**Assumptions**

The following assumptions are made while building test cases for the Ticket System -

Seating arrangement at the high demand performance venue consists 1 auditorium with 9 rows of 33 seats each.

The API’s communicate with the server using JSON formatted request and response. Seating and customer information will be stored and retrieved from a data source. A call to retrieve or store data to the data store will be performed only after the parameters in the request are validated.

The test cases are created based on the following design of the system -

**Design 1:** Since the interface provided in the exercise does not specify a show id, we will assume the show runs only on 1 day. For simplicity the show start date will be on 01/01/2018 at 4:00 PM.

**Design 2:** Since the number of tickets that will be available for purchased in Design 1, will not help generate significant load on the system, in Design 2 the show is assumed to run for 30 days. Each day has 2 scheduled shows. The maximum number of tickets that can be sold at this venue will be (#seats)\*(#days)\*(#shows)=297 \*30 \*2=17,820.

**All test cases are built based on Design 2. However, by varying test configurations, the tests case be modified to run Design 1 cases also.**

The services provided by Ticket Service system consists of the following 3 API modules:

1> Discovery API Module: Finds the number of seats that are available at the venue and returns the number of available seats that are neither held nor reserved. This API will be referred to as *numSeatsAvailable* for the rest of the document.

A simple call to this API will look like

<https://ticketsystem.com/numSeatsAvailable>

JSON Response:

{"status": "OK", "mobile\_url": "", "hash": "a7fd6484f414f467aaa56d4564fd06c1", "url": "https:/[ticketsystem.com/numSeatsAvailable](https://ticketsystem.com/numSeatsAvailable)/", "Date": "2017-12-27T11:11:42", "available": 60}

Where -

status: defines http status

mobile\_url: loads a mobile site if user agent is identified as mobile

hash:hash value for cookies

url: referrer to the site: used for campaigns

date: Day on which the request was made

Available: seats available

2> Find and Hold API Module: Finds and holds the best available seats on behalf of a customer. Hold on each seat will expire within 300 seconds, where 300 seconds is the think time for payment detail entry. When this API is called, seats are temporarily added to the Hold status. The total number of seats available in the system is reduced by the number of seats held. The hold on the seat will be released if the user does not call the Reservation API within 300 seconds. Last seat that can be bought for the show will be 2 hours before the show. User can hold seats for only 1 show at a given time. This API will be referred to as *findAndHoldSeats* for the rest of the document.

A simple call to this API will look like:

[https://ticketsystem.com/](https://ticketsystem.com/numSeatsAvailable)*findAndHoldSeats?seats=15&customerEmail=abc@abc.com*

JSON Response:

{"status": "OK", "hash": "a7fd6484f414f467aaa56d4564fd06c1", "Date": "2017-12-27T11:11:42", “Confirmation”: “hold”, “Settime”:”300”, “*customerEmail* ”:”[abc@abc.com](mailto:abc@abc.com)”, “seats”:”15”, “holdId”:”4”}

Where -

status: defines http status

hash:hash value for cookies

Data: day on which the hold was requested

Confirmation: hold/reserve status of the request

settime: think time

customerEmail: identifier for customer

Seats: number of seats held

holdId: id for the successful hold request

3> Reservation API Module: Reserve and commit a specific group of held seats for a customer. Last seat that can be booked for the show will be 2 hours before the show. seats that are purchased cannot be cancelled or refunded. seats must be held before being reserved, i.e you cannot directly reserve the seat without a reservation. When Reservation API is called, seats move from hold status to reserve status. This gives the user sufficient time to enter details to purchase the seat. Each user is allowed to reserve seats only once. Unique user is identified using the users email address. User can reserve seats for only 1 show at a time. This API will be referred to as *reserveSeats* for the rest of the document.

A simple call to this API will look like:

[https://ticketsystem.com/](https://ticketsystem.com/numSeatsAvailable)*reserveSeats?holdId=4&customerEmail=abc@abc.com*

JSON Response:

{"status": "OK", "hash": "a7fd6484f414f467aaa56d4564fd06c1", "Date": "2017-12-27T11:11:42", “Confirmation”: “reserve”, “Settime”:”600”, “*customerEmail*”:”[abc@abc.com](mailto:abc@abc.com)”, “Seats”:”15”.”ReservationId”:”res4”, “holdId”:”4”}

Where -

status: defines http status

hash:hash value for cookies

Data: day on which the hold was requested

Confirmation: hold/reserve status of the request

settime: think time

customerEmail: identifier for customer

Seats: number of seats held

ReservationId: id for the successful reserve request

holdId: id for the successful hold request

*Assumptions made specifically for non functional tests*

1. Tester is familiar with JMeter
2. The terminology used in the test plan and test cases are specific to JMeter.
3. The following JMeter Plugins are installed on the test system: Ultimate Thread Group
4. JMeter Test responses will be saved to a file. The file name will use the following format: <Thread Group Name>.<date>
5. Application Performance Monitoring tools like DataDog or NewRelic will be used to measure memory leaks, CPU,, database performance during endurance testing
6. Results of Default Test will serve as a baseline for all tests
7. API names used while writing test cases: *numSeatsAvailable*, *findAndHoldSeats*, *reserveSeats*
8. Based on user trend and historic data, the system is assumed to expect high load between 11:00 AM to 2:00 PM on all days. Minimum load is expected between 2:00 AM to 4:00 AM
9. **Design 1:** At the venue, we have 33\*9 = 297 seats. Since the interface provided in the exercise does not specify a show id, we will assume the show runs only on 1 day. For simplicity the show start date will be on 01/01/2018 at 4:00 PM. The maximum number of tickets that can be sold at this venue will be (#seats)\*(#days)\*(#shows)=297 \*1 \*1=297. The maximum number of tickets that can be purchased per reservation will be 20 and the minimum number of tickets that can be purchased per reservation will be 1. **Design 2:** At the venue, we have 33\*9 = 297 seats. The show runs for 30 days. Each day has 2 scheduled shows. The maximum number of tickets that can be sold at this venue will be (#seats)\*(#days)\*(#shows)=297 \*30 \*2=17,820. The maximum number of tickets that can be purchased per reservation will be 20 and the minimum number of tickets that can be purchased per reservation will be 1.
10. **Design 1:** Since this is high demand show, we are making an assumption that the maximum number of requests *numSeatsAvailable* API will be limited to (max #tickets)+10% ~ 326. **Design 2:** Since this is high demand show, we are making an assumption that the maximum number of requests *numSeatsAvailable* API will be limited to (max #tickets)+10% ~ 20K.
11. Response size of *numSeatsAvailable* API is estimated by considering the bytes returned by the integer returned by the API and the header information. In this document we estimate this as 100bytes.
12. Response size of *findAndHoldSeats* and *reserveSeats* API will be estimated as 200bytes, since these 2 API’s return an object containing reservation information
13. The following terminology & KPI’s are used for determining the performance and readiness of the system -

* Response time: Total time to send a request and get a response.
* Wait time:Time taken to receive the first byte after a request is sent.
* Average load time: The average amount of time it takes to deliver every request
* Peak response time:This is the measurement of the longest amount of time it takes to fulfill a request.
* Error rate: Percentage of requests resulting in errors compared to all requests.
* Concurrent users: How many concurrent users are supported.
* Requests per second: How many requests are handled.
* Transactions rate: Measures the total numbers of successful or unsuccessful requests.
* Throughput: The amount of bandwidth used during the test measured in kbps

1. All tests are run from the same geographical location unless specified

*Standard List of Parameter Validation*

These checks will be performed on all parameters unless specified in the tests:

* Check all fields for NaN or Null
* Number of seats should not be negative or greater than Max
* Empty string will be allowed on fields according to requirements specified
* Perform date field validation, convert to UTC if not already in UTC, check against current date & time
* Encode the values in response if it contain special characters
* Ensure double encoding is prevented
* Hash values to hide data if in requirement or contains PII
* Unhash to be able to decode data before checking the database
* Values passed are in correct order to the API
* Parameters are case sensitive parameter
* Number of arguments check sent to API should match signature
* Backward compatibility check - if the original API accepted 9 parameters in version 1, and in version 2 design changes to accept 10 parameters, and the parent script is coded to send only 9 parameters, the API should be backward compatible
* Log message should be written for missing parameter, user friendly message should be sent to the user, if request cannot be complete
* Logs should be written for each request and response
* Extended logs should be written only in debug mode o> standard email address validation

*Assumptions made specifically for testing with call to this API using frontend*

A wrapper script with business logic will be *responsible for calling these API’s.* Responsibilities of this script is not in scope, but should include tests for -

1. Check user environment information - browser, IP, cookies etc
2. Limiting the number of refreshes on the API - check for DOS, bots, fraud
3. Collect user demographics from Google tracking API, essential during promotions
4. Final price calculation, promo code logic
5. Credit card validation logic
6. Customer billing address validation logic