TEST FRAMEWORK

Requirements

A Web Based System to Support Testing Multiple Program Modules

David Howick, Miriam Farrington, Mudit Vats,
Elona Vabishchevich, Jeffrey Alexovich

Table of Contents

Preface 2

1 Introduction 3

1.1 System Overview 3

1.2 System Users 5

1.3 User Roles and Accessibility 5

1.4 System Usability 6

Glossary 7

2. Requirements 8

2.1 User Requirement 1: Test System Capability 8

2.2 User Requirement 2: Logging and Results Viewing 8

2.3 User Requirement 3: Concurrent Test Execution 9

2.4 User Requirement 4: Platform Support 9

2.5 User Requirement 5: Error Handling 10

2.6 User Requirement 6: User Interaction 10

2.7 User Requirement 7: High Availability 11

3 Availability and Business Continuity Requirements 12

3.1 Availability Requirement 1: Continuous System Uptime 12

3.2 Availability Requirement 2: Recovery Time 12

3.3 Availability Requirement 3: High Availability 12

4 Constraints 13

4.1 Technical Constraints 13

4.2 Operational Constraints 13

4.3 Business Constraints 13

5 System Models 14

5.1 Use Case Models and/or Scenarios 14

5.2 Sequence Diagram 16

5.3 Class Diagram 17

Preface

Developing large scale software consisting of multiple packages requires frequent testing. If the software has complex features, interactions, or other complex system interfaces (APIs), then we want to build it incrementally and test each increment. The development team first designs and implements a very basic core with a small number of packages, then adds features one-at-a-time by adding new packages, or adding a few lines of code to an existing package. Each time new functionality is added, the application is built and tested. That way, if additions break existing code, the developers know where to look, e.g., in the newly added few lines of code. This Test Framework application will allow the development team(s) to use this incremental approach more efficiently.

The Test Framework application will allow the development team(s) and other users of the system to define many small tests, each of which run with exception handling and results logging. The goal of the Test Framework application is to do that without proliferating code with many try-catch blocks, debug statements, assertions, and abundance of verbose logging statements.

The Test Framework application will provide test results via logging as well as in saved test results files. So the logging mechanism will provide for several levels of logging. One level is just for quick basic test results and information about the test, such as how long the test took to complete. Another level will be verbose messages that dive into the details of a particular test and why that particular test failed.

The Test Framework application will be easy to use as it can be accessed by the user's web browser. The system itself is cloud hosted in multiple regions so performance is always superb.

1 Introduction

There is a desire for a web based, cloud hosted solution that can test multiple program modules or blocks of program code simultaneously. This system will also have multiple methods for accessing the test module (dynamic link library, XML file, JSON file, etc.) and for delivering test results to the Test Framework application via logging and test results files.

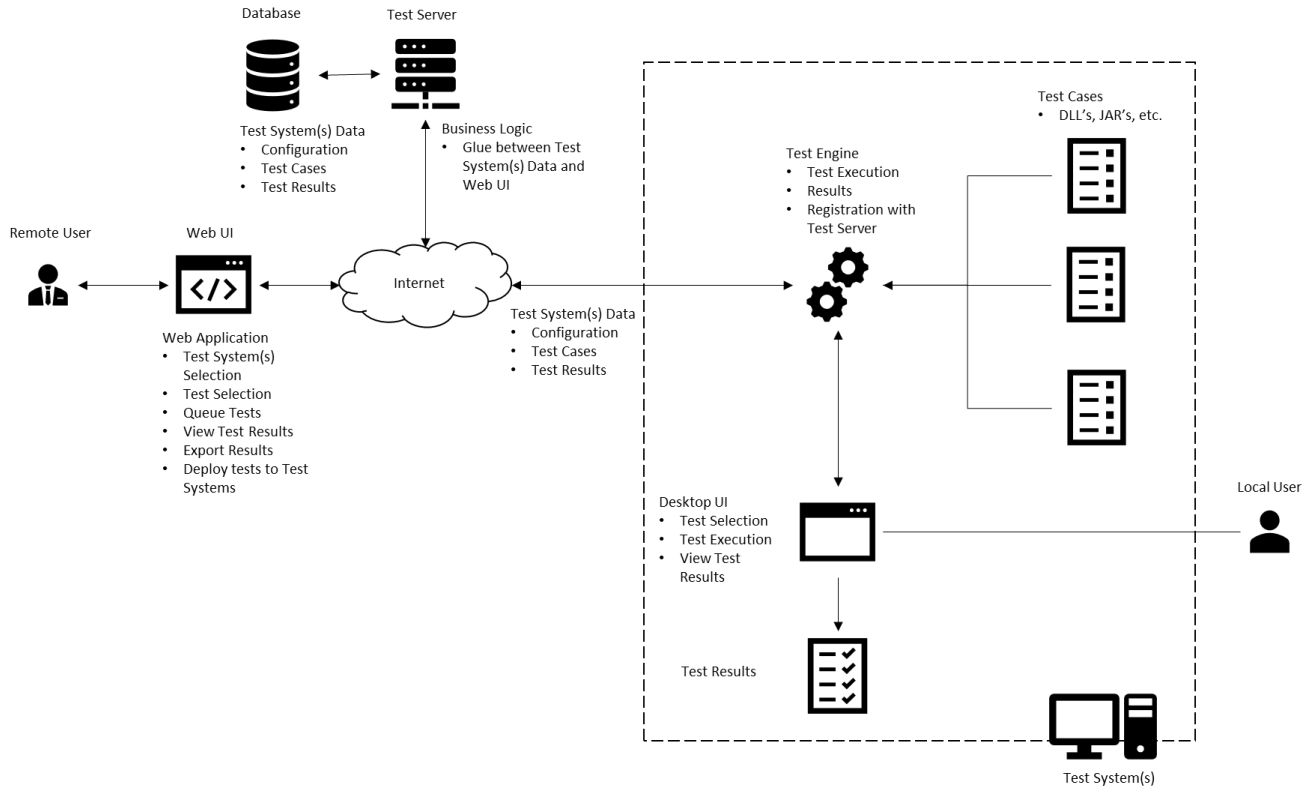
In addition, there is the need to be able to test with multiple languages (C++, C#, Java, Python) and to add additional program language support to the application as needed for expandability.

Further, scalability is a concern as there will be multiple teams (multiple developers, engineers, testers, - users of the system) that will use the system at the same time so the system needs to be scalable, to support massive parallel processing/testing and support multiple concurrent users.

Because of the scalability issue, massive parallel testing that can be going on at any given time, the concurrency of tests and concurrency of users, system performance is also a prime concern. The system shall be hosted on cloud platforms and in multiple regions to address both performance requirements as well as system availability and business continuity concerns.

1.1 System Overview

The following diagram shows the overall system architecture of the Test Framework:



The key components are:

Users

- Local User
- Remote User

Web UI

Test Server

- Test Server
- Database

Test System

- Test Engine
- Test Cases
- Desktop UI
- Test Results

In the context of this diagram Users refer to how users are interacting with the system and, not necessarily, the System Users defined later in this document. There are two types of interaction: Local and Remote. As the name implies, a local user will access the Test Framework on the same system that is being tested. This user interface is a native Desktop UI and can only interact with that one test system.

A Remote User, on the other hand, can interact with multiple test systems typically at a location other than where the test system resides. For example, in an enterprise setting, and administrator may wish to execute, view results or manage test cases for multiple test systems across multiple geographies. It would be impractical for him/her to drive to each location to perform these actions (cost and time prohibitive). Instead, the Administrator can use the Web UI to access the Test Systems and Test Systems Data for all systems he/she wishes to manage.

The Test System contains the Test Engine, Test Cases, Desktop UI and Test Results. The heart of the testing occurs via the Test Engine. The Test Engine is responsible for reading test configuration data, test cases and providing test results. In the local case, the test selection and results can be viewed by the Desktop UI. In the Remote case, the Test Engine registers Test Data with the Test Server to make the Test System and Test Cases available for remote management and execution. In this case, all interaction is accomplished via the Web UI.

The Test Server serves at the business logic between the Web UI and Test Systems. It interacts with the Test Database to store Test System Data for each test system and corresponding test configuration, test cases and latest test execution results.

1.2 System Users

There are 9 types of users for the system. They are:

1. Software Developers. These users are the developers of the software. They create designs for new features, build and run tests for each new feature.
2. Programmer Analysts. These users are also developers of the software, but usually working from a design document with strict guidance as to what is produced and tested.
3. Software Architects. These users are the designers of the overall application software, the structure of the application code (modules), the classes, dynamic behavior, and help elicit and define requirements.
4. Software Engineers. These users are the technical leads. In some organizations they are considered the software developers and in others they are the architects of the system.
5. System Architects. These users are responsible for the architecture and structure of the entire system, including the software, the hardware it runs on, the infrastructure it uses, the processes it follows, and the other systems the solution interfaces with.
6. System Engineers. These users are the technical leads. In some organizations they are considered the architects of the system.
7. Test Engineers (or QA personnel). These users develop detailed test cases from the requirements specifications, run the tests, and document the results in the test cases.
8. IT Managers (Mostly line level managers).
9. System Administrators. These users manage and maintain the system, accessibility and perform system maintenance upgrades.

1.3 User Roles and Accessibility

Local User – user with locally installed test engine on PC/laptop. Able to register with Test Server to register test engine in database. Able to run tests locally. Uses UI with internal engine.

Remote User – user with same capability as local user, but has ability to view available test engines across the web and to use other hardware, servers, infrastructure for test purposes as well as capability to view archived results.

Administrator – user with the ability to install application remotely, update application services, add remote users, de-register test engines, perform HA/DR testing.

1.4 System Usability

The system will be used by a range of professional IT development staff. This is a system that the developers, architects, engineers, and others should be able to learn to use quickly, enable quick testing of program code, get results back and view logs or other test output. The system shall have:

1. Graphic User Interface (GUI).
2. Web enabled front end.
3. Capability to run multiple tests simultaneously.
4. Capability to ensure that no one test can tie up system resources.
5. Ability to allow multiple users to use the system at the same time.
6. System is highly available, disaster recoverable, and located in multiple regions of a cloud platform that allow for excellent performance, local scalability, and reduction in network latency.

Glossary

MVP – Minimum Viable Product. This is the minimally acceptable product the user can test with and utilize for test purposes.

GUI – Graphic User Interface. Web based or Windows based client front-end to allow the user ease of use of the application.

Concurrency – Has multiple meanings depending upon context. Can mean multiple users, can also mean multi-threaded.

Scalability – There are two types. Horizontal scaling is the adding of additional infrastructure (namely servers) to handle increased loads or parallel processing. Vertical scaling is adding more resources (more memory or more CPU) to existing infrastructure.

API – Application Programming Interface. Program code that allows for communication between one application to another via a defined set of protocol (rules).

HA – High Availability. Usually measured by "nines", like four nines (99.99%) is the measure of uptime and available for user use, of the system. Sometimes measured as AEC (Availability Environment Classification) scheme codes. A system with an AEC value of 2 is considered highly available.

DR – Disaster Recovery. This is the failover/recovery method of the system. Recovery levels usually range from zero to five, but there are two important measurements or requirements for determining system recoverability. RTO (Recovery Time Objective) is the measurement of how long the system can be down before it must be online again and available for use. RPO (Recovery Point Objective) is the measurement of how much time elapses between snapshots, copies or other replication of data. In other words, how much data can you afford to lose? The smaller the numbers in either case, the higher the Recovery level has to be.

WebUI – Web User Interface. Web enabled Graphic User Interface (GUI) using the client machine web browser.

2. Requirements

The top-level requirements in the following section represent user requirements and sub-requirements represent system requirements.

2.1 User Requirement 1: Test System Capability

The users of this Test Framework system shall have the ability to setup and run individual unit tests of program code, hardware, and infrastructure, as well as run multiple tests simultaneously. They need to be able to stress performance, ensure scalability, and diagnose system interface issues. Test results shall be saved for future recall, and configuration of the system maintained and stored.

2.1.1 The test engine shall not require changing and recompiling the program each time a test is run. [0]

2.1.2 The system shall be implemented as a client-server system wherein the Web User Interface (Web UI) communicates with the test engine and test server over the internet. [1]

2.1.3 The system shall employ a database in which to store all test data, including configuration data, test cases, and test results. [1]

2.1.4 The system shall employ a test server, which will provide the business logic and serve as the API between the database and the WebUI. [1]

2.1.5 The system shall be hosted on a cloud platform to support ease of resource acquisition and hosting, automatic scaling of system resources, built-in network infrastructure, managed services where needed. [2]

2.1.6 The system shall allow the ability to create additional test environments for specialized testing upon demand. [2]

2.2 User Requirement 2: Logging and Results Viewing

The user shall have the ability to view the results of the test as each test completes. He/she shall also have the ability to view the log files where all results and relevant output are stored. The logs should contain various levels of information depending on the specified log level and shall display a time and date stamp.

2.2.1 The system shall display whether each test failed or succeeded. [0]

2.2.2 The application shall log all test results to the output file specified in the GUI. [0]

2.2.3 The log component shall show different levels of logging (INFO, DEBUG, ERROR). [1]

2.2.3.1 INFO shall describe specific information for test pass/fail reporting.

2.2.3.2 DEBUG shall describe information provided by the programmer/developer to aid in debugging the test.

2.2.3.3 ERROR shall describe the most detailed debugging output for examination of software test failures.

2.2.4 The log shall display the time and date stamp for each test. [1]

2.2.5 The log shall display the duration of each test. [1]

2.3 User Requirement 3: Concurrent Test Execution

Users shall have the ability to provide a test case where several tests are sent in quick succession to demonstrate the application executes tests concurrently.

2.3.1 The test case shall consist of several tests of varying duration that can run simultaneously. [0]

2.3.2 Upon completion, the system shall post a ready status message and await the next test. [0]

2.3.3 The application will allow the tests to run asynchronously so that no one test will hold up the other tests by tying up resources and starving the other processes (threads). [0]

2.3.3.1 The system shall allow specification of the thread pool size. [1]

2.3.3.2 The starting default minimum thread count shall be 5. [1]

2.3.3.3 The starting default maximum thread count shall be 15 (this keeps the application from spawning too many threads). [1]

2.4 User Requirement 4: Platform Support

The test engine shall run on Windows platform and should run on Linux and Mac platforms. User access to the system shall be provided through a Web User Interface (WebUI). The WebUI shall support the Firefox web browser and should support Google Chrome and Microsoft Edge.

2.4.1 The test engine shall be an application that can run on the Windows platform. [0]

2.4.2 The test engine should run on Linux and Mac platforms. [1]

2.4.3 Client access to the system shall be provided through a WebUI to be accessed via a standard web browser. [1]

2.4.3.1 The system shall support the Firefox web browser. [1]

2.4.3.2 The system should support the Microsoft Edge and Google Chrome web browsers. [2]

2.5 User Requirement 5: Error Handling

The system shall handle application failures and errors as well as test failures and errors.

2.5.1 The system shall handle exceptions thrown by the application during testing with clear user output. [0]

2.6 User Requirement 6: User Interaction

The system shall be user friendly, have a graphic user interface, and use a web-enabled client (browser). The system shall be installable locally on the user's machine and use its own desktop interface.

2.6.1 The system shall have a desktop interface for locally installed users. [0]

2.6.2 The system shall have a WebUI for remote users. [1]

2.6.3 The system shall be available on demand remotely via the WebUI. [1]

2.6.4 The desktop interface shall:

- 2.6.4.1 Display all possible tests and allow the user to select all tests they wish to run. [0]
- 2.6.4.2 Display the selected list of all tests to be run (container object on GUI). [0]
- 2.6.4.3 Show test progress and status on the GUI. [0]
- 2.6.4.4 Display the results of each test in real-time. [1]
- 2.6.4.5 Allow the user to specify an output file in which to log the test results. [1]

2.6.5 The WebUI shall:

- 2.6.5.1 Display all possible tests and allow the user to select all tests they wish to run. [1]
- 2.6.5.2 Display the selected list of all tests to be run (container object on GUI). [1]
- 2.6.5.3 Show test progress and status on the GUI. [1]
- 2.6.5.4 Display the results of each test in real-time. [2]
- 2.6.5.5 Allow the user to specify an output file in which to log the test results. [2]
- 2.6.5.6 Execute tests on available clients. [1]
- 2.6.5.7 Export current and prior test results for specific clients. [2]

2.7 User Requirement 7: High Availability

The system shall be highly available and disaster recoverable. The system will have a production environment that is usable by multiple users implemented in multiple regions and availability zones in the cloud. [2]

2.7.1 The user shall have the ability to install the test engine on multiple machines (redundancy, performance, latency). [0]

2.7.2 The system shall maintain all program code in scripts that can be deployed to the cloud platform. System source code and data shall be stored in a fault-tolerant, distributed file system such as Amazon S3 or HDFS where it can be accessible and deployed to support disaster recovery and availability requirements. [2]

2.7.3 Backup copies of all scripts shall be located in a separate region. [2]

2.7.4 System source code shall be deployed to cloud instances hosted in several regions and availability zones. [2]

2.7.5 Access to the system shall be controlled using defined, cloud managed IAM roles, which will allow for configurable levels of access to and control over the system and its resources. [1]

2.7.6 The test environment shall be implemented in multiple zones and multiple regions to enable testing of HA/DR requirements rather than taking production down. [3]

3 Availability and Business Continuity Requirements

3.1 Availability Requirement 1: Continuous System Uptime

The system shall support 24/7 availability. Routine downtime in a particular region necessary for maintenance or enhancement to the system shall take place after 21:00 EST on Saturday and shall end before 23:00 EST on Sunday. [1]

3.1.1 Any scheduled downtime which takes place outside of the designated hours shall be reported to users no less than 48 hours in advance. In the case of emergency system outage, the notice period shall be waived but users shall be informed as soon as possible of any unplanned system outages. [1]

3.2 Availability Requirement 2: Recovery Time

The system shall be able to quickly recover from outages due to unforeseen circumstances while minimizing downtime. The system shall support the ability to create and issue automated alerts when downtime is encountered for any of the reasons stated below. [1]

3.2.1 In the event of an unplanned outage due to the loss of a particular region or availability zone, the system shall immediately fail over to another region or availability zone as determined by the cloud provider. [1]

3.2.2 In the event of an unplanned outage due to the failure of an instance on which the system is hosted, the system shall immediately fail over to a backup instance. In the event a backup instance does not exist, the system shall have the ability to immediately spin up a new instance and fail over to it using automated deployment. [1]

3.2.3. In the event of an unplanned outage in any or all regions due to a software error, the system shall support the ability to quickly identify and create a restore point from the last known working backup. This process shall take no more than 1 hour to complete from the time the software malfunction is identified. [1]

3.3 Availability Requirement 3: High Availability

The system shall support high availability by being quickly accessible to users attempting to access it from any geographic region. [1]

3.3.1 The system homepage shall take no more than an average of five (5) seconds to load from the time the URL is input from a web browser in any geographic region. This average shall be taken from 10 consecutive attempts to access the homepage. [1]

3.3.2 Navigation actions (paging, links, etc.) should take no more than an average of three (3) seconds to load from the time the action is triggered. This average shall be taken from 10 consecutive attempts to perform the action. [1]

4 Constraints

4.1 Technical Constraints

4.1.1 The system shall be developed using the C++ programming language and the C++ Standard Template Library (STL). [0]

4.1.2 The system shall be developed using a publicly available source code editor which supports the C++ language. [0]

4.1.3 The system shall have at least 75% unit test acceptance coverage of the source code. [1]

4.2 Operational Constraints

4.2.1 Granting access to a new user of the system shall take no more than 1 business day to complete. [0]

4.2.2 Modifying or removing a user's access to the system shall take no more than 1 business day to complete. [0]

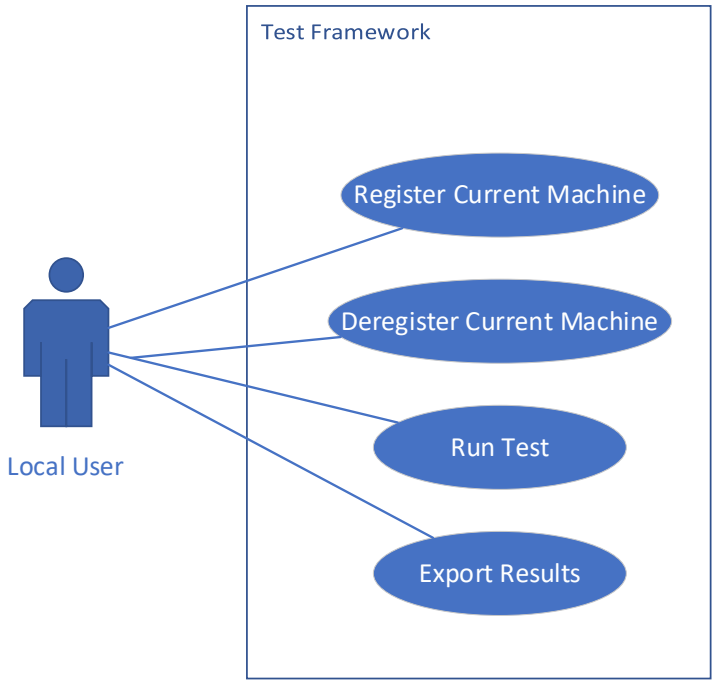
4.3 Business Constraints

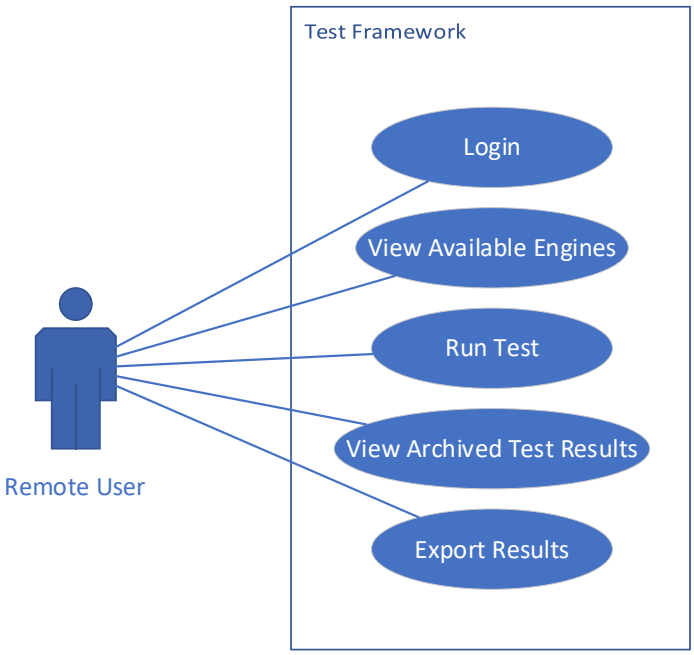
4.3.1 Disaster recovery shall be cost-effective and managed through the fault tolerance and high availability features of the cloud-based system architecture. [1]

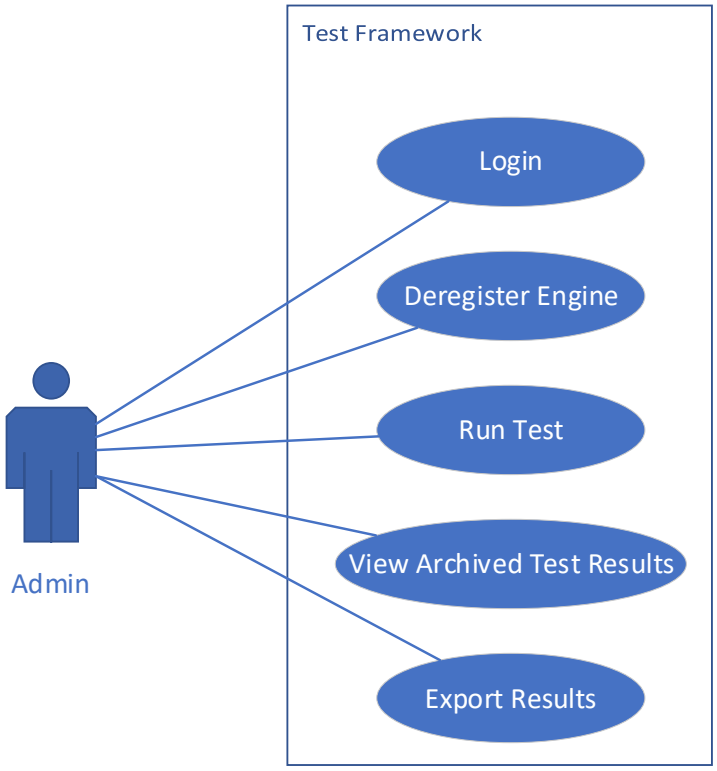
4.3.2 User training shall take no more than 1 business day to complete, regardless of the user's role. [2]

5 System Models

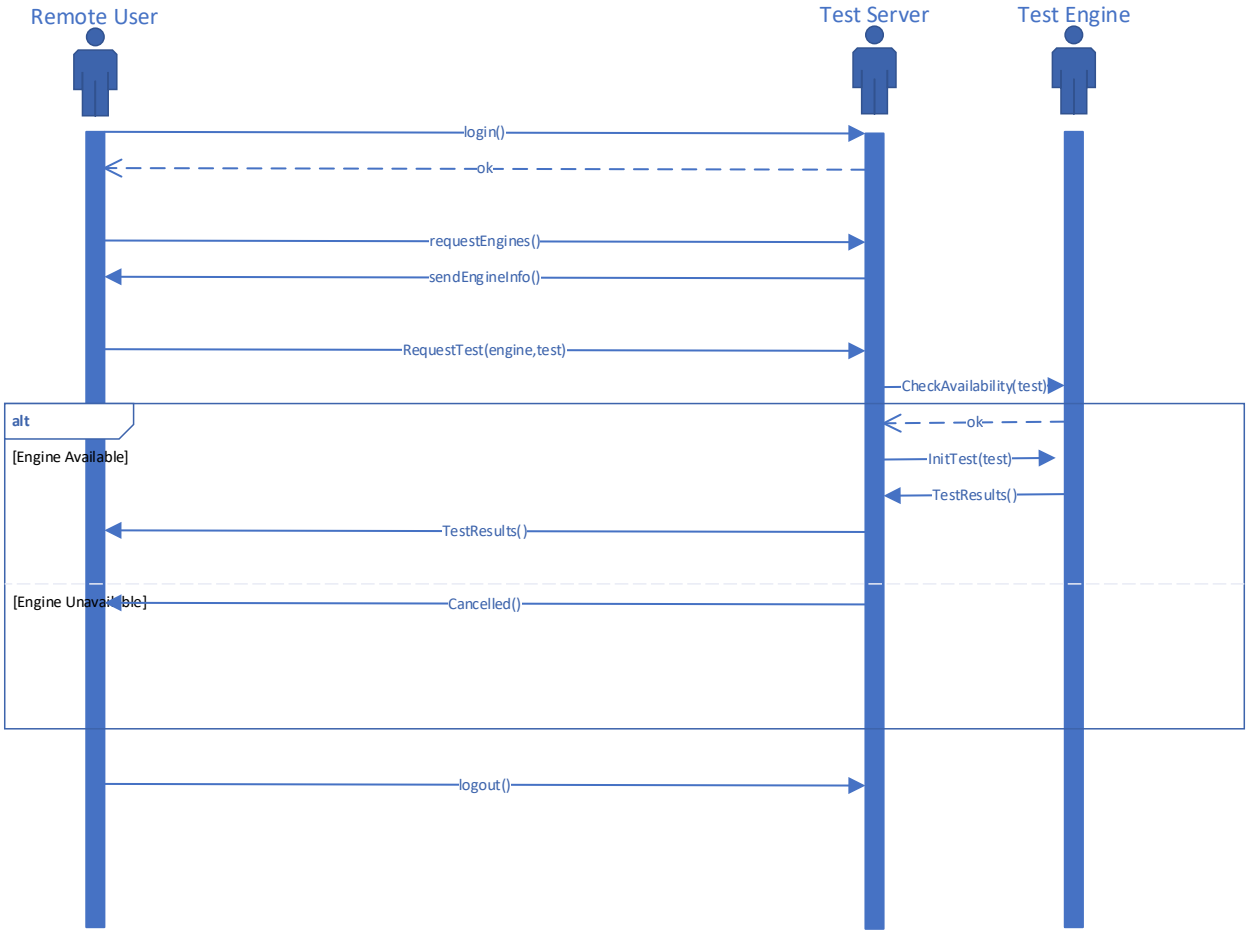
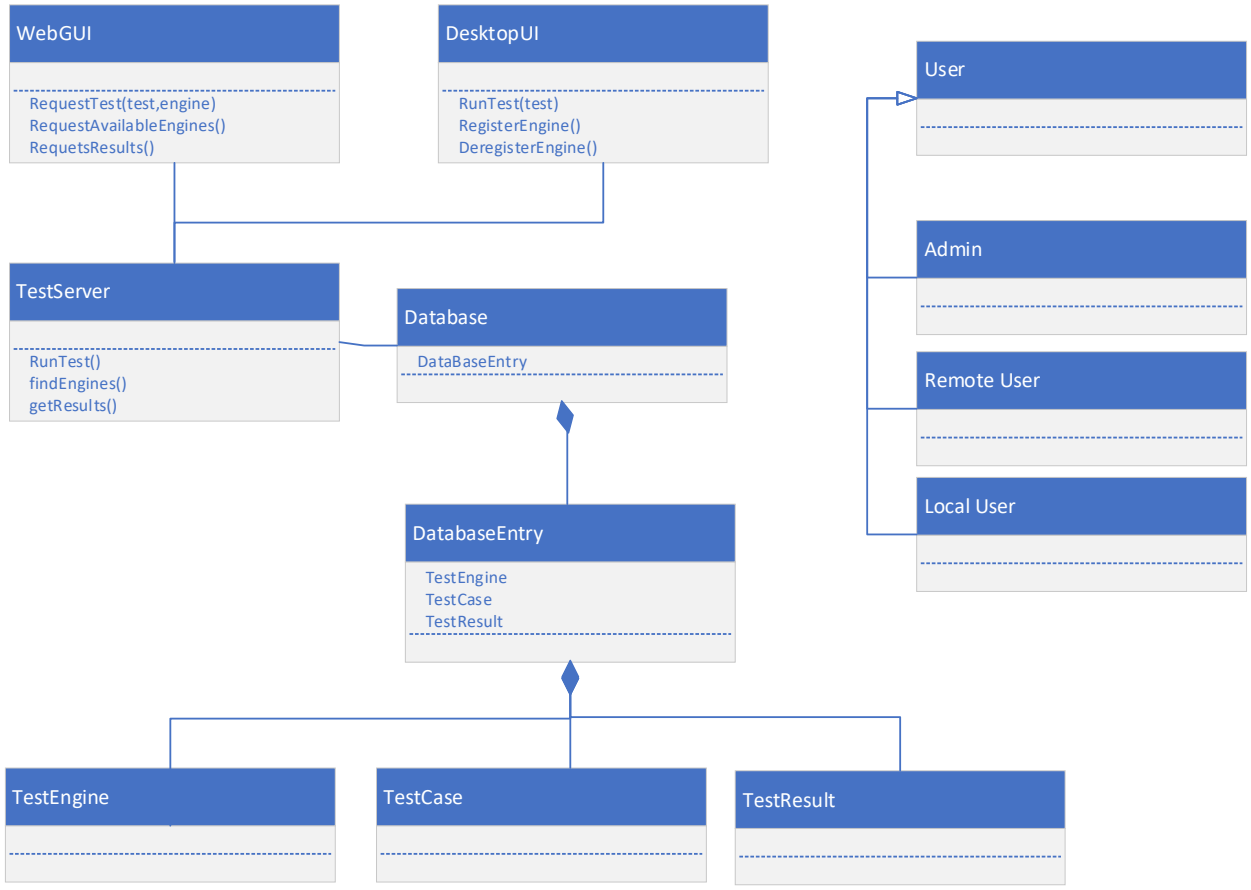
5.1 Use Case Models and/or Scenarios






Test Framework
Login
Deregister Engine
Run Test
View Archived Test Results
Export Results
Admin

5.2 Sequence Diagram



5.3 Class Diagram