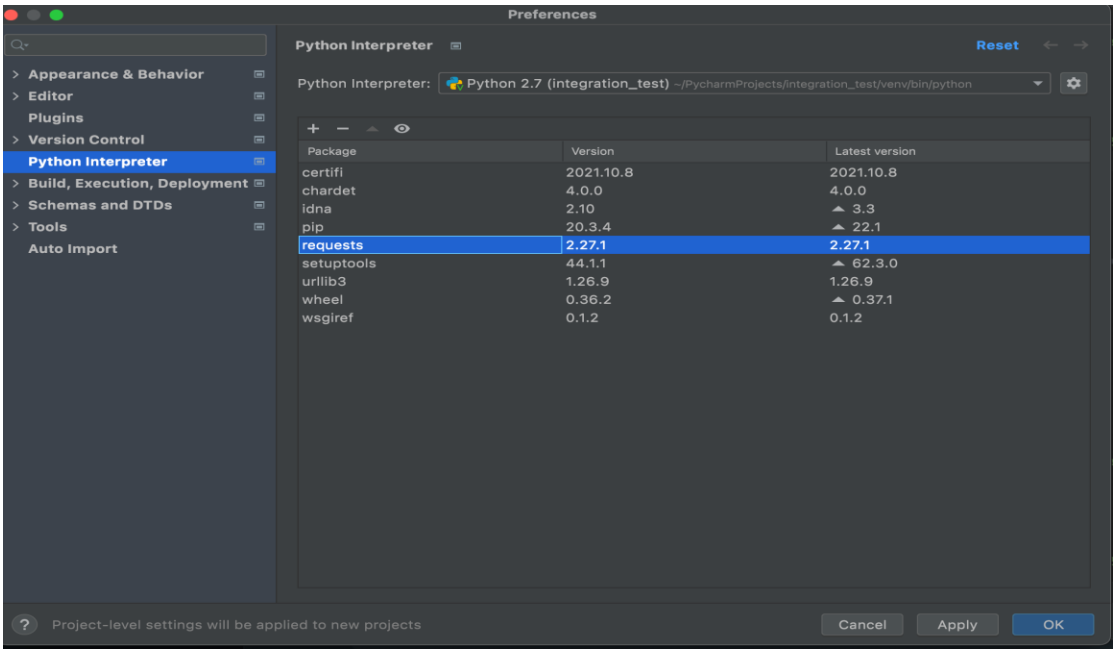

Figure 2: Packages that need to be installed to run the code.

- We have segregated our code into four types of files:
    a. A general library called “api_methods_lib.py” which consists of methods that will be used by all API integration tests.
    b. Libraries specific to each of the following servers: search_movies , payment_proccessing, food _booking and ticket_booking.
    c. A class containing all the integration test cases specific to each server.
    d. Test suites for each class of test cases, which correspond to specific servers.
- Run only the four test suite files to execute tests for each server. They are called food_booking_testsuite.py, payment_processing_testsuite.py, search_movies_testsuite.py and ticket_booking_testsuite.py .
- The results of each of the test suites are summarized in the pycharm terminal.
- We used python’s unittest library as a framework for our **integration tests.** All the tests in our “tests.zip” folder are integration tests.
- Each API described above has multiple methods, and we have created at least two tests for each method: one positive using the appropriate parameters and API Key and one negative without the API Key or with a valid API Key and invalid query parameters.
    a. The first integration test makes sure that the API endpoint can be accessed using a valid API key and appropriate lambda function executes the business logic by interacting with the database. **(Tests the integration between the middle tier and database)**
    b. The rest of the integration tests will be negative tests. They verify whether our business logic is utilizing the parameter inputs provided by the client appropriately to query or modify the database. **(Negative integration test for testing the interaction between middle tier and database)**

Tests for Search Movies Lambda Function:

- **API → addnewuser → Post**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
post_add_new_user_positive_test_one	{“first_name”: “Jane”, “second_name”: “Austen”, “email”: “jane.austen@gmail.com”, “phone”: 2380942358}	{message: "Sucessfully added the user to the database"}	Positive	Inputting valid described inputs should return a successful message.
post_add_new_user_negative_test_two	{}	{‘message’: ‘Invalid Input’ }	Negative	Lambda function must throw an error message and must not try to add to the USERS table in the database without user details.

- **API → getuser → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
get_user_positive_test_one	{“user_id”: 1}	{ "user": [{‘user_id: 1, ‘first_name’: ‘Jane, ‘second_name: ‘Austen’, ‘email’: ‘jane.austen@gmail.com’, ‘phone’: 2147483647}]}	Positive	Inputting valid described input should return a successful message.
get_user_positive_test_two	{“user_id”: “1”}	{ "user": [{‘user_id: 1, ‘first_name’: ‘Jane, ‘second_name: ‘Austen’, ‘email’: ‘jane.austen@gmail.com’, ‘phone’: 2147483647}]}	Positive	Inputting valid described input should return a successful message.
get_user_negative_test_one	{}	{‘message’: ‘Invalid Input’ }	Negative	Inputting an invalid request should return an error message.
get_user_negative_test_two	{“theater_id”:” 12abc”}	{‘message’: ‘Invalid Input’ }	Negative	Inputting an invalid request should return an error message.
post_add_new_user_negative_test_three	{“user_id”: None}	{‘message’: ‘Invalid Input’}	Negative	Inputting a null user_id should return an error message.

- **API → ListOfCities → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_cities_positive_test_one	{ “user_id”: 1 }	{ "city": [{‘city_code’: 1, ‘city_name’: ‘Seattle’, ‘State’: ‘WA’, ‘Pincode’:‘98101’}, {‘city_code’: 2, ‘city_name’: ‘Kirkland’, ‘State’: ‘WA’, ‘Pincode’:‘98034’}, {city_code’: 3, ‘city_name’: ‘Redmond’, ‘State’: ‘WA’, ‘Pincode’:‘98107’}] }	Positive	Inputting a valid “user_id”, which must be numeric, should return the details of the user.
Get_list_of_cities_negative_test_one	{ “user_id”:”abc”}	{‘message’: ‘Invalid Input’ }	Negative	Lambda function must throw an error message and must not query the database without a userid
Get_list_of_cities_negative_test_two	{ “user_id”:Null }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting Null in place of “user_id” should return an error message.
Get_list_of_cities_negative_test_three	{ “user_id”: “99999999yep” }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting an invalid “user_id” should return an error message.

- **API → ListOfCities → Post**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Post_ Selected_city _positive_ test_one	{ “user_id”: 1, “city_code”: 1 }	{ “city_name”: “Seattle” }	Positive	Inputting a valid “user_id” and “city_code” returns the selected city. Lambda function must update / query the database appropriately and return correct responses
Post_ Selected_city _negative_ test_one	{ “user_id”: 1, “city_code”:” abc”}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
Post_ Selected_city _negative_ test_two	{“user_id”: “abc”, “city_code”: 1}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
Post_ Selected_city _negative_ test_three	{ “user_id”: “abc”, “city_code”:” abc”}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.

- **API → ListOfTheaters → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_theaters_positive_test_one	{ “user_id”: 1, “city_code”: 1}	{ {“theater_id: 1, “theater_name”: “Cinemark”}, {“theater_id: 3, “theater_name”: “Lincoln Memorial Theater”}, {“theater_id: 4, “theater_name”: “Regal”}}	Positive	Inputting “user_id”, and “city_code” returns a list of theaters in the city. Lambda function must query the database appropriately and return a correct response.
Get_list_of_Theaters_Negative_test_one	{ “user_id”: 1, “city_code”: “abc”}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.
Get_list_of_Theaters_Negative_test_two	{ “user_id”: 1, “city_code”: “AB56”}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters. Lambda function must query the database appropriately and return a error response
Get_list_of_Theaters_Negative_test_three	{ “user_id”: “abc”, “city_code”: 1}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user input should not display a list of theaters.

- **API→ ListOfTheaters → Post**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Post_selected_ theater_positiv e_ test_one	{ “user_id”: 1, “theater_id”:2 }	{ “theater_name ”: “Cinemark” }	Positive	Inputting a valid API Key, “user_id”, and “theater_id” returns the selected theater. Lambda function must query / update the database appropriately and return a correct response
Post_selected_ theater_negativ e_ test_one	{“user_id”: 1, “Theater_id”: “abc”}	{‘message’: ‘Invalid Input’}	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.
Post_selected_ theater_negativ e_ test_two	{“user_id”: “abc”, “theater_id”:”ab c”}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.
Post_selected_ theater_negativ e_test_three	{“user_id”: None,“theater_i d_: None}	{‘message’: ‘Invalid Input’}	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.

- **API → ListOfMovies → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_ movies_ positive_test_ one	{ “user_id”: 1, “theater_id”:2 }	{ {“movie_id”:1, “movie_name”: “Top Gun: Maverick”}, {“movie_id: 2, “movie_name”: “Pushpa: The Rise – Part 1”}}	Positive	Inputting a valid API Key, “user_id”, and “theater_id” returns a list of movies in the theater. Lambda function must query the database appropriately and return a correct response
Get_list_of_ movies_ negative_ test_one	{ “user_id”: 1, “theater_id”: “abc”}	{ ‘message’: ‘Invalid Input’}	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.
Get_list_of_ movies_ negative_ test_two	{ “user_id”: “abc”, “theater_id”:1 }	{ ‘message’: ‘Invalid Input’}	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.

Get_list_of_ movies_ negative_ test_three	{“user_id”: None, “theater_id”: None}	{ ‘message’: ‘Invalid Input’}	Negative	Inputting a valid API Key but invalid user input should not display a list of theaters.

Tests for Ticket Booking Lambda Function:

- **API → listofshowdates → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_date s_positive_test_ one	{ “theater_id”: 2, “movie_id”: 1 }	{ “date”: “2022-06-01”, “date”: “2022-06-02”, “date”: “2022-06-03” }	Positive	Inputting a valid API Key, “theater_id”, and “movie_id” returns a list of dates the movie is playing on. Lambda function must query the database appropriately and return a correct response
get_list_of_dates _negative_test_o ne	{ “theater_id”: None, “movie_id”: None }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key and invalid user inputs should display an error message.
get_list_of_dates _negative_test_t wo	{ “theater_id”: 2, “movie_id”: None}	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should display an error message.
get_list_of_dates _negative_test_t hree	{ “theater_id”: None, “movie_id”: 1 }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should display an error message.

- **API → showTimes → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_ times_positive_tes t_one	{ “theater_id”: 2, “movie_id”: 1, “show_date”: “2022-06-01”}	{ “time”: “12:30:00”, “time”: “15:45:00”, “time”: “20:00:00” }	Positive	Inputting a valid API Key, “user_id”, “movie_id”, and date returns a list of times the movie is playing on. Lambda function must query the database appropriately and return a correct response
Get_list_of_ times_negative_tes t_one	{ “theater_id”: 2, “movie_id”: 1}	{‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key and invalid parameters should display an error message.
Get_list_of_ times_negative_tes t_two	{ “theater_id”: 2, “show_date”: “2022-06-01”}	{‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key and invalid parameters should display an error message.
Get_list_of_ times_negative_tes t_three	{ “theater_id”: 2, “movie_id”: None, “show_date”: None}	{‘message’: ‘Invalid Input’ }	Negative	Inputting a valid API Key and invalid parameters should display an error message.

- **API → listOfSeats → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_ Seats_positive_ test_one	{ “show_id”: 6 }	{“seats”: [{ “seat_row”: “B”, “seat_number”: 12 “price”: 25, “seat_type”: “Premium”,	Positive	Inputting a valid API Key, “user_id”, and “show_id” returns a list of movie seats. Lambda function must query the database appropriately and return a correct response
Get_list_of_ seats_negative_test_one	{ 'user_id': 1}	{ ‘message’: “Invalid Input” }	Negative	Inputting a valid API Key and invalid input parameters should display an error.
Get_list_of_ seats_negative_test_two	{‘key’: 6}	{ ‘message’: “Invalid Input” }	Negative	Inputting a valid API Key and invalid input parameters should display an error.
Get_list_of_ seats_negative_test_three	{'show_id': None }	{ ‘message’: “Invalid Input” }	Negative	Inputting a valid API Key and invalid input parameters should display an error.

- **API → addSeats → Post**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Post_selected_Seat_positive_test_one	Params = { 'user_id': 1, 'show_id': 6, 'number_of_seats': 2, 'seats_remaining': 3, 'theater_room_id': 2} json_val={'booked_seats': ["1", "2"]}	{ “total_booking_cost”: 20 }	Positive	Inputting a valid API Key, “user_id”, and “seat_id” returns a “booking_id”. Lambda function must query /update the database appropriately and return a correct response
Post_selected_Seat_negative_test_one	Params = { 'user_id': None, 'show_id': None, 'number_of_seats': None, 'seats_remaining': None , 'theater_room_id': None} json_val={‘booked_seats': ["1", "2"]}	{‘message’ : “Invalid Input” }	Negative	Inputting a valid API Key and invalid input parameters should display an error message.
Post_selected_Seat_negative_test_two	Params = {'user_id': 1, 'show_id': None, 'number_of_seats': 2, 'seats_remaining': 3, 'theater_room_id': 2} json_val={'booked_seats': ["1", "2"]}	{‘message’ : “Invalid Input” }	Negative	Inputting a valid API Key and invalid input parameters should display an error message.
Post_selected_Seat_negative_test_three	Params = {'user_id': None, 'show_id': None, 'number_of_seats': None, 'seats_remaining': None, 'theater_room_id': None} json_val={‘booked_seats': None}	{‘message’ : “Invalid Input” }	Negative	Inputting a valid API Key and invalid input parameters should display an error message.

Tests for Food and Beverages Lambda Function:

- **API → listOfFood → Get**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
get_list_of_food_positive_test_one	{ "user_id": 1, "theater_id": 1 }	{ {"food_id": 1, "food_name": "Butter Popcorn (Small)"}, {"food_id": 2, "food_name": "Butter Popcorn (Medium)"}, {"food_id": 3, "food_name": "Butter Popcorn (Large)"} }	Positive	Inputting a valid API Key, "user_id", and "theater_id" returns a list of food/beverages. Lambda function must query the database appropriately and return a correct response
Get_list_of_food_negative_test_one	{ "user_id": 1, "theater_id": "abc" }	{ 'message': 'Invalid Input' }	Negative	Inputting an invalid user input parameters should return the negative message "Invalid User Inputs".
Get_list_of_food_negative_test_two	{ "user_id": 1, "theater_id": "123b" }	{ 'message': 'Invalid Input' }	Negative	Inputting an invalid user input parameters should return the negative message "Invalid User Inputs".
Get_list_of_food_negative_test_three	{ "user_id": 1, "theater_id": None }	{ 'message': 'Invalid Input' }	Negative	Inputting an invalid user input parameters should return the negative message "Invalid User Inputs".

- **API → listOfFood → Post**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
post_selected_food_positive_test_one	{ “booking_id”: 1, “food_id”: 1, “quantity”: 1 }	{ “message”: “Success }	Positive	Inputting a valid API Key, “food_id”, “booking_id”, and “quantity” returns the name of the food selected. Lambda function must query /update the database appropriately and return a correct response
post_selected_seat_negative_test_one	{ “booking_id”: 1, “food_id”: “abc”, “quantity”: “abc” }	{‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user input parameters should return the negative message “Invalid User Inputs”.
post_selected_seat_negative_test_two	{ “booking_id”: 1, “food_id”: “abc”, “quantity”: 1 }	{‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user input parameters should return the negative message “Invalid User Inputs”.
post_selected_seat_negative_test_three	{ “booking_id”: 1, “food_id”: None, “quantity”: None }	{‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user input parameters should return the negative message “Invalid User Inputs”.

Tests for Payment Processing Lambda Function

- **API → bookingCost → Post**

Name of Test	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Post_booking_Cost_positive_test_one	{ “booking_id”: 8 }	{ “cost”: “140” }	Positive	Inputting a valid API Key and “booking_id” returns the calculated cost of the booking. Lambda function must query /update the database appropriately and return a correct response

Post_booking_ Cost_negative_ test_one	{ “booking_id”: “abc” }	{‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
Post_booking_ Cost_negative_ test_two	{ “booking_id”: “123j” }	{‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
Post_booking_ Cost_negative_ test_three	{ “booking_id”: None }	{‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.

- **API → bookingCost → Get**

Name of Test	API Key	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_booking_ Cost_positive_ test_one	valid	{ “booking_id”: 1, }	{ “total_cost”: “4”, “seat_cost”:4, “food_cost”:0 }	Positive	Inputting a valid API Key and “booking_id” returns the calculated cost of the booking. Lambda function must query the database appropriately and return a correct response
Get_booking_ Cost_negative_ test_one	valid	{ “booking_id”: “abc”, }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
Get_booking_ Cost_negative_ test_two	valid	{ “booking_id”: “123j”, }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
Get_booking_ Cost_negative_ test_three	valid	{ “booking_id”: None, }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.

- **API → listOfTickets → Get**

Name of Test	API Key	Inputs (JSON)	Expected Value (JSON)	Type of Test Case	Description
Get_list_of_tickets_positive_test_one	valid	{ “booking_id”: 1, }	{“city_name: “Kirkland”, “theater_name”: “Cinemark”, “movie_name”: “Pushpa: The Rise - Part 1”, “show_date”: “Thu Jun 02”, “start_time”: “12:30:00”, “end_time”: “15:29:00”, “room_name”: “Screen Room C”, “cost”: “4”}	Positive	Inputting a valid API Key and “booking_id” returns the calculated cost of the booking. Lambda function must query the database appropriately and return a correct response
get_list_of_tickets_negative_test_one	valid	{ “booking_id”: “abc”, }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
get_list_of_tickets_negative_test_two	valid	{ “booking_id”: “123j”, }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.
get_list_of_tickets_negative_test_three	valid	{ “booking_id”: None, }	{ ‘message’: ‘Invalid Input’ }	Negative	Inputting invalid user inputs should return the negative message “Invalid User Inputs”.

Tests for Front End Client:
We will execute this test case manually once we finish programming our Front End HTML, Node.js, and CSS files. To do so, the client device would need to have browser configured with the ability to connect to Amazon CORS gateway. To run the client code, we would open the index.html file in the respective browser. Then, right click on the page to inspect the page. This will allow us to view the requests sent to the API and responses received from it. The HTML page prompts the user, asking for their manual inputs. It will call the API endpoints as needed and provide the inputs given by the user. Then, it will use the responses of the API to drive the next set of user interactions through further HTML, JS scripts in the same folder and API calls. This application logic will continue until the user has gotten their booked movie tickets.

Description of Test	**Execution Steps**	**Expected Value**
Web page must accept appropriate inputs from the user to search movies in Seattle's Cinemark Theater (integration of the above SearchMoviesServer API positive test cases). It must send correct API requests with both • valid API key • user inputs in the query string. It must send it to the SearchMoviesServer lambda function. It must provide appropriate response from the lambda function to the user.	- Open the client HTML code in a browser. - Input valid details into the text boxes or choose from the drop-down menu. - Click the button to submit responses. - This will send an API request with the appropriate URL, headers, parameters, and keys.	Response is displayed to the user in the form of moving on to the next page. Looking at the logs in the browser through right-clicking "inspect" web page should tell us whether the request was received successfully or failed in the middle of execution. The response of the API is also logged here.
Web page must accept appropriate inputs from the user to select and lock a seat for "Top Gun" movie in Seattle's Cinemark Theater (integration of the above TicketBookingServer API positive test cases). It must send correct API requests with both • valid API key • user inputs in the query string. It must send it to the TicketBookingServer lambda function. It must provide appropriate response from the lambda function to the user.	- Open the client HTML code in a browser. - Input valid details into the text boxes or choose from the drop-down menu. - Click the button to submit responses. - This will send an API request with the appropriate URL, headers, parameters, and keys.	Response is displayed to the user in the form of moving on to the next page. Looking at the logs in the browser through right-clicking "inspect" web page should tell us whether the request was received successfully or failed in the middle of execution. The response of the API is also logged here.

Web page must accept appropriate inputs from the user to book desired food and beverages in Seattle's Cinemark Theater (integration of the above FoodBookingServer API positive test cases). It must send correct API requests with both • valid API key • user inputs in the query string. It must send it to the FoodBookingServer lambda function. It must provide appropriate response from the lambda function to the user.	- Open the client HTML code in a browser. - Input valid details into the text boxes or choose from the drop-down menu. - Click the button to submit responses. - This will send an API request with the appropriate URL, headers, parameters, and keys.	Response is displayed to the user in the form of moving on to the next page. Looking at the logs in the browser through right-clicking "inspect" web page should tell us whether the request was received successfully or failed in the middle of execution. The response of the API is also logged here.
Web page must accept appropriate inputs from the user to process the payment and finish booking movie tickets based on the previous details provided by the user. It must send correct API requests with both • valid API key • user inputs in the query string. It must send it to the PaymentProcessingServer lambda function. It must provide appropriate response from the lambda function to the user.	- Open the client HTML code in a browser. - Input valid details into the text boxes or choose from the drop-down menu. - Click the button to submit responses. - This will send an API request with the appropriate URL, headers, parameters, and keys.	Response is displayed to the user in the form of moving on to the next page. Looking at the logs in the browser through right-clicking "inspect" web page should tell us whether the request was received successfully or failed in the middle of execution. The response of the API is also logged here.

References

1. Unittest - Unit Testing Framework. unittest - Unit Testing Framework - Python 3.10.4 documentation. (n.d.). Retrieved May 9, 2022, from https://docs.python.org/3/library/unittest.html
2. Request library - https://docs.python-requests.org/en/latest
3. https://stackabuse.com/the-python-requests-module/