CUBIXEL

Test and Integration Plan

TutorPoint

2019 / 2020

Group 2

MEng Software Engineering Project

Department of Electronic Engineering

University of York

Document Approval

All authors of the document are required to proofread, mandate, and sign-off before the document's official publication.

Author	Signature	Date
Oliver Still	[illegible]	12/03/2020
Daniel Bishop	[illegible]	12/03/2020
Eric Walker	[illegible]	12/03/2020
James Gardner	[illegible]	12/03/2020
Che McKirgan	[illegible]	12/03/2020

Document Version Control

History of edits and alterations to the CUBIXEL Test and Integration Plan, including the document version, date, author, and description of the edits.

Version numbering is based on the significance of change.

Key: (major-change/milestone**.**new-section**.**section-edit)

Version	Author	Date	Section Modified	Remarks
0.0.1	JG	28/02/20	All Sections	Document created and structure laid out. Created all section headers and filled out basic information for each section.
0.0.5	JG	05/03/20	All Sections	1.1 Test Objectives 1.3 System Overview 4.1 Test Team 6 Modules to be Tested
0.1.0	EW	05/03/20	All Sections	N/A
0.1.2	JG	05/03/20	Unit Testing Tools Modules to be tested.	Described tools used for unit testing. JUnit and Mockito. 6 Modules to be Tested
0.1.3	JG	09/03/20	2.4 Unit Testing Example	Included an example with key points highlighted.
0.1.4	EW	10/03/20	8.3, 4.4	N/A
0.1.7	JG	11/03/20	Preface 3.3, 3.3.2, 3.4 6.6 Iteration 6	Document abstract. Integration testing tools. Manual/Automated integration testing. Added Iteration 6 and modules.
0.2.2	JG	11/03/20	Section 3/4 3.5 Test Methods 5.4 Risks	Moved Integration Testing Procedure to better position. Wrote descriptions of Black, Grey and White box testing.
0.2.3	EW	11/03/20	Section 6,7, 4	Added to sections 6/7. Restructured and edited section 4.
1.0.0	**OS**	**11/03/20**	**All Sections**	**Document approved by team.**

Preface

This document is the Test and Integration Plan for CUBIXEL software development and specifically the development of the TutorPoint application. It provides an overview of the TutorPoint product, detailing the areas of the product that are suitable for testing. It provides an in depth view of the types of testing processes that will be used, the tools required for those processes and the members of CUBIXEL responsible for overseeing these tests. A plan for how contracted modules, developed by external teams, will be sufficiently tested and integrated is also described. Iteration plans and expected module integration testing has been described along with a the full integration testing process, documentation and fault reporting procedure.

The company, Cubixel (“we”, “our”, “us”), may be stylised as “CUBIXEL” and may be referred to as the COMPANY.

Contents

1. Introduction

1.1 Test Objectives

Unit Testing

CUBIXEL will use Test Driven Development (TDD) throughout all software production to minimise unnecessary/redundant code, to ensure that all code produced is performing as expected, and to reduce time after integration ensuring that each module still performs as expected once merged with the existing code base.

Integration / Exploratory Testing

Integration testing should expose faults in the interaction between integrated modules after the modules have been combined into a common code base. With seperate groups working on distinct but interacting features it is imperative that any issues with these interactions are highlighted to developers early. It is also important that the root cause of the bug is understood and the software team made aware to reduce similar bugs in future code. Integration testing should:

- Ensure all code is signed off by the Senior Development Team before being integrated to the main code base.
- Ensure all unit tests still pass successfully post integration/merge.
- Confirm the correct functioning of interacting modules.
- Highlight any bugs and understand the root cause of those bugs.
- Ensure development teams are made aware of bugs and their causes.

Contracted Module Testing

Modules that have been contracted to another party will be tested via Integration Testing.

System Testing

System Testing will be the final stage in testing, run after the end of development. Its aims are similar to integration testing, to ensure that the code that has been developed works as intended, and runs without bugs.

1.2 Scope of Testing

We will perform two varieties of testing during Development, each with their own scope

- Test Driven Development: only concerned with testing the functionality of small blocks of code within a user story.
- Integration Testing: performed on completed user stories after integration into the development branch, concerned with finding bugs in the entire user story's section of code, and ensuring that the user story interacts as intended with the rest of the code.

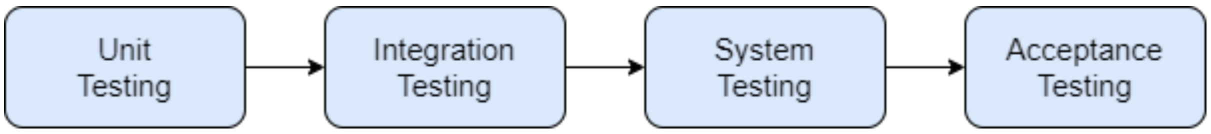


Figure 1 - Flow diagram of the general model testing sequence.

1.3 System Overview

TutorPoint is a desktop based, Java application intended to provide a digital lecture environment for both students and lecturers. It is open to any subject that a tutor would like to teach but with a particular focus on STEM subjects at the college and university level. It provides the tools to enable delivery of lecture content live to an audience with instant feedback from users on topics being covered. Users will be able to create an account as either a student or a lecturer and begin watching or producing content on a subject of their choice.

TutorPoint will provide an online platform for students and specialists to share, converse and develop their respective fields. The platform will incorporate a login and register screen with information provided by the user sent to a server and stored/checked against a TutorPoint database. It will include a Main Landing Screen with information on current live and pre-recorded content. The lecture environment screen will combine a section for XML presentations with an interactive whiteboard and text/video chat.

CUBIXEL are also developing the back-end, server-side application that will run all TutorPoint services and development of this will follow all the same Unit and Integration testing as the front end. The Servers functionality includes the ability to handle multiple clients simultaneously. Access to a MySQL Database storing all necessary program data from user details to subject information and live tutors as well as the location of various resources for presentations e.g. videos and pictures.

CUBIXEL will handle the content distribution of both pre-recorded and live lecture content, the media will be multicast from a central server to multiple clients when required.

1.4 Condensed Program Flow Diagram

The condensed program flow diagram details all the required screens and views in order to meet the program’s functionality. A full version of the program flow diagram can be found in the TutorPoint Functional Specification document.

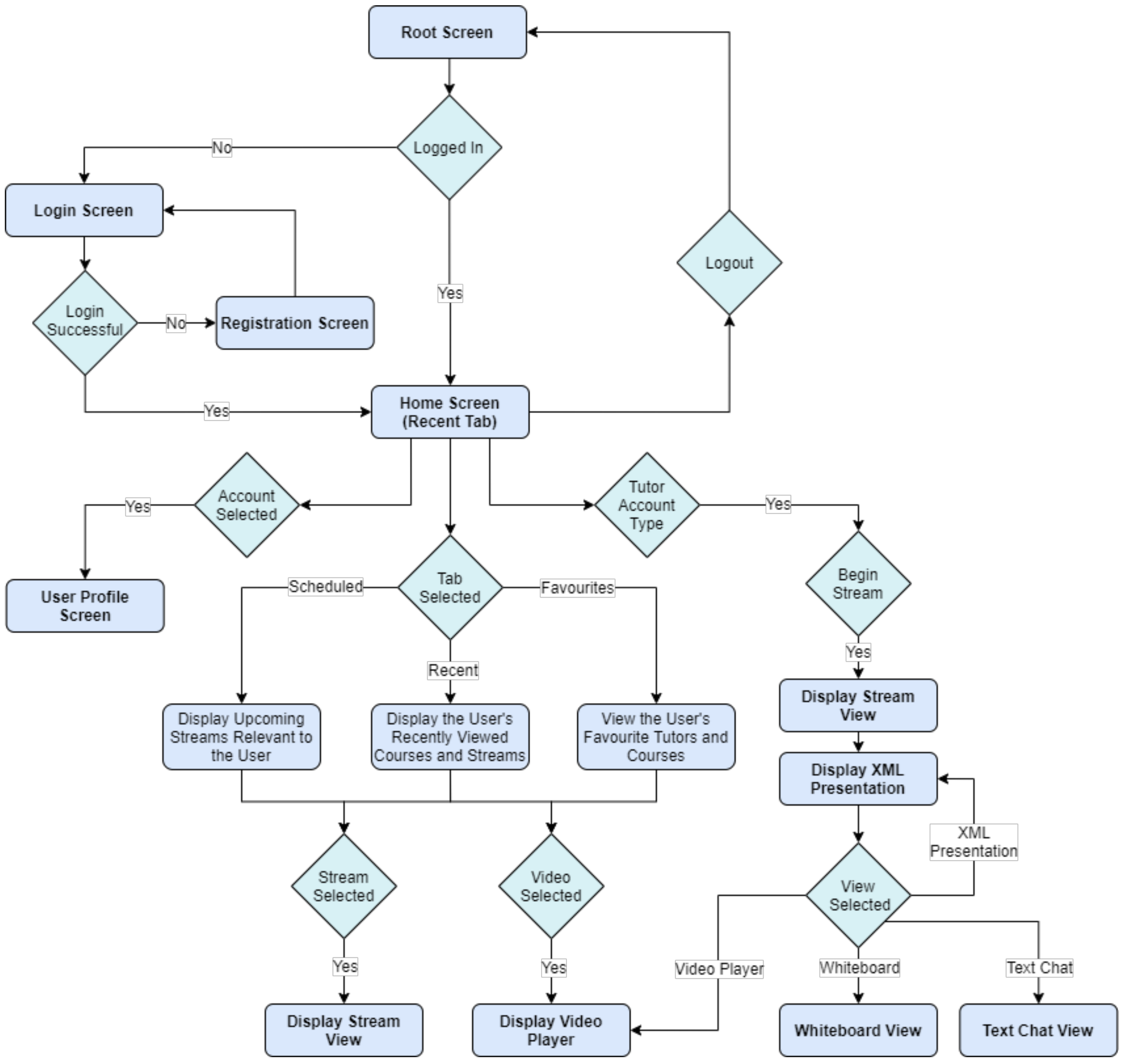


Figure 2 - Condensed program flow diagram of TutorPoint.

2. Unit Testing Methodologies

2.1 Unit Testing Introduction

Unit testing will be used to implement Test Driven Development throughout all production code. All members of the development team are expected to write Unit Tests for any testable function of their code with the aim that if sufficient and detailed unit tests are written, then the time spent finding integration issues should be significantly reduced.

2.2 Unit Testing Objectives

The objectives of unit testing are to:

- Use TDD to drive the development of all new features.
- Target specific modules/methods and should be isolated from other program dependencies.
- Be maintainable and readable.
- Verify a single-use case.
- Target a small section of code.

2.3 Unit Testing Tools

JUnit5

The JUnit Platform serves as a foundation for launching testing frameworks on the JVM. All unit tests will be written using the JUnit framework. JUnit provides:

- Annotations to identify test methods.
- Assertions for testing expected results.
- Test runners for running tests.
- Tests can be run automatically and they check their own results and provide immediate feedback.
- Tests can be organized into test suites containing test cases and even other test suites.
- Test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails providing immediate visual feedback.

Mockito

Mockito is a mocking framework that enables mocking of Java objects for more readable tests with stronger separation from other modules. This enables tests to directly target a module's functionality without worrying about its dependencies. It features:

- Mocks concrete classes as well as interface.
- Annotation syntax - @Mock
- Verification errors are clean - click on stack trace to see failed verification in test; click on exception's cause to navigate to actual interaction in code. Stack trace is always clean.
- Allows flexible verification in order (e.g: verify in order what you want, not every single interaction)
- Supports exact-number-of-times and at-least-once verification
- Flexible verification or stubbing using argument matchers (anyObject(), anyString() or refEq() for reflection-based equality matching)
- Allows creating custom argument matchers or using existing hamcrest matchers

2.4 Unit Testing Example

```java
public class LoginServiceTest {

  private LoginService loginService;

  private String returnedString;
```

Use of Mocks to only test UUT

```java
  @Mock
  private MainConnection mainConnectionMock;

  @Mock
  private Account accountMock;

  /**
   * Sets up the JavaFX Toolkit for running JavaFX processes on.
   */
  @BeforeAll
  public static void setUpToolkit() {
    /* This method starts the JavaFX runtime. The specified Runnable will then be
     * called on the JavaFX Application Thread. */
    Platform.startup(() -> System.out.println("Toolkit initialized ..."));
  }


  /**
   * Initialises Mocks, sets up Mock return values when called and creates
   * an instance of the UUT.
   */
  @BeforeEach
  public void setUp() {
    initMocks(this);
```

Set Mock function call return

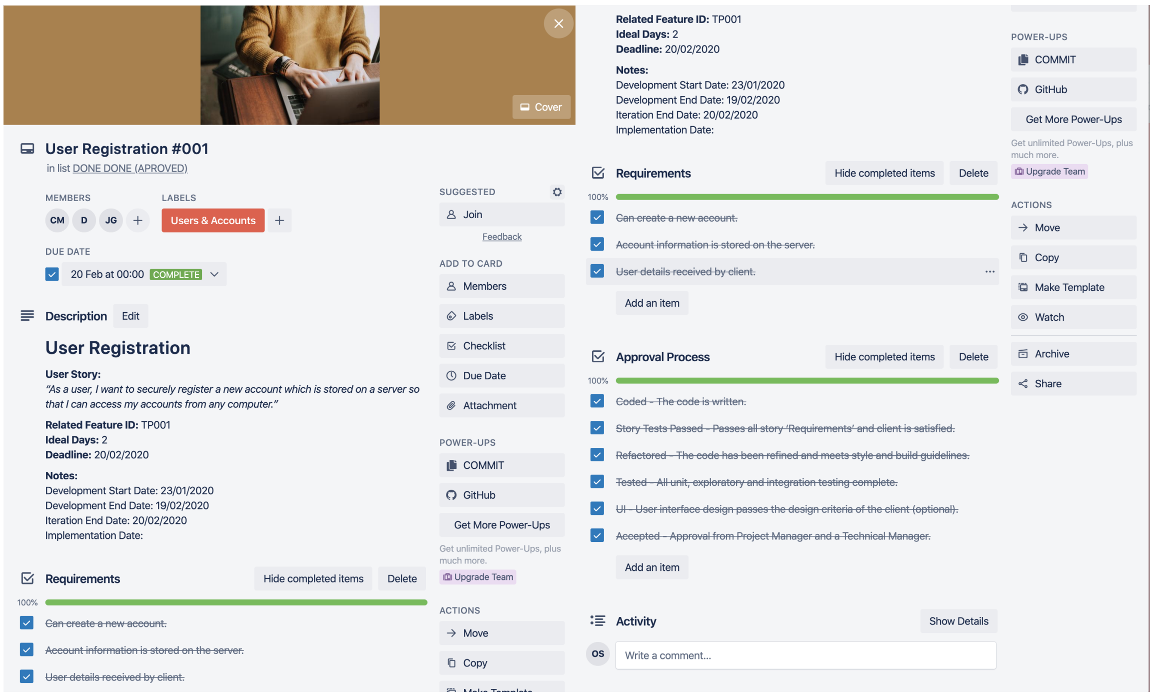
```java
    try {
      when(mainConnectionMock.listenForString()).thenReturn(returnedString);
    } catch (IOException e) {
      fail(e);
    }

    loginService = new LoginService(accountMock, mainConnectionMock);
  }

  @Test
  public void successfulResultTest() {
    // Setting Mock return value.
    returnedString = String.valueOf(AccountLoginResult.SUCCESS);

    Platform.runLater(() -> {
      loginService.start();
      loginService.setOnSucceeded(event -> {
        AccountLoginResult result = loginService.getValue();

        assertEquals(AccountLoginResult.SUCCESS, result);
      });
    });
  }
```

Testing one specific function

```
  @Test
  public void networkFailResultTest() {
    // Setting Mock return value.
    returnedString = String.valueOf(AccountLoginResult.FAILED_BY_NETWORK);

    Platform.runLater(() -> {
      loginService.start();
      loginService.setOnSucceeded(event -> {
        AccountLoginResult result = loginService.getValue();

        assertEquals(AccountLoginResult.FAILED_BY_NETWORK, result);
      });
    });
  }

  @Test
  public void unexpectedErrorResultTest() {
    // Setting Mock return value.
    returnedString =
              String.valueOf(AccountLoginResult.FAILED_BY_UNEXPECTED_ERROR);

    Platform.runLater(() -> {
      loginService.start();
      loginService.setOnSucceeded(event -> {
        AccountLoginResult result = loginService.getValue();

        assertEquals(AccountLoginResult.FAILED_BY_UNEXPECTED_ERROR, result);
      });
    });
  }

}
```

2.5 Unit Testing Checklist for User Stories

All User Stories must have been checked for unit testing before the software portion of their progress through an iteration is considered complete. This is demonstrated in the Trello (the project management tool used by CUBIXEL) cards for each user story, as shown below:



User Registration #001

in list DONE DONE (APROVED)

MEMBERS: CM D JG

LABELS: Users & Accounts

DUE DATE: ✓ 20 Feb at 00:00 COMPLETE

Description Edit

User Registration

User Story:
"As a user, I want to securely register a new account which is stored on a server so that I can access my accounts from any computer."

Related Feature ID: TP001
Ideal Days: 2
Deadline: 20/02/2020

Notes:
Development Start Date: 23/01/2020
Development End Date: 19/02/2020
Iteration End Date: 20/02/2020
Implementation Date:

Requirements Hide completed items Delete

100%

- ✓ ~~Can create a new account.~~
- ✓ ~~Account information is stored on the server.~~
- ✓ ~~User details received by client.~~

Related Feature ID: TP001
Ideal Days: 2
Deadline: 20/02/2020

Notes:
Development Start Date: 23/01/2020
Development End Date: 19/02/2020
Iteration End Date: 20/02/2020
Implementation Date:

Requirements Hide completed items Delete

100%

- ✓ ~~Can create a new account.~~
- ✓ ~~Account information is stored on the server.~~
- ✓ ~~User details received by client.~~

Add an item

Approval Process Hide completed items Delete

100%

- ✓ ~~Coded – The code is written.~~
- ✓ ~~Story Tests Passed – Passes all story 'Requirements' and client is satisfied.~~
- ✓ ~~Refactored – The code has been refined and meets style and build guidelines.~~
- ✓ ~~Tested – All unit, exploratory and integration testing complete.~~
- ✓ ~~UI – User interface design passes the design criteria of the client (optional).~~
- ✓ ~~Accepted – Approval from Project Manager and a Technical Manager.~~

Add an item

Activity Show Details

OS Write a comment…

Figure 3 - Example of a completed user story card on Trello. It details the tests that need passing and all stages of the approval process must be completed before the card can move to 'Done, Done'.

3. Integration Testing Methodologies

3.1 Assumptions / Constraints

3.1.1 Assumptions

- The first build of the TutorPoint system will be ready for system integration testing on Date, 05/03/2020.
- Each build of the TutorPoint system will have passed unit testing before it is transferred to the integration testing team.
- Each Build of the TutorPoint System will have been fully refactored before it is submitted for integration testing

3.1.2 Constraints

Constraints on the quality and quantity of integration testing include:

- Time. Each iteration lasts two weeks and integration testing on the previous iteration must be completed within **five days** of the end of the iteration.
- Money. There is a limited budget assigned to each member of the integration and testing team. This means there is a limit to the number of hours that integration testing can be performed.
- System Completion. Some test modules may not be able to have all of their interactions with other modules tested due to the other modules having not yet been written
- Content Completion. Some test modules may not have all of the content they are required to display completed yet. When this occurs the module will be tested the dummy content to simulate the minimum required features of the module.
- For some modules with minimal interaction with the rest of the code or that have no notable user interaction, integration testing may be redundant. Unit Testing will suffice in these instances.

3.2 Coverage

3.2.1 Software Components

All modules on the Client / Front End side will be tested, this includes:

- Interactive Whiteboard
- XML Presentation
- Media Player and Video Chat
- Text Chat
- GUI / Useability

All modules on the Server / Back End side will be tested, this includes:

- Client-Server Interactions.
  - Login
  - Register
  - Live Whiteboard
  - Multiple Clients
  - Streaming Media

- Server-MySQL Interactions.
    - User Details
    - Subjects
    - Live Tutors
    - Content Locations

3.2.2 Customer Requirements

Each module will be checked against the functional specification to ensure that it is meeting the customer requirements. When requested, throughout development, the most complete, fully functional system snapshot will be presented to the customer to ensure that development is producing desired results. These demos will be used to gain customer feedback and demonstrate successful interactions of core components.

3.3 Integration Test Tools

TutorPoint and all CUBIXEL software is developed using a Multi-module Maven Project structure. One of these modules should always be a global testing module. This enables the use of JUnit5 and Mockito Tests to be written that confirm the successful integration of multiple modules by simulating their interactions. This will be used to quickly confirm the code stability following each iteration user stories merge into the development branch.

3.4 Test Types

3.4.1 Manual Testing

Manual testing will be performed on each module by the testing team during the integration phase of each iteration. Manual testing involves the testers physically interacting with the program. Testers create tests, performing those tests and comparing the result against expected output. During System Integration Testing, testing will be performed in a grey box fashion.

The system will be tested:

- For functionality - to ensure that the new modules perform the tasks that they are expected to do
- For errors - deliberate attempts to cause the program to crash or error
- For security - can the program be breached, or user or company data be stolen?
- Regression testing - have the newly introduced modules cause errors elsewhere in the program

Tests will be performed in an exploratory fashion, with a 'charter' stating the aim of the testing. Tests will then be performed to fulfill this charter, with the results of each test determining what test to perform next.
Examples of the varieties of test that may be performed include:

- **Boundary Testing** - Inputting values that are just below, right on or just above a limit on an input field
- **Quantity Testing** - Testing how the program responds to having none, one or many of something

- **Input Sanitisation** - Ensuring that input fields cannot be used to input external code, e.g. SQL Injection.

Manual testing may not always be completely accurate due to human error. Therefore, the results of manual testing should be used to inform the process of writing automated integration tests.

3.4.2 Automated Testing

Using a seperate Testing Maven Module, JUnit and Mockito, automated tests between interacting modules can be written. This can be used to quickly check that any changes have not affected essential functionality. Writing these tests after performing Manual Testing means that the development code can be handed back to developers with a set of failing test cases, enabling the development team to quickly address these issues.

Automated Testing also ensures that once bugs have been fixed, they no longer reoccur as new features are written as these global integration tests are now a core feature in the development teams coding branch and must be passed before any pull request is accepted.

3.5 Test Methods

3.5.1 Black Box Testing

Black box testing is a testing method that analyses the performance or functionality of a system without knowledge of its internal structure. In the case of code this would be done without looking at the source code for the module or system under test. This could be used when integrating other company IP into the software or when simulating a hack into the server or database.

Black Box Testing:

- No knowledge of internal system structure.
- Trial and error testing.
- Simulating attacks on the system.
- Less exhaustive than Grey or White Box testing.
- End user can be involved.

3.5.2 Grey Box Testing

Grey box testing is a testing methodology that lays between black and white box testing. It could involve access to documentation on a module function or a description of the algorithms used. This method can be beneficial as it takes the simplicity of black box testing and combines it with some of the targeted testing methods of white box testing.

Grey Box Testing:

- Partial knowledge of internal system structure.
- Not suitable for algorithmic testing.
- Less time consuming than white box testing.
- Based on high-level system descriptions.

3.5.2 White Box Testing

White box testing is a testing method that performs tests whilst full knowledge of the internal working and structure of the system being tested is known. For example having full access to the source code when performing integration tests. This form of testing means that specific tests can be planned to target features or weaknesses of the design. This form of testing lends itself well to being automated as tests can be written with advanced knowledge of the expected outcome and checked against this.

White Box Testing:

- Full knowledge of internal working structure of UUT.
- Deeply explores internal workings of a system.
- Suitable for algorithmic testing.
- End user not involved.
- Most exhaustive testing method.

3.6 Test Data

Some tests may require external data to be run. This data will be created by the tester, and includes:

- Database of user accounts and logins, stored on the company Raspberry Pi.
- Example XML presentations, stored locally to tester.
- Test media files such as audio, video and images.
- Test corrupted data for system stability.

4. Integration Testing Process

4.1 Testing Procedure

For each requirement or system feature to be tested, the tester will execute a set of predefined test cases. Each test case will have a series of actions and expected results. As each action is performed, the results are evaluated. If the observed results are equal to the expected results, the test is passed. If the observed results are not equal to the expected results, a note of the result and an evaluation of failure severity is made. Within each iteration's Integration Testing, each module will be tested according to a charter. For each of these user story charters, tests will be performed, with each test helping to determine the next test.

The results of each of these tests will be recorded and used to determine:

- Whether the iteration is accepted without needing further development or bug-fixing.
- If a development process is the cause of the bugs and needs addressing.
- If developers need training in a certain development process.
- If unit testing is being performed correctly.

4.2 Testing Documentation

Use the TutorPoint Integration & Testing Report shown in the Appendix.

4.3 Regression Testing

When functionality is added to the software, we will perform regression testing on any adjacent sections of code (modules that either rely on or are relied on by the added module) to ensure that no new errors are introduced by the addition.

4.4 Order of Testing

User stories within the same iteration can be tested in any order, but a logical order should be followed to ensure all areas are tested. For example, if two user stories follow each other in the program flow, they should be tested in order.

If there are user stories that need to be tested with higher priority, however, for example a user story that failed a first round of integration testing and was sent back to be corrected by the developer, that user story should have a higher priority than new user stories. Older such stories have priority over newer ones.

4.5 Pass/Fail Criteria

After each round of Integration Testing, the system must:

- Not have any defects found at or above a Severity of 3 (section 4.6)
    - Or, any found bugs must have been fixed and re-tested.
- All features must work as described.

If these criteria are met, the Iteration passes, otherwise, it does not until they are met.

4.6 Error Handling

If a test result returns something other than what was expected the level of defect should be attached to the report. Each level determines the priority of the team to address the issue.

Severity	Level	Symptom	Response
5	Critical	Software won't launch. Full system crash / freezes.	Development team responsible for the module will halt all further work until a patch is released fixing the issue.
4	High	Soft-crash. Locked into the program section. Infinite loops. Security flaw. Risk to user data or of server crashes.	Development team responsible for the module will halt all further work until a patch is released fixing the issue.
3	Medium	Incorrect data or program behaviour, without crashing. No risk to user data.	Development team responsible for the module will halt all further work until a patch is released fixing the issue.
2	Low	Aesthetic bug that is easily fixed, e.g. by reloading screen.	Make note of bug, pass back to developers. Do not halt development.
1	Info	No effect on the program but could be improved.	Inform developers/relevant team member.
0	Pass	Used for successful tests.	No action.

4.7 Issue Reporting

Once integration testing has been completed and a set of known issues has been identified it is important that the development team responsible for those modules that need addressing are informed. This will be done in person at the weekly meeting but should also be done using the GitHub Issues tab. All found faults should be uploaded to the development branch issues tab and assigned to the correct member of the development team. If the issue is of Critical Severity then the member of the development team should be contacted immediately.

4.8 Environmental Needs

4.8.1 Test Hardware

- A computer of sufficient specification to run the software.
- A server to host the database.

4.8.2 Test Software

- IntelliJ or Visual Studio Code IDE.

5. Integration Testing Plan (TutorPoint)

5.1 Test Team

Name	Title	Level of Involvement	Responsibilities
Daniel Bishop	Testing and Integration Manager	Approximately 4 Hrs / Week Per iteration basis.	Design and execute test cases to validate system functionality.
Eric Walker	Testing and Integration Manager	Approximately 4 Hrs / Week Per iteration basis.	Design and execute test cases to validate system functionality.
James Gardner	Technical Lead / QA Manager	Approximately 1 Hr / Week Per iteration basis.	Technical assistance as needed during the testing.
Che McKirgan	Technical Lead / Test Engineer	Approximately 1 Hr / Week Per iteration basis.	Technical assistance as needed during the testing.
All Programmers	Software Development Team	Approximately 1 Hr / Week each Per iteration basis.	Sit in on Integration Testing of relevant modules

5.2 Team Reviews

The following reviews will be conducted by the entire test team and a representative from the QA department. Refer to the work schedule for the planned review dates.

- Test and Integration Plan document review.
- Integration testing progress review each iteration. Any critical errors will be highlighted immediately and estimate of completion given.
- Post-integration testing review detailing any issues and highlighting most important to the correct development team.

5.3 Major Tasks and Deliverables

Task	Start date	Stop date	Deliverables
Iteration 1 Development	20/02/20	05/03/20	Iteration 1 Code
Iteration 1 Integration Testing	05/03/20	10/03/20	Iteration 1 Exploratory Testing Document
First Customer Demo	13/03/20	13/03/20	Demo for Customer on Current Software State
Iteration 2 Development	05/03/20	19/03/20	Iteration 2 Code
Iteration 2 Integration Testing	19/03/20	24/03/20	Iteration 2 Exploratory Testing Document
Iteration 3 Development	19/03/20	02/04/20	Iteration 3 Code
Iteration 3 Integration Testing	02/04/20	07/04/20	Iteration 3 Exploratory Testing Document
Iteration 4 Development	02/04/20	16/04/20	Iteration 4 Code
Iteration 4 Integration Testing	16/04/20	21/04/20	Iteration 4 Exploratory Testing Document
Iteration 5 Development	16/04/20	30/04/20	Iteration 5 Code
Iteration 5 Integration Testing (Including contracted modules)	30/04/20	05/05/20	Iteration 5 Exploratory Testing Document
Iteration 6 Development	30/04/20	14/05/20	Iteration 6 Code
Iteration 6 Integration Testing	14/05/20	19/05/20	Iteration 6 Exploratory Testing Document
Iteration 7 Integration Iteration	14/05/20	28/05/20	Iteration 7 Dedicated to module integration and testing.
Iteration 7 Integration Testing	28/05/20	31/05/20	Iteration 7 Exploratory Testing Document

5.3.1 Iteration 1 - 20th Feb - 5th March

Interactive Whiteboard

- User Story #016 Whiteboard window is created and the user can draw on it using a mouse.

XML Presentation

- User Story #008 Open an XML File
- User Story #009 Parsing an XML File
- User Story #010 Creating presentation object.

Media Player/Video Streaming

- User Story #014 Stream video from the server.

Server

- User Story #001 Connection made with Client.
- User Story #001 Multiple clients possible.
- User Story #001 User details stored and retrieved from the database.
- User Story #003 Subjects stored and retrieved from the database.
- User Story #003 Files uploaded to clients.
- User Story #020 Client-Server autodisconnect after set time of inactivity.

GUI / Ease of Use

- User Story #004 Login screen with username and password fields.
- User Story #004 Password is hashed for security.
- User Story #003 Main screen design and navigation
- User Story #005 Successful login and register taken to the main screen.

5.3.2 Iteration 2 - 5th March - 19th March

Interactive Whiteboard

- User Story #017 Mirror of the whiteboard via the server to another client.

XML Presentation

- User Story #015 Changing between slides on the presentation.
- User Story #012 Display images from a url on the presentation canvas.

5.3.3 Iteration 3 - 19th March - 2nd April

Interactive Whiteboard

- User Story #018 Creating shapes on the interactive whiteboard.
- User Story #019 Various pen styles on the interactive whiteboard.

Server

- User Story #006 View all user account details.
- User Story #006 Update all user account details.

GUI / Ease of Use

- User Story #006 View all user account details.
- User Story #006 Update all user account details.

5.3.4 Iteration 4 - 2nd April - 16th April

Interactive Whiteboard

- User Story #021 User can add and remove text on the whiteboard.
- User Story #022 A tutor can control access to the whiteboard preventing other users from drawing.

XML Presentation

- User Story #023 Live presentation over server with multiple clients viewing.

Server

- User Story #023 Live presentation over server with multiple clients viewing.

GUI / Ease of Use

- User Story #027 Lesson page with whiteboard, video-chat and text-chat and presentation on one screen.
- User Story #028 Switch view between slide and whiteboard.

5.3.5 Iteration 5 - 16th April - 30th April

XML Presentation

- User Story #024 Video controlled by Tutor displayed to other Clients.
- User Story #### Graphics handler for XML presentation.
- User Story #### Audio handler for presentation.

Media Player/Video Streaming

- User Story #024 Video controlled by Tutor displayed to other Clients.
- User Story #025 Webcam Video streamed to the server.

Server

- User Story #024 Video controlled by Tutor displayed to other Clients.
- User Story #025 Webcam Video streamed to the server.
- User Story #030 List of subject favourites.

GUI / Ease of Use

- User Story #030 List of subject favourites.

Contracted Modules

- User Story #013 Graphics handler for XML presentation.
- User Story #033 Audio handler for presentation.

Iteration 6 - 30th April - 14th May

XML Presentation

- User Story #011 #Text on XML Presentation
- User Story #013 #Graphics on XML Presentation

Media Player/Video Streaming

- User Story #026 Live webcam streaming

Server

- User Story #026 Live webcam streaming
- User Story #029 Text chat, multi-user

GUI / Ease of Use

- User Story #026 Live webcam streaming
- User Story #029 Text chat, multi-user

5.4 Risks and Contingencies

Module	Risk Chance	Risk Severity	Risk
MySQL Database	Low	Med	Loss of connection with server.
Server	High	High	System crash stopping all Clients from functioning. Overloaded from too many users. Loss of connection with database. Dropping client connections. Lack of HDD Space. DDOS Susceptibility. Weakness to SQL Injection attacks. Loss of user data.
Login/Register	Low	Med	Users gaining access to the accounts of other users. Users unable to access their own accounts. Users able to compromise security and steal data. Man in the middle attacks.
GUI	Low	Med	Displaying incorrect data. Not matching design standards. Unable to pull images down from the server. Slow response to user input or complete freeze during background processes. Scaling poorly to different resolutions.

Lesson Screen	Whiteboard	High	High	Not drawing when the user desires. Not sending data between clients on a shared board. Loss of drawn lines when sent to server.	All these systems are interacting on one window simultaneously. Therefore there is a high risk of issues when integrating these together. Modules making calls to server simultaneously and affecting each other. Simultaneous modules making large demands on cpu load. Lots of threaded modules making calls to single fields simultaneously. Multithreading race conditions.
	Presentation	High	High	Poorly formatted XML causing errors. Presentation too large.	
	Video Chat	High	Med	Large delay between clients. Video quality below acceptable standards. Audio and video out of sync.	
	Video Player	High	Low	Unable to connect to server. File types incorrect. Video freezing. Video not synced for all clients. Audio and video out of sync.	
	Text Chat	Med	Med	Data lost between clients. Users sending dangerous links. Chronology of the messages.	

6. Contracted Module Testing

6.1 Overview

Contracted modules will be tested in a similar fashion to in-house modules, with the addition of a check to ensure that they adhere to the CUBIXEL coding style guide.
Modules contracted for other parties will be developed in a similar fashion to in-house modules, but tested as defined in contract.

6.2 Outsourced Modules

- Company Style Compliance check.
- Unit testing performed by outsource group
- Integration testing on CUBIXEL side.
- Also included in System Testing

6.3 Contracted/Sold Modules

- Unit Tested to ensure quality.
- Developed using TDD to ensure that client requirements are met.
- Assist with integration to the level specified in the contract.

7. System / Acceptance Testing

7.1 System Testing Introduction

System testing will be the penultimate round of testing before releasing the finished product. It will be performed at the end of the development schedule, after all user stories and modules have been finished and integrated.
The objectives of acceptance testing is mainly to prove that the finished system fulfills the functional specification and works as intended, without bugs.

7.2 System Testing Procedure

The responsibility for acceptance testing will be spread amongst the team. Each team member will test their section of the program in a similar fashion to integration testing, by using exploratory testing to verify the function of the program.
System testing should:

- Perform regression testing on all areas of the code developed so far
- Be the penultimate stage in development
- Prove to the client that the product developed is as was specified.

7.3 Acceptance Testing Introduction

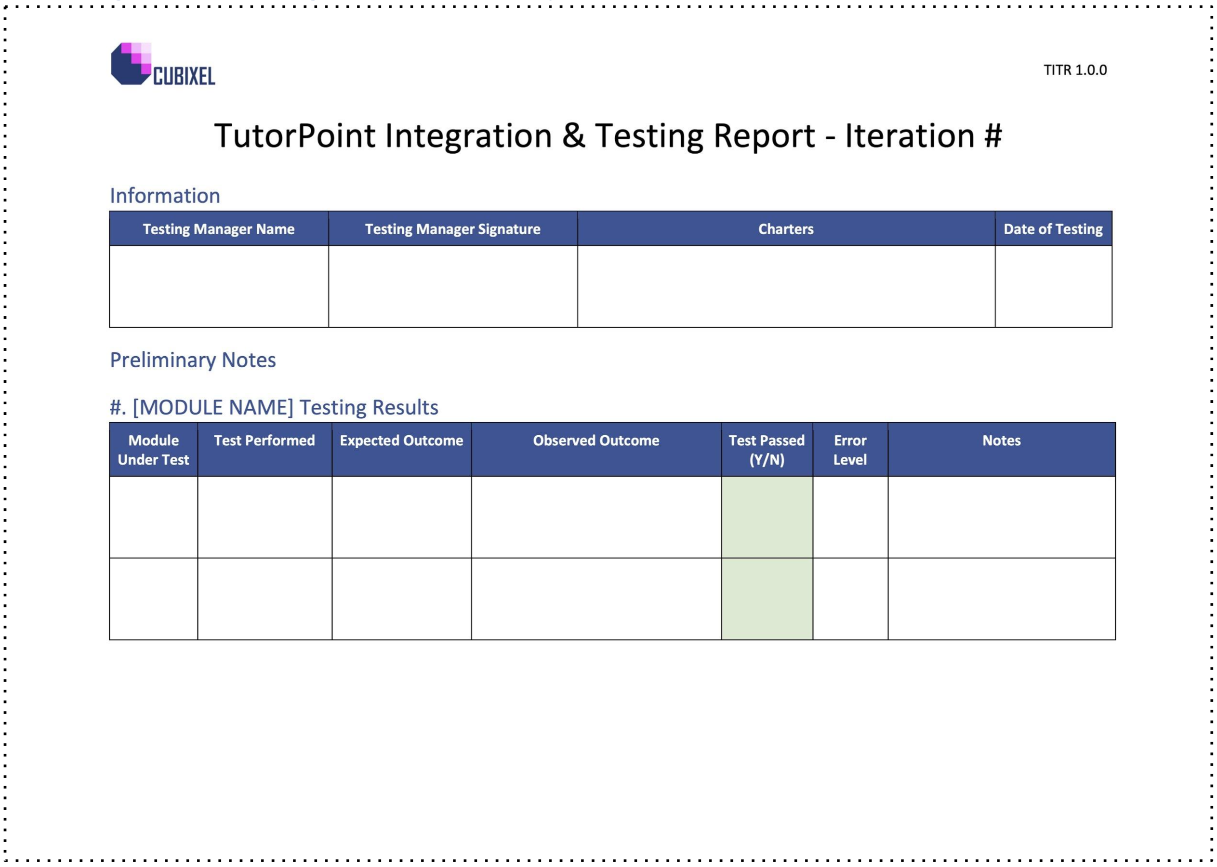
Acceptance testing will be the very last stage of testing before release. It aims to test whether the product produced meets the user's requirements. Unlike system testing, acceptance testing is more user-oriented and non-technical.

7.4 Acceptance Testing Procedure

Testers will assess whether the product meets the user requirements by testing the functionality of the program against the functional spec to ensure that all features have been included.

8. Appendix

8.1 Integration Testing Document Template



TITR 1.0.0

TutorPoint Integration & Testing Report - Iteration #

Information

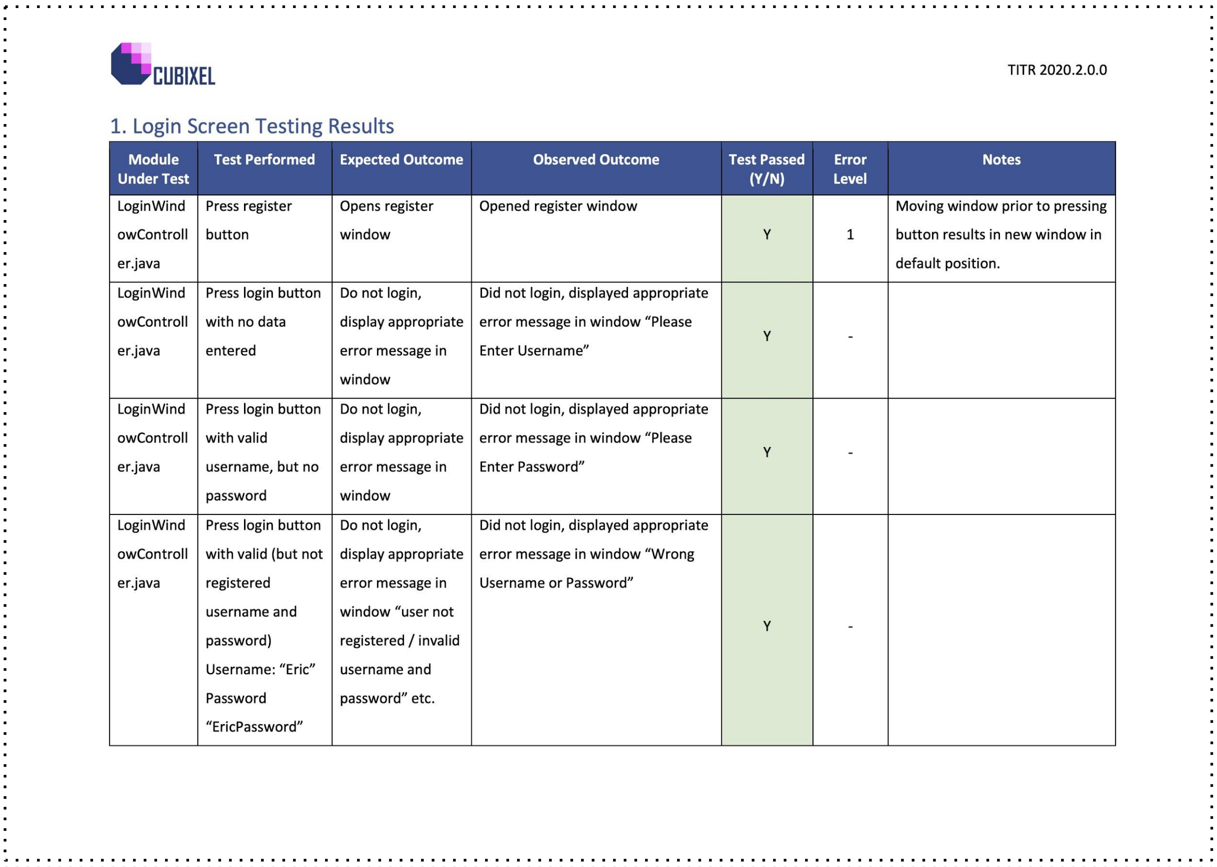
Testing Manager Name	Testing Manager Signature	Charters	Date of Testing

Preliminary Notes

#. [MODULE NAME] Testing Results

Module Under Test	Test Performed	Expected Outcome	Observed Outcome	Test Passed (Y/N)	Error Level	Notes

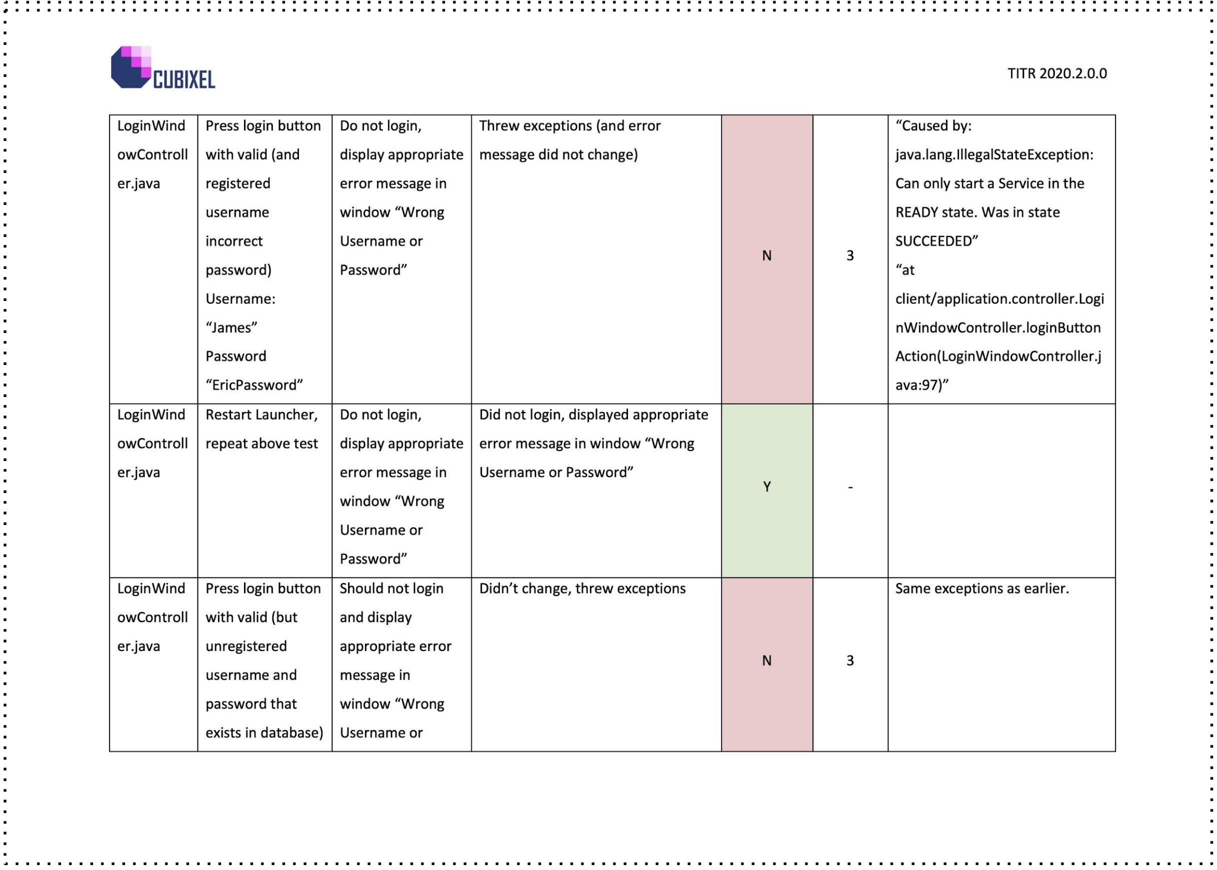
8.2 Integration Testing Document Completed Example



TITR 2020.2.0.0

1. Login Screen Testing Results

Module Under Test	Test Performed	Expected Outcome	Observed Outcome	Test Passed (Y/N)	Error Level	Notes
LoginWindowController.java	Press register button	Opens register window	Opened register window	Y	1	Moving window prior to pressing button results in new window in default position.
LoginWindowController.java	Press login button with no data entered	Do not login, display appropriate error message in window	Did not login, displayed appropriate error message in window “Please Enter Username”	Y	-	
LoginWindowController.java	Press login button with valid username, but no password	Do not login, display appropriate error message in window	Did not login, displayed appropriate error message in window “Please Enter Password”	Y	-	
LoginWindowController.java	Press login button with valid (but not registered username and password) Username: “Eric” Password “EricPassword”	Do not login, display appropriate error message in window “user not registered / invalid username and password” etc.	Did not login, displayed appropriate error message in window “Wrong Username or Password”	Y	-	



TITR 2020.2.0.0

LoginWindowController.java	Press login button with valid (and registered username incorrect password) Username: “James” Password “EricPassword”	Do not login, display appropriate error message in window “Wrong Username or Password”	Threw exceptions (and error message did not change)	N	3	“Caused by: java.lang.IllegalStateException: Can only start a Service in the READY state. Was in state SUCCEEDED” “at client/application.controller.LoginWindowController.loginButtonAction(LoginWindowController.java:97)”
LoginWindowController.java	Restart Launcher, repeat above test	Do not login, display appropriate error message in window “Wrong Username or Password”	Did not login, displayed appropriate error message in window “Wrong Username or Password”	Y	-	
LoginWindowController.java	Press login button with valid (but unregistered username and password that exists in database)	Should not login and display appropriate error message in window “Wrong Username or	Didn’t change, threw exceptions	N	3	Same exceptions as earlier.