Testing Report

for

CZ3003 Hydra-Defence

Version 1.0 approved

**Prepared by Mohamed Shafiq Bin Peer Mohamed**

**Heather Chew**

**Yeong Wei Xian**

**Team Hydra**

**13/4/2021**

Contents

[1. Introduction 1](#_Toc69242421)

[1.1 Testing strategies adopted 1](#_Toc69242422)

[1.1.1 Black Box Testing 1](#_Toc69242423)

[1.1.2 White Box Testing 1](#_Toc69242424)

[1.1.3 Unit Testing 1](#_Toc69242425)

[1.1.4 Load Testing 1](#_Toc69242426)

[1.3 Testing techniques adopted 2](#_Toc69242427)

[1.4 Testing Tools Adopted 2](#_Toc69242428)

[1.4.1 Unity Test Framework (Unit Testing Tool) 2](#_Toc69242429)

[1.4.2 Postman (API Endpoint Testing Tool) 2](#_Toc69242430)

[1.4.3 Locust (Automatic Performance Testing Tool) 2](#_Toc69242431)

[2. Black Box Results 4](#_Toc69242432)

[2.1 Login Test For Teacher and Student 4](#_Toc69242433)

[2.2 Registration Test 4](#_Toc69242434)

[2.3 Student Mode Test 5](#_Toc69242435)

[2.3.1 Story Mode 5](#_Toc69242436)

[2.3.2 PVP 6](#_Toc69242437)

[2.3.3 Leader board 7](#_Toc69242438)

[2.3.4 Assignment 7](#_Toc69242439)

[2.4 Teacher Mode Test 7](#_Toc69242440)

[2.4.1 View Reports and Analytics 7](#_Toc69242441)

[2.4.2 Login to Assignment System 8](#_Toc69242442)

[2.4.3 Create Assignment 9](#_Toc69242443)

[3. White Box Report 11](#_Toc69242444)

[3.1 Account Creation 11](#_Toc69242445)

[3.2 Account Login 12](#_Toc69242446)

[3.3 Assignment Creation 13](#_Toc69242447)

[4. Unity Unit Testing 14](#_Toc69242448)

[5. Load Test 15](#_Toc69242449)

[5.1 User Capacity Analysis 15](#_Toc69242450)

[5.2 Preamble 15](#_Toc69242451)

[5.3 Test Results 16](#_Toc69242452)

[5.4 Visualizations 17](#_Toc69242453)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Mohamed Shafiq | 13/04/2021 | Initial Creation | 1.0 |
|  |  |  |  |

# Introduction

## Testing Strategies Adopted

Below underlined are the testing strategies adopted and the reasons we chose to adopt these specific methods.

### Black Box Testing

We conducted Black Box Testing as it helps to find the gaps in functionality, usability and other features of our system. This gives us an overview of our software performance and its output. In the event if there is a case failure, we can immediately rectify the problem. It helps to reduce the risk of software failures at the user’s end.

### White Box Testing

We conducted White Box Testing to ensure that every path through the System Under Test has been identified and tested.

### Unit Testing

We have performed unit testing to validate that each unit of the software code performs as expected. With unit testing, we can easily refactor or re-write our code if needed.

### Load Testing

We conducted load testing once the server was ready and deployed to ensure that we understood, analyzed, and fixed errors, bugs, and bottlenecks that could occur when the server experienced real-world traffic.

This also enabled us to determine whether our server was capable of living up to the non-functional requirements set out by the client. In this case, with NTU being the client, it was determined that the server would likely have to service about 150 students regularly throughout a semester.

While it was informally known that load was likely to be distributed as opposed to sustained, how the load would likely be distributed was unknown. As a result, we decided to test the system against the assumption that traffic would be sustained and recorded our results accordingly. Refer to the Load Test Report (**page 4 for now**) for more information on our methodology and results.

## Testing Techniques Adopted

**Functional Testing Techniques**

Black and White Box Testing

Unit Testing

Integration Testing

**Non-functional Testing Techniques**

Load & Stress Testing

## Testing Tools Adopted

### Unity Test Framework (Unit Testing Tool)

Unit testing is an automated testing framework that ensures that functionalities are not broken when changes are made to the code. This helps to save time as the developers are not required to explore every part of the game when changes are made to ensure that there are no errors. This is especially true when there are numerous scenes and scripts.

### Postman (API Endpoint Testing Tool)

**Postman** is an interactive and automatic **tool** for verifying the APIs of your project. **Postman** is a Google Chrome app for interacting with HTTP APIs. It presents you with a friendly GUI for constructing requests and reading responses. It works on the backend, and makes sure that each API is working as intended.

*Source:* [*https://www.axelerant.com/resources/team-blog/api-testing-with-postman*](https://www.axelerant.com/resources/team-blog/api-testing-with-postman)

Using Postman allowed our back-end development team to easily test any new APIs that were created and store simple smoke tests in specific folders to allow for easy functionality verification before each deployment.

Postman also allowed the front-end development team to easily check the expected output of an API call and verify if their application behaved according to expectations. This way, they could easily determine if the problem lied at the game-level or the back-end level.

### Locust (Automatic Performance Testing Tool)

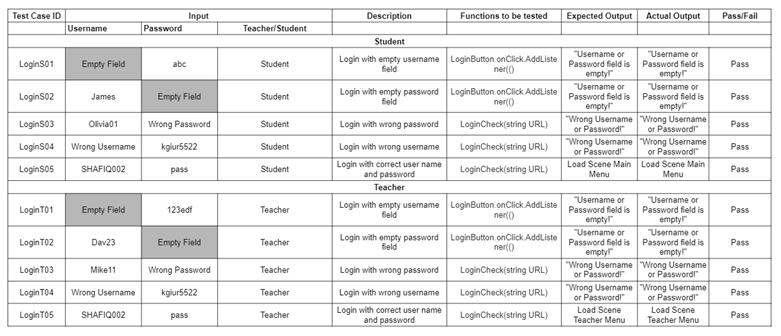
**Locust** is an open-source load-testing tool written in **Python**. It lets you write tests against your web application which mimic your user's behavior, and then run the tests at scale to help find bottlenecks or other performance issues.

*Source: https://www.promptworks.com/blog/load-testing-with-locust/*

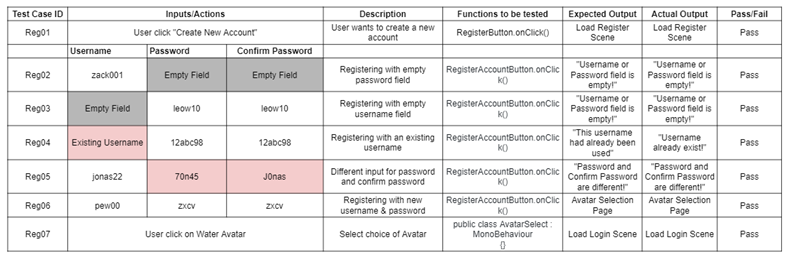
Locust was used as an alternative to JMeter Apache as it allowed for slightly more granular control while giving aesthetically pleasing and user-friendly results.

# Black Box Results

## Login Test For Teacher and Student

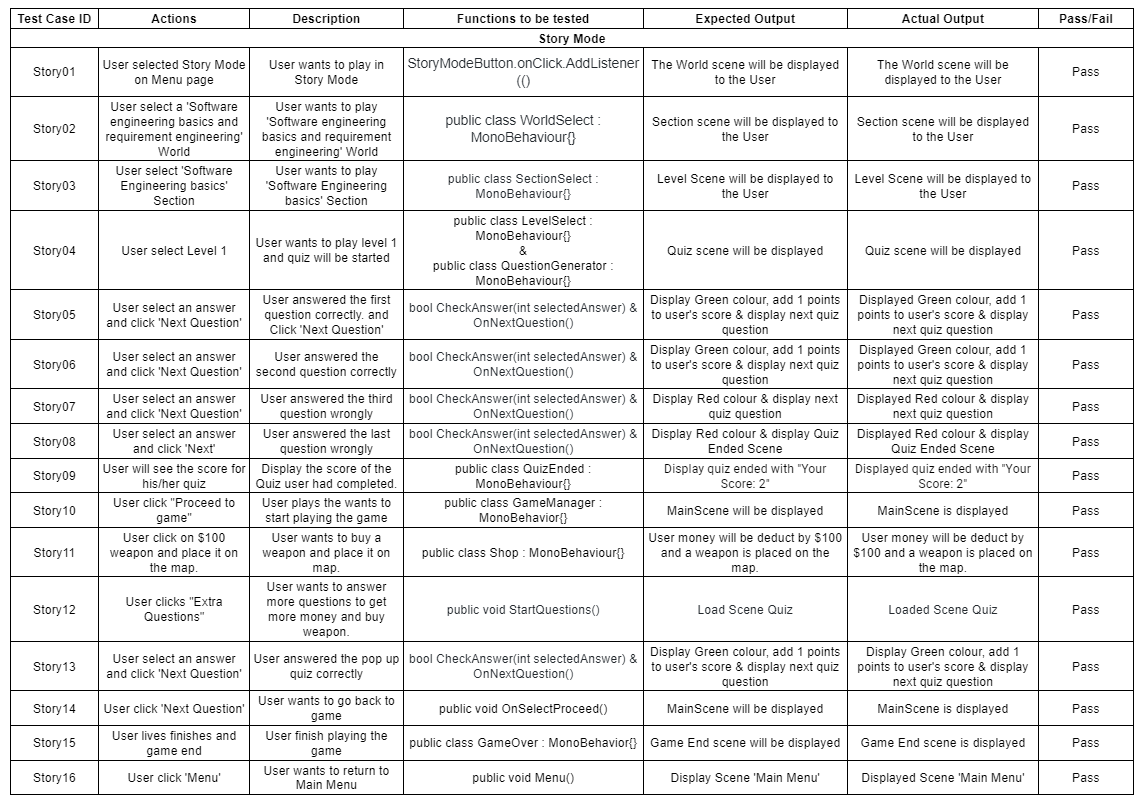


## Registration Test

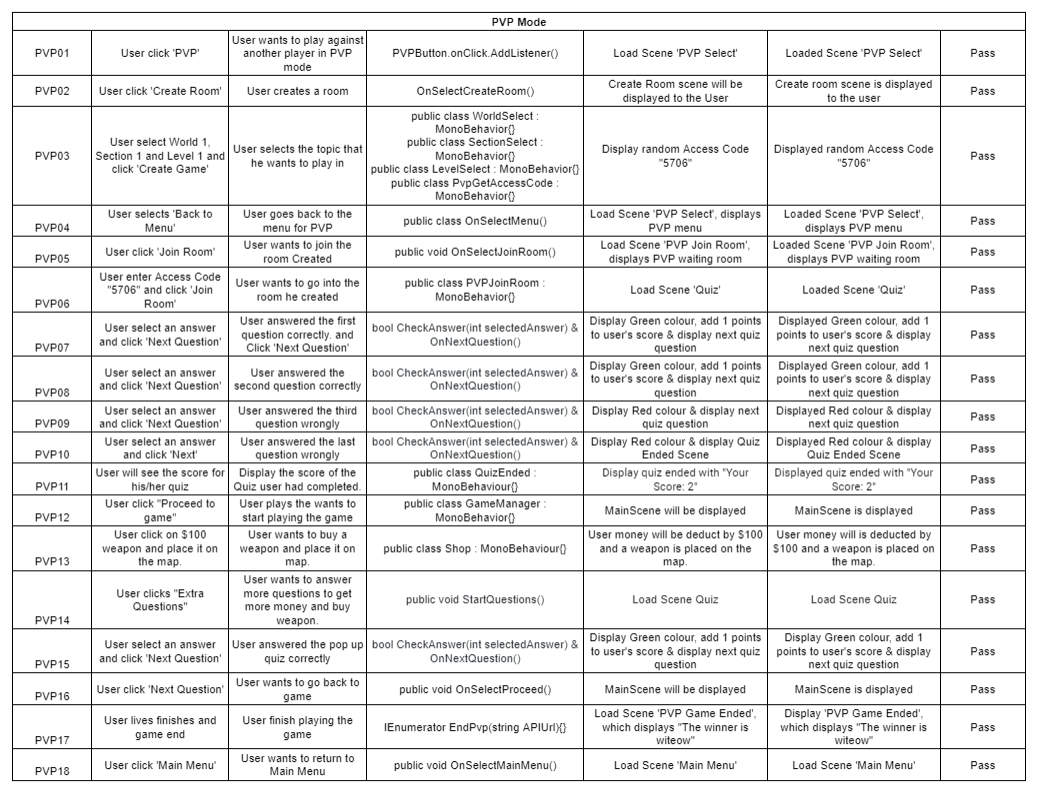


## Student Mode Test

### Story Mode



### PVP

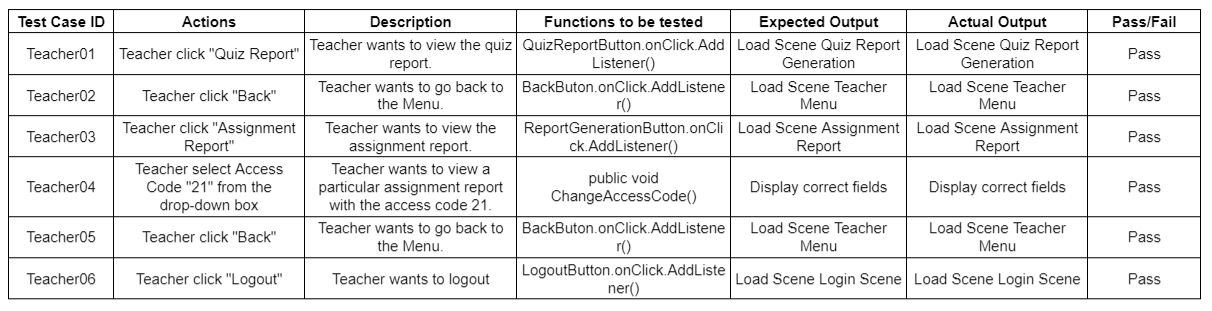


### Leader board

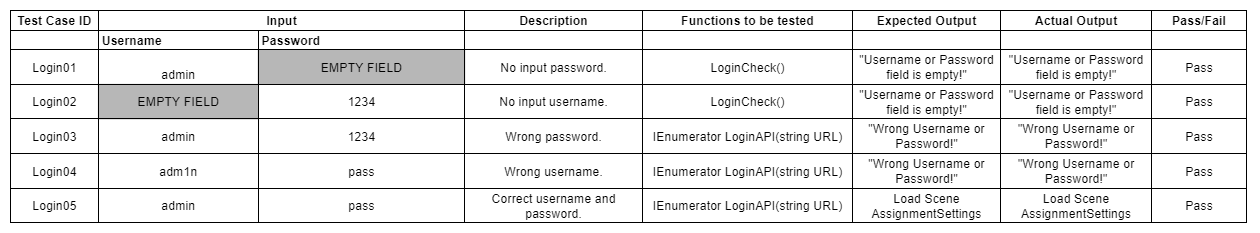
### Assignment

## Teacher Mode Test

### View Reports and Analytics

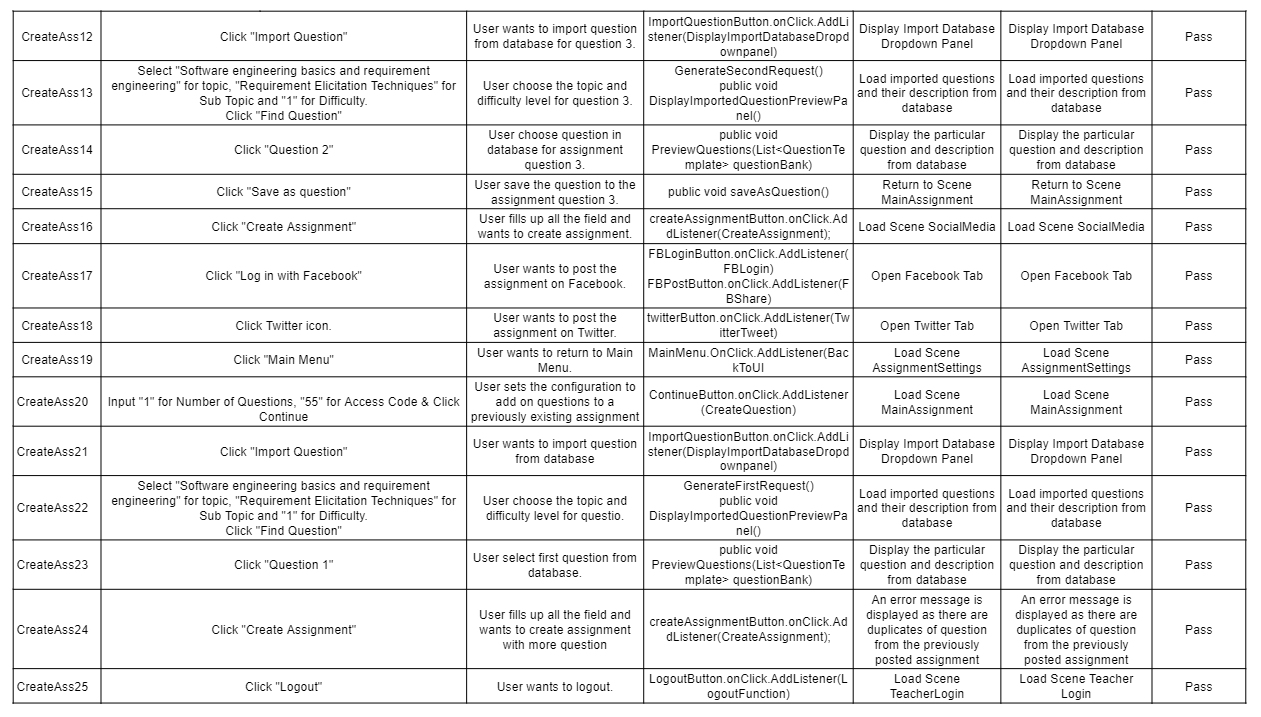
****

### Login to Assignment System

****

### Create Assignment

#### Table 1

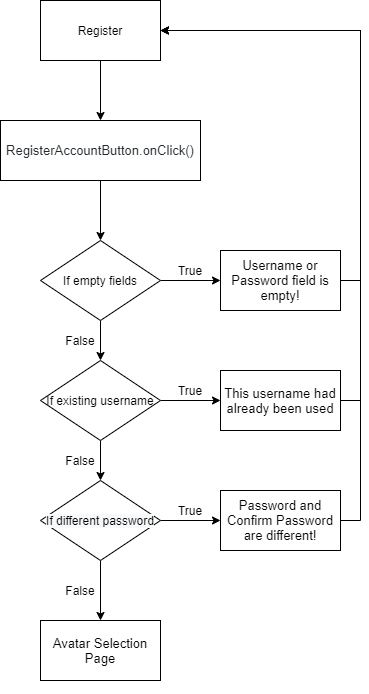


#### Table 2

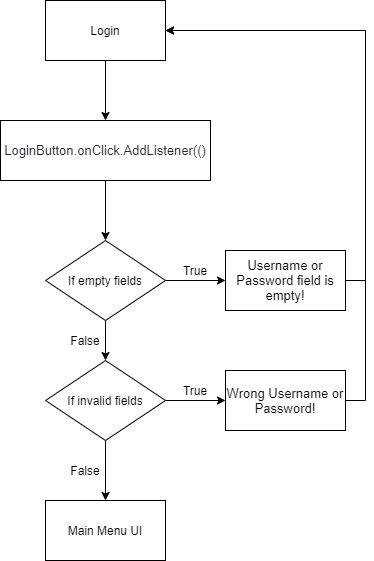
# White Box Report

White Box Testing enables us to find flaws in the logic of the system. We are able to efficiently realise the parts that have errors or problems. There must be a limitation set for user inputs, such that only those inputs that we are looking for are accepted, while other irrelevant inputs will be rejected accordingly. Any empty or invalid field entered will result in an error message to prompt the user for relevant inputs.

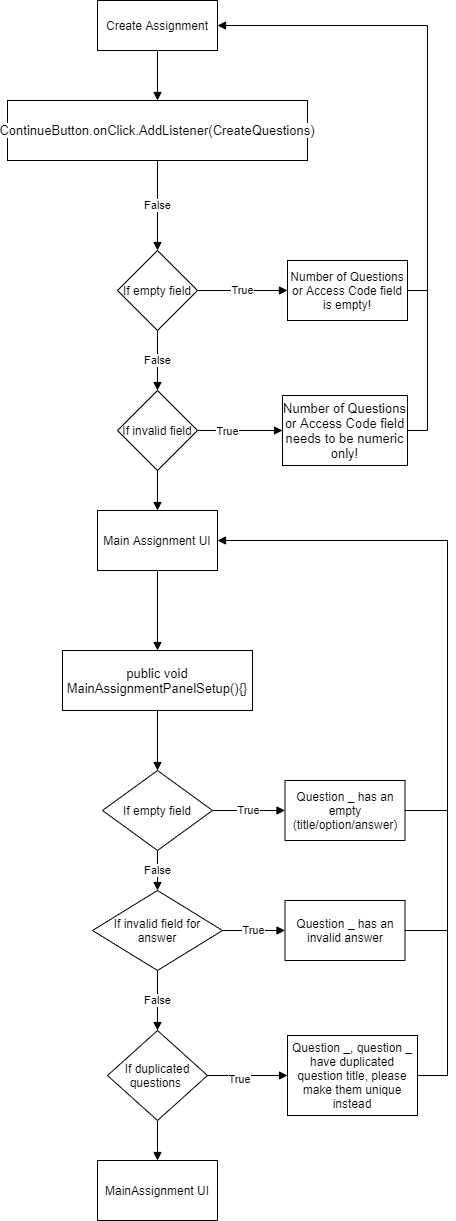
## Account Creation



## Account Login



## Assignment Creation



# Unity Unit Testing

The development team had decided to handle most of the logic in the backend, as within Unity, there are many interdependencies within scenes and scripts. This allowed us to minimise the potential errors we would have to face when developing the game in Unity.

Therefore, we have implemented unit tests to ensure that core functionalities are not broken when the program is updated and when the different subsystems are integrated together.

All of our classes in the unity program are inherited from the Monobehaviour class. The Monobehaviour class does not allow for instantiation and thus we were limited in our capabilities.

One possible workaround to this issue is to create an interface for each class and subsequently mock the interfaces using NSubstitute (https://technicallyshane.com/2019/10/29/unit-testing-unity.html). Since our other test methods have sufficiently tested our program, we have decided not to pursue this method.

|  |  |  |
| --- | --- | --- |
| Subsystem | Number of tests | Pass rate |
| Quiz | 2 | 100% |
| Leaderboard | 1 | 100% |
| Game | 2 | 100% |

# Load Test

## User Capacity Analysis

*The User Capacity Analysis calculates how many users the application can support based on the configured performance goals.*

**Estimated User Capacity** : Between 100 to 150

**Maximum Users Analyzed** : 300

**Performance Testing Range:** Between 50 to 300

## Preamble

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Functions Tested** | **API URL Used** | **Justification for choosing this function** |
| 1 | Start-up tester | https://223.25.69.254:10002/ | The Start-up tester function behaves as the ideal API call our server can service as it has next to no internal logic, which can help show us the performance ceiling we can expect with the current server configuration. |
| 2 | Retrieve Leaderboard | https://223.25.69.254:10002/get\_leaderboard/username=SHAFIQ002 | This function will act as a proxy for any average API Call that seeks information from the server and backend, with no data being written. |
| 3 | Update Quiz Performance | https://223.25.69.254:10002/update\_performance/username=SHAFIQ002&world=1&section=1&no\_of\_correct=3 | This function will act as a proxy for any average API Call that seeks and writes information from the server and backend, with no data being written. This API will help us identify the performance floor we can expect with the current server configuration. |

While all the tests were conducted together, note that any tests conducted with users = 300 were the equivalent of a stress test. Thus, both load and stress tests were conducted during the testing phase of deploying this application.

## Test Results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Start-Up Tester** | | | | | | | | |
| **Max Users** | **# of Requests** | **# of Fails** | **Median** | **90%ile** | **Mean** | **Min** | **Max** | **Failures/s** |
| 50 | 11949 | 0 | 950 | 1200 | 961 | 16 | 1235 | 0 |
| 100 | 38991 | 0 | 1900 | 1900 | 1532 | 16 | 2313 | 0 |
| 150 | 25121 | 0 | 3300 | 5200 | 3519 | 203 | 2313 | 0 |
| 200 | 13462 | 5 | 2500 | 9100 | 3801 | 31 | 21032 | 0 |
| 250 | 11957 | 36 | 2500 | 9300 | 4554 | 31 | 21079 | 0.2 |
| 300 | 14379 | 475 | 3600 | 10000 | 5821 | 125 | 21109 | 3.9 |
| **Leaderboard** | | | | | | | | |
| **Max Users** | **# of Requests** | **# of Fails** | **Median** | **90%ile** | **Mean** | **Min** | **Max** | **Failures/s** |
| 50 | 12644 | 0 | 1200 | 1200 | 1159 | 141 | 1563 | 0 |
| 100 | 13296 | 0 | 2300 | 2400 | 2294 | 156 | 3328 | 0 |
| 150 | 15774 | 4 | 3000 | 5500 | 3462 | 15 | 21062 | 0 |
| 200 | 13001 | 360 | 3800 | 5900 | 4611 | 31 | 32734 | 0.7 |
| 250 | 14136 | 141 | 4100 | 9800 | 5620 | 31 | 21110 | 1.4 |
| 300 | 13102 | 696 | 5700 | 10000 | 6322 | 16 | 21094 | 6 |
| **Update Quiz Performance** | | | | | | | | |
| **Max Users** | **# of Requests** | **# of Fails** | **Median** | **90%ile** | **Mean** | **Min** | **Max** | **Failures/s** |
| 50 | 11204 | 0 | 2000 | 2100 | 2050 | 2000 | 2110 | 0 |
| 100 | 23120 | 0 | 2000 | 2100 | 2050 | 2000 | 2110 | 0 |
| 150 | 11619 | 48 | 2000 | 2100 | 2048 | 2000 | 2125 | 0.1 |
| 200 | 5000 | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 250 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 300 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

**NOTE:** A NULL cell refers to a test case that led to unstable server behavior that made testing inaccurate and too difficult to conduct.

## Visualizations

**NOTE:** NULL Values are presented as exponentially increasing fault cases, instead of being omitted from visualization