AGRILAND MEDIA

Development of an Online Property Listing Web Application

20th March 2015

Prepared by: Robert Boland

Supervisor: Aideen Keaney
Declaration

I declare that the work described in this dissertation has been carried out in full compliance with the ethical research requirements of the School of Computer Science and Statistics.

Signed: ____________________

Robert Boland

20th March 2015
ABSTRACT

The objective of this project was to design and develop a web based property listing application for Agriland Media. The project involved identifying the feature requirements of the application as well as developing the application to fulfil these requirements. The application allows users to browse and search the site for property listings as well as place their own listings on the site. The application is also fully responsive allowing it to work across mobile, tablet and desktop devices. It has been developed using the Laravel PHP framework, MySQL, JavaScript, HTML and CSS.
Agriland Media is the largest digital publisher in the agricultural sector in Ireland. Established in 2013, Agriland aims to create one online resource for all the information, news and services required by people involved in the agricultural and farming sectors in Ireland. In an effort to continue expanding its range of services, Agriland is now looking to deploy an online property service unique to the agricultural industry. Cormac Farrelly, the managing director of Agriland, was the main client contact throughout and will hereafter be referred to as ‘the client’.

The design and development of this property listing web application has been completed to satisfy the client’s requirements. The application implements a responsive web design which allows the site to work across a variety of web browsers and screen sizes. As such this web application is compatible for mobile, tablet and desktop devices. It is expected that a non-technical audience will be using the application. As a result of this, an emphasis has been placed on ensuring the application is intuitive and user friendly.

As Agriland is a rapidly growing start up company, they expect to continue to expand this application and the range of services they will supply. The application has been developed with this in mind. Working closely with the client, the applications development has been carefully planned and structured in such a way that the platform will be easy to maintain and extend. This will allow future developers to continue to manage the application and develop new functionality as the need arises.

I would like to thank Cormac Farrelly of Agriland for all his help and input on this project. Cormac was very generous with his time and was always available to answer any questions I had. Finally I would like to thank my supervisor Aideen Keaney for providing consistent support and concise direction throughout the project.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>NO.</th>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION AND SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1 The Client</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 The Project Background</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.3 Terms of Reference</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.4 Summary of Remaining Chapters</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>SYSTEM OVERVIEW</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.1 System Objectives</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.2 Technical Environment</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.3 System Overview Diagram</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>DESCRIPTION OF THE WORK DONE</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>3.1 Defining the application requirements</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>3.2 Choosing The Development Language</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3.3 Designing the Application</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>3.4 Development of the application</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3.5 Testing</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.6 Additional Features</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.7 Difficulties Encountered</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4.1 Conclusions</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4.2 Recommendations</td>
<td>24</td>
</tr>
<tr>
<td>NO.</td>
<td>CONTENT</td>
<td>PAGE</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A</td>
<td>Original Project Outline</td>
<td>A-1</td>
</tr>
<tr>
<td>B</td>
<td>Interim Report</td>
<td>B-1</td>
</tr>
<tr>
<td>C</td>
<td>Administrator Manual</td>
<td>C-1</td>
</tr>
<tr>
<td>D</td>
<td>Application Screenshots</td>
<td>D-1</td>
</tr>
<tr>
<td>E</td>
<td>Design Documentation</td>
<td>E-1</td>
</tr>
<tr>
<td>F</td>
<td>User Journeys</td>
<td>F-1</td>
</tr>
<tr>
<td>G</td>
<td>Technical Documentation</td>
<td>G-1</td>
</tr>
<tr>
<td>H</td>
<td>Database Tables and Relationships</td>
<td>H-1</td>
</tr>
<tr>
<td>I</td>
<td>Testing Documentation</td>
<td>I-1</td>
</tr>
<tr>
<td>J</td>
<td>Application Code</td>
<td>J-1</td>
</tr>
</tbody>
</table>

References

Glossary
1 INTRODUCTION AND SUMMARY

The purpose of this chapter is to introduce the client, provide background information regarding the project and the terms of reference agreed with the client. It then provides a summary of the remaining chapters in this report.

1.1 The Client

Agriland was established in 2013 with the aim of providing a central online resource for news, services and information required by people in the agricultural and farming sectors in Ireland. Since the company has launched, it has become Ireland’s largest digital publisher and the first daily content provider in the agricultural industry. The company now reaches close to 100,000 users each month through its website and mobile application.

1.2 The Project Background

In order to continue expanding, Agriland has been looking for additional services to supply to its ever growing audience. After analysing their audience and the services available to them, they have identified that there is currently no online platform to allow estate agents and land owners to list agricultural land for sale or available for renting/sharing. Current popular online property services only cater for residential and commercial properties which are not suitable for agricultural listings. These existing services do not allow the user to upload a listing specific for agricultural property as they have no facility to allow the user to provide details of the size of the property nor do they allow a property to be listed without a fixed residential address. Furthermore, the audience visiting these popular online property services are not interested in agricultural property. There is currently no online resource which advertises a wide selection of agricultural listings. As a result agricultural listings are only advertised in agricultural print media in an effort to reach their target audience. Agriland is looking to fill this gap by creating the first online resource for agricultural listings. By doing so it can take advantage of its presence in the agricultural space and launch a property service where it can guarantee customers a targeted agricultural audience.

This project involves developing an online property application that will allow users to upload agricultural property which is available for sale, for renting or for sharing. Sharing is an agreement between individuals to use a section of land and share profits.

The project will:

- Provide an easy to use and intuitive interface. The site will be designed to cater for a non-technical audience and ensure it is easy to perform tasks such as searching and placing a listing. See Section 2 for more information.
- Have a responsive design to work on multiple devices and platforms. This will allow the site to work on mobile, tablet and desktop devices. This is essential as close to 50% of Agriland’s users access their site on mobile devices. See Section 2 for more information.
• Provide helpful tools for administrators of the website. This will allow administrators to moderate listings and content on the site as well as managing user accounts and resetting passwords. See Section 2 for more information.

Finally since Agriland is continuing to grow and expand, they require the application to be easy to maintain and extend. This will allow the client to continue to manage the application and develop additional elements as needed in the future.

1.3 Terms of Reference

The final terms of reference for the project are outlined below:

• Define the feature requirements for an online property listing service tailored for the agriculture industry.
• Identify appropriate technologies to develop the application.
• Ensure the application is scalable and expandable for future development.
• Design an intuitive and user friendly interface that is responsive to work across multiple screen sizes.
• Draw wireframes for the application.
• Develop tools to allow administrators to manage listings and users.
• Integrate online payments.
• Provide the necessary technical documentation and user manuals for administrators.

The final terms of reference are different from the initial terms of reference as additional information was discovered while defining the application requirements. See Section 3.1 for more information.

The project has exceeded the initial terms of reference and includes additional features that were not specified. See Section 3.6 for more information.

1.4 Summary of Remaining Chapters

• Chapter 2: System Overview provides an overview of the system. It describes the objectives of the system, summarises the technical environment and provides a system overview diagram.
• Chapter 3: Description of the Work Done describes the work completed throughout the project. It outlines the methodology for defining the site requirements, the technologies used, the development environment and the rationale for certain development decisions. It outlines the procedure for creating a responsive design as well as issues and problems encountered throughout the project.
• Chapter 4: Conclusions and Recommendations provides a summary of the conclusions drawn from the project as well as recommendations for future development and expansion of the system.

A glossary of terminology is also available in the appendix.
2 SYSTEM OVERVIEW

This chapter provides a basic overview of the final application that has been developed. It provides details of the final objectives of the system, describes the system, the third party API’s and gives an overview of the technical environment. A system overview diagram is also provided giving a high level overview of the application and its primary functions.

2.1 System Objectives

The objective of the system is to provide the client with a new platform to increase the range of services it offers to the agricultural industry. This new system will provide estate agents and users with a platform that they can upload agriculture property for sale, for renting or sharing. It will also allow users to easily search for agricultural property available online, something which is not currently possible as there is no central resource for this. The system will also be designed so that it works on mobile, tablet and desktop devices. In order to provide certain features, the system will also have to integrate with a number of third party API’s.

The following are the finalised requirements of the system:

- To identify an appropriate platform to build the listings application which will align with the clients current systems. The system needs to be well organised and easy to maintain. As the client’s number of services grow, it will become increasingly important that they are easy to maintain.
- The client plans to develop additional features for the application in the future. As a result the application needs to be easy for developers to extend.
- It should allow users to register a new account or login to an existing one. Once logged in users will be able to place a listing for sale or for rent. The user can populate the listing details, choose the property location from a map as well as upload multiple images. Users will then proceed to a checkout to pay for their listing in order for it to become active on the site.
- The application should make use of the Google Maps API. The API will be used to display maps of current listings on the application and allow a user to select a location for a new listing. The Google Maps API will also be used to automatically predict the address of a listing on the site.
- Allow users to advertise opportunities for sharing a property or to be added to a directory to express their interest if they are seeking to share a property.
- Provide an intuitive interface which would allow users to quickly and easily browse the site. Relevant search criteria should also be provided for each type of listing so users can filter listings by location, price and size.
- Create an administrative section which will allow administrators to manage and moderate listings on the site. Summary statistics of the listings on site will also be provided in this section. Administrators will also have the ability to manage user accounts and reset passwords.
- The site needs to follow best practice Search Engine Optimisation techniques and coding standards. This will improve the speed of the application as well as increasing the likelihood of the site showing up in search engine results.
• The site should also make use of responsive web design. By doing so, the site will automatically restructure depending on the size of the screen that the user is using. This means that the site will display and function correctly a wide range of devices and browsers such as mobiles, tablets and desktops. Currently almost 50% of the client’s users are accessing their services on mobile devices.
• The site should have the ability to send automated emails via a third party API. Emails to confirm a user’s registration or reset their password should be set up.
• The necessary technical documentation and administrator user manuals should also be provided.

2.2 Technical Environment

The application has been developed using PHP as the server side language, MySQL as the relational database with HTML5, CSS3 and JavaScript used to power the front end of the website.

In order to develop using PHP, the files need to be run on a server so the code can be compiled. For development purposes a program called XAMPP was used. XAMPP is an open source cross platform web server which can be installed onto any computer. XAMPP simulates how the application would run if it was uploaded to a web server. By using XAMPP it meant that the application would operate as expected, even if it was deployed on a Windows or Linux based server in the future as they all compile PHP the same way.

Due to the size of the application and in order to ensure it was scalable, the application has been developed using the Laravel framework. This framework is based on PHP and uses a MVC architecture which allows the application to be easily extended and maintained (See Section 3.2 for more information).

The tools used in the XAMPP environment included phpMyAdmin for managing the database and Sublime Text Editor 2 was used for writing the applications code. The web application has been tested to work with all modern browsers. GitHub was used to manage different versions of the program and to store the code securely in cloud storage. See Section 3.2 for more information.

Additional technical documentation for the application is also available in Appendix G.

2.3 System Overview Diagram

The complex design of the system means that there are several different API’s and programs interacting with the system at any one time. Some of these API’s are used to retrieve information, such as the Google Maps API. Other API’s are used for sending information such as the Mandrill Email API. A visual representation of the API’s and entities interacting with this system can be seen in Figure 2.3.1. Further discussion of these components is presented below:

• **Client:** The client is somebody who is using the application via a web browser. A client is able to search and browse listings on the website. They are also able to create a new listing and upload content to the site. The client can be a regular user of the website or an administrator of the system.
• **Google Maps API:** The Google Maps API is used to generate maps with the location of the listings on them. This allows the client to see a visual overview of the listings and the areas surrounding them. The Google Maps API is also used to provide GPS co-ordinates and address details to the application which are stored in the database. See Section 3.4 for more information.

• **Database:** The database is used to store all of the information that the system needs. When a user searches for listings, this information is retrieved from the database and presented to the user. When a user places a new listing on the site, this information is saved to the database so it can be accessed again. See Section 3.4 for more information.

• **Realex Payments API:** The Realex Payments API is used to process payments in the system. In order to ensure security, credit card details are passed to the secure Realex Payments system. Realex Payments process and charge the credit card the appropriate amount. Realex Payments then communicate with the application to confirm a successful credit card charge has been completed. See Section 3.4 for more information.

• **Mandrill API:** The Mandrill API is used to send emails from the application to the user. For example once a user has signed up, the application will send a request to the Mandrill API and trigger a welcome email to be sent to the new users email address. See Section 3.4 for more information.

![Figure 2.3.1 – Visual representation of API's and entities interacting with the system]

The system can be divided up into two main components. The front end of the application and the back end of the application.

• **Front End:** The front end of the application is responsible for displaying the information to the user. The front end is the part of the application which is visible to the user. It is also the part of the system which the user will interact with by inputting
text, clicking links and uploading images. The front end of this system is designed to be responsive. This means that the front end of the application will adapt to the screen size of the user's browser. This allows the information to be presented to the user in an easy to read format if they are on a mobile, tablet or desktop device. See section 3.3 for more information.

- **Back End:** The back end of the application contains the logic required for the system to operate. It is made up of three parts: the server, the application code itself and the database. When the user interacts with the front end of the website, the information is passed to the application code in the back end. The application then interacts with the database and saves, updates or retrieves information as needed. It then will pass information to the front end of the website to be displayed to the user. The backend is also responsible for interacting with third party systems such as the Google Maps API and the Realex Payments API.

Figure 2.3.2 provides a sitemap for the web application. Certain elements of the site require the user to be logged in. For example the user cannot upload a new listing to the site if they are not logged in. The system also supports administrator accounts. If a user has an administrator account they can access all areas of the site and edit a listing or a user’s account details.
An overview of the key functions of the application and their interactions with the system are described below:

- **Searching listings:** When a user performs a search for listings on the application, they input their search criteria into a form on the front end of the website. This information is then passed to the backend of the website. The application looks up listings in the database based on the search criteria provided by the user. The listings that have been retrieved are then passed to the front end of the website to be viewed by the user.

- **Placing a listing:** A user can place a new listing on the website by inputting information into the new listing form. Figure 2.3.3 shows a system overview diagram for this process. It illustrates front end and back end interactions in the system. First the user chooses the type of listing they would like to create. The application checks if they are logged in, and if so presents the user with a form. The user completes the form details and interacts with a map generated from the Google Maps API. The user selects the location of their listing from the map and the Google Maps API automatically predicts the address of their listing and populates the form. Once the user has input all the information into this form, it is validated and checked for errors by the backend of the website. If the validation checks are passed, the application saves the information to the database and then communicates with the Realex Payments API to process a credit card payment. If a payment is successful, the application communicates with the Mandrill API to send a receipt to the users email address.

- **Register a new account:** The user can register a new account by completing the registration form on the front end of the website. This information is then passed to the backend of the website to be validated. If it passes validation then the users account details are stored in the database. The users password is also encrypted for privacy and security reasons. The application then communicates with the Mandrill API to send a welcome email to users email account.

Additional system activity diagrams and data flow diagrams can be found in Appendix E.
Figure 2.3.3 – System overview of front end and back end interactions when placing a listing
3 DESCRIPTION OF THE WORK DONE
This chapter details the work which has been carried out over the course of the projects life. The system was developed using the Incremental Model to allow modifications to be made to the system. The incremental model consists of multiple stages including requirements, design, development, testing and implementation (Larman & R.Basili, 2003). This model is very flexible and allows for modifications to the system as the client is constantly providing feedback at each stage. See Figure E.6, p. E.5.

The project has been divided up into the following sections:

- Defining the application requirements;
- Choosing the development language;
- Establishing user journeys and designing the look of the application;
- Development of the application;
- Testing and issues identified.

3.1 Defining the application requirements
A critical part before developing and designing the application was establishing the requirements with the client. Currently if a user wishes to advertise an agricultural property they have to use print based media. There is no online resource that provides a dedicated service for listing agricultural land and property available for sale, for rent or for sharing. Since the residential and commercial industries have numerous services such as www.daft.ie and www.myhome.ie, these were used as a starting point to determine the basic features the application should provide. The user journeys for the client’s application were determined, and these were then performed on these websites. Each of these user journeys was then analysed to see what could be improved. See Section 3.3.

While analysing these services, it was noticed that they were geared towards the residential industry and did not allow for the input of information specific to the agricultural industry such as the size of the property. Extensive consultation was carried out with the client and the staff in Agriland to gain a better understanding of the agriculture industry to determine the needs and requirements of such a system. The following items were deemed to be important elements of the application:

- Each listing should have the size of the property in acres and in hectares.
- Due to the rural location of many of these listings, a field to provide directions should be available. GPS co-ordinates for each listing should also be present.
- When viewing the property location on a map it should default to a satellite view, allowing individuals to see the terrain and surrounding land (an important factor for agricultural land).
- In order to improve the user experience of the site, the main functions (such as placing a listing) should require as few steps as possible.
- The administrators of the site will need the ability to set up and manage listings on behalf of users. Administrators will also need to be able to manage a user’s account and reset passwords.
- Initially the client had proposed that wireframes for the site should be drawn up. However after presenting the client with the results of the user journey analysis, it
was determined that wireframes would not be necessary. Wireframes are more appropriate for sites with multiple levels of content and information that require a lot of user navigation. This project aimed to reduce the amount of user navigation as much as possible, so it was determined that wireframes were unnecessary.

The client also consulted with potential customers and discovered additional elements of the application that may need to be developed in the future. For example, real estate agents would prefer an API that could be connected to their systems to automatically publish listings. In order to ensure that it was possible to develop these features in the future, it was determined that the application needed to be extendable and easy to maintain. By developing the application like this, it would allow future developers to expand on existing features and develop new ones as needed.

### 3.2 Choosing The Development Language

The current Agriland platform is built using PHP and a MySQL database with a content management system provided by Wordpress. This system is used by the editorial staff to produce a number of news articles each day. Initially it was proposed that a plugin would be developed with PHP for Wordpress which would provide the property listing feature. However after speaking with the client and deciding in the terms of reference that the platform needed to be extendable, it was decided the best solution would be to build the platform independently of the current Wordpress system.

There are a number of reasons for this decision:

- The current Wordpress site experiences high volumes of traffic and can experience slow performance during peak times. Adding the property listings service to the current system would place even more pressure on the system, particularly at peak times.
- Wordpress is primarily designed for displaying content and news stories, and while there are plugins available to extend its functionality, it is not always efficient to do this. It involves using Wordpress for a purpose other than what it was intended for. By developing the new application independently it will allow for code to be developed for its intended purpose.
- It will be much easier to maintain the Wordpress system and the property listing application independently. If Wordpress is updated it will have no impact on the property listing application and vice versa. This will make it much easier to maintain and develop each of the systems as developers will not have to worry about changes conflicting with each other.
- The client wishes to develop future components outside the scope of this project such as an API, CRM system and a mobile application. In order to do this the client will need a solid reliable platform that is extendable. If the application was developed as a Wordpress plugin it would not allow for these additional features to be implemented further down the line.
- By developing the property listing application independently of the current Wordpress platform, it will allow the client to deploy the application on a separate server. This will allow for better allocation of server resources.
While there is a significant benefit to developing the application independent of the current Wordpress platform, there are still some concerns to be addressed:

- In time the client plans to integrate certain components of the property listing application with the Wordpress based platform. This will be more difficult if they are on separate platforms, but provided the property application is developed using PHP and MySQL this should not be an issue as they will be using the same language.
- The current Wordpress platform allows users to login and register when posting comments on the site. In order to ensure a consistent user experience the new application should allow these users to log into both systems with the same credentials. After discussing this with the client this was not deemed as a vital requirement as they are not currently satisfied with the Wordpress user registration system. They are looking to migrate to using a social login for their commenting system in the future.
- Developing the property listing application independently will mean that there will be an additional library of code to be maintained. As such it is essential that the technical documentation provided is clear and concise so future developers can understand the application and its structure.

Taking into account the project requirements and the development requirements outlined above, it was decided that the application should be built with PHP using a MySQL database. It was also decided that a web development framework should be used. A web development framework greatly reduces development time as it provides built in libraries for working with databases, HTTP requests, templates, forms, caching and more.

There are a number of popular PHP frameworks available:

- Laravel
- CakePHP
- Smyfony
- Codeigniter
- Zend Framework 2

A comparison of these frameworks is presented in Table E.1, p. E.4.

After comparing each of the frameworks, it was decided that Laravel would be used to develop the property listing application. The reasons for this are summarised below:

- Laravel is the newest PHP framework of those compared in Table E.1, p. E.4. It uses modern libraries and techniques to ensure applications are fast and efficient. A recent survey found that 25% of PHP applications are built on the Laravel framework - more than any other framework (Skvorc, 2013).
- The clients developers are currently working on another project using this framework and felt comfortable that they would be able to extend and maintain the application if it was built using this framework.
- As can be seen in Table E.1, p. E.4 Laravel makes use of the MVC architecture which makes it easier to extend and maintain the application. See Section 3.4.
3.3 Designing the Application

As highlighted during the requirements gathering phase of the project, it is imperative that the application is simple and intuitive to use. All the main actions a user would wish to perform should require a minimal number of steps. The application was designed so that the user would be just one click away from all of the main functions on the site at any one time. This was achieved by placing a navigation bar with links to the main functions that are present on each page of the site.

A variety of access levels also had to be considered when designing the application. A number of different user types will be accessing the application, each of which has access to different components:

- **Public**: Any user who is not logged into the site is a public user. They have the ability to browse and search listings as well as contact listing owners via the contact form.
- **Standard Registered User**: This user has the same privileges as a public user but they also have the ability to place a listing on the site. They have access to their personal dashboard where they can view and edit all of their own listings. They will also be able to create a favourites list of listings and edit their personal details.
- **Estate Agent User**: This user has the same privileges as a standard registered user. In addition, they also have access to a more detailed page to edit their personal details. It will allow them to input their company details, description, address, and a company logo.
- **Administrator**: This user has full access to all areas of the site. They can edit and manage any users details or listings. They will also have access to the admin panel which contains site statistics, a user search, and all the listings on the site.

For security reasons, all users who sign up will be set up as a Standard Registered user. It will then be up to the administrator to manually grant higher permissions to an account.

In order to determine the appropriate structure and layout of the application, several user journeys were devised. A user journey allows us to understand the steps a user would need to take in order to complete an action. These user journeys were performed on popular residential/commercial property listing sites and analysed to see if any unnecessary steps could be removed. These user journeys can be found in Appendix F.

The main results of these user journey tests showed the following results:

- **Placing A Listing**: Other property listing services require the user to navigate to multiple pages when placing a listing. For this application, the process has been streamlined to allow a user to populate all the information at once on one page.
- **Browsing**: When trying to view properties for sale or for rent, other property listing services require you to input search criteria before any results are shown. This means a minimum of two clicks is required to view listings on these sites. In this application, it will automatically show the user all listings by default and allow them to then input search criteria if they wish. This reduces the number of clicks to just one.
- **Registering**: Registering on other property listing services requires you to confirm your email address or confirm your mobile number. While these additional steps help reduce fake accounts, they do increase the time it takes to register and login to the
site. It was decided that the application would provide a very quick and easy registration process with no confirmation of email accounts or mobile numbers needed.

- **Multiple Pages:** When browsing listings and search results on other property listing services, it requires a user to click onto another page to load additional results once they reach the end of the page. This requires another page load each time, slowing down the browsing process. The Agriland application makes use of Ajax pagination to automatically load the next page of listings and append them to the current page without requiring a page reload, speeding up the browsing process for the user.

### Responsive Design

One of the client's requirements is that the property listing application is responsive to work across multiple screen sizes and platforms. A responsive website dynamically repositions the content and text on the screen depending on the screen's resolution. All of the services Agriland currently offers are responsive and over 50% of the users who visit their website are using mobile devices. As such it is paramount that the application adapts to all screen sizes, particularly mobile.

In order to make a website responsive, a fluid, proportion based grid system is used. This is implemented by using CSS3 and media queries to reposition and adjust elements depending on the screen's resolution. When designing the layout of the website, this grid based system was kept in mind throughout. Each new layout was tested on desktop, tablet and mobile screen sizes to ensure that the site adapted appropriately. The various different layouts of the application on devices can be seen in Appendix D.

### Aesthetic Design

The aesthetic appearance is critical to ensuring that the application is intuitive and easy to use. A site that is well designed is more likely to encourage users to stay on the site and to explore the application. A well designed site also makes it easier for users to complete tasks on the site, for example - searching for listings.

### Brand Guidelines

The sites colour scheme, fonts and logo are all in line with the clients brand guidelines. Brand guidelines are a set of rules that explain which colours, logos, fonts etc… can be used when representing a brand.

A full width navigation menu has also been used. This is similar to the one the client uses on their current site to ensure a consistent brand experience across sites.

The clients existing site and brand guidelines determined the majority of the colour scheme of the site.

### Best Practices

As previously discussed, a grid system needs to be used for a responsive design. A challenge when designing a responsive site is ensuring certain elements will function correctly on a smaller screen size.
For example, the full width navigation bar running across the top of the site would not fit on a mobile device. The text would overflow and the navigation menu would be unusable as the space on a mobile screen is too small. In order to solve this, a simple dropdown menu is used when on mobile devices. The user can open and close the dropdown menu by tapping on the menu icon. This design of menu is easy to use on mobile devices and fits correctly on the small screen size. This dropdown menu can be seen closed in Figure 3.3.1 and open in Figure 3.3.2.

A decision was also made to ensure that important information would be displayed 'above the fold'. The term 'above the fold' represents the section of a website that is visible to the user without having to scroll their browser window. It is estimated that users spend 80% of their time 'above the fold' on a website (NIELSEN, 2010).

To take advantage of this, the search box (See Figure 3.3.3) on all pages and the homepage is presented above the fold. These are functions the user will use regularly so it is important that they are easily accessible.
On the homepage of the site, a number of ‘Call To Actions’ are evident. A ‘call to action’ provokes the user to an immediate response. For example, the search box on the homepage is large and prominent with easy to use buttons. This encourages the user to start searching for listings on the site immediately.

Below the search box, a number of listings on the site are displayed. This shows the user that the site is active and encourages them to click into a listing. The selection of listings on the homepage is randomly selected to ensure it is different each time the user visits. This shows the user the site is constantly being updated with new listings.

Icons have also been used in several locations throughout the site. Icons are visually appealing to the user and can indicate what a button does quicker than text. A sample of some of these icons is shown in D.1.11, p. D.9.

### 3.4 Development of the application

This section will explain the program architecture, the database and the features of the application. Additional technical documentation can also be found in Appendix G.

#### Program Architecture

The final application utilizes a MVC (model view controller) architecture. The purpose of the MVC framework is to separate the representation of information from user interaction (Reenskaug & Coplien, 2009). A MVC architecture breaks code into the following areas:

- The “Model” contains the required logic for updating and retrieving information from the database.
- The “View” contains the code that presents information to the user. The view will often contain a mixture of HTML/CSS/JavaScript as well as PHP output.
- The “Controller” contains the code required to handle user input from the views, fetch information from the model and prepare it to be passed to a view for the user.

A visual representation of MVC is show in Figure 3.4.1
A MVC approach allows the code to be separated into different sections of the programme, making it easier to follow the flow of information through the application. For example, the logic for placing a listing is stored in the “ListingController” whereas the code for registering a user is located in the “UserController”. This clear division of code also helps with maintaining the application and makes it easier to develop additional components in the future as per the projects requirements.

The entire application was developed using XAMPP to run a server locally on a windows machine and Sublime Text Editor 2 for writing the code. As each section of the programme was completed it was thoroughly tested using a web browser with dummy content. Further information on testing can be found in Section 3.5. Once each section was completed it was committed and pushed up to a cloud repository hosted with GitHub.

GitHub is a web based Git repository hosting service. Git allows different versions of software to be saved at regular intervals throughout the development cycle. The primary benefit of this is that each piece of code is tracked, and any modifications done to it can be undone at a later date if they cause issues. GitHub provides a secure location to store code in the cloud so a backup is available in case of a hard drive failure on the developers machine. A screenshot of GitHub can be found in Figure E.5, p.E.4.

**Database Design**

The database is an essential part of the property listing application and is powered by MySQL. The database is used to store all the information in the application. The database follows a logical structure as each database table is linked to a “model” in the application. A model represents an entity such as a “user” or a “listing”. Further details on the relationships in the database can be found by looking at the Entity Relationship Tables and the Entity Relationship Diagram in Appendix H.

The database was designed to be logical, efficient and extendable. Best practice normalization techniques were used. Normalisation minimises data redundancy by efficiently organising the database and ensuring no duplicate data is stored. It improves storage efficiency, data integrity and scalability (Hillyer, 2005). The database was constructed using the Eloquent ORM library that is part of the Laravel framework. This library automatically
generates the appropriate code that is required to query and construct the database. This library also integrates with Git, so it stores a record of database modifications. This allows for changes to the database to easily be undone if needed.

**Application Features**

The main features of the application have already been explained in the system overview in Section 2. This section aims to elaborate on a number of key features and the impact they have on the user experience and the applications performance.

**Placing a Listing**

As previously discussed in section 3.3, it was decided that the process of placing a listing on the site should be done with a single form to make it as quick and easy as possible for the user. In order to place a listing the user is prompted first to choose what type of listing it is (ie. sale, rental or sharing) they are then displayed the price and can then proceed to populating the listings details provided they are signed in. This new listing form is shown in Figure D.1.8, p. D.7.

The form is clear, concise and divided into four sections:

- **Contact Details:** This section is automatically populated from the details stored in the database linked to the logged in user. The application gets the ‘id’ of the current user and looks up their credentials in the database. It then pre-populates the form with the users contact details to reduce the amount of input required. The user can manually override these if they wish.
- **Listing Details:** Each field is provided with a label defining the type of input expected. The input fields have real time validation where possible. For example the area and price input fields accept numeric characters only and will refuse non numeric characters inputted. The validation is performed using JavaScript. An event listener is set on the form input fields. When the user begins to type in one of these fields it triggers the validation function. This function will examine the input and determine if it is correct or not.
- **Choose Location:** This section presents the user with a Google Map of Ireland. The user is requested to navigate to the location of their property on the map and place a pin in the exact location of the property.

The map is restricted to only show Ireland so a user cannot select a location on the map that is outside of Ireland. An event listener is set on the map with JavaScript. As soon as the user clicks a location on the map, it drops a pin. The event listener then triggers a function which passes the GPS co-ordinates of the pin to the Google Maps API. The API then performs a reverse geocode to predict the address of the location on the map. JavaScript is then used to interpret the results of the Google Maps API prediction and it automatically populates the address fields for the user. This greatly improves the accuracy of a listings address but also means the user has less work to do. A user can still manually adjust the address if the application is incorrect. The code used to perform this can be found in Appendix J.1.
• **Upload Photos:** Photos are an integral part of making a listing look visually appealing. JavaScript and Ajax technology has been used to ensure the process of uploading images is quick and easy. A user is able to select multiple images from their computer at a time. The application will then instantly begin to upload the images and provide a progress bar indicating that the upload is in progress.

The images are uploaded in real time using JavaScript and Ajax. Once the images are uploaded to the site, PHP is used to validate that the correct file size and type is uploaded. The images are then resized to three different sizes:

- **Small:** 300px X 250px
- **Medium:** 500px X 400px
- **Large:** 800px width X auto height

The purpose of this is to greatly reduce the file size of the images and reduce the amount of space needed to store them. It also provides significant performance benefits when a user is browsing the site as only images of the appropriate size are displayed to them. Once the images have been resized, the application stores them to the server’s hard drive and saves the location to the database.

Once uploaded, processed and saved, the URL of the image is passed from the application to the front end using Ajax. JavaScript is then used to display the images to the user so they can confirm they are correct. The user also has the option to choose a default image which will be the primary image displayed on the site for that listing. The code for this can be found in Appendix J.2.

Once a user has populated the form and submitted the information, the form is validated with Ajax and JavaScript. If any fields have failed validation the field will be highlighted and the error message displayed to the user. Even if for some reason the user has disabled JavaScript on their machine (which generally indicates a spam bot) there is a fall back PHP validation on the server which cannot be bypassed. This ensures a listing cannot be placed without valid data.

**Searching Listings**

Searching and browsing listings is an important element of the application as this is more likely what users will be doing most on the site. The search bar is slightly different depending on the type of listing, but it functions the same way. A screenshot of the search bar can be found in Figure D.1.1, p. D.1. The search feature allows the user to filter listings by the following:

- Location
- Minimum Price
- Maximum Price
- Minimum Size
- Maximum Size

The user can also dictate the sorting of the search results using the sorting toggle. This can be seen in Figure D.2.3, p. D.12. The sorting toggle has the following options to sort by:
Once a user selects an option, the search results are automatically updated and sorted as requested by the user. The default sorting option is “Most Recent First”.

When a search result is displayed, the application only retrieves 15 listings from the database at a time. The reason this is done is so that it reduces the amount of CPU and memory needed to perform a search. As the number of listings on the application grows, it will take longer and longer to retrieve them all from the database and it will also consume more of the server's resources. This would slow down the application and could eventually crash the server.

When the user is finished browsing the first 15 results, they can load the next 15 by clicking the “load more” button Figure D.1.5, p. D.4 which will use Ajax to retrieve the next 15 listings. Ajax is a JavaScript technology which works behind the scenes to perform an action so a page reload is not required. By using Ajax to retrieve the next set of listings it reduces the amount of time the user has to wait before they are displayed. On the popular residential property sites they require the user to reload the page before the next set of results are displayed which takes time. In this application only the necessary elements needed to display the next set of listings is loaded which is much quicker.

Map View

An important element for the user is being able to see where the property is located in Ireland. To aid with this process the Google Maps API was used to show the location of properties on a map. There is an option to view all the listings for sale or for rent in a map view as some users may prefer to visually interact with the application. The map view of listings for sale is show in Figure D.1.3, p. D.3.

Each listing is represented by a pin on the map. The user can zoom in and out of the map using the toggles provided. In some areas there may be too many listings in one location which clutters up the map and makes it difficult to see. To combat this, the map will automatically cluster together listings in the same area depending on how close the map is zoomed in or out. This can be seen Figure D.1.3, p. D.3.

Clicking on a cluster point will cause the map to automatically zoom in, at this point the individual listing pins will then appear on the map. Clicking a pin will cause the map to zoom in further and show more detail of the listing as seen in the Figure 3.4.2.
Figure 3.4.2 – A listing shown on a Google Map

Search Engine Optimisation

SEO (Search Engine Optimisation) is the process of affecting how visible a site is to a search engine. A website with good SEO is more likely to have traffic directed to it by search engines like Google and Bing. SEO plays a central role in the client’s business as they rely heavily on search engines to direct traffic to their website. In order to drive traffic to the new property listings website, it will need a number of SEO features.

Several features have been built into the application to improve its SEO:

- **Semantic URLs** intend to improve the usability and accessibility of a website’s URL’s. These URL’s are easily read by search engines and greatly improve the website’s SEO. Semantic URL’s have been used throughout the application and are automatically generated from a listings address. An example of this is shown below
  - Non-Semantic URL: http://property.agriland.ie/listing/23
  - Semantic URL: http://property.agriland.ie/land-for-sale/farmholding-miltown-kerry/

- **Image Alternative Text Tags** allow a search engine to understand what a picture represents. Search engines are unable to decipher an image, but by adding an image alt tag it tells the search engine what the image is about. The application ensures that images are all provided with alt tags automatically to improve its SEO.

- **Search Engine Titles** tell the search engine the name of a page. When a user performs a search with a search engine, the search engine is more likely to show a page if the SEO title contains terms or keywords that the user searched for. All search engine titles are automatically added to listing pages on the site to improve their SEO.

Refactoring

One of the requirements for the application was to ensure that it was easy to maintain and extendable. In order to achieve this, it is important that the code is well structured and organised. While developing this application, the code was constantly refactored to ensure it was easy to understand and remove unnecessary complexity. Refactoring is the process of restructuring code, without changing its behaviour. By refactoring the code it is easier to
read, easier to maintain and makes it easier to implement changes to the application (F.Opdyke, 1992).

3.5 Testing
Since the incremental development model was used, the system was tested throughout the development process. At the end of each stage, the system was tested by entering data into the forms and ensuring it was correctly stored in the database. The validation checks in place on forms were also tested by deliberately inputting invalid data to see if the application would detect it and display the correct error message to the user. The tests carried out can be found in Appendix I.

As each stage of the project was completed it was presented to the client. The client and staff then tested the application to ensure all the various links, buttons and forms were working correctly. Any issues that were discovered were noted and corrected before moving to the next stage of the project.

As the site is responsive; the front end was checked at a variety of resolutions to ensure it displayed correctly. This was done by resizing the browser window to simulate the screen on a smaller device and also by checking the site on a mobile device. The main screen sizes targeted were mobiles, tablets and desktop devices. The frontend styling is controlled by CSS so this was adjusted to ensure the design worked correctly at these different screen sizes.

The site was also checked in a variety of different browsers. Most modern browsers will display sites as expected, but some older browsers can sometimes have compatibility issues. The client was able to provide analytics of their current websites traffic. These analytics showed the most common browsers which were used to access the website. Google Chrome was the most popular browser. The site was tested in Google Chrome and no issues were found. The other popular browsers tested were Internet Explorer 11, Internet Explorer 10, Internet Explorer 9 and Mozilla Firefox 35 +. The site was tested in each of these browsers to determine if there were any compatibility issues. There was an issue with Internet Explorer 9. Due to the older technology used in this browser it did not upload files the same way as the other browsers. A fix was provided in the application to allow for this. Apart from this all other elements of the site functioned as expected across all browsers tested.

3.6 Additional Features
A number of additional features have been developed in the application. These are outside of the terms of reference. These have been included to improve the user experience of the site as well as adding additional functionality. These include the following:

Favourite Feature
A favourite feature has been added which allows a user to create a personalised list of favourite listings. These listings are then easily accessible from the user’s dashboard for quick access at a later time.
Contact Form

On each listing a contact form has been built into the page which allows the user to quickly and easily get in touch with the listing owner. The contact form is automatically populated with the user’s details if they are logged in to reduce the amount of input required. The user also has the option to CC (carbon copy) the message to themselves for confirmation. When the contact form is submitted, a message is sent to the listing owner. This is populated with the user’s details, message and the listing they are enquiring about. The contact form can be seen in Figure D.2.5, p. D.13.

Social Sharing Buttons

Buttons for sharing a listing on Facebook and Twitter have been added to each listing. These buttons make use of the social networks sharing features to automatically pre-populate posts. This allows a user to quickly and easily share a listing on their social network profiles. These sharing buttons can be seen in Figure D.1.6, p. D.5.

Print Button

A print button has been provided in various locations throughout the site. This button provides the user with a quick link to print off the page. The application has special print styles to ensure that unnecessary colors and elements are not printed. This also formats the page so it is suitable for printing.

Redirect On Login

The application will automatically redirect users to a login page (As seen in D.2.7, p. D.13) if non-logged in users attempt to access certain areas of the site. When redirected to the login form, a status message will be displayed informing the user to login. Once logged in, the user will then be automatically redirected back to the page they were previously accessing. This application does this by temporarily saving the URL the user was trying to access and redirecting them to this once they’ve logged in successfully. Normally a user would just be redirected back to the homepage or a standard login success page. This improves the users experience as they do not have to re-navigate back to the same page they were attempting to access previously.

3.7 Difficulties Encountered

As with all software development, the project was not without its difficulties. The main difficulties that were encountered are discussed below.

Generating Semantic URL’s:

As discussed in Section 3.4, semantic URL’s were generated from a listings address. However an issue was discovered when two or more listings have the same address, something which is quite possible as two fields could be side by side. If this happened, then two or more listings would have the same Semantic URL. This would mean that only the most recent listing would be shown, effectively over-riding the others in the system.

In order to solve this problem the application generates the semantic URL from the listings address as before. However before it is saved, the application will check the database to see
if there is any other semantic URL’s with the same address. If there is, then the application will append a number to the end to ensure the URL is unique before saving it. This alleviates the problem described above as each semantic URL is now unique.

For example:
Semantic URL 1: my-property-address-wicklow
Semantic URL 2: my-property-address-wicklow-1
Semantic URL 3: my-property-address-wicklow-2

Saving Images:

When processing the uploaded images, it was discovered that if a user uploaded an image which had the same filename as an image that another user had uploaded, then it would overwrite the old image. This would mean that the wrong images would be shown for the wrong listing.

In order to solve this problem, each image that is uploaded has a timestamp attached to the beginning of its filename when it is being saved. This ensures that each image has a unique filename and can’t overwrite another image.

For example: An image with the filename of ‘myphoto.jpg’ is saved as ‘1426370984myphoto.jpg’.  
1426370984 is the timestamp of 03/14/2015 @ 10:09pm (UTC)

Realex Payments:
The code for the payment system has been developed, however it has not been possible to test this feature within the time scales of the project. In order for the payment system to be tested, Realex Payments require the application to be live on the internet so they can verify its validity. Since the client does not wish to set the site live at this time, it has not been possible to fully test the payment system with the Realex Payments API.

The client’s developers are familiar with the Realex Payments API and are confident they will be able to complete this testing process. It is suggested that the client continues to use the MVC approach and familiarise themselves with the technical documentation (Appendix G) when integrating the payments system to retain the applications maintainability.

Mobile Devices

Due to the size of screens on mobile devices, it is difficult to display all the information from the desktop version of the site on a mobile screen. Constant testing on mobile devices was required to check which elements of the site needed to be adjusted to fit on a mobile screen.

As previously discussed in Section 3.3 a number of changes to the responsive design was required to ensure that everything would be displayed correctly on mobile devices.

Additional issues were also encountered with touch screen devices such as mobiles and tablets. As they are touch screen, they do not support hovering like a traditional desktop computer. For example, the gallery of images requires the user to hover over the images to see the arrows for the next image. Since this effect is not possible on touch screen devices, the gallery was modified to allow a user to touch and swipe the images.
4 CONCLUSIONS AND RECOMMENDATIONS

This section discusses the conclusions reached and provides recommendations to the client on the upkeep and maintenance of the application.

4.1 Conclusions

This project involved developing a property listing application for the agriculture industry. This new application will allow the client to extend the range of services they offer to their users and provide the client with an additional source of revenue.

The application developed has exceeded the initial terms of reference and includes some additional functions that were not in the initial requirements (See Section 3.6). The application is built using PHP, HTML, CSS3, JavaScript and MySQL. It allows users to browse listings and place a new listing on the site. It also provides an administrator panel for administrators to manage listings and users. The application is fully responsive and works on mobile, tablet and desktop devices (See Section 3.3).

The system has been rigorously tested and all forms have validation where required. The client requested that the application would be easy to maintain and extend. This has been ensured by following the MVC architecture and refactoring the code throughout (See Section 3.4). Technical documentation can also be found in Appendix G and an administrator manual is attached as separate document.

As this application is the first of its kind for the industry it was a challenging and interesting project. The project incorporated a wide range of material that is covered in the Management Science and Information Systems Studies course. Valuable skills learnt from Management Science, Programming, Software Engineering, Software Applications, Information Systems and Human Factors were all used during the course of this project.

4.2 Recommendations

The following is a list of recommendations for the future development and upkeep of the website:

Code Structure

As previously discussed in Section 3.4, the application has been developed using a MVC architecture. In order to ensure that the program is easy to maintain when new functionality is added, all new development should follow this structure. Developers should also be sure to refactor any new code to keep the application well-structured and organised. It is also advised that developers continue to use Git to keep track of changes to the code and provide a way to rollback changes if need be.

A/B Testing

The application has been designed to be as simple as possible for the user to use (See Section 3.3). However there are always additional improvements that can be made to the user interface. It is recommended that the client tries A/B testing to identify changes to the user interface which could increase sales. A/B testing is a simple way of testing how changes to the current design of a page may affect sales. It involves presenting a portion of
live traffic with the control design (A) and the rest receive the test design (B). Each design is then tracked to see which has the biggest impact on some outcome variable such as sales.

Security

Currently there is a minimum password length requirement in the application, but this can be improved. It is recommended that the client enforces a minimum password length with a combination of uppercase and lowercase characters for administrator accounts. The administrator accounts have access to all areas of the website as well as users personal details, so it is important that these accounts cannot be easily breached. The client should also look at establishing additional user account access levels. As the application grows the client may wish to restrict administrator access to only certain types of listings, for example - sales listings.

It is also recommended that a SSL certificate is installed on the server when the application is made live. An SSL certificate will allow to site to operate with the https:// extension. This will mean that any information transferred will be encrypted and provide additional security to the users of the website. Though not legally required (as no credit card details are stored in the database), the client could also look into PCI compliance to maximise data security.

Location Based Features

As the users of the application will often be searching for listings in their current vicinity, location based features could be added to improve the user experience. For example when a user lands on the homepage of the website, the application could detect the current users location and present them with listings in their current area. This would help improve the user experience of the site even further.

Content Distribution Network

The client hopes to extend the services with features such as an API when it becomes more popular. As the application grows in size and as the number of listings increases, this will start to consume storage space on the server. It will also place additional strain on server resources during periods of high traffic. To help the application to scale and grow, all images could be stored on a content distribution network (CDN). A CDN is a network of servers provided by a third party which are optimised to store files and serve them as quickly as possible. This will reduce the number of images that will need to be stored on the main server and reduce the amount of resources needed. There are a variety of CDN providers such as Cloudflare, MaxCDN, Amazon Web Services and Akamai.

Handling API Error

The application does not currently have a system in place to handle errors that may occur if an API is offline. It is recommended that a procedure is put in place to display a temporarily unavailable message to users if a key API is not available like the Google Maps API or the Realex Payments API. This will help maintain a positive user experience and prevent the user from seeing error messages when they are trying to use the application.
TRINITY COLLEGE DUBLIN
Management Science and Information Systems Studies
Project Report

AGRILAND MEDIA
Development of an Online Property Listing
Web Application
Appendices
A  ORIGINAL PROJECT OUTLINE

Client: Agriland Media Ltd  
Project: Online Land Listings  
Location: Unit 19 Docklands Innovation Park, East Wall Road, Dublin 3  
Client Contact: Cormac Farrelly  Email: Cormac@agriland.ie  Phone: + 353 01 449 8103

Client Background:

Agriland Media is Ireland’s first dedicated online agriculture portal. Founded just over a year ago, Agriland has experienced rapid growth to become Ireland’s largest online agriculture publication with over 60,000 unique visitors and 160,000 pageviews a month. Agriland has also provided the first ecommerce platform for the agriculture industry to sell their products online as well as its own online agriculture store.

Project Background:

Agriland have identified a demand for a dedicated online platform to allow the agriculture industry to list land/property available for sale. Currently the agricultural sector is very under-represented in this area, and Agriland want to expand their online offering to help achieve our vision of bringing the power of the internet to the farming community in Ireland.

Client Requirement:

Agriland is now seeking assistance establishing requirements, architecting and developing this online platform. The new platform should integrate with the existing agriland.ie portal and allow users of the website to quickly and easily list land/property for sale, or to allow them to easily browse and filter available listings.

What is involved for the student?

This project will involve the student working with the marketing department and directly with the CEO of Agriland. It will provide the student with the opportunity to go through all the phases of planning and developing an online to realise the requirements of the marketing team. The project will give the student experience in a fast moving start up where their contribution will have direct input into the strategy of the business. The application will require the student to prepare wireframes, state diagrams etc... in order to prepare a feature requirements document and to devise a timeline with key milestones outlined for the development of the application. The application will have two components. An interactive responsive front end design, built using the latest HTML5/CSS3 technologies which will be enhanced with Javascript and make use of various public web based API’s. The backend will be powered with a LAMP (Linux/ Apache/ MYSQL /PHP) stack or another similar solution based on the students research. The student will also need to come up with a solution to integrate the new application with the current Agriland.ie platform.
INTERIM REPORT

Project: Online Land Listings
Client: Agriland Media Ltd
Student: Robert Boland
Supervisor: Aideen Keaney

Review of Background

Agriland Media launched in 2012 with the aim of bringing the power of the internet to the agricultural industry. Since then they have become the largest online publication of agricultural news and resources with close to 100,000 unique visitors every month.

Agriland have identified a demand for a dedicated online platform to allow the agriculture industry to list land/property available for sale. Currently there is no online platform dedicated to listing agricultural land/property in Ireland. Existing services are targeted towards general consumers for the sale/rental of residential and commercial property. Agriland want to expand their online offering to include a dedicated platform for the agricultural industry where users can list land/property available for sale, to rent or to share. The web application will allow users to login, upload their listings, pay the listing fee and manage their listings on the site. Users will also be able to browse listings, filter by type, county and other attributes.

Terms of Reference

- Define the feature requirements and wireframes of the web application for the property/land listings
- Develop the web application as per the requirements
- Integrate online payments
- Ensure a user friendly interface to allow users to browse and place listings on the site.
- Provide necessary technical documentation and user manuals for the administrators

Further Work

Research will be carried out to establish if Agriland’s existing platform will be sufficient to build the new web application on, or if an alternative platform is needed. Analysis of various different programming and development frameworks will be used to decide the optimum platform for the application.

In order to keep up to date with current trends, the site will be developed to be responsive across all devices, ensuring a positive user experience across mobile, tablet and desktop devices. Extensive testing of the website will be carried out throughout to ensure it is fully functional before deployment.

Work to Date

As of November 2014, a number of meetings have been carried out with the client to identify and establish the requirements for the web application. A formal feature requirements document has been drawn up outlining user scenarios and the features of the application.
ADMINISTRATOR MANUAL

Please see the administrator manual accompanying this report. The administrator manual details the various administrator functions available in the application.

An electronic copy is also provided in the CD at the back of this report.
D  APPLICATION SCREENSHOTS

D.1  Desktop Version of the Application

These screenshots were taken on a desktop device with a large screen size.

Figure D.1.1 – The homepage
Figure D.1.2 – Listings for sale (List View)
Figure D.1.3 – Listings for sale (Map View)

Figure D.1.4 – Zoomed in view of map
Figure D.1.5 – The ‘Load More’ button that loads additional listings via Ajax. This prevents a page refresh.
Figure D.1.6 – How a listing looks on the site
Figure D.1.7 – Placing a new listing on the site
Figure D.1.8 – The form for placing a new listing
Figure D.1.9 – The admin bar to manage a listing
Figure D.1.10 – The administrator panel dashboard

Figure D.1.11 – A selection of the icons used in the application
Figure D.1.12 – The sharing opportunities available on the site

Figure D.1.13 – The form to submit an interest in a sharing opportunity
D.2 Mobile Version of the Application

These screenshots were taken on a mobile device.

Figure D.2.1 – Homepage of the application

Figure D.2.2 – The listings results page
Figure D.2.3 – The sorting options on the listings results page

Figure D.2.4 – The listing page
Figure D.2.5 – The enquiry form for a listing

Figure D.2.6 – The location of a listing on a map
Figure D.2.7 – The login form
E  DESIGN DOCUMENTATION

This section provides activity flow diagrams and data flow diagrams. These illustrate how a user proceeds through the application and the flow of information.
Figure E.1 – Activity diagram for placing a listing

Figure E.2 – Activity diagram for registering a new user

Figure E.3 – Activity diagram for viewing a listing
Figure E.4 – Data flow diagram for user registration and viewing a listing
Table E.1 – Comparison of PHP frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Laravel</th>
<th>CakePHP</th>
<th>Symfony</th>
<th>Codeigniter</th>
<th>Zend Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Release Date</td>
<td>March 14</td>
<td>May 06</td>
<td>October 05</td>
<td>February 06</td>
<td>May 13</td>
</tr>
<tr>
<td>Requirements</td>
<td>&gt;= 5.3.7 with mcrypt extension</td>
<td>&gt;= PHP 5.2.8</td>
<td>&gt;= PHP 5.3.2</td>
<td>&gt;= PHP 5.1.6</td>
<td>&gt;= PHP 5.3</td>
</tr>
<tr>
<td>Template System</td>
<td>Blade, PHP</td>
<td>Smarty, Custom</td>
<td>PHP, Twig</td>
<td>PHP</td>
<td>None</td>
</tr>
<tr>
<td>Cache System</td>
<td>File System, Database, Memcache, APC, Redis, Xcache, WinCache, Memory (Arrays)</td>
<td>Memcache, APC File, APC, Memcache, Redis</td>
<td>APC, eAccelerator, XCache</td>
<td>File, apc, memcached, xcache</td>
<td>Zend_cache</td>
</tr>
<tr>
<td>Architecture</td>
<td>MVC</td>
<td>MVC</td>
<td>MVC</td>
<td>MVC</td>
<td>MVC</td>
</tr>
</tbody>
</table>

Figure E.5 – A screenshot of GitHubs commit history
Figure E.6 – The incremental model
**USER JOURNEYS**

A user journey is the experience a person has when using a piece of software. User journeys were performed on [www.myhome.ie](http://www.myhome.ie) and [www.daft.ie](http://www.daft.ie) to see if any of these journeys could be adjusted to improve the user experience.

The following user journeys were carried out:

**F.1 Registering / Signing Up**

The goal of this user journey was to register a new account with the site.

---

**Daft.ie**

Figure F.1.1 - The homepage of daft.ie has a quick and easy to access link to register. This link is in a prominent location above the fold and was easy to find.

Figure F.1.2 - The registration form has an almost overwhelming number of fields. There is no indication that the fields are all required, the user is only informed when attempting to
sign up. Having a username also seems unnecessary and is something a user is likely to forget.

Figure F.1.3 - Once the form is completed, the signup process is finished. However the page displayed is not very appealing and looks like it’s broken.

**Daft.ie - Conclusion**

The signup process only took 3 steps and was easy to start. However the signup form required a lot of details which may discourage users from signing up. It may be better to ask a user to provide these details after they have signed up.

When a signup is completed, the page presented to the user is not very appealing and looks like it’s broken. It would be better to redirect the user to the homepage or another visually appealing page.

Instead of requiring a username to login, it would be better if the user could use the email they registered with. It is much easier to remember an email address than a username, as someone may have multiple different usernames but only one or two email addresses.
MyHome.ie

Figure F.1.4 - The homepage of myhome.ie has a quick and clear link to create a new account. It is in a prominent location above the fold and is very easy to find.

Figure F.1.5 - The signup form is very short and requires just three fields. It’s quick and easy to fill in this form.
Once the form has been completed, the registration process appears to be complete. The page presented has a number of calls to actions. However there is no message confirming the registration.

MyHome.ie - Conclusion

The registration process on myhome.ie was quick and easy to start. The sign up form required only the basic information. However the page presented after sign up did not show any indication that the registration process was successful.

Additionally, before you can proceed any further on the site, you are requested to verify your mobile number, an extra step that is not initially apparent.
F.2  **Browse and Searching Listings**

The goal of this user journey was to see how easy it would be to browse and search listings on Daft.ie.

![Image of Daft.ie homepage](image1)

**Figure F.2.1** - The homepage of daft.ie has a number of links to begin browsing listings available on the site. A prominent search box is also placed on the home page above the fold.

![Image of Daft.ie search page](image2)

**Figure F.2.2** - However clicking any of the links in the navigation bar brings you to a search page again. The user has to input search criteria or press the search button a second time before any results are presented.
Figure F.2.3 - After a search is performed, initially it is not very clear that anything has happened. The search results are presented very far down the page, below the fold depending on the size of the users screen. Too much space has been given to advertising and other features.

Daft.ie - Conclusion

The additional step in the search form is frustrating and makes the user think they have to enter search criteria.

Too much space has been given to advertising and other buttons on the search results page. It makes it very difficult to see the listings as they are placed below the fold of the website, even though these are what the user is looking for.
The homepage of myhome.ie has a number of clear search forms and links to being searching and browsing listings.

Figure F.2.4 - The homepage of myhome.ie has a number of clear search forms and links to being searching and browsing listings.

Figure F.2.5 - Clicking the search button brings the user directly to a listing of results. It is quite clear where the search results are, and the search bar is present across the top of the page to allow the user to filter the results further if they wish.

**MyHome.ie - Conclusion**

The homepage of the website has numerous buttons which will initiate the search process. Only one click is needed to view search results. The search results are easy to see and prominently placed on the user screen, making it a quick and simple process.
F.3 Placing A Listing

The goal of this user journey was to place a listing on the site Daft.ie

Figure F.3.1 - A clear button to advertise your listing is located in the top right corner in a prominent location. This makes it easy to start the listing process. The user is then asked to choose the type of property and what type of listing plan they would like.

Figure F.3.2 - There is then a number of steps required to complete the listing, 5 in total.
Figure F.3.3 - The contact details form has automatically used the information stored with the users account. A map is presented, but rather confusingly you cannot move or interact with it. Instead you manually have to choose your county, area and type in your address.
Figure F.3.4 - The details of the property are then requested. This is a very long form and it is difficult to know where to start as there is buttons and icons everywhere in the user’s line of vision.

Daft.ie - Conclusion

It is easy to begin the process of placing a new listing on the site. The button is prominent on all pages of the site. There are clear instructions for all the forms; however the forms themselves are quite long and difficult to follow. There are multiple forms to complete which makes the process seem longer.
Figure F.3.5 - It is difficult to start the process of placing a listing on myhome.ie. There is only a very small link located halfway down the page of the site which is difficult to find.

Figure F.3.6 - The user is then requested to choose the type of listing and the plan they would like to place.
Figure F.3.7 - At this stage the user is requested to verify their mobile phone number.

Figure F.3.8 – Once the user has verified their mobile number, they are presented with the property details form. This form is well laid out and clear. The map allows the user to interact with it, but it does not allow the user to select the address. Instead it requires the user to manually input the address, and the map will update to show the area.
Figure F.3.9 – Once the user has completed the details form, they are then presented with a second form to upload images to their listing. Once this process is complete then the user can pay for their advert.

MyHome.ie - Conclusion

It is very difficult to find the initial button to place a listing on myhome.ie. It is not prominent on all pages. The verification of phone number is inconvenient for those who do not have a mobile number as they are required to email the administrator manually. The form to enter the listing details and address is user friendly and clear. The photo uploading tool worked well but provided little user feedback when images were being processed.
G TECHNICAL DOCUMENTATION

This section contains the required technical documentation for the application. It provides information on what should be done when deploying the application. It also provides information on the different code libraries and tools that were used when developing the application.

G.1 Server Requirements and Application Deployment

The property listing application has been developed using Laravel Framework Version 4.2. Extensive documentation for the framework can be found on the Laravel website - http://laravel.com/docs/4.2.

This application has the following server requirements:

- PHP >= 5.4
- Mcrypt PHP extension

The Mcrypt extension is required for encrypting a user's password and must be installed in order for the application to function correctly.

It is recommended that the application is deployed using Git. This will make it easier to manage the different versions of the application.

The application will need a MySQL database configured with full privileges granted to the database user. The configuration details for the MySQL database should be updated in the ‘app/config/database.php’ file so that the application can communicate with the database.

Once the database connection has been established, the necessary database tables can be created automatically by the application by running the ‘php artisan migrate’ command from the command line interface. Alternatively the database tables can be configured manually following the schema outlined in Appendix F.

When deploying the application, ensure that all error messages are logged to an error log. The debug setting on the application should also be disabled. This can be set in the app.php file located at ‘app/config/app.php’.

404 and Offline pages should be hosted on an external server so that they are still accessible if the server is unavailable.

A .htaccess file is required in the root directory of the application to enable pretty URLs. If pretty URL’s are not working with the current .htaccess files, a new one can be created with the following details:

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^index.php [L]
```
G.2 Application Development

Coding Standards and Naming Conventions

The following coding standards and naming conventions have been used in this application:

- Variable naming – all variables follow the camel case naming convention. I.e. ‘myNewVariable’.
- Function naming – all functions follow the camel case naming convention. I.e. ‘myNewFunction’.
- Variable and function names have been named to describe their purpose as much as possible.
- Code has been refactored where possible.
- Models are used for storing getter and setter methods.
- All dates are saved as either a UNIX timestamp or YYYY-MM-DD in the database. These are then modified to DD-MM-YYYY when displayed on the front end to the user.
- Comments have been left in code where it was deemed necessary.
- Database lookup statements make use of the ‘FindOrFail’ feature in Laravel so errors can be caught.
- Unobtrusive JavaScript has been used where possible, with most of the JavaScript files being loaded in the footer of the website. In some cases inline JavaScript was required (Google Maps API) but this has been only included on the relevant pages.

Application Structure

The application has been designed using a MVC architecture. Any future development of the application should follow this practice to ensure the code is maintainable and extendable.

The code for the application has been refactored extensively, with getter and setter methods used for models wherever possible. Developers should check to see if any getter and setter methods already exist or can be extended before writing new functions.

The applications views make use of the blade template syntax that is included with the Laravel framework. This syntax should be used where possible in views to ensure consistency and take advantage of the tools it provides. Further information on the blade template syntax can be found at http://laravel.com/docs/4.2/templates#blade-templating.

View partials have been used where possible to make it quicker and easier to build application views. Partials should be used as much as possible throughout the application as they make it much easier to make changes to the applications front end if needed. All partial files follow conventional naming practice by prepending an underscore to the filename ie. _list-view.blade.php.

CSS Styling

The applications CSS styling makes use of the Zurb Foundation framework. The Foundation framework provides the CSS grid system and some utility styling classes. This framework
should be followed for future development so that the CSS is maintainable. More information on the Foundation framework can be found at http://foundation.zurb.com/.

SASS has been used to write the CSS for this application. Foundation has also been integrated with SASS to allow for bulk changes to the applications CSS if needed. The SASS files can be found in 'public/assets/stylesheets'. The relevant SASS files end with the .scss extension. More information on SASS can be found at http://sass-lang.com/.

The application makes use of Grunt. The command grunt watch should be run when editing the SASS files. This command will automatically compile the SASS files into CSS files when they are edited. The Grunt configuration file can be found in the root directory. More information on Grunt can be found at http://gruntjs.com/

**JavaScript**

A combination of pure JavaScript and the jQuery JavaScript library have been used in this application. More information on the jQuery library can be found at http://jquery.com/.

Additional JavaScript libraries used in this application can be found in the 'public/assets/vendor' directory. An overview of these JavaScript libraries is explained below:

- **jQuery File Upload** – Responsible for enabling Ajax image uploads for the application. More information can be found at https://blueimp.github.io/jQuery-File-Upload/
- **Fastclick** - The fastclick JavaScript library is used to speed up the response time of the application on mobile devices. More information can be found at https://github.com/ftlabs/fastclick
- **Foundation** – The JavaScript files that come with the foundation framework have also been included with this application. More information on these files can be found at http://foundation.zurb.com/docs/
- **Modernizr** – Modernizr is a polyfill library used to enable HTML5 and CSS3 technologies on browsers that do not support these technologies. More information can be found at http://modernizr.com/
- **Galleria** - The galleria library is used for the responsive sliders found on the listings page of the application.

The custom JavaScript that has been written can be found in 'public/assets/javascript/custom.js'. This file contains the JavaScript required to interact with the Google Maps API and the JavaScript libraries mentioned previously.

**Mandrill API**

The Mandrill API is used for sending emails from the application. The configuration details for the API can be found in ‘app/config/mail.php’ and ‘app/config/services.php’.
Figure H.1 – Entity relationship diagram for the property listing application

Table H.1

<table>
<thead>
<tr>
<th>Application Design Worksheet Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table Name:</strong> Listings</td>
</tr>
<tr>
<td><strong>Brief Description:</strong> Stores the details of the listings in the application</td>
</tr>
<tr>
<td><strong>Related Tables:</strong> Users, Sales, Favourites, Photos</td>
</tr>
<tr>
<td><strong>Data Name</strong></td>
</tr>
<tr>
<td>id (Primary Key)</td>
</tr>
<tr>
<td>created_at</td>
</tr>
<tr>
<td>updated_at</td>
</tr>
<tr>
<td>user_id</td>
</tr>
<tr>
<td>Field</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>title</td>
</tr>
<tr>
<td>status</td>
</tr>
<tr>
<td>listingType</td>
</tr>
<tr>
<td>listed</td>
</tr>
<tr>
<td>price</td>
</tr>
<tr>
<td>description</td>
</tr>
<tr>
<td>townland</td>
</tr>
<tr>
<td>town</td>
</tr>
<tr>
<td>latitude</td>
</tr>
<tr>
<td>longitude</td>
</tr>
<tr>
<td>county</td>
</tr>
<tr>
<td>Province</td>
</tr>
<tr>
<td>directions</td>
</tr>
<tr>
<td>sold</td>
</tr>
<tr>
<td>structures</td>
</tr>
<tr>
<td>residential</td>
</tr>
<tr>
<td>ber</td>
</tr>
<tr>
<td>address1</td>
</tr>
<tr>
<td>sale_type</td>
</tr>
<tr>
<td>size</td>
</tr>
<tr>
<td>contact_name</td>
</tr>
<tr>
<td>contact_number</td>
</tr>
<tr>
<td>contact_email</td>
</tr>
<tr>
<td>default_photo</td>
</tr>
<tr>
<td>Field</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>slug</td>
</tr>
<tr>
<td>contact_times</td>
</tr>
<tr>
<td>views</td>
</tr>
<tr>
<td>rent_available_from</td>
</tr>
<tr>
<td>minimum_lease</td>
</tr>
<tr>
<td>rent_rate_timeframe</td>
</tr>
<tr>
<td>rent_period_rate</td>
</tr>
<tr>
<td>agent</td>
</tr>
<tr>
<td>enterprise</td>
</tr>
</tbody>
</table>
### Table H.2

#### Application Design Worksheet Tables

<table>
<thead>
<tr>
<th>Table Name:</th>
<th>Favourites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description:</td>
<td>Stores the details of the listings in a user's favourites list</td>
</tr>
<tr>
<td>Related Tables:</td>
<td>Users, Listings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id (Primary Key)</td>
<td>Int(10)</td>
<td>Unique ID of a favourite item</td>
<td>Unique</td>
<td></td>
</tr>
<tr>
<td>listing_id</td>
<td>Int(11)</td>
<td>The ID of the Listing associated with the favourite</td>
<td>Must be an integer</td>
<td>Listings</td>
</tr>
<tr>
<td>user_id</td>
<td>Int(11)</td>
<td>The ID of the User associated with the favourite</td>
<td>Must be an integer</td>
<td>Users</td>
</tr>
<tr>
<td>created_at</td>
<td>Timestamp</td>
<td>Time the favourite item was created</td>
<td>Timestamp format</td>
<td></td>
</tr>
<tr>
<td>updated_at</td>
<td>timestamp</td>
<td>Time the favourite item was updated</td>
<td>Timestamp format</td>
<td></td>
</tr>
</tbody>
</table>

### Table H.3

#### Application Design Worksheet Tables

<table>
<thead>
<tr>
<th>Table Name:</th>
<th>Password Reminders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description:</td>
<td>Stores the details of the password reminders</td>
</tr>
<tr>
<td>Related Tables:</td>
<td>Users</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>Varchar(255)</td>
<td>Email address the reminder token is for</td>
<td>Unique, Must be less than 255 characters</td>
<td>Users</td>
</tr>
<tr>
<td>token</td>
<td>Varchar(255)</td>
<td>Auto generated token for resetting password</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>timestamp</td>
<td>Time the password reset was requested</td>
<td>Timestamp format</td>
<td></td>
</tr>
</tbody>
</table>

### Table H.4

#### Application Design Worksheet Tables

<table>
<thead>
<tr>
<th>Table Name:</th>
<th>Photos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description:</td>
<td>Stores the details of the photos uploaded to the application</td>
</tr>
<tr>
<td>Related Tables:</td>
<td>Listing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>id (Primary Key)</td>
<td>Int(10)</td>
<td>ID of the photo</td>
<td>Unique integer</td>
<td></td>
</tr>
<tr>
<td>listing_id</td>
<td>Int(11)</td>
<td>ID of the listing linked to the photo</td>
<td>Integer</td>
<td>Listings</td>
</tr>
<tr>
<td>Data Name</td>
<td>Data Type</td>
<td>Description</td>
<td>Validation Rule</td>
<td>Linked To</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-------------------------------------------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>title</td>
<td>Varchar(255)</td>
<td>Title of the photo</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>Timestamp</td>
<td>When the photo was created</td>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>updated_at</td>
<td>Timestamp</td>
<td>When the photo was updated</td>
<td>Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

Table H.5

**Application Design Worksheet Tables**

**Table Name:** Photo Sizes

**Brief Description:** Stores the details of different sizes of photos uploaded

**Related Tables:** Photos

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>id (Primary Key)</td>
<td>Int(10)</td>
<td>ID of the photo size</td>
<td>Unique</td>
<td></td>
</tr>
<tr>
<td>original_photo</td>
<td>Int(11)</td>
<td>The original photo ID</td>
<td>Integer, Required</td>
<td>Photos</td>
</tr>
<tr>
<td>url</td>
<td>Text</td>
<td>The URL of the photo</td>
<td>Text, Required</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>Text</td>
<td>The size of the photo</td>
<td>Text, Required</td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>Timestamp</td>
<td>The time the photo size was created</td>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>updated_at</td>
<td>Timestamp</td>
<td>The time the photo size was updated</td>
<td>Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

Table H.6

**Application Design Worksheet Tables**

**Table Name:** Sales

**Brief Description:** Stores the details of the listings that have been marked as sold

**Related Tables:** Users, Listings

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Int(10)</td>
<td>The ID of the sale</td>
<td>Unique</td>
<td></td>
</tr>
<tr>
<td>listing_id</td>
<td>Int(11)</td>
<td>The ID of the listing it is linked to</td>
<td>Integer</td>
<td>Listing</td>
</tr>
<tr>
<td>user_id</td>
<td>Int(11)</td>
<td>The ID of the users whose sale it is</td>
<td>Integer</td>
<td>Users</td>
</tr>
<tr>
<td>price</td>
<td>Double</td>
<td>The price of the sale</td>
<td>Must be a double</td>
<td></td>
</tr>
<tr>
<td>soldAt</td>
<td>Datetime</td>
<td>The date it was sold at</td>
<td>Date format</td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>Timestamp</td>
<td>The timestamp the sale was recorded</td>
<td>Timestamp format</td>
<td></td>
</tr>
<tr>
<td>updated_at</td>
<td>Timestamp</td>
<td>The timestamp of when the sale was updated</td>
<td>Timestamp format</td>
<td></td>
</tr>
</tbody>
</table>
Table H.7

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Int(10)</td>
<td>ID of the sharing record</td>
<td>Unique</td>
<td></td>
</tr>
<tr>
<td>user_id</td>
<td>Int(11)</td>
<td>ID of the User associated with the sharing</td>
<td>Integer</td>
<td>Users</td>
</tr>
<tr>
<td>title</td>
<td>Text</td>
<td>Title of the sharing interest</td>
<td>Must be text</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>Varchar(255)</td>
<td>The status of the sharing opportunity</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>Text</td>
<td>The description of the sharing opportunity</td>
<td>Must be text</td>
<td></td>
</tr>
<tr>
<td>county</td>
<td>Text</td>
<td>The county of the sharing opportunity</td>
<td>Must be text</td>
<td></td>
</tr>
<tr>
<td>enterprise</td>
<td>Text</td>
<td>The type of enterprise</td>
<td>Must be text</td>
<td></td>
</tr>
<tr>
<td>contact_number</td>
<td>Text</td>
<td>The contact number</td>
<td>Must be text</td>
<td></td>
</tr>
<tr>
<td>contact_email</td>
<td>Text</td>
<td>The contact email</td>
<td>Must be text, Email, Required</td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>Timestamp</td>
<td>The timestamp of when the sharing opportunity was created</td>
<td>Timestamp format</td>
<td></td>
</tr>
<tr>
<td>updated_at</td>
<td>Timestamp</td>
<td>The timestamp of when the sharing opportunity was updated</td>
<td>Timestamp format</td>
<td></td>
</tr>
</tbody>
</table>

Table H.8

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Int(10)</td>
<td>Unique id for the user</td>
<td>Unique, integer</td>
<td>Listings, Favourites, Sales</td>
</tr>
<tr>
<td>firstname</td>
<td>Varchar(255)</td>
<td>A users first name</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>lastname</td>
<td>Varchar(255)</td>
<td>A users last name</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>group</td>
<td>Int(11)</td>
<td>What user group they belong to</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>company</td>
<td>Varchar(255)</td>
<td>Company name</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
<td>Constraints</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td>Varchar(255)</td>
<td>Users phone number</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>email</td>
<td>Varchar(255)</td>
<td>Users email</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>Varchar(255)</td>
<td>Users encrypted password</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>remember_token</td>
<td>Varchar(255)</td>
<td>Session token to auto login a user</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>lastip</td>
<td>Varchar(255)</td>
<td>The users last ip address</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>created_at</td>
<td>Timestamp</td>
<td>The time the user was created</td>
<td>Timestamp format</td>
<td></td>
</tr>
<tr>
<td>updated_at</td>
<td>Timestamp</td>
<td>The time the user was updated</td>
<td>Timestamp format</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>Text</td>
<td>Users description for their profile</td>
<td>Text format, nullable</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>Text</td>
<td>Url of their profile image</td>
<td>Text format, nullable</td>
<td></td>
</tr>
<tr>
<td>address</td>
<td>Text</td>
<td>Users address</td>
<td>Text format, nullable</td>
<td></td>
</tr>
<tr>
<td>facebook</td>
<td>Varchar(255)</td>
<td>Users facebook page</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>twitter</td>
<td>Varchar(255)</td>
<td>Users twitter page</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>website</td>
<td>Varchar(255)</td>
<td>Users website</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>fax</td>
<td>Varchar(255)</td>
<td>Users fax number</td>
<td>Must be less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>slug</td>
<td>Varchar(255)</td>
<td>The users unique slug for semantic url</td>
<td>Unique and less than 255 characters</td>
<td></td>
</tr>
</tbody>
</table>
Table H.9

<table>
<thead>
<tr>
<th>Table Name:</th>
<th>Payments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description:</td>
<td>Stores the details of the payments processed in the application</td>
</tr>
<tr>
<td>Related Tables:</td>
<td>Users, Listings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
<th>Description</th>
<th>Validation Rule</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Int(10)</td>
<td>Unique id</td>
<td>Unique, Integer</td>
<td></td>
</tr>
<tr>
<td>User_id</td>
<td>Int(11)</td>
<td>Id associated with the user who made the payment</td>
<td>Integer</td>
<td>Users</td>
</tr>
<tr>
<td>Listing_id</td>
<td>Int(11)</td>
<td>The ID associated with the listing the payment is for</td>
<td>Integer</td>
<td>Listings</td>
</tr>
<tr>
<td>Status</td>
<td>Varchar(255)</td>
<td>The status of the payment</td>
<td>Less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>Double</td>
<td>The amount the payment is for</td>
<td>Must be a double</td>
<td></td>
</tr>
<tr>
<td>Transactionid</td>
<td>Varchar(255)</td>
<td>The transaction ID for the payment</td>
<td>Less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>PaymentMethod</td>
<td>Varchar(255)</td>
<td>The payment method that was used for the transaction</td>
<td>Less than 255 characters</td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td>Boolean</td>
<td>If the payment has been completed or not</td>
<td>1 or 0</td>
<td></td>
</tr>
<tr>
<td>Created_at</td>
<td>Timestamp</td>
<td>The time the payment was created at</td>
<td>Timestamp format</td>
<td></td>
</tr>
<tr>
<td>Updated_at</td>
<td>Timestamp</td>
<td>The time the payment was updated at</td>
<td>Timestamp format</td>
<td></td>
</tr>
</tbody>
</table>
This appendix gives an overview of the tests that were undertaken on the application during development and on the final application. These tests can be re-performed to ensure that the application is functioning correctly.

Table I.1 – The tests conducted and the expected results

<table>
<thead>
<tr>
<th>Test</th>
<th>Testing Procedure</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register a new account</td>
<td>Attempt to register a new account by entering a valid email address and password</td>
<td>The application should accept the registration and display a success message to the user. A confirmation email should also be sent to the registered email address.</td>
</tr>
<tr>
<td>Register a new account with</td>
<td>Attempt to register an account with an invalid email address, blank password,</td>
<td>The application should reject the registration and display the reasons why the registration failed.</td>
</tr>
<tr>
<td>incomplete data</td>
<td>unmatching confirmation password and a previously registered email address</td>
<td></td>
</tr>
<tr>
<td>Login</td>
<td>Enter valid login credentials</td>
<td>The application should log the user in and display a success message.</td>
</tr>
<tr>
<td>Login with invalid credentials</td>
<td>Enter invalid login credentials</td>
<td>The application should not log the user and show an error message that the credentials are incorrect.</td>
</tr>
<tr>
<td>Place A New Listing</td>
<td>Attempt to place a new listing without logging into the application</td>
<td>The application should redirect the user to the login page and display a message requesting them to login.</td>
</tr>
<tr>
<td>Place A New Listing (Unauthorised)</td>
<td>Attempt to place a new listing after logging into the application</td>
<td>The user should be presented with the new listing form.</td>
</tr>
<tr>
<td>Attempt to preview listing</td>
<td>Click the “Preview Listing” button without filling in any information to the form</td>
<td>The application should not proceed to the preview listing stage. It should display an error message and highlight the fields that are invalid.</td>
</tr>
<tr>
<td>without completing the form</td>
<td>Partially complete the form and click the “Preview Listing” button</td>
<td>The application should not proceed to the preview listing stage. It should display an error message and highlight the fields that are invalid.</td>
</tr>
<tr>
<td>Upload a large image</td>
<td>Upload an image larger than 10mb in size</td>
<td>The application should reject the image and display an error that the file is too large.</td>
</tr>
<tr>
<td>Upload an unsupported file</td>
<td>Upload a file that is not a jpeg, png or gif</td>
<td>The application should reject the image and display an error that the file type is unsupported.</td>
</tr>
<tr>
<td>Upload a supported file</td>
<td>Upload a file that is less than 10mb in size</td>
<td>The application should accept the file and display a success message.</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
<td>Expected Outcome</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Attempt to make a payment</td>
<td>Enter valid credit card details at the payment stage</td>
<td>The system should accept the payment, display a confirmation message, set the listing live and send an email confirmation</td>
</tr>
<tr>
<td>Attempt to make a payment with invalid details</td>
<td>Enter invalid credit card details at the payment stage</td>
<td>The system should reject the payment and display a reason why the payment failed</td>
</tr>
<tr>
<td>Logout</td>
<td>Click the logout button on the website</td>
<td>The application should log the user out and display a confirmation message</td>
</tr>
<tr>
<td>Reset Password</td>
<td>Attempt to reset a password by inputting a valid email address</td>
<td>The system should inform the user to check their email for a reset link. An email should be sent to the user. Clicking this link should allow the user to enter a new password</td>
</tr>
<tr>
<td>Reset Password with false address</td>
<td>Attempt to reset a password by inputting an invalid email address</td>
<td>The application should display an error message that the user could not be found</td>
</tr>
<tr>
<td>View Listings For Sale</td>
<td>Click the link to view the listings for sale</td>
<td>All the listings for sale on the application should be displayed. If there are no listings available a message should be displayed explaining this</td>
</tr>
<tr>
<td>View Listings For Rent</td>
<td>Click the link to view the listings for rent</td>
<td>All the listings for rent on the application should be displayed. If there are no listings available a message should be displayed explaining this</td>
</tr>
<tr>
<td>View Listings For Sharing</td>
<td>Click the link to view the listings for sharing</td>
<td>All the listings for sharing on the application should be displayed. If there are no listings available a message should be displayed explaining this</td>
</tr>
<tr>
<td>Express interest in a sharing opportunity</td>
<td>Complete the form to express an interest in sharing a property</td>
<td>The application should accept the request and display a confirmation message. The new sharing opportunity should then appear on the page</td>
</tr>
<tr>
<td>Express interest in a sharing without completing information</td>
<td>Don’t complete the form to express an interest in sharing a property</td>
<td>The application should reject the request and display an error message with the fields that are required</td>
</tr>
<tr>
<td>View a listing</td>
<td>Select a listing to view from</td>
<td>The application should</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
<td>Expected Result</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Sales, Rentals and Sharing results</td>
<td>display the listing to the user</td>
<td></td>
</tr>
<tr>
<td>Contact the listing owner</td>
<td>Fill in the contact enquiry form on a listing page</td>
<td>The application should send an email to the listing owner with the details of the enquiry. If the CC box was checked, a copy should also be sent to the user.</td>
</tr>
<tr>
<td>Contact the listing owner with incomplete form</td>
<td>Omit details from the enquiry form on a listing page</td>
<td>The application should not send an enquiry email and should highlight the fields required.</td>
</tr>
<tr>
<td>Search listings</td>
<td>Complete the search box with a selection of criteria</td>
<td>The application should return results matching these criteria.</td>
</tr>
<tr>
<td>Sort search results</td>
<td>Change the sorting options for the search results</td>
<td>The application should reload the list of results in the sorting order defined. The application should retain any previous search criteria inputted as well.</td>
</tr>
<tr>
<td>Adjust the browser size</td>
<td>Change the size of the browser window or view on a mobile device</td>
<td>The application should resize to correctly fit the screen size.</td>
</tr>
<tr>
<td>Print page</td>
<td>Click the print button</td>
<td>The browser should display a print preview of the current page.</td>
</tr>
</tbody>
</table>
J APPLICATION CODE

A full copy of the applications code can be found on the attached CD. Some excerpts of specific sections are found below:

J.1 Google Maps API and Address Lookup

This section of JavaScript code makes use of the Google Maps API. It allows a user to click on the map, get the GPS co-ordinates and lookup the address of those co-ordinates. It will then populate the listing form with this information. The map is also restricted to Ireland only.

```
//dont do anything until the page has been fully loaded
$(document).ready(function() {
  //global variables
  var map = null;
  var marker = null;
  //the popup window that appears on the map
  var infowindow = new google.maps.InfoWindow({
    size: new google.maps.Size(200, 200)
  });
  var geocoder;

  // A function to create the marker and set up the event window function
  function createMarker(latlng, name, html) {
    var contentString = html;
    var marker = new google.maps.Marker({
      position: latlng,
      map: map,
      mapTypeControl: true,
      mapTypeControlOptions: {
        style: google.maps.MapTypeControlStyle.DROPDOWN_MENU,
      },
      navigationControl: true,
      zIndex: Math.round(latlng.lat() * -100000) << 5
    });
    //event listener that listens for the user to click on the map
    google.maps.event.addListener(marker, 'click', function() {
      infowindow.setContent(contentString);
      infowindow.open(map, marker);
    });
    google.maps.event.trigger(marker, 'click');
    return marker;
  }

  //loads the actual map on the page
  function initializeMap() {
    var mapProp = {
      //sets the map to center in Ireland
      center: new google.maps.LatLng(53.4, -7.6),
      zoom: 7,
      //show in earth view by default
      mapTypeId: google.maps.MapTypeId.HYBRID,
    }
    //geocode object for address lookup
    geocoder = new google.maps.Geocoder();

    // GPS Bounds for Ireland Only
    var strictBounds = new google.maps.LatLngBounds(
      new google.maps.LatLng(50.72, -12.28),
    );
  }
});
```
new google.maps.LatLng(55.45, -5.0);
);
var maxZoomLevel = 7;
//Draw the map on the page
map = new google.maps.Map(document.getElementById("listingsMap"),
mapProp);
//when the user clicks on the map, get the GPS co-ordinates and place them into the form
google.maps.event.addListener(map, 'click', function(event) {
  $("input[name='latitude']").val(event.latLng.lat());
  $("input[name='longitude']").val(event.latLng.lng());
  codeLatLng(event.latLng.lat(), event.latLng.lng());
  console.log("Latitude:" + event.latLng.lat() + " Longitude:" + event.latLng.lng());
});
//Close the popup marker when they click on the map
google.maps.event.addListener(map, 'click', function() {
  infowindow.close();
});
//open a new popup marker in the location they clicked
google.maps.event.addListener(map, 'click', function(event) {
  if (marker) {
    marker.setMap(null);
    marker = null;
  }
  marker = createMarker(event.latLng, "name", 
"<b>Property Center Location</b><br/>(<i>Please Ensure Its Correct</i>)");
});

// Listen for the dragend event and check if they are in bounds of the map
//this prevents them from navigating to outside of ireland
google.maps.event.addListener(map, 'dragend', function() {
  if (strictBounds.contains(map.getCenter())) return;
  // We're out of bounds - Move the map back within the bounds
  var c = map.getCenter(),
    x = c.lng(),
    y = c.lat(),
    maxX = strictBounds.getNorthEast().lng(),
    maxY = strictBounds.getNorthEast().lat(),
    minX = strictBounds.getSouthWest().lng(),
    minY = strictBounds.getSouthWest().lat();
  if (x < minX) x = minX;
  if (x > maxX) x = maxX;
  if (y < minY) y = minY;
  if (y > maxY) y = maxY;
  map.setCenter(new google.maps.LatLng(y, x));
});
// Limit the zoom level
google.maps.event.addListener(map, 'zoom_changed', function() {
  if (map.getZoom() < maxZoomLevel) map.setZoom(maxZoomLevel);
});
//end initialize
//call the initialize function to get things going
google.maps.event.addDomListener(window, 'load', initializeMap);
//this function is used to look up the gps co-ordinates and get the address
function codeLatLng(lat, lng) {
  var lat = parseFloat(lat);
```javascript
var lng = parseFloat(lng);
// perform the lookup with the Google Maps API
var latlng = new google.maps.LatLng(lat, lng);
geocoder.geocode(
    {'latLng': latlng
    }, function(results, status) {
        if (status == google.maps.GeocoderStatus.OK) {
            // if we have results, then lets parse the return json array
            if (results[0]) {
                //console.log(results);
                var county;
                var neighbourhood;
                if (results[0]) {
                    for (var i = 0; i < results.length; i++) {
                        // google uses strange naming conventions for Ireland
                        // county
                        if (results[i].types[0] == "administrative_area_level_2") {
                            county = results[i].address_components[0].long_name;
                        }
                        // area
                        if (results[i].types[0] == "neighborhood") {
                            neighbourhood = results[i].address_components[0].long_name;
                        }
                    }
                    // automatically populate the form
                    $('#input[name=townland]').val(neighbourhood);
                    $('#select[name=county]').val(county).attr("selected", true);
                    // For some reason Google saves Donegal differently..
                    if (county == "County Donegal") {
                        county = "Donegal";
                        $('#select[name=county]').val("Donegal").attr("selected", true);
                    }
                    if (county == "") {
                        $('#select[name=county]').prop('selectedIndex', 0);
                    }
                    // get the provience that the county is in
                    provience = getProvience(county);
                    if (provience) {
                        $('#select[name=provience]').val(provience).attr("selected", true);
                    }
                } else {
                    // we couldn't find the location, so the user will have to enter it manually
                    console.log("Unable to find the location");
                }
            }
        }
    });
```
J.2 Process, Resize and Store Images

This section of PHP code is used to process uploaded images, resize them and save them to the application.

```php
<?php
public function uploadImages(){
    $imageID;
    $imageURL;
    //Loop through each uploaded file
    foreach (Input::file('files') as $photo) {
        //the rules for validating the images
        $validator = Validator::make(
            array('file' => $photo),
            array('file' => 'required|mimes:jpeg,png,gif|image|max:10000')
        );
        // this should already have been validated by client side javascript, but this
        // will catch any malicious attempts without js
        if ($validator->passes()) {
            $tempPhoto = $photo;
            $imageName = time().$photo->getClientOriginalName();
            //handle the original image first, for some reason we can't move the file till the
            // end or it breaks the resizing
            $photo = new Photo(array(
                "title"=>$imageName,
                "url="/uploads/".$imageName, "size"=>"original"
            ));
            $photo->save();
            $images['url'][] = "/uploads/".$imageName;
            $images['id'][] = $photo->id;
            //make the sizes and save them
            //decided to use fit as it resizes and crops, needed for consistency
            //create the small image 300px x 250px
            $newImage = Image::make($tempPhoto)->fit(300,250);
            $newImage->save("uploads/small-".$imageName);
            $photo_size = new Photo_Sizes(array("url="/uploads/small-".$imageName,"size"=>"small"));
            $photo->sizes()->save($photo_size);
            //create the medium image 500px x 400px
            $newImage = Image::make($tempPhoto)->fit(500,400);
            $newImage->save("uploads/medium-".$imageName);
            $photo_size = new Photo_Sizes(array("url="/uploads/medium-".$imageName,"size"=>"medium"));
            $photo->sizes()->save($photo_size);
            $imageURL = "/uploads/medium-".$imageName;
            //dont crop the large images - 800px x auto height
            $newImage = Image::make($tempPhoto)->resize(800,null,function($constraint){
                $constraint->aspectRatio();
            });
            $newImage->save("uploads/large-".$imageName);
            $photo_size = new Photo_Sizes(array("url="/uploads/large-".$imageName,"size"=>"large"));
            $photo->sizes()->save($photo_size);
            $original = $tempPhoto;
            $original->move('uploads/',$imageName);
            $imageID = $photo->id;
        } else {
            // Collect error messages
        }
    }
}
```
$error_messages[] = 'File '' . $upload->getClientOriginalName() . ''. $validator->messages()-
>first('file');
return Redirect::route('home')->withMessage("You tried upload either invalid files or files that were too big.");
}

$images = json_encode($images);
//return a json response to the Javascript to display on the front end of the site
return Response::json(array('success'=>200,'imageID'=>$imageID,'imageURL'=>$imageURL));
?>

J.3 Searching Listings For Sale

The PHP code below shows how the query for searching the listings for sale is built depending on the search parameters passed in by the user.

```php
function searchSales(){
    //get all the search parameters from the form
    $county = Input::get('county');
    $provience = Input::get('provience');
    $minSize = Input::get('minSize');
    $maxSize = Input::get('maxSize');
    $townland = Input::get('townland');
    $minPrice = Input::get('minPrice');
    $maxPrice = Input::get('maxPrice');
    $sorting = Input::get('sorting');
    if($sorting == ""){
        $sorting = sortOrder("1");
    }else{
        $sorting = sortOrder($sorting); //calls helper method to determine parameters to be sorted by
    }
    //start building the query depending on the parameters
    $query = Listing::where('listingType','=','sale');
    $query->where('status','=','active');
    if($county != ""){
        $query->where('county','=',$county);
    }
    if($provience != ""){
        $query->where('provience','=',$provience);
    }
    if($maxSize != ""){
        $query->where('size','<=',
            floatval($maxSize));
    }
    if($minSize != ""){
        $query->where('size','>=',
            floatval($minSize));
    }
    if($townland != ""){
        
```
$query->where('townland', '=', $townland);

if($minPrice != "")
{
    $query->where('price', '>', floatval($minPrice));
}
if($maxPrice != "")
{
    $query->where('price', '<=', floatval($maxPrice));
}
//order the results by the current sorting order
$query->orderBy($sorting['sortby'], $sorting['order']);
//only show the first 10 for pagination
$results = $query->paginate(10);
if (Request::ajax())
{
    //return the result as a JSON string if it's an ajax request for pagination
    return Response::json(View::make('listings.ajax.ListingGrid')-
    >withListings($results)->render());
}
//return the search results page with the results
return View::make('listings.forSale')-
    >withListings($results);

J.4 Registering A New User
The PHP code below is used when a new user registers with the application.

```php
public function saveUser()
{
    $rules = array(
        'email' => 'required|email|unique:users',
        'password' => 'required|min:5|same:password_confirmation');
    //check that the information they provide passed validation rules
    $validator = Validator::make(Input::all(), $rules);
    if($validator->fails())
    {  //if it fails redirect back with the error messages
        return Redirect::to('register')->withErrors($validator);
    }else{
        //passed validation so lets save them
        $user = new User();
        $user->email = Input::get('email');
        //encrypt their password
        $user->password = Hash::make(Input::get('password'));
        $user->group = Config::get('constants.groups.User');
        //save them
        $user->save();
        $data = ['user' => $user];
        //send the welcome email with the Mandrill API
        Mail::send(['emails.welcome', $data, function($message) {
            $message->from('info@agriland.ie', 'Agriland Property');
            $message->to(Input::get('email'));
            $message->subject('Welcome To Agriland Property');
```
J.5 CSS Code Used To Style The Application

The following code is an example of responsive CSS used to style the application. Media Queries can also been seen in the code which are used to make the site responsive for different screen sizes.

```css
.loadMore.button {
  width: 100%;
  background: #00664B;
  color: #fff;
  text-align: center;
  padding: 0;
}
.loadMore.button li a {
  color: #fff;
  font-size: 1rem;
  display: block;
  width: 100%;
  padding: 1.25rem;
}
.loadMore.button li.disabled {
  display: none;
}
.loadMore.button:hover {
  background: #7AC043;
}
.a.dashboard-tab {
  display: inline-block;
  width: 100%;
  background: #747474;
  color: #fff;
  padding: 0.3125rem;
  font-size: 1.125rem;
  text-align: center;
}
a.dashboard-tab:hover {
  background: #7AC043;
}
@media only screen and (min-width: 40.063em) {
  .half-width-field {
    width: 47%;
    margin-right: 3%;
    float: left;
  }
}
@media only screen and (max-width: 40em) {
  .uploaded-photo {
    width: 100%
  }
}
```
REFERENCES


GLOSSARY OF TERMS USED

API – Application Program Interface

CSS – Cascading Style Sheets
A style sheet language used for describing the look and feel of websites

CSS3 – Cascading Style Sheets 3
The latest version of CSS which supports animations and media queries

Git
A version control system that maintains different versions of code

GitHub
A web based service which stores Git repositories

HTML – HyperText Markup Language
The language used to create websites

HTML5 – HyperText Markup Language 5
The latest version of HTML which supports additional features such as semantic mark-up.

JavaScript
A programming language used on the web. It is used to create interactive elements on a website

Linux
An open source computer operating system

MySQL
A popular open source database management system

MVC – Model View Controller
A way of structuring code in a software project

PHP – Hypertext Pre-processor
A server side scripting language used for programming websites

Sublime Text Editor 2
A text editor designed for code

Semantic
A meaningful way of writing code that is easy to understand

URL – Uniform Resource Locator
Specifies the unique location a webpage can be found at

Windows
A popular operating system developed by Microsoft

XAMPP
Free open source software that allows you to install a server on a computer