Loop Solutions Inc.

For Self Start System

Version <1.0>

[Note: Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

[Note: The Software Requirements Specification (SRS) captures the complete software requirements for the system, or a portion of the system.  The Modern SRS is a typical SRS outline for a project **using use-case modeling**. This artifact consists of a package containing use cases of the use-case model and applicable Supplementary Specifications and other supporting information. For a template of an SRS **not** using use-case modeling, which captures all requirements in a single document, with applicable sections inserted from the Supplementary Specifications (which would no longer be needed), see[\\program](file:///\\program) \program files\Rational\ RequisitePro\Outlines\ rup\_srs.dot.]

Many different arrangements of an SRS are possible. Refer to [IEEE93] for further elaboration of these explanations, as well as other options for SRS organization.]

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <09/11/2017> | <1.0> | Created UML Use Case Diagram with group | Everyone |
| <14/11/2017> | <1.1> | Updated Use Cases and Use Case Diagram | Everyone |
| <15/11/2017> | <1.2> | Finished the Actor, Use Case descriptions | Samuel Mallabone, Andrew Black |
| <15/11/2017> | <1.3> | Completed the Introduction and Survey Description for Task Two | Samuel Mallabone, Andrew Black |
| <15/11/2017> | <1.4> | Completed Interface and Applicable Standards | Robert Northmore |
| <15/11/2017> | <1.5> | Completed Supportability, Design Constraints, User Online Documentation and Help System Requirements | Craig Cook |
| <15/11/2017> | <1.6> | Completed Use-Case Model Hierarchy | Samuel Mallabone, Andrew Black |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Overall Description 4

2.1 Use-Case Model Survey 4

2.1.1 Introduction 4

2.1.2 Survey Description 4

2.1.3 Use-Case Model Hierarchy 5

2.1.4 Diagrams of the Use-Case Model 5

2.2 Assumptions and Dependencies 5

3. Requirements 5

3.1 Use-Case Specifications 5

3.2 Functionality 5

3.2.1 <Functional Requirement One> 5

3.3 Usability 6

3.3.1 <Usability Requirement One> 6

3.4 Reliability 6

3.5 Performance 6

3.5.1 <Performance Requirement One> 6

3.6 Supportability 6

3.6.1 <Supportability Requirement One> 7

3.7 Design Constraints 7

3.7.1 <Design Constraint One> 7

3.8 Online User Documentation and Help System Requirements 7

3.9 Purchased Components 7

3.10 Interfaces 7

3.10.1 User Interfaces 7

3.10.2 Hardware Interfaces 7

3.10.3 Software Interfaces 7

3.10.4 Communications Interfaces 7

3.11 Licensing Requirements 7

3.12 Legal, Copyright and Other Notices 7

3.13 Applicable Standards 7

# Introduction

[The introduction of the Modern SRS should provide an overview of the entire Modern SRS. It should include the purpose, scope, definitions, acronyms, abbreviations, references and overview of the Modern SRS.]

## Purpose

[Specify the purpose of this Modern SRS. The Modern SRS should fully describe the external behavior of the application or subsystem identified. It also describes nonfunctional requirements, design constraints and other factors necessary to provide a complete and comprehensive description of the requirements for the software.]

## Scope

[A brief description of the software application that the Modern SRS applies to; the feature or other subsystem grouping; what Use Case model(s) it is associated with, and anything else that is affected or influenced by this document.]

## Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| SRS | Software Requirements Specification. The document that describes the software system that is to be developed, including the functional requirements, non-functional requirements and the use cases and their interaction with the actors. |
| GUI | Graphical User Interface. |
| Ember | An open-source JavaScript web framework, based on the model-view-viewmodel pattern. |
| Socket (also known as Network Socket) | An internal endpoint for sending or receiving data at a single node in a computer network. |
|  |  |
|  |  |

[This subsection should provide the definitions of all terms, acronyms, and abbreviations required to interpret properly the Modern SRS.  This information may be provided by reference to the project Glossary.]

## References

[This subsection should provide a complete list of all documents referenced elsewhere in the Modern SRS. Each document should be identified by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

## Overview

[This subsection should describe what the rest of the Modern SRS contains and explain how the Modern SRS is organized.]

# Overall Description

[This section of the Modern SRS should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in section 3, and makes them easier to understand. Include such items as product perspective, product functions, user characteristics, constraints, assumptions and dependencies, and requirements subsets.]

## Use-Case Model Survey

[This section contains an overview of the use-case model or the subset of the use-case model that is applicable for this subsystem or feature.  This includes a list of names and brief descriptions of all use cases and actors, along with applicable diagrams and relationships. This section describes the use-case model comprehensively, in terms of how the model is structured into packages and what use cases and actors there are in the model. If you are using packages, the document shows the model structure hierarchically.]

### Introduction

[Introduction to the use-case model.]

### Survey Description

[Survey description of the use-case model.]

### Use-Case Model Hierarchy

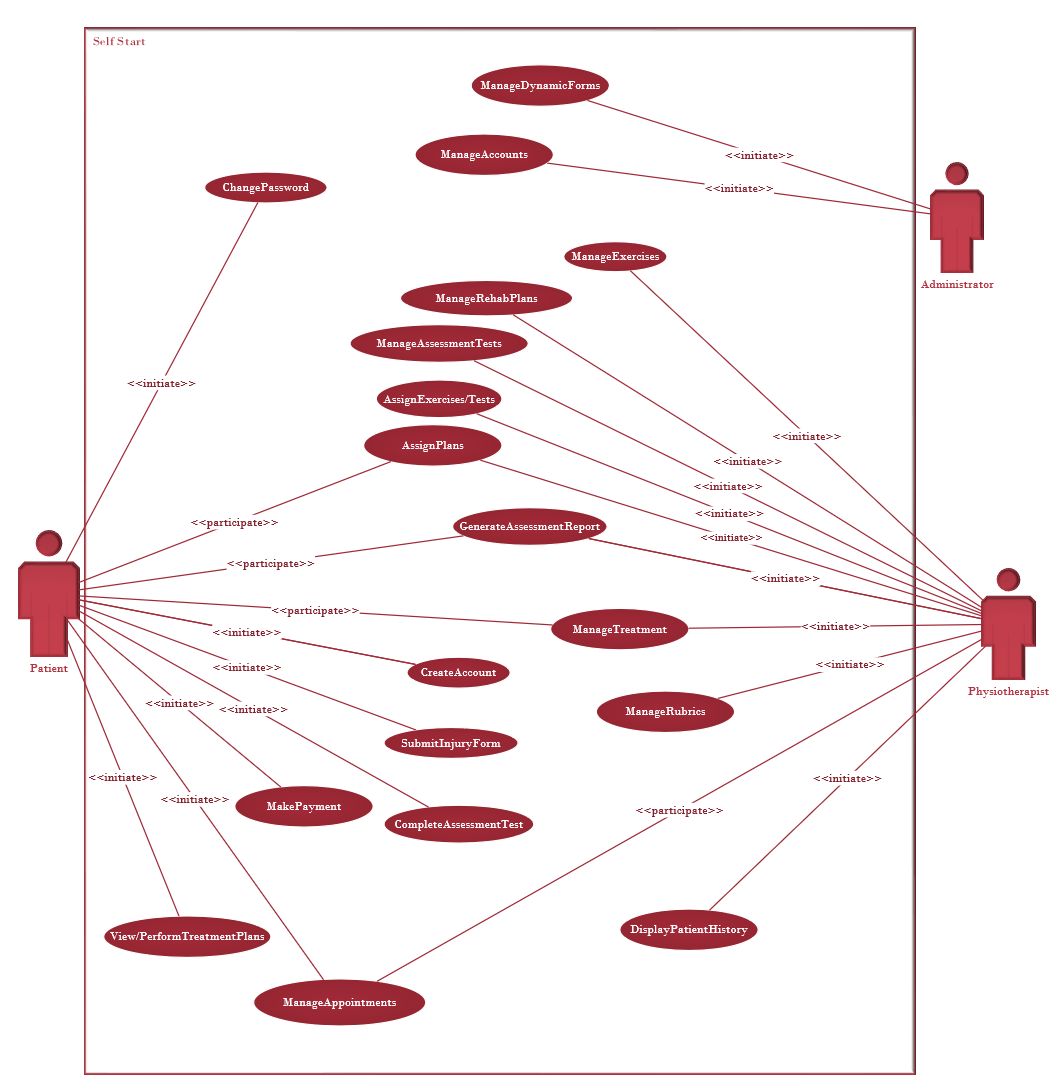
|  |  |
| --- | --- |
| **Actors** | **Descriptions** |
| Patient | The Patient utilizes the system in-order to receive their desired medical assistance via a Physiotherapist. |
| Administrator | The Administrator maintain Patient and Physiotherapist accounts within the system as well as make relevant changes to the dynamic forms within Self Start |
| Physiotherapist | The Physiotherapist utilizes the system in order to communicate, treat and store about their patients. |

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| ManageDynamicForms | Self Start allows the Administrator to make any required changes to the dynamic forms that exist within the system. |
| ManageAccounts | Self Start allows the Administrator to add, edit or delete Patient or Physiotherapist accounts from the system. The Admin can also reset passwords upon request. |
| ManageExercise | Self Start allows Physiotherapists to add, edit or delete exercise. |
| ManageRehabPlans | Self Start allows Physiotherapists create, edit or delete standard and custom rehab plans. |
| ManageAssessmentTests | Self Start allows Physiotherapists to add, edit or delete assessment tests. |
| AssignExercises/Tests | Self Start allows a Physiotherapist to add exercises and tests to rehab plans and assessment tests. |
| AssignPlans | Self Start allows a Physiotherapist to assign rehab plans and assessment tests to Patient’s that are assigned to them |
| GenerateAssessmentReport | Self Start allows a Physiotherapist to generate assessment reports about one of their assigned Patient’s. |
| ManageTreatment | From the results of a Patient’s assessment tests, the Self Start system allows the Patient’s Physiotherapist to make decisions about the future of their treatment. |
| ManageRubrics | Self Start allows a Physiotherapist to add, update or delete rubrics. |
| DisplayPatientHistory | Self Start allows a Physiotherapist to display all of the information about one of the Patient’s assigned to them. |
| ChangePassword | Self Start allows a Patient to change their password that is stored within the system |
| CreateAccount | Self Start allows an individual to create an account in the Self Start system. |
| SubmitInjuryForm | Self Start allows a non-registered user to submit an injury form. |
| CompleteAssessmentTest | Self Start allows a Patient to complete an assessment test that is assigned to them by their Physiotherapist. |
| MakePayment | Self Start allows a Patient to pay any outstanding fees that they owe. |
| ManageAppointments | Self Start allows a Patient to book or cancel appointments with their Physiotherapist. |
| View/PerformTreatmentPlans | Self Start allows a Patient to view or perform any of the treatment plans that are assigned to them. |

[This section presents the use-case packages hierarchically, explains the dependencies among them, and shows the content of each package recursively. If the model has several levels of packages, those at the top-level are presented first. The packages within these are presented next, and so on, all the way down to the packages at the bottom of the hierarchy. For each package include:

* The Name.
* A Brief Description explaining the package's function and role in the system. The description must be understandable to any developer who wants to use the package.
* A list of the use cases owned by the package, including the name and brief description of each use case.
* A list of actors owned by the package, including the name and brief description of each actor.
* A list of relationships owned by the package, including the name and brief description of each relationship.
* A list of the packages directly owned by the package, with each package presented in the same hierarchical manner as above]

### Diagrams of the Use-Case Model



## Assumptions and Dependencies

[This section describes any key technical feasibility, subsystem or component availability, or other project related assumptions on which the viability of the software described by this Modern SRS may be based.]

# Requirements

[This section of the Modern SRS should contain all the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements.   When using use-case modeling, the majority of these requirements are captured in the use cases.]

## Use-Case Specifications

|  |  |
| --- | --- |
| **Use case name** | ManageDynamicForms |
| **Participating actors** | Initiated by Administrator |
| **Entry condition** | The Administrator has successfully logged in to Self Start. |
| **Flow of events** | 1. The Administrator selects the manage dynamic form option.  2. Self Start responds by displaying a screen showing all dynamic forms within the system along the options add, change or delete them.  3. The Administrator selects one of the forms to manage (change or delete). Alternatively they can choose to add a new dynamic form to Self Start.  4. Self Start saves the changes made to the dynamic forms and displays the updated forms. |
| **Exit condition** | Self Start confirms the changes made to the system and all future requests includes these changes. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | ManageAccounts |
| **Participating actors** | Initiated by Administrator. |
| **Entry condition** | The Administrator has successfully logged in to Self Start. |
| **Flow of events** | 1. The Administrator selects the manage accounts option in Self Start  2. Self Start responds by showing the manage accounts window which displays all current accounts registered to the system, both Patient and Physiotherapist.  3. For each Physiotherapist or Patient, the system displays their name (family name, given name), gender, date of birth, address (city, region, postal code), telephone number, health card number, marital status, occupation and others.  4. The Administrator chooses to update an account (update, delete) or to create a new Patient or Physiotherapist account. If creating a new account, the Administrator must fill in all the required information.  5. Depending the Administrator actions, the registered accounts within the system will update accordingly. |
| **Exit condition** | Self Start saves all changes made to the registered accounts. |
| **Quality requirement** | At any given time, the Administrator can choose to filter the list of accounts to only show Patients or Physiotherapists.  On the occurrence of a Patient password request, the Administration is given the ability to fulfill or deny the request. |

|  |  |
| --- | --- |
| **Use case name** | ManageExercises |
| **Participating actors** | Initiated by Physiotherapist |
| **Entry condition** | The Physiotherapist has successfully logged in to Self Start. |
| **Flow of events** | 1. The Physiotherapist has clicked on the ManageExercise button.  2. Self Start directs the Physiotherapist page where they can create, update, or delete an exercise.  3. The Physiotherapist makes their desired changes and submits these to the system.  4. The Self Start system saves the changes that the Physiotherapist has made. |
| **Exit condition** | All changes to Self Start have been save to the database. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | ManageRehabPlans |
| **Participating actors** | Initiated by Physiotherapist. |
| **Entry condition** | The Physiotherapist has successfully logged in to the Self Start. |
| **Flow of events** | 1. The Physiotherapist chooses the ManageRehabPlans button.  2. Within the ManageRehabPlans tab, the Physiotherapist can create (standard or custom), manage or update existing plans.  3. For each rehab plan, the system displays it’s unique identification code, name, description, author name, overall rehabilitation goal, list and order of exercises, a time frame to complete the plan and the assessment tests.  4. The Physiotherapist manages these rehab plans however they choose too. If they choose to create a new plan they must fill out all the required forms.  5. The system reflects any Physiotherapist changes. |
| **Exit condition** | Self Start reflects the changes to the plans and updates the database. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | ManageAssessmentTests |
| **Participating actors** | Initiated by Physiotherapist |
| **Entry condition** | The Physiotherapist has successfully logged in to Self Start |
| **Flow of events** | 1. The Physiotherapist chooses the ManageAssessmentTests button  2. Upon the button being clicked, the Self Start system displays the all the registered assessment tests along with their unique identification code, name, description, author name, assessment tools and, assessment rubric.  3. Self Start gives the Physiotherapist the option to create a new assessment test, update an existing one or delete an existing one.  4. The Physiotherapist manages the assessment tests. If they choose to create a new one they must fill in the appropriate information.  5. Self Start reflects the updated information altered by the Physiotherapist. |
| **Exit condition** | Self Start saves all changes made to the assessment tests to the database. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | AssignExercises/Tests |
| **Participating actors** | Initiated by Physiotherapist |
| **Entry condition** | The Physiotherapist has successfully logged in to Self Start |
| **Flow of events** | 1. The Physiotherapist clicks on the assign exercise and tests button.  2. The system responds by displaying the assign exercise and tests screen for the Physiotherapist.  3. The Physiotherapist is able to assign an exercise or an assessment test to a desired plan.  4. Self Start updates the rehabilitation plan according to the Physiotherapist ‘s input. |
| **Exit condition** | The system saves all changes to the rehabilitation plans, in the database, performed by the Physiotherapist. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | AssignPlans |
| **Participating actors** | Initiated by Physiotherapist. Participated in by Patient. |
| **Entry condition** | The Physiotherapist has successfully logged in to Self Start. |
| **Flow of events** | 1. The Physiotherapist selects the AssignPlans button.  2. Self Start will display all user’s that are currently being treated by the Physiotherapist. It will only display the user’s name.  3. The Physiotherapist selects one of the user’s from the list presented to them.  4. Self Start than displays only the user’s name, gender, age and their injury report.  5. The Physiotherapist designates a rehabilitation plan, either standard or custom, to the selected user.  6. Self Start updates the user’s treatment plan with the selected rehabilitation plan designated by the physiotherapist. Self Start then sends the user a notification that new plan has been has been added to their treatment plan. |
| **Exit condition** | Self Start saves the updated user’s treatment plan to the database and contacts the user based on the changes. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | GenerateAssessmentReport |
| **Participating actors** | Initiated by Physiotherapist. Participated in by Patient. |
| **Entry condition** | The Physiotherapist has successfully logged in and a user has completed their assessment test. |
| **Flow of events** | 1. The Physiotherapist has selected the generate assessment report button.  2. Self Start directs them to a window that shows all their designated users.  3. The Physiotherapist selects the desired user given they have completed an assessment test.  4. Self Start displays the current assessment plan and examination result for the user as well as their completed associated rubrics to the assessment tests.  5. The system displays the following options to the Physiotherapist: print/send an examination summary with the user’s corresponding treatment plan, based off the assessment tests and rubrics the system generates data analysis to show the physiotherapist, as well as generate a summary report (which includes patient personal information, the diagnose case, the treatments, the appointment calendar, invoice payments and the final outcome) to be displayed or printed.  6. The Physiotherapist chooses one of the options displayed by the Self Start system.  7. Based off the Physiotherapists decision, the system completes the desired action and notifies the Patient if necessary. |
| **Exit condition** | The system updates the required information pertaining to the Patient’s treatment. If necessary, the Patient is notified by the system regarding the changes to their treatment plan. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | ManageTreatment |
| **Participating actors** | Initated by Physiotherapist. Participated in by Patient |
| **Entry condition** | The Physiotherapist has successfully logged in to the system. |
| **Flow of events** | 1. The Physiotherapist selects the manage treatment button.  2. Self Start displays a list of all the Patient’s assigned to the Physiotherapist.  3. From this presented list, one of the Patient’s is selected by the Physiotherapist.  4. Based off the Patient’s past assessment test results, the Self System will recommend a course of action. The options to continue treatment or close treatment is presented to the Physiotherapist along with the treatment rubrics.  5. The Physiotherapist acknowledges system’s recommendations along with the assessment rubrics and selects a treatment option.  6. The system notifies the Patient based on the action taken. |
| **Exit condition** | The system sends the Patient a note that they’re treatment will continue OR the system sends the Patient a note that their treatment is concluded. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | ManageRubrics |
| **Participating actors** | Initiated by Physiotherapist. |
| **Entry condition** | The Physiotherapist has successfully logged in. |
| **Flow of events** | 1. The Physiotherapist selects the manage rubrics button.  2. The Self Start system displays the screen with all of the registered rubrics within the database. The options to create, update or delete rubrics are shown to the Physiotherapist.  3. The Physiotherapist selects one of the options that is presented to them.  4. The system reflects all changes made to the rubrics. |
| **Exit condition** | All changes to the rubrics are saved to the database by the system. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | DisplayPatientHistory |
| **Participating actors** | Initiated by Physiotherapist. |
| **Entry condition** | The Physiotherapist has successfully logged in to the system and have selected to see all users. |
| **Flow of events** | 1. The Physiotherapist chooses a Patient from their list of assigned Patients.  2. The system displays all assessment tests through the Patient’s progression through their treatment. |
| **Exit condition** | The Patient’s history has been successfully retrieved and displayed. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | ChangePassword |
| **Participating actors** | Initiated by Patient |
| **Entry condition** | The Patient has successfully logged in. |
| **Flow of events** | 1. The Patient selects to the change password button from their account settings.  2. The system displays a form to the Patient prompting them to re-enter their old password and then enter their new password twice.  3. The Patient enters all requested information.  4. The system verifies all the information such as the old password is correct and the new password is identical in the two textboxes. It then sends a confirmation message whether the request was successful or unsuccessful. |
| **Exit condition** | The Patients password has been successfully changed and is updated within the database OR the Patients request to change their password was denied as their old password was entered incorrectly or the two new passwords entered don’t match |
| **Quality requirement** | If the Patient couldn’t log in because they were unable to enter the correct password, they can request to have their password reset by the Administrator. |

|  |  |
| --- | --- |
| **Use case name** | CreateAccount |
| **Participating actors** | Initiated by Patient |
| **Entry condition** | The Patient has accessed the Self Start website. |
| **Flow of events** | 1. The Patient selects the register account button on the home page.  2. Self Start responds by displaying the sign-up form, prompting the user to enter a username(email) and password. Additionally, they must enter their full name, gender, date-of-birth, address, telephone number, health card number, marital status, occupation and any additional information.  3. The Patient fills in all required information and submits the form.  4. The Self Start system checks if the information is valid (ie email not in user) use and replies to the Patient. |
| **Exit condition** | Self Start alerts the Patient that their account was successfully and saves the account to the database OR Self Start alerts the Patient some information was entered invalidly and prompts them to correct it. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | SubmitInjuryForm |
| **Participating actors** | Initiated by Patient. |
| **Entry condition** | The successfully created/logged in to their account in Self Start. |
| **Flow of events** | 1. The Patient navigates to the submit injury form option.  2. The Self Start system displays the form to the Patient and prompts them to enter general information about their injury.  3. The Patient fills out all required information.  4. The system acknowledges the information entered, determines if the all required information has been entered correctly, and responds accordingly. |
| **Exit condition** | The information has been entered correctly, the system saves all the information to the database and contacts a physiotherapist OR the system determines that information has be incorrectly entered and the user is prompted to fix it. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | CompleteAssessmentTest |
| **Participating actors** | Initiated by Patient |
| **Entry condition** | The Patient has successfully logged in to their account within Self Start and have an active rehabilitation plan with assessment tests. |
| **Flow of events** | 1. The Patient navigates to the treatment plan page and selects the complete assessment test option.  2. The system brings up all Patient’s relevant assessment tests assigned to them from a treatment plan.  3. The Patient now fills out the Assessment Test.  4. The system updates the related assessment rubrics and updates the progress by the Patient within the treatment plan |
| **Exit condition** | The system updates the filled-out assessment tests within the database for the Patient’s Physiotherapist to see. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | MakePayment |
| **Participating actors** | Initiated by Patient |
| **Entry condition** | The Patient has successfully logged in to Self Start and outgoing payment to be made. |
| **Flow of events** | 1. The Patient selects the make payment button.  2. Based on their past requests (ie. Booking an initial or supplementary appointment) the system displays the required payment amount. A form is also provided for to enter their payment information.  3. The Patient enters their payment information and clicks the submit button.  4. The system processes the payment, checking to see if the information provided is valid and responds accordingly. |
| **Exit condition** | The Patient has entered valid information and the system displays that the payment was successfully processed OR the Patient has entered incorrect information and the system prompts the Patient to enter correct their information. |
| **Quality requirement** | The system must use HTTPS to ensure that the payment request sent to the system is secure. |

|  |  |
| --- | --- |
| **Use case name** | ManageAppointments |
| **Participating actors** | Initiated by Patient. Participated in by Physiotherapist. |
| **Entry condition** | The Patient has accessed the Self Start website. |
| **Flow of events** | 1. The Patient selects the manage appointments button.  2. The system checks if the Patient has already logged in. If they have not they are taken to the initial appointment page where they can request an initial appointment or fill out an injury form. Otherwise, the Patient is logged in and is shown the book follow-up appointment screen.  3. The logged in Patient can either request a supplementary appointment, and/or cancel future appointments they have already booked as they choose.  4. The system acknowledges these changes made and displays a confirmation message to the Patient if their requests were successfully completed. |
| **Exit condition** | The system displays a confirmation message to the Patient that their requests were successful and have been saved. The system saves these changes to the database. |
| **Quality requirement** | **None** |

|  |  |
| --- | --- |
| **Use case name** | View/PerformTreatmentPlans |
| **Participating actors** | Initiated by Patient. |
| **Entry condition** | The Patient has successfully logged in and has a treatment plan assigned to them. |
| **Flow of events** | 1. The Patient navigates to their treatment plan page.  2. The system responds by displaying their treatment plan history and current status.  3. The Patient chooses to either view their progression through their treatment plan, view their current treatment plan or proceed with the next steps of their treatment plan(ie complete an assessment test)  4. The system acknowledges the Patient’s request and displays the appropriate page. |
| **Exit condition** | The system has successfully fulfilled the Patient’s request and the Patient is displayed the correct screen. |
| **Quality requirement** | **None** |

[In use-case modeling, the use cases often define the majority of the functional requirements of the system, along with some non-functional requirements. For each use case in the above use-case model, or subset thereof, enclose the use-case specification here. If you have documented use cases in a separate document, cross reference to all applicable external use-case specifications in this section. Make sure that each requirement is clearly labeled.]

## Functionality

[This section describes the functional requirements of the system for those requirements that are expressed in the natural language style. For many applications, this may constitute the bulk of the Modern SRS Package and thought should be given to the organization of this section. This section is typically organized by feature, but alternative organization methods, for example organization by user, or organization by subsystem may also be appropriate. Functional requirements may include: **feature sets, capabilities and security**.

Where application development tools (requirements tools, modeling tools, etc) are employed to capture the functionality, this section document will refer to the availability of that data and indicate the location and name of the tool which is used to capture the data.]

### <Functional Requirement One>

[The requirement description.]

## Usability

[This section should include all of those requirements that affect usability. Examples:

1. Specify the required training time for a normal users and power users to become productive at particular operations.
2. Specify measurable task times for typical tasks, or
3. Base usability requirements of the new system on other systems that the users know and like.
4. Specify requirements to conform to common usability standards – e.g., IBM’s CUA standards, or the GUI standards published by Microsoft for Windows 95.]

### <Usability Requirement One>

The requirement description.

## Reliability

[Requirements for reliability of the system should be specified here. Suggestions:

1. Availability – specify % of time available ( xx.xx%), hours of use, maintenance access, degraded mode operations etc.
2. Mean Time Between Failures (MTBF) – this is usually specified in hours, but it could also be specified in terms of days, months, or years.
3. Mean Time To Repair (MTTR) – how long is the system allowed to be out of operation after it has failed?
4. Accuracy – specify precision (resolution) and accuracy (by some known standard) that is required in the systems output.
5. Maximum bugs or defect rate – usually expressed in terms of bugs/KLOC (thousands of lines of code), or bugs per function-point.
6. Bugs or defect rate – categorized in terms of minor, significant, and critical bugs: the requirement(s) must define what is meant by a “critical” bug (e.g., complete loss of data, complete inability to use certain parts of the functionality of the system).]

#### *<Reliability Requirement One*>

[The requirement description.]

## Performance

[The performance characteristics of the system should be outlined in this section. Include specific response times. Where applicable, reference related Use Cases by name.

1. Response time for a transaction (average, maximum)
2. Throughput (e.g., transactions per second)
3. Capacity (e.g., the number of customers or transactions the system can accommodate)
4. Degradation modes (what is the acceptable mode of operation when the system has been degraded in some manner)
5. Resource utilization: memory, disk, communications, etc.]

### <Performance Requirement One>

[The requirement description.]

## Supportability

[This section indicates any requirements that will enhance the supportability or maintainability of the system being built, including coding standards, naming conventions, class libraries, maintenance access, maintenance utilities.]

### <Supportability Requirement One>

[The requirement description.]

GitHub will be used for version control during the rapid app development process to ensure accuracy and consistency throughout the life cycle. Versions will be individually committed by each developer until finished, followed by the collection of all versions to form a final report that can be found in a single location.

### <Supportability Requirement Two>

[The requirement description.]

The webpage should be available and accessible on any web browser the customer wishes to view it on. Cross-browser support systems will be introduced to ensure the developers priority to create a supported application, and will be maintained for a variety of different devices. Browsera is a free to download application used to compare browser output and report Javascript errors that might arise from various browsers.

### <Supportability Requirement Three>

[The requirement description.]

The system needs to be installed and hosted on the Marcotte Physiotherapy Clinic central server that is capable to run Node.js framework. The software team will develop the system in a way that allows for compatibility on the Clinic’s central server, maintained as needed in order to ensure availability 24/7 for patients.

### <Supportability Requirement Four>

[The requirement description.]

PayPal API will be adopted to handle user payment transactions on the webpage with verification. PayPal’s reputable brand name third party interface will prevent any errors that might be a result of building a unique payment system from scratch. Embedding this in the system offers peace of mind for the clients that their money is being handled with care.

### <Supportability Requirement Five>

[The requirement description.]

New video exercises and tutorials will be added regularly to avoid staleness of an out-of-date website. Material will be added to the website to ultimately better the patient’s experience by allowing access to as much useful content as possible.

## Design Constraints

[This section should indicate any design constraints on the system being built. Design constraints represent design decisions that have been mandated and must be adhered to. Examples include software languages, software process requirements, prescribed use of developmental tools, architectural and design constraints, purchased components, class libraries, etc.]

### <Design Constraint One>

[The requirement description.]

The Self Start system holds private information about a variety of patients, including medical and billing information. As a result, it should be heavily protected to maintain privacy between patients, and abide by any and all doctor-patient confidentiality laws. This will be completed by using HTTPS as its internet communication protocol to generate secure encryption keys between the server and client browser’s.

### <Design Constraint Two>

[The requirement description.]

The exercise tutorial videos uploaded on the web application must be accurate, and free from flaws in technique. The videos must be easy to follow along with, considering it will be serving patients of all levels of capability.

### <Design Constraint Three>

[The requirement description.]

The webpage should be constructed in a way that includes menu items, guidance, and animations that allow for smooth and easy access throughout the user’s experience. The user should not spend a lot time to understand how to use the application, but instead be seamlessly introduced to the format of functionalities in order to get the full efficiency out of Self Start’s online physiotherapy service.

### <Design Constraint Four>

[The requirement description.]

The database containing scheduling information on the physiotherapists will be created in such a way that prevents conflicts between appointments. If a patient has previously booked an appointment with a physiotherapist, a different patient cannot schedule an appointment for that time with the same physiotherapist. The database will continuously update to keep scheduling information accessible at all times to ensure this is not the case.

## Online User Documentation and Help System Requirements

[Describes the requirements, if any, for on-line user documentation, help systems, help about notices, etc.]

The Developers plan to complete the program under the rules of modular programming and delegated individual tasks. Delegated tasks ensure that each developer is an expert in at least one area of the entire system, and can provide a complete technical documentation on the system with combined knowledge. The documentation and system help will be consistent and thorough, yet still basic enough for the system administrator to follow when they have any questions or concerns with how the program is supposed to operate. The Self Start system will include 1 year worth of personal support from any or all team members, and provide each member’s contact information, since all of the developers are returning for a final undergraduate year at Western University.

## Purchased Components

[This section describes any purchased components to be used with the system, any applicable licensing or usage restrictions, and any associated compatibility/interoperability or interface standards.]

As the system developers are not qualified physiotherapists, experts need to be hired in order to model the required exercises correctly. Since physiotherapy is a complex art form at a small scale, even the smallest errors in technique could be dangerous to the patient’s health, and should be avoided by hiring professionals. This is the main area of concern regarding paid components for the system. The open source PayPal API to authenticate user payments is free to use, and therefore not a concern for budgeting. Payment authentication is an important feature that needs to operate flawlessly in order to maintain relationships with paying customers, and PayPal was sought to be the most efficient way to complete this feature after much research.

## Interfaces

[This section defines the interfaces that must be supported by the application. It should contain adequate specificity, protocols, ports and logical addresses, etc, so that the software can be developed and verified against the interface requirements.]

### User Interfaces

The application will support two primary views which will depend on the permissions of the user. In short, there will be a Physiotherapist view and a Client view. These interfaces will be designed for usability as it needs to allow easy navigation and understandability. Lastly, there will be a tertiary Admin view solely for the administer of the system. This view allows extensive permissions and administration of the application.

### Hardware Interfaces

This application will require hardware interfaces. A server that is able to be accessed through HTTP requests is required. Experimental tests and modifications may be performed using port 8080 however by default the server will be listening on port 80, therefore our client-side port will be 80 by default. For security, the Secure-Socket Layer will be used also known as Transport Layer Security. IP addresses are unknown at this time. The server should also be running 24/7 without interruption.

### Software Interfaces

1. For this application, we will be using an external JavaScript framework by the name of ember.js, as this requires a large amount of detail and many different user interactions. The use of this framework allows for complex user-interactions by managing the complexity of the web app. This is a third-party framework that is free and provides the described extension to JavaScript.
2. Like many other projects, this will require a purchased component. Due to the need to continuously run the application, a web server that is always active must be purchased and used to host the final product as well as dealing with user traffic. For testing purposes a development server will be used however, the deployed product will contain this purchased component.
3. Finally, a database will be required for this. Similar to many new systems, a new database will be used. This will possibly be filled with prior data from the previous software system Marcotte Physiotherapy had implemented. This component is an interface to view the records in an organized manor and will interact with the newly developed system. It is a newly created component.

### Communications Interfaces

There must be a communication interface to handle the exchange of information between the client machine and the server that is hosting the application (Unknown at this time). As discussed above, an HTTP server will be used along with Transport Layer Security (TLS) primarily on port 80. This will be directly connected to the internet server and will ensure the safe transmission of data.

## Licensing Requirements

[Defines any licensing enforcement requirements or other usage restriction requirements that are to be exhibited by the software.]

## Legal, Copyright and Other Notices

[This section describes any necessary legal disclaimers, warranties, copyright notices, patent notice, word mark, trademark, or logo compliance issues for the software.]

## Applicable Standards

To accompany this project, a set of applicable standards must be complied to. These standards consist of; legal standards, quality and regulatory standards and finally industry standards for usability, interoperability, and operating system compliance.

The first set of applicable standards require that the application takes extra care in dealing with patient records and information. The document states “Provide secure authentication system to protect the privacy of the patients and to guarantee the confidentiality of their data”. This creates the need to provide data protection for all users of the application. Measures will also be implemented to protect the database, ensuring that data is handled securely and without corruption.

There must also be a set of usability standards to ensure accessibility to those with visual impairment. These standards will conform with any current policies in effect at Marcotte Physiotherapy.

The last of standards is based on the following objective “Develop a Web-based application to be 24/7 available to the patients of Marcotte Physiotherapy Clinic and their physicians and administration stuff”. It will be assured that this application will run constantly with no issues. As well, it is necessary to deliver a product with multi-device compatibility across multiple browsers to ensure usability.