Test Framework

Requirements

A Web Based System to Support Testing Multiple Program Modules

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

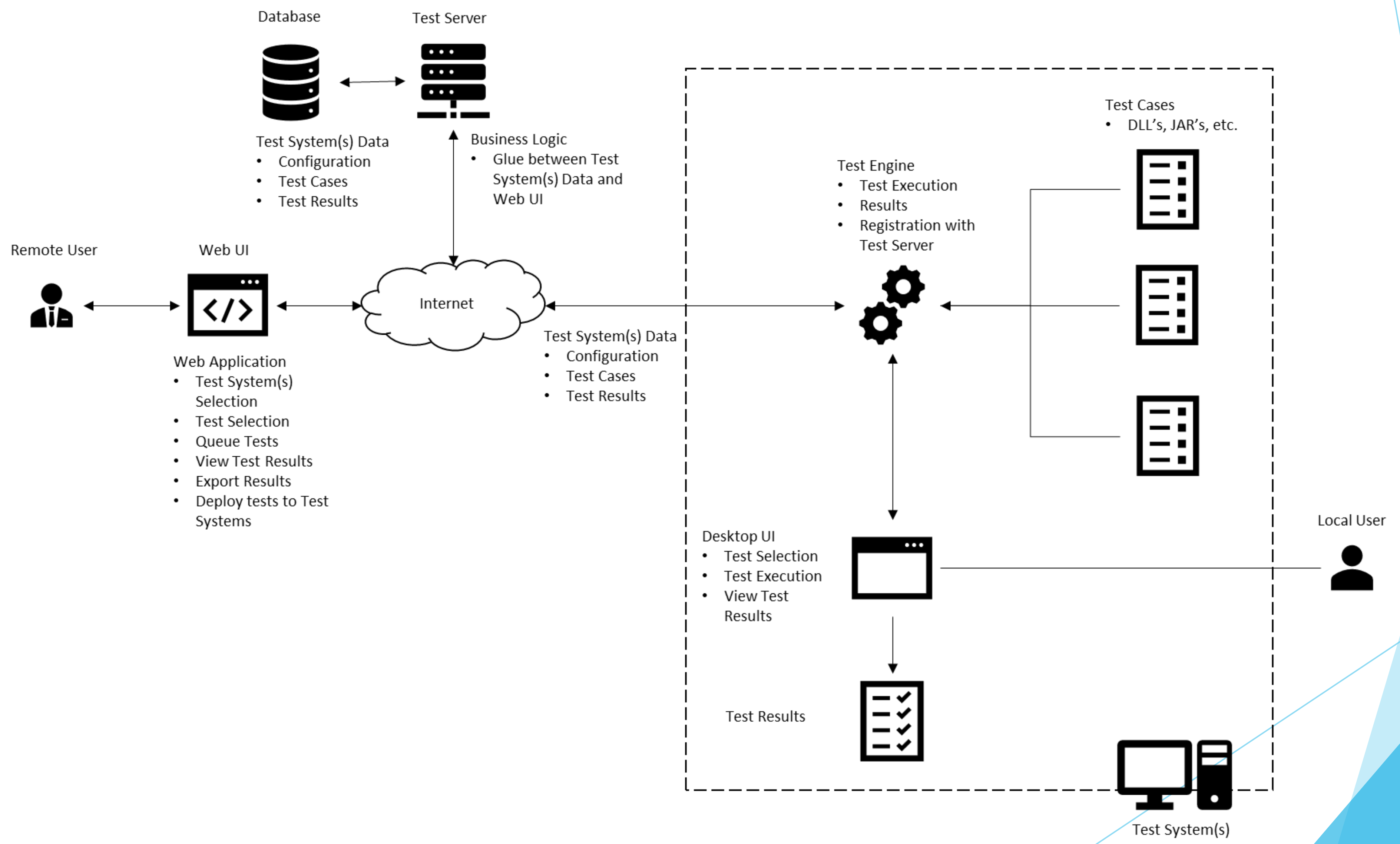
Preface

- Developing large scale software with complex features, interactions, or complex system interfaces (APIs) is best built and tested incrementally.
- Build a basic core, with a small number of packages, then add features one-at-a-time with new packages or new program code to existing packages.
- Each time new functionality is added, the application is built and tested.
- The Test Framework Application will allow the development team(s), and other users of the system to define tests which run with exception handling and results logging.
- Goal of the Test Framework is to allow program testing with out proliferating code with many try-catch blocks, debug statements, assertions, or verbose logging.
- The Test Framework will be easy to use and accessed by the user’s web browser. The system itself being cloud hosted in multiple regions so performance is always top notch.

Introduction

- Web based, cloud hosted solution.
- Ability to run locally; i.e. sans web / cloud hosted.
- Can test multiple program modules or blocks of program code simultaneously.
- Multiple methods of access or delivery (dynamic link library, XML, JSON, etc.).
- Support for Multiple Languages (C++, C#, Java, Python).
- Scalable solution.
- Cloud hosted in multiple regions for best performance, reduction in latency, support high availability and disaster recovery of solution.

System Overview



System Overview

The key components are:

- Users
  - Local User
  - Remote User
- Web User Interface
- Test Server
  - Test Server
  - Database
- Test System
  - Test Engine
  - Test Cases
  - Desktop User Interface
  - Test Results

System Users

There are nine types of users of this system. They are:

- Software Developers. These users are the developers of the software. They create designs for new features, build and run tests for each new feature.
- 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.
- 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.
- 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.
- 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.
- System Engineers. These users are the technical leads. In some organizations they are considered the architects of the system.
- 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.
- IT Managers (Mostly line level managers).
- System Administrators. These users manage and maintain the system, accessibility, and perform system maintenance upgrades.

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.

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:

- Graphic User Interface (GUI).
- Web enabled front end.
- Capability to run multiple tests simultaneously.
- Capability to ensure that no one test can tie up system resources.
- Ability to allow multiple users to use the system at the same time.
- 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.

Requirements (User and System Level)

- 2.1 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 (WebUI) 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]
- 2.2.4 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 environments for specialized testing upon demand. [2]

Requirements (User and System Level)

- 2.2 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]

Requirements (User and System Level)

- 2.3 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]

Requirements (User and System Level)

- 2.4 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]

Requirements (User and System Level)

- 2.5 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]

Requirements (User and System Level)

- 2.6 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]

Requirements (User and System Level) 2.6 Continued

- 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]

Requirements (User and System Level)

- 2.7 The user shall log into the system via the WebUI and the system will authenticate the user. Once logged into the system and user role determination has been made, the user can register/de-register his/her machine with the Test Server.

- 2.7.1 The Remote user shall be required to log into the system via the web client. The login is for accessing the Test Server. [1]
- 2.7.2 The system shall authenticate the user. If the user is authenticated, they are allowed to proceed. If authentication cannot be made, an error message describing the issue is displayed. [1]
- 2.7.3 The user shall have the ability to register their local test engine with the test server. [1]
- 2.7.4 The user shall have the ability to de-register their local test engine with the test server. [1]

Requirements (User and System Level)

- 2.8 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]

- 2.8.1 The user shall have the ability to install the test engine on multiple machines (redundancy, performance, latency). [0]
- 2.8.2 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]
- 2.8.3 Backup copies of all scripts shall be located in a separate region. [2]
- 2.8.4 System source code shall be deployed to cloud instances hosted in several regions and availability zones. [2]
- 2.8.5 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]
- 2.8.6 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]

Availability and Business Continuity

- **3.1 Availability Requirement 1: Continuous System Uptime [1]**
- 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.
- **3.2 Availability Requirement 2: Recovery Time [1]**
- 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.
- **3.3 Availability Requirement 3: High Availability [1]**
- The system shall support high availability by being quickly accessible to users attempting to access it from any geographic region.

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 coverage of the source code. [1]

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]

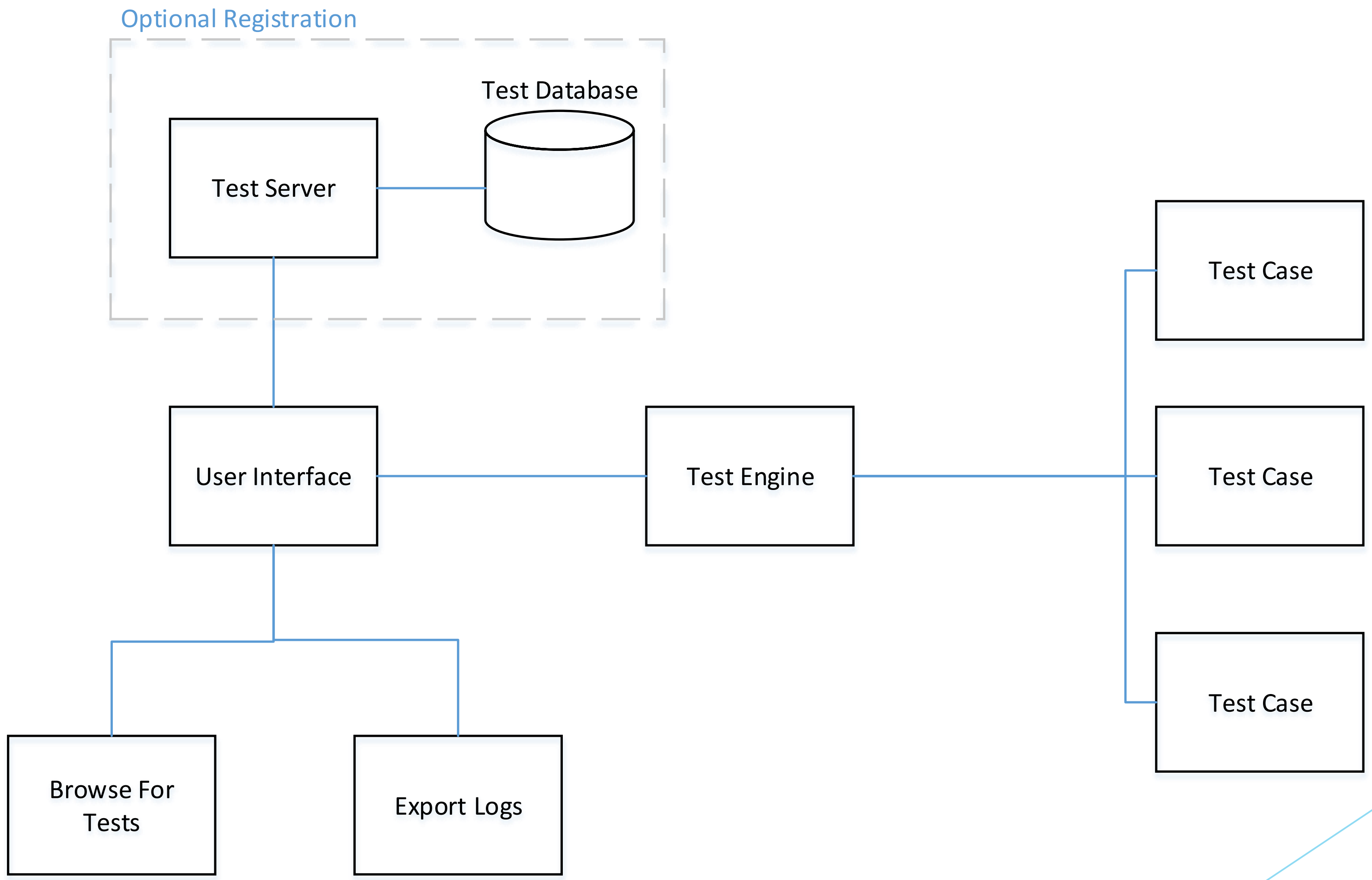
Business Constraints

- 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]

Architectural Design

- The Test Framework System consists of the procedures, documentation, user interfaces, test engine(s), test server(s), database, test cases, test logs, and test results. The Test System can be utilized in one of two ways:
  - Installed locally on the user’s workstation machine (desktop or laptop), the same machine that will be running the test.
  - Accessed remotely with the user communicating with one or more test servers to utilize various test engines that may or may not be located in the same place as the test server.

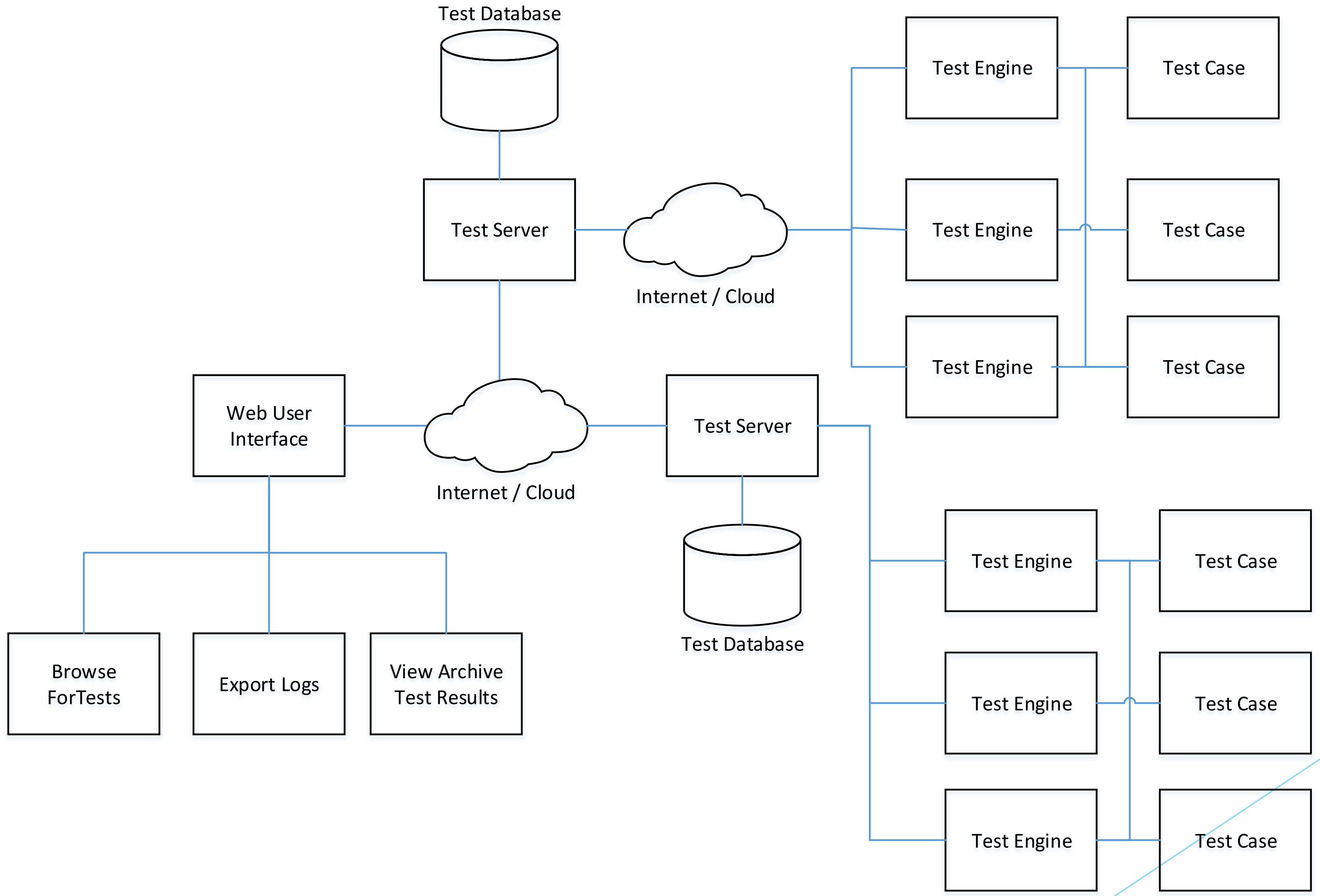
Architectural Design – Locally Installed



Architectural Design – Local Install

- In this context, the entire Test Framework System is installed locally on the user's machine.
- The system presents a built-in native GUI, which allows the user to interact with the system.
- The Test engine in this case is the user's own workstation or laptop. It can execute one or more tests in succession as well as run multiple tests concurrently via multiple threads. The number of threads can be set in the GUI.
- The test cases are the test data or tests to be run. These exist as DLL files, XML files, or JSON, but not limited to these types.
- The local user is permitted to register his or her machine with the enterprise test server as an additional resource for that test server.
- Support for high availability and disaster recovery can be made by installing the application on multiple machines for redundancy.

Architectural Design – Remote System



Architectural Design – Remote System

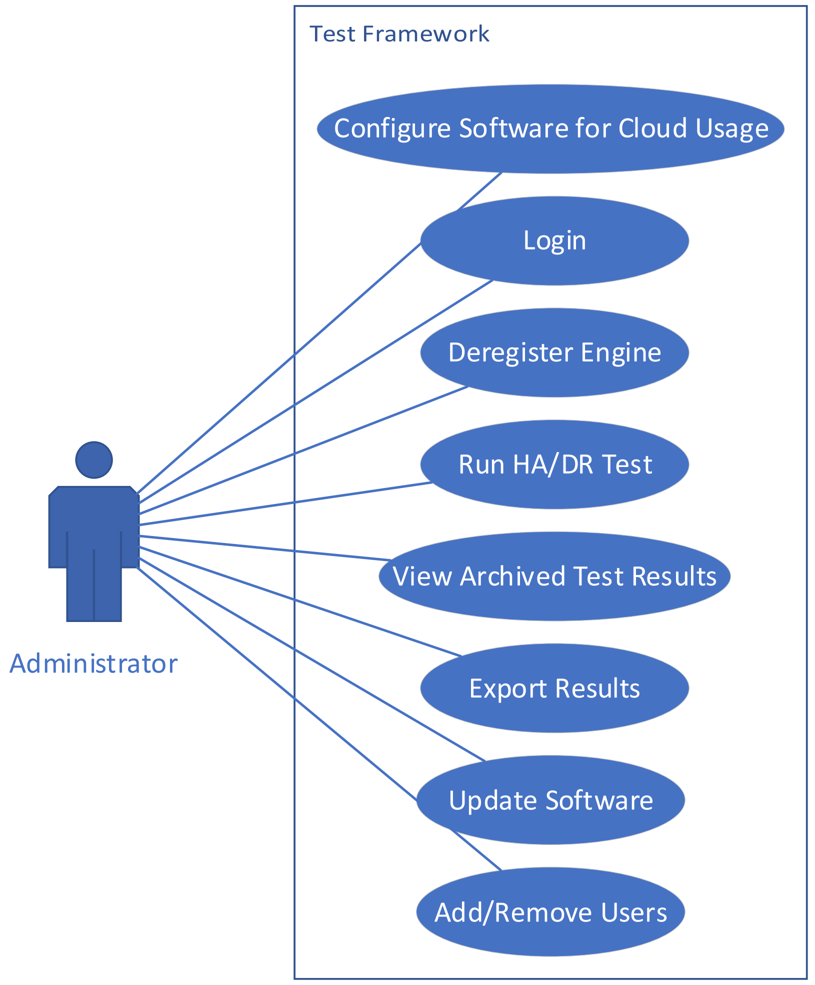
- A Remote User can interact with multiple test systems typically at a location other than where the test system resides.
- The Web User Interface is accessed via the user's web browser.
- The GUI will show and allow management of available Test Engines contained and configured in the Test Server database.
- The GUI will allow multiple test cases to be selected and run with the Browser for Tests Dialog.
- The Test Server acts as the web/application server in this environment. It contains the routines for login, authentication, list of Test Engine servers/workstations, and configuration of each Test Engine, and archived test results.
- The Test Engine(s) are the heart of this system. They are where the actual test cases are run.
- The Test Database stores Test System Data for each test case, as well as system and corresponding test engine configuration, test cases and latest test execution results.
- The test cases are the test data or tests to be run. These exist as DLL files, XML files, or JSON.
- The remote user is permitted to register his or her machine with a test server as an additional resource for those test servers in the system.
- Support for high availability and disaster recovery can be made by use of multiple cloud instantiations, storing all application code as scripts that can be rapidly deployed and installed, and installing the application on multiple machines/servers for redundancy.

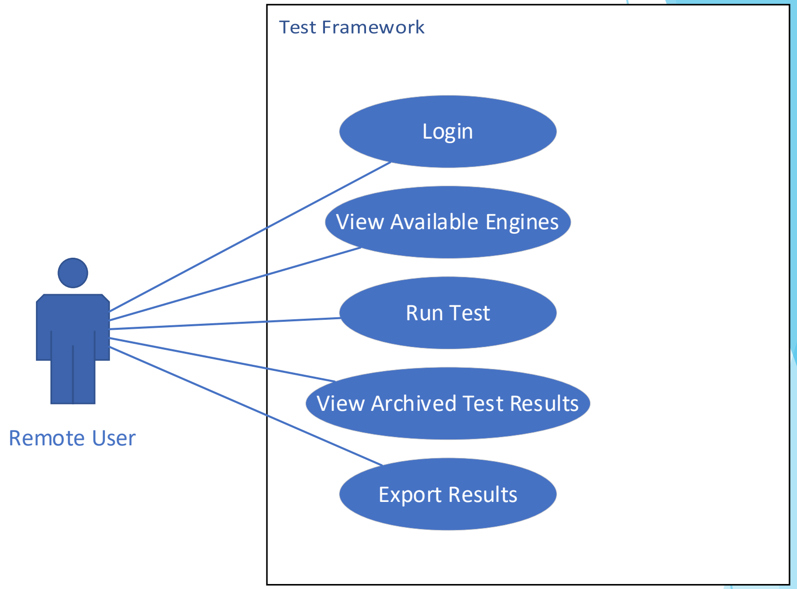
Key Architectural Components of the System

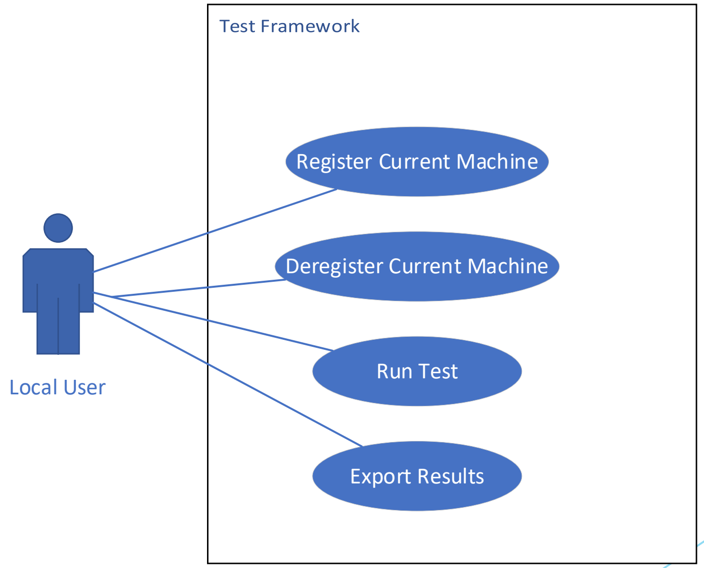
The key components of the system are:

- **Users**. There are three roles:
  - Local User
  - Remote User
  - Administrator
- **Test Framework Procedures**. The procedures for using the system, running and monitoring tests as well as saving results.
- **Documentation**. User guide for installation, procedures, and read me file(s).
- **System User Interface**. One of two interface capabilities
  - Web UI (for remote access to various Test Servers, Test Engines) accessed by a web browser.
  - Desktop UI (for local access). Built-in native GUI for accessing the system.
- **Test Server**. The Test server manages (registers/deregisters) test engine machines and what their configuration is. Test engines can be user laptops, workstations, physical servers in the enterprise or cloud hosted servers including specialized servers such as high performance computing servers or CPU/GPU machines for running graphics or data parallelism.
- **Test Server Database**. Stores the number of test engines, each test engine configuration, available test engines for use, and archived test results.
- **Test Engine**. Can be physical workstation(s), servers, cloud hosted servers, CPU/GPU machines, etc.
- **Test Cases**. These are the tests that are to be run.
- **Test Results**. Results show on the screen (GUI) as a **test log**, as well as can be saved in a log file.

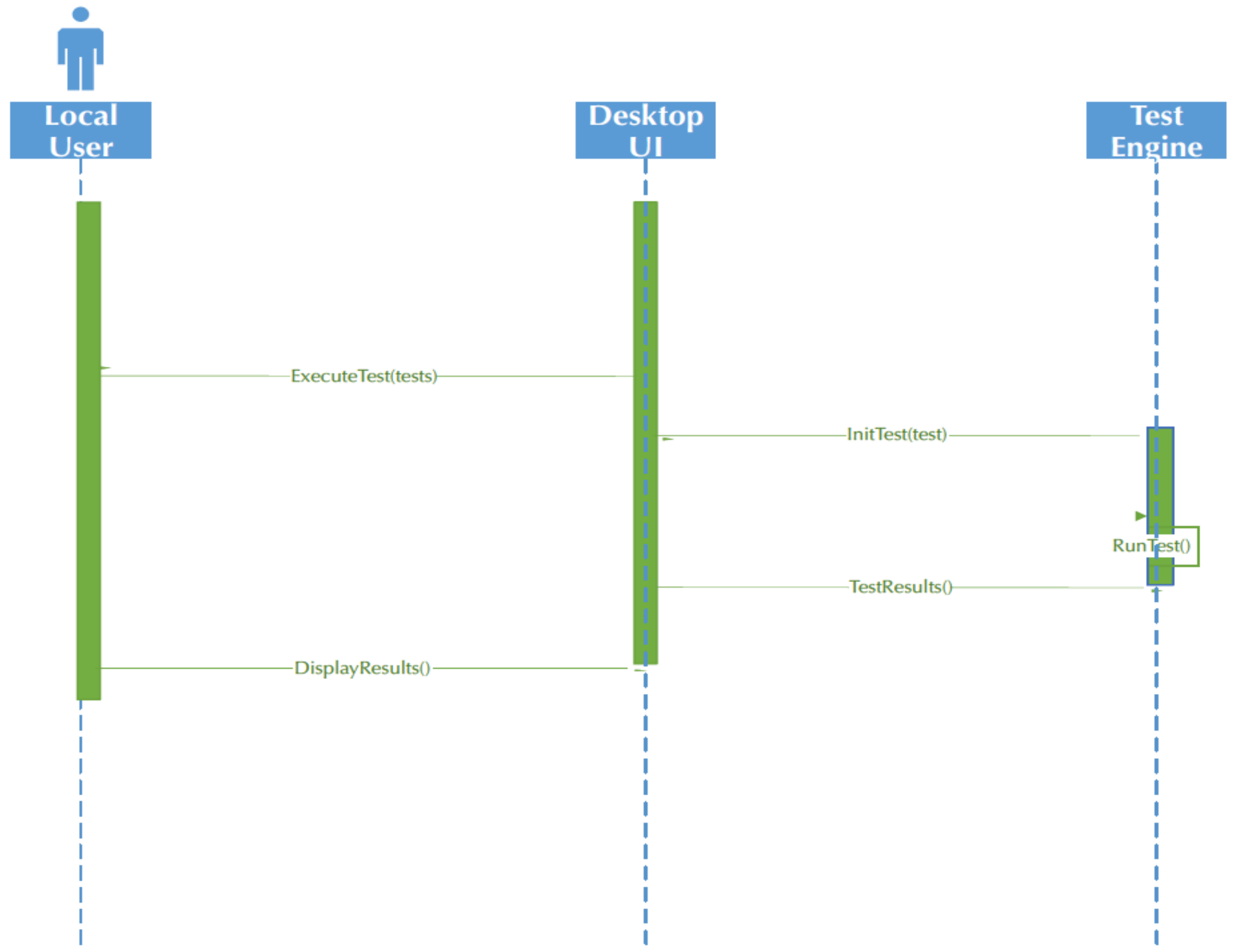
Architectural View Perspectives: Use Cases



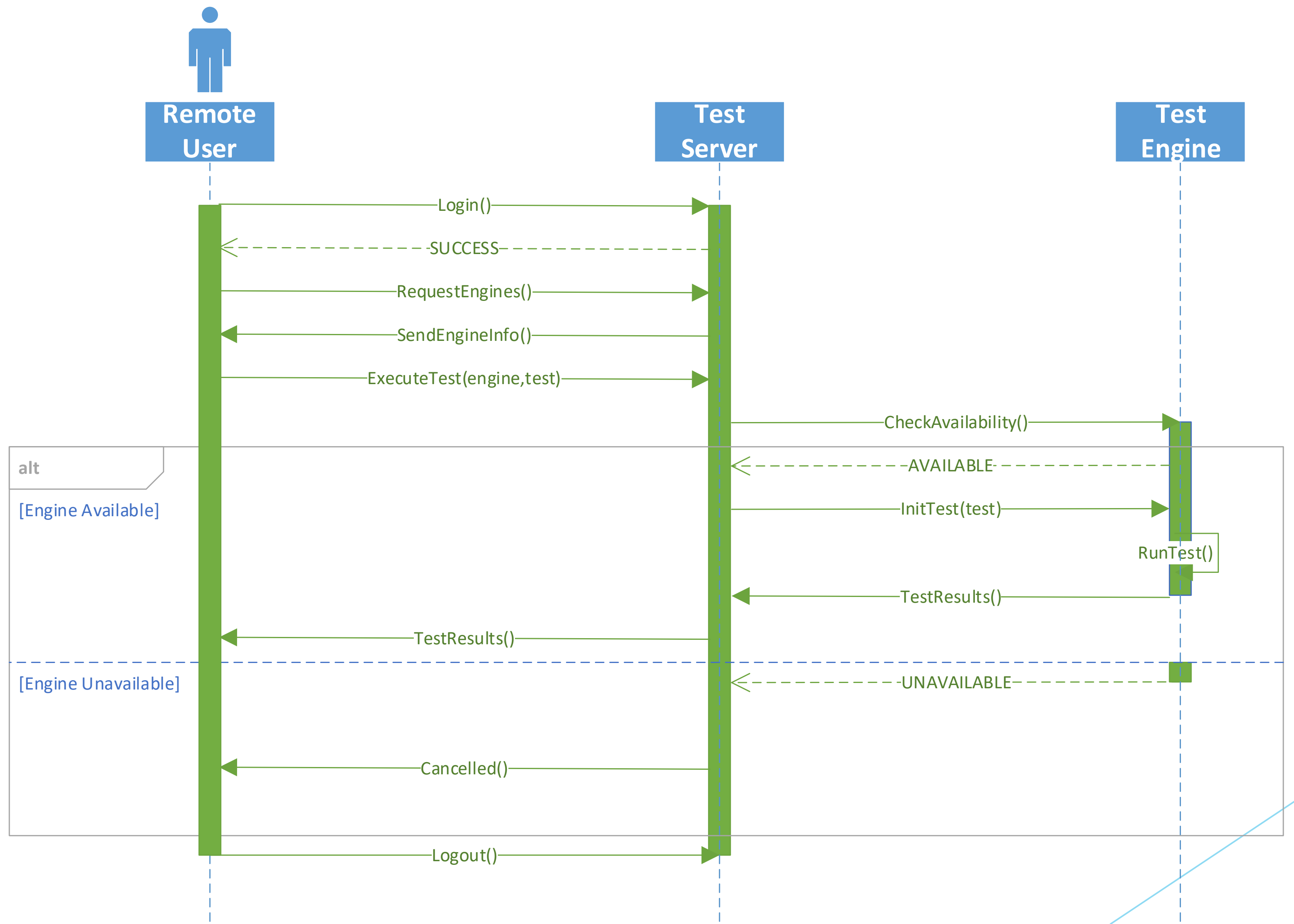




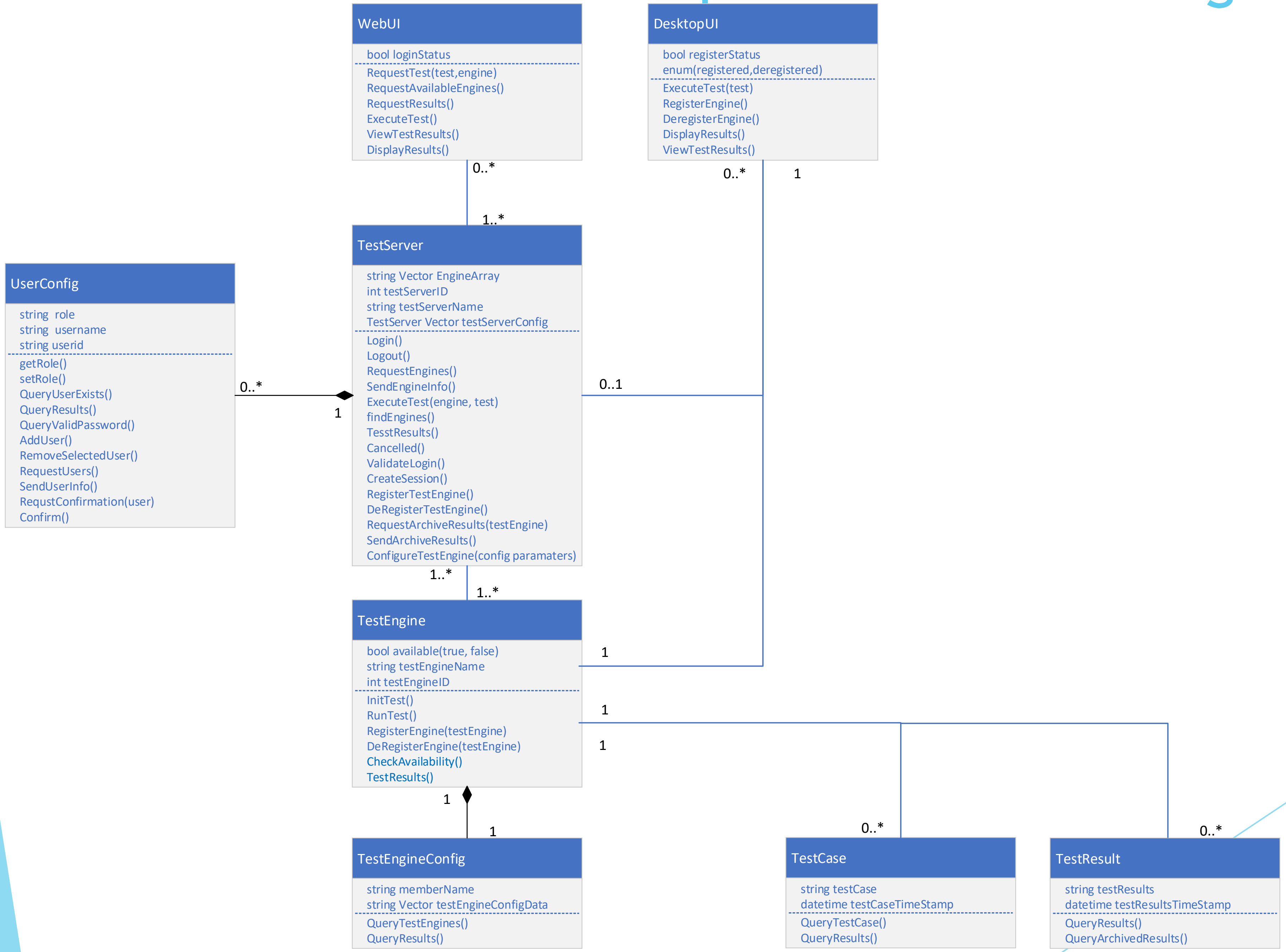
Architectural View Perspectives: Sequence



Architectural View Perspectives: Sequence



Architectural View Perspectives: Class Diagram



Application Architecture Model

- The application architecture model or application template most closely approximates an event driven model. “Event-processing systems respond to events in the system’s environment or user interface”.
- All applications have more than one model. This could actually been viewed as a combination of models, event driven for the activity in the user interface (button clicking and test selection), as well as the tests that are assigned to a thread waiting in the thread pool and then executed.
- Could also be viewed as a transaction processing system. One transaction for each test.

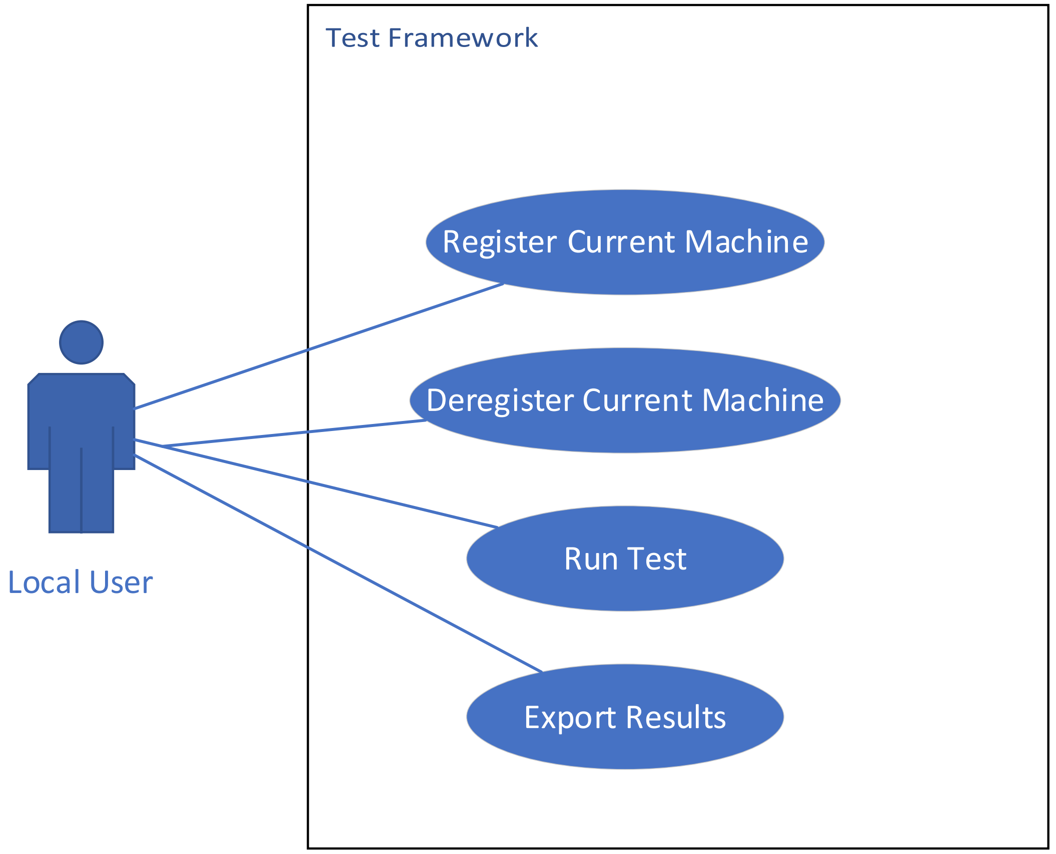
Architectural Patterns

- **Architectural Patterns for Remote Installation**
- The Multi-tier Client-Server pattern was chosen for the remote installation.
  - The web browser functions as the presentation or thin client front-end.
  - The Test Server is the web/application server in this environment. It serves requests to the web browser client for screen updates in the way of test results, Test Engine availability, and Test Engine configuration information.
  - The Test Server Database stores test results, test cases, as well as Test Engine availability and configuration data in the Test Database.
  - Test Server Databases are replicated which solves two architectural problems:
    - Allows for a distributed system and multiple test engines
    - Allows for HA/DR capability
- The Test Engine(s) are resources consumed by the Test Framework System and can be considered another tier of the Client Server Pattern.
- The Multi-tier Client Server Pattern will:
  - Deliver sufficient performance.
  - Allow for various levels of scalability
    - Scale up (vertical scaling).
    - Scale out (horizontal scaling)
    - Increase thread count
    - Run multiple instantiations
  - Allow for loose coupling to make modifications and maintenance of the system easier. Easier maintainability.
  - Security addressed via Test Server login by remote user and Administrator

Architectural Patterns

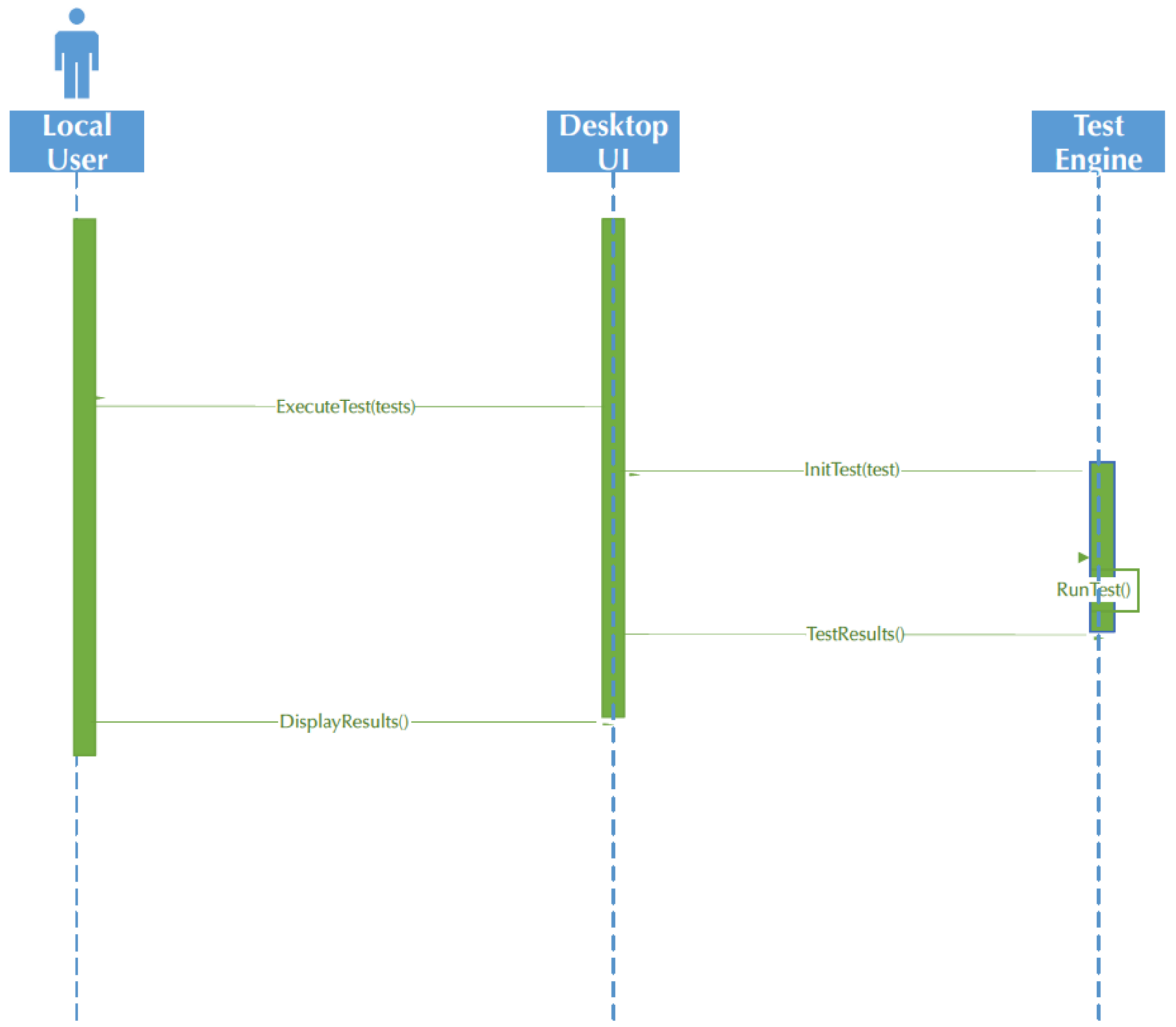
- **Architectural Patterns for Local Installation**
- The architectural pattern that most closely approximates the local installation is the Model View Controller (MVC) pattern.
- In this case this architecture pattern was chosen as the view is the GUI and separate from the test engine.
  - View changes as test data / test results are updated.
- The Test engine is the controller which is executing multiple threads of execution (test cases) in the background concurrently.
- The actual test cases and test data is the data model in this architecture.
- This particular pattern was chosen as it will:
  - Deliver sufficient performance.
  - Allows for some level of scalability on a single machine.
    - Increase the thread count.
    - Run multiple instantiations on multiple cores.
  - Allow for loose coupling to make modifications and maintenance of the system easier. Easier maintainability.
  - Separate the functionality of presentation, management, and test execution.
  - It further lends itself to a possible web GUI for a local installation as a future enhancement.

System Models: Use Case – Local User

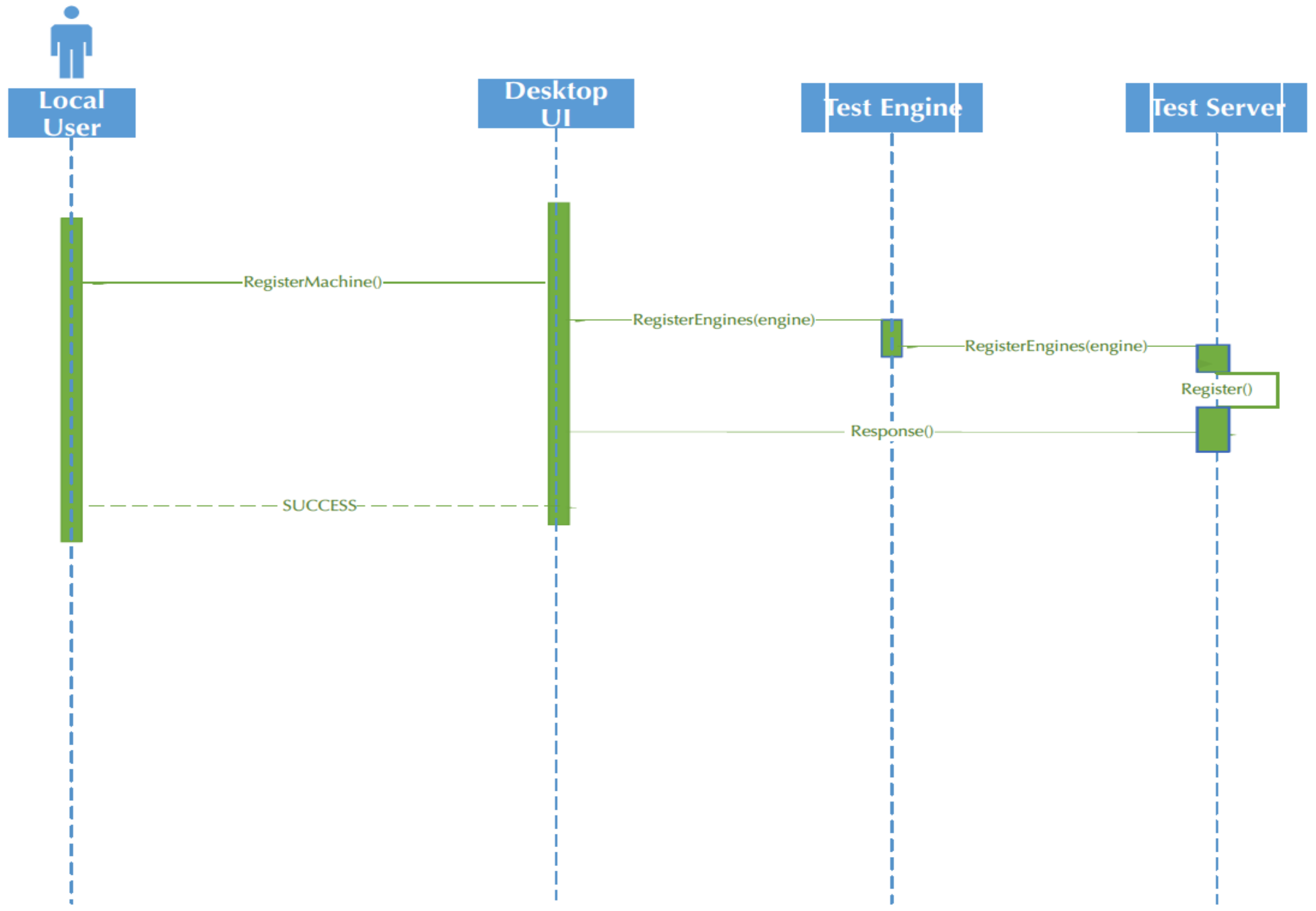


Title:	Local User Use Case
Description:	The local user may interact with the locally installed Desktop UI to perform several actions. These can include running a test, registering or de-registering their machine with the test server and viewing or exporting a log of the most recent test run. Upon completion of any of the activities, the local user may continue to initiate another activity or exit the application.
Actors:	Local User
Stimulus (Trigger)	Local User opens the desktop application installed on their machine which launches the Desktop UI.
Preconditions:	The Test Framework application is correctly installed on the Local User's system. The Local User's system has the minimum required hardware specifications to run the application.
Postconditions:	The desired actions are performed. 1. Test(s) successfully executed. 2. Machine successfully registered with the Test Server. 3. Machine successfully de-registered with the Test Server. 4. Latest test log successfully viewed. 5. Latest test log successfully exported.
Main Success Scenario:	1. Local user launches the Test Framework application. 2. Local user registers their machine with the Test Server. 3. Local user runs test on test engine. 4. Local user de-registers their machine with the Test Server. 5. Local user exits the application. * Other operation supported, as mentioned, such as viewing and exporting log files. The "test run" scenario, however, is the main success scenario.
Extensions:	1. The local user is unable to register/de-register their machine with the test server. → The Test Server returns failure back to the UI and allow the local user to re-try. 2. Test engine fails to execute the test. → The test engine returns failure back to the Desktop UI.
Priority:	1

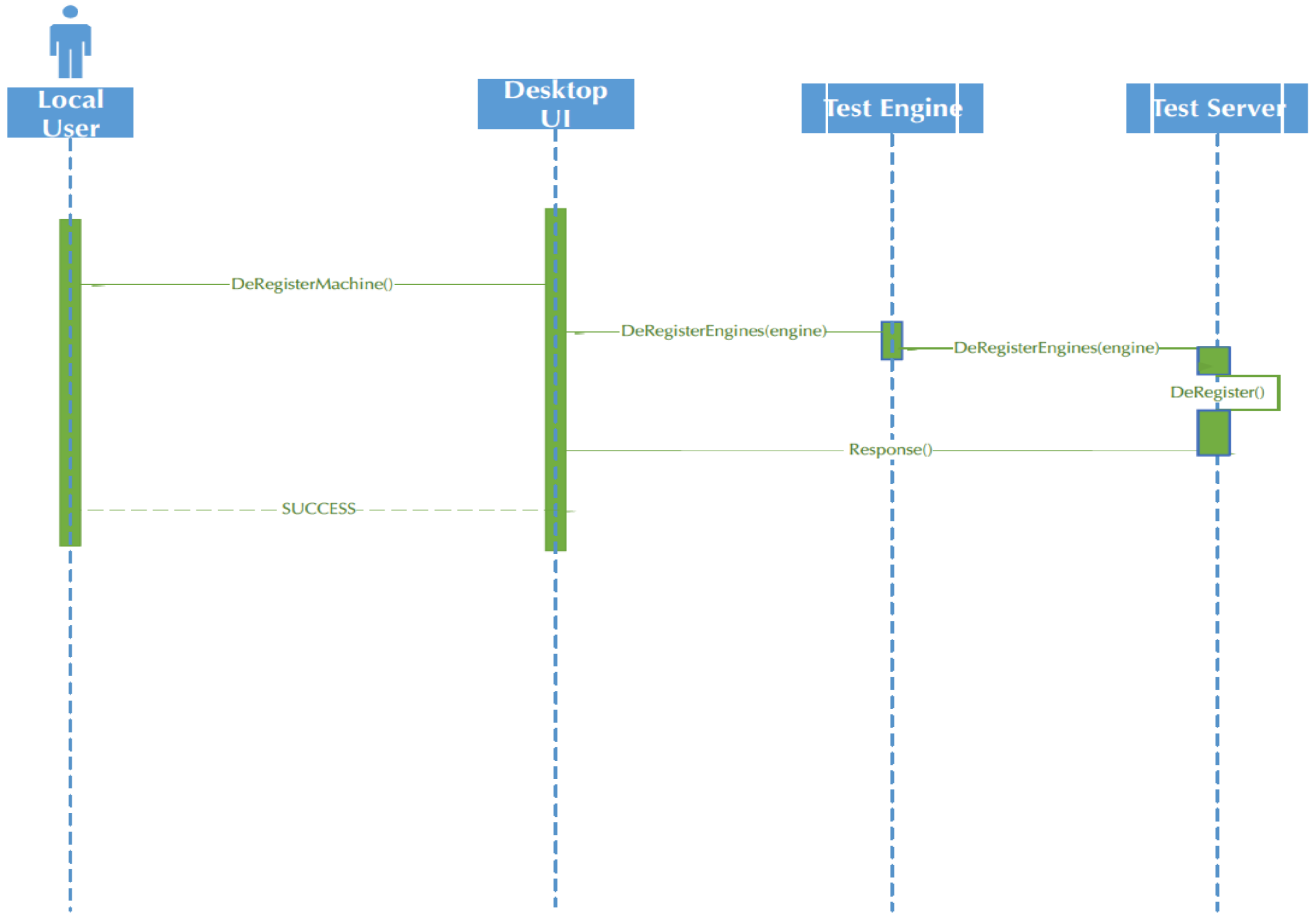
Local User – Run Test



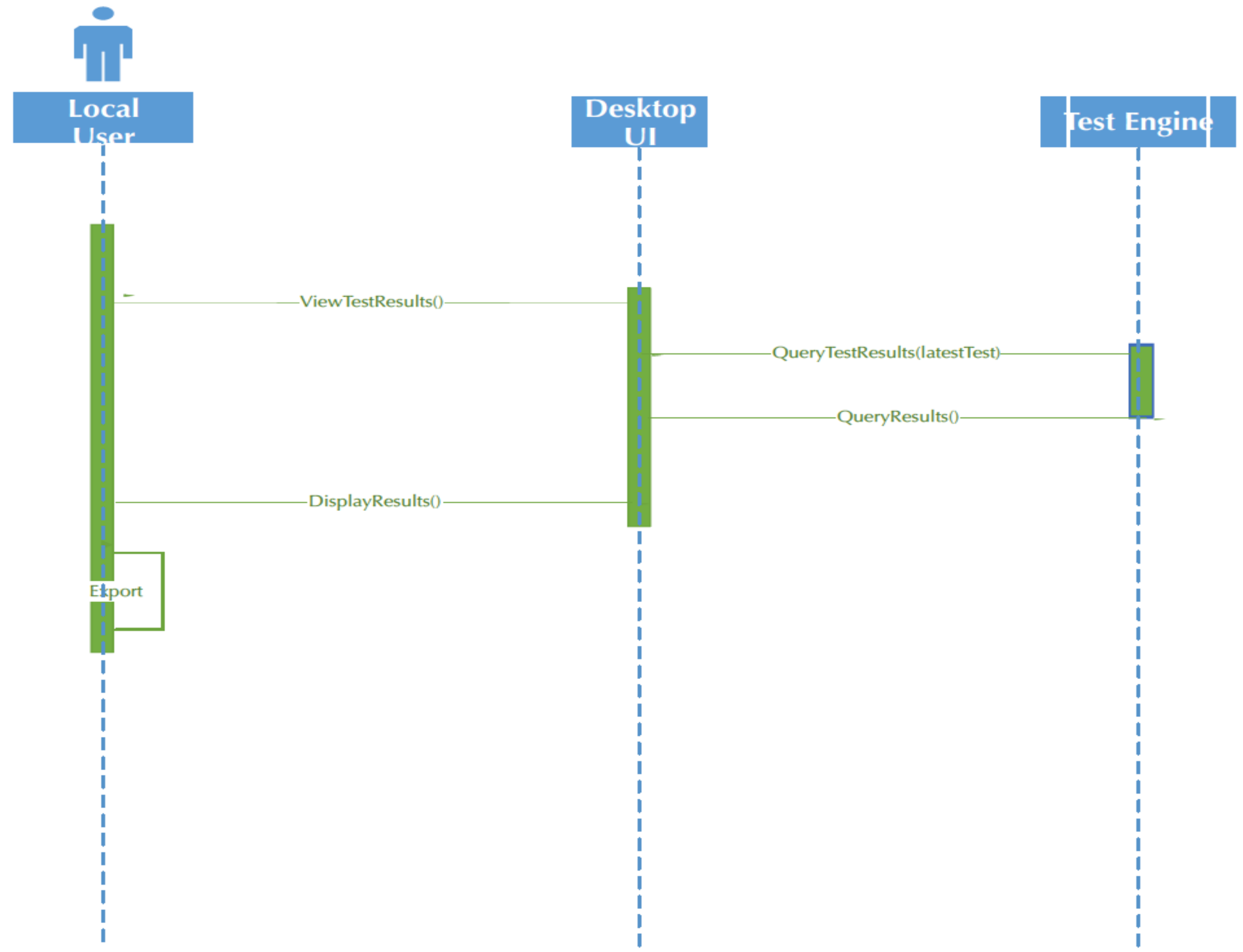
Register Local Machine as Test Engine



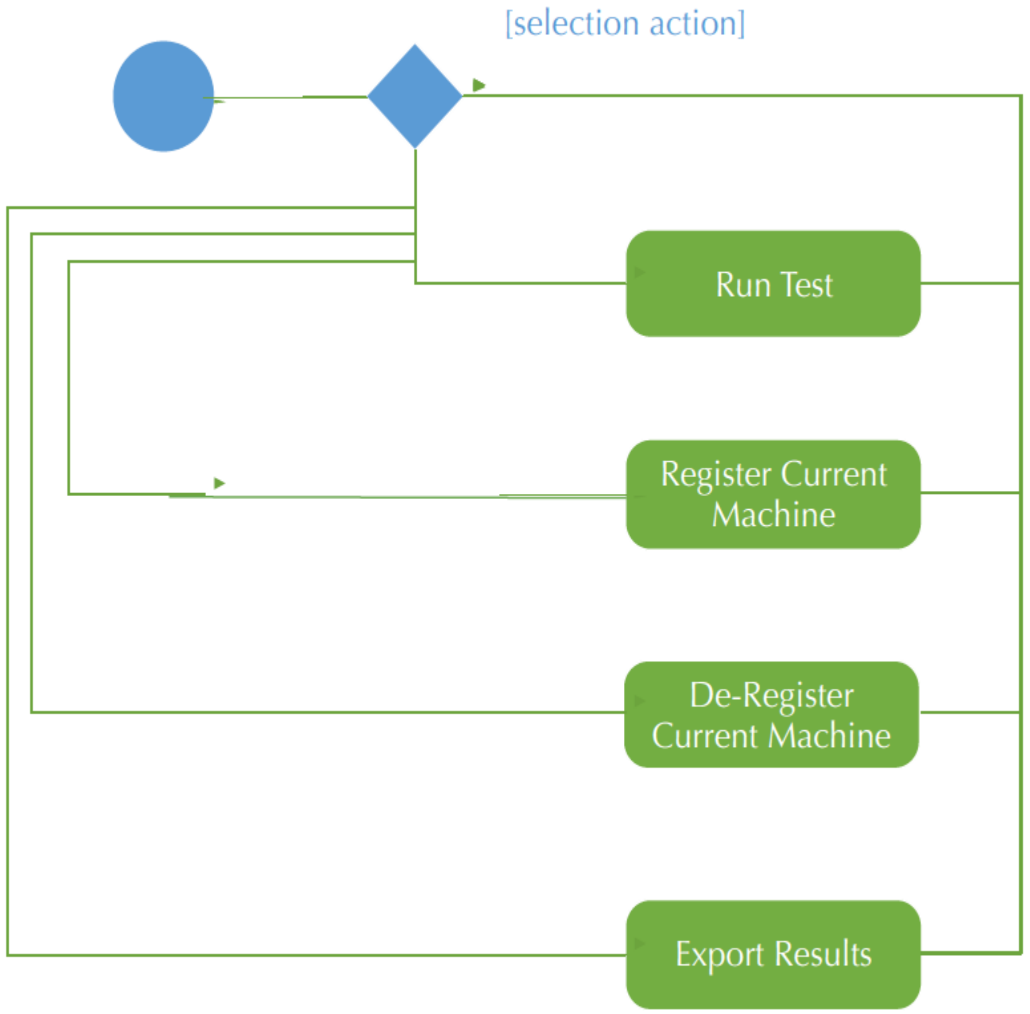
Deregister Local Machine with Test Server



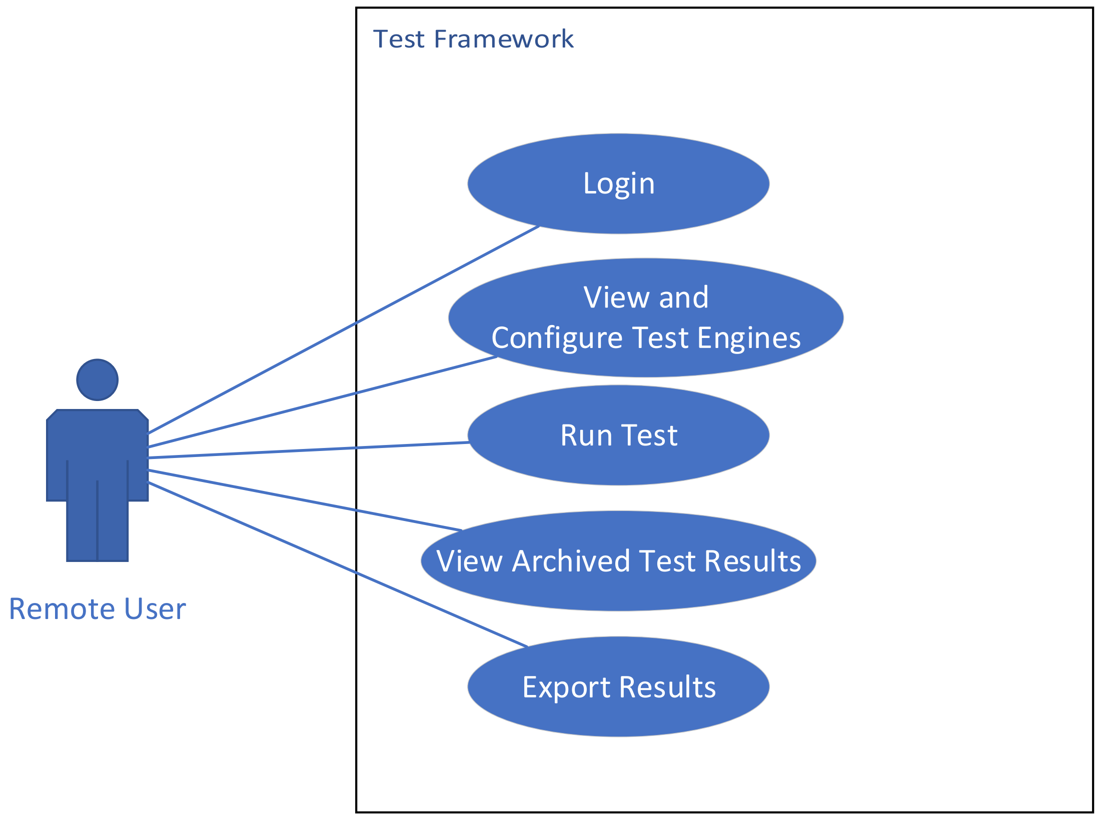
Local User - Export Test Results



Activity Diagram – Local User

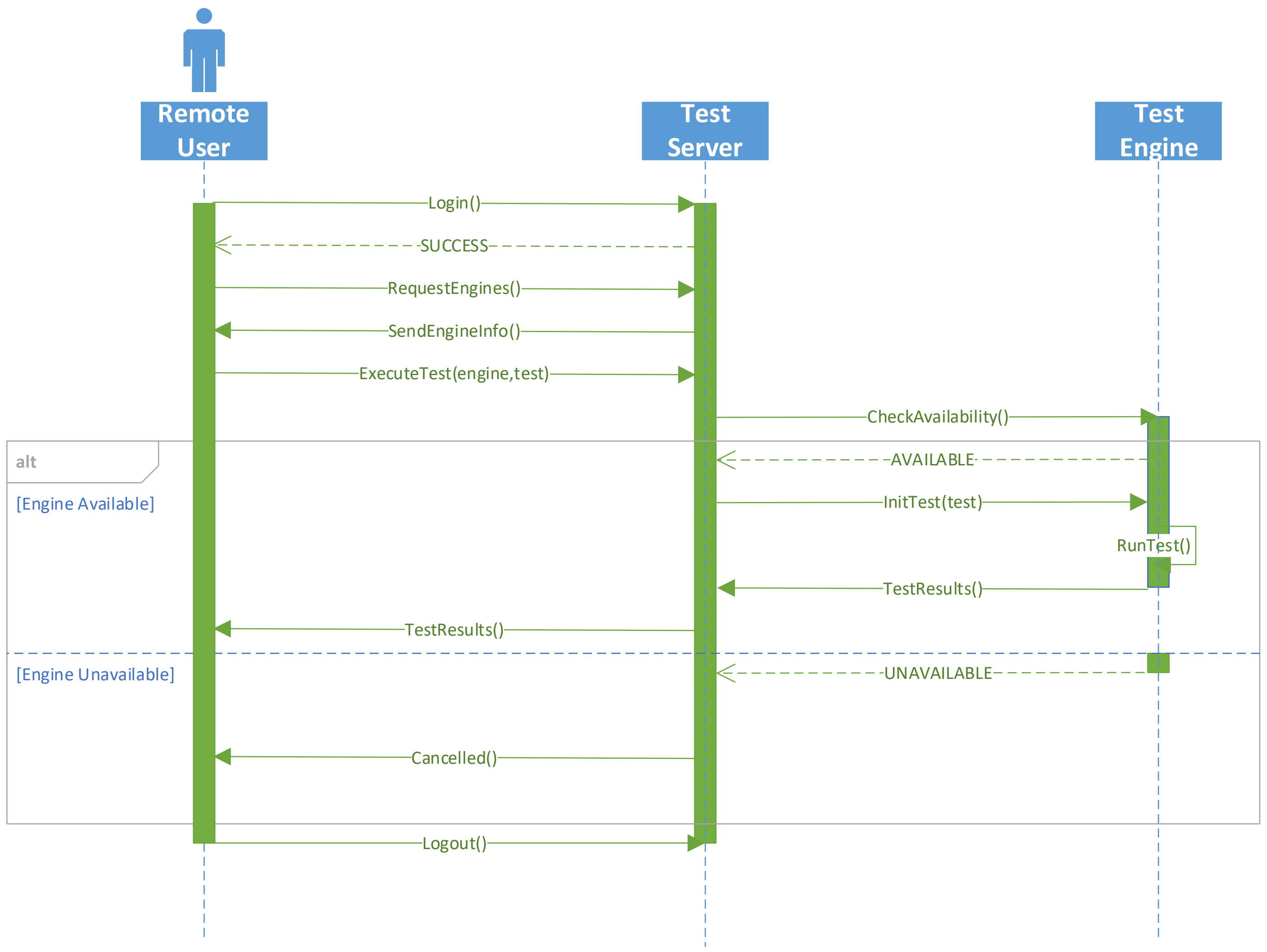


System Models: Use Case Remote Use

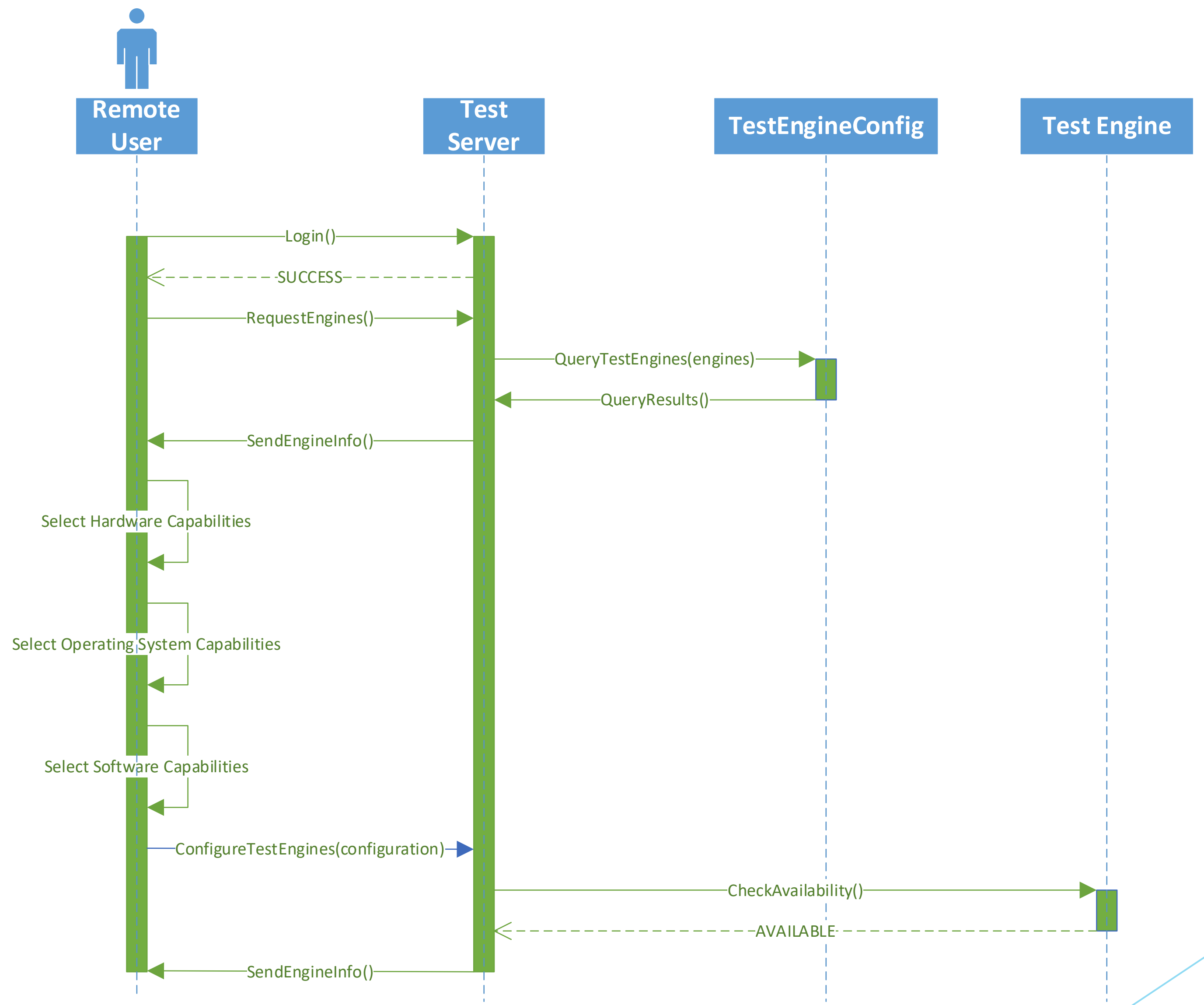


Title:	Remote User Use Case
Description:	The remote user may login to the test server via a web application to perform several actions. These can include viewing test engine configurations so to understand the test platform capabilities, configuring test engines to a selectable to of platform capabilities, run test (or tests) on selected test engines, viewing and exporting archived test for a particular test engine. Upon completion of desired test activities, the remote user may also logout of the system.
Actors:	Remote User
Stimulus (Trigger)	Remote User enters the URL of the web application and successfully logs into the test server.
Preconditions:	Test Server URL is accessible and available to host the remote user web application.
Postconditions:	The desired actions are performed. 1. Test configuration(s) successfully viewed. 2. Test configuration(s) successfully configured. 3. Test(s) successfully executed. 4. Test logs successfully viewed. 5. Test log archives successfully exported. 6. Login successful. 7. Logout successful.
Main Success Scenario:	1. Remote user logs into the system. 2. Remote user views test engine configurations. 3. Remote user selects platform capabilities and configures test engines. 4. Remote user runs test on test engines. 5. Remote user logs out. * Other operation supported, as mentioned, such as viewing and exporting log files. The “test run” scenario, however, is the main success scenario.
Extensions:	1. The remote user’s login information is rejected by the web application. → The remote user is presented with the login screen again. 2. Test engines unavailable to execute. → The test engine returns failure back to the Test Server. 3. No test engines match the capabilities selected. → Test engine prohibits test execute. Allows, test capabilities to be reset.
Priority:	2

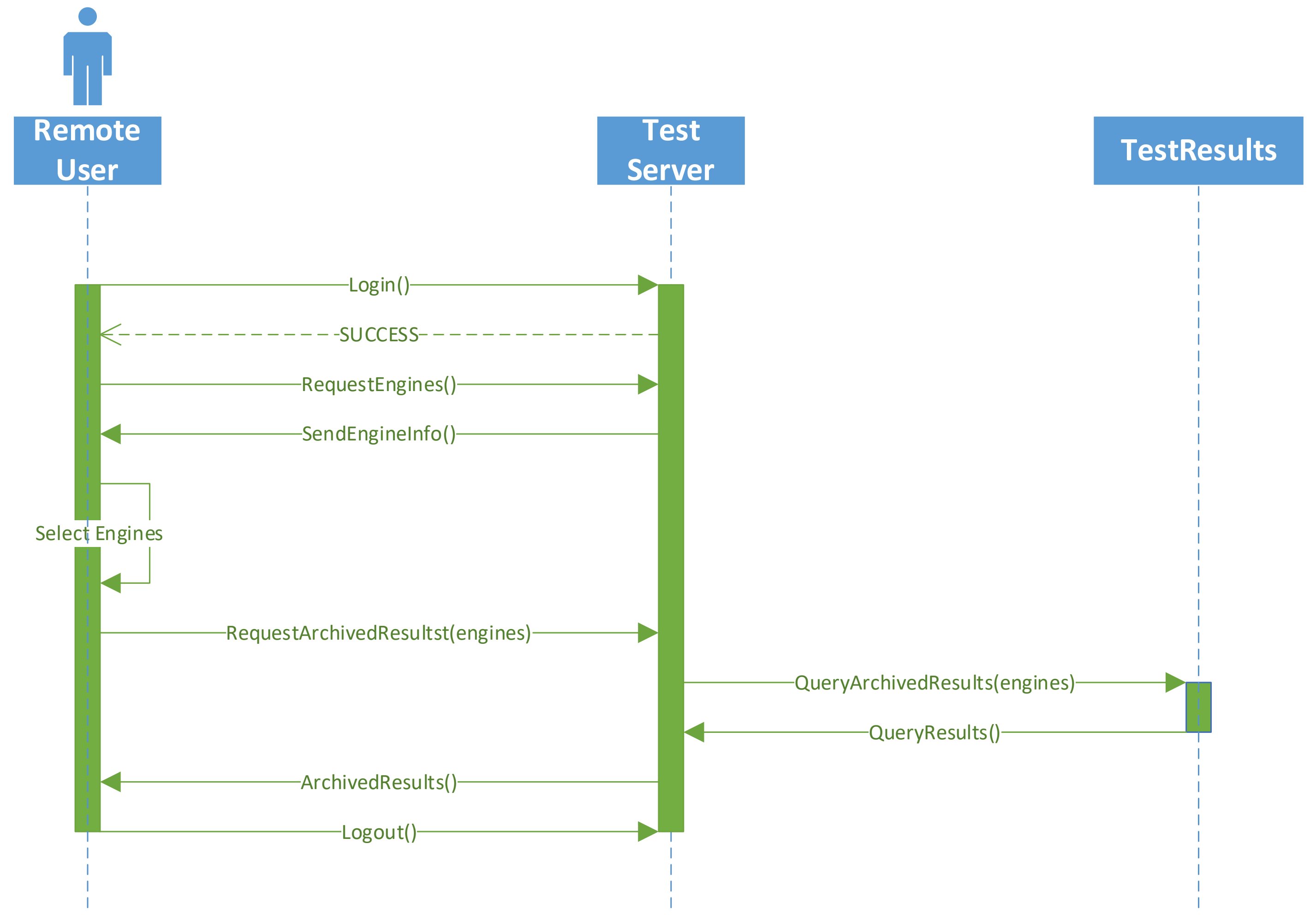
Remote User – Run Test



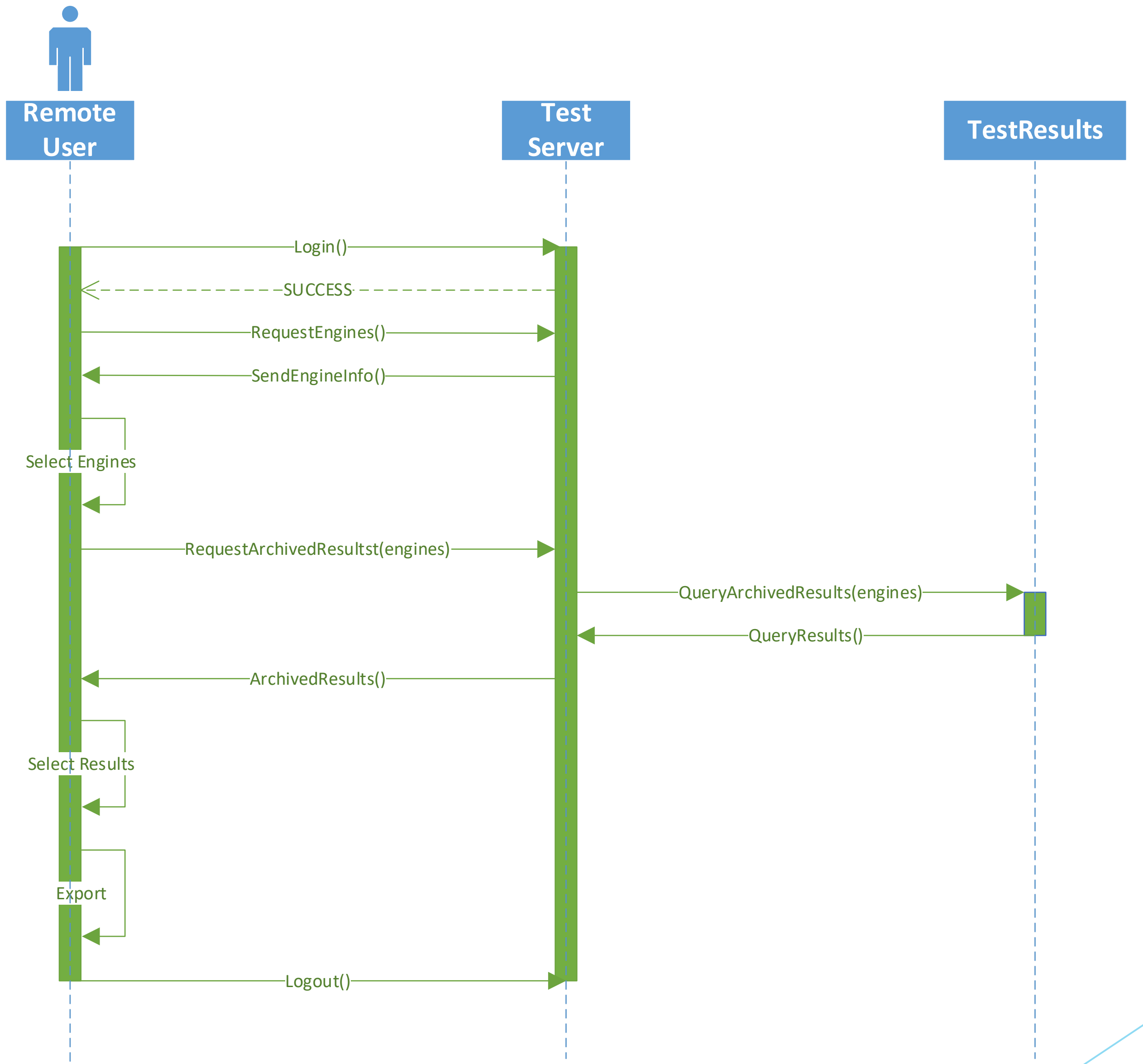
Remote User - View and Configure Tests



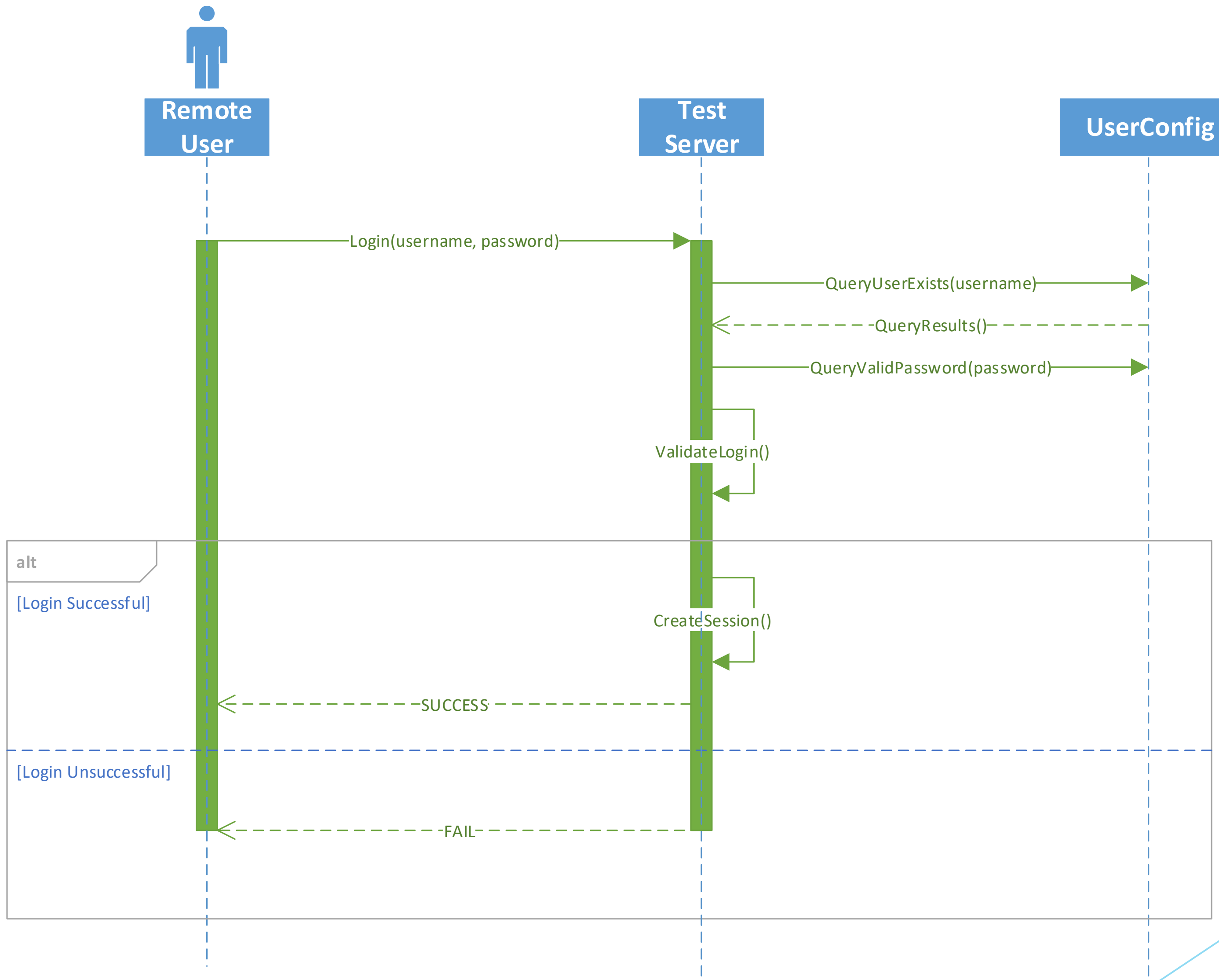
Remote User - View Archived Results



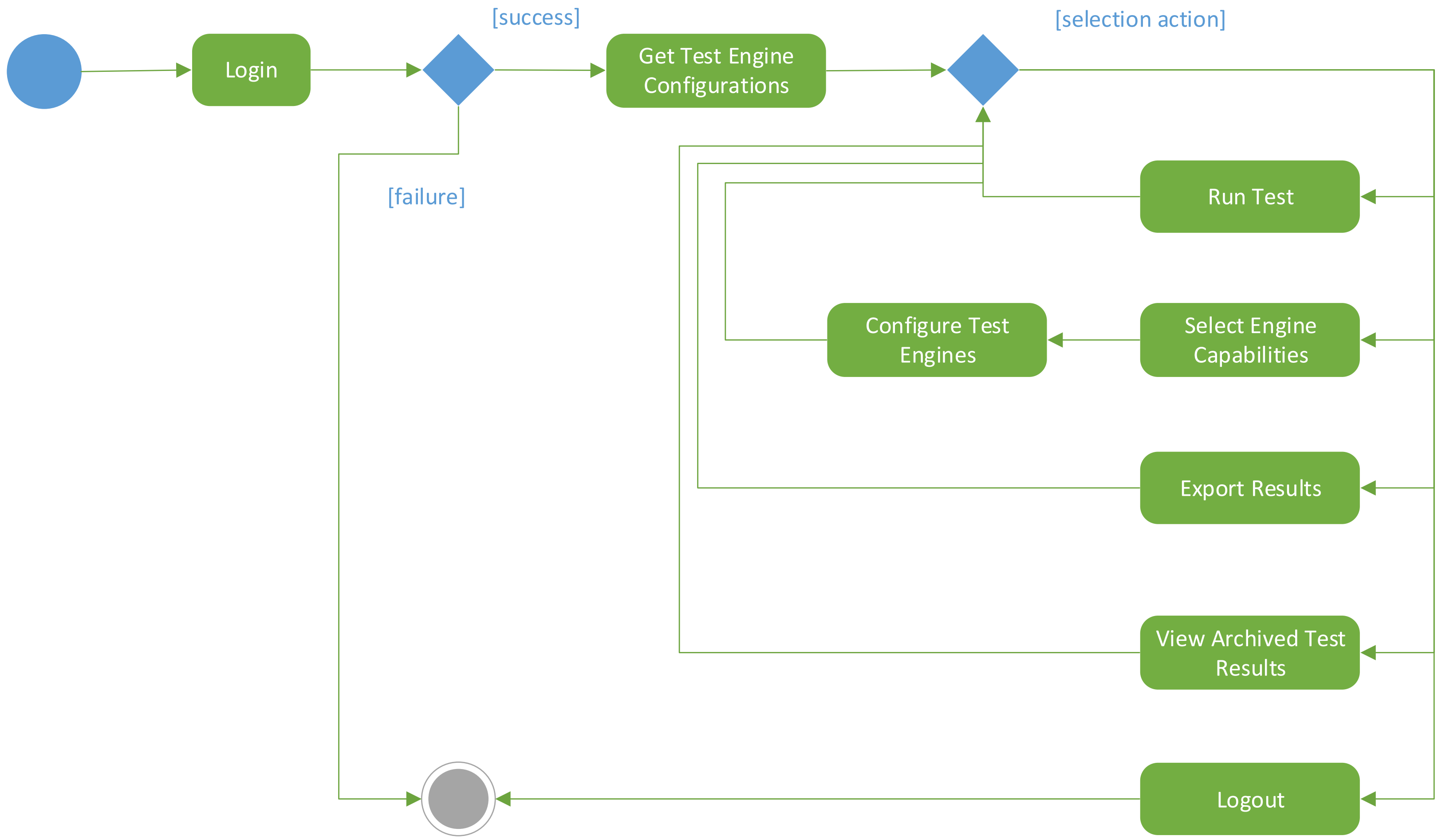
Remote User Export Test Results



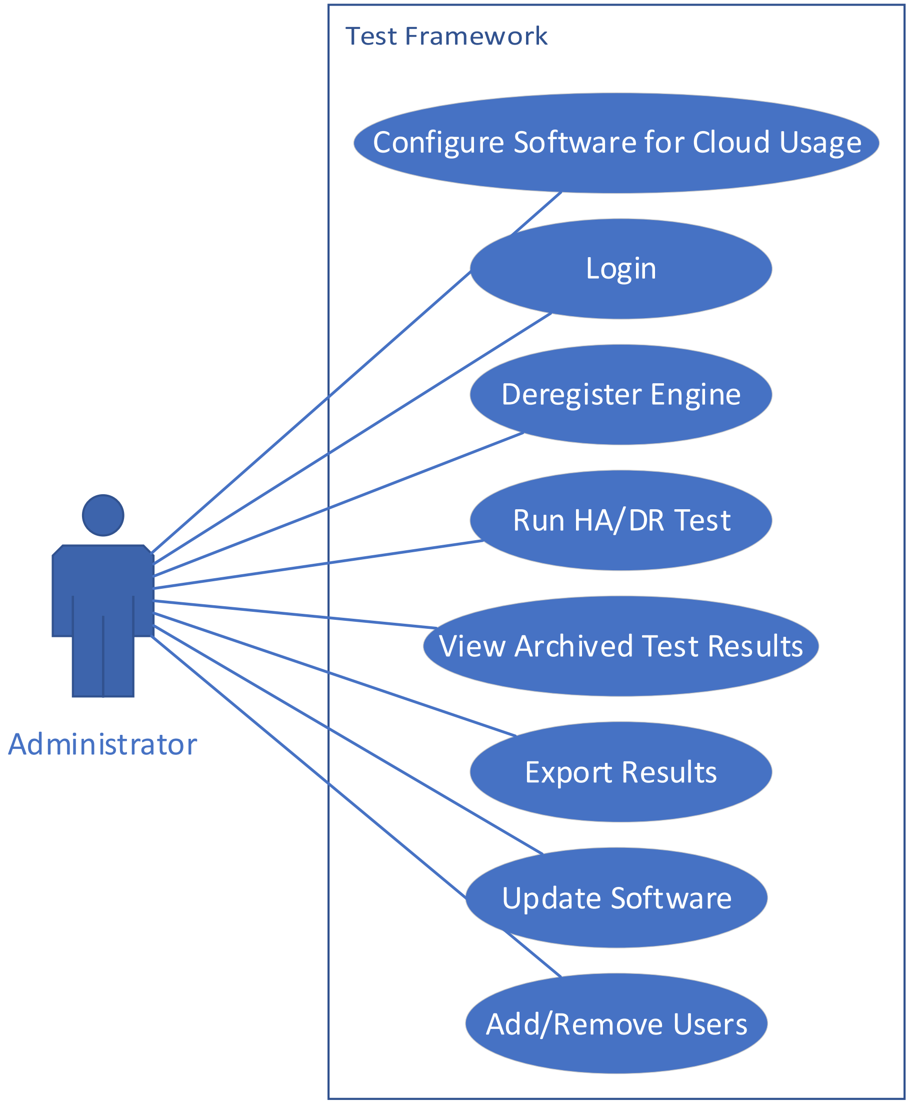
Remote User Login



Remote Use: Activity Diagram

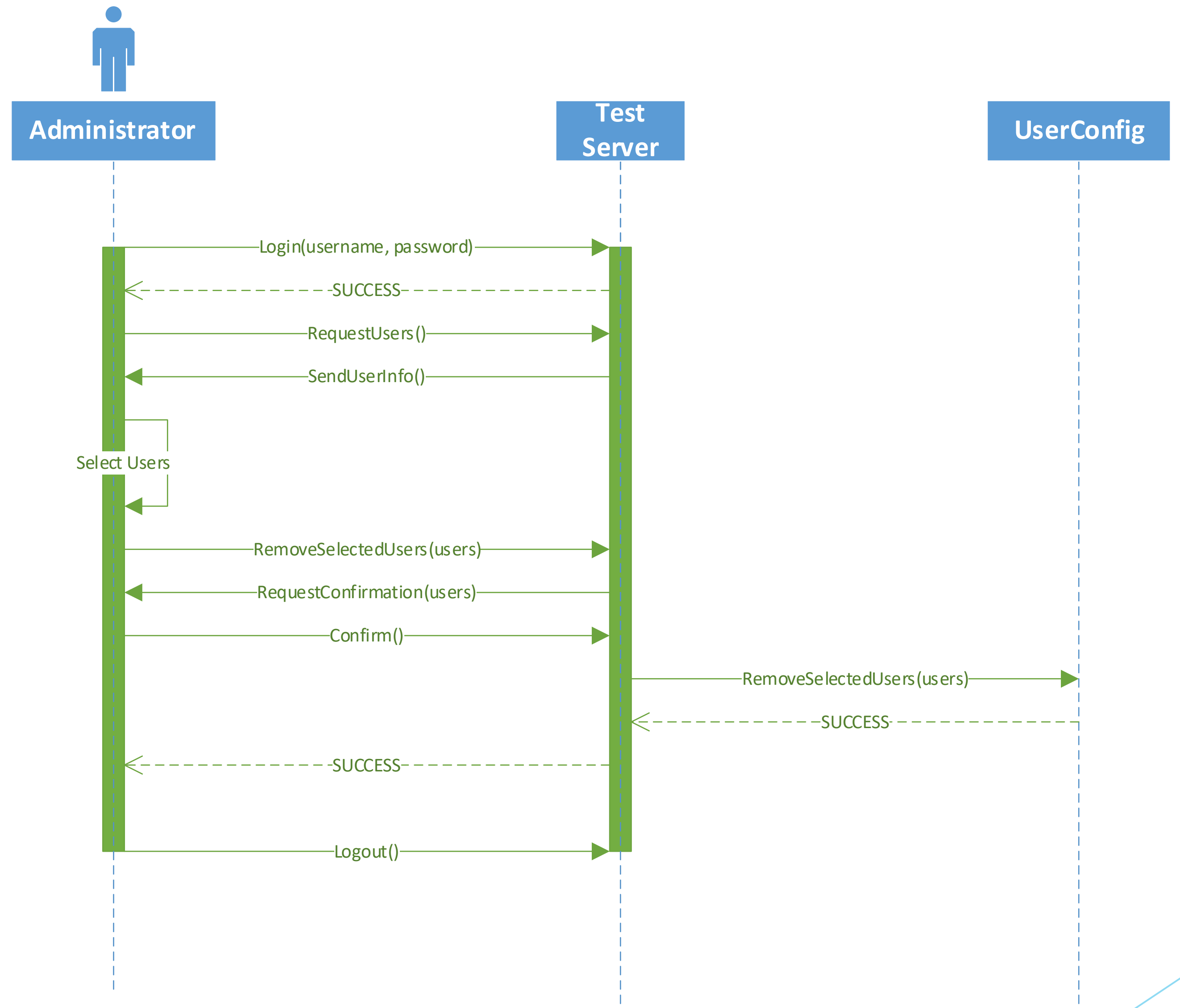


System Models: Use Case - Administrator

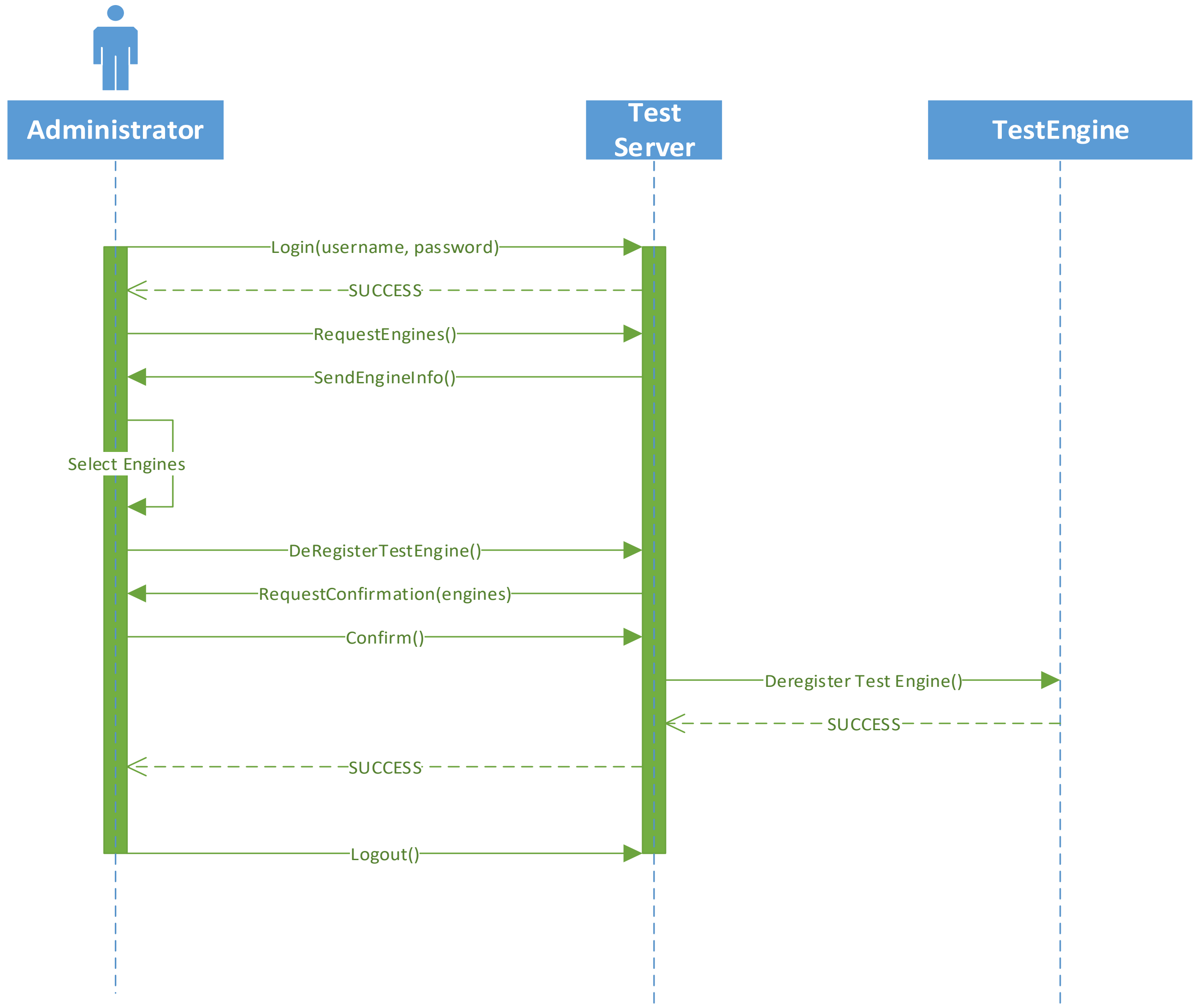


Title:	Administrator Use Case
Description:	The administrator may login to the test server via a web application to perform several actions. These can include viewing test engine configurations so to understand the test platform capabilities, configuring test engines to a selectable to of platform capabilities, run test (or tests) on selected test engines, viewing and exporting archived test for a particular test engine. In addition to the use cases available to the remote user, the administrator can also deregister engines, add/remove users from the system, run HA/DR tests, and configure the software for cloud usage. Upon completion of desired test activities, the administrator may also logout of the system.
Actors:	Administrator
Stimulus (Trigger)	Administrator enters the URL of the web application and successfully logs into the test server.
Preconditions:	Test Server URL is accessible and available to host the administrator web application.
Post conditions:	The desired actions are performed. 1. Test configuration(s) successfully viewed. 2. Test configuration(s) successfully configured. 3. Test(s) successfully executed. 4. Test logs successfully viewed. 5. Test log archives successfully exported. 6. Login successful. 7. Logout successful. 8. HA/DR Test executes successfully. 9. User added to system successfully. 10. User removed from the system successfully. 11. Engine removed from the system successfully. 12. Cloud resources successfully configured.
Main Success Scenario:	1. Administrator logs into the system. 2. Administrator views test engine configurations. 3. Administrator selects platform capabilities and configures test engines. 4. Administrator runs test on test engines. 5. Administrtor logs out. * Other operation supported, as mentioned, such as viewing and exporting log files. The “test run” scenario, however, is the main success scenario.
Extensions:	1. The administrator’s login information is rejected by the web application. → The administrator is presented with the login screen again. 2. Test engines unavailable to execute. → The test engine returns failure back to the Test Server. 3. No test engines match the capabilities selected. → Test engine prohibits test execute. Allows, test capabilities to be reset.
Priority:	2

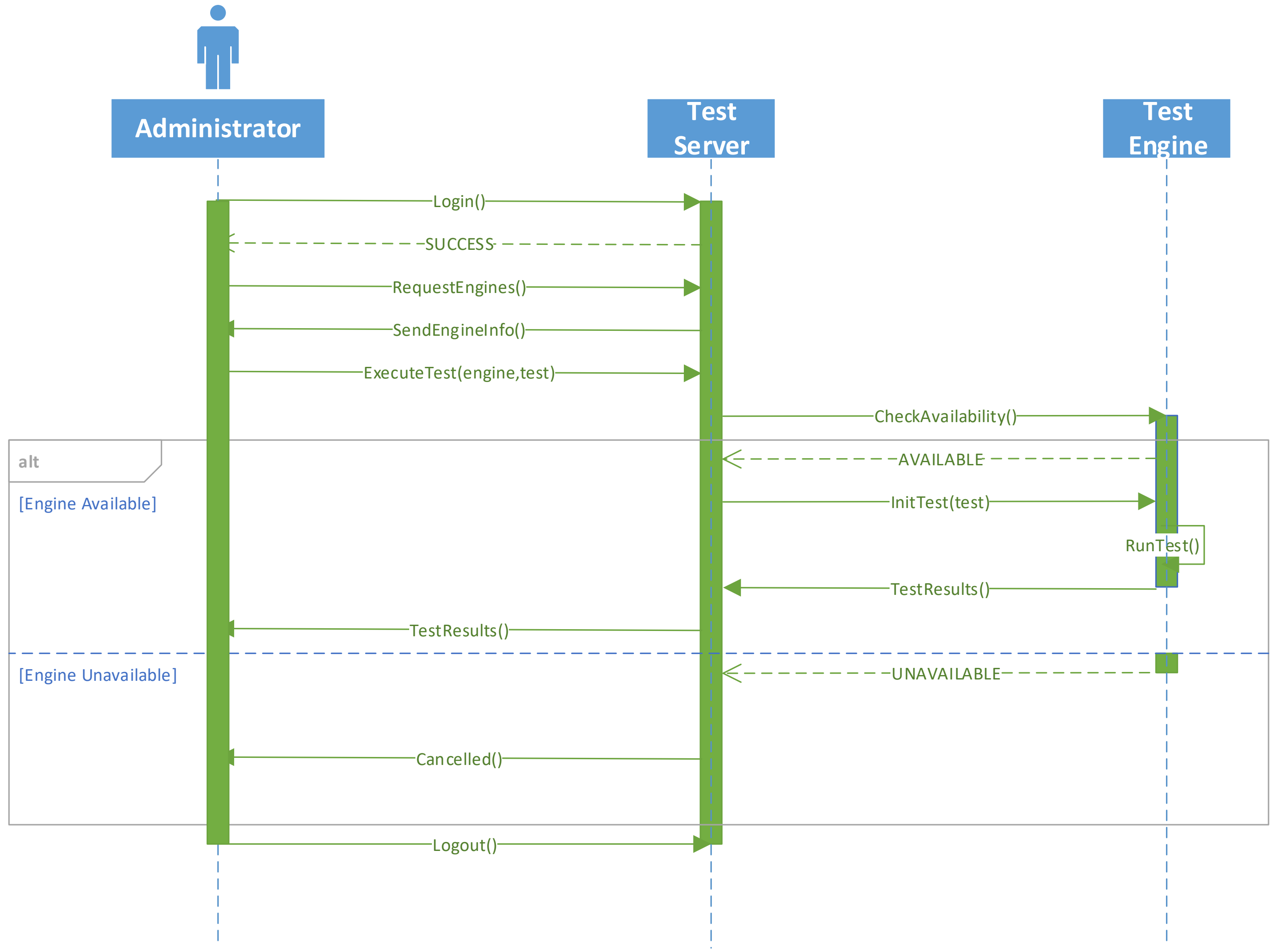
Administrator – Remove Users



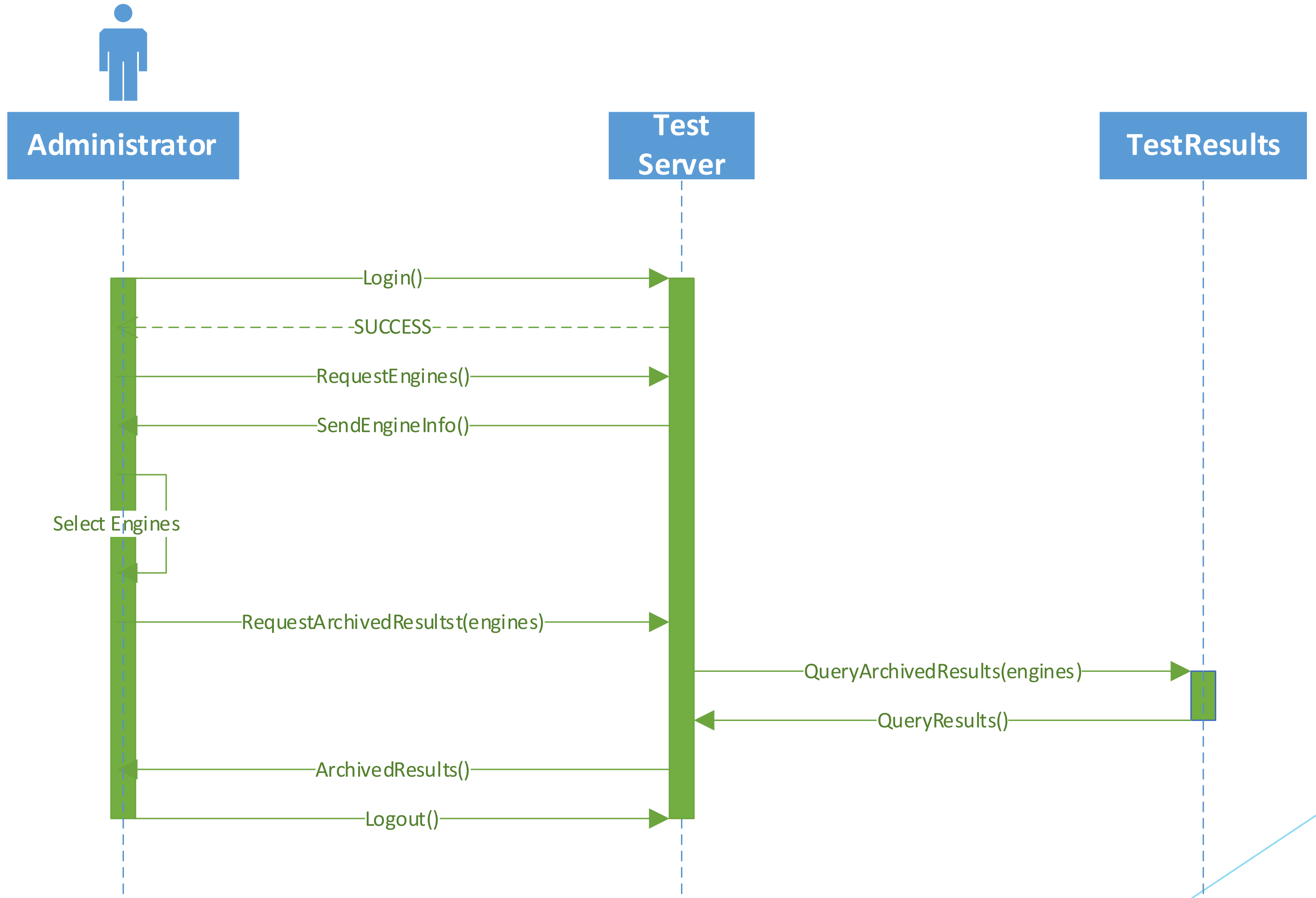
Administrator - Deregister a Test Engine



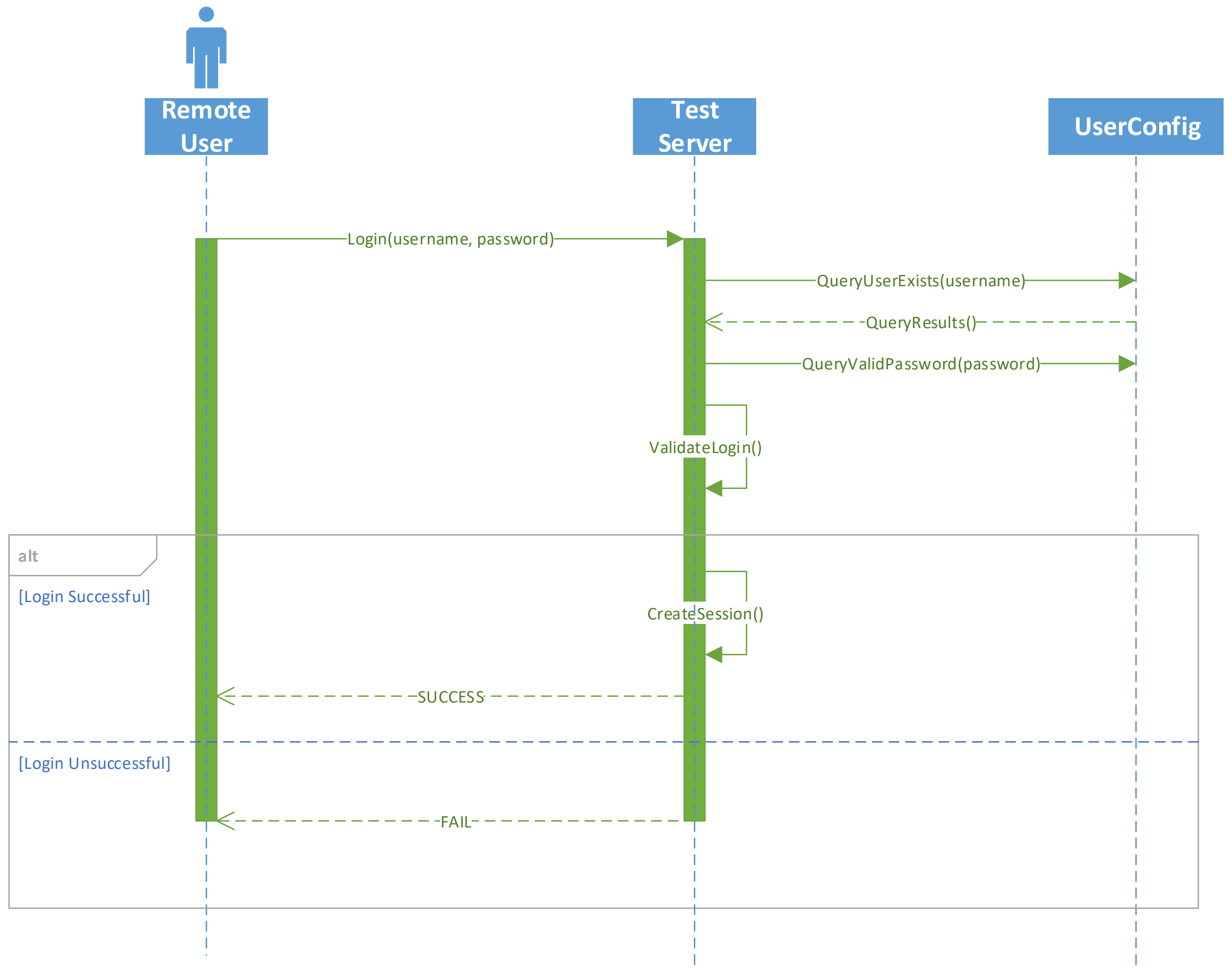
Administrator – Run Test



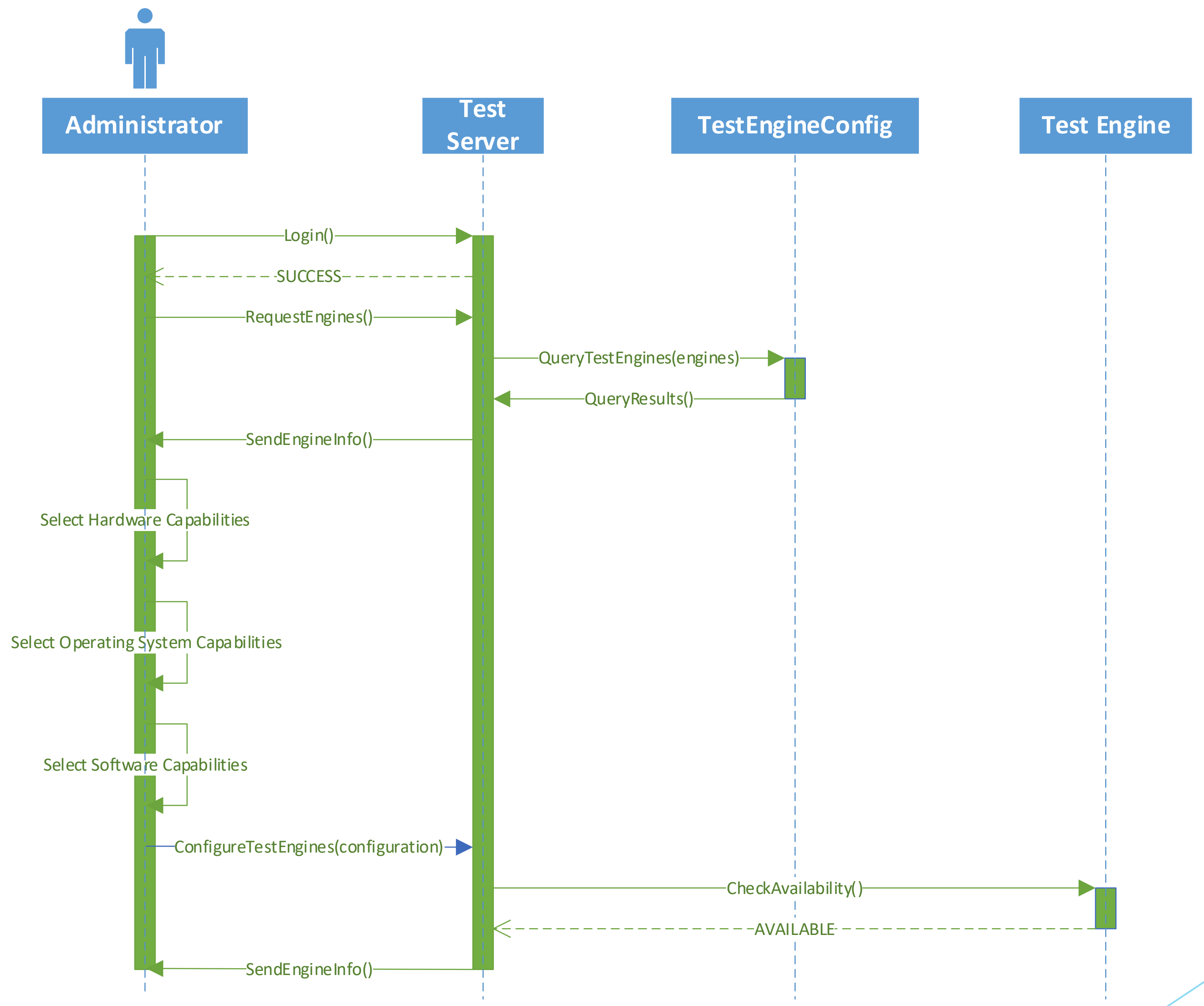
Administrator – View Archived Results



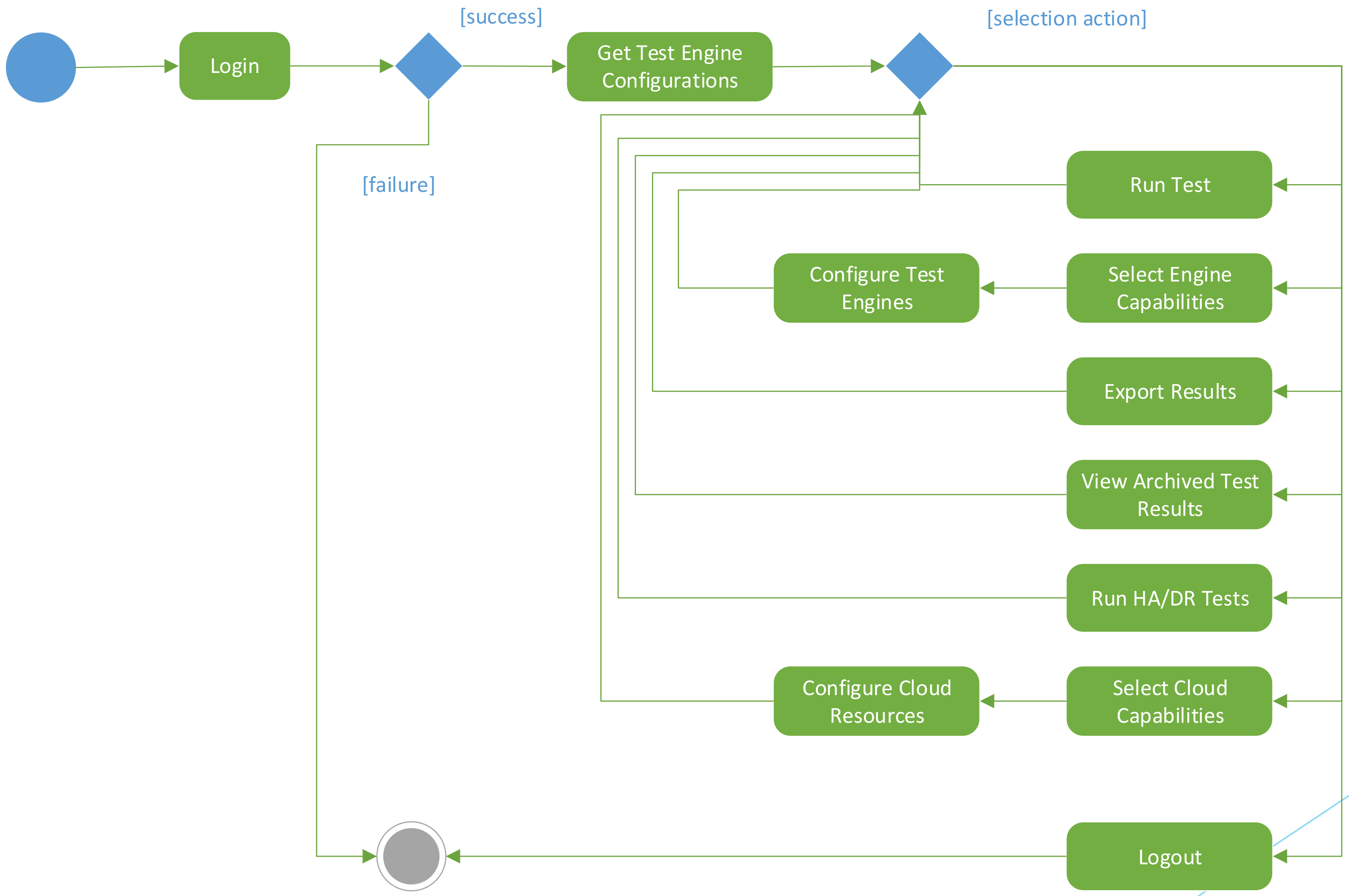
Administrator - Login



Administrator – View and Configure Tests



Administrator – Activity Diagram

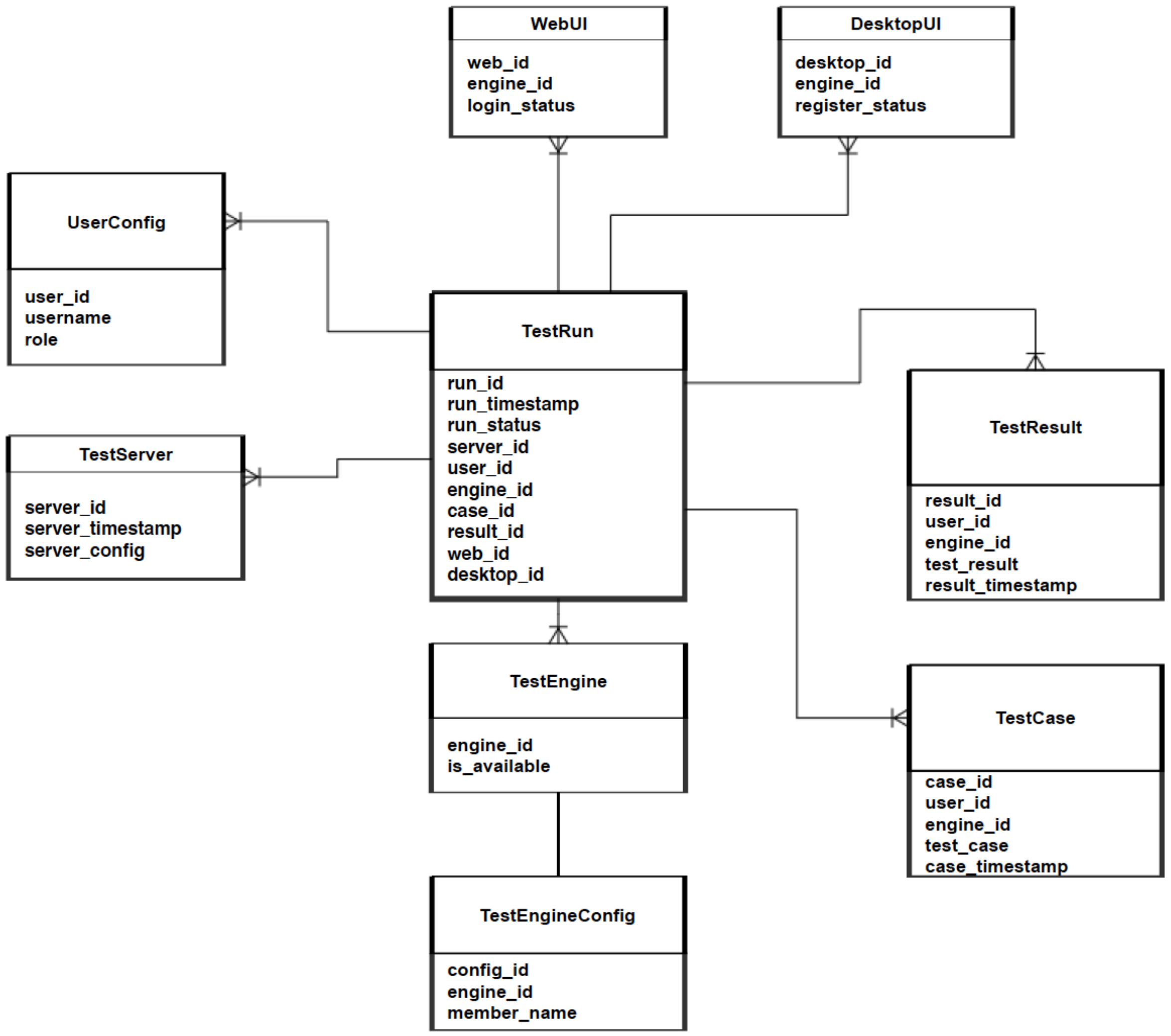

Login
[success]
[failure]
Get Test Engine Configurations
[selection action]
Run Test
Select Engine Capabilities
Configure Test Engines
Export Results
View Archived Test Results
Run HA/DR Tests
Select Cloud Capabilities
Configure Cloud Resources
Logout

Database Specification

- The Test Framework will rely on a relational database system (RDBMS) for a number of purposes and functions as summarized in the following table:

Function	Description	Visibility
System	Persistent system data, logs and meta information, e.g. tracking users registering machines, storing user permissions, storing historical test metadata	Not user visible
Content	Test run content and results	User visible
Privileged Identify Management (PIM)	User profile information, permission levels and location data	User visible

Database Schema



Database Schema

Table	Description
Content Functions	
TestRun	Primary fact table that contains transactional data associated with each test run available to the system. It contains foreign keys to allow for lookups to the dimensional tables.
TestEngine	Contains data related to the test engine and whether is registered for remote use.
TestEngineConfig	Stores configuration meta-data associated with registered test engines.
TestCase	Paths for file-based test case content associated to a particular user and test engine.
TestResult	Paths for file-based test case content associated to a particular user and test engine.
UserConfig	Stores role information for users registered with the test server
DesktopUI	Stores information about local users' test engines which have been registered for remote use.
WebUI	Stores information about remote users' login status and test engines which have been selected for remote use.

*This is not an exhaustive list of all possible tables in the system; additional temporary tables may be added as needed to process data and perform certain functions.

Thank You

- Demo
- Questions?

Backup

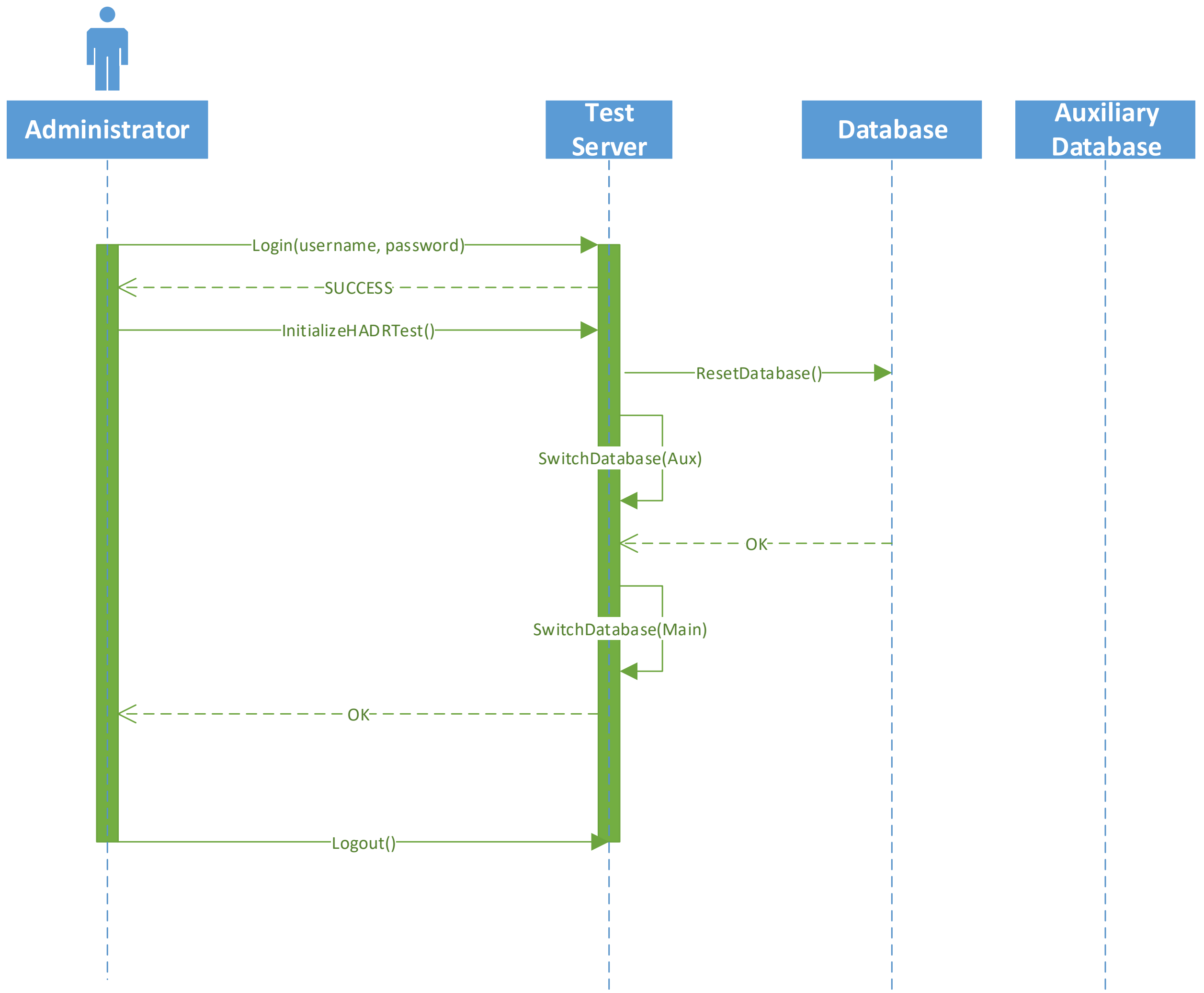
Availability and Business Continuity

- **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]

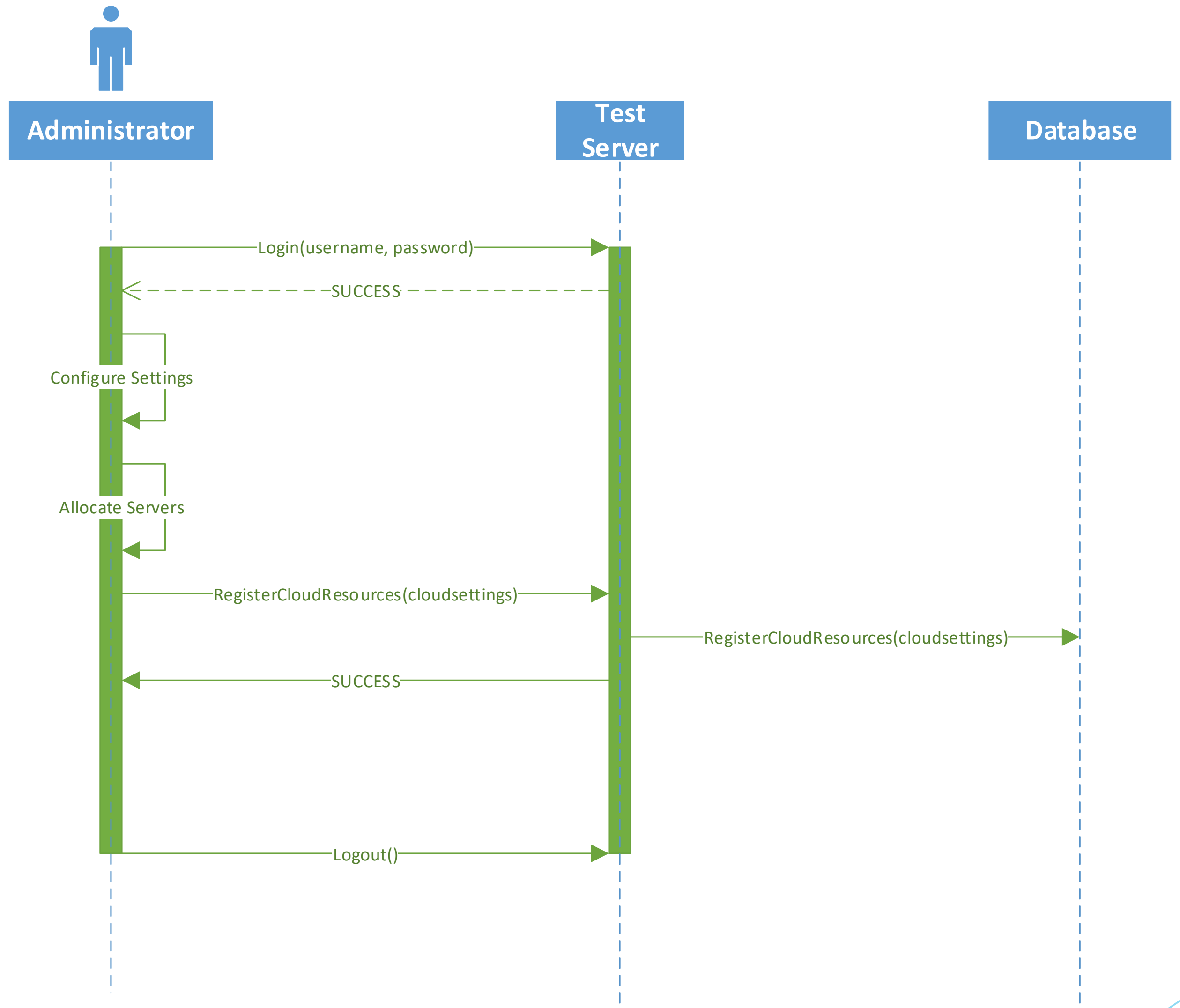
Availability and Business Continuity

- **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]
- 3.3.3 The system shall maintain all program code in scripts that can be deployed to the cloud platform. [1]
- 3.3.3.1 Backup copies of all scripts shall be located in a separate region. [2]

Administrator – HA/DR Tests



Administrator – Configure/Update Settings


Administrator
Test Server
Database
Login(username, password)
SUCCESS
Configure Settings
Allocate Servers
RegisterCloudResources(cloudsettings)
RegisterCloudResources(cloudsettings)
SUCCESS
Logout()