Feeding Dashboard

# Aims and Objectives

The main aim of this project is to create an application, with the primary goal of identifying which patients need referring to a dietitian. The Critical Care Unit (CCU) is a branch of the hospital which is underfunded: prioritising those most in need is key. Using Electron as a base framework, combined with Next.js and React.js for the rendering, it is possible to create an application which will run efficiently on all operating systems.

The critical path, detailed in the Project Planning phase, will be continually monitored to prevent any potential delays. This involves identifying key milestones in the project. Said milestones will be used as an easy way to measure how much of the project has been completed: ensuring the project will be completed within the planned timeframes. A risk assessment will be taken to ensure that if issues do arise, there is at least one fallback option that can help to deliver the project on time. This is a time sensitive project which has a definitive end point, finishing behind schedule is not an option: hence the severity of maintaining the critical path. Without this software being delivered on time; patients are at risk of going untreated, and the CCU is forced to use more of its budget to spread themselves thinner. Proactive measures will be taken once if the critical path suffers deviation. As an example, the project will be planned for a 10% leniency in duration. This ensures that if a large, unforeseen issue is discovered, it can be met with accordance without disrupting the deadline. A simple solution for these issues can be in the form of focus-group sessions, with emphasis on diagnosing specific problems, to switching the internal structure allowing for the best use of resources. Following this, the project should have no issue with the due date of 17th April.

Creating an application, capable of running on multiple operating systems, is the first major step. With their limited budget, it is unknown how much can be spent on proprietary licences. Allowing for access to multiple operating systems gives the CCU freedom to expand, or constrict, their technology budget according to their needs, not the needs of the software. Using the same hardware, Virtual Machines will allow for collection of metric data on different operating systems. Examples of metric data collected include Memory Usage and speed at which the Machine Learning algorithm returns a value. The chosen approach for this objective: using a mixture of Electron and Next.js. This will free the project team, letting them focus on the project itself, instead of the different versions traditionally needed. While it would be possible to develop this as a Web Application, given the confidentiality of the CCU, it is important to note the security implications. Restricting this to a desktop application allows entire separation from the internet, widely restricting the possibility of a data breach. The use of Electron as a Multi-Operating-System framework means that there is not much consideration needed to add extra platform support. This boost in efficiency helps to deliver the project on time because code errors are no longer unique to the OS, but project instead.

Given the CCU is dealing with too many patients for them to manage without assistance, it is crucial that the application is capable of accurately predicting those who would benefit from seeing a dietitian. This will be accomplished by allowing users to upload new csv files, containing patient data; then analysing it with the bespoke Machine Learning algorithm. Patients who meet the criteria will be automatically flagged for referral, upon review. It is vital that this algorithm is validated as having a high accuracy, there is no sense delivering an extra feature which will just further complicate the CCU worker. Accuracy can be clearly measured by determining the ratio of perfectly categorised patients to those who were mischaracterised. This objective will be achieved by executing Python code within the Electron application, then outputting the returned python data to graphs for easy viewing.

Creating and exporting graphs and forms of patient data is necessary for continued operations. It allows the workers in the CCU to easily access the data one time, then continue carrying it throughout the day for easy references. Examples of graphs and forms generated include individualised patient overviews, a full list of referrals for the dietitian, and a full list of patients currently admitted. These graphs and forms will also have 3 quantifiable metrics, visual aesthetics, accessibility, and exportability, their importance is paramount. Without the styling having careful consideration, there is a risk of it being too complicated: left unused. To ensure that these are met, different design options will be presented to a small focus group, independent to the development group. This focus group will consist of people from a variety of age brackets and technological ability, ensuring fair accessibility for all users. Options should also be included for different themes, with classic examples being Light and Dark mode, including high contrast mode and options for text size. As previously mentioned, the CCU is spread quite thin. Giving them the ability to export these generated reports would mean they are able to spend less time running back and forth to a computer, and more time spent with the patients who need them. To ensure that this doesn’t cause the projects time constraint to dwindle, the designs shown to the focus group will take place before any development begins. This allows for any revisions to be made before concrete steps have been taken.

# Literature Review

## Methodology

The linear methodology, first presented by Royce (1970), documented how was presumed best to handle Software Development Lifecycles (SDLC) of large-scale applications. His original theory stems from the idea that development of any size can be split into two hypothetical partitions: Analysis, and Coding. It was believed that for small scale projects, it “is in fact all that is required” to maintain successful development (Royce 1970).

If larger scale projects attempt to adopt this small-scale practice, multitudes of risks enter the scenario. One of the biggest is the potential for large scale setbacks. In cases where the product is not what the customer envisioned; projects often must be reset entirely. This can extra cost time, equating to lost money and business reputation (Royce 1970). For the development of an application for the healthcare industry, this cannot be an option. Delayed deployment would result in a continued overwhelmed medical support staff, and patients who are possibly going without treatment they desperately need.

Given larger projects, Royce suggested his linear model. This aimed to split analysis and coding phases into seven sequential and separated steps. Analysis was given two precursors, System Requirements and Software Requirements. Coding phases were also given two additions in the form of Program Design and Testing, the former before and the latter afterwards. These two precursory stages were believed to be the root of any issues in the final product (Royce 1970).

It wasn’t until Bell and Thayer (1976) questioned this principle that the methodology was coined “Waterfall”, largely based on its theoretical design structure. Their analysis of this methodology under the microscope proved Royce’s initial theory correct, most errors in the SDLC could be attributed to a failure to understand the Requirements. These failures could commonly be defined by subcategories: missing, incomplete, inadequate, or unclear. (Bell and Thayer 1976)

Waterfall is not without its pitfalls. During early stages in the history of Software engineering, the methodology practices were very similar to that of standard engineering; namely the practice of front-loading design stages to negate time from the development stage. As discussed, the Waterfall method, designed by Royce, attributes a heavy weight on the initial Requirements to the final product’s success. This ignores a crucial element of Software Development that the environment and requirements can continually change.

One of the biggest challenges faced when approaching development with Waterfall was the progressive timeline. Given the severity of the level of planning which must go into a project, when mistakes are made, and phases must be reset, lots of time becomes wasted. While this does provide a slight betterment over the initial Analysis and Coding scheme laid out by Royce, waterfall can still involve being sent back to the previous core-stage of functional development. This emphasised the effects of changes in environment and dynamic requirements of the application. Given waterfalls linear nature, if requirements are to change: the entire project must often be redesigned from the origin point (Royce 1970).

Many mythologies were derived from this initial framework, of which, most sought to fix oversights in how the project viewed Requirements. These were all radicalised and had no real over-arching agreements on how successful Software Engineering projects should exist. This changed in 2001 when a group, including seventeen of the most progressive project planners for software engineering, collected in Utah for a summit to discuss a set of ground rules referred to as the Manifesto for Agile Software Development (Highsmith 2001). These were to be the proceeding best practices to deliver customer orientated light weight project methodologies.

The mantra for The Agile Alliance, signed by all seventeen attendees:

**Individuals and Interactions** over processes and tools,

**Working software over** comprehensive documentation,

**Customer collaboration** over contract negotiation,

**Responding to change** over following a plan.

(Highsmith and Fowler 2001)

Loosely interpreted, the principles define how projects must: continually involve the critical end user, prioritise simplicity and most importantly a low-scale timeline. This aimed to increase development teams focus on the end user and increase flexibility over the pre-existing accepted models’ emphasis on static Requirements (McQuade, Moore and Hunter 2019).

Analysing the effectiveness of these practices came later, at the expense of Dayton and Barnum (2009). Completion of surveys given to the development team and analysis of the product, made researchers able to determine many key factors in the relevance of different principles laid out in the initial Manifesto (Highsmith and Fowler 2001). The primary focus of these surveys was to determine the effect of User Centred Design on the final product and employee attitudes and opinions were to the newly adopted Agile methodology.

They found the biggest stand out of the Agile methodology was the use of end-user collaboration. This often led to developers feeling more engaged with their projects and result in more user-centric design. Employees felt more connected to the end-user through formal collaboration, leading to more nuanced and intuitive solutions for the design. It was also observed that employees’ attitudes to the projects would often shift, more positively, after formal user testing. The physical witnessing of a user attempting their application greatly motivated developers, often making them work much more vigorously as they could better understand challenges associated with the current design (Dayton and Barnum 2009).

Without the formal user testing, employees regularly felt like management would fake feedback. to tweak the application without any pushback from the development team: as this feedback would often be a direct change to the existing understood project requirements. Witnessing these events allowed for real connection to the issues, instead of reading the reports in an email. These findings of Dayton and Barnum (2009) directly correlate with the Manifesto published by Highsmith and Fowler (2001), who stated that “the most effective method of conveying information within the development team is face-to-face”.

This makes the agile methodology a perfectly suited application to the project at hand, developing an application for the CCU to assist in referring patients to a dietitian. Its lightweight nature will allow the project to move at a fast pace, helping to deliver the project within the timescale. The end-client orientation of the methodology will also help to ensure the project delivered will be easily accessible to the members of the health staff, who might not be technologically advanced.

## Existing Multi-Platform Solutions

There are a few ways to attack the problem of multi-platform applications. Given different system requirements, they often require vastly different code to create the same application. This form of development would not be acceptable for Agile development. The process would add extended timelines, resulting in three undeniably different projects. To maintain accordance with the principles of Agile, saving time where possible, a single code-base and project would be preferable. This should also help to reduce operational costs, as there are less end-projects to manage.

Many of the biggest application development companies employ this same methodology, using a singular codebase stretched across multiple operating systems. This can be accomplished through the use of frameworks, like Electron. Developed by GitHub; this provides developers a way of creating desktop applications using standard web frameworks. This was originally developed solely for their custom code editor, Atom.

When GitHub decided to release this framework as an Open-Source project, many developers jumped on the opportunity (Electron, 2024). The clear advantages of not having to split focus between multiple projects far outweigh the lessened customisability for systems seen in individualised projects. Not only freeing expenses on development costs but also operational costs. Two of the biggest names who’ve adopted this framework are Discord and Skype, two rivalling communication applications.

By choosing to use this framework, the development team for the CCU application can focus their time on progressing new features and ensuring a high level of accessibility, via continual end-user collaboration. Conversely, choosing to develop three individual applications, without the use of a framework like electron, will result in a protracted timeline three-fold and the potential for unequal applications: leading to confusion in the workers using different versions. This would undercut the main aim for this project, to provide an easy-to-use application for the use of medical staff to aid in dietitian referral.

## Tools

### Multi-Platform and Interfacing

The chosen Framework for managing the cross-platform support is Electron. As stated, this will allow developers to focus on the important elements of the project requirements, without delegating extra time to ensuring compatibility on multiple platforms. Electrons heavy documentation show possibilities of using Front-End web libraries like React and Next. These will be the User Interface (UI) Frameworks chosen for this project.

Frameworks, such as these, have been widely accepted as the best options for UI development in recent years, with additional libraries like Tailwind providing easy uniformity and customisation. Preloaded with tons of configuration options, these UI frameworks allow for quick development in accordance with the agile methodology: delivering the project faster to the CC, helping to expediate necessary referrals.

### Machine Learning

At the core of this application lies a Machine Learning algorithm used for binary classification, meaning to classify between “is” and “isn’t”, “True”, or “False”: In need of referral, or not in need of referral. Many options for effective binary classification exist, the most relevant for the type of data processed are Search Vector Machine and Ensemble learning methods. These both offer relatively light weight solutions to binary classification and are idealistic under different scenarios.

A comparison of these two models has previously been conducted (Hornblower 2024), highlighting the benefits of an Adaptive Boosted ensembles learning method. In a diagnostic setting such as this, where the number of patients in need missed are of the highest importance: an adaptive boosted model typically preforms much better. The aggregation of multiple weak-learners allows the final prediction to be taken at a vote, aiming to provide a more comprehensive prediction based on more discovered relationships between patient data.

The choice of library for this functionality will be SKLearn. They offer a multitude of excellent python functions necessary for machine learning. Given these premade classifiers, it will be much easier to keep in line with Agile development and stay swift in moving through different phases.

# Requirements

## Functional

A key functionality of the proposed healthcare management system is the ability to display a detailed list of all patients within the system. There should also be options allowing users to filter the results and display who is most in need of seeing a specialised dietitian. This feature is designed to help streamline the patient management of the Critical Care Unit, allowing for better delegation of resources.

The application must also be capable of presenting a comprehensive overview of individual patient data. This helps to give doctors, working in the ward, data in a more accessible way which can be analysed quicker. Aims of presenting individual patients in this manor are largely for quality of life for medical staff, including shorter wait times for patients given their data should be easily printable and ready for transport inside of the CCU.

Without the ability to analyse the patients’ records, the application serves no more purpose than a standard patient record viewing tool. This is where the Machine Learning algorithm takes physiological measurements of patients within the Critical Care Unit, flagging them if they should be referred to a specialist. This step needs to happen before the Operator can filter results, otherwise no new records would be processed for referral.

Operators must be able to upload new patient records for analysis under the new Machine Learning model. The format accepted should be CSV, allowing for easy processing inside the algorithm. Giving Operators the option to add new CSV files means new admissions to the ward can be easily analysed without any technical knowledge of the system.

Operators must also be able to generate graphs overviewing all patients inside the CCU, with the ability to export and save them when needed. This will assist the medical staff by allowing for instant bulk analysis, gaining an overall understanding of the patients in the ward’s care. This can help to plan what the remaining of the day might look like for the medical staff, expediating the process of referring for diagnosis by allowing for task prioritisation.

One of the most fundamental requirements is that the application must run identically on different operating systems. This mitigates the need for multiple training sessions and promotes uniformity. Additionally, it is understood that the budget may shift in the future: forcing the CCU to adopt different operating systems. Ensuing the program can run on all systems allows the CCU this freedom, meaning their budget will not limit the success of the application.

## Non-Functional

Producing an efficient design capable of running on minimal hardware and any operating system, using the same codebase, is paramount in allowing the CCU’s budget to stay as flexible as possible. Additionally, reducing three individual projects to one shared project helps to minimise the amount of maintenance a project will require post release: reducing financial strain on the CCU. If a bug is discovered, it only needs to be fixed in one place: not multiple.

Not every staff member of the CCU will be technologically advanced. Given this, the solution’s design must be intuitive and minimalistic. Simplicity is a key aspect associated with a successful agile delivery, as stated by the Manifesto (Highsmith and Fowler 2001), so this aligns perfectly with the goals of the project.

With the confidential nature of medical documents, the completed application must have some form of security in place. A simple option would be a pass key that has been encrypted. This should be able to be changed, but never be able to be decrypted. This is designed to protect the workstations from potential security threats. Another option is to setup a custom account for each practitioner in the ward, but this introduces complications with people forgetting passwords. Security protocols should also be introduced, forcing the change of a passcode every week.

Different viewing options must be made available to the users of this application, ranging from contrasting colours to resizable text. This is in the effort of assisting those with impaired vision, making the application more widely accessible to workers in the CCU.

Analysis of patients should not take longer than 5 minutes. Time is critical in the CCU, and the less time spent waiting for a computer to analyse the better. Longer analysis times can also create frustration in the staff, leading to a negatively impacted workplace culture. This toxic workplace culture would directly negate an aim of the project: making the CCU flow far smoother and lessen stress on the staff.

The model must have a high recall score, the metric used to calculate the correctness of the model’s predictions. If the recall score is low, this means more positive predictions were incorrect. This is valuable information, as keeping referrals decongested is a key element in the project’s goals. With a higher recall score meaning lower congestion, a good aim for this is a recall ability of 70%. End results of the model should mean no less than 70% of referrals made are valid patients in need of seeing a specialist dietitian.

# Project Plan

Gnatt Chart

Breakdown of tasks

Risk Register

# References

Dayton, D.D. and Barnum, C.B. (2009) The Impact of Agile on User-centered Design: Two Surveys Tell the Story. Society For Technical Communication [online]. 56 (3), pp. 219-234. [Accessed 13 April 2024].

McQuade, M.R.M., Hunter, A.P.H. and Moore, S.M. (2019) Appendix A: How Agile Software Development Came to Enable Adaptable Systems (2019) Acquisition of Software–defined Hardware–based Adaptable Systems [online]. Washington, Dc: Center for Strategic and International Studies, pp. 31-36 [Accessed 13 April 2024].

Royce, W.W.R. (1970) Managing the Development of large Software Systems. ***Ninth International Conference on Software Engineering.***Los Angeles, March 1987. Washington D.C: IEEE Computer Society Press, pp. 328-338

Bell, T.E.B and Thayer, T.A.T. (1976) Software Requirements: Are they really a problem?. ***Second International Conference on Software Engineering.*** San Francisco, CA, October 13-15, 1976. Washington D.C: IEEE Computer Society Press, pp. 61-68

Highsmith, J.H and Fowler, M.F. (2001) ***Manifesto for Agile Software Development***[online], Utah, 11-13 February 2001. The Agile Alliance. Available from: <https://agilemanifesto.org/> [Accessed 14 April 2024]

Highsmith, J.H (2001) History. ***The Agile Manifesto*** [blog]. 2001 Available from: <https://agilemanifesto.org/history.html> [Accessed 14 April 2024]

Electron: Showcase [online] (2024) Available From: <https://www.electronjs.org/apps> [Accessed 14th April 2024]

Hornblower, C.H. (2024) ***Machine Learning: A Comparison of Two Models*** [online].BSc, University of the West of England. Available from: <https://github.com/Sp0xF8/Individual-Project/blob/main/22019799.ipynb> [Accessed 14th April 2024]