

Patient Handling System

Final Year Project Report



Author: Sean Mcloughlin

Student ID: 12347451

Supervisor: Dr. Owen Molloy

Acknowledgements

I would like to thank my project supervisor, Dr. Owen Molloy, for his help and guidance throughout the course of this project.

Table of Contents

1	Introduction	5
1.1	Current Practices	5
1.2	Automating the Process	6
2	Technical Review, Development Process and Previous Work.....	7
2.1	Source Control	7
2.1.1	Benefits	7
2.2	Application Architecture.....	7
2.2.1	Facility Architecture Possibilities	8
2.3	Product Backlog	8
2.4	Weekly Meetings	8
2.5	Previous Solutions	9
2.5.1	Impact on Technology Choice	9
3	Technical Issues.....	10
3.1	Database Design	10
3.1.1	Webpages Tables	10
3.1.2	Decision Tree Storage.....	11
3.2	Server Instances.....	11
3.3	Decision Tree Creator	12
3.3.1	Displaying the Trees	12
3.3.2	Updating the Tree	13
3.3.3	Entering Solutions	14
3.4	User Accounts	16
3.5	Unit Tests	17
3.5.1	The Mocking Framework	17
3.5.2	Unit Test Integration	17
3.6	Deployment Issues.....	18
3.6.1	Initial Deployment.....	18
3.6.2	Move to Azure.....	18
3.7	Code Refactoring	18
4	Results	19
4.1	Application Overview.....	19
4.1.1	Patients	19

4.1.2	Attributes	20
4.1.3	Equipment.....	21
4.1.4	Trees.....	22
4.2	Handling Plans	23
4.3	User Accounts.....	24
4.4	Validation.....	25
4.4.1	Server Side.....	25
4.4.2	Client Side Validation	25
4.5	Bootstrap	25
4.6	Cross Browser Compatibility.....	26
5	Conclusion	27
5.1	Readiness for Production.....	27
5.2	Future Work.....	27
5.3	Overall Conclusion	27

1 Introduction

Handling patients who have mobility problems, is not surprisingly, an intricate task. It involves the consideration of a number of different variables, and the use of many different movement tasks, with different processes required for each. The patients themselves, for one thing, can vary significantly in a number of ways, such as in weight, co-operation level, weight bearing capacity, whether they are injured or not, and more. Other factors that also need to be included are the equipment available to the patient handler, and the features of each one. With different movement tasks, the status of some or all of these attributes need to be included in the final action taken to move the patient. Neglecting any could result in poor outcomes, ranging from injury to the patient or patient handler, to undue stress for both parties.

1.1 Current Practices

To aid patient handlers with the above process, a series of algorithms are in place, which present the patient handler with a handling plan, based on the values of various different patient and equipment attributes. Figure 1.1 shows an example of such an algorithm.

Algorithm 4: Reposition in Bed: Side-to-Side, Up in Bed
Last rev. 10/01/08

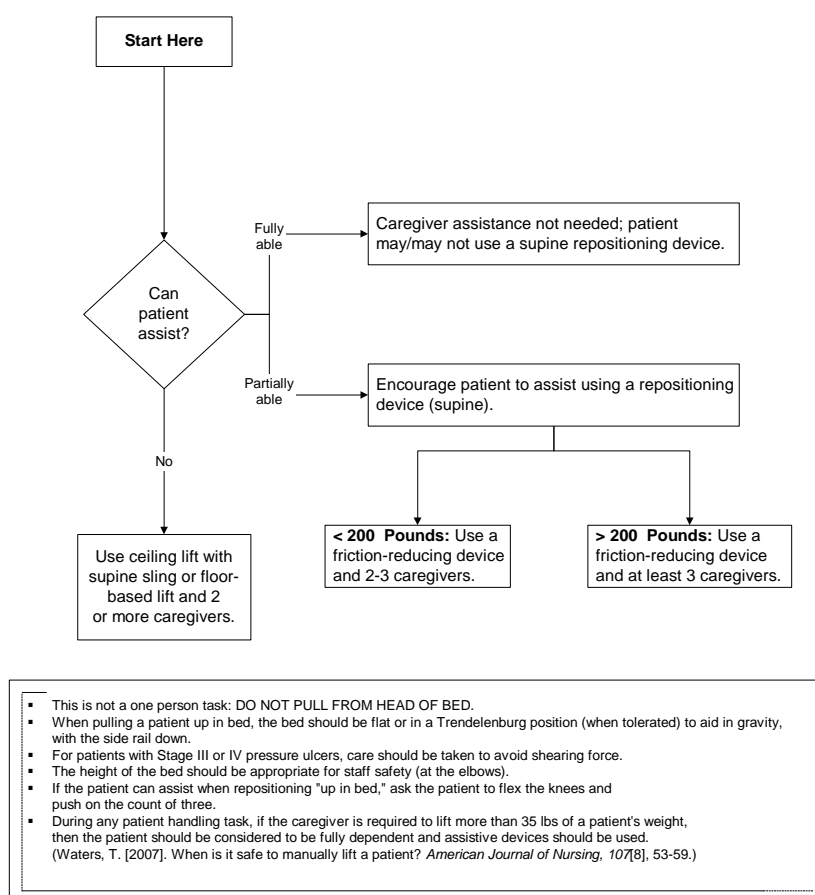


Figure 1.1 – A sample patient handling algorithm

In total, there are sixteen algorithms that a patient handler has to either learn off or have on their person at all times when on duty. They also need to know of, or have access to the various

patient/equipment attributes, such as weight, co-operation level, etc. in order to iterate through the algorithm. This results in a process that, if due care was not taken, could produce many errors - for example if the patient handler was under time constraints and guessed at some of the patients attributes, or if they read a logical operator incorrectly - something easily done by someone not technically minded. The purpose of creating an application for this process is to avoid these errors, which would lead to a better quality of service, and ultimately more satisfied patients and patient handlers.

1.2 Automating the Process

With all of the above in mind, there is a need for an application that would aid in this process. This application would be beneficial in different ways, the main one being that the application would iterate through each algorithm, producing handling plans for any or all patients, based on data entered into the system. How this is integrated into the patient handler's day to day job could vary, however a likely scenario is that they would print off the required handling plans for each patient, and store them in a location that is easily accessed when moving the patient. Other implementations would also be possible, such as using a tablet to access the handling plan for a patient. This would be a more expensive option however, but could be more effective if the patient or equipment attributes changed regularly.

This information led to a decision that a web application would be the best type of implementation for these particular requirements. Given the fact that the application would only be accessed by users within the facility, it would make sense if the application was hosted on the facilities internal network. However, this may not be possible in some facilities where there could be no current network in place, and in that case, the application and associated database could then be hosted on cloud based services, such as Microsoft Azure, which, on a basic subscription plan, consisting of a web application and a database, is estimated to cost €600 annually*. The web application itself would store all of the patients' details, along with the various handling plans, which would both be editable.

*Figure received from the following link: <https://azure.microsoft.com/en-us/pricing/calculator/>

2 Technical Review, Development Process and Previous Work

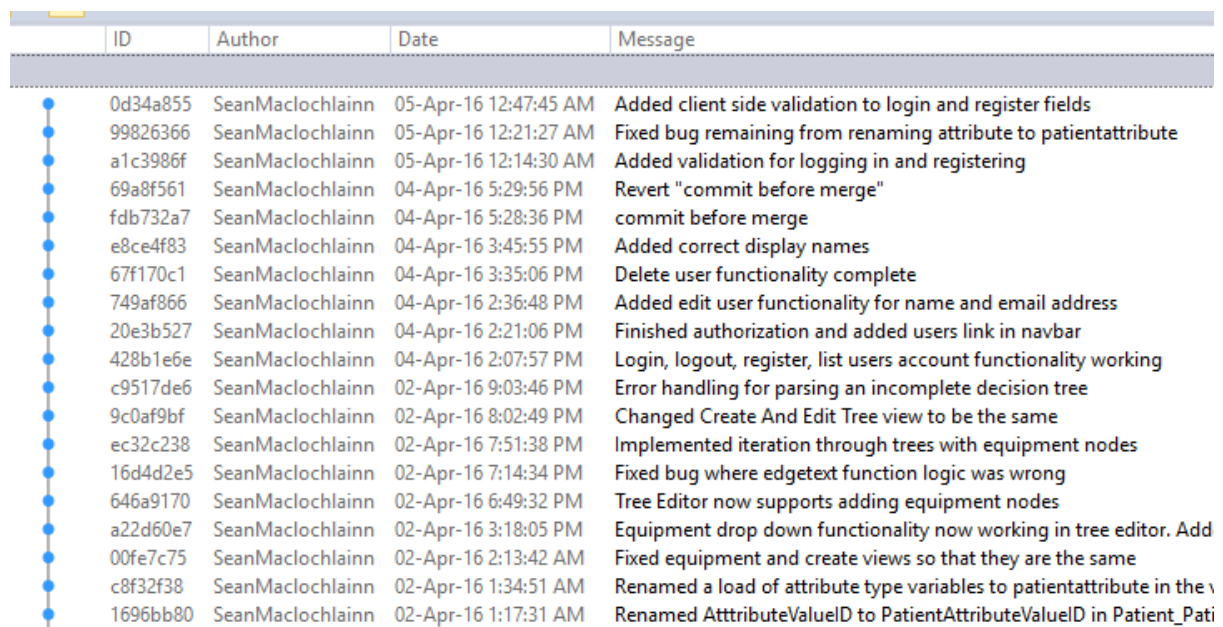
This section will expand upon the technologies suggested during the project definition document, and will detail which of these were chosen. It will also describe the development process used during the completion of the project.

2.1 Source Control

The source control system eventually chosen was git, with the repository then being hosted on GitHub. A Visual Studio extension for GitHub was used to connect to the repository and complete all source control operations.

2.1.1 Benefits

Even though there was only one developer working on the application, the use of source control still provided some key benefits to the development process. Firstly, it was very useful to be able to revert changes made to a file. An example of this was, when exploring the different packages that could be used for the document editor, changes were made to some of the configuration files when installing these packages. It was unclear which changes were added from installing the package, and which were there originally. With the benefits of source control however, the file could be reverted back to the previous state before the package was installed, which removed the additional changes made to the file. Secondly, committing each change to source control provided documentation and timestamps for what changes were made, and when.



	ID	Author	Date	Message
•	0d34a855	SeanMaclochlainn	05-Apr-16 12:47:45 AM	Added client side validation to login and register fields
•	99826366	SeanMaclochlainn	05-Apr-16 12:21:27 AM	Fixed bug remaining from renaming attribute to patientattribute
•	a1c3986f	SeanMaclochlainn	05-Apr-16 12:14:30 AM	Added validation for logging in and registering
•	69a8f561	SeanMaclochlainn	04-Apr-16 5:29:56 PM	Revert "commit before merge"
•	fdb732a7	SeanMaclochlainn	04-Apr-16 5:28:36 PM	commit before merge
•	e8ce4f83	SeanMaclochlainn	04-Apr-16 3:45:55 PM	Added correct display names
•	67f170c1	SeanMaclochlainn	04-Apr-16 3:35:06 PM	Delete user functionality complete
•	749af866	SeanMaclochlainn	04-Apr-16 2:36:48 PM	Added edit user functionality for name and email address
•	20e3b527	SeanMaclochlainn	04-Apr-16 2:21:06 PM	Finished authorization and added users link in navbar
•	428b1e6e	SeanMaclochlainn	04-Apr-16 2:07:57 PM	Login, logout, register, list users account functionality working
•	c9517de6	SeanMaclochlainn	02-Apr-16 9:03:46 PM	Error handling for parsing an incomplete decision tree
•	9c0af9bf	SeanMaclochlainn	02-Apr-16 8:02:49 PM	Changed Create And Edit Tree view to be the same
•	ec32c238	SeanMaclochlainn	02-Apr-16 7:51:38 PM	Implemented iteration through trees with equipment nodes
•	16d4d2e5	SeanMaclochlainn	02-Apr-16 7:14:34 PM	Fixed bug where edgetext function logic was wrong
•	646a9170	SeanMaclochlainn	02-Apr-16 6:49:32 PM	Tree Editor now supports adding equipment nodes
•	a22d60e7	SeanMaclochlainn	02-Apr-16 3:18:05 PM	Equipment drop down functionality now working in tree editor. Add
•	00fe7c75	SeanMaclochlainn	02-Apr-16 2:13:42 AM	Fixed equipment and create views so that they are the same
•	c8f32f38	SeanMaclochlainn	02-Apr-16 1:34:51 AM	Renamed a load of attribute type variables to patientattribute in the \
•	1696bb80	SeanMaclochlainn	02-Apr-16 1:17:31 AM	Renamed AttributeValueID to PatientAttributeValueID in Patient_Pati

Figure 2.1 - A selection of the recent commits made to the project

2.2 Application Architecture

The application itself is a distributed system, albeit a relatively simple one currently. It is also currently hosted on a Microsoft Azure server. This then communicates with the database, which is located on an NUIG server on the college's campus. Ideally the application and database would be both on the same network, as this would give an improved response time, however the current solution is still satisfactory. If the application was taken to a further stage, an e-mail server would

also need to be communicated with, however this would most likely not be within the same network.

2.2.1 Architecture Possibilities for Facility

If the application was to be used by a facility, an ideal scenario would be to host the application on the facilities LAN (Local Area Network). This would be optimal for performance, and also potentially security. However, it would require the maintenance of the servers if there weren't any on site beforehand. As mentioned previously, an alternate option would be to host the application on the cloud.

2.3 Product Backlog

New features and bugs found in the project were created and found regularly during the project development. These ranged from minor stylistic changes, to major feature requests. Using elements from the agile software development methodology, a product backlog was maintained, which stored all of these development tasks in one place. This was in the form of an excel spreadsheet. The details of each feature or bug, mainly the description, priority, and status were all documented within this spreadsheet.

	A	B	C
1	Item	Priority	Status
2	Fix bug so that you can add a new attribute and still be able to view patient details		Done
3	Add "Are you sure you want to delete?" page when deleting attribute, similarly to when you delete a patient		Done
4	Rename PatientAttribute table to PatientsAttributes or Patients_Attributes, and all references to table		Done
5	Create functionality to add different amounts of attribute values, not just three		Done
6	Make nodes keep away from each other enough		Done
7	Add support for editing tree name		Done
8	Make sure solutions are deleted when tree is deleted		Done
9	Disable automatic creation of tree with no nodes		N/A
10	Add equipment, similarly to patients		Done
11	Look into adding arrows onto tree		Done
12	Put tree editor buttons on one side and display on the other		Done
13	Look at complicated trees and see can they be implemented		Done
14	Add a delete function to delete particular nodes		Done
15	Add error handling to tree editor		Done
16	Put bootstrap nav tabs in the tree editor		Not Started
17	Rename all instances of attribute to patient attribute		Done
18	Make form stuff in attributes and patients bootstraphish		Done
19	Make the create tree the same as edit tree		In Progress
20	add login/logout for patient handler or administrator		Not Started
21	Email nuig it about publishing to iis server problem		In Progress
22	Make tree name long enough		Done
23	Make a test plan		In Progress
24	Do client side and server side validation		Not Started
25	Refactor code to separate data layer from logic layer		In Progress
26	Error handling for tree that can't be parsed		Done
27	Include equipment in tree		Done
28	Generate printable handling plans		Done
29	Add loading gif for ajax in tree editor		Not Started

Figure 2.2 - The product backlog for the application

Towards the end of the project, when the product backlog was maintained in a more serious manner, there was a noticeable improvement in productivity when working on the project. It was also much easier to see how close the project was to being completed, by looking at how much work was outstanding.

2.4 Weekly Meetings

Every week or two, a meeting was had with the project supervisor, where feedback was given on any features added since the last meeting, and problems encountered and their potential solutions, were discussed. This helped to keep the application development focused on the most important areas, and also ensured that the product was being built in a way in which the end users would be satisfied.

2.5 Previous Solutions

There have been two previous attempts at building an application for the company for this specific process. Both attempts were done by final year students, and while being functioning software, were never brought into a working environment by the company. Having access to the reports for both projects (however, not the actual software created), the previous work done provided a good resource for the work on this application. Mainly, the reports provided an excellent introduction to the problem domain. The software developed for this report was quite different than the software developed by the other two developers, so for this reason, the previous work was not as beneficial as could have been hoped.

2.5.1 Impact on Technology Choice

From analysing the previous work done, it was clear there were some advantages and disadvantages to the approaches taken by both. Ultimately however, for this application it was decided to go with the proven environment of ASP.NET MVC. This is a popular choice when building applications with many CRUD (Create, Read, Update and Delete) type operations, which make up a significant portion of this one – i.e. the storing and editing of patient, user, and equipment data.

3 Technical Issues

This section will detail some of the more technical aspects of the project that were encountered, and how they were dealt with – ranging from designing the architecture of the application, to describing how different front end plugins were used to solve various problems.

3.1 Database Design

The goal when designing the database for this application was to create a flexible and easily understood database schema. This was achieved using database normalization techniques, which among other things, eliminated data redundancy, and ensured that making changes to the schema and performing CRUD operations did not involve much complexity. For example, initially it would seem to make sense to store the attributes for each patient (weight bearing capacity, co-operation level, etc.) in the Patient table. Instead however, they are put in a table of their own (PatientAttribute), making adding new patient attributes at any point a trivial operation, one that would have otherwise involved adding a new column to the Patient table.

3.1.1 Webpages Tables

The tables listed below whose names begin with webpages_ have been automatically created by a .NET Membership provider called SimpleMembership. This is a tool that helps to streamline the process of integrating user accounts and user access into ASP.NET applications, of which more will be discussed about later. It is also worth noting that the Password attribute of the webpages_Membership table created by SimpleMembership is encrypted. This means that anyone with access to the database would be unable to see what any user's password is. The process of encrypting and unencrypting these values when they need to be accessed is handled by SimpleMembership.

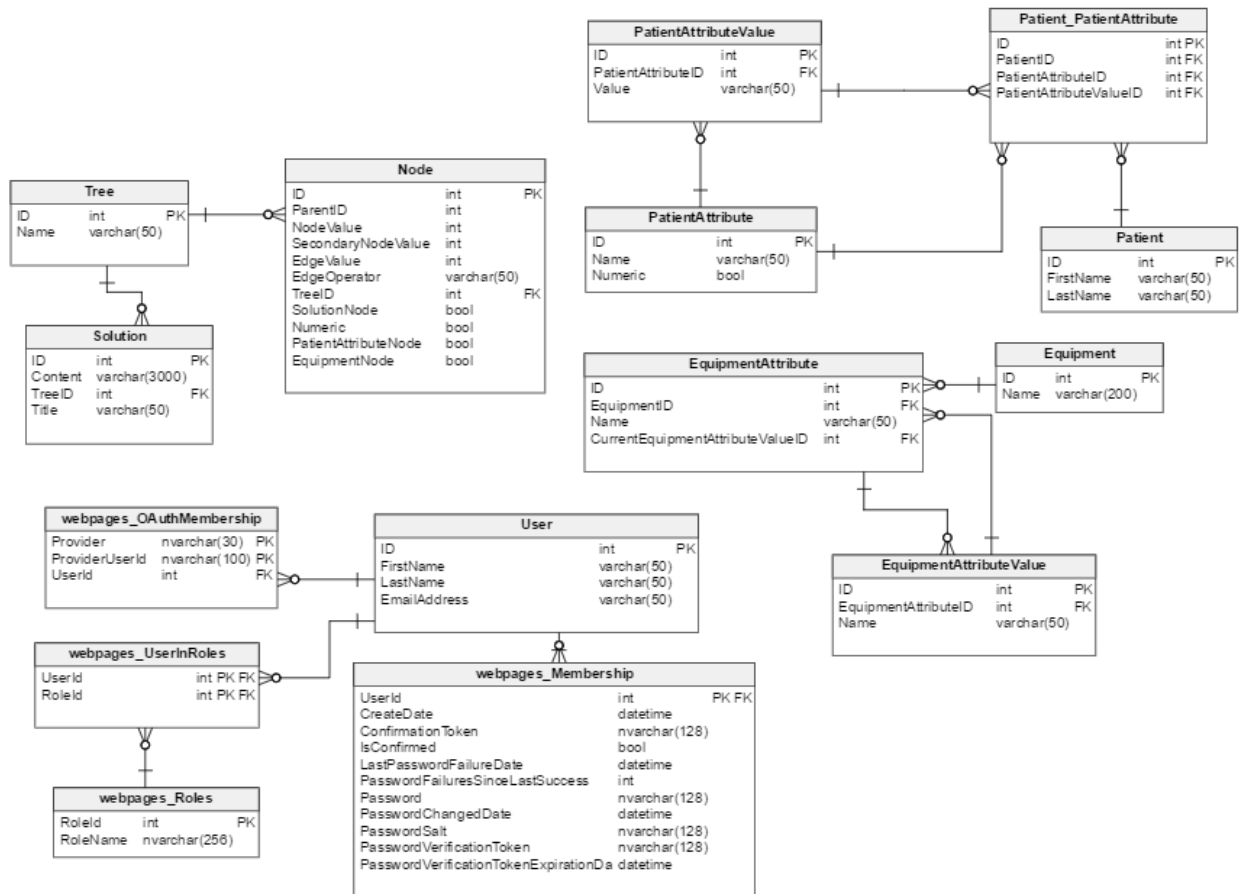


Figure 3.1 - The entity relationship diagram for the database

3.1.2 Decision Tree Storage

One of the more complex parts of the database is the Node table, which is a table that stores the nodes for each tree. In general, there are many different ways to store decision trees, from storing them in an XML document, to the various different relational or graph database models that can be used. After some research into the advantages and disadvantages of each, it was decided to use an adjacency list model to store each tree.

A brief explanation of this model is as follows: each row in the table represents a node in the tree, and also contains a pointer to some other row that represents the parent node of the current node. Edge values and any other values that each node has are then represented as attributes in each row.

3.2 Server Instances

There were two different database server instances created, one stored locally, on the computer used to develop the application, and one on the NUIG server. The NUIG database server and the Microsoft Azure cloud platform was treated as a UAT (User Acceptance Testing) environment, so that anybody wishing to view/test the application during the course of its development, could do so. To ensure that both databases performed in the same way, the schema of each one was scripted, and these scripts were then compared using text comparison software. This ensured that each database was an exact replica of the other.

If the database was to be moved to a server at one of the company's facilities at some point in the future, a script of either database could be generated and then executed on said server.

3.3 Decision Tree Creator

Designing an interface for creating the decision trees was the most technically challenging aspect of this application. Various different JavaScript plugins and other front end technologies were used to achieve this. The goal was to design an interface that was both easy to use and effective.

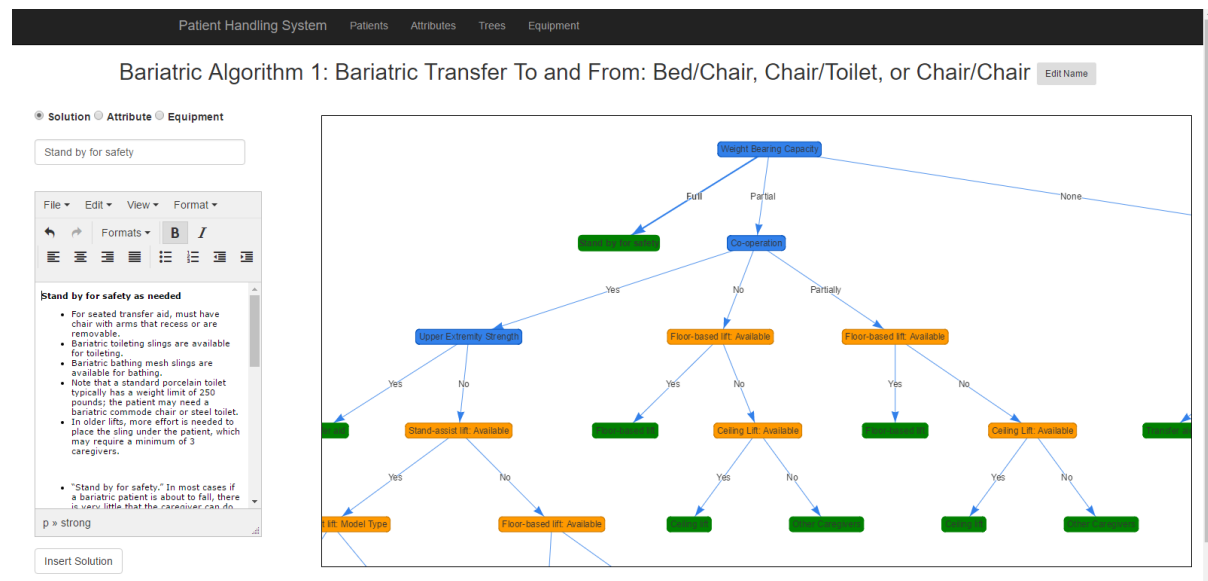


Figure 3.2 - The decision tree editor

3.3.1 Displaying the Trees

Research on how to display the trees was completed during semester one, where different front end tools were examined too see which were the most suitable to perform this task. Eventually a decision was made to use Vis.js, a JavaScript visualization library, of which one of its applications is that of displaying decision trees. Trees are created by specifying the nodes, elements and various configuration settings within the JavaScript script eventually executed within the HTML document created.

```

<script type="text/javascript">
  var i;
  var nodes = new vis.DataSet();
  @foreach (var node in Model)
  {
    if(node.SolutionNode)
    {
      @:nodes.add({ id: @node.ID, label: '@node.NodeText()', shape: 'box', color: 'green' });
    }
    else if(node.NodeValue == 0) //leaf node (one that isn't a solution node)
    {
      @:nodes.add({ id: @node.ID, label: 'Select', shape: 'circle', color: 'red' });
    }
    else if(node.PatientAttributeNode)//patient attribute node
    {
      @:nodes.add({ id: @node.ID, label: '@node.NodeText()', shape: 'box', color: '#3280EA' });
    }
    else //equipment attribute node
    {
      @:nodes.add({ id: @node.ID, label: '@node.NodeText()', shape: 'box', color: '#ff9900' });
    }
  }
}

```

Figure 3.3 - A mixture of Razor and JavaScript is used to render each tree

To do this a mixture of Razor and JavaScript was used. Razor is a programming syntax used to retrieve data from the model, and render this data onto a HTML file. In this case, as well as generating markup for the HTML file, Razor was also used to generate data for the Vis.js script containing the decision tree. In this case, the model object contains the list of node objects that are to be displayed in the tree. An example of the final output of this process is shown below. The edges for the tree are also generated in a similar fashion, further down in the script.

```

<script type="text/javascript">
  var i;
  var nodes = new vis.DataSet();
  nodes.add({ id: 225, label: 'Weight Bearing Capacity', shape: 'box', color: '#3280EA' });
  nodes.add({ id: 226, label: 'Stand by for safety', shape: 'box', color: 'green' });
  nodes.add({ id: 227, label: 'Co-operation', shape: 'box', color: '#3280EA' });
  nodes.add({ id: 228, label: 'Co-operation', shape: 'box', color: '#3280EA' });
  nodes.add({ id: 229, label: 'Upper Extremity Strength', shape: 'box', color: '#3280EA' });
  nodes.add({ id: 230, label: 'Floor-based lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 231, label: 'Floor-based lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 232, label: 'Upper Extremity Strength', shape: 'box', color: '#3280EA' });
  nodes.add({ id: 233, label: 'Floor-based lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 234, label: 'Floor-based lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 235, label: 'Transfer aid', shape: 'box', color: 'green' });
  nodes.add({ id: 236, label: 'Stand-assist lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 239, label: 'Floor-based lift', shape: 'box', color: 'green' });
  nodes.add({ id: 240, label: 'Ceiling Lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 241, label: 'Ceiling lift', shape: 'box', color: 'green' });
  nodes.add({ id: 242, label: 'Other Caregivers', shape: 'box', color: 'green' });
  nodes.add({ id: 246, label: 'Stand-assist lift: Model Type', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 247, label: 'Floor-based lift: Available', shape: 'box', color: '#ff9900' });
  nodes.add({ id: 248, label: 'Stand assist', shape: 'box', color: 'green' });
  nodes.add({ id: 249, label: 'Stand assist', shape: 'box', color: 'green' });

```

Figure 3.4 – A sample of a Vis.js script

This was a flexible approach to adding data to the script; as you can see in the above screenshot, the shape, colour and contents of the node can be specified based on the values of node object.

3.3.2 Updating the Tree

To update the tree, Ajax and partial views were used. Each time the tree was updated, a node and relevant stub nodes were added to the tree. The parent node for the node to be inserted is specified

by utilizing an onclick() method, which upon clicking a node, adds a hidden element containing the nodes id to the form. This node then will be the parent node of the next set of nodes added. The left hand side of the screen acts as a form, containing the details of the node that is to be entered into the tree. When entering equipment nodes, JavaScript is also used to populate the contents of a second dropdown list in this form, based upon what equipment is chosen in the first.

This Ajax POST request is then submitted, the node is added to the tree in the database, and the partial view, which contains the tree is then updated upon this submission. Other features, such as deleting nodes, and error handling (if a non-stub node is selected) are also implemented.

3.3.3 Entering Solutions

The leaf node on each tree contains a handling plan, referred to in the database as a solution. This is a set of instructions for the patient handler, detailing what to do in the current specific patient and equipment circumstances. These set of instructions can typically be quite lengthy, and also may require additional images/video content to be added. As such it was decided to use a text editor plugin, TinyMCE. TinyMCE is a JavaScript plugin, designed to create HTML documents, which are formatted in a similar style to a word document. It is designed to be used to replace a textarea element. When a form containing a TinyMCE plugin is submitted to the server, the output of the textarea element will be a HTML document in the form of a string.

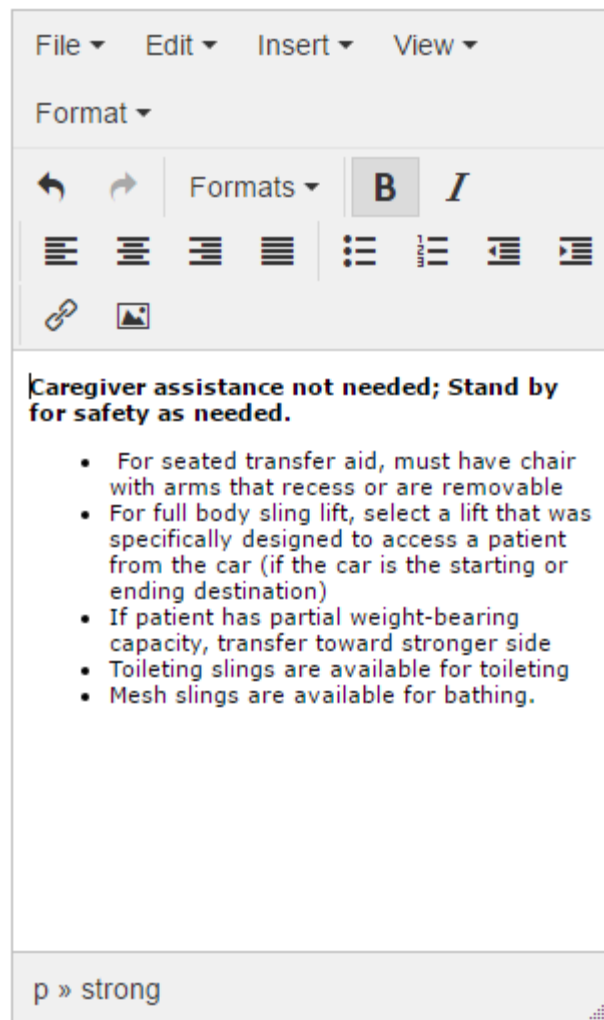


Figure 3.5 - The TinyMCE Editor

ID	Content
80	<code><p>Caregiver assistance not needed; Stand by for safety as needed</...</code></code>
81	<code><p>Stand-and-pivot technique using a gait/transfer belt (1 caregiver) or powered...</p></code></code>
82	<code><p>Use full-body sling lift and 2 caregivers</p> For seated tra...</code></code>
83	<code><p>Stand-and-pivot technique using a gait/transfer belt (1 caregiver) or powered...</p></code></code>

Figure 3.6 - A handling plan, stored as a string with HTML formatting created by the TinyMCE plugin

The editor supports many typical document editing functionalities, such as bullet points, italics, numbered lists, tables and much more.

Images and videos are also supported on TinyMCE. These are displayed by using `` and `<iframe>` tags, which can be seen upon inspection of the HTML markup. This means that all that is required to utilize these elements is a source for the images and videos. This could be a local file, stored on the server itself, or else a link to an image/video hosting website. Either method would be a valid approach. Image and video hosting websites also usually support private uploads, where the content is not listed anywhere on the site, and a direct link is the only way in which the content can be

viewed. This could potentially be an option if the web application was eventually used in a handling facility.

Algorithm 1: Transfer to and From: Bed to Chair, Chair to Toilet, Chair to Chair, or Car to Chair

Handling Plan for Test Patient

Use a ceiling lift



- For seated transfer aid, must have chair with arms that recess or are removable.
- Bariatric toileting slings are available for toileting.
- Bariatric bathing mesh slings are available for bathing.
- Note that a standard porcelain toilet typically has a weight limit of 250 pounds; the patient may need a bariatric commode chair or steel toilet.
- In older lifts, more effort is needed to place the sling under the patient, which may require a minimum of 3 caregivers.

Figure 3.7 - A handling plan containing an embedded image

As can be seen in the above image, the final format of the handling plan is similar to that of a word document. The patient handlers could then print out this document to have on hand. Using CSS it was possible to add page breaks to this document, thus enabling each handling plan to have its own page.

3.4 User Accounts

User accounts are an important part of many web facing applications, and this one is no exception. Accounts needed to be implemented to enable users to log on, and provide different users with different levels of access to various parts of the application. The addition of user accounts to a web application brings with it many time intensive development overheads, such as registering, logging in, password resetting, and user roles. This would take a lot of time to successfully implement if it was to be built from scratch. Fortunately there are providers that help with this process, and in this application a provider called SimpleMembership was used. SimpleMembership allowed the quick creation of these above functionalities, saving a lot of development time.

```
public void RegisterUser(string emailAddress, string password, string firstName, string lastName, string userType)
{
    WebSecurity.CreateUserAndAccount(emailAddress, password, new { FirstName = firstName, LastName = lastName });
    var roleProvider = new SimpleRoleProvider(Roles.Provider);
    roleProvider.AddUsersToRoles(new string[] { emailAddress }, new string[] { userType });
}
```

Figure 3.8 – Sample code from the SimpleMembership provider

3.5 Unit Tests

The concept of integrating unit tests and test driven development into the development process of an application is one that is widely used in industry. This is due to three reasons. Firstly, when any changes are made to the application (including major rewrites that could have effects on other sections of code), it ensures that the change made does not affect the functionality of the rest of the application. Each unit test runs when the application compiles – so any functionality that has been broken should be immediately found. Secondly, they are an effective way of proving that a section of code written does what it was supposed to do. Thirdly, they are a useful way for developers not familiar with the code base to understand what certain sections of code are supposed to do.

3.5.1 The Mocking Framework

Much of unit testing in ASP.NET MVC is performed using a concept called dependency injection. Essentially, a mock repository is created that is a replica of the real database, and unit tests are then performed on this repository. This is done by using constructors in the controller classes, where an instance of the object containing the database tables is passed in. Usually, this object would represent the database itself, however when unit testing, this object is mocked, and thus contains test data, specified in the creation of the object. The framework used to accomplish this overall process was called Moq, an open source .NET mocking framework.

```
[TestMethod]
public void UpdateTreeController()
{
    //set up context and dataservice instance
    patientHandlingContext = MockDatabase.GetNodeMockDbSet(new List<Node> {
        new Node { ID = 1, ParentID = 0, NodeValue = 1, EdgeValue = 0, EdgeOperator = null, TreeID = 1, SolutionNode = false, Numeric = false },
        new Node { ID = 2, ParentID = 1, NodeValue = 0, EdgeValue = 1, EdgeOperator = "=", TreeID = 1, SolutionNode = false, Numeric = false },
        new Node { ID = 3, ParentID = 1, NodeValue = 2, EdgeValue = 2, EdgeOperator = "=", TreeID = 1, SolutionNode = false, Numeric = false },
        new Node { ID = 4, ParentID = 3, NodeValue = 1, EdgeValue = 3, EdgeOperator = "=", TreeID = 1, SolutionNode = true, Numeric = false }
    }, patientHandlingContext);
    var mock = new Mock<ITreeRepository>();
    mock.Setup(i => i.DeleteRegularNode(It.IsAny<int>(), It.IsAny<string>()));
    mock.Setup(i => i.EnterSolutionNode(It.IsAny<string>(), It.IsAny<int>(), It.IsAny<string>(), It.IsAny<string>()));

    TreesController treesController = new TreesController(patientHandlingContext.Object, mock.Object);

    //tests when a parent node hasn't been selected
    //arrange
    TreeEditorViewModel treeEditorVM = new TreeEditorViewModel { ParentNodeID = null, Tree = new Tree { ID = 1 } };

    //act
    treesController.UpdateTree(treeEditorVM, "true");

    //assert
    Assert.AreEqual(treesController.ModelState["NoNodeSelected"].Errors[0].ErrorMessage, "Please select a parent node");
}
```

Figure 3.9 – A sample unit test that mocks the database

Of course it would have been possible to use an actual database to perform unit tests, however this would mean that the tests would not run in main memory. This would mean that they would take a longer time to execute, thus slowing down the process.

3.5.2 Unit Test Integration

Overall, it was a useful exercise to integrate the unit tests into the application. A lot was learned during the process, and the groundwork has been built to fully implement test driven development into the project. However, the process of implementing unit tests was not as successfully applied as had been hoped for. To successfully implement unit tests, a specific type of code layout has to be implemented to all parts of the code base that need to be tested. It is a modular layout, where different parts of the application are separated, allowing for easier testing. For example, the data access layer and the business logic layer are put into two different sections, allowing each to be tested separately. Unfortunately, the knowledge necessary to implement this approach was only

gained towards the later stages of the project, however if more time was had, this would have been implemented.

3.6 Deployment Issues

Deployment of the application was an important task that had to be undertaken during the development of this application. Successful deployment would mean that the application users, or anyone else concerned, could view the progress made.

3.6.1 Initial Deployment

Firstly, on deploying to the NUIG IIS (Internet Information Services) server, an error message was received. After researching the error, it was discovered that the .NET version running on the server was version 4.5. This meant that the only applications that could be ran on the server were those using the .NET 4.5 framework. The application was then downgraded to .NET 4.5 (from .NET 4.5.2) and then successfully deployed. Fortunately, this did not have any impact on any of the applications functionality.

3.6.2 Move to Azure

Towards the end of the application development, the application suddenly stopped deploying to the NUIG IIS server, for an unknown reason. It was determined however, based on the evidence that other applications had also stopped deploying, that it was an issue with the server and not with the application itself. A ticket was raised with the IT support team to fix this issue, however at the time of writing it has not been resolved. This meant that the application would have to be deployed onto a different server, or not at all. It was then decided to use Microsoft Azure, a cloud based platform, to deploy the application. Students are able to use this service for free, allowing it to be successfully deployed to Microsoft Azure for no additional cost.

3.7 Code Refactoring

Code refactoring was done at various stages throughout the development of the project. This consisted mainly of renaming variables, creating functions, and separating the data access layer from the business logic layer, as discussed previously. At a later stage of the application development, only freshly created code could be refactored, as refactoring older code occasionally brought in bugs into the application, something that would be very detrimental when demoing the project. However the refactoring done up until this point was effective, and helped to improve the overall flexibility and clarity of the code base.

Even though older code written was not interfered with too much towards the end of development, the mistakes made writing this code was kept in mind when creating new code for newer features. This meant that some of the newer features were then written in an improved fashion. For example, the creation of the login and register functionalities were done in quite a modular way, which meant that adding client side and server side validation was then a very easy task, and if unit tests were to also have been written for these features, these would have also been relatively easy to create.

4 Results

This section will describe the main features of the application, and overview the different aspects of the final product created.

4.1 Application Overview

This section will detail each part the application that was developed.

4.1.1 Patients

Details

Patient

First Name	Joe
Last Name	Bloggs
Weight Bearing Capacity	None
Assistance Ability	None
Upper Extremity Strength	No
Assistance Ability	Partially
Sustain Limb Position	No
Bear Weight and Ambulate	Partially
Lower Extremity Injured	No
Minor Injury Weight	No
Co-operation	200
Surgical Position	None
Surgical Position	To/From semi-Fowler

[Back to List](#)

Edit

Patient

First Name	Joe
Last Name	Bloggs
Weight Bearing Capacity	None
Assistance Ability	None
Upper Extremity Strength	No
Assistance Ability	Partially
Sustain Limb Position	No
Bear Weight and Ambulate	Partially
Lower Extremity Injured	No
Minor Injury Weight	No
Co-operation	200
Surgical Position	None
Surgical Position	To/From semi-Fowler

Figure 4.1 – CRUD functionality for patients

Basic CRUD functionality is supported for patients. Patients all have an equal amount of attributes (defined in the next section), and a value must be selected for each one.

4.1.2 Attributes

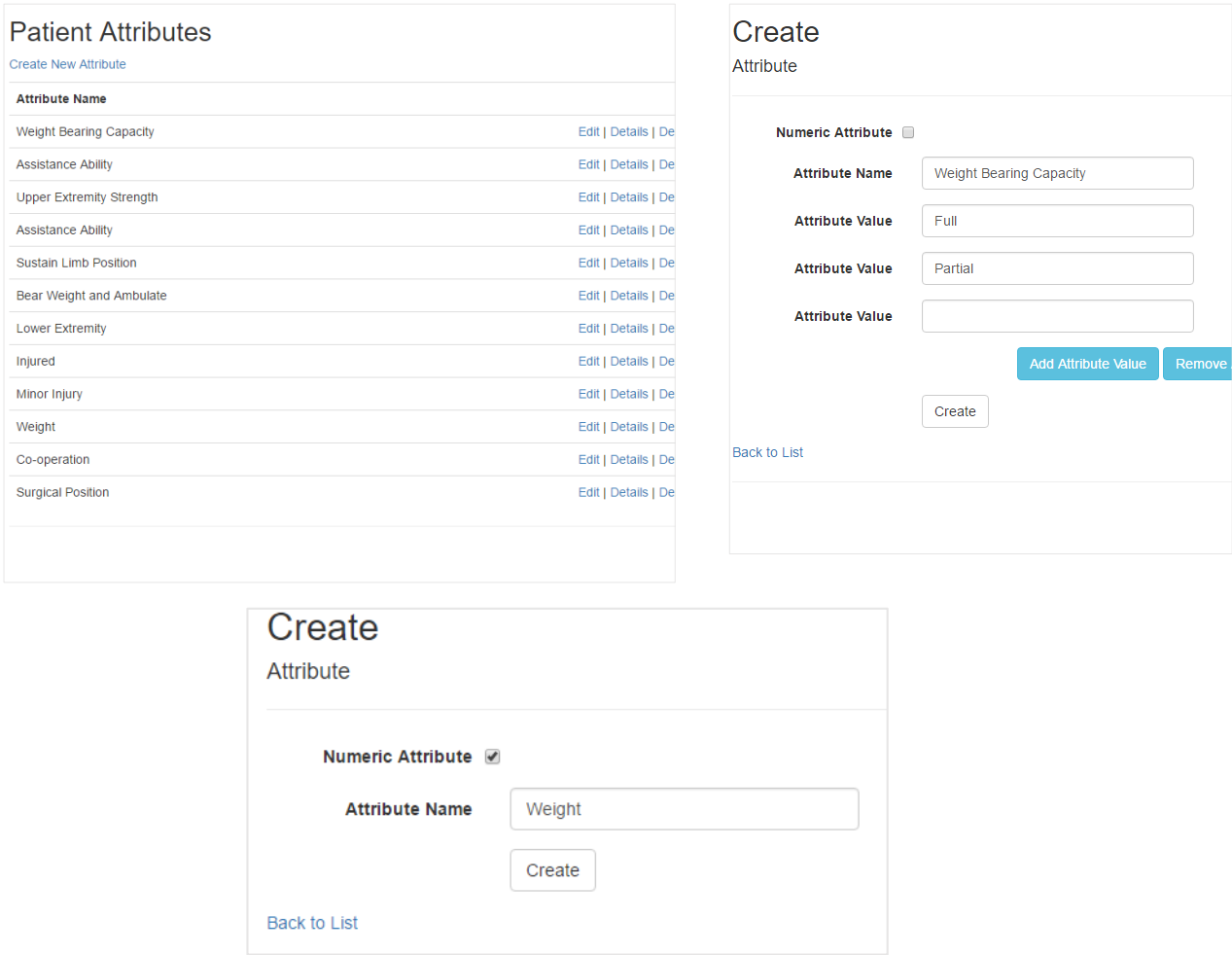


Figure 4.2 – CRUD functionality for attributes

CRUD functionality is also in place for patient attributes. These can have a variable amount of attribute values, or else can be numeric.

4.1.3 Equipment

Equipment

[Create New Equipment](#)

Name	
Ceiling lift	Edit Details
Floor-based lift	Edit Details
Friction-Reducing Device	Edit Details
Seated Positioning Device	Edit Details
Seated Transfer Aid	Edit Details
Mechanical Lateral Transfer Device	Edit Details

Update

Update Current Equipment Values

Name

Floor-based lift

Attribute Name

Available

Attribute Value

Yes

Attribute Name

Extends to Floor

Attribute Value

Yes

Save

[Back to List](#)

Create

Equipment

Name

Ceiling Lift

Attribute Name

Available

Add Attribute Value

Attribute Value Name

Yes

Attribute Value Name

No

Attribute Name

Extends to Floor

Add Attribute Value

Attribute Value Name

Yes

Attribute Value Name

No

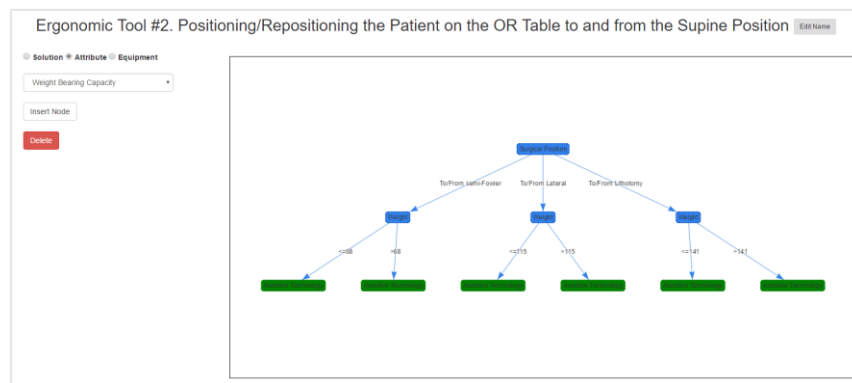
Add Attribute

Create

Figure 4.3 – CRUD functionality for equipment

Similarly, equipment can be created, viewed and deleted. Each piece of equipment can have any number of attributes and attribute values. Editing functionality for the number of attributes and attribute values is not in place as of yet. At the moment the equipment values can be updated for the current facility, e.g. if a piece of equipment was broken, the Available attributes Attribute Value could be set to No.

4.1.4 Trees



Trees

[Create New Tree](#)

Name
Algorithm 1: Transfer to and From: Bed to Chair, Chair to Toilet, Chair to Chair, or Car to Chair
Algorithm 2: Lateral Transfer To and From: Bed to Stretcher, Trolley
Ergonomic Tool #2. Positioning/Repositioning the Patient on the OR Table to and from the Supine Position
Bariatric Algorithm 1: Bariatric Transfer To and From: Bed/Chair, Chair/Toilet, or Chair/Chair

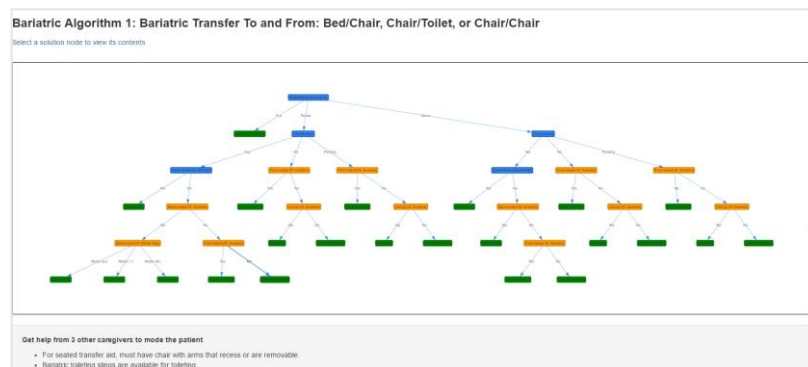


Figure 4.4 – Decision tree CRUD functionality


A more complex CRUD type functionality was created for the trees. These are made up of patient attribute nodes (blue) and equipment nodes (orange), each with edges or branches representing all the possible values that each could have. This can be used to simulate theoretically any possible state that the equipment and patient could be in. For example, a typical path down a tree could lead to a handling plan for the following scenario: The patient in question is uncooperative and weighs over 200 pounds, the stand assist lift is not currently available, and model number 2 of the floor based lift is available. A handling plan with instructions for the patient handler for this scenario can be constructed in advance, and the patient handler could then immediately access this on the system, and then print out the handling plan itself, as will be detailed in the next section.

4.2 Handling Plans

Patient Handling Plans	
Create New Patient	
First Name	Last Name
Test	Patient
Thomas	Grant
Adam	Hamilton
Joan	Metcalf
Joe	Bloggs

Algorithm 1: Transfer to and From: Bed to Chair, Chair to Toilet, Chair to Chair, or C
Handling Plan for Test Patient

Use a ceiling lift



- For seated transfer aid, must have chair with arms that recess or are removable.
- Bariatric toileting slings are available for toileting.
- Bariatric bathing mesh slings are available for bathing.
- Note that a standard porcelain toilet typically has a weight limit of 250 pounds; the patient may need a bariatric commode chair or stool.
- In older lifts, more effort is needed to place the sling under the patient, which may require a minimum of 3 caregivers.

Algorithm 2: Lateral Transfer To and From: Bed to Stretcher, Trolley
Handling Plan for Test Patient

Caregiver assistance not needed; stand by for safety as needed.

Figure 4.5 – Handling plan generation and viewing

A set of handling plans, or else a singular handling plan can be generated by a patient handler, through the system. Each handling plan for a certain patient movement task is generated based on how the patient and equipment data iterates down the tree for that specific movement. The handling plan can be viewed in a printable format, where each handling plan is given its own page.

The handling plans are generated in real time, meaning any changes made to the patient, equipment, or trees will be reflected immediately.

4.3 User Accounts

Register

User

First Name

Last Name

E-mail Address

Password

Confirm password

Role

Administrator ▼

Create

[Back to List](#)

Login

E-mail Address

Password

Login

Users

[Register a new user](#)

First Name	Last Name	E-mail Address	Role
Joseph	Cress	test2@gmail.com	Administrator
Becky	Lynch	test@gmail.com	Administrator
Kenneth	Johnson	testh@gmail.com	Patient Handler

Figure 4.6 – User management sections

There are two types of users, Administrators and Patient Handlers. Administrators have access to the complete application, and all of its functionalities. Patient Handlers have more limited access rights however. They are only allowed to view and edit patient data, and view and print patient handling plans. If they attempt to access other areas of the application, they are redirected to the login screen.

CRUD functionality is also in place for all users, however this can only be done by the administrator.

4.4 Validation

4.4.1 Server Side

Server side validation has been implemented in some parts of the application, however not all. It has been completed for the login and register forms, and also for most use cases when creating a decision tree. It has not been implemented in many other areas of the application, as other development items had a higher priority and were therefore completed first. The design of the application however, has made it so that implementing validation for these forms will not be a difficult task.

4.4.2 Client Side Validation

Client side validation has been implemented in the login and register parts of the application. Because of the specific way that ASP.NET MVC implements client side validation, once server side validation has been added, it is trivial to then add client side validation. This is an important part of the application, as it reduces the load on the server, and also makes the application slightly more responsive.

4.5 Bootstrap

Bootstrap, a CSS and JavaScript framework, was used to design many of the front end elements of the application. Bootstrap contains a series of CSS classes that are applied to HTML elements, making design overall much easier. This greatly helped to save development time, as it was simply a matter of finding a correct class to apply to certain elements, rather than generating the CSS classes by hand.

Another added benefit to Bootstrap is that its features and classes were designed with cross platform support in mind. This means that because Bootstrap was used, many of the application's functionalities can be used without issue on mobile or tablet platforms.

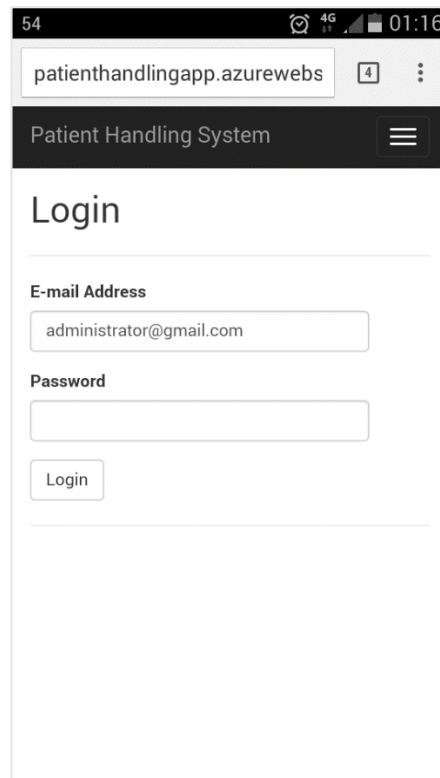


Figure 4.7 - The login screen from a mobile device

Overall, how the application looks and is designed is an important part of the end product, as if the product looks good, users and clients will be more easily convinced that it works well too. This was kept in mind throughout the development.

4.6 Cross Browser Compatibility

While not tested extensively, the site's core functionalities, such as the decision tree editor, work as they are supposed to in Google Chrome, Mozilla Firefox and Internet Explorer 11. It is assumed that there will not be too many major issues if extensive testing was done for these browsers, as the main frameworks used, i.e. jQuery and Bootstrap, both support cross browser compatibility between the above browsers. However this is still something that warrants more extensive testing.

5 Conclusion

This section will attempt to review all of the work done on the project, speculate as to what is next for the application, and provide an overall conclusion.

5.1 Readiness for Production

The majority of the main features required for the application to perform in a production environment have been added. However some exceptions do exist, such as connecting to an e-mail server and other items. These will be discussed in the next section. There is still some work to be done on some of the features added, mainly in the areas of validation and testing for bugs, but for the most part, they function as expected. A significant amount of testing still remains to be done on the application in general however. This could be aided by unit tests, but would mainly consist of integration testing. Once a test plan was developed however, it would not take a significant amount of time to completely test the application – probably a day at most.

5.2 Future Work

There are still a non-trivial amount of outstanding items that remain to be completed on the project. As well as these items, it is likely that if the business users were to view the application and consider implementing it, they would likely have a list of their own features that they would want added before the application was released. Of course, the process could be staggered, by holding off on some of these features for now, and releasing them on future versions of the application.

These outstanding items, some of them mentioned previously, are as follows:

- An e-mail service throughout the application, where patient handlers could reset their password through their e-mail address, or if this was not possible, a password reset functionality for the administrator.
- A data extractor, which would be capable of extracting data from an equipment database, and add this to the equipment tables in the applications database.
- The remaining issues and bugs detailed in the product backlog.

5.3 Overall Conclusion

In general, the main goals defined in the project definition document were fulfilled. Naturally there were also many things which were planned on, that did not quite work out, however, for the most part any major development tasks were seen to completion, which is a reasonably good result.

Much was learned during the course of the project. From the initial research, to delivering a usable final product, almost the full development lifecycle was covered during the course of events. The experience gained during the development of the project will be extremely useful in the future.