

# CSC 648 Software Engineering

## Team 2



### MILESTONE 5 DOCUMENTATION

Version History	Date
M4V1	07DEC2020
M5V1	18DEC2020

### Team

Team lead/Scrum Master:

Jarett Koelmel: [jkoelmel1@mail.sfsu.edu](mailto:jkoelmel1@mail.sfsu.edu)

Front-end Lead: Michael Canson

Front-end Dev: Chiu Wong

Front-end Dev: Paul Borst

Back-end Lead: Brooke Porter

Back-end Engineer/Git Master: Eric Chen

## Table of Contents

1. Product Summary .....	3
2. QA Test Plan.....	4
3. Code Review .....	10
4. Non-Functional Specification Check.....	13

## 1. Product Summary

PorTal is a simple, yet feature-rich, web application designed from the ground-up with the modern physical therapist in mind. This application seeks to fill the gap between physical therapists and remote availability which is so critical given the current business landscape and will provide extended accessibility to a larger addressable market for physical therapists of all types and in various locales previously unreachable through traditional means.

What sets our product apart from the other offerings on the market currently is the ability to easily integrate any workout/exercise video from YouTube easily and seamlessly into the application and provide a crowd-sourced reference list of exercises curated by the physical therapists in the network, shifting the burden of source material from one therapist to many therapists.

Major features/functionality to be delivered by Milestone 5:

- PT profile information stored in backend can be easily queried and formatted which will allow for seamless integration for advertisement of expertise, skill sets, and specializations to prospective patients/users on patient-facing dashboard web application
- PT dashboard provides quick access to basic information about all current and prospective patients
- PT can access an exercise library of instructional videos to create custom workouts for their patients
- PT can add their own videos into the exercise library for personal and network-wide usage
- PTs can save custom workouts for later use across any of their patients
- Adding videos to a custom playlist is as simple as clicking a checkbox
- Exercise library is easily searched by keywords
- Assigning one or more workouts to multiple patients is equally as easy; simply choose all workouts and patients and click 'assign'
- PT has multiple avenues of asynchronous communication to maintain contact with their patients
- Quickest method is via the in-app messaging tool
- Patient progress feedback, either text or video, can be commented on by the PT
- Messaging is encrypted via AES/SHA-512 so database has no plain text messages stored
- Updates to PT profile easily done via Settings page

The focus of this application is primarily on ease-of-use and intuitive design to expedite the physical therapist's workflow and maximize productivity. This allows the physical therapist to see exactly what they need to see quickly and communicate as needed with their patients fluidly throughout their workday.

The production build of our team's application is hosted at: <https://pthealth.club>

## 2. QA Test Plan

### A. Testing Stack and Feature Selection:

- Front-end
  - Jest
  - Enzyme
- Back-end
  - junit5
  - Mockito
- Features Selected:
  - Assign Workout to Patient(s)
  - Patient Progress Log Entries and Videos

### B. Front-end Unit Testing:

- a. The front-end has utilized the common tools Jest and Enzyme to create their unit tests for components in our React App.
- b. The individual tests are held in their respective Component folders and run through the basic functionality of that component and ensure that UI elements are responsive and data population standards are met upon loading of the component and change as needed.
- c. All of the tests currently created can be ran from the command-line using "npm test"
- d. For the purpose of showing the test output, I have included screenshots of each relevant component's unit tests below.
- e. Jest's code coverage was largely ineffectual and only shows line coverage, and not method coverage, but the front-end team has attempted to implement as much coverage as possible without being able to display such a metric, as such,

they have currently developed 38 unit tests across the targeted features and then some

- f. Assumption: Material-UI components used are themselves rigorously tested before publication and are assumed to be functional. Those elements are still tested for appearance in the DOM and functionality resulting from using them is included in the tests.

- g. Screenshot:

```
Watch Usage
> Press f to run only failed tests.
> Press o to only run tests related to changed files.
> Press q to quit watch mode.
> Press p to filter by a filename regex pattern.
> Press t to filter by a test name regex pattern.
> Press Enter to trigger a test run.
PASS src/Components/SavedWorkout/SavedWorkout.test.js (11.831s)
PASS src/Components/Header/Header.test.js (12.002s)
PASS src/Components/PatientVideos/PatientVideo.test.js (11.847s)
PASS src/Components/PatientDashboardInfo/PatientDashboardInfo.test.js (12.325s)
PASS src/Components/SearchReport/SearchReport.test.js
PASS src/Components/AssignWorkout/AssignWorkout.test.js
PASS src/Components/PatientActivitySummary/ActivitySummary.test.js
PASS src/Components/PatientInfo/PatientInfo.test.js
PASS src/Components/CreateWorkout/CreateWorkout.test.js (21.102s)
PASS src/Components/ExerciseLibrary/Library.test.js (21.236s)

Test Suites: 10 passed, 10 total
Tests:       38 passed, 38 total
Snapshots:   0 total
Time:        25.013s
Ran all test suites.

Watch Usage: Press w to show more.
```

C. Back-end Unit Testing:

- a. The back-end has utilized a combination of jUnit5 and Mockito for the unit tests for the database manipulation and API endpoint functionality involving CRUD operations.
- b. The business logic is contained within the Java classes named after their representative tables in the database and are insulated from the end-user via utility classes, denoted by the Util in their class name.
- c. For testing of the business logic: object constructors, accessors and mutators, and basic database insertions, deletions, and updates, jUnit5 methodology is applied.
- d. For testing of the API endpoints that call the methods in the business logic classes, the calls are mocked via Mockito, essentially ensuring that JSON object creation is being handled properly and response codes can be returned as needed.
- e. Because of an incompatibility between jUnit5 and Mockito, the jUnit5 test coverage tool was not operating properly on those tests in the Util classes. Mockito receives the call before it is actually called, in order to mock it, and this does not allow jUnit5 to register coverage for those methods as being tested. However, API endpoint testing should be mocked in order to allow performing such testing without dependency on a live database or network connection at all.
- f. Included below is the screenshot of the code coverage report on the entire backend source code. The main.server class only has one method which is the driver for the entire Spark server and as such was not tested as the API endpoints contained within are tested individually via Mockito and the functionality for the driver is assumed via integration testing conducted later.
- g. Tests were ran in IntelliJ using the “run tests with coverage” tool from the base package of main.server, the entire back-end currently has 75.8% code coverage, not including the Mockito-based tests.

## h. Screenshot:

[ all classes ]

## Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	75.8% (50/ 66)	76.6% (412/ 538)	48.7% (1518/ 3116)

## Coverage Breakdown

Package ^	Class, %	Method, %	Line, %
main.server	0% (0/ 1)	0% (0/ 32)	0% (0/ 118)
main.server.AES	100% (2/ 2)	77.8% (7/ 9)	66% (31/ 47)
main.server.Activity	75% (3/ 4)	83.7% (36/ 43)	56.5% (187/ 331)
main.server.Assignment	75% (3/ 4)	83.7% (36/ 43)	44% (117/ 266)
main.server.Contain	75% (3/ 4)	84% (21/ 25)	47.7% (74/ 155)
main.server.Entry	75% (3/ 4)	83.3% (25/ 30)	53.4% (78/ 146)
main.server.Exercise	75% (3/ 4)	84.2% (32/ 38)	47.5% (106/ 223)
main.server.Injury	75% (3/ 4)	81.8% (18/ 22)	54.8% (68/ 124)
main.server.PT	75% (3/ 4)	64% (16/ 25)	40.6% (106/ 261)
main.server.PTMessage	75% (3/ 4)	75% (21/ 28)	51.5% (101/ 196)
main.server.PTSpec	75% (3/ 4)	69.6% (16/ 23)	44.8% (73/ 163)
main.server.Patient	75% (3/ 4)	79.4% (27/ 34)	50.5% (106/ 210)
main.server.PatientAssignment	100% (2/ 2)	100% (39/ 39)	100% (80/ 80)
main.server.PatientInjury	75% (3/ 4)	84% (21/ 25)	52.8% (75/ 142)
main.server.PatientMessage	75% (3/ 4)	87.1% (27/ 31)	54.5% (85/ 156)
main.server.PatientVideo	75% (3/ 4)	81.8% (27/ 33)	48% (83/ 173)
main.server.Specialization	75% (3/ 4)	66.7% (14/ 21)	39.4% (56/ 142)
main.server.User	100% (1/ 1)	92.9% (13/ 14)	96.3% (26/ 27)
main.server.Workout	75% (3/ 4)	69.6% (16/ 23)	42.3% (66/ 156)

## D. Integration Test Strategy:

For this section, team members are testing specific functionalities related to the chosen features across two major browsers, Firefox and Chrome. Basic functionality as well as null/edge cases are considered to ensure the web app is not easily broken. This relies on the ability for the changes in the backend to be reflected accurately and quickly in the front end and the requests sent from the front end are successfully connecting to the backend and the UI is updated properly. Key points of testing are listed in the table below:

Testing Table

Test	Desired Action	Pass/Fail (Firefox)	Pass/Fail (Chrome)	Corrective Action (if applicable)
1. Clicking on assign without selecting any patient or workout	Do nothing	Pass	Pass	N/A
2. Assigning one workout to many patients	All patients are now assigned proper workout, alert window showing success	Pass	Pass	N/A

3. Simulate removing assigned workout from patient	Current workout field in profile should be blank	Pass	Pass	N/A
4. Simulate new patient progress log entry	New entry is shown in correct order upon page refresh	Pass	Pass	N/A
5. Simulate removing patient progress log entry	Entry is no longer shown upon page refresh	Pass	Pass	N/A
6. Simulate adding new patient video	New video is shown in correct order upon page refresh	Pass	Pass	N/A
7. Simulate removing patient video	Video is no longer shown upon page refresh	Pass	Pass	N/A
8. Clicking each patient name	Entry and video data should be different for each patient and reproducible every time	Pass	Pass	N/A
9. Clicking current assigned workout	Modal showing exercises in that workout with links to videos	Pass	Pass	N/A
10. Clicking patient video	Video player shows with playback options for viewing in-page	Pass	Pass	N/A

### User Stories and Actions Taken

(User Story # from M1 docs, Test # from table above)

Test Number/User Story	Steps Taken by User
1. / 1.	1. User logs in 2. User is automatically routed to the dashboard 3. User clicks the assign button without selecting any workout or patient to assign to 4. There should be no action taken by the application
2. / 1.	1. User logs in 2. User is routed to the dashboard



	<ol style="list-style-type: none"> <li>3. User checks one workout to assign</li> <li>4. User selects one to many patients to assign workout too</li> <li>5. User clicks assign button</li> <li>6. Success message should appear on screen</li> </ol>
3. / 2.	<ol style="list-style-type: none"> <li>1. User logs in</li> <li>2. User checks out which workout specific patient is assigned</li> <li>3. User finds a workout assigned to particular patient</li> <li>4. User clicks trash can icon to delete that workout</li> <li>5. Confirmation popover menu should appear prior to deletion</li> <li>6. User clicks delete to confirm</li> <li>7. Confirmation of deletion message should appear</li> <li>8. Patient profile for current workout should be blank</li> </ol>
4. / 3.	<ol style="list-style-type: none"> <li>1. Add new patient entry to database manually</li> <li>2. User logs in</li> <li>3. User goes to patient with new progress log entry</li> <li>4. New entry should appear at the top of the list with timestamp it was added</li> <li>5. If needed, comment button allows for ease of adding response to patient</li> </ol>
5. / 3.	<ol style="list-style-type: none"> <li>1. Remove a patient entry from database manually</li> <li>2. User logs in</li> <li>3. User goes to patient with removed entry</li> <li>4. Patient entry log should not show the removed entry</li> <li>5. UI should not have empty spots where patient entry was removed</li> </ol>
6. / 3.	<ol style="list-style-type: none"> <li>1. Add new patient video to database manually</li> <li>2. User logs in</li> <li>3. User goes to patient with new video</li> <li>4. New video should appear at the top of the list with timestamp it was added</li> </ol> <p>If needed, comment button allows for ease of adding response to patient regarding workout video</p>
7. / 3.	<ol style="list-style-type: none"> <li>1. Remove a patient video from database manually</li> <li>2. User logs in</li> <li>3. User goes to patient that removed their video</li> <li>4. Patient video should not show the removed video</li> <li>5. UI should not have empty spots where patient video was removed</li> </ol>
8. / 1.	<ol style="list-style-type: none"> <li>1. User logs in</li> <li>2. User is presented with a list of all current patients</li> <li>3. User clicks through each patient to see all pertinent data</li> </ol>

	<ol style="list-style-type: none"> <li>4. Data is accurate to what is contained in the database</li> <li>5. Continually switching between patients does not ruin data's integrity or consistency, showing ACID compliance</li> </ol>
9. / 2.	<ol style="list-style-type: none"> <li>1. User logs in</li> <li>2. User clicks the patient they wish to review workout for</li> <li>3. Current workout, if any, is shown</li> <li>4. User clicks on workout to see what exercises it contains</li> <li>5. A modal showing title, thumbnail and description of videos contained in that workout</li> <li>6. Clicking on an exercise video opens a new tab to the YouTube link for that exercise</li> </ol>
10. / 2.	<ol style="list-style-type: none"> <li>1. User logs in</li> <li>2. User clicks a patient to see new patient uploaded videos</li> <li>3. User clicks on a video to review</li> <li>4. User is shown a modal with an embedded YouTube player in app</li> <li>5. User can view the video in that modal with working controls for playback and speed</li> </ol>

### 3. Code Review

Our team has chosen the Google Java coding format for all of our backend source code, and utilizing ESLint with Prettier configured via YAML files to format our front-end source code in accordance with AirBnB JavaScript coding guidelines. We have automated our GitHub workflow as code reaches the Development branch, so that as features are submitted from the Feature branch into the Development branch, we can ensure that all code is automatically formatted according to our desired standards. If the automated checks cannot be rectified via GitHub Actions, the code is not able to be merged and the PR will be denied. The alerts on GitHub can show the submitter of the PR what is causing the issues so that they can be fixed and re-submitted for approval.

Automating our workflow as a team in this way, allows our different developers to work in a manner that is efficient to them, so long as their syntax is "close-enough" to be modified easily by GitHub upon committal. This frees the team up by worrying less on the minutiae of their code and focus more on creating features and completing assigned modules for milestones during each code sprint.

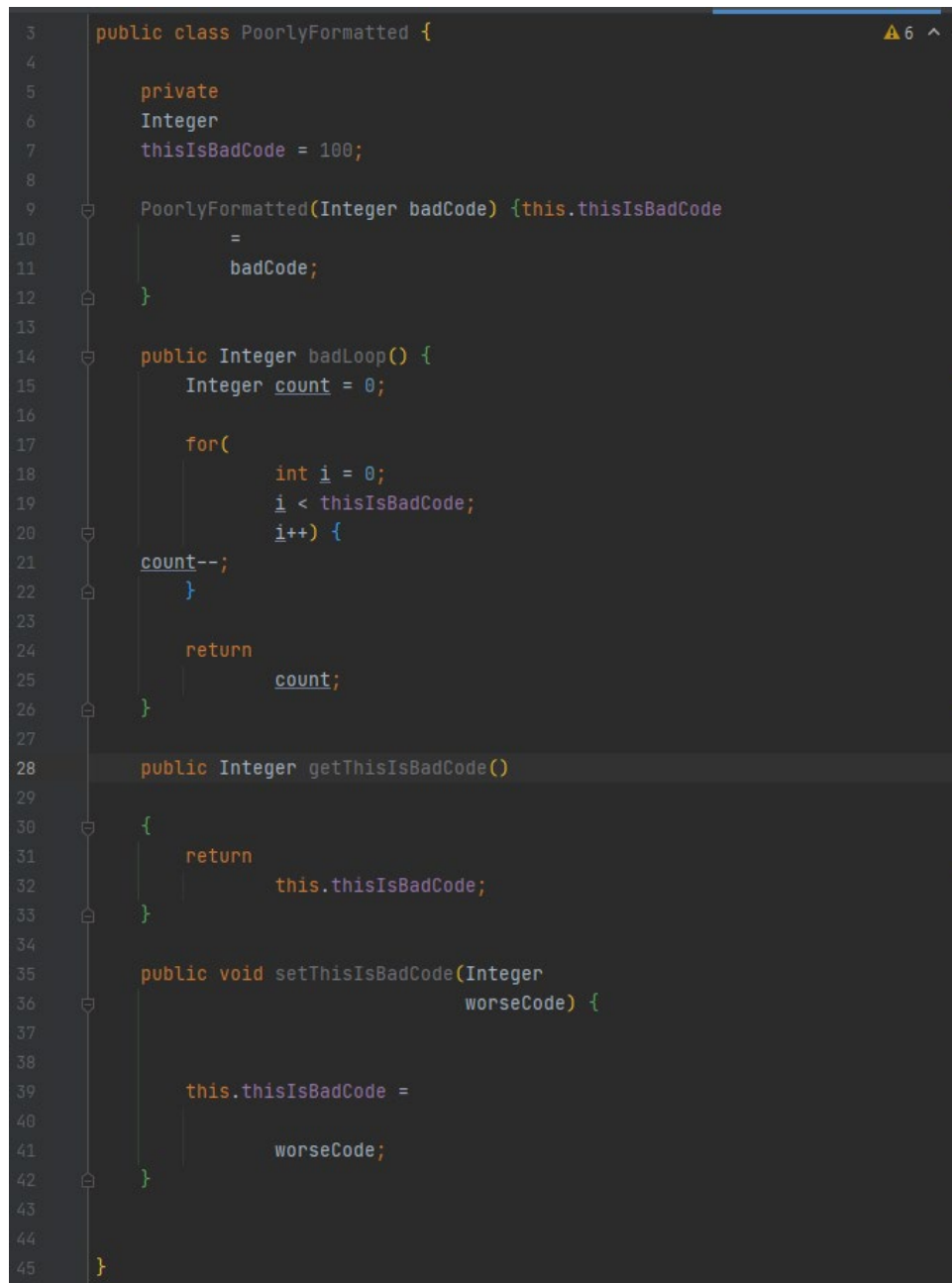
Below are screenshots of purposefully poorly coded Java pushed into a newly created CodeFormat package, the first one is the code prior to automated formatting to the Google Java coding guidelines and the second one is after formatting has been applied.

The link to this file is: <https://github.com/echen22/CSC648-01-Fa20-Team02/blob/Development/Back-End/SparkServer/src/main/server/CodeFormat/PoorlyFormatted.java>

The GitHub Actions used can be found at: <https://github.com/echen22/CSC648-01-Fa20-Team02/actions>

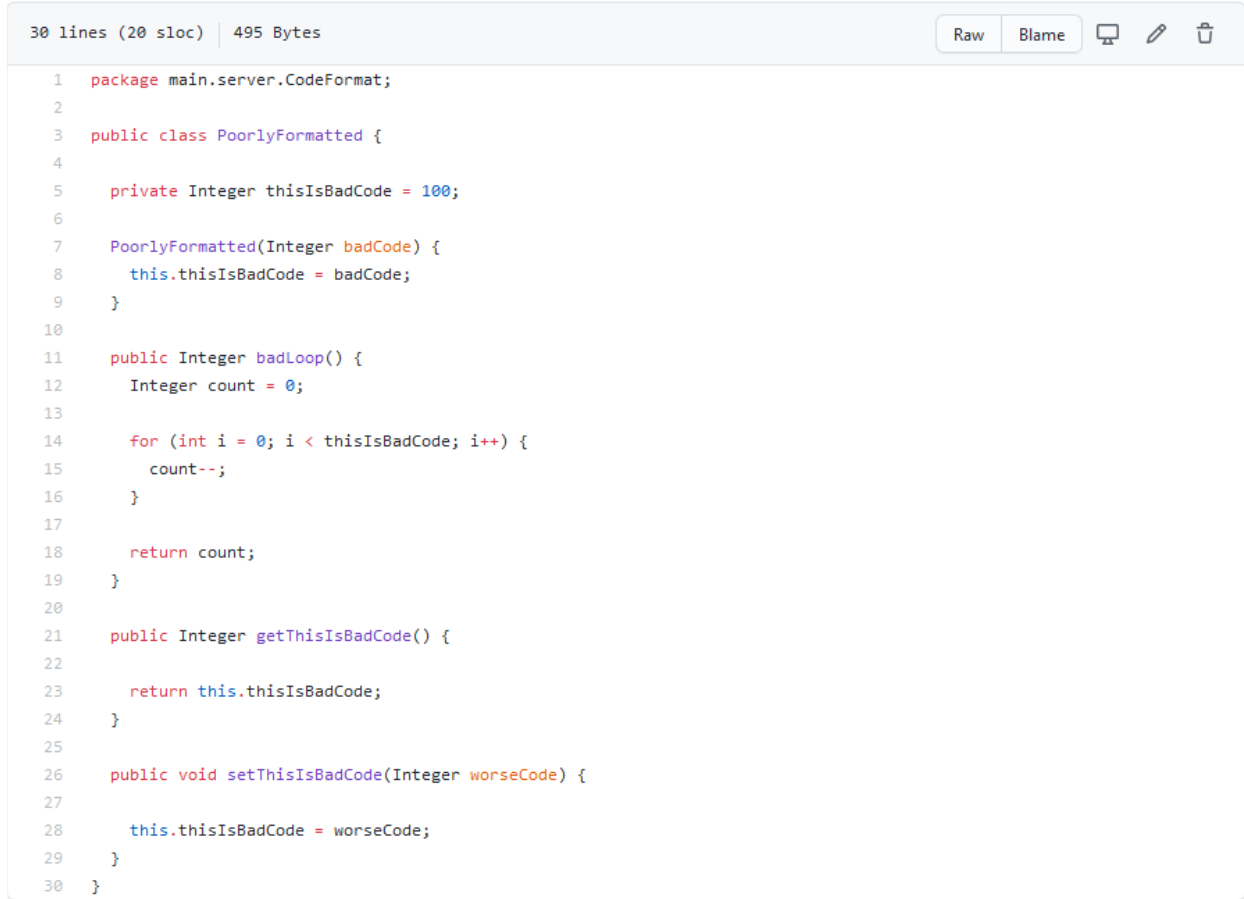
Screenshots:

#### Bad Code Format



```
3 public class PoorlyFormatted {
4
5     private
6     Integer
7     thisIsBadCode = 100;
8
9     PoorlyFormatted(Integer badCode) {this.thisIsBadCode
10         =
11         badCode;
12     }
13
14     public Integer badLoop() {
15         Integer count = 0;
16
17         for(
18             int i = 0;
19             i < thisIsBadCode;
20             i++) {
21             count--;
22         }
23
24         return
25             count;
26     }
27
28     public Integer getThisIsBadCode()
29
30     {
31         return
32             this.thisIsBadCode;
33     }
34
35     public void setThisIsBadCode(Integer
36                                     worseCode) {
37
38
39         this.thisIsBadCode =
40
41             worseCode;
42     }
43
44
45 }
```

## Auto-Formatted Code when pushed to Development

A screenshot of a code editor interface. At the top, a status bar shows '30 lines (20 sloc) | 495 Bytes'. To the right of the status bar are buttons for 'Raw', 'Blame', and icons for a monitor, a pencil, and a trash can. The main area displays 30 lines of Java code, which has been auto-formatted. The code defines a package, a class, a private field, and three methods: a constructor, a loop method, and getter/setter methods. The formatting includes consistent indentation, line wrapping, and color-coding for keywords and identifiers.

```
1 package main.server.CodeFormat;
2
3 public class PoorlyFormatted {
4
5     private Integer thisIsBadCode = 100;
6
7     PoorlyFormatted(Integer badCode) {
8         this.thisIsBadCode = badCode;
9     }
10
11     public Integer badLoop() {
12         Integer count = 0;
13
14         for (int i = 0; i < thisIsBadCode; i++) {
15             count--;
16         }
17
18         return count;
19     }
20
21     public Integer getThisIsBadCode() {
22
23         return this.thisIsBadCode;
24     }
25
26     public void setThisIsBadCode(Integer worseCode) {
27
28         this.thisIsBadCode = worseCode;
29     }
30 }
```

This shows that even very badly written code is rectified by the methods we have implemented via GitHub actions using ESLint and the Google Java formatting to minimize the need for specific team members to manually review code to keep it within the standards we have established.

## 4. Non-Functional Specification Check

### Operations:

Requirement	Status
Access Security: Safeguards from intrusion, password protection, and ensuring users only see pertinent data	ON TRACK
Accessibility: Provides multiple systems to facilitate asynchronous physical therapy, currently only expecting support for English.	DONE
Availability: User access provided as needed and on demand.	DONE
Confidentiality: HIPPA compliance and protection of sensitive user data	ON TRACK
Efficiency: Extent software handles capacity and response time during abnormal usage	ON TRACK
Integrity: Data maintenance and accuracy	ON TRACK
Reliability: System performs functions without failure	DONE
Survivability: System functionality and recovery	ON TRACK
Usability: Ease of use; intuitive and effective	ON TRACK

### Revisions:

Requirement	Status
Flexibility: Software is adaptable to different environments and configurations	ON TRACK
Maintainability: Ease of fault discovery and rectifying issues	ON TRACK
Modifiability: Cost effective development, maintenance and production	DONE
Scalability: Ability off the system to scale both vertically and horizontally	DONE
Verifiability: Extent needed to prove system functions as desired	ON TRACK

### Transitions:

Requirement	Status
Installation: System requirements and availability	DONE
Interoperability: Ability to interface with many diverse systems	ON TRACK
Portability: Software works across environments/ OS	ON TRACK
Reusability: Ease of conversion between systems	DONE