University of Sussex

WebWizard

Block Based Website Creation Tool for Small Businesses

Alex Davis
118424

Supervisor: Dr. Martin White
Department of Informatics
April 2017

Declaration

This report is submitted as part requirement for the degree of Computing for Business Management at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Signature:

Alex Davis

UNIVERSITY OF SUSSEX

Alex Davis, Computing for Business Management

Summary

Websites have become an essential tool for almost every business, from huge international corporations to freelance photographers. However, with increased user expectations of website features and design, creating a website is an expensive proposition. My proposed project is an application that aims to provide the user with an extremely simple and quick way of creating a professional website for their business that is able to scale with them as they grow. There are currently many existing tools that allow users to create a website, however the ones that offer more advanced features, such as e-commerce support, come with significant drawbacks. There are expensive subscription costs, restrictions on the user's hosting options as well as convoluted and often proprietary code which prevents upgrading the site manually should the company want to move away from the tool which created it. My solution is to use pre generated 'blocks' of code which are organised by the user. The user can then export a folder of code which can be hosted anywhere and edited by a professional should they scale enough to need more advanced features. During this project I will be carrying out various forms of research and surveys in order to produce a genuinely useful tool which can be used to create a modern, fully functional and mobile-friendly website. The impact of such a tool could change the way people view websites from an expensive luxury which only successful, established businesses or tech savvy entrepreneurs can afford to a simple wizard which can produce something from only the most elementary information.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This report aims to analyse the competitive market of website building software, research issues with the current software, and finally design and implement a system which solves these problems. Due to various constraints, it is not the aim of this project to produce an application which can directly rival the current market, the aim is simply to create a usable piece of software which solves issues held by the existing system.

In order to achieve these aims, this report will first explore the existing market and define basic objectives for the product. Following this, the report will look into the professional and ethical considerations of the project before defining a set of functional and non-functional requirements based on further research into the problem. Once the requirements are finalised, the product will be designed, implemented and tested. Finally the evaluation section will study how many of the initial requirements have been met, discussing the main challenges faced during the process and work that could be undertaken to further improve the project in the future.

1.1 Background Research

As of April 2017 there are over 1.1 billion active websites in existence [1]. Coupled with the relatively recent advancements in web technologies allowing effortless information, money and content transfers across the world. In the past, marketing, sales and advertising each required relatively large investments such as physical stores, billboard advertisements and sales staff. However, the internet has rapidly become an ideal mechanism in addressing all of these points with a much lower upfront cost. Even the smallest business, such as a school pupil looking to earn some pocket money proof-reading essays, could dramatically increase sales and customer base by removing the limitations of physical location with a simple website.

However, websites are not as widespread as one would image, figure 1.1 shows that nearly half of small businesses do not have a website despite the advantages listed above.

These results clearly show that the process of building a website is considered very difficult by certain types of business owners. Furthermore, an investigation into small businesses in the UK concluded that 84% of business owners said that their website must be mobile friendly and 40% stated that current website builders were too complicated to use. The top reasons given by those businesses without a website were:

1. Not relevant to business/industry

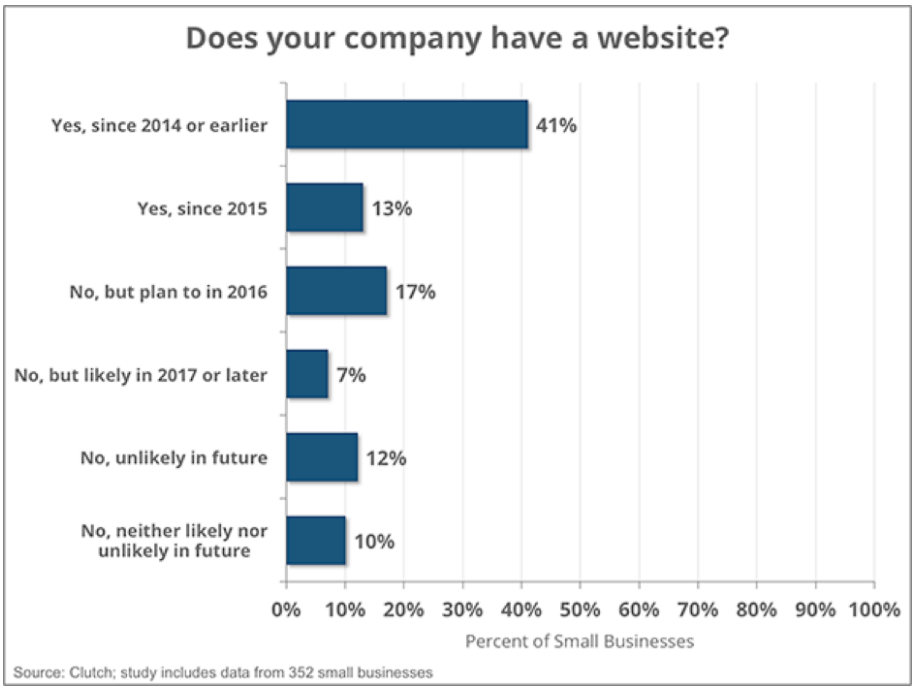


Figure 1.1: Research regarding website ownership of 352 small businesses conducted in December 2015

2. Cost
3. Lack of technical knowledge
4. Need for ongoing maintenance

Moreover, the results concluded that 30% were looking for an Internet company that could provide easy to use tools to help set up a website. This shows that despite the currently available tools, there is still a big market for something easier to use. If you would like a website but have neither the technical experience required to write your own, nor the money to pay for a professional designer, by far the most popular option is to use a WYSIWYG (what you see is what you get) online page creator. These aim to make the process of creating a website a visual based process using drag and drop functionality, and either automatically generate code or offer pre-coded templates which can be customised.

Although large amounts of money have been invested into these applications, they are still missing features that can lead some businesses to feel forced into financing a custom built site, or even giving up on building a site altogether. In March 2017, a mobile app for building mobile only websites received over $3 million in funding, a strong indication that, despite the already crowded market, there is still plenty of room for mobile focused tools [2]. Figure 1.2 shows the amount of funding raised by investors for existing website building applications.

1.2 Problem Statement

From analysing this research it becomes clear that current methods of creating websites each have their drawbacks. It is seen as an expensive process requiring both technical knowledge and design skills that can be a huge obstacle for the average small business

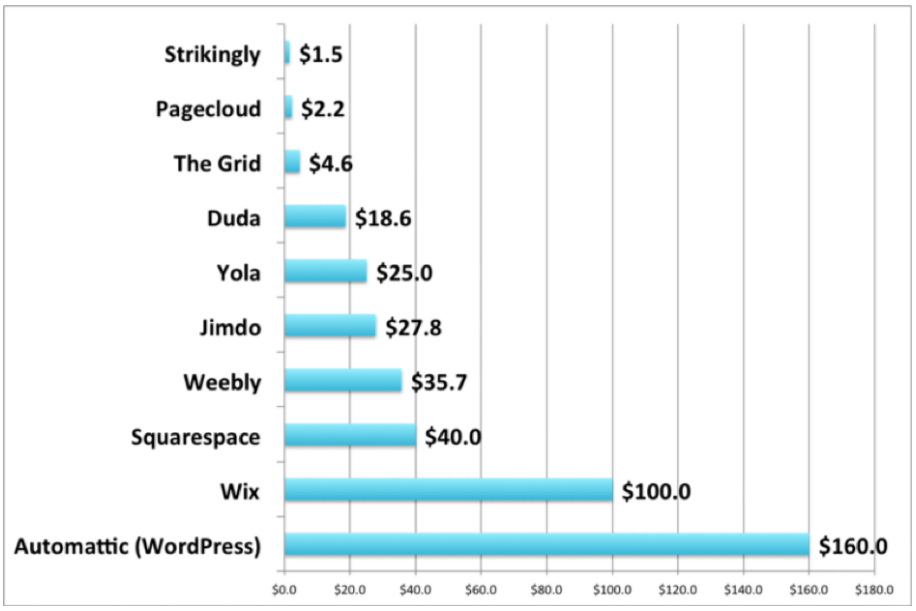


Figure 1.2: Venture capital funding raised (millions)

owner or start up [3]. Furthermore, research shows that the cost of not having a website can often be greater than the cost of producing one [4]. However, many small businesses simply do not have the means to create one, even if it was 100% guaranteed to lead to a boost in profits. Even the businesses that currently have websites created using one of the many online tools have frequent complaints regarding the ongoing costs and the lack of flexibility regarding hosting and editing the site [5].

1.3 Objectives of this project

The main aim of this project is to create a simple to use website builder that can easily produce a fully responsive website suitable for a small business.

To elaborate, my vision is a method of creating a fairly complex website designed so that the initial creation, as well as any future upgrades to the site, are as simple to carry out as possible. A drawback to using this approach is a lack of control over the detailed design of the site. However, the average business owner wanting a quick and easy website will be much more concerned with the applications ease of use rather than a lack of control. Furthermore, the application should produce correct and well commented code which will allow detailed design changes to be made. This will be a useful feature if the user happens to understand code or hires a developer.

1.3.1 Primary Objectives

My proposed system will not be able to achieve everything currently possible with the existing systems due to strict time and resource considerations. However, there are a set of essential requirements which are both achievable and vital to the projects success.

1. The application should be easy to understand and use
2. The application should run on all operating systems and browsers

3. The application should output a correctly formed folder structure with properly named files
4. The websites code should conform to WW3[1] standards and be well commented to allow future developers to extend the site should the company grow to a point where they can afford to upgrade the site
5. The application should allow the user to add text and multimedia content from their computer
6. The application should allow the user to add or remove various site functionality including:
    - Image slideshows/light-boxes
    - Contact forms
    - Embedded content from other sites such as YouTube
7. The application should provide simple instructions and should not require technical knowledge of code to use
8. The generated website should be functional on all screen sizes including mobile devices
9. The user can upload the generated website to their web hosting provider from within the app

1.4 Secondary Objectives

In addition to the required primary objectives, I have created a list of desirable and potentially achievable secondary objectives:

1. The ability to list products available for sale
2. The ability for a customer to pay for a product through the website securely
3. A simple way of managing the inventory of products for sale (e.g. adding, removing, modifying etc.)
4. The colour scheme of the website can be automatically generated based on analysis of the company's logo or other images
5. The user can copy and paste rich text from the web or a word processor into the application and all styles such as font, colour, bold, italics etc. will be maintained
6. The application allows the user to easily implement a shopping cart system and secure online checkout
7. The layout and design of the page can be automatically generated by analysing the amount and types of content.
8. The application includes a way to view and edit the HTML code of the site

[1] https://www.w3.org/standards/

Chapter 2

Professional and Ethical Considerations

The British Computer Society (BCS) define a set of rules and standards for those working in IT [6]. It is important that this project meets all these requirements, therefore I have outlined each of them in respect to the project below:

2.1 Public Interest

Section 1a. states: *'You shall have due regard for public health, privacy, security and well-being of others and the environment.'*

Both the application and its outputted websites will be secure and send personal data only over encrypted channels. There should be no concern regarding public health or the environment as I will not be creating any physical products and neither the program nor the website will contain flashing images with the potential to cause epileptic seizures.

Section 1b. states: *'You shall have due regard for the legitimate rights of Third Parties.'*

As I will be using several third party frameworks and APIs it is important that they are referenced correctly and used in accordance to the copyright rules laid out for each individual piece of software I will use.

Section 1c. states: *'conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement'*

As my project does not contain any content that I have written other than the user interface, I do not believe that it breaks any of the rules stated in section 1c.

2.2 Professional Competence and Integrity

Section 2b states *'You must not claim any level of competence that you do not possess.'*

Although the program will use multiple third party frameworks, I will properly reference all of these in the application and will not claim to be responsible for anything carried out by a piece of third party software.

Section 2d states *'ensure that you have the knowledge and understanding of Legislation* and that you comply with such Legislation, in carrying out your professional responsibilities.'*

I am fully aware of all applicable laws, statutes and regulations that may affect the integrity of my project.

Section 2e states *'respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.'*

One of my requirements is to allow user submitted feedback regarding the application, I will also be sending surveys to my prospective user base regarding feedback on the design of the program.

Section 2g states *'you shall reject and will not make any offer of bribery or unethical inducement.'*

I would not make any offer of bribery or unethical inducement as any reward I would get from such an action would not make the consequences worthwhile.

Chapter 3

Requirements Analysis

The requirements analysis stage consists of the following steps:

1. Requirements Gathering - Undertaking both primary and secondary research to fully understand the target audience and their requirements in the current market
2. Requirements Analysis - Creating use cases and defining target users
3. Requirements Specification - Using the previously gathered information to finalise the required and optional requirements defined in chapter 1

3.1 Requirements Gathering

The key to successfully drawing up a list of requirements is good research. Both primary and secondary research methods were used to define the user base and gather the primary and secondary requirements.

3.1.1 Existing Applications

As shown in section 1.1, the online website builder market is extremely competitive with the top players receiving millions of dollars in funding. As a result, there is a lot that can be learned by analysing both the positive and negative features of these sites.

3.1.2 Existing Site Builders

A full feature list of the current top 5 most popular (as rated by independent review site 'SiteBuilderReport'[1]) is shown in table 3.1 and shows that none of them allow a user to move their website to a different host. In the case of a business wishing to add features that are not supported by the WYSIWYG[2] platform they are using, all of the work put into the existing site would have to be scrapped. Consequently, a new one would need to be built from scratch using a different platform. This puts businesses in the tricky position of either needing to spend large amounts of money upfront on a web designer or using the site builder method and hoping that no reason to move arises.

[1] www.sitebuilderreport.com
[2] WYSIWYG = What you see is what you get

Table 3.1: Table of popular site builders and their features

	Weebly	SquareSpace	Wix	Strikingly	Voog
Blog	✓	✓	✓	✓	✓
E-Commerce	✓	✓	✓	✓	
Form Builder	✓	✓			✓
Mobile Ready	✓				
Multilingual	✓				
Full control over Design			✓		
Platform Independent					

Furthermore, all of these solutions rely on proprietary applications or demand a regular subscription. The first major drawback of this is if the service you are using shuts down or becomes too expensive, you must start from the beginning with a new provider. The second major drawback is the complexity of the code that is generated. For example, sites which are built automatically from generated code are often inefficient and can contain errors which make the site unresponsive to use and difficult to edit in the future. Finally, as of the time of writing, the major website building sites do not offer fully responsive websites. In other words, a website created through one of these sites will not adapt automatically to different screen sizes and a separate site will need to be made for phones, tablets, desktops etc.

Studying these applications will allow me to combine and modify the best features and design choices from each, while adding additional features missing from all of them. To see exactly how each of the current products handled various actions, I made an account with Wix [7], one of the biggest and most popular site builders.

3.1.3 Wix User Experience

Wix has a very simple design and it was easy to login using a Google linked account. However, the homepage is quite complex and it is not immediately obvious how to get started with building a site (see figure 3.1). After clicking the create site button, I was presented with a list of site types. I found this a very easy process to understand and use. Clicking a site type presented led to a list of templates with thumbnail previews. There were 34 pages of these and many looked very similar so I felt this could have been simplified. Upon entering the main builder interface (see figure 3.2), I felt somewhat overwhelmed with the amount of options. The snapping feature was frustrating to use and it took a while to move an element where I wanted it. Nevertheless, it did support a very wide range of features, ranging from e-commerce to blogging. Mobile sites are supported, however they have to be created separately to the desktop sites. This was frustrating as many changes had to be repeated to make the mobile site function properly.

Overall, it is clear that a website builder which sacrificed some design flexibility in order to provide a less frustrating user experience could have a place in the market. Furthermore, a website builder that is capable of producing clean, reusable, editable code will have a big advantage over the currently available site builders for businesses that do not wish to be tied to a specific platform without loosing the simplicity of a WYSIWYG design. As a result, this will allow me to achieve the aim of my project, which is to create a simple to use website builder that can easily produce a fully responsive website suitable for a small business.

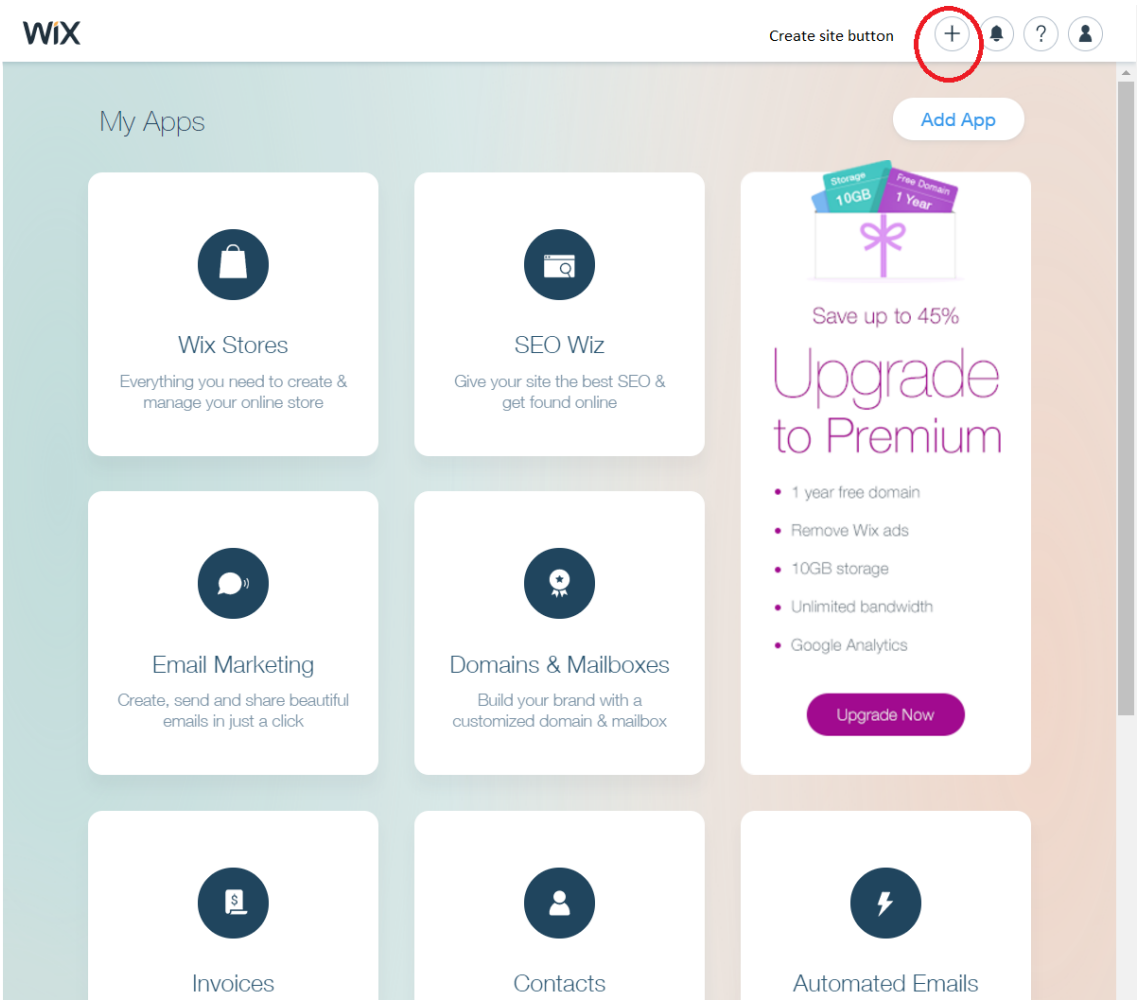


Figure 3.1: Wix home page with create site button annotated

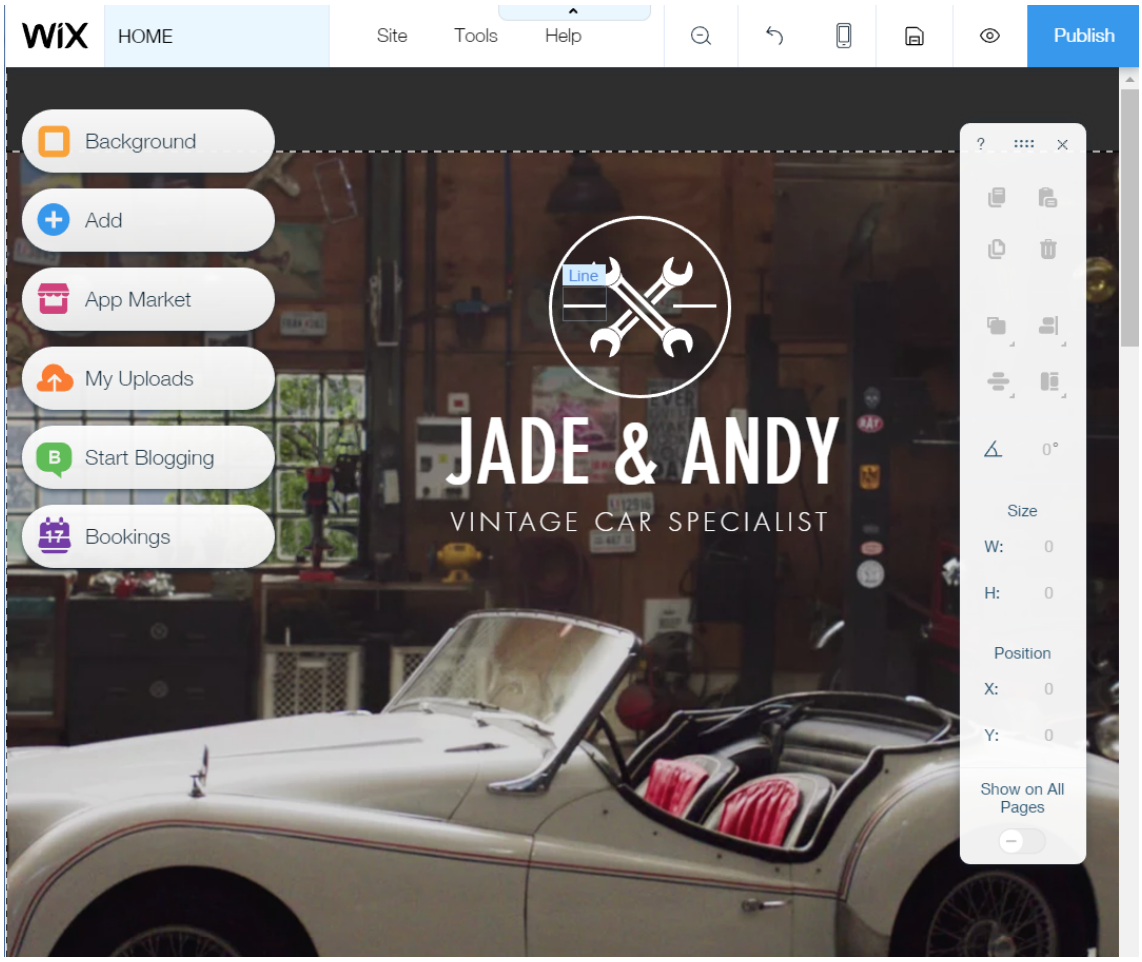


Figure 3.2: Wix builder page

To better understand the positives and negatives of the current site builders, I created a survey and sent it to various groups of people. The results are summarised in figure 3.3 and 3.4, full survey results are in Appendix 8.4:

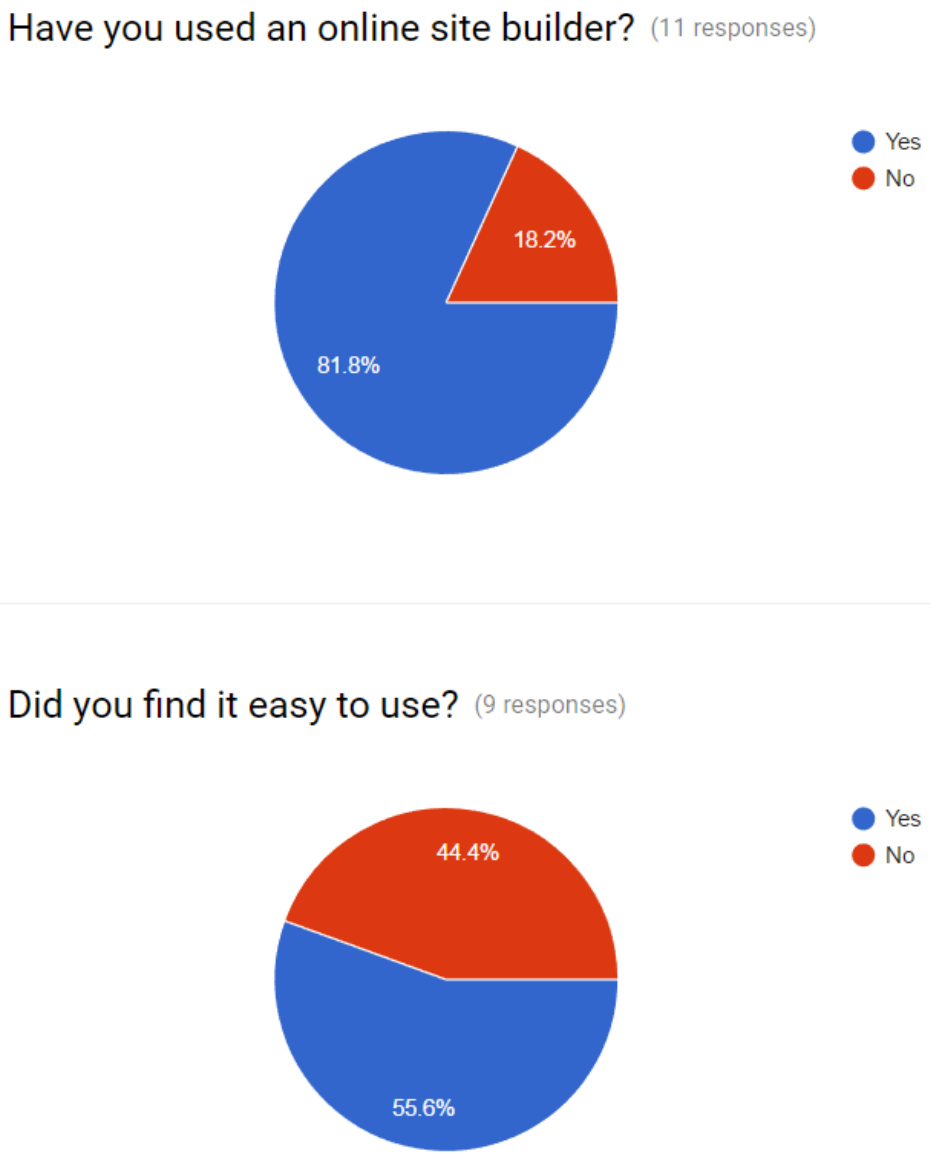


Figure 3.3: Survey regarding existing website builders

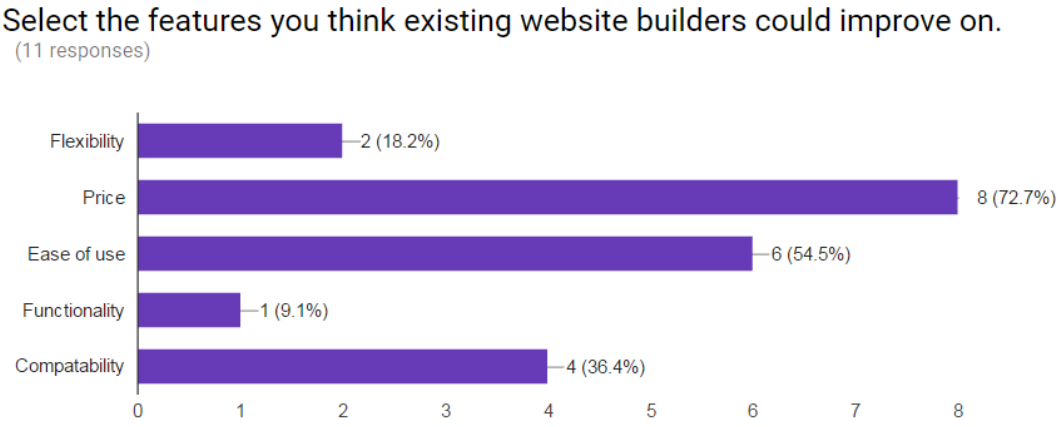


Figure 3.4: Survey regarding existing website builders

In addition to the primary research I conducted, I also gathered data from various relevant reports and studies. Despite not being a feature in many of the most popular site building platforms, figure 3.5 shows how important mobile compatibility is [8]. Figure 3.7 suggests current methods take too long and are too complex as the number of sole traders without a website is much greater than it is for businesses with 2-5 employees. Figure 3.6 reinforces this theory by showing a 30% of people planning to build a website in the next two years who are looking for an easy-to-use tool. Furthermore, studies show that people without a technical background are highly likely to be put off from making a website using an online tool [9].

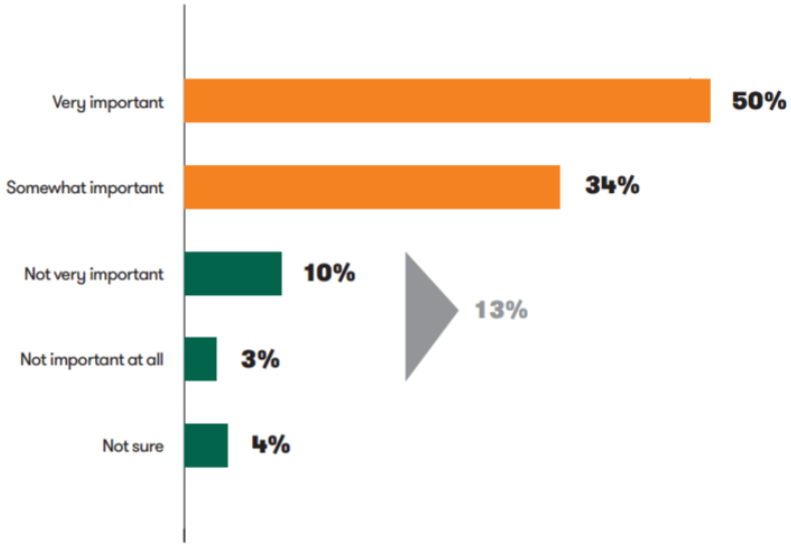


Figure 3.5: Importance of Mobile-Friendly Websites [8]

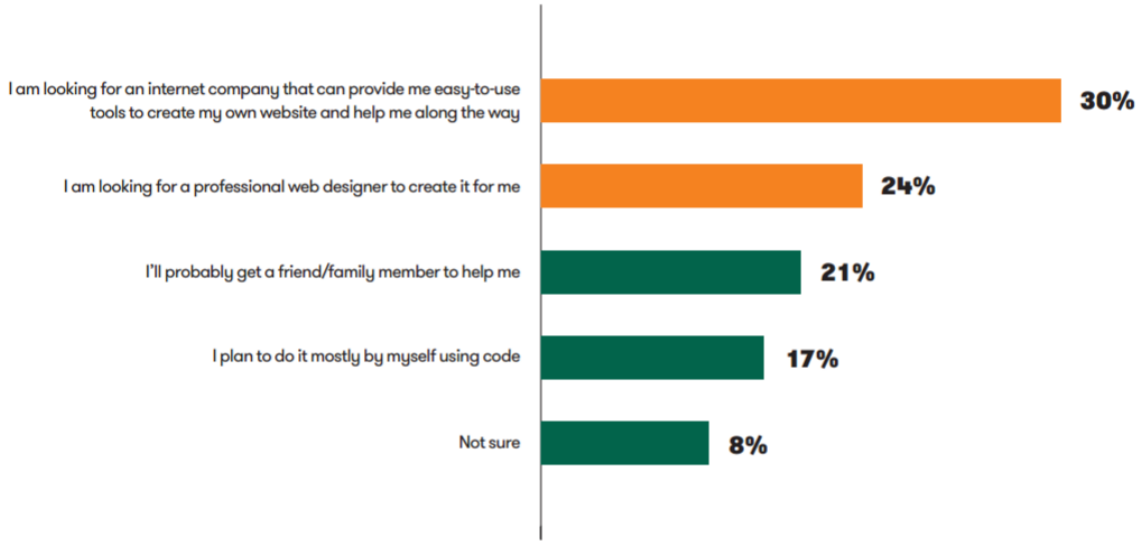


Figure 3.6: Plans To Build A Website [8]

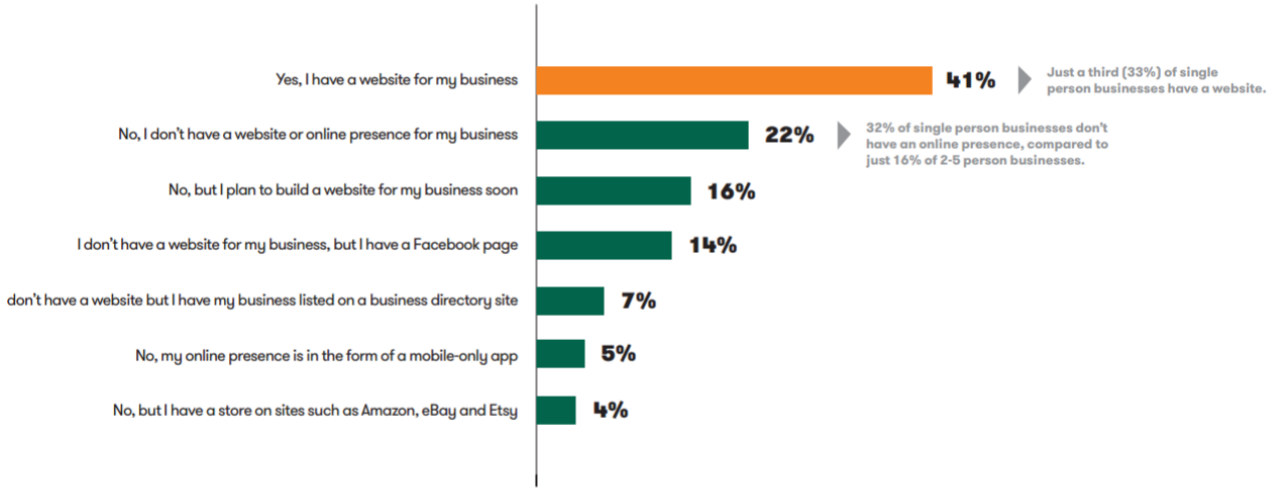


Figure 3.7: Online Presence [8]

3.2 Use Case Diagram

From this research I was able to create a use case diagram and table to define each of the stakeholders in my proposed system (see figure 3.8 and table 3.2).

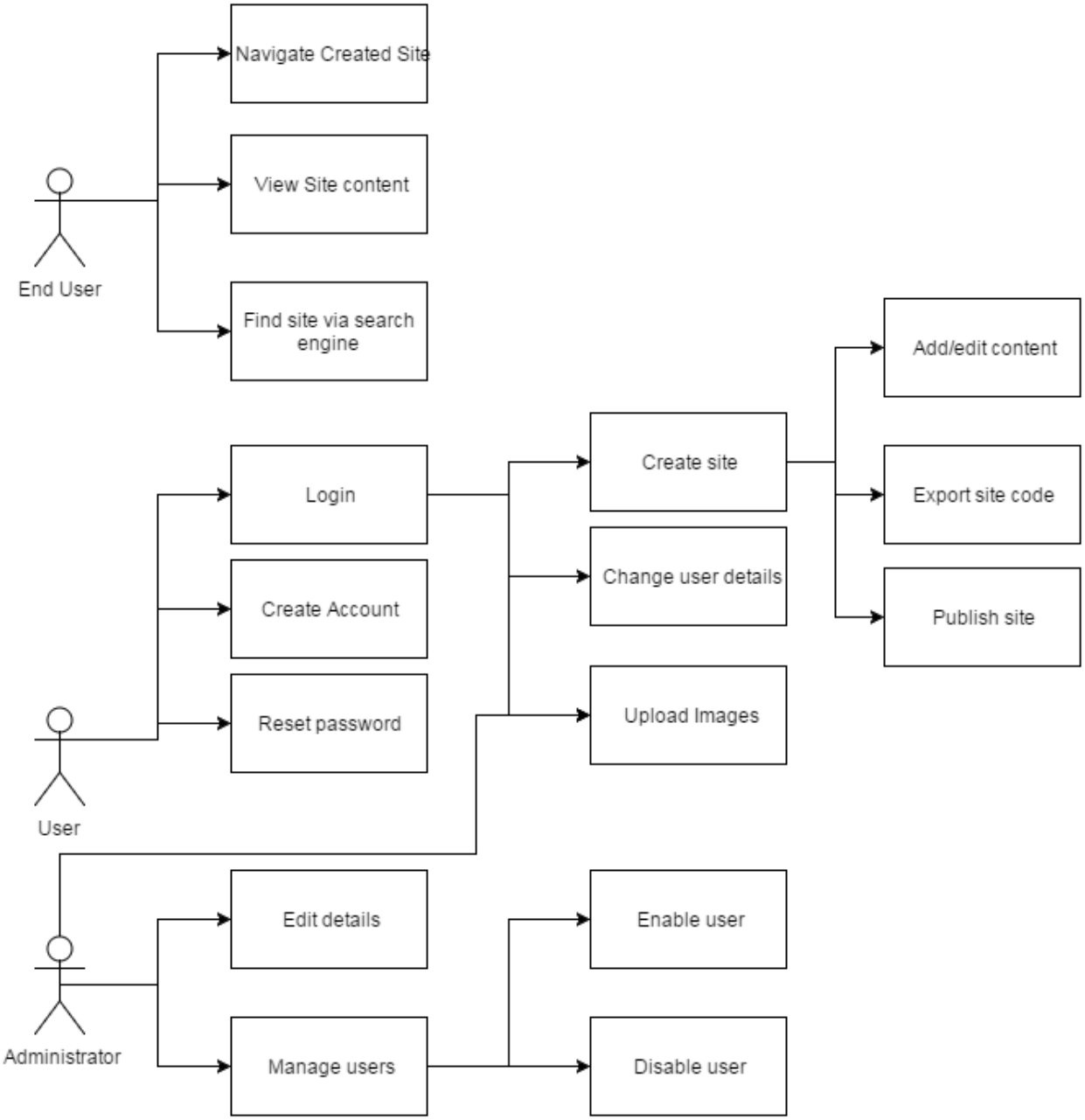


Figure 3.8: Use Case Diagram

Table 3.2: Use Case Analysis Table

Actor	**Case**	**Description**	**E/D**
End User	Navigate Site	The user will want to get around the site	E
End User	View Content	The user must be able to view all content on the page regardless of the device or browser they are using	E
End User	Find site via search engine	The user may want to add meta-data tags to make the site show in search engine listings	D
End User	e-commerce functionality	The user will want to purchase multiple products at once and have a way of organising there desired purchases	D
User	Login	The user will need to be able to login to the application	E
User	Create an Account	The user will need to create an account if they do not already have one	E
User	Reset Password	If the user forgets their password, they may want to reset it	D
User	Change user details	The user may want to change there login details	D
User + Administrator	Upload Images	The application should support image uploads	E
User	Export Site code	The user will want to export the code of their site	D
User	Publish site	The user will want to host the created site	E
Administrator	Edit details	The administrator needs to be able to edit details of the application	E

Administrator	Manage Users	The administrator may want to enable/disable users	D

Key: End User = Anyone using a site generated by the application
E = Essential D = Desirable

3.3 Requirements Analysis

This section will attempt to define a target user by creating a use case diagram and user testing the existing solutions.

3.3.1 User testing

In order to compete with the currently existing website development products, my application must contain some of the features people love about those websites while adding or changing features that are currently missing or irritating to the customer. To get a more accurate idea of what these features are exactly, I created some tests and gave them to people who fitted my target market.

Test Procedure

I chose eight people who had varying levels of technological knowledge and asked them to carry out a set of simple instructions:

1. Create a simple one-page website with a title, a picture and a sentence or two of text
2. Publish the page and view it on a mobile device
3. Add a product that a customer can buy from the webpage

These tests briefly explore the average experience of a business owner attempting to set up a webpage with no prior knowledge.

Results

Only five of the eight people managed to complete the second step and no one could complete the third step without paying a third party. As expected, the people without as much technical knowledge performed worse at this task. The main complaints were the over complication of the sign up process, every one of the test subjects stated 'it took twice as long to begin designing the site as it did finishing it'. Also, they found the interface too complicated and were shocked at the prices for extra support such as e-commerce. The full results of the research are in appendix 8.4.1.

Conclusion

From the test results it can be concluded that the main tagert user for this application is a small business owner who does not have a technical background.

3.4 Requirements Specification

From analysing my research, I can draw up a final requirements specification (see table 3.3) which will describe every feature, both essential and desirable, that should be incorporated into this project.

3.4.1 Functional Requirements

Table 3.3: Functional Requirements

Function	Description	Testing method	E/D	UC	P/S
Code created must be well formed	Code is pre-written and split into modules	Validation using W3schools online validation tools.	E	Admin	P4
User interface is simple to use	Written using as few components as possible	Usability survey	E	Owner Admin	P1
Displays audio and video	Implemented using HTML5 to ensure user friendly experience	Can video and audio be played on the generated web site	E	User	P5
Contains e-commerce functionality	Ecwid API allows ecommerce functionality to be added for free with the option of upgrading to paid plans as the business scales	Client should be able to add product catalog and receive payments through a specified method.	D	Owner Admin	S1,2,3,6
Allows copy and paste of rich text e.g bold/italic text	Several JavaScript libraries support this	Visual test, text copied from a word document or another website should not lose its formatting	D	Owner Admin	S2
The site should have navigation functions	Each created site with more than one page will include a navigation bar and site map with editable links	Usability survey	E	User	P7

The system should be able to create user accounts	Possible with a server side database	Integration test	D	User	P7
Instructions	Instructions should contain information on both using the program, and hosting and editing the site afterwards	Usability survey	E	Owner Admin	P6
Site is mobile friendly	Written in a fully responsive framework such as bootstrap. This should happen automatically without any user input	Usability survey	E	User	P7
User should be able to add their own images to the site	Site should have image upload functionality	Integration test	E	User	P7
Sites design can be automatically generated	The application can generate a starting template based on information given by the user	Usability survey	D	Owner	S4

Key: E = Essential, D = Desirable, UC = Use Case, P = Primary, S = Secondary

3.4.2 Non-functional Requirements

As with the functional requirements, I have created a table (see table 3.4) to show the non-functional requirements.

Table 3.4: Non-functional Requirements

Function	Description	E/D	UC	P/S
Program must not store personal data	Data is only saved if the user requests to output the site	E	Admin Owner	P3
Users data must be secure	Moltin API handles security	E	User	P8
The style and colour scheme of the site can be determined by analysing a logo or photo chosen by the user	Should be possible using a colour analysis JS library	D	Owner	S1
The site should appear in search engines	Needs to be crawlable and include the correct meta data etc. This should all happen automatically without any user input	D	User	P5
User interface is platform agnostic	Written using Java	E	Owner	P1

Key: E = Essential, D = Desirable, UC = Use Case, P = Primary, S = Secondary

Chapter 4

Design

This section will explore the overall design process of the project, justifying the choices that were made and laying out a solid structure which can be relied on when implementing the application.

4.1 System Architecture

To define the structure and behaviour of the system, it is important to spend time designing the architecture of the system before beginning work implementing it.

4.1.1 Platform

The first decision to be made regarding the design is which platform the application will run on. The two main choices are to make either a web application or a stand alone application. By far the most popular platform that currently existing solutions use is the web. Although desktop and mobile applications do exist, I believe they are too inconvenient for my target user, who is someone looking for the simplest possible way to build a website. Downloading and installing a program is an unnecessary step when I can fulfil all of my requirements with a web based solution.

From looking at the requirements of my project, I will need somewhere to store data concerning the users and their sites, a visual user interface for the application, and some way of controlling the interactions between the user interface and the database. Research has shown that the Model View Controller (MVC) design pattern (see figure 4.1) performs better than any other design pattern in web applications involving databases [10]. Therefore, it can be concluded that the MVC pattern is best suited for the architecture of this application.

In order to define the structure and flow of the application, an abstract system diagram is demonstrated in figure 4.2.

On visiting the site for the first time, the user will be prompted with a simple log in page. If they already have an account, they can sign in from this page, otherwise a button labelled 'create account' will take them to an account creation page. An account will consist of a user name, email address (for password recovery purposes) and password.

Once an account is set up, the user can then log in from the home page. Upon logging in, they will be presented with the 'sites' page. This page will show a list of each website the

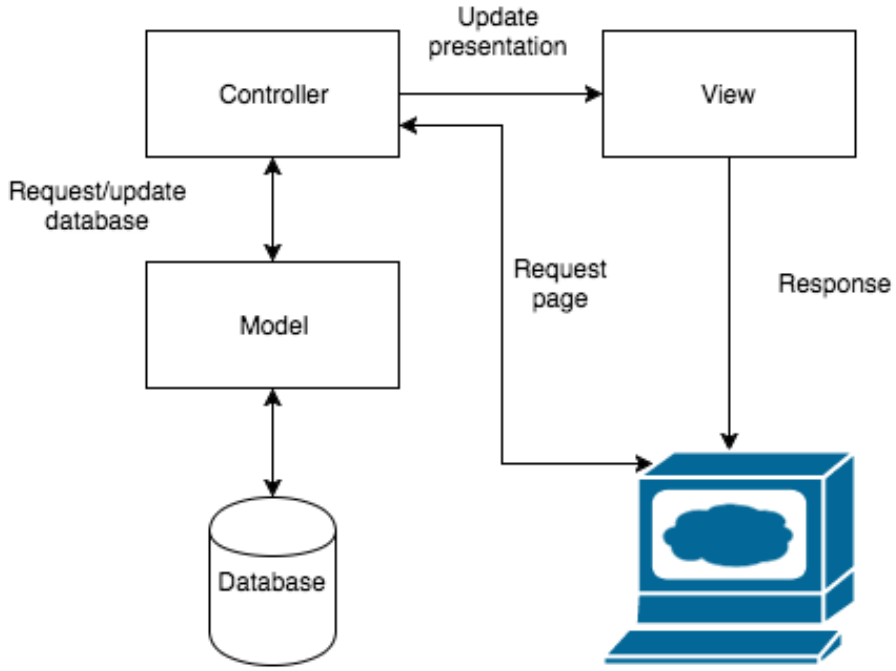


Figure 4.1: MVC Diagram

user has created (if any), and a 'create site' button. If the user has not created any sites yet, it must be a new user. Therefore some introductory text will appear in place of the list of sites.

When the user clicks the 'create site' button, they will be presented with a drop down list of options, each one representing a different type of website (blog, photography etc). Choosing one of these options will open the builder interface and load in the 'blocks' most appropriate to the website type selected.

From the builder page, the user has the ability to drag and drop new blocks, delete or edit existing blocks, save the current version of the site to the database, preview the site in a new tab or upload the entire site to an FTP[1] server.

Aside from the main sites page and the builder page, there will be a page showing user details (if logged in as an admin) allowing the admin to manage accounts. If the user is not an admin, they will only be able to modify their own account (email address, password etc.). Finally, there will be a page allowing the user to upload images to be used in the users website. Images are a vital part of any website, and many websites use them multiple times throughout the site (such as a logo), so it is important that there is an easy way to manage them.

[1]FTP - File transfer protocol

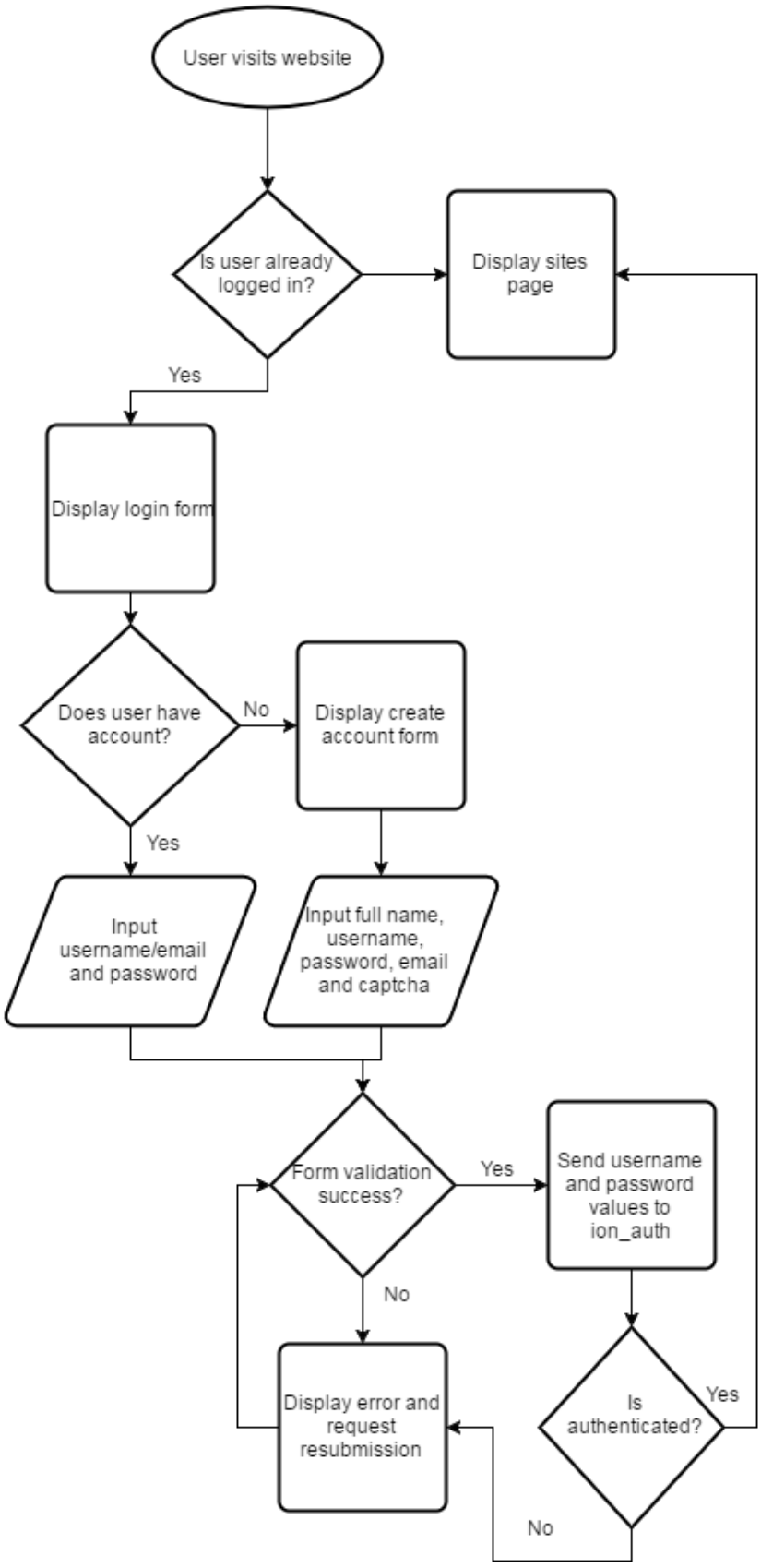


Figure 4.2: System flow diagram

4.2 Draft Sketches

Figure 4.4 shows a rough sketch of the 'sites' page that a user will see immediately after logging in. Figure 4.5 shows the builder page, figure 4.6 shows the images page and figure 4.3 shows the user accounts page.

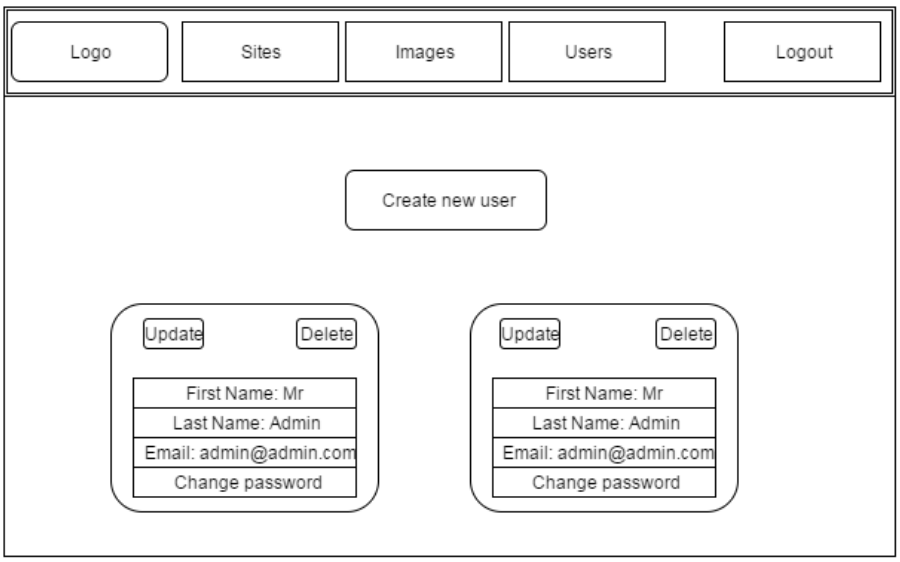


Figure 4.3: Users page sketch

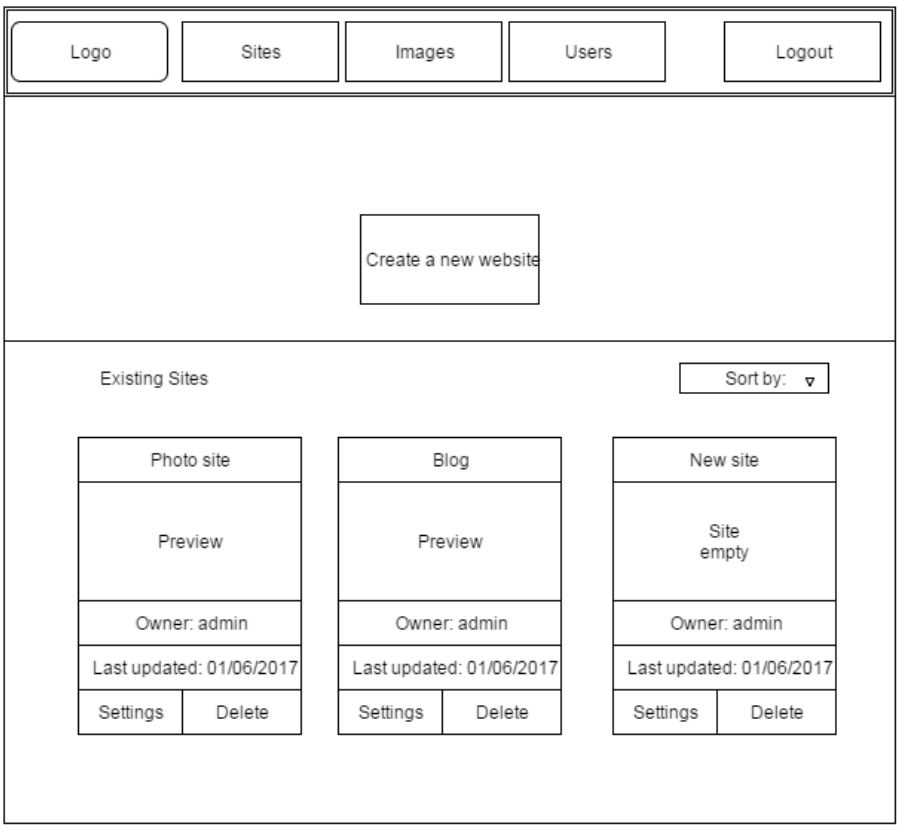


Figure 4.4: Sites page sketch

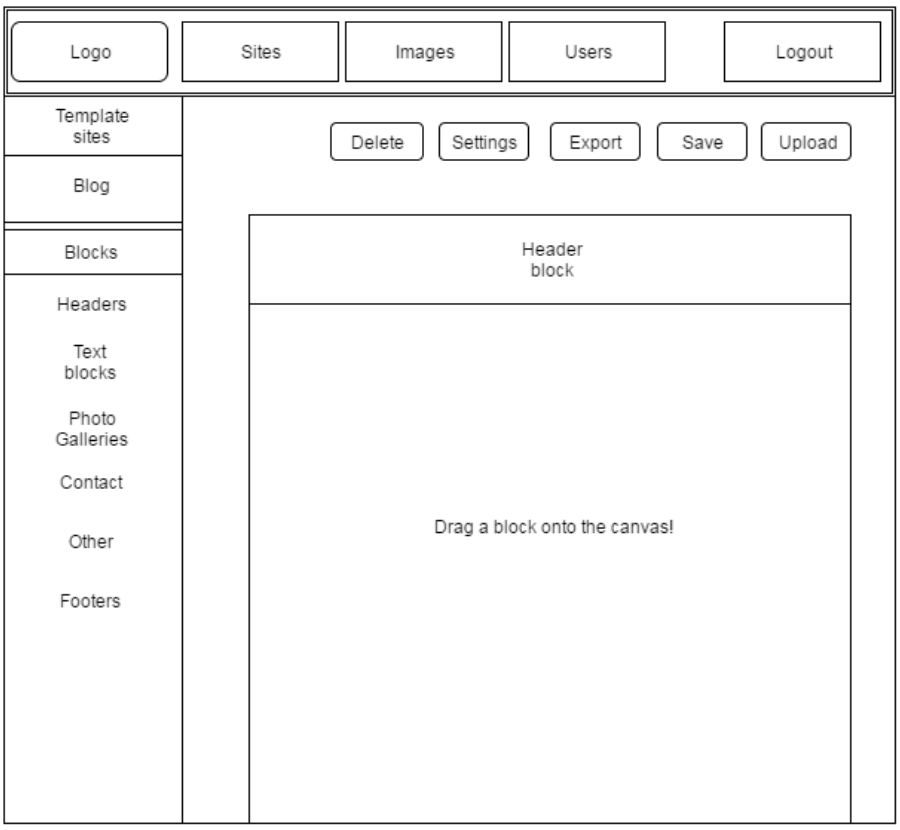


Figure 4.5: Builder page sketch

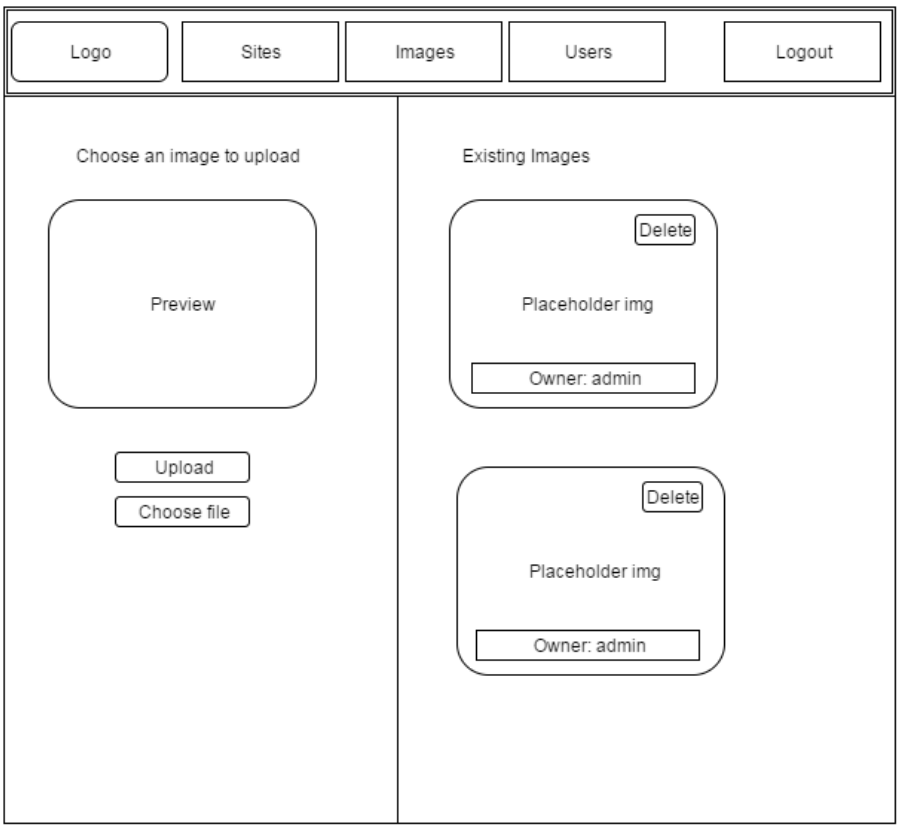


Figure 4.6: Images page sketch

4.3 Constraints

As this project has a set deadline of April 2017, it is vital that appropriate time management techniques and careful design choices are made to avoid falling short on the necessary requirements.

4.3.1 Skill Set

In order to meet the deadline while achieving the project requirements, I will be using technologies that I am comfortable with to avoid spending valuable time learning new languages or frameworks. Codeigniter is a popular MVC framework that I have several years of experience with and I believe it is well suited to handle the back-end of the web application.

During my research of existing technologies, I came across a piece of software called 'Bskit builder'[11]. The site sold bootstrap templates and had created a site builder called 'bskit builder' allowing clients to build a website using pre-written blocks of HTML. I was concerned that I would not finish the project on time if I attempted to write a similar application myself, and this application largely did what I had envisioned the 'builder' page of my application doing. Therefore, I decided to use the source code from it as a starting point for my implementation and modify it as necessary.

4.3.2 Technology Stack

In recent years, modern web applications have trended towards using server side JavaScript frameworks such as NodeJS in their development[12, 13]. However due to my inexperience with these technologies, I chose to use the Codeigniter PHP framework in its place. Although it is slightly outdated, it is still very capable of producing an application that covers all of the set requirements, and due to my experience with creating Codeigniter applications, the project should be able to meet its requirements to a greater extent. For the front end I will be adopting the technology stack used in the bskit application; Bootstrap[2], JavaScript and JQuery. I have experience with JavaScript but the other technologies are fairly new to me, so some prior research was required before the implementation phase.

Codeigniter

Codeigniter is an application framework, written in and using PHP. Aside from my previous experience with this framework, there are several other reasons why I am choosing to use it for this project:

1. It is well supported with a wide selection of plug-ins for common tasks such as user authentication
2. It provides a wide range of features relevant to this project such as FTP support and unit testing classes
3. MVC design pattern is strongly encouraged

[2] http://getbootstrap.com/

4. Codeigniter uses a MySQL database which is one of the most popular database systems and is well supported.

Bootstrap

Originally open-sourced by Twitter, this popular front end framework allows for simple implementation of fully responsive websites without needing to manually write responsive CSS. This means that the website will always scale to best fit the size of the browser window, regardless of the device (e.g. mobile, tablet, desktop) used.

FlatUI

FlatUI[3] is an extension of bootstrap used by the bskit application. It provides hundreds of icons and CSS shortcodes allowing for fast development of attractive websites. Using a user interface framework such as this will allow me to maintain a consistent and professional look across the entire site.

JavaScript and JQuery

JavaScript and its commonly partnered library JQuery are used to write the bskit builder, I will also need to use these technologies to provide interactivity such as buttons and AJAX calls for loading data from the database without refreshing the page.

4.4 Implementation Plan

As this project will be using code from other sources, it is important to get a general idea of what will need to be written from scratch, and what will need to be modified from the existing code before moving into the implementation phase. Following this, an implementation plan will be created with the aim of abstracting each major task into manageable sub tasks.

4.4.1 Bskit Builder Overview

The bskit builder [11] is an integral part of this application and will need to be carefully analysed in order to successfully implement it into the project. Screenshots of the application can be seen in figures 4.7, 4.8 and 4.9.

[3] https://designmodo.com/flat/

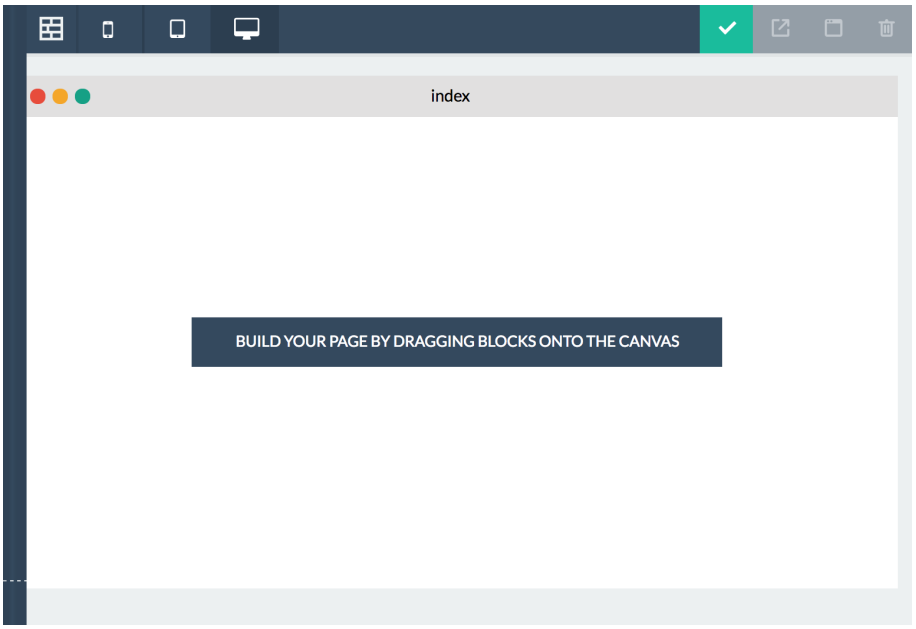


Figure 4.7: Bskit builder default view

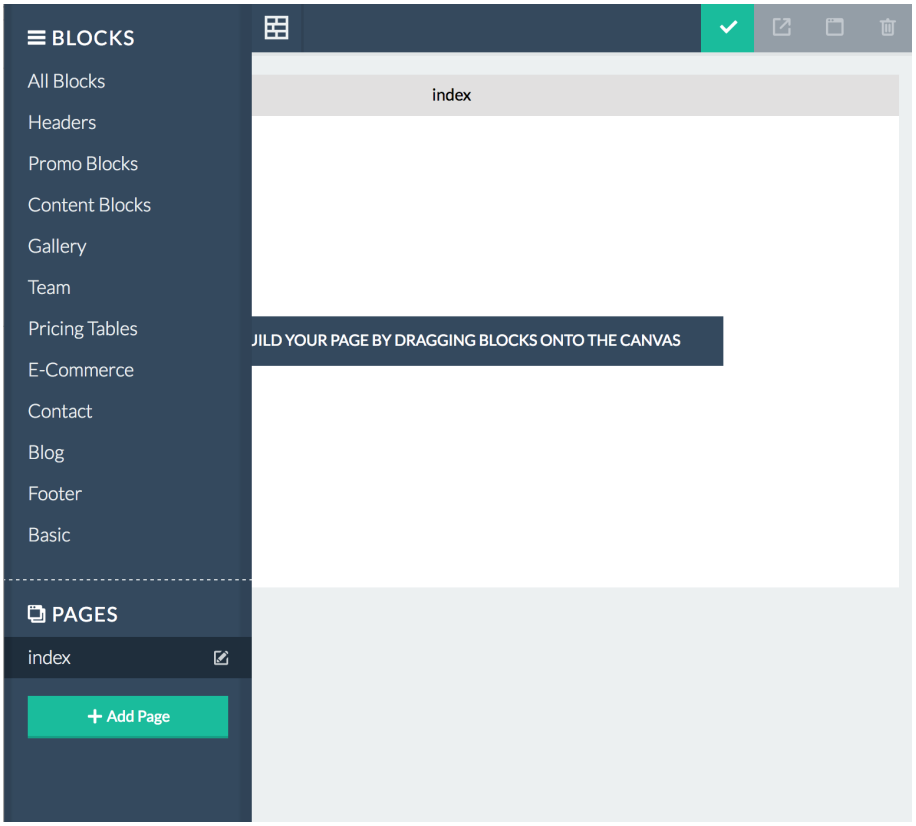


Figure 4.8: Bskit builder sidebar

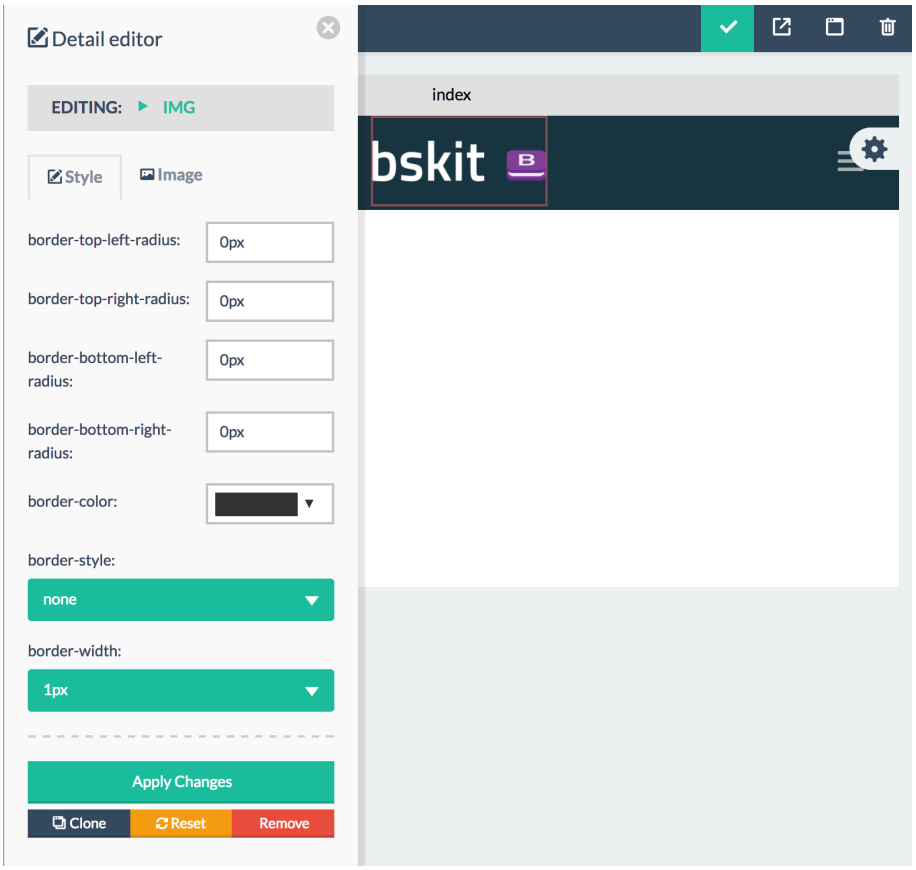


Figure 4.9: Bskit builder styles view

The existing code contains the following:

1. HTML 'blocks'
    - these are listed in a JSON file which is used to create the sidebar seen in figure 4.8
    - There are HTML components which cover all requirements (blog, photo gallery etc), however the E-Commerce block is not linked to an E-Commerce platform
2. Ability to edit text, style properties, and images
    - Images can be uploaded but there is no way to store them
    - Hovering over text can interfere with dragging and dropping
3. Contains a list of 'editable items' which are css ID tags that make elements clickable or unclickable
    - These will need to be stored for each site in the database
4. Allows multiple pages for a site
5. Allows the user to preview a created site
6. Allows the user to export a zip file containing the site contents

As well as integrating this code into my application, I will need to add the following features to meet the primary requirements:

1. Create an image library allowing images to be uploaded to a users account
2. Allow the user to save the current site to the database
3. Allow site settings to be saved for each site
4. Allow site templates to be created, this can be a separate section in the sidebar

Furthermore, I believe the following features are possible to implement in the given time frame:

1. Allow the user to upload the site to a web server
    - This can be done by storing ftp information in the database, and using Codeigniter's ftp upload functionality
2. Allow the user to see a version history of each site
    - This can be done by simply creating a 'revision' column in the database, and incrementing it each time the site is saved
3. Implement E-Commerce functionality by editing the existing E-Commerce HTML blocks

4.4.2 Major Tasks

As this project will be built on a MVC (Model View Controller) pattern, the project can be split into three major tasks.

1. **View** - This stage involves designing the front end of the application. HTML layouts, buttons, forms, navigation and the bskit builder will all be implemented during this stage

2. **Controller** - This stage will handle all business logic that involves updating the view with content retrieved from the database. The database cannot be accessed through the view (client side) as this would be a huge security flaw.
3. **Model** - This stage will instantiate the database, as well as methods which retrieve data from the database, but do not directly affect the view.

4.4.3 Minor Tasks

For each of the three major tasks, a list of minor tasks can be created.

Login page

1. View
    - Create simple login form and create user form
2. Controller
    - Form validation
    - User authentication
    - Change password form
    - Create account form
3. Model will be handled by Codeigniter's ion_auth extension

Sites Page

1. View
    - Create html file approximating the design in figure 4.4
    - Previews for each page will need to be generated, this can be done by modifying code used to generate thumbnails in the bskit builder sidebar (see figure 4.8).
2. Controller
    - Loads view with sites and user data from model, if admin load all sites, else only load current user's sites
3. Model
    - Return all users sites
    - Delete a site from the database

Users page

1. View
    - Link to users page only appears in nav bar for admin users
    - for each user, show users full name, login details and buttons to change password, delete account, or disable account

2. Controller
    - implement methods for updating user details, deleting account, disabling account and creating new account
3. Model
    - Model will be handled by Codeigniter's ion_auth extension

Images page

1. View
    - Display each image that a user has uploaded, and a form allowing a user to upload a new image. Next to each existing image should be a button allowing the user to delete the image
2. Controller
    - Produce a list of images that are linked to the users id
    - Given a form, upload a new image
    - Delete an image
3. Model
    - Store rules on image directory, size/dimensions etc
    - Store image reference and user ID

Builder page

1. View
    - Use bskit builder HTML as starting point and update CSS to match application
    - Fix sidebar in place
    - Add templates option in sidebar
    - Add option to edit site settings
    - Add save button
    - Add versions dropdown
2. Controller
    - Retrieve site data and pass to builder
    - Save site data
    - Save site as template
    - retrieve/update site settings
    - Update page settings
    - Publish site via FTP
    - Generate preview - use code from bskit

- Delete page - use code from bskit
- Add new page - use code from bskit
- Export site - use code from bskit

3. Model
    - Return/update site settings in database
    - save frame to database
    - save page to database
    - save site to database

Modals

A modal is a pop-up window which can be used in situations which require the users attention or input before an action can occur. The three types of modal which will need to be implemented in this project are:

1. Error Modals - will be generated using Codeigniter's flashdata functionality.
2. Confirmation Modals
    - Delete site confirmation
    - Delete user confirmation
3. Form Modals
    - Site settings - modal will contain editable options for the site such as metadata and title
    - Create site - modal will contain template creation wizard
    - Create user - modal will contain necessary information to add a new user
    - Export site - Adapt from bskit
    - Publish site - Add FTP details

Chapter 5

Implementation

The development of the application was carried out by following the plan in section 4.4, however, certain features were modified or unable to be fully implemented due to unforeseen circumstances which will be discussed later in this section. A user guide containing images of the finished product can be seen in Appendix 8.3 and the final application can be accessed at http://earlsconefarm.myqnapcloud.com/WebWizard.

5.1 Code Structure

According to the Codeigniter documentation, the code is organised into a strict directory structure [14]. A diagram showing each folder used in the project is shown in figure 5.1. Furthermore, the code was commented with sufficient detail such that the project could be continued in the feature should it be taken over by another person.

The 'config', 'contorllers', 'language', 'models' and 'views' folders are the only folders containing code written for this application. All other folders are pre-generated by Codeigniter.

The 'config' folder contains pre-generated files but the contents of these needed to be edited to give Codeigniter the information necessary for routing (see section 5.3.3), database connection and the base URL of the server. The 'controllers' folder contains the main business logic such as user authentication. The 'language' folder is pre-generated

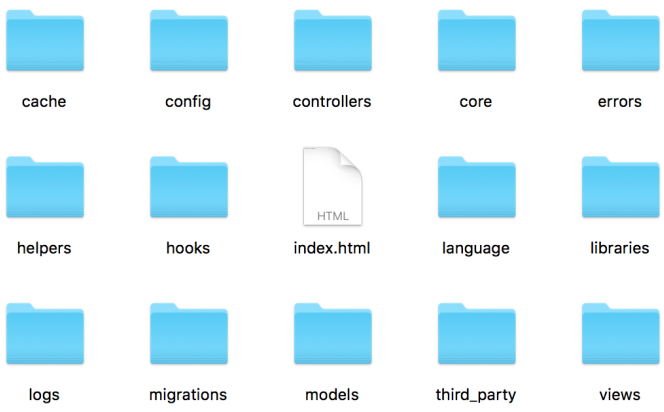


Figure 5.1: Codeigniter folder structure

by Codeigniter and contains a folder for each major language. Although only the English language file has been edited for this application, this setup would make it easy to translate the application to any other language in the future. The 'models' folder retrieves data from the database and sends it to the controller, and the 'views' folder contains the HTML code which is served to the browser.

Codeigniter's flashdata class was used to catch and display exceptions, the user can also edit the Codeigniter environment to 'Development' to enable console logs (see [15]).

5.2 Database design

Since user data and site data will need to be modified and retrieved by the application, a database needs to be developed. Codeigniter by default stores session data (cookies) and configuration data, and the ion_auth user authentication library stores data regarding login attempts, users and user groups. This application will also need to store data regarding each site that is created. A site can have multiple pages and each page is made of HTML frames. The database was built using PHPMyAdmin, a web based MySQL administration tool. MySQL and PHPMyAdmin were chosen for their reliability, ease of use, and synergy with Codeigniter and PHP.

Before building the database, it is important to first produce a relational model such that the duplication of data and referential integrity is reduced as much as possible. The database for this project should conform to the third normal form (3NF), famously defined as '[Every] non-key attribute must provide a fact about the key, the whole key, and nothing but the key.'[16]. Following these guidelines, I designed the database ensuring it conformed to 3NF. The final database (not including tables generated by Codeigniter) is shown in figure 5.2.

5.3 Development

The development of this application can be split into two parts:

1. The code handling the sites view, users page and image library, written using Codeigniter
2. The implementation of the bskit builder code into the application, written mainly in JavaScript

This section will focus on the Codeigniter setup and development of the application, the first step being creating HTML pages based on the designs in section 4.2.

5.3.1 View

Each view page was first written in HTML and adapted into a dynamic PHP file using Codeigniter. The pages were built on a standard Bootstrap 3 grid layout to make them adaptable to any screen size as stated in the requirements. The FlatUI extension was used for styling elements (see figure 5.3).

Once the rough HTML files were complete, they were split into different components and organised in the following structure:

1. Assets

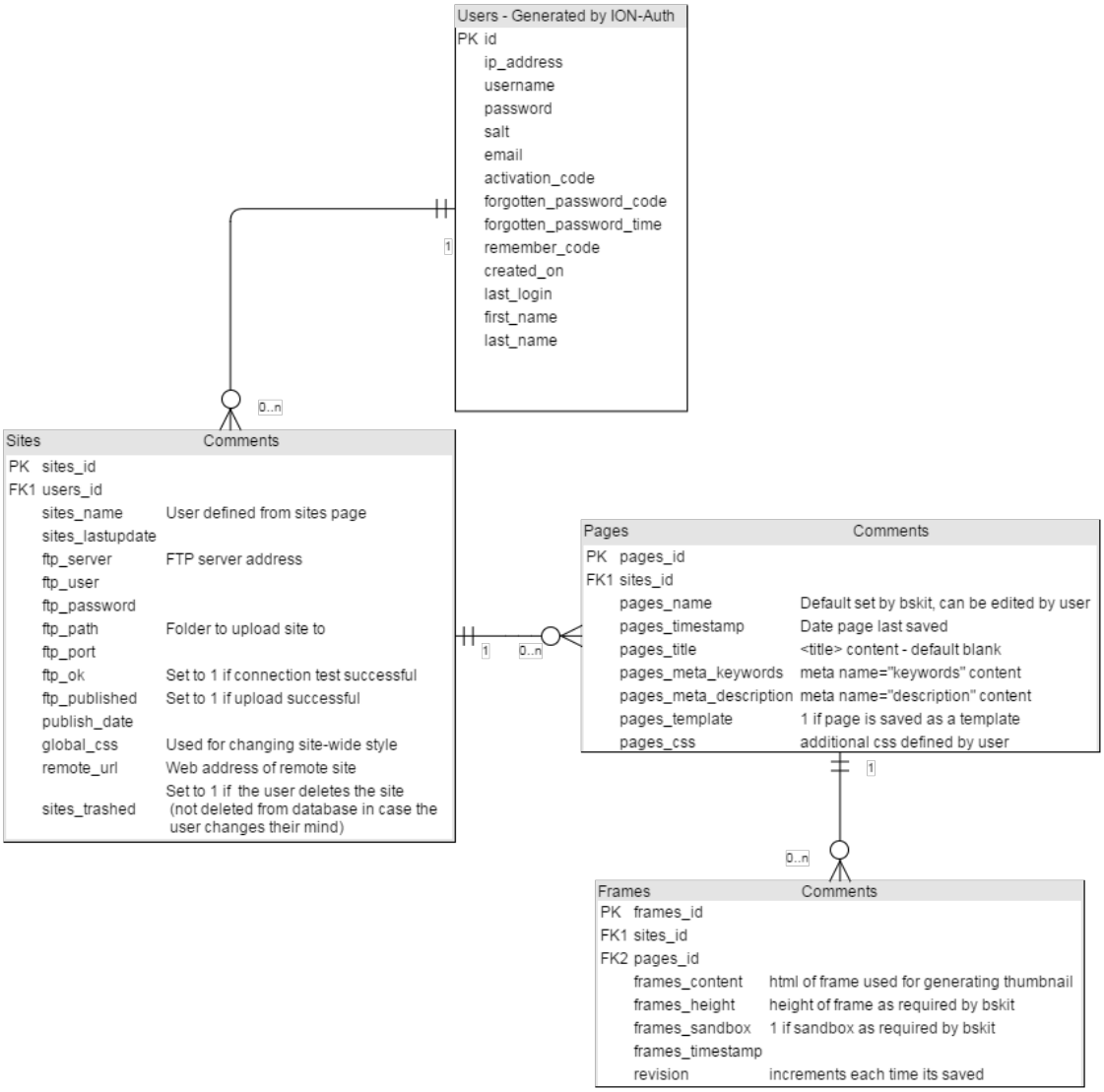


Figure 5.2: Relational database diagram

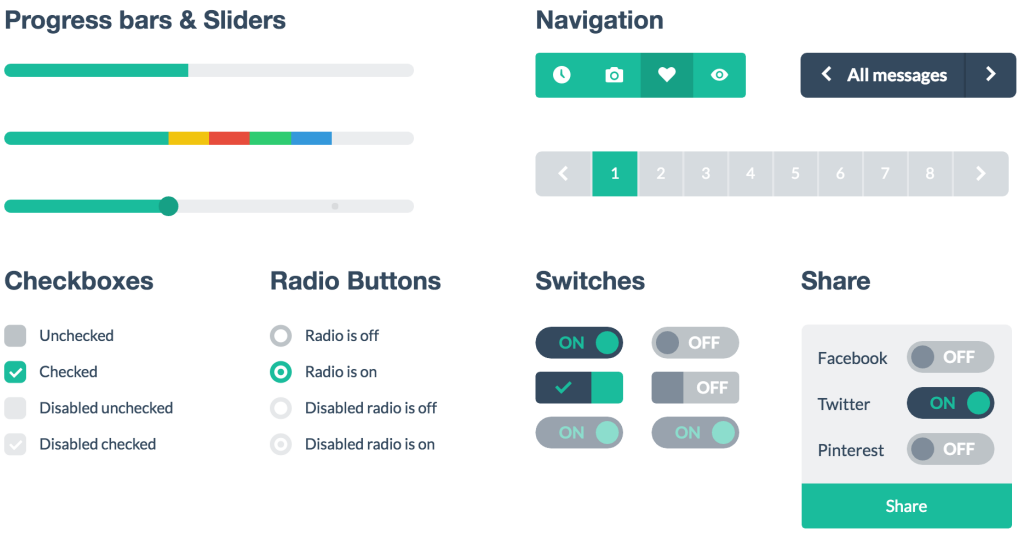


Figure 5.3: Example of some standard Flat UI elements

- Images.php - see section 5.5.3 for more detail

2. Auth - contains files based on the ion_auth documentation to manage user actions [17]
    - create_user.php
    - forgot_password.php
    - login.php
    - reset_password.php
3. errors - preset pages for handling various errors
4. partials - HTML fragments which are used in various repeatable situations such as modals
    - exportdetails.php - form used for inputting ftp details
    - ftplist.php - shows folder directory of remote server
    - myimages.php - grid of each image linked to a users ID (see section 5.5.3)
    - pagedata.php - form for updating page meta-data
    - revisions.php - drop down list containing each revision in database
    - sitedata.php - form for updating site data
    - success.php - success modal
    - templateframes.php - see section 5.6.3
    - userdetailsform.php - form for updating user details
    - users.php - list of users in database
5. shared - these files are used accross multiple views but remain constant, see section 5.3.2
    - footer.php - appended after main body - contains js links
    - header.php - *¡head¿* section - contains css links
    - modal_account.php - see section 5.5.4
    - modal_createsite.php - see section 5.5.4
    - modal_deletesite.php - see section 5.5.4
    - modal_sitesettings.php - see section 5.5.4
    - nav.php - see 5.3.2
6. sites
    - create.php - builder interface
    - sites.php - displays list of sites
7. users
    - users.php - see section 5.4

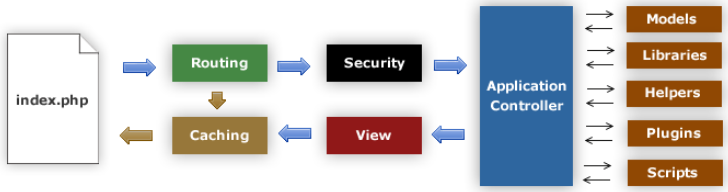


Figure 5.4: Codeignitor data flow

5.3.2 Navigation

When a user is logged in with an admin account, a 'users' link becomes available in the navigation bar, otherwise the only links are to the sites page and the images page. As is the standard with Codeigniter applications [18], the navigation bar view is stored in the 'shared' folder and is imported into each page with the following code:

```
1 <?php $this->load->view("shared/nav.php");?>
```

This improves the readability of the code, as well as lowering the coupling and improving maintainability by removing the need to change the same code in each file that uses the navigation bar.

5.3.3 Page requests

Codeignitor provides the diagram shown in figure 5.4 to demonstrate how a page request in handled.

The routing is handled by a provided file in the 'config' folder called 'routes.php'. Here, routes are defined for logging in, logging out and loading a specific site or template (by site ID). Commonly accessed pages are automatically cached in the server to increase speed and reduce server processing time. Codeigniter also provides security, preventing unwanted access to the web servers directories and applying preventative messures against cross site scripting and SQL injection attacks [18]. This helps the project meet the security requirements set in section 3.4

5.4 Multiple user support

Since this project requires that multiple users be able to create and save websites, the application needs to store data in such a way that it is accessible by only an admin user, or the user that created it. WebWizard stores site data (which in turn stores page and frame data) with a reference to the user ID that created it. When loading a site or displaying it in the sites page (after logging in), the site is only loaded if the user ID in the database matches the user ID of the current user. Simply by using PHP if statements, admin users are able to load and see all sites stored in the database.

5.4.1 Authentication

For this project, I used the authentication library ion_auth [17] to handle the secure storing of passwords in the database, as well as providing easy implementation of password reset and captcha features.

5.4.2 User settings

As the 'users' page can only be accessed by admin users, there was a need to add a method for a non-admin user to edit their personal email address and password. This was implemented by changing the 'Logout' button in the navigation bar into a drop down list containing the items 'Logout' and 'My Account'. The 'My Account' button displays a model containing the same information as the 'users' page, but instead of looping through each user in the database, only the current users information is shown. This simply uses the ion_auth *is_admin()* function.

5.5 Sites Page

The Sites page is the first page the user will see after successfully logging in. If the user has never created a website before (even if the user deletes a site, the entry remains in the database so this can be determined by a simple check for the users ID in the database), the existing sites list are replaced with a welcome message.

5.5.1 Create wizard

The create wizard was implemented using a drop down box and a form inside a modal. The form contains ten options and each of these links to the first ten entries in the Sites table stored in the database. The first ten entries are marked as 'deleted' so they will not appear in the existing sites panel, however if a site is accessed which has an ID of less than 10, the system will ignore the fact that it is deleted, allowing it to be opened. When the site is opened, saving it also has a special effect if the ID is less than 10. If the user is an admin user, the template is updated and the site saves as normal. If the user is not an admin, the site will be saved as a brand new site by first running the *createNew()* function which inserts a new row in the database with the users ID, and then the *update($siteData, $pages)* function with the newly created sites ID passed as an input.

Although it may seem to be an obvious limitation that the system is restricted to ten template sites, this number can easily be extended by simply increasing the site ID of any existing sites to make room for more templates, and then changing the 'less than 10' line to the new number of templates. This will work as the system does not hard code site IDs for any reason other than templates.

5.5.2 Existing sites

The second half of the sites page displays a list of existing sites owned by the user (or all sites in the database if logged in as an admin). This is implemented in two main steps.

1. Use controller function *siteData()* to return the *$sites* variable containing data for all sites (if admin) or existing sites linked to current user ID
2. In the *view* → *sites* file, for each site in *$sites*, create a new 'site' div which contains:
    - The title of the site
    - A preview of the site
    - The Users full name

- The dates the site was created and last updated
- The number of pages in the site
- If the FTP_Published variable is 1, display the URL the site was published to and the date it was last uploaded, else display 'site not yet published'
- An 'Edit this site' button which links to the builder page for the current sites ID
- A 'Delete site' button which calls the *deleteSite* method in the model
- A 'Settings' button which brings up the settings modal (see section 5.5.4).

5.5.3 Images Page

The images page provides a way of storing images for use in the builder application. Images are selected using the Codeignitor File Uploader class [19] and are validated using the constraints on file size, width and type as specified in the codeignitor configuration. Because a user should only be able to see images that they have personally uploaded, the images can not simply be saved in the same folder regardless of who the user is. Therefore, images are uploaded to *$this->config->item('images_uploadDir')."/".$userID* which produces a new subfolder with the users ID as its name in the images directory. The images are then uploaded and retrieved from the folder which matches the users ID, this ensures that no user can access other users images and no extra, unnecessary information needs to be stored in the database.

5.5.4 Modals

Although the majority of modals used in this application simply inform the user of success or failure of a task, or ask for confirmation before deleting a site or a user, some of the modals are more complex as they are needed to update settings in the database via AJAX calls.

Settings Modal

This modal is found in two places, on the builder page and on the existing sites page. The HTML for the modal is split into two files, one which contains the header and footer (title text and 'save' or 'cancel' buttons), and one which contains a form laying out each editable component of a site. This includes site title, metadata etc. As with other modals, when the 'site settings' button is clicked, a JavaScript method is called which loads the first view. Unlike normal modals, it also calls a method in the controller via AJAX. This method fetches the requested site data from the database using a model method and returns it to the JavaScript function along with a response code. If the response code indicates success, the function fades in the settings view containing the existing data.

Upon clicking the 'save' button, another AJAX call is made using the controller to update the new information. If form validation fails, the method will return an error and the modal will highlight the appropriate field with an error message, otherwise the modal will fade out and the data will be updated.

Publish Modal

When the user clicks the 'publish' button, the publish modal will be shown. The publish modal contains a directory list of site assets and site pages. The user then selects the assets and pages they wish to upload and clicks 'upload' which will call the *publishSite()* JavaScript function. This function first tests the FTP connection with the provided details and returns the appropriate error if there is one. It then saves the site via AJAX before preparing the assets and pages by looping through each page using code modified from the bskit builder to create full HTML files from each block on the page. Finally an AJAX call will upload a file, displaying a loading icon until a response code is received. If the response code indicates an error, the icon changes to an X and an error message is shown, otherwise the icon changes to a tick-mark and the application will attempt to upload the next file. If all of the files have successfully been uploaded, the application will show a success message and will update the *ftp_published* column in the database to '1'.

5.6 Site Builder Frontend

As mentioned in section 4.3.1, this application will use an existing piece of software called 'Bskit Builder' to handle the functionality of the site builder page.

5.6.1 Bskit Builder Overview

The bskit builder is written using the JQuery JavaScript library. It allows a user to drag HTML blocks onto a page, create a site using multiple blocks and pages, edit the HTML and CSS of each block using a 'style' menu and export or preview the finished site.

The code uses nine JavaScript files, packaged together using a tool called Browserify in order to reduce page load times [20]. The files that will need to be edited are:

1. builder.js
    - As I found the auto-hiding sidebar frustrating to use, I will remove the code that hides the sidebar to make it static
    - The bskit builder has three states, edit content, edit style and drag block. In its current form there is no differentiation between the three. Therefore, I will add a toggle to provide greater clarity to the user
    - Adding template functionality
2. content.js - For security purposes and to allow easier integration with the database image uploads will be handled by PHP

The code also relies on several third party JavaScript libraries and frameworks:

1. Bootstrap - Responsive framework discussed in section 4.3.2
2. Flat-UI - Bootstrap extension used for styling
3. ChosenJS - jQuery plugin used for select boxes
4. Zoomer - jQuery plugin used to generate thumbnails given an iFrame
5. spectrumJS - used to add colourpicker functionality to the style menu

5.6.2 Modifying the UI

The first step in building the application was modifying the user interface (UI) of the bskit builder to bring it inline with the design laid out in section 4.2. To make the sidebar static I used the jQuery plugin 'mCustomScrollBar' which contains detailed documentation on how to implement it [21]. The top bar (see figure 4.7) was also removed to make space for the sites navigation bar. However this meant the 'save', 'preiview', 'export' and 'delete' buttons had to be relocated. This was done using Flat-UI buttons.

The next step was to implement the state toggle between 'drag', 'content' and 'style' mode. In the bskit code, there is a button called 'move' which when clicked, called the activateBlockMode() function which allowed the block to be dragged and dropped.

To make this an easier process for the user, three radio buttons where added to the top of the page in place of the bskits screen re-size buttons. As the screen can easily be resized using the browser, I felt it was more important to have control over the blocks edit mode than redesign the page to make room for the sizing buttons.

Elements Mode

This is the default mode which is activated whenever there is one or more frames in the canvas. The function 'toggleFrameCovers('on')' is used to make the blocks draggable (called by clicking the move button in the original code), and the clicking of individual elements is disabled.

Content Mode

When this radio button is clicked, a function called *activateContentMode()* is called which disables drag-and-drop and uses an edited bskit function to allow HTML text elements to be clicked. When clicked, a modal will appear, using the plug-in 'summernote'[1] to allow the user to edit the text content. In the bskit builder code, the plug-in was overlaid on top of the standard builder screen, but as Codeigniter makes it easy to add modals and due to the increased clarity given by them, I chose to implement it in this way.

Styling Mode

Similarly to the content mode, the styling mode radio button calls a function which allows certain HTML elements to be clicked. When clicked, the content editor will appear, unchanged from the original bskit builder.

5.6.3 Template blocks

The biggest change made to the Bskit builder code was the addition of template blocks. A template block is seen by the bskit builder as a single frame until is is dragged onto the canvas, at which point it is loaded as multiple frames. In order to achieve this, a value called 'pages_template' was added to the database. This value is set to 0 if a page is saved normally, but as 1 if a page is saved as a template. A new JavaScript file was also created and is called when a page is saved as a template. This file makes an ajax call to the PHP function 'tsave' which is shown below:

[1] http://summernote.org/

```
<?php foreach( $pages as $key => $frames)?>
    <?php
        $frameIDs = array();
     $indivHeights = array();
     $totalHeight = 0;

     //for each frame in current page
     foreach( $frames as $frame ) {


      //frame ID's
      $frameIDs[] = $frame['id'];


      //total height
      $totalHeight +=$frame['height'];


      //individual heights
      $indivHeights[] = $frame['height'];



     }
     //sets page name and id
     $pageName = $frames[0]['pageName'];
     $pageID = $frames[0]['pageID'];
     //joins all frame ID's separated by -
     $frameIDstring = implode("-", $frameIDs);
        //joins all frame heights separated by -
     $indivHeightsString = implode("-",$indivHeights);
    ?>
    <li class="templ"
        data-frames="<?php echo $frameIDstring;?>"
        data-heights="<?php echo $indivHeightsString;?>"
        data-name="<?php echo $pageName?>"
        data-pageid="<?php echo $pageID;?>">
     <iframe frameborder="0" scrolling="no" src="<?php echo site_url('
    temple/index/'.$key)?>"
                data-height="<?php echo $totalHeight;?>"></iframe>
    </li>
?>
```

As shown by this code, the iframe is generated using the combined total height of all frames, but the individual frame ID's and heights are still stored, allowing the bskit builder to process it as multiple frames.

5.6.4 FTP Upload

Implementing the ability to upload a site to a server via FTP was relatively straightforward due to the FTP functions and documentation provided by Codeigniter ([22]). The *pathInfo* function was used to retrieve a list of folders used by the site (images, css, js etc.) and used to create a list of site assets which can be selected by the user (see figure 5.5). This allows the user to opt out of uploading unnecessary files, as I found it frustrating to repeatedly wait for images to be uploaded when I had only made text based changes to the site during testing.

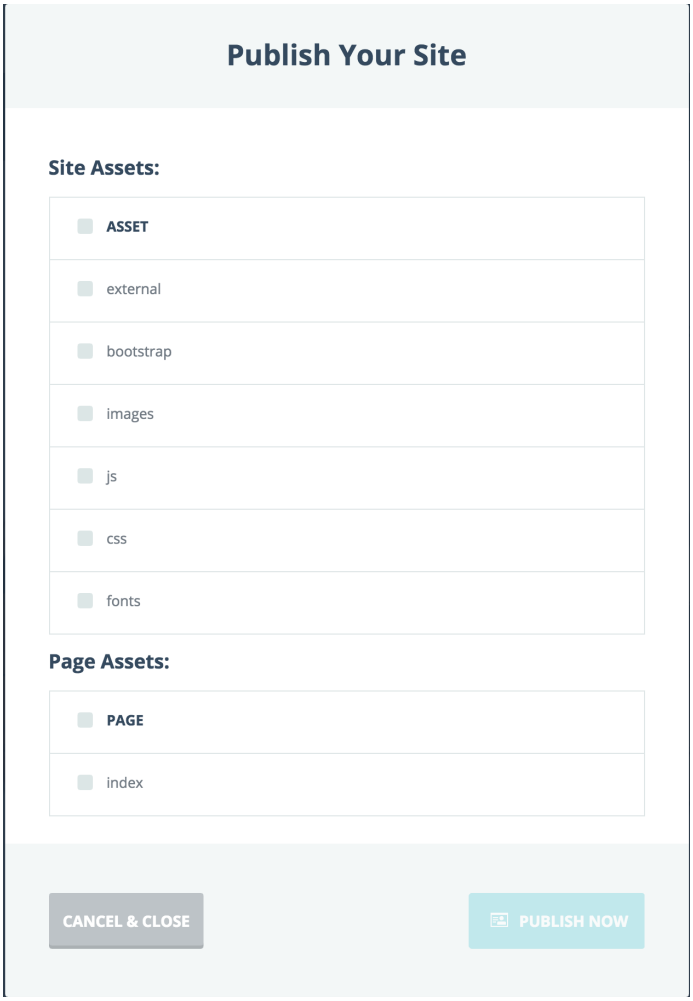


Figure 5.5: Site assets, as generated by the pathInfo function

5.6.5 Revisions

Simply by adding a 'version' field in the 'sites' table in the database, version functionality can be added. The application stores page data in a new row each time the site is saved, upon saving the new data, the 'version' value is incremented. Next, a 'versions' drop down box was added to the user interface which contains a new item for each row in the database where the page ID and site ID are equal, ordered by 'version'. Upon clicking a version, the page is refreshed using that versions data. So that users can get a preview of the revision before loading it into the database, I also added a 'preview' icon which simply calls the *preview($page)* method with the first page of the site as a parameter.

5.7 Optimisations and issues

During the implementation process, I faced several issues which forced me to make changes from my original design.

5.7.1 Persisting issues

Whilst there were many features I was unable to implement due to time technological constraints, I was still able to achieve all of my primary requirements and given more time, am confident that some of the following issues are easily solvable.

Create site wizard form

Although the site wizard functions well, the form requesting details such as the sites title is not currently linked to anything. This was seen as a low priority feature as it was not included in either the primary or secondary requirements and as a result, I did not have enough time to implement this feature.

Delete Image

In the final version of the application, clicking the 'delete' button next to an image in the images page does not call the 'deleteImage' method. At a glance, the code looks correct to me but efforts to fix it where unsuccessful, it was therefore determined that I would come back to it if I had time, and that other issues took priority. This issue does not affect the primary or secondary requirements, however it was pointed out by several users during testing (see 6.2.4) and would most likely need to be fixed before this application is released to the public.

Preview Site

Although the preview produced by the bskit *previewSite()* function looks identical to the final export, links to other pages in the site do not work due to the bskit builder being a front end only application. Previewing all pages with working links would be possible with PHP by storing the sites generated HTML in the database, however it is not a necessary feature to meet any requirements and a simple message in the preview modal warns that sites with multiple pages will not have working links. One page sites with links to blocks

on the same page will function fully however, so it is also suggested to use this as a workaround.

Site previews

In the final version of the application, the thumbnails generated for each site in the 'sites' page only contain the first frame of the site. This results in sites which have the same header element but different bodies looking identical in the preview. As the code for generating these thumbnails is adapted from the bskit builder application (used to generate thumbnails in the sidebar), I was unable to debug the problem before the time limit was up.

5.7.2 Optimisations

The bskit builder application is resource heavy, and in an attempt to make the application load quickly, the creators generated a single minified[2] JavaScript file using the tool 'browserify'[3]. This tool contains a source map which allows supported browsers to view the JavaScript in its original un-minified form. As I was adding several JavaScript files of my own to the application, I needed to reconfigure browserify to compile my files along with the origional bskit builder files.

Although some issues were immediately apparent during implementation, more in depth testing needed to be carried out in order to analyse how well the application had met the requirements.

[2]Minified - stripped of all unnecessary text to make a file that is unreadable by humans but faster to load by browsers

[3]http://browserify.org/

Chapter 6

Testing

In order to ensure the application met the requirements listed in section 3.4, it is important to thoroughly test the system using multiple methods of testing to cover all aspects of the project. This section will focus on the two most important testing approaches for a web application, integration and usability tests [23].

6.1 Integration Testing

The integration testing process involved testing each component of the application with respect to the corresponding requirements. The full test results can be viewed in Appendix 8.1.

6.1.1 Findings

The test results show almost every feature works as intended, however there are a few notable exceptions:

1. The button for deleting images in the images tab does not work
2. Adding a title for the site in the create wizard does not affect the site
3. Text within click-able elements can not be edited in the builder
4. Google maps does not load in the contact blocks
5. Portfolio 3 template does not load

The image deletion function and the site wizard form were expected to fail the tests because they were not able to be implemented in time as mentioned in section 5.7. However, the issues with the builder interface and template were not noticed during implementation.

The Google maps issue was simple to trace; the maps implementation was part of the bskit builder HTML files and was using a depreciated implementation of the Google maps API. This was fixed by simply updating the code to match Google's current guidelines.

The issue with editing certain elements in the builder was investigated, but fixing it would have required a large amount of time editing the bskit code and was deemed a low priority fix.

The template that would not load was due to it not being saved properly during the template creation process, this was fixed quickly.

6.2 Usability Testing

In addition to the integration testing, a simple usability test was created and 5 volunteers were found to test the system.

6.2.1 Target audience

Due to time constraints, only one volunteer was a small business owner, however, as the other volunteers all had varying levels of technological knowledge, the test results still provided valuable insight into the overall usability of the system.

6.2.2 Test environment

In addition to the test instructions, each user was given a link to the web application and a copy of the user manual (see Appendix 8.3).

6.2.3 Test Process

The test consisted of 5 steps, which were repeated once for this application, and once for the top rated[1] existing website builder 'weebly'. Test steps:

1. Create a new account and log in
2. Create a website with an image gallery
3. Edit the site to include 'index', 'contact' and 'about' pages
4. Add the logo 'logo.png' to the site.
5. Publish the site (ftp server details were given)

6.2.4 Findings

All five users were able to complete each step, and although the time taken varied fairly considerably, each user posted a faster time for this application when compared to the competitor. The block based design was also praised for its simplicity and the ability to upload the site to any webserver was seen as an advantage over weebly. However, there were several weak points pointed out by the testers. User feedback indicated that although it took longer to set up the competitor application, it felt more professional and fluid compared to this project. Additionally, the two features that were not implemented correctly (see section 5.7) were noticed and cited as a major reason for the competitor site being smoother and more professional.

[1] As rated by independent review site www.sitebuilderreview.com

6.3 Testing Evaluation

As this project was completed with strict time and resource restraints, it is fully expected to not perform as fluidly as an application that has been under development for years and received millions of dollars in funding. I find the test results to be very positive and regard the project as a success. Although not every feature works as intended, the main objectives have been exceeded and most of the test participants completed the task very quickly and admired the simplicity and design of the product.

Due to time limitations and sub-optimal planning, I was unable to perform unit tests on the application. If I were to repeat this project, I would make sure to build a unit testing platform using standard software such as phpt[2]. However, I am confident that the integration and usability test results show that the project has comfortably met all of its primary, and some of its secondary requirements.

Additionally, several potential improvements were suggested by the users which have been explored in section 7.5.

[2]https://qa.php.net/write-test.php

Chapter 7

Evaluation and Conclusion

This section aims to critically assess the success of the finished project. Furthermore, it will discuss alternative methodologies that, in hindsight, may have improved the development process of the project and led to a better system. Finally, it will suggest future extensions and improvements that could be made to the project in order to improve its functionality.

7.1 Project Results

To understand the extent to which the project has succeeded, the finished system will be cross referenced with the initial requirements laid out in section 3.4.

7.1.1 Requirements cross reference tables

From the requirements cross reference tables (see tables 7.1, 7.3 and 7.2) we can see that all primary requirements, along with several secondary requirements, have been met to a satisfactory level.

Table 7.1: Use Case Evaluation

Actor	**Case**	**Description**	**E/D**	**Present?**
End User	Navigate Site	The user will want to get around the site	E	✓
End User	View Content	The user must be able to view all content on the page regardless of the device or browser they are using	E	✓

End User	Is search-able by search engines	The user may want to add meta-data tags to make the site show in search engine listings	D	✓
End User	e-commerce functionality	The user will want to purchase multiple products at once and have a way of organising there desired purchases	D	X
Owner	Create site	The owner of the site will need to add and edit features of the site	E	✓
Administrator	Edit site	The owner may act as or hire an administrator who will want to edit the site to update content or add features	E	✓
Administrator	Host site	The administrator will need to know how to get the site online	E	✓

Key: E = Essential D = Desirable

Table 7.2: Functional Requirements Evaluation

Function	**Description**	**Testing method**	**E/D**	**Present?**
Code created must be well formed	Code is pre-written and split into modules. The user can selects any number of modules but cannot modify the code	Validation using W3schools online validation tools	E	✓
User interface is simple to use	Written using as few components as possible	Usability survey	E	✓

Displays audio and video	Implemented using HTML5 to ensure user friendly experience	Can video and audio be played on the generated web site	E	✓
Contains online shopping functionality	Ecwid API allows ecommerce functionality to be added for free with the option of upgrading to paid plans as the business scales	Client should be able to add product catalogue and receive payments through a specified method	E	X
Allows copy and paste of rich text e.g bold/italic text	NoteJS seems to support this functionality	Visual test, text copied from a word document or another website should not lose its formatting	D	✓
The site should have navigation functions	Each created site with more than one page will include a navigation bar and site map	Usability survey	E	✓
Site should have a shopping basket system for purchasing multiple products	Possible with JavaScript	Usability survey	D	X
Instructions	Instructions should contain information on both using the program, and hosting and editing the site afterwards	Usability survey	E	✓
Site is mobile friendly	Written in a fully responsive framework such as bootstrap. This should happen automatically without any user input	Usability survey	E	✓
Sites design can be automatically generated	The application can generate a starting template based on information given by the user	Usability survey	D	✓

Key: E = Essential, D = Desirable

7.2 Requirements shortcomings

Although many of the secondary requirements were implemented in some form, there are two that were not. The reasoning behind these are listed for each failed requirement below:

1. **E-commerce functionality**: This was originally intended to be implemented using the ECWid e-commerce API. However, because ECWid requires a user account for each API key, the user would have to sign up for an account, copy the API key and input it into the site settings form. This wouldn't be particularly difficult to implement but there was not enough time to complete it without some desirable requirements being omitted.

2. **Automatic colour scheme generation**: Similarly, it seemed more appropriate to focus on the essential requirements than implement this functionality.

7.3 Conclusion

As stated in section 1.3: '*The main aim of this project is to create a simple to use website builder that can easily produce a fully responsive website suitable for a small business.*'

I believe the final product, although not without its issues, fulfils this objective. The application is capable of handling multiple users whilst keeping personal data secure, generating preset templates which can be modified without any knowledge of code whilst always maintaining a fully responsive layout. Because of this, I consider the project a success, however, it is far from perfect and there are many limitations and improvements which can be made.

The most challenging part of this project was learning how to effectively combine existing technologies into a brand new product. Although many of the libraries used contained detailed documentation and guides for developers, the bskit builder code was difficult to understand. Although its code contained clear comments, its complexity made it time consuming to modify and debug. However, I have learnt a huge amount from this project, both in terms of programming, and also in terms of good analysis and design practises.

One of the original objectives defined in section 1.3.1 was: *'The application should be easy to understand and use'*. User testing in section 6.2 indicates that despite the bugs, the application was as easy, or even easier, to use than the highest rated existing site builders. This is because, as was intended from the beginning, the application sacrifices design flexibility by using pre built blocks in order to increase ease of use. Overall it is evident that the final application has exceeded its expectations and with some additional work, could become a commercially viable product.

Table 7.3: Non-functional Requirements Evaluation

Function	Description	E/D	Present?
Program must not insecurely store personal data	A users data is only accessible if the user is logged in	E	✓
Users data must be secure	ion_auth handles security	E	✓
The style and colour scheme of the site can be determined by analysing a logo or photo chosen by the user	Should be possible using a colour analysis JS library	D	X
The site should appear in search engines	Needs to be crawl-able and include the correct meta data, the user can add additional meta data if they choose to	D	✓
User interface is platform agnostic	Written for the web using plugins supported by all browsers	E	✓

Key: E = Essential, D = Desirable

7.4 Limitations

The first major limitation of the application are the bugs which hinder the user experience and make the product look unprofessional. The two major bugs which should be top priority to fix are the site preview bug and the delete image bug, mentioned in section 5.7. The next set of issues that should be addressed are issues with the bskit builder code. Although these issues may be fixed by future versions of the bskit builder, the project has implemented the code in such a way that migrating to a newer version of the bskit builder would involve a decent amount of code refactoring. In hindsight, although it would have taken longer, it would have been a better design choice to split code into modules written by me and modules belonging to the bskit builder. This would significantly improve both code readability and ease of future maintenance work. Finally, the create site wizard would benefit from thumbnails of each template and a working user-data input form. Although this project is based on a block based principle which is different from other site builders, there are many features that the leading website builders contain which would be valuable if implemented into this project. These include e-commerce functionality, Automatic input of users contact information and a larger selection of templates.

7.5 Future Work

Based on the user testing (see section 6.2) as well as my own experiences with developing this application, there are several additional features that could be developed in the future:

1. E-commerce features - could be implemented using a third party API, or built into the site using the database to store product info.
2. Intelligent template creation based on user provided data - In late 2016, Wix (one of the leading site builder applications) announced an artificial intelligence based website generator[7], this idea could be incorporated into this product by choosing certain blocks based on the users requirements.
3. Monetising the application, either by creating a subscription based platform which disabled user accounts if they stopped paying, or by using a 'freemium' model[24] and charging for extra blocks.
4. Implementing rigorous unit testing and conducting more detailed user tests to improve the maintenance side of the project.

If these additional features were implemented to a high standard, I believe that this product could be a valuable tool for business owners wanting a quick but professional website.

Bibliography

[1] Internet Live Stats. *Total number of websites in real time*. http://www.internetlivestats.com/watch/websites/. 2017. (Visited on 16/04/2017).

[2] Ingrid Lunden. *Universe, a mobile-only website builder, lets you create pages in 'under a minute'*. 2017. URL: `https://techcrunch.com/2017/03/22/universe-a-mobile-only-website-builder-lets-you-create-pages-in-under-a-minute/` (visited on 16/04/2017).

[3] Mike Thelwall. "Effective websites for small and medium-sized enterprises". In: *Journal of Small Business and Enterprise Development* 7.2 (2000), pp. 149–159.

[4] Geoff Simmons, Gillian A Armstrong and Mark G Durkin. "A conceptualization of the determinants of small business website adoption: Setting the research agenda". In: *International Small Business Journal* 26.3 (2008), pp. 351–389.

[5] Johnny Lee. *Should You Consider Sitebuilders?* 2017. URL: `https://www.webhostingplanguide.com/reviews/sitebuilder-com-review/` (visited on 11/03/2017).

[6] BCS. "Code of Conduct for BCS Members". In: (2017). URL: `http://www.bcs.org/upload/pdf/conduct.pdf` (visited on 11/03/2017).

[7] Wix. *Wix ADI: Design AI That Will Change Website Creation*. 2016. URL: `https://www.wix.com/blog/2016/06/wix-artificial-design-intelligence/` (visited on 16/04/2017).

[8] GoDaddy, RedShift Research. *Small business survey 2015*. https://www.godaddy.com/garage/wp-content/uploads/2015/09/GoDaddy-Global-Small-Business-Report-2015.pdf. 2015. (Visited on 16/04/2017).

[9] M Usman and John Davidson. *How to Make a Free Website*. Mendon Cottage Books, 2016.

[10] Manisha Jailia et al. "Behavior of MVC (Model View Controller) based Web Application developed in PHP and. NET framework". In: *ICT in Business Industry & Government (ICTBIG), International Conference on*. IEEE. 2016, pp. 1–5.

[11] Bootstrap starter kit. *Bootstrap Starter Kit Demo*. http://bootstrapstarterkit.com/bskit-demo/. 2016. (Visited on 16/04/2017).

[12] Kai Lei, Yining Ma and Zhi Tan. "Performance comparison and evaluation of web development technologies in php, python, and node. js". In: *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*. IEEE. 2014, pp. 661–668.

[13] John Hightower. "Mean: a full JavaScript stack for web development: conference tutorial". In: *Journal of Computing Sciences in Colleges* 31.5 (2016), pp. 109–110.

[14] Rob Foster. *CodeIgniter web application blueprints*. Packt Publishing Ltd, 2015.

[15] Arjun. *Uploading file via FTP using CodeIgniter*. https://arjunphp.com/uploading-file-via-ftp-using-codeigniter/. 2016. (Visited on 16/04/2017).

[16] William Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory". In: *Commun. ACM* 26.2 (Feb. 1983), pp. 120–125. ISSN: 0001-0782. DOI: 10.1145/358024.358054.

[17] Ben Edmunds. *Ion Auth Docs*. http://benedmunds.com/ion_auth/. (Visited on 16/04/2017).

[18] CodeIgniter 3.1.4 documentation. *CodeIgniter User Guide*. https://www.codeigniter.com/userguide3/. 2017. (Visited on 16/04/2017).

[19] CodeIgniter 3.1.4 documentation. *File Uploading Class*. https://www.codeigniter.com/userguide3/libraries/file_uploading.html. 2017.

[20] Tim Ambler and Nicholas Cloud. "Browserify". In: *JavaScript Frameworks for Modern Web Dev*. Springer, 2015, pp. 101–120.

[21] Malihu. *jQuery custom content scroller*. 2016. URL: http://manos.malihu.gr/jquery-custom-content-scroller/ (visited on 16/04/2017).

[22] CodeIgniter. *FTP Class — CodeIgniter 3.1.4 documentation*. https://www.codeigniter.com/userguide3/libraries/ftp.html. (Visited on 16/04/2017).

[23] Hasan Javed et al. "Model Based Testing for Web Applications: A Literature Survey Presented." In: *JSW* 11.4 (2016), pp. 347–361.

[24] Clarence Lee, Vineet Kumar and Sunil Gupta. "Designing freemium: a model of consumer usage, upgrade, and referral dynamics". In: *Retrieved Dec* 17 (2013), p. 2014.

[25] Theodoros Amanatidis and Alexander Chatzigeorgiou. "Studying the evolution of PHP web applications". In: *Information and Software Technology* 72 (2016), pp. 48–67.

[26] Sujan Pandey. "Development of a Web Application (iDomain) in the CodeIgniter Framework: Hosting Accounts Management System Development". In: (2016).

[27] Justin Khoi Bao Han-Nguyen. "Developing a PHP CMS". PhD thesis. California State University, Northbridge, 2016.

[28] Avenir.ro. *Create a CMS using CodeIgniter - tutorial*. http://avenir.ro/create-cms-using-codeigniter-3/. 2015. (Visited on 16/04/2017).

[29] Jon Peck. *Learning PHP CodeIgniter (2013)*. https://www.lynda.com/CodeIgniter-tutorials/Up-Running-PHP-CodeIgniter/126122-2.html. 2013. (Visited on 16/04/2017).

[30] Site Builder Report. *The Ultimate Guide to Website Builders*. https://www.sitebuilderreport.com/. 2017. (Visited on 16/04/2017).

Chapter 8

Appendix

8.1 Test Results

8.1.1 Functional testing results

Table 8.1: Functional testing results

Requirement	Test	P/F	Comments
Code created must be well formed	Each element block is entered into the W3 validator at validator.w3.org	P	Element HTML is provided by bskit so expected to be well written
User interface is simple to use	User testing indicates this is true	P	
Displays audio and video	video and audio files work as expected	P	video and audio files as part of bskit blocks
Contains e-commerce functionality	Can a product be added and sold on the website	F	This feature was not implemented fully due to time restraints
Allows copy and paste of rich text	Copy text containing different fonts, styles and colour from word to the site	P	
The site should have navigation	Each navigation item takes the user to the expected page	P	

The system should be able to create user accounts	After using the create user form, you should be able to login with the new details	P	
Instructions	Instructions are present	P	
Site is mobile friendly	Open site on mobile device (iPhone and iPad)	P	
User can add own images to the site	Upload test image	P	
Site design can be automatically generated	Using create site menu	F	The generated sites are pre-made and not automatically generated

Key: P = Pass, F = Fail

8.1.2 Non-functional Testing Results

Table 8.2: Non-functional Testing Results

Requirement	Test	P/F	Comments
Application must not store passwords as plain text	The database is checked after a user has been created	P	element HTML is provided by bskit so expected to be well written, any additional blocks would have to be re-validated
A user must not be able to access other user data	Attempts to access user sites (by url) and images	P	
The style and colour scheme of the site can be determined by analysing a logo or photo chosen by the user	video and audio files work as expected	P	video and audio files as part of bskit blocks

The site should be search-able by search engines	Does the site contain plain HTML and appropriate meta-data	F	This feature was not implemented fully due to time restraints
Works across all browsers	Site accessed from all major browsers	P	
Site is responsive	When browser page resized, application adapts to new size	P	
Each function takes a reasonable amount of time	Each function is timed	P	Every function executed within 4 seconds with the exception of the ftp upload, this is acceptable it is dependant on internet speed only
Access to admin pages is restricted	Attempt to access admin pages while logged in as non-admin	P	
Application should be error free	Test each button and feature	F	Delete image button and create sites form do not work

Key: P = Pass, F = Fail

8.2 Ethical Approvals

As this project involves surveys an ethical approval guideline checklist signed by my supervisor has been attached.

Ethical Compliance Form for UG and PGT Projects*
School of Engineering and Informatics
University of Sussex

This form should be used in conjunction with the document entitled "Research Ethics Guidance for UG and PGT Projects".

Prior to conducting your project, you and your supervisor will have discussed the ethical implications of your research. If it was determined that your proposed project would comply with **all** of the points in this form, then both you and your supervisor should complete and sign the form on page 3, and submit the signed copy with your final project report/dissertation.

If this is not the case, you should refer back to the "Research Ethics Guidance for UG and PGT Projects" document for further guidance.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical, mental and emotional harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, physical hazards or discomfort, emotional distress, use of sensory deprivation (e.g. ear plugs or blindfolds), sensitive topics (e.g. sexual activity, drug use, political behaviour, ethnicity) or those which might induce discomfort, stress or anxiety (e.g. violent video games), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.

2. The study materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and tablet computers is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   Participants cannot take part in the study without their knowledge or consent (i.e. no covert observation). Covert observation, deception or withholding information are deemed to be high risk and require ethical approval through the relevant C-REC.

---

*This checklist was originally developed by Professor Steven Brewster at the University of Glasgow, and modified by Dr Judith Good for use at the University of Sussex with his permission.

If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, the data is to be published or there are future secondary uses of the data), then it will be necessary to obtain signed consent from each participant. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script (see Appendix 1).

4. No incentives were offered to the participants.
   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle. People volunteering to participate in research may be compensated financially e.g. for reasonable travel expenses. Payments made to individuals must not be so large as to induce individuals to risk harm beyond that which they would usually undertake.

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information from participants or misleading them is unacceptable without justifiable reasons for doing so. Any projects requiring deception (for example, only telling participants of the true purpose of the study afterwards so as not to influence their behaviour) are deemed high risk and require approval from the relevant C-REC.

6. No participant was under the age of 18.
   Any studies involving children or young people are deemed to be high risk and require ethical approval through the relevant C-REC.

7. No participant had a disability or impairment that may have limited their understanding or communication or capacity to consent.
   Projects involving participants with disabilities are deemed to be high risk and require ethical approval from the relevant C-REC.

8. Neither I nor my supervisor are in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any study.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script (see Appendix 1).

10. All participants have been informed of my contact details, and the contact details of my supervisor.
    All participants must be able to contact the investigator and/or the supervisor after the investigation. They should be given contact details for both student and supervisor as part of the debriefing.

11. The evaluation was described in detail with all of the participants at the beginning of the session, and participants were fully debriefed at the end of the session. All participants were given the opportunity to ask questions at both the beginning and end of the session.

    Participants must be provided with sufficient information prior to starting the session, and in the debriefing, to enable them to understand the nature of the investigation.

12. All the data collected from the participants is stored securely, and in an anonymous form.

    All participant data (hard-copy and soft-copy) should be stored securely (i.e. locked filing cabinets for hard copy, password protected computer for electronic data), and in an anonymised form.

Project title: _____

Student's Name: _____

Student's Registration Number: _____

Student's Signature: _____

Date: _____

Supervisor's Name: _____

Supervisor's Signature: _____

Date: _____

8.3 User Guide

WebWizard

Block-based responsive site builder

User Guide

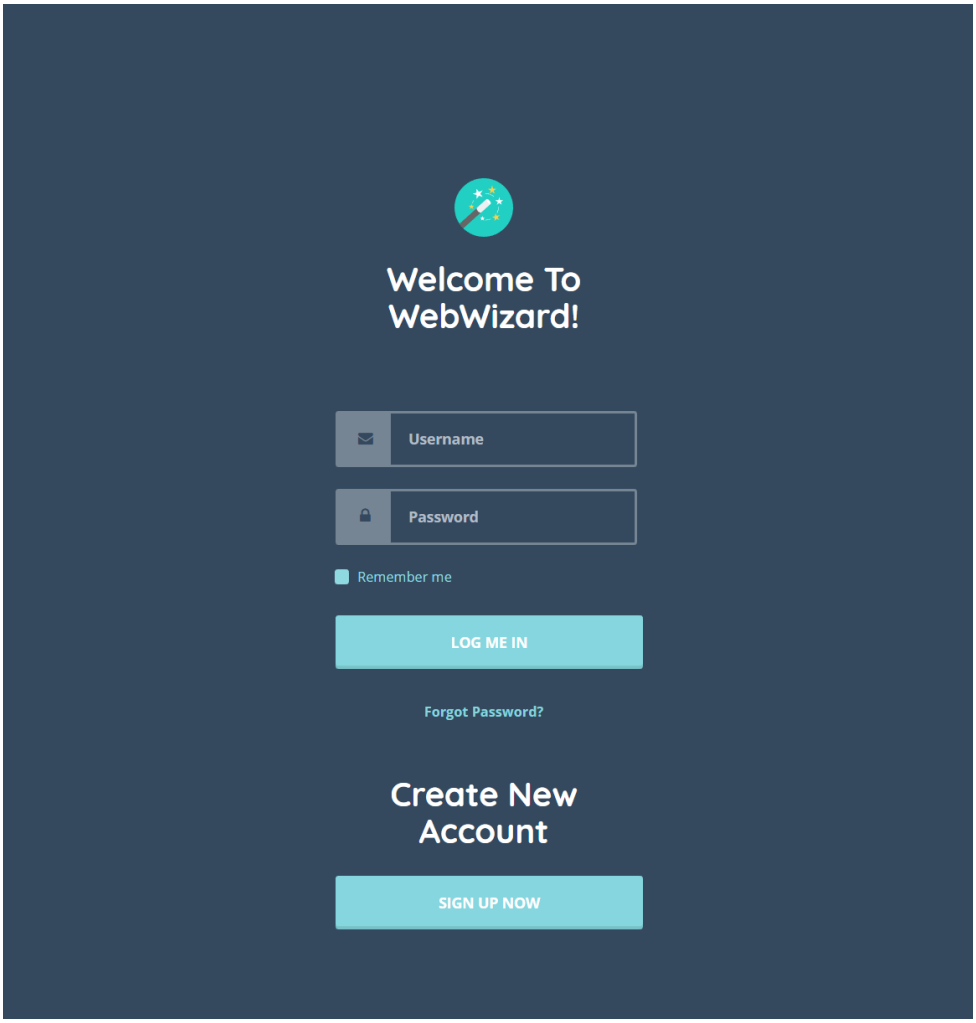
Table of Contents

Chapter 1: At a glance

Are you fed up of online site builders forcing you into proprietary platforms with buggy design tools? WebWizard offers 100% guaranteed correct, accurate and customisable code output, with very little design efforts required.

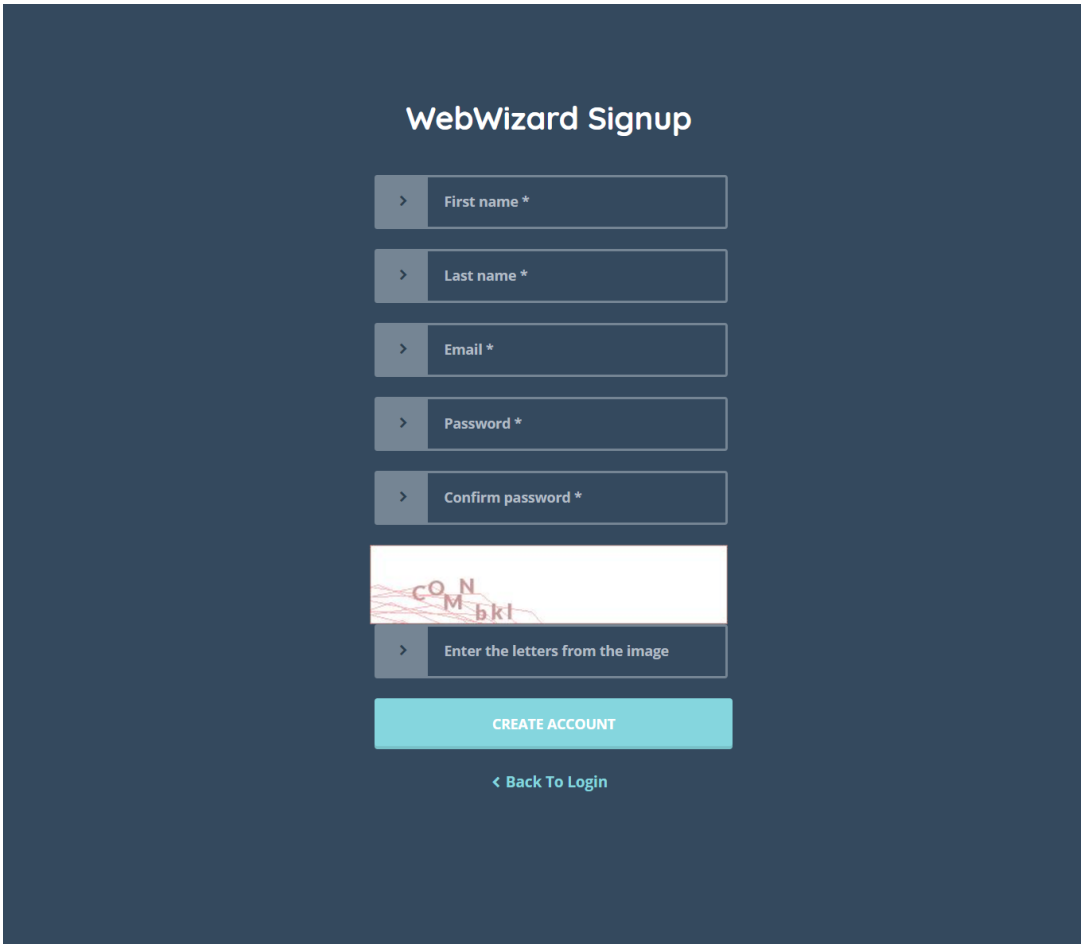
Login Page

This is the first page you will see, if you already have an account, go ahead and log in! Otherwise, create a new one by clicking the ‘sign up now’ button.



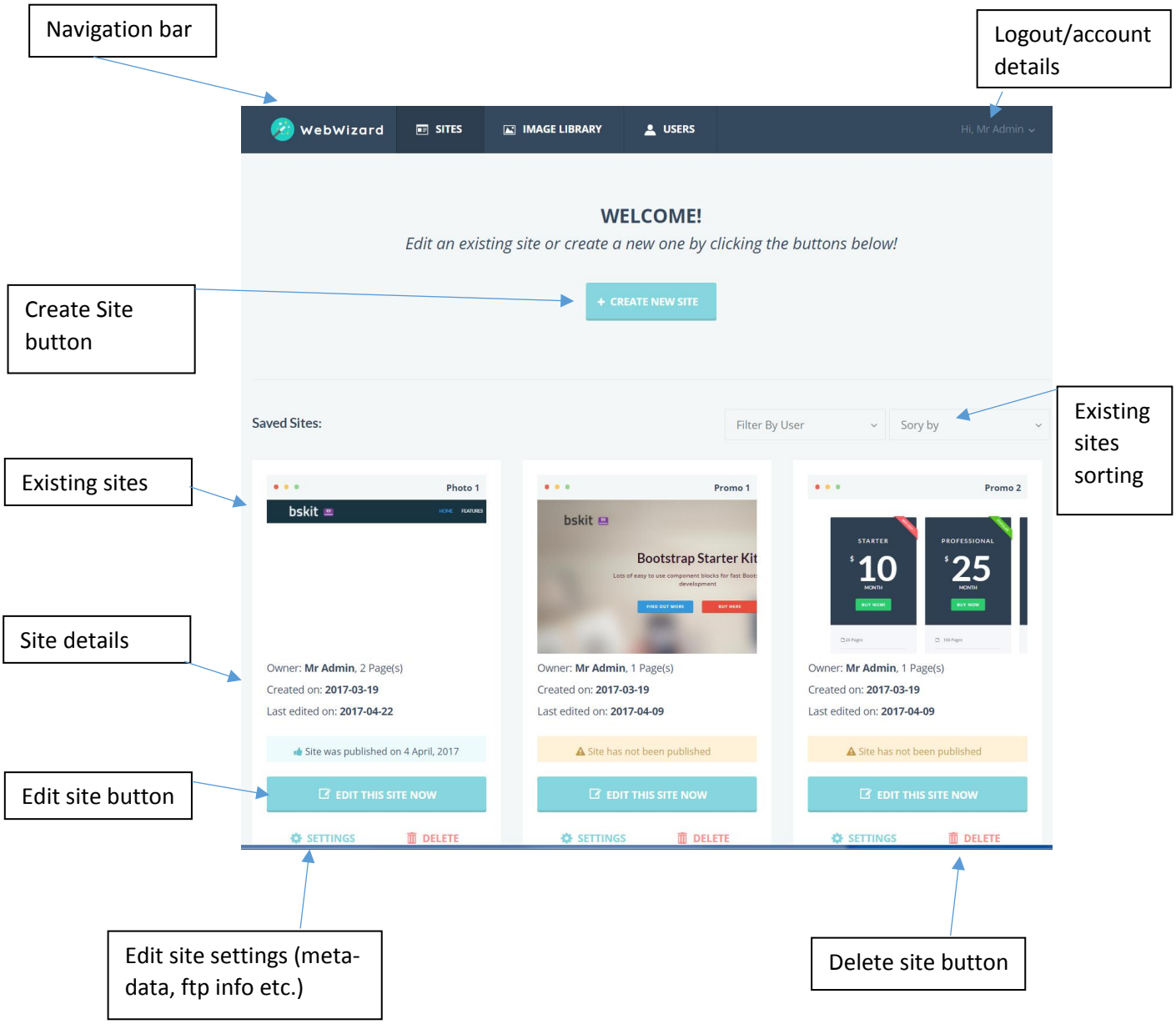
Create account page

To create an account, simply fill in the required details and click ‘create account’. You will then be redirected to the login page.



Sites Page

You will see this page after successfully logging in (note: may look slightly different if you are a new user)



Builder interface

This is where you can build your site, sites are made up of blocks, the details of which are listed below.

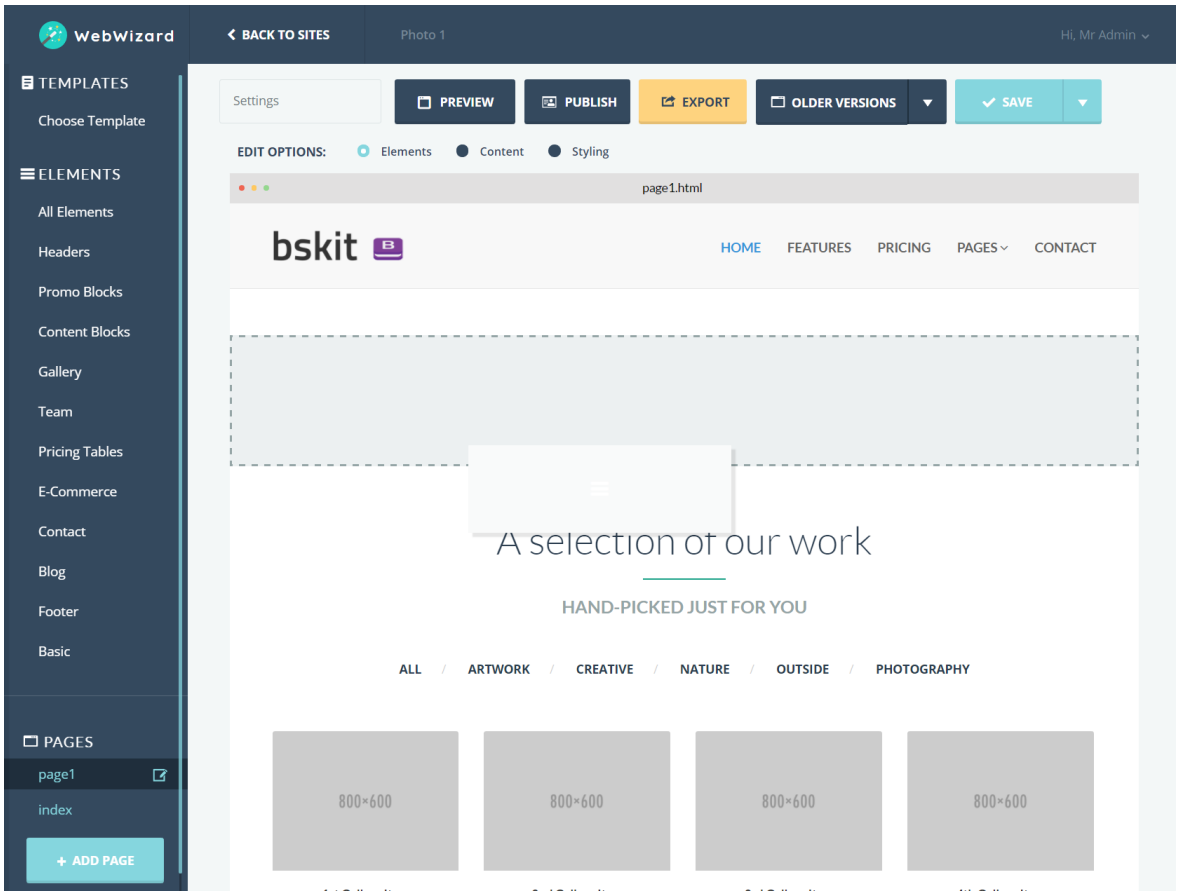
Blocks

There are many blocks to choose from, each of them sorted into a different category:

- Headers
  - Contain the logo and navigation bar for the site
- Promotion
  - Used for eye catching front pages
- Content
  - Contains blocks of text in various column layouts
- Gallery
  - Contains various photo gallery formats
- Team
  - Contains image boxes with text underneath
- Pricing Tables
  - Images with pricing tables
- E-Commerce
  - Shop-like functionality – Note – actual e-commerce functions are not yet implemented, but can be added if you have knowledge of HTML
- Contact
  - Contact forms with maps, email forms etc
- Blog
  - Blog element
- Footer
  - Footer elements with various styles
- Basic

Adding a Block

To add a block, simply click on the element type you would like (e.g. ‘Headers’), then choose the style you like and drag the thumbnail onto the canvas.



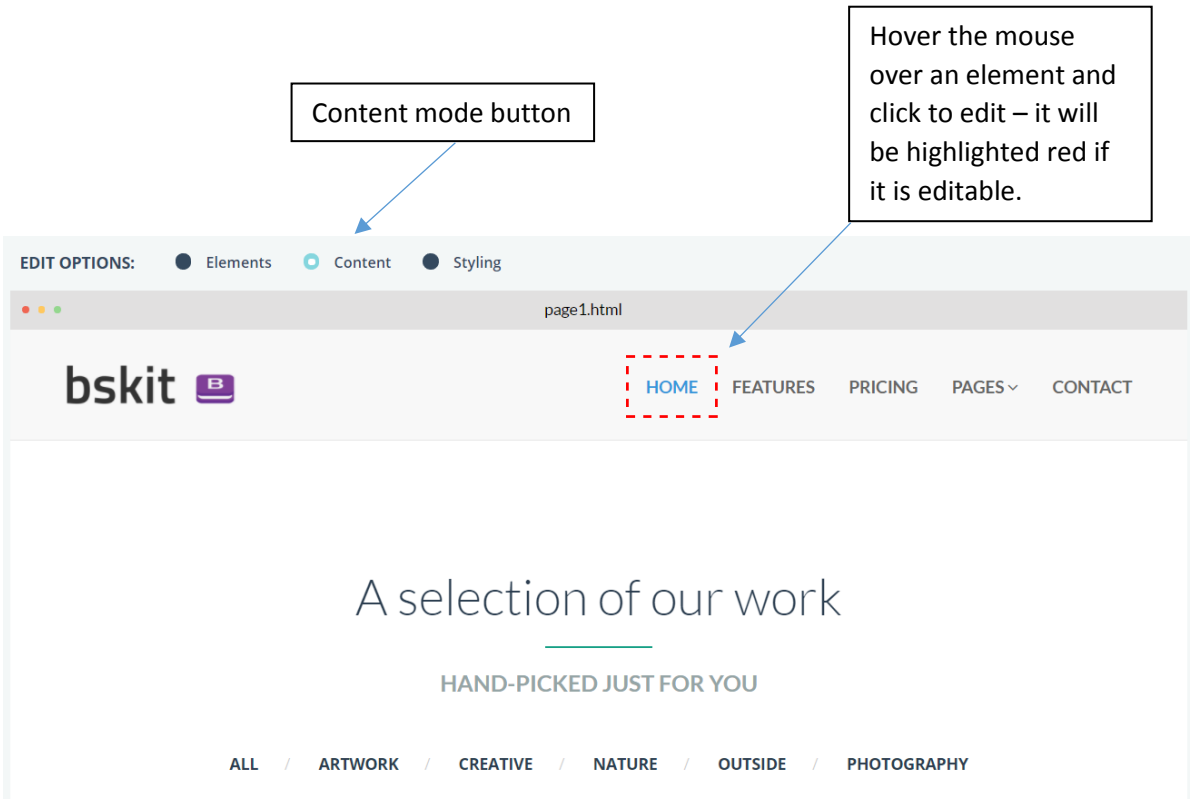
Editing a block

Blocks can be edited in various ways. They can be:

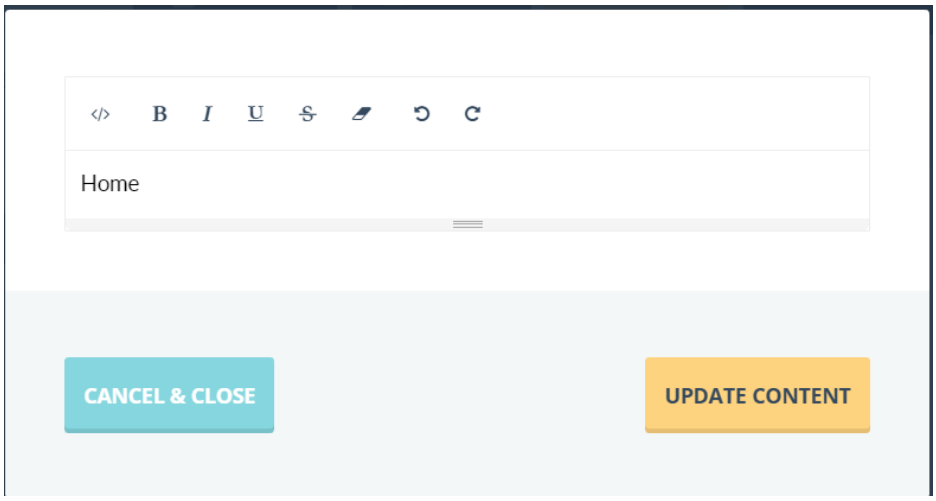
- Deleted
  - Removed from the canvas (note: still remain in the sidebar)
- Modified
  - To change the text, click the ‘content’ button and then click the text you would like to edit
  - To change the styling (colour, image, font etc.) click the ‘styling’ button and then the element you would like to change (see below for details)
  - To directly edit the source code, hover over the element and click the ‘source’ button
  - To reset the block to its default style, hover over the element and click ‘reset’
- Moved
  - To move an element, make sure you are in ‘elements’ mode and simply drag and drop the block to where you want it.

Content editing

To change the builder interface into content mode, simply click the ‘content’ button. This will allow you to edit, insert and delete text from the page.



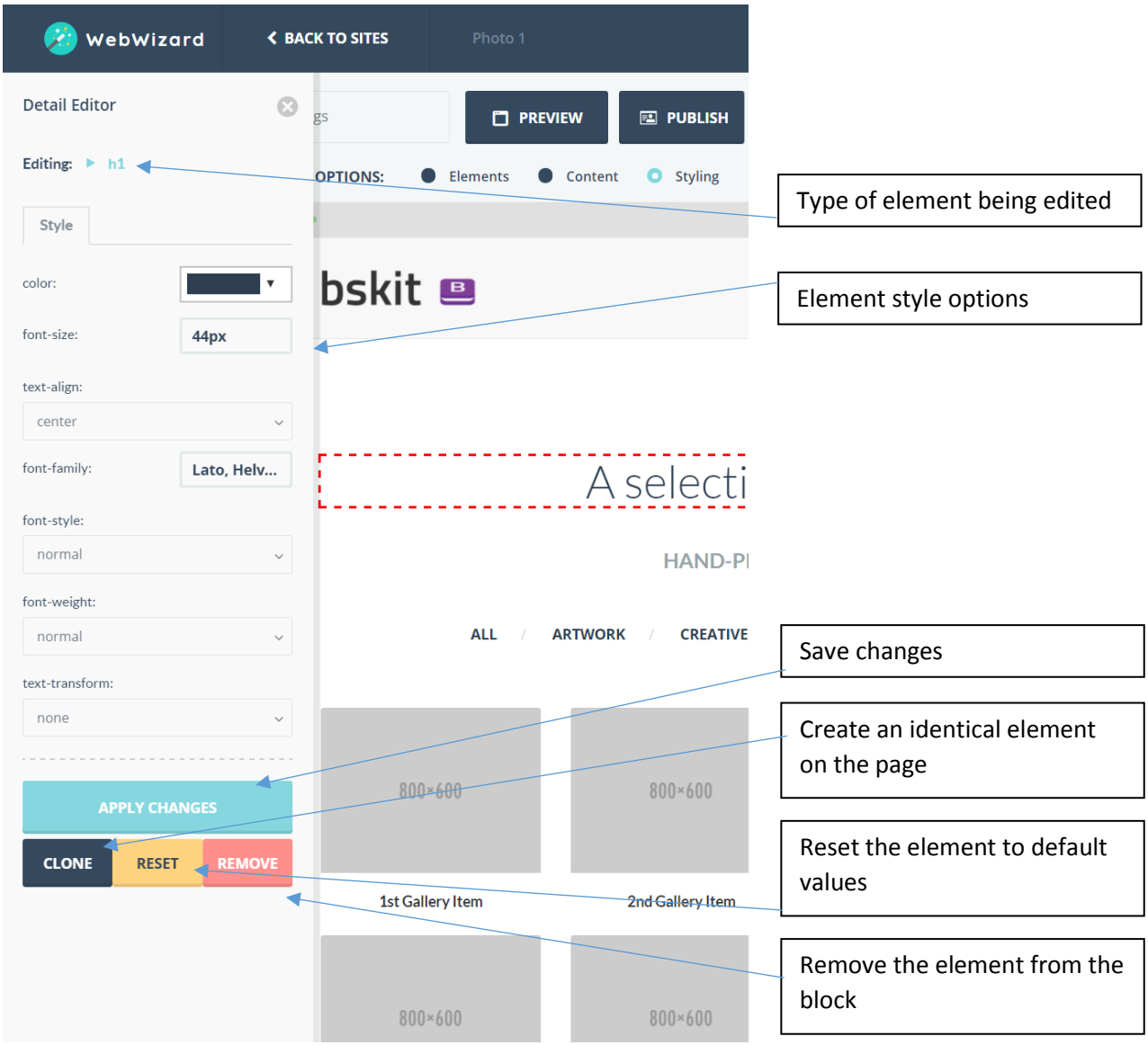
When you click an element, the text editor appears and you can enter and edit text. You can also copy and paste from a text editor like word and it will keep the font and style intact



Styling Mode

This mode allows you to edit the styling of an element, as with the content editing mode, simply select the styling button and click a highlighted element.

Text Styling



Link Styling

Links can also be edited by click one of the navigation options, you can link to a separate page on the site, a web address of your choice, or to a block on the same page.

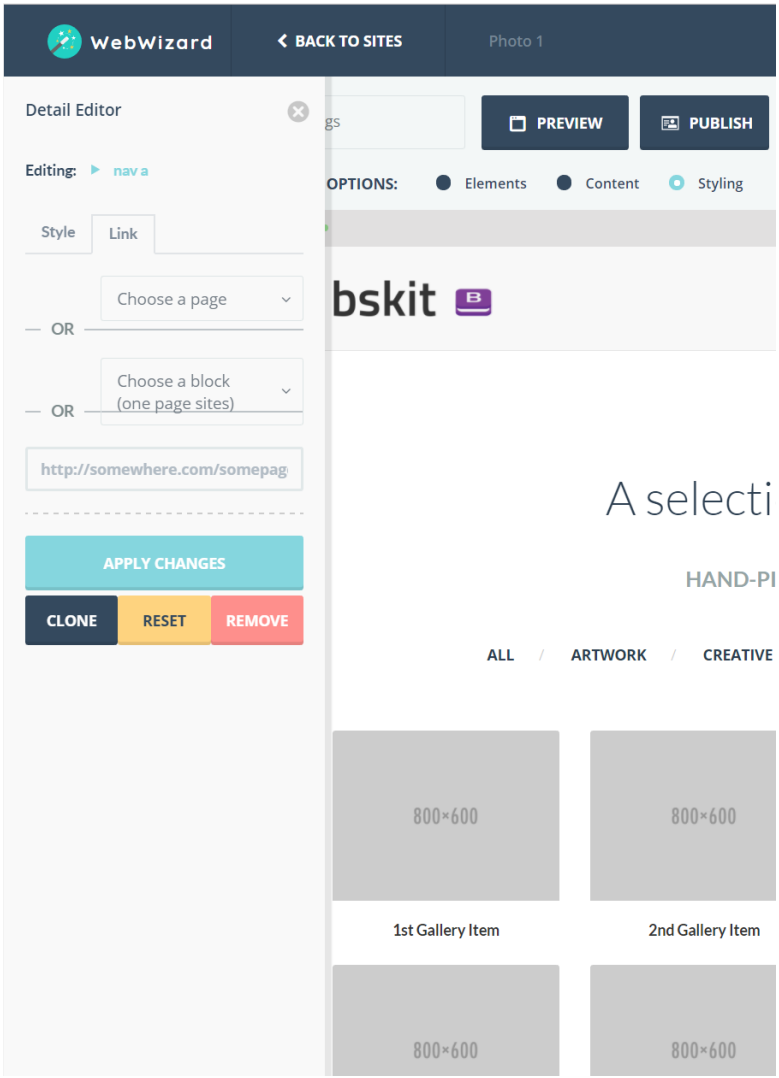


Image Styling

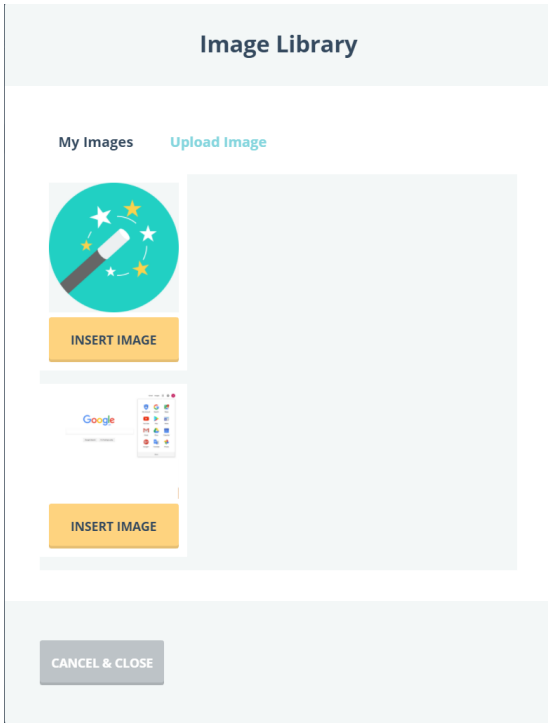
By selecting an image, you can upload or add a new image (see images section).

Images

Images can be uploaded to the application from within the builder application, or from the ‘images’ tab in the navigation bar.

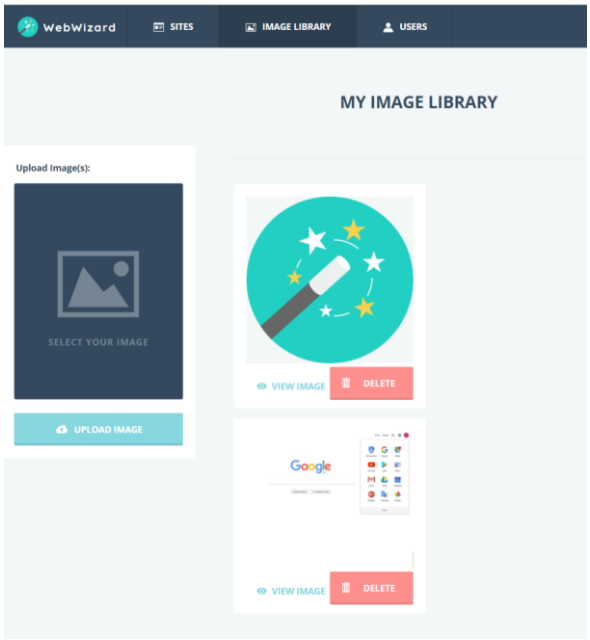
Within the builder application

Upon selecting an image in styling mode, you will see the image library modal. This allows you to upload an image from your computer or select an existing image in the image library.



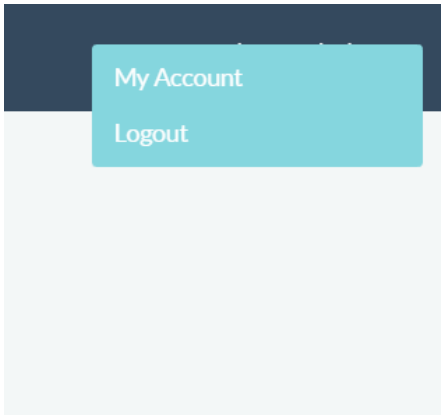
Images page

This page allows the same functionality as from within the site

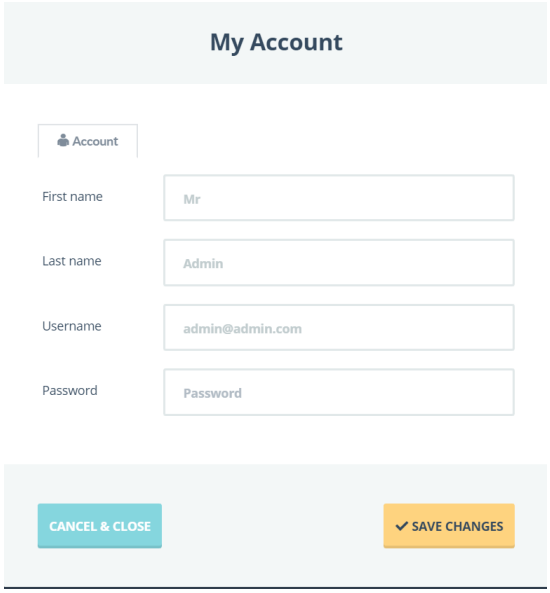


Editing account details

If you would like to change your email, password or username, click on your username in the top right of the application. You will be greeted by the following dropdown menu:

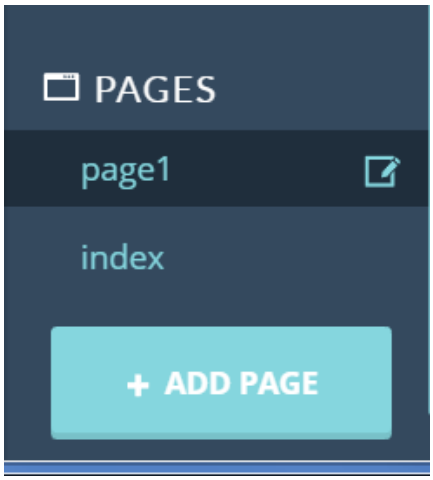


Upon clicking the My Account button, you will see the account settings page. Simply add the required information and click the update button!



Adding a Page

To add a new page, just click the Add page button, you can name the page by clicking the edit icon next to the page name.



Previewing a site

To prieview a site in a new browser tab, simply click the preview button from the main builder interface. Note: this will only preview the selected page; links to other pages will not function.

Downloading a site

If you would like to download the code for the site, click the ‘export’ button. Leave the doctype as default unless you have changed it. This will create a zip file containing all of the assets (HTML, CSS, JS, Images) for your site, you can then further edit it and upload it to a web host of your choice.

Publishing the site to a web host

Alternatively, if you are given FTP details for your web host, you can upload the site directly from the builder interface. First, click the ‘settings’ button and enter the host details in the box, make sure to ‘test’ the connection to make sure the details are correct (see figure 1 below). After the settings have saved, click the ‘publish’ button to bring up the ftp details. If this is the first time you are publishing the site, select all assets. Otherwise, only select the assets you have changed as this will make the upload process much quicker. Once published you should be able to access your site from the web!

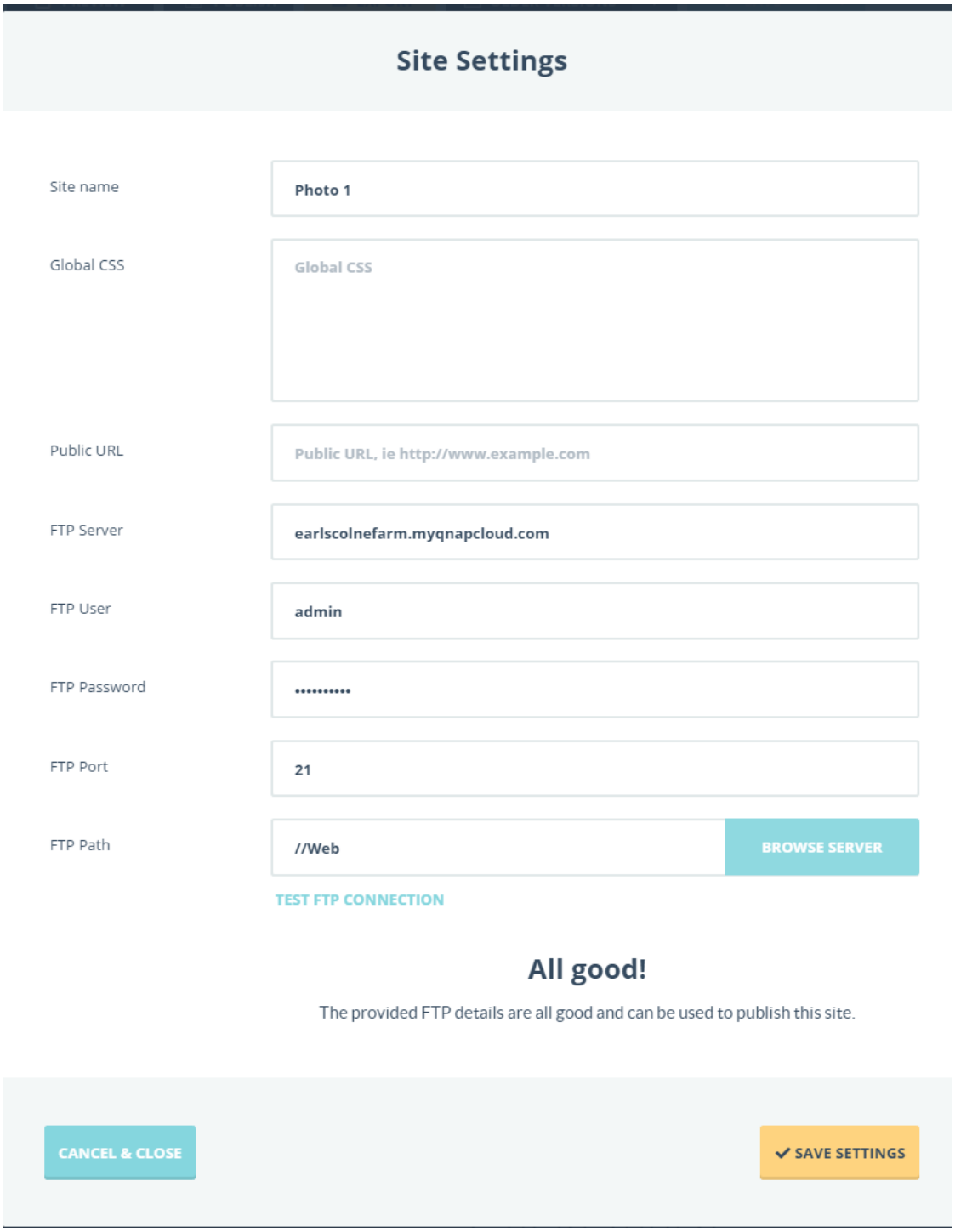


Figure 1 FTP settings

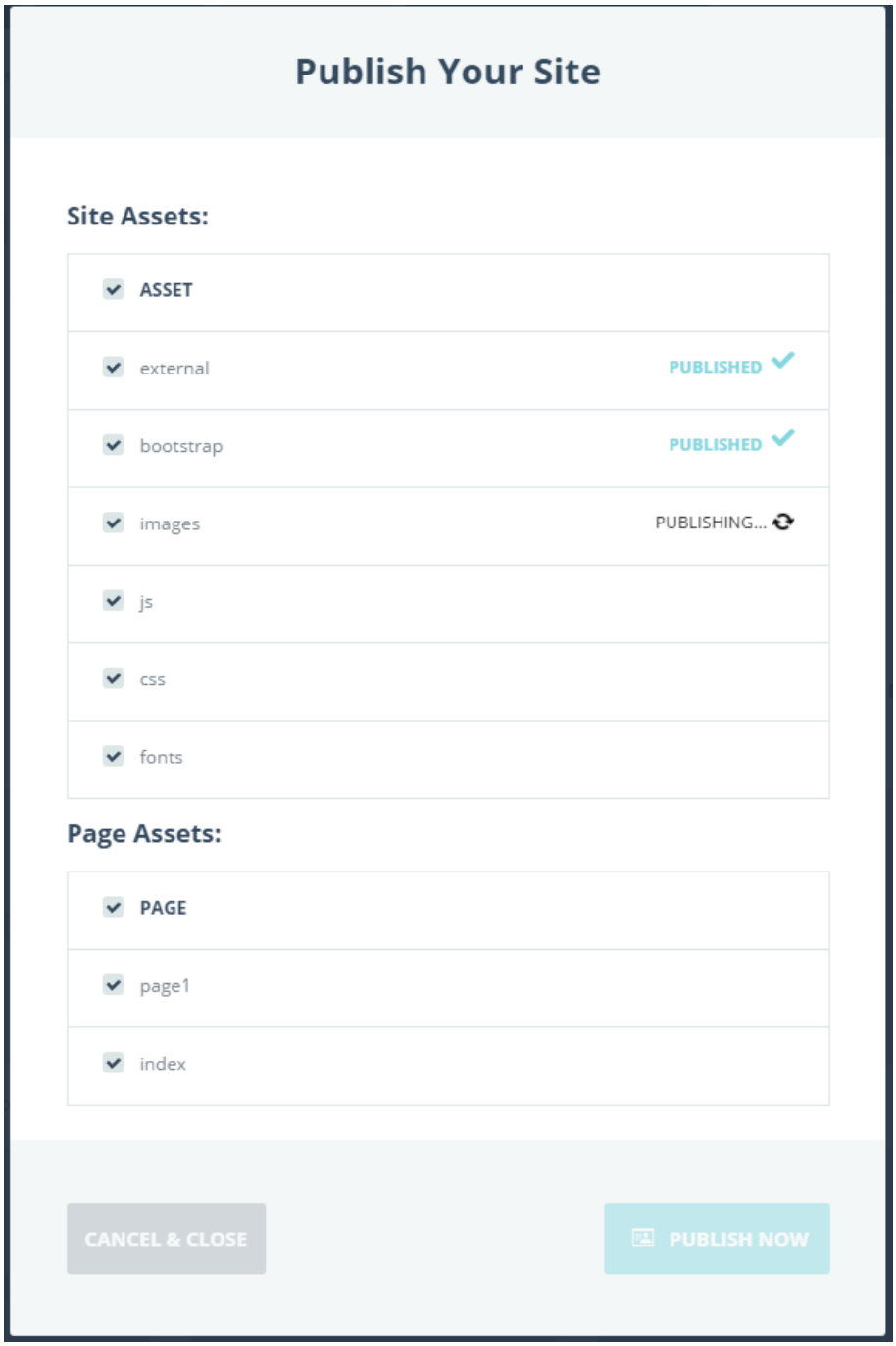


Figure 2 FTP uploading

8.4 Surveys

This section contains full results of each of the 11 surveys that were completed. All were filled out without personal data and were used to produce requirements.

8.4.1 Research Surveys

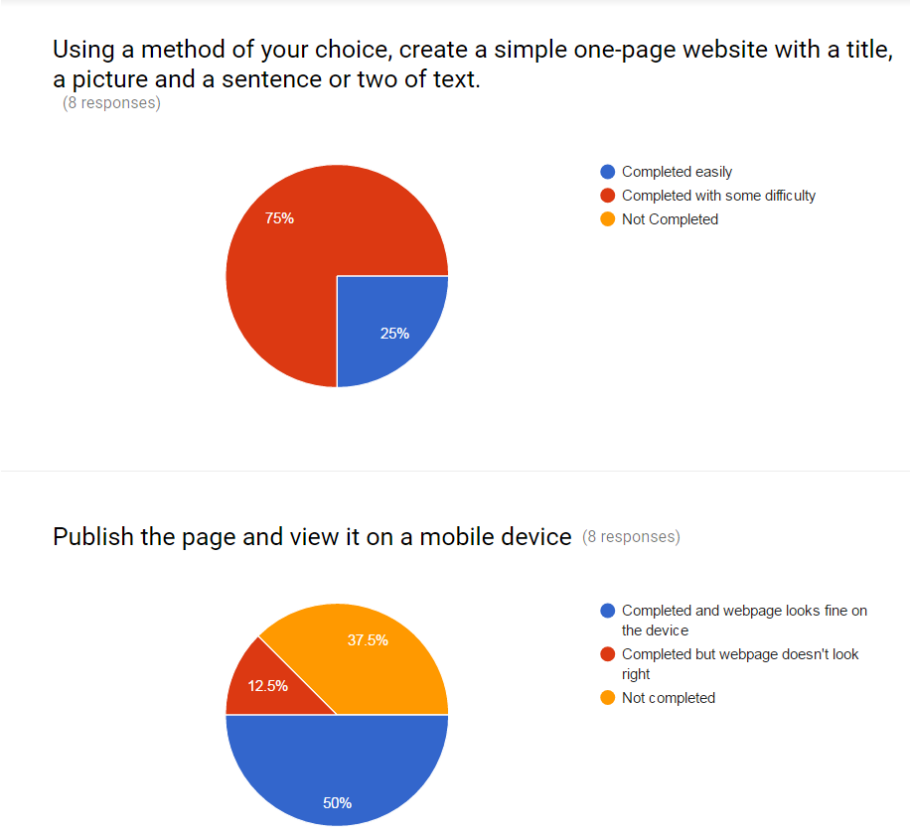


Figure 8.1: Research survey results

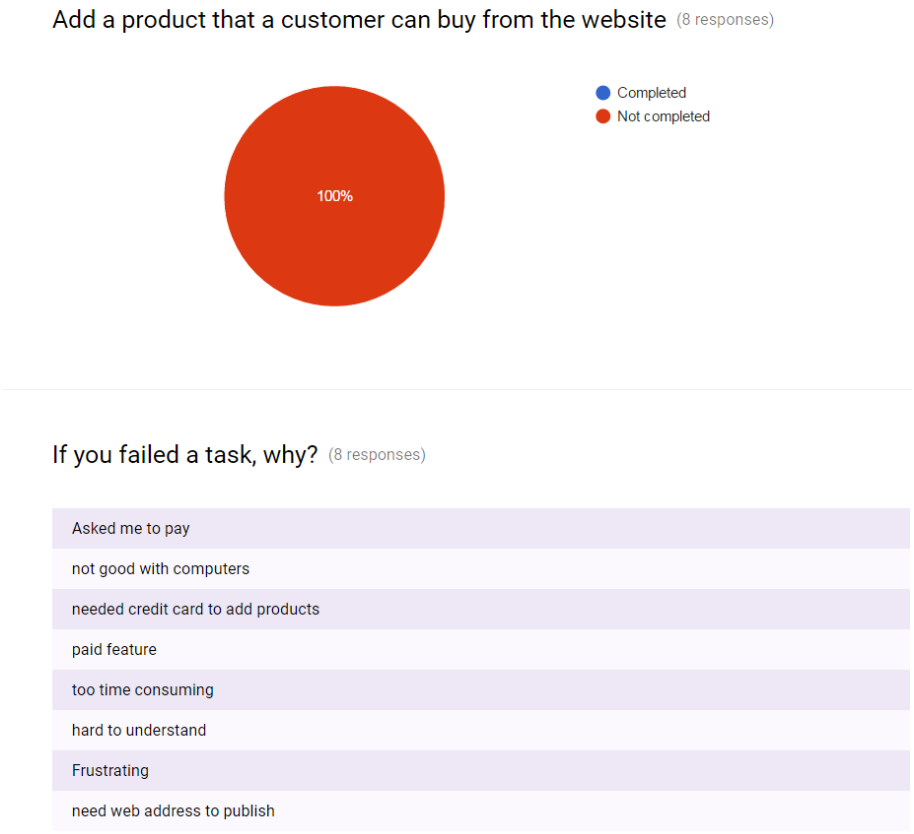


Figure 8.2: Research survey results

8.4.2 Finished Product User Survey

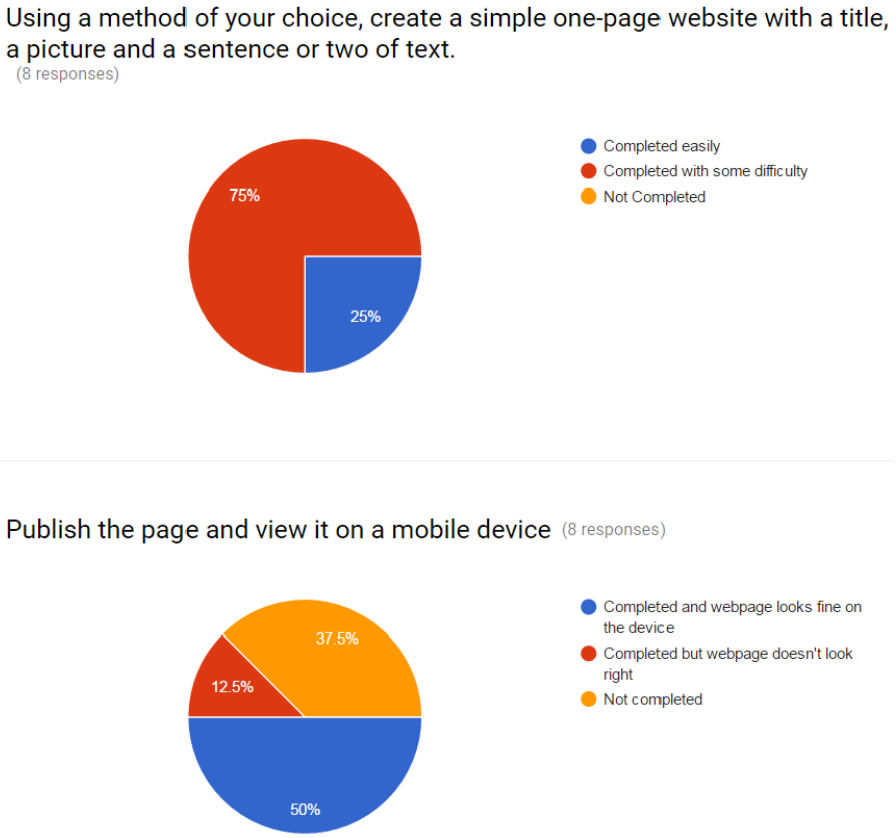


Figure 8.3: Survey Results

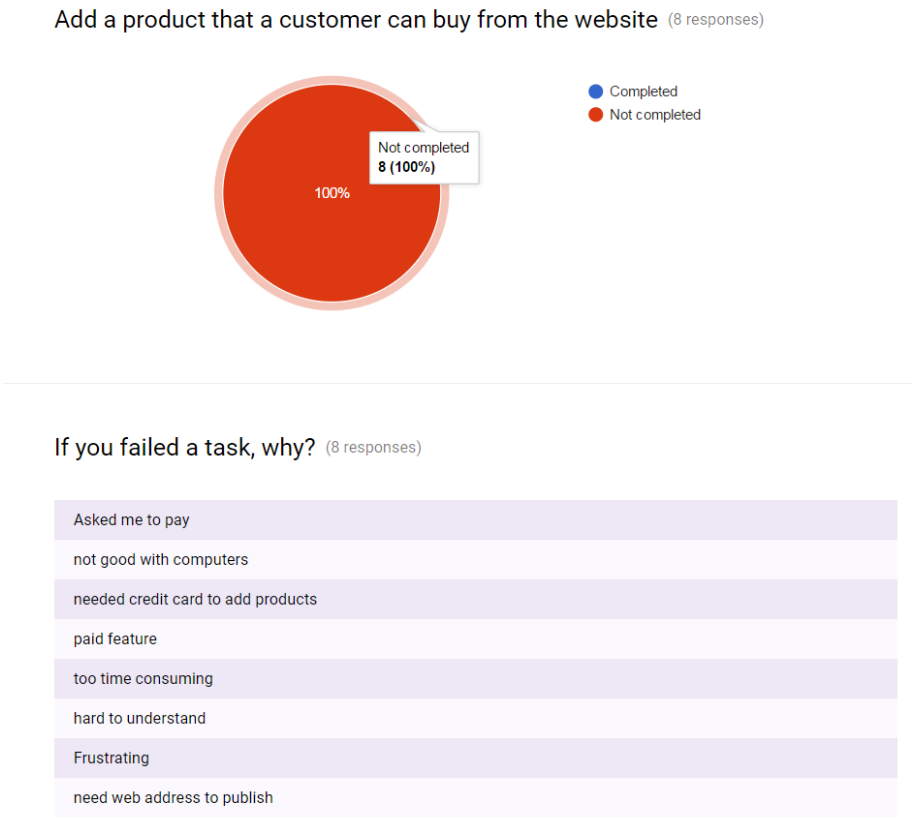


Figure 8.4: Survey Results

8.5 System Manual

System manual

This guide aims to explain the structure of the applications code and the installation process.

The included files are:

- Application – This folder contains all server side php code
- Css – This folder contains the CSS used for the application
- Elements – This folder contains the HTML, JS, CSS and images used in the HTML blocks
- Fonts – This folder contains fonts used for the application
- Images – This folder contains the images used in the application
- JS – Contains the JS used
- Less – Generated by FlatUI – do not change
- System – Generated by codeigniter – do not change
- Tmp – Generated by ion_auth – do not change
- Elements.JSON – contains references to the HTML files in /elements
- Index.php – A file like this exists in each directory and is generated by codeigniter to prevent unwanted access to the server – do not change
- Original bskit code – This folder contains the original code as copied from bootstrapstarterkit.com/bskit-demo for reference

Installation

To install this application, you will need a server running PHP 5.4 or later and a database supporting MySQLi.

First create the database by copying the UML diagram found in Section 5.2.

Next copy all files (with the exception of the 'original bskit code' folder) to your server.

After copying the files to your web director, edit the database.php and config.php files in Application->config to match your server and database settings.

The application should now be accessible.

Editing HTML blocks

Adding custom blocks is possible by adding the required files to the elements directory, updating the JSON file and finally updating skeleton.html with links to the new CSS and js used.