“RepairMart”

A dissertation submitted in partial fulfilment of

The requirement for the degree of

MASTER OF SCIENCE in Software Development

in

The Queen’s University of Belfast

By

Ciaran Neeson

04-May-2025

Contents

[Acknowledgements 3](#_Toc194569150)

[Abstract 1](#_Toc194569151)

[Introduction 2](#_Toc194569152)

[Understanding the Problem 4](#_Toc194569153)

[User Interface Design 8](#_Toc194569154)

[**Design Process** 8](#_Toc194569155)

[Architecture Design & Algorithm Explanation 9](#_Toc194569156)

[**Database** 9](#_Toc194569157)

[**UI Development** 9](#_Toc194569158)

[Testing 10](#_Toc194569159)

[Evaluation and Conclusion 11](#_Toc194569160)

[References/Bibliography 12](#_Toc194569161)

[Appendices 13](#_Toc194569162)

# Acknowledgements

I would like to thank my supervisor, Olga Pishchukhina.

# Abstract

# Introduction

An outline of this project’s structure is below, giving the key components of each section.

**Understanding the Problem**

This section outlines the problem area and motivation for the current project, highlighting the current solution gap and contrasting it with analogous solutions in other areas. Following this, the software choices used for developing this project are outlined, with justification of the choices made.

**User interface design**

In this section, existing user interfaces are analysed and used to inform the design for the project. Rationale is provided for the current project’s interface design choices, and an overview of the interface is shown.

**Architecture design**

This section gives a high-level overview of the project’s solution files, describing the project structure and how the system’s front-end and back-end components are connected.

**Testing**

This section describes the project’s testing methodology, with examples of manual testing....

**Evaluation and Conclusion**

This section assesses the degree to which the final developed solution meets the outlined requirements, and discusses areas for potential future improvements and developments.

# Understanding the Problem

The student must clearly describe the perceived problem and the target audience. It should be obvious from the chapter that the student has a thorough understanding of the problem domain and current applications used (if available) to address the problem at present. This section should demonstrate a good understanding of possible languages, libraries and frameworks that could be used to develop the project. This chapter should explain and justify the process by which the requirements have been elicited. It should then also clearly identify the requirements of your project, which can be later tested. Depending on the chosen development strategy this chapter may be written retrospectively. The student’s academic supervisor will provide additional advice on this where required. The development strategy must be clearly described, adequately justified in terms of the problem and appropriate to the project.

**Problem Specification**

Waste from electronic consumer goods (e-waste) is a big problem in today’s world (World Health Organization, 2024). Consumers frequently dispose of their electronic products in an inefficient manner when they no longer function correctly, while in reality these products may be repairable and their useful life may be extendable. Market conditions mean that it is frequently cheaper to buy a brand new product and throw away a broken one, creating excess waste and undesirable secondary effects (in addition to causing people to spend more than they need to). The current consumer economy is primarily oriented towards profit, and this largely excludes easily repairing faulty items: although ‘Right to Repair’ laws have been introduced in the UK and the EU in July 2021 (Conway, 2021) and April 2024 (García Molyneux & Oberschelp de Meneses, 2024), respectively, the new legislation is limited in scope, and in duration (European Commission, 2023). Furthermore, when seeking repairs, consumers are directed towards a single point of contact (the manufacturer or vendor of their non-functioning product), which limits their options in terms of budget, location, viability, etc.

The aim of this project is to develop a repair ‘marketplace’ (an ‘eBay’ equivalent for electronics repairs) where consumers can create a request to repair a non-functioning product, uploading details so that their request is visible to multiple repair specialists (e.g. large manufacturers, small businesses or independent individuals with the relevant expertise). The specialists will be able to review requests, ask any relevant questions for clarification on the product and/or its defects, and submit a quote for repair to the consumer. Consumers should then be able to review a list of quotes received and choose the most suitable option for them – based on price, location, estimated turnaround, etc. – making the process of repairing their product easier and cheaper, and resulting in the extension of their product’s useful life, a reduction in electronic waste, and an overall cultural shift away from throwaway consumerism.

**Current options**

When dealing with electronic goods that no longer work correctly, consumers can either (a) dispose of them as waste, (b) potentially try to re-sell a second-hand item for spare parts, or (c) try to repair the item.

**Waste**

There are numerous websites giving recycling instructions and allowing consumers to find their nearest recycling points. A number of problems exist, though:

* these websites are not well-publicised: many people do not know about their existence
* they often have limited (geographical) scope, serving local communities/council areas while lacking coordination across different localities/jurisdictions
* the lack of a ‘recycling culture’ among some demographics can be difficult to overcome
* the lack of incentives for recycling (and of penalties for failing to properly dispose of electronic goods) causes unnecessary and avoidable waste

**Resale**

Consumers can try to re-sell their faulty/broken goods, often using online marketplaces such as eBay, however the return on doing so is often negligible and may not compensate for having to deal with things like non-standard postage & packaging (as well as handling potential refunds/returns for unsatisfied customers).

**Repair**

Repairs have become somewhat less prohibitive for certain classes of electronic goods thanks to the aforementioned new legislation, however the cost is often still prohibitively expensive since customers typically only deal with a very small number of potential experts, only receiving one or two quotes (often from the items’ manufacturers, who tend to charge significantly more for their services and also prohibit competition by voiding product warranties if consumers seek the same service outside of a manufacturer-approved network (Brannon, 2024)).

**Solution**

Creating a centralised marketplace where consumers can submit requests for repair can solve many of the above-mentioned problems, since the onus can be placed on repair specialists to offer their services in a more transparent manner, lowering prices for consumers and creating a greater incentive to avoid electronic waste through extending the useful life of products (in contrast to the lack of incentive that currently exists, failing to discourage the common consumer ‘throwaway’ reflex). Such a solution, if developed correctly, could also be pitched to government/local authorities who may be interested in lowering the burden on them for waste disposal and helping to reduce the carbon footprint inherent in the current ‘throwaway’ economy (from manufacturing new products and importing them from far-away locations).

The solution to this problem is envisaged as an online platform containing some elements which can be found in websites for purchasing consumer electronics (e.g. [amazon.co.uk](https://www.amazon.co.uk/)) and for the resale of used goods (e.g. [ebay.co.uk](https://www.ebay.co.uk), [gumtree.com](https://www.gumtree.com/), [donedeal.ie](https://www.donedeal.ie), etc.).

While there are websites currently offering similar services (e.g. [https://uk.electronic.partners](https://uk.electronic.partners/)), these only offer contact with a single repair specialist (or single organisation) at a time: they do not centralise a range of repair options in the manner proposed for this project.

The advantage of a centralised platform to consumers is that instead of having to send multiple enquiries about the desired repair to multiple specialists – all via different channels, potentially having to register/create an account on multiple platforms, and without necessarily ever receiving a response to an inquiry – the proposed solution will provide one central marketplace and point of contact through which a consumer may register once and be contacted by multiple repair specialists, making the selection process much easier while also providing transparency over order management, status updates, feedback/customer satisfaction, etc.

The advantages to suppliers (i.e. repair specialists) include the ability to reach a wider number of customers, and the removal of a potential hurdle in no longer needing to set up and maintain their own proprietary website for handling repair enquiries and orders.

## **Users/Stakeholders**

The target user audience will consist of:

1. regular consumers who prefer to repair their electronic goods at a reasonable cost, with the intention to (a) avoid having to buy a replacement item, and (b) reduce/avoid the waste associated with the disposal of electronic items
2. electronics repairs specialists who wish to offer their services to a wider market, providing their services through the platform

Other potential interested parties (while not being direct users) may be government departments and/or local authorities who wish to promote a greater culture of re-using/refurbishing/recycling electronic goods, to reduce the cost and burden placed on them when it comes to the disposal of such goods.

## **Requirements elicitation**

Requirements have been gathered by investigating some well-known existing websites:

1. electronic goods retailers
2. marketplaces for second-hand goods
3. electronics repair specialists

The proposed solution will contain selected elements from each of the above categories, adapted for our proposed solution.

|  |  |
| --- | --- |
| **Website category** | **Features** |
| Electronic goods retailers (e.g. Amazon) | Account creation/management  Search/filter according to keyword, manufacturer, product type |
| Second-hand goods marketplaces (e.g. eBay, Gumtree) | Account creation/management  Account type differentiation (seller/buyer)  Search/filter according to keyword, manufacturer, product type  Auction system allowing bids from customers  Listing creation & management (incl. image/video upload)  System-generated email notifications (email verification, confirmation of different user actions, status updates, etc.)  Feedback system attached to user profiles  User-to-user messaging (e.g. for queries on listings) |
| Electronics repair specialists | Account creation/management  Assessment of product’s defects and receipt of quote with the option to accept/decline |

## **Adaptation to the proposed solution**

The below table gives greater detail on the features outlined above, and how these will be incorporated into the proposed solution.

|  |  |
| --- | --- |
| **Functionality group** | **Details** |
| User registration | Registration via email confirmation after providing user details (min. requirements: email address & password)  Password reset functionality |
| Account management | Account setup – addition of account type (consumer/repair specialist) & name/location details (these will be required for in-person collection/drop-off of items and/or postal information delivery  Ability to update account details (password, name, location, and potentially also email address) |
| Listing creation & management | Consumers |
| Search/review listings | Searching & sorting listings on the platform using keywords, with the ability to filter results by location, date (age of listing), product type, manufacturer |
| Repair quote creation & management | Repair Specialists |
|  |  |

In addition to the above, and as with most standard web applications in use today, we will aim to permit user registration via email, with password reset functionality.

## **Languages, libraries & frameworks**

In order to choose the right development framework for the proposed solution, a number of options have been considered. The table below contains details of these.

| **Option** | **Advantages** | **Disadvantages** | **Examples** |
| --- | --- | --- | --- |
| **React (JavaScript)**  Front-end library developed by Facebook for building user interfaces, particularly single-page applications.  It uses a component-based architecture and a virtual DOM to optimize rendering. | **Component Reusability**: Promotes code reuse and modularity.  **High Performance**: Virtual DOM ensures efficient updates.  **Strong Ecosystem**: Extensive libraries and tools. | **SEO Challenges**: Requires server-side rendering for better SEO.  **Learning Curve**: Requires understanding of JSX and modern JavaScript. | **Facebook** [[1]](https://www.intelivita.com/blog/popular-websites-built-with-react/)  **Airbnb** [[2]](https://www.cmarix.com/blog/most-popular-websites-built-with-react/)  **Instagram** [[1]](https://www.intelivita.com/blog/popular-websites-built-with-react/) |
| **Angular (TypeScript)**  Comprehensive front-end framework developed by Google. It uses TypeScript and provides a full suite of tools for building dynamic web applications. | **Two-Way Data Binding**: Simplifies synchronization between model and view.  **Comprehensive Toolset**: Includes everything needed for front-end development.  **Strong Community Support**: Backed by Google. | **Complexity**: Can be overwhelming for small projects.  **Performance Issues**: Can be slower compared to other frameworks for large applications. | **Gmail** [[3]](https://seclgroup.com/10-best-examples-of-websites-and-apps-built-with-angular/)  **Upwork** [[3]](https://seclgroup.com/10-best-examples-of-websites-and-apps-built-with-angular/)  **PayPal** [[3]](https://seclgroup.com/10-best-examples-of-websites-and-apps-built-with-angular/) |
| **Vue.js (JavaScript)**  Progressive front-end framework that can be integrated incrementally into projects. It is known for its simplicity and flexibility. | **Easy to Learn**: Simple syntax and structure.  **High Performance**: Lightweight and fast.  **Flexibility**: Can be used for both small and large projects. | **Smaller Community**: Compared to React and Angular.  **Limited Enterprise Adoption**: Less common in large-scale enterprise projects. | **Alibaba** [[4]](https://trio.dev/websites-using-vue/).  **Xiaomi** [[4]](https://trio.dev/websites-using-vue/).  **Behance** [[5]](https://vuejsfeed.com/blog/8-popular-websites-that-use-the-vue-js-framework). |
| **Django (Python)**  Django is a high-level back-end framework for building web applications quickly and with clean, pragmatic design. It follows the "batteries-included" philosophy. | **Rapid Development**: Built-in admin panel and ORM.  **Security**: Comes with built-in security features.  **Scalability**: Suitable for both small and large applications. | **Monolithic Framework**: Less flexibility compared to microservices architecture.  **Learning Curve**: Requires knowledge of Python and Django’s conventions. | **Instagram** [[6]](https://codment.com/famous-websites-built-with-the-django-framework/).  **Pinterest** [[6]](https://codment.com/famous-websites-built-with-the-django-framework/).  **Disqus** [[7]](https://djangostars.com/blog/10-popular-sites-made-on-django/). |
| **Ruby on Rails (Ruby)**  Ruby on Rails is a full-stack web application framework that emphasizes convention over configuration. It is known for its developer-friendly syntax and rapid development capabilities. | **Convention Over Configuration**: Reduces the number of decisions developers need to make.  **Rapid Development**: Scaffolding and built-in tools.  **Strong Community**: Extensive libraries and plugins. | **Performance**: Can be slower compared to other frameworks.  **Scalability Issues**: May require additional optimization for large-scale applications. | **Shopify** [[8]](https://sumatosoft.com/blog/top-20-websites-built-on-ruby-on-rails).  **GitHub** [[9]](https://prograils.com/top-10-famous-sites-built-with-ruby-on-rails).  **Airbnb** [[9]](https://prograils.com/top-10-famous-sites-built-with-ruby-on-rails). |
| **Laravel (PHP)**  PHP framework that follows the Model-View-Controller (MVC) architectural pattern. It is designed to streamline web development by providing a structured and expressive syntax. | **Elegant Syntax and Expressive Code**: Enhances productivity and reduces the learning curve.  **MVC Architecture**: Improves code organization, maintainability, and scalability.  **Built-In Authentication and Authorisation**: Simplifies user registration, login, and role-based access control.  **Eloquent ORM**: Simplifies database interactions with an intuitive syntax.  **Robust Security**: Protects against common vulnerabilities such as SQL injection, XSS, and CSRF.  **Scalability**: Suitable for both small and large-scale projects.  **Community and Ecosystem**: Large and active community with extensive documentation and third-party packages.  **Cost-Effective**: Pre-built components and tools reduce development time and costs. | **Performance Overhead**: The middleware pipeline and various components can slow down execution speed compared to leaner, micro-frameworks or custom-built solutions[[2]](https://courseya.com/disadvantages-of-laravel/).High-traffic applications may require extensive optimization to achieve optimal performance.  **Frequent Updates:** Laravel's active development community leads to frequent updates. While updates are essential for security and new features, they can pose challenges for projects that prioritize stability and continuity[[1]](https://www.sinelogix.com/laravel-disadvantages/).  **Scalability Challenges**: While Laravel can handle large-scale applications, it may require additional optimization and architectural considerations to scale effectively[[1]](https://www.sinelogix.com/laravel-disadvantages/). | **Invoice Ninja**: An open-source invoicing solution[[10]](https://infostride.com/sites-built-with-laravel/).  **Alison**: An e-learning platform offering online courses[[10]](https://infostride.com/sites-built-with-laravel/).  **Neighborhood Lender**: A financial services platform[[11]](https://codetheorem.co/blogs/websites-built-with-laravel-framework/). |

From the above, Laravel/PHP stands out as a strong choice for the development of the proposed solution. The key advantages which stand out in the context of this project are:

* built-in authentication and role-based access control (since the proposed app will behave differently for consumers vs. for repair specialists)
* comprehensive documentation: this will be vital in aiding developer understanding while creating the solution
* scalability: high-traffic considerations will not be an immediate concern for this project, but if the final product is successful it may lead to higher user numbers and roll-out to a larger audience
* MVC architecture will allow us to build a well-organised code base easily favouring modularity, maintainability and extensibility

### **Advantages Over Other Frameworks**

|  |  |
| --- | --- |
| **vs. React/Angular/Vue** | Laravel provides a full-stack solution with built-in backend capabilities, whereas React, Angular, and Vue are primarily frontend frameworks that require additional backend setup. |
| **vs. Django/Ruby on Rails** | Laravel’s syntax and structure are often considered more intuitive and easier to learn. Additionally, PHP is widely supported across various hosting environments, making deployment easier. |
| **vs. Symfony/CodeIgniter** | Laravel offers a more comprehensive set of features out-of-the-box, including built-in authentication, Eloquent ORM, and a powerful templating engine (Blade). |

The disadvantages of Laravel (outlined above) are not significant enough to deter us from using this framework. The performance overhead is not a major concern for this kind of website and is a minor concern, while any potential scalability issues would only occur if the developed solution becomes very popular and widely-adopted (which would likely take a long time) – even if this were to happen in the future, it should be welcomed as an opportunity for optimisation rather than being viewed as a constraint. Similarly, new versions of Laravel should be welcomed rather than avoided and should not deter us from using this framework.

# User Interface Design

User interface (UI) design is paramount for the success of any online platform, and particularly crucial for a website like an online marketplace. In today's digital landscape, characterised by an ever-expanding ocean of online content and services, capturing and maintaining user attention is a significant challenge. A well-crafted UI is no longer a mere aesthetic choice; it's a critical factor directly impacting user engagement, conversion rates, and ultimately, the platform's overall success. For an online marketplace specialising in the repair of consumer electronics, a seamless and intuitive UI is especially vital, as it must effectively guide users through the process of describing repair needs, and selecting an appropriate service provider. Key design principles to be considered include intuitive navigation, clear visual hierarchy, consistent branding, and accessible design to accommodate diverse user needs. The sections below detail the specific UI design choices implemented in this project which provide a pleasant and effective user experience.

## **Design Process**

### **Prototyping**

In the first instance, [Figma](https://www.figma.com/) was used as a tool to create mock-ups the different pages that make up the application (see [Appendix 1](#_Appendix_1:_Original)). Figma uses vector-based design and its browser-based accessibility eliminates the need for cumbersome software installations. It also offers robust version history and prototyping capabilities, facilitating seamless iteration and efficient design validation.

The original designs allowed features and processes to be identified, aiding the construction of a data model for the application. The final UI design varies significantly from the Figma mock-ups, however: the mock-ups were deemed to be less optimal since they only loosely relied on HTML/CSS design standards while suggesting the need for unnecessary additions (e.g. implying nested window management where this was not required). User feedback along the way also favoured a more simplistic font choice and colour scheme (this is discussed further below).

### **Investigation/examination of similar solutions**

The [requirements elicitation](#_Requirements_elicitation) section mentions some of the key functionality/features from existing websites operating in a similar domain that this project aimed to incorporate (with some adaptation).

Some long-established and extremely popular websites from adjacent domains (e.g. eBay, Amazon) were examined, however the overhead in creating a ‘clone’ of these websites – which have been established for over 2 decades, and are supported by huge budgets and teams of developers – would have been prohibitive to the completion of this project.

A more realistic target was deemed to be emulating some of the features of a modern, more simplistic marketplace-style website such as the below examples:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure 1: [gumtree.com](https://www.gumtree.com/for-sale) listings page | Figure 2: [donedeal.ie](https://www.donedeal.ie/all) listings page | Figure 3: [marktplaats.nl](https://www.marktplaats.nl/) listings page |

These marketplace websites offer most of the required features identified in the [requirements elicitation](#_Requirements_elicitation) section, using simple card-based UI design without complicated animations, overly-elaborate colour schemes and other features that more mature websites with massive user bases have been able to incorporate. They all include an upper navbar that is visible throughout the user experience, with relevant search filters on the left-hand side of the screen where appropriate, and greater detail displayed when individual listed items are selected. Individual listings pages contain many options, and not all of these are incorporated into the RepairMart site (for example, sharing a listing via WhatsApp, Twitter, etc. and showing breadcrumb navigation were not deemed to be key requirements, and user feedback during the development process never requested these functionalities – although they are possible future developments that could be incorporated). The design also took into account the need to avoid having too many buttons/options on screen at one time (Thorne, 2021).

### **Final Design**

Having considered the above, this project has been designed in a more simplistic manner using card-based presentations with paginated results and a basic colour scheme which presents the necessary information clearly, concisely and intuitively. The [Ubuntu font](https://design.ubuntu.com/font/) has been chosen for use, driven by a positive experience of using the [Ubuntu operating system](https://ubuntu.com/): it is a modern, sharp typeface which has advantages over other options in terms of clarity (Rodriguez, 2019) and has been made available as open source by the creator (Beckwith, 2023). The site’s main colour palette has been generated by [coolors.co](https://coolors.co/), and an orange-based option has been chosen for its attention-grabbing quality (McLean, 2023) and its association with affordability (Raj, 2024).

The UI design process also involved gathering feedback from a selection of volunteers along the way. Feedback was given on existing developments/layouts presented to potential users (and was therefore reactive rather than prescriptive). Comments were gathered on the site’s user friendliness, intuitiveness and overall appeal and suggestions were, within reason, incorporated – although not everything compromise was needed due to technical and time-based constraints. Compromise is, however, a normal part of the design process (Craig, n.d.).

|  |  |
| --- | --- |
|  |  |
| Figure 4: RepairMart logo & landing page | Figure 5: RepairMart signup/login pages and user dashboard |

The landing page uses a [hero slider](https://www.npmjs.com/package/hero-slider) presentation with slides to signal the site’s purpose to the 2 user categories (customers and repair specialists), and relevant links to sign up/create a new listing. The slider contains a minimalist diagram showing the use of a soldering iron to repair a circuit board – capturing the overall idea of the site. A logo for the RepairMart site generated by the Opera browser’s [Aria AI](https://www.opera.com/features/browser-ai) has been incorporated into the site’s navbar, and also contains a basic representation of the site’s purpose in the form of various electronics items. Both the slider image and the site’s logo complement the site’s overall colour scheme and visual design.

The navbar contains Login and Signup links (for users browsing as guests) and functional links in a dropdown menu (see Figure 5, top-right) for logged-in users. The ‘Search Listings’ and ‘Add new Listing’ links remain in the navbar for all users (although a user must first create an account and assign the ‘customer’ role to themselves before being allowed to proceed with creating a new listing).

|  |  |
| --- | --- |
|  |  |
| Figure 6: RepairMart sign-in with Google / password reset form | Figure 7: RepairMart account management |

The signup page in Figure 5 allows users to create an account with the site using their Google or Facebook account, and sign in to the site using one of those accounts (see Figure 6, top). Upon creating an account via one of these methods, the user will be directed to their account management page where they need to choose at least one role for themselves (customer/repair specialist) and enter address details before they are allowed to create a listing or interact with other listings in the site.

|  |  |
| --- | --- |
|  |  |
| Figure 8: RepairMart search results/watchlist view | Figure 9: Individual listing details |

As shown in Figure 4, the lower part of the landing page also contains a display of recent listings posted in the app, and a search form which takes users to the search results page (Figure 8).

Clicking on a listing card, either from the landing page or from the search results page, takes users to the individual listing details (Figure 9) where they can see full details of the listing: title, description, published date, expiry date, manufacturer, customer budget/currency, product categories (of which up to 3 are allowed per listing), a link to the customer’s profile page, and role-dependent action buttons: here, customers may edit their own listing or delete it, or view a summary of quotes that have been received for it, and repair specialist users may send a message to the customer (e.g. if they need clarification on some details) or submit a repair quote to the customer for the listing being viewed and/or view a list of any other quotes that they have previously submitted for the same listing.

Users also have the option to add a listing to their Watchlist view, by clicking on the star icon – either on the listing’s card in the search results page, or on the individual listing details page. This feature allows quick navigation when returning to a listing (e.g. if a repair specialist needs to gather information before submitting a repair quote).

|  |  |
| --- | --- |
|  |  |
| Figure 10: Customer listings summary | Figure 11: Quotes/Orders summary pages |

Customers can view a summary of listings that they have created in the site (see Figure 10) and both customers and specialists have summary views for Quotes and Orders (Figure 11) which show the main details for each record and include some contextual links:

* to view the listing/order/quote
* to edit/delete the record
* to relist (in the case of an expired listing)
* to view a summary of quotes received for a specific listing
* to navigate quickly to the listing associated with a quote/order record
* to view the profile of a customer/repair specialist who has been assigned to an order

The Quotes and Orders summary pages have specific tabs for the ‘customer’ and ‘repair specialist’ roles (Quotes Received/Quotes Submitted and Customer Orders/Specialist Orders). In the case where a user has one role, they will only see one of these tabs, but if they have assigned both roles to their account, they will see both tabs as indicated in Figure 11.

|  |  |
| --- | --- |
|  |  |
| Figure 12: Listing creation | Figure 13: Quote creation |

The Listing and Quote creation screens allow users to enter details for the records they are creating, and contain validation for inputs (as can be seen in Figure 12) and the ability to upload attachments (which are also validated according to their size and MIME type). When creating a quote, a user will see brief summary details for the associated listing and customer at the top of the page.

|  |  |
| --- | --- |
|  |  |
| Figure 14: Order creation | Figure 15: Order view including comment history with status updates |

The order creation screen follows the design of the quote creation screen (see Figure 13), listing summary details of the related listing/quote and allowing the user to add attachments and a comment when creating an order.

The order view screen also has a similar layout, and contains a section at the bottom giving context-specific actions to the customer and repair specialist who are assigned to the order. The order is initially created with a status of ‘Created’ and specific status updates are permitted to one or other party according to the current status (as can be seen in the comment history in Figure 15).

Upon completion of an order, the status will be marked as ‘Closed’ and at this point both the customer and the repair specialist can add a feedback rating and comment to it. The feedback ratings for customers and repair specialists can then be seen on their profile page (visible to other logged-in users) – see Figure 16 below. This is useful to give assurance to future potential customers/specialists about the other party with whom they are dealing.

|  |  |
| --- | --- |
|  |  |
| Figure 16: User profile view | Figure 17: User profile search (available to admin only) |

In addition to feedback ratings, the user profile view contains summary information about their location, the role(s) assigned to their account, and their activity in the site (in terms of listings/quotes/orders – shown as appropriate according to their role).

An admin-only view (Figure 17) exists for administrator users to search for specific users in the site and edit/delete their profiles, if necessary, or message them through the site’s messaging facility.

|  |  |
| --- | --- |
|  |  |
| Figure 18: Compose message/Read message view | Figure 19: View Message / My Messages view |

Individual message compose/view screens are shown above with links to the sender profile (for the recipient) and the listing/quote (for the sender) in cases where the user has viewed a listing or a quote and used the ‘send message’ option in relation to it.

Each user has a ‘My Messages’ view (Figure 19) where messages sent within the site are shown (separated by tabs into received/sent items, and shown with a read/unread status). Note: these are not always the same as external email notifications that are sent by the site, although there is some overlap. Not every external email notification will generate an internal message (the rationale behind external email notification is discussed later in this report).

# Architecture Design & Algorithm Explanation

A high-level overview of the architecture of the code should be provided in this section. The overview should be designed to help another person seeking to adapt and maintain the software and should refer to the source files used in the project. This chapter should also include a description of any complex algorithms that may be hard to understand simply by reading the code and its comments. Where algorithms and architecture are based on other work this should be clearly explained as well as any references to external explanations of algorithms or architecture used in developing the software.

The RepairMart project has been developed in Laravel and consists of:

* Front-end user interface
* Back-end database framework
* File storage system

The Laravel codebase interacts with these different parts of the architecture to authorise, validate and facilitate the different user actions and perform CRUD operations in the back-end database.

The following diagram outlines the high-level system architecture. The choices made in relation to architecture options are briefly discussed below.

|  |
| --- |
|  |
| Figure 20: RepairMart system architecture |

## **Front end**

Bootstrap

## **Back end**

SQLite

## **File storage**

Best practice is followed for file management (e.g. images/videos added as attachments for quotes, listings and orders in the system) (Corey, 2021): this means that these files are not stored as Varbinary in the database itself, but in disk storage at a location which is accessible to the Laravel code via a folder link.

|  |
| --- |
|  |
| Figure 21: RepairMart file storage |

This configuration is easily transferable when deploying the system in different environments, and avoids several costs associated with the storage of Varbinary files in the database itself – namely, much larger database sizes and increased wait times on database backup and restore. Although this would be unlikely to cause problems for the application’s development, it could prove to be very significant for a production application with a large number of users and listings. Additionally, if/when such a large user base is acquired, the scaling of file storage space is much cheaper when using in-built server file management systems vs. Varbinary database storage.

## **Laravel code base**

xxx

## **Laravel development framework**

[Laravel](https://laravel.com/) was selected as the primary framework for developing the RepairMart application, due to several key advantages which it offers in relation to this type of project.

Laravel's elegant syntax and robust structure is well-suited to the development process, allowing efficient implementation of several essential features:

* Eloquent ORM (object-relational mapping) permits easy management of the relationships between different entities in the database (users, listings, quotes, orders and emails), and greatly simplifies the process of querying and updating database tables, optimising joins, filters and other database operations while using simple and logical syntax (Shukla, 2023)
* Migrations, Factories and Seeders are easy to define and run, and allow efficient database creation, population and version control (Gacovski, 2025) while also facilitating transferability between different environments (e.g. dev/test/prod) (Ditya, 2025) and deployment on a range of SQL database management systems (Dev, 2023)
* User authentication is critical for the RepairMart application, and greatly simplified in Laravel (Barot, 2025)
* Role-based permission management is also an important area of focus for the application, as the different actions allowed for customers and repair specialists need to be clearly separated. Laravel easily facilitates this (Katariya, 2025)

Its built-in routing, templating and database interaction were able to streamline the coding process, reducing development time and effort. Furthermore, Laravel's MVC (Model-View-Controller) architecture promoted a clean and organized codebase, enhancing maintainability and scalability. The framework's extensive documentation and large, active community provided invaluable support throughout the development lifecycle. While other frameworks like React or Angular could have been used for the front-end, Laravel's full-stack capabilities simplified the development process by providing a unified framework for both front-end and back-end development. The choice of SQLite as the database management system was primarily driven by its ease of use and suitability for this project's scale. While other SQL databases like MySQL or PostgreSQL offer more advanced features, SQLite's simplicity and ease of setup made it an ideal choice for rapid prototyping and development. Laravel's database-agnostic nature ensures that the application can be easily deployed with other SQL databases in the future, should the need arise. Although Laravel has some potential disadvantages, such as a slightly steeper learning curve compared to some other frameworks, and potential performance limitations at extremely large scales, these were not critical concerns for this project, given its current size and scope. The benefits of Laravel's structure, features, and community support far outweighed these minor drawbacks, making it the optimal choice for this application.

## **Database**

### **UML diagrams**

In the early phases of developing the RepairMart application, UML (Unified Modelling Language) diagrams were valuable for visualising and planning the system's architecture. For this project, Use Case and Sequence diagrams were produced:

* Use Case diagrams helped to define user interactions and system functionalities, clarifying requirements from the perspectives of customers, repair specialists, and administrators.
* A Sequence diagram mapped out the interactions between different system components, such as the website, database, and email server, to identify potential bottlenecks and ensure smooth operational flows.

Diagrams were not needed for every workflow (e.g. the workflow for sending a message to another user in the system was simple enough to implement without one) but the main characteristics of the account creation and repair request listing processes are detailed in UML Use Case and Sequence diagrams in [Appendix 2](#_Appendix_2:_UML).

### **Entity-Relationship diagrams**

in parallel with UML diagram creation, a discovery exercise was carried out to identify the key data points from the [original UI designs](#_Appendix_1:_Original). The results from this exercise were used to create a preliminary database model (see below), and from this it was possible to create a reasonably accurate entity-relationship diagram. This was deemed sufficient to start database development work, and as a result it was decided that creating UML Class diagrams would not be necessary for this project.

|  |
| --- |
|  |
| Figure 22: Preliminary Entity-Relationship diagram |

The preliminary ERD was ‘workshopped’ in Microsoft SQL Server: using the data points identified in the discovery exercise, tables and stored procedures were created and tested – separated from the application code base.

### Static data

As part of the above workshopping, the following static data tables were populated:

* **countries & currencies**

A list of ISO country & currency codes and names, retrieved from [iso.org](https://www.iso.org/iso-4217-currency-codes.html)

* **manufacturers**

A list of electronic goods manufacturers: these are often used to filter product searches on websites such as Amazon and eBay, and will be incorporated into listing details on the RepairMart application. The list has been gathered from [Wikipedia](https://en.wikipedia.org/wiki/List_of_electronics_brands), although it has been filtered to exclude many less well-known manufacturers. An ‘—Other-Unknown—’ value has also been included in this list, which can be selected when a manufacturer is not known or cannot be found. In such cases, it is envisaged that a data quality analysis task could be performed to examine products listed with an unknown manufacturer and potentially add new names to the list for future use.

* **products**

A product category/subcategory classification for use in listings in the application.

Consideration was given to how complicated the RepairMart product classification needed to be, and various options were examined. Since there is no know analogue at present which is fully equivalent to the solution that we intend to build with this project, no ready-made list of electronics product categories was available to copy (unlike the case of ISO countries and currencies).

In order to avoid overcomplicating both our database design and the user experience, it was decided to use a 2-tier product classification, which is a slimmed-down version of the Amazon classification found [here](https://www.asinspotlight.com/amz-categories-list-csv).

* **feedback\_types**
* listing\_statuses, order\_statuses, quote\_statuses
* **deliverymethods**

A fixed list of delivery methods that repair specialists can choose in the system, when creating a quote in response to a repair request. These are: (1) Pick-up/Drop-off at Customer, (2) Drop-off/Pick-up at Repair Specialist, (3) Postage (tracked), and (4) Postage (untracked).

### Database relationships

#### Quote/Delivery Method Relationship

Choosing a delivery method for orders is one area which can be very complicated. When considering options and configuration to use, analogous examples such as eBay’s shipping method selection were considered, however mirroring eBay’s offering (which includes the ability to quote multiple delivery methods per listing, and pay for shipping & print out packaging labels directly from the eBay website) was deemed to be overly complex for the initial purposes of the RepairMart application.

A decision was made to simplify the process of quote creation: rather than permitting multiple delivery methods per quote submitted, allowing a variable quote amount which would be calculated based on the shipping method selected by a customer, it was decided to only allow a single delivery method per quote. In the event that a repair specialist wishes to offer multiple delivery methods to a customer, they will be able to submit more than one quote for the same listing, choosing a different delivery method for the new quote. In this case, the RepairMart system is not burdened with calculating the delivery cost for each method offered – instead, repair specialists are responsible for calculating the overall cost for their quote (including delivery fees) and quoting a single amount in the system.

### **Choice of database option (SQL variant)**

Table creation: data items, datatypes, stored procedures. Deletion/Archiving (facilitate soft delete for data preservation/warehousing if retention is needed)

## **UI Development**

Front-end organisation (UI layout) – use existing template & customise

Link to back-end – API or not?

# Testing

This section should include a justification for the approaches used to test the resulting system. These may include such approaches as unit tests, manual and/or automated end 2 end testing, performance testing etc. The section should explain what parts of the project are at greatest risk of having uncaught bugs within them, for example because of their complexity, and how the approach to testing has been developed to analyse those areas in detail. For particularly complex algorithms for example this section can outline the design of the test cases to ensure good test coverage. The section should clearly reference any automated testing code, manual testing plans and test results included with the project that provides clear evidence that testing has taken place and can be easily used by anyone further developing the project.

The degree to which the student has performed testing which can include unit, end 2 end, user testing, a manual testing plan with evidence of having performed it, automated testing, performance measurement/optimisation if relevant.

# Evaluation and Conclusion

This section should include a general evaluation of the success of the project measured against the criteria stated in the introduction and/or requirements. An evaluation of the hardware/software environment and language used may also be presented. Suggestions for further work should also be discussed. Do not be afraid to be critical or to draw a negative conclusion; not all projects will be successful. This section should provide a thorough and honest reflection on the process followed in the project and the results of that process. To do this well, the student should not leave any blind spots in their reflection and should identify the most and least successful aspects of the project. It should be written in such a way as to be helpful to a person seeking to adapt the project or to create a similar project in the future.

# References

Beckwith, K. (2023, October 30). *Ubuntu’s new variable fonts*. Retrieved from typenetwork.com: https://typenetwork.com/articles/ubuntus-new-variable-fonts

Brannon, I. (2024, Spring). *A Criticism of ‘Right to Repair’ Laws*. Retrieved from CATO Institute: https://www.cato.org/regulation/spring-2024/criticism-right-repair-laws

Conway, L. (2021, September 24). *Right to Repair Regulations.* Retrieved from UK Parliament - House of Commons Library: https://researchbriefings.files.parliament.uk/documents/CBP-9302/CBP-9302.pdf

European Commission. (2023, March 22). *‘Right to repair': Questions & Answers*. Retrieved from European Commission: https://ec.europa.eu/commission/presscorner/detail/es/qanda\_23\_1795

García Molyneux, C., & Oberschelp de Meneses, A. (2024, June 10). *The EU Adopts Right to Repair Directive*. Retrieved from Inside Energy & Environment: https://www.insideenergyandenvironment.com/2024/06/the-eu-adopts-right-to-repair-directive/

Rodriguez, C. (2019, April 29). *Ubuntu's Use and Color Theory. Comparing to Helvetica*. Retrieved from Medium: https://medium.com/@rodriguez.cesar009/ubuntus-use-and-color-theory-2442a779aab1

World Health Organization. (2024, October 1). *Electronic waste (e-waste)*. Retrieved from World Health Organization: https://www.who.int/news-room/fact-sheets/detail/electronic-waste-(e-waste)

# Appendices

These may include:

▪ Testing code, datasets and results.

▪ Printout of code for section/sub-sections of the application developed in relationship to the submission.

▪ In most cases this information should be present within the code submission of the project

## Appendix 1: Original Figma UI designs

A sample of the original Figma UI designs for this project are shown below.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

The full set of pages can be viewed at the following links:

[Presentation of individual pages](https://www.figma.com/proto/IMPhnXOBdeJeiRnOl32IMa/RepairMart?node-id=5-3&p=f&t=UGb5pkvM5wl9wqY0-0&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=5%3A3)

[Overall design](https://www.figma.com/design/IMPhnXOBdeJeiRnOl32IMa/RepairMart?node-id=0-1&t=ywKro3jD78llIxHB-1)

## Appendix 2: UML diagrams

UML use case diagrams for account creation & repair request listing

|  |
| --- |
|  |
|  |

Sequence diagram for repair request listing

|  |
| --- |
|  |