‘Title of Dissertation’

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

‘Date of Submission’

Contents

[Acknowledgements 3](#_Toc191370956)

[Abstract 4](#_Toc191370957)

[Introduction 5](#_Toc191370958)

[Understanding the Problem 7](#_Toc191370959)

[User Interface Design 10](#_Toc191370960)

[Architecture Design & Algorithm Explanation 11](#_Toc191370961)

[Testing 12](#_Toc191370962)

[Evaluation and Conclusion 13](#_Toc191370963)

[References/Bibliography 14](#_Toc191370964)

[Appendices 15](#_Toc191370965)

# Acknowledgements

I would like to thank my supervisor, Olga Pishchukhina.

# Abstract

# Introduction

**Problem Specification**

Waste from electronic consumer goods is a big problem. Many consumers currently dispose of products which no longer function correctly in an inefficient way when they could still be repaired for use. This creates unnecessary waste and causes people to spend more than they need to on replacement products. The current consumer economy is not geared towards repairing faulty items: although ‘Right to Repair’ laws have been introduced in the UK and the EU (in July 2021 and April 2024, respectively) the new legislation is limited (e.g. in the case of the UK, the law only covers dishwashers, washing machines, washer-dryers, refrigeration appliances, and televisions & electronic displays). Furthermore, when seeking repairs, consumers are directed towards a single point of contact (the manufacturer or vendor of their non-functioning product), which could limit their options in terms of price, location, viability, etc.

What this project proposes is to develop a repair ‘tendering’ system (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 (whether these are large manufacturers, small businesses or independent individuals with the relevant expertise). The specialists can review requests, ask any relevant questions for clarification on the product and/or its defects, and submit a quote for repair to the consumer. The consumer should then be able to view a list of quotes received and choose the most suitable option for their needs – resulting in the extension of their product’s useful life and a reduction in electronic waste.

**Current options**

When dealing with electronic goods that no longer work correctly, consumers can either (a) dispose of it 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
* 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 above-mentioned 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).

**Solution**

Creating a centralised platform where consumers can submit requests for repair can solve a lot 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).

# 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.

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/>), these websites only offer contact with a single repair specialist (or single organisation) at a time, and 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, in order 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

|  |  |
| --- | --- |
| **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.

#### **Possible 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

This chapter should describe, illustrate and justify the user interface design of your proposed system. Not all projects will have a significant user interface component, for example if they are back-end algorithms or experimental projects. For projects without a GUI a short overview of the interfaces to the software should be outlined. For projects that have more substantial graphical user interfaces there should be an explanation for how the design has been developed including any feedback that shaped the design. The goal being to ensure that anyone building on the system understands the reasoning behind the UI and the feedback of users that led to its design.

### **Design Process**

#### **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 we wish to incorporate (with some adaptation) into our solution.

|  |  |
| --- | --- |
| Feature/functionality |  |
|  |  |
|  |  |
|  |  |
|  |  |

In the first instance, Figma used as a tool for designing the different pages that

#### **Identification of functionality & design aspects to include**

#### **UML diagrams**

# 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. Where relevant this section should also explain the design of any api interfaces that have been designed for the project to enable others to easily interface with 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.

### **Database**

#### **Identification of data points (following UML design)**

#### **Database design – tables, PKs, foreign keys**

#### **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/Bibliography

A list of relevant references to other documents (books, papers, web-sites, etc.) which are cited in the main body of the text.

For books, record:

▪ The author’s name or the editor’s name

▪ The year the book was published

▪ The title of the book

▪ The edition of the book

▪ The name of the publisher

▪ The ISBN of the book

For electronic resources, record:

▪ The author’s name or the editor’s name where available

▪ The year the article was published where available

▪ The title of the article

▪ The date the source of the information was accessed

▪ The electronic address or email

▪ The type of electronic resource (email, discussion forum, WWW page, etc)

# 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