

Software Requirements Specification

for

Project Wixs

Version 2.0

Prepared By:

Group 02

Drayton Williams (5925342)

Jasdeep Grewal (5757828)

Cameron Hammel (5808746)

Ian LeMasters (5877667)

Curtis Honsberger (6630362)

Liam Howes (5880331)

Kieran Colaco (6054654)

Nathan Hellinga (6002620)

4F00 Project Wixs Analysis & Design

Friday, April 24th, 2020

Table of Contents

Table of Contents	2
Revision History	4
Section 1 - Introduction	6
1.1 Purpose	6
1.2 Document Conventions	6
1.3 Intended Audience and Reading Suggestions	6
1.4 Project Scope	6
1.5 References	7
1.5.1 Initial System premise	7
1.5.2 Initial CMS architecture	7
1.5.3 Password strength	7
1.5.4 Minimum software versioning	7
1.5.5 Branding Standards	7
1.5.6 Coding Standards	7
1.5.7 Accessibility Standards	7
Section 2 - Overall Description	8
2.1 Product Perspective	8
2.2 Product Features	8
2.3 User Classes and Characteristics	9
2.3.1 Classification by Skill Level	9
2.3.2 Classification by Privileges	10
2.5 Design/Implementation Constraints	11
2.6 User Documentation	11
2.7 Assumption and Dependencies	11
Section 3 - System Features	11
3.1 Persistent page features	11
3.2 Landing Page/Product Demo	12
3.3 User Registration/Login	12
3.4 User Template Selection (Dashboard)	13
3.5 Template Creation/Edit	14
3.6 Template Auto Load/Save	15
3.7 Help Section and video guides	15
3.8 Publish User Template	16
3.9 Delete User Template	16
3.10 View Live Template	16
3.11 Load site statistics (Admin)	17

3.12 User Content Management	18
Section 4 - Functional Requirements	18
4.1 Connect to Postgres Database	18
4.2 Validate/Sanitize Inputs	19
4.3 Create User	20
4.4 Retrieve (Get) User	20
4.5 Update User	21
4.6 Delete User	21
4.7 Logout User	22
4.8 Check if User is Admin	22
4.9 List Users	23
4.10 Create Template	24
4.11 Retrieve (Get) Template	24
4.12 Update Template	25
4.13 Delete Template	26
4.14 List User Templates	26
4.15 Get User Active Template	27
4.16 Upload (Create) Content	27
4.17 Retrieve (Get) Content	28
4.18 Delete Content	28
4.19 List User Content	29
4.20 HTTP AJAX Data Transfer	30
Section 5 - External Interface Requirements	30
5.1 User Interfaces	30
5.2 Hardware Interfaces	31
5.3 Software Interfaces	31
5.4 Communication Interfaces	31
Section 6 - Other Nonfunctional Requirements	32
6.1 Performance Requirements	32
6.2 Safety Requirements	32
6.3 Security Requirements	32
6.4 Software Quality Attributes	33
6.4.1 Maintainability	33
6.4.2 Availability	33
6.4.3 Reliability	33
6.4.4 Portability	33
Section 7 - Change Management Process	34

7.1 Discord	34
7.2 GitHub	34
7.3 Google Drive	34
Section 8 - Legal, Copyright, and Other Notices	34
8.1 MIT License	35
Section 9 - Other Requirements	36
9.1 Style Guide	36
Appendix A - Glossary	37
Acronyms/Abbreviations	37
Definitions	38
Appendix B - Analysis Models	40
System Architecture	40
ER Model Diagram	41
Use Case Diagrams	42
Concept Page Layouts	44
Sequence Diagrams	48
Appendix C - Issues List	50

Revision History

Name	Date	Reason For Changes	Version
Added Missing Group Member	07/02/2020	SRS initially submitted missing one group member	1.1
Introduced accurate system constraints	13/03/2020	After implementing commenced, accurate system constraints were able to be produced	1.2
Bolstering of section detail	24/03/2020	Team went back to enhance the details of existing sections following professors critique	1.3
Adding of new section 'Functional Requirements'	16/04/2020	Based on feedback from a prior stage, more thorough description of low-level system processes was deemed necessary	1.4
Adding of new	18/04/2020	Adding extra detail to explain how	1.5

section 'Change Management Process'		the team handles changes to be made in the system	
Adding of new section 'Legal, Copyright, and Other Notices' under Section 4	19/04/2020	Adding license info for anyone who may consider using this system on their own.	1.6
Final document revisions	23/04/2020	Final passthrough to check for spelling/grammar errors and integrity of information	2.0

Section 1 - Introduction

1.1 Purpose

Project Wixs is a Content Management System allowing for a seamless creation of single-page web pages featuring an easy-to-use drag and drop component interface. Project Wixs aims to provide an intuitive experience for a wide range of users; from complete beginners unfamiliar with any form of web design or HTML, to veteran designers with multiple years of experience. The system will be able to have up to fifteen user accounts that can create and edit web pages with a range of content typical of any modern day website (such as text and images). Users will be able to create, edit, and delete up to four templates and publish one at any time for live viewing. The system will also allow for the uploading of user photos to be used within the creation of their templates

1.2 Document Conventions

All document in-text citations are foot-noted in an MLA format.

1.3 Intended Audience and Reading Suggestions

This document is intended for use by developers, project managers, testers, and users with sufficient knowledge in a technical background, as all may benefit from different snippets of information provided. Sections 2 and 3 (Overall Description & System Features) provide a detailed overview of system design and features that may benefit users and testers looking to obtain a more thorough understanding of the application than that provided on the Project Wixs Help/FAQ web page. Sections 5 and 6 (External Interface Requirements & Other Nonfunctional Requirements) provide an in-depth look at internal communications present within the system architecture and the environment necessary for sufficient operation of Project Wixs. Developers, testers, and project managers will benefit from the aforementioned section information. Diagrams and models contained in Section 6 provide an abstracted description of different aspects of the system for those who seek further information.

1.4 Project Scope

Project Wixs is intended to be used by those with access to the internet who wish to create web pages without the prerequisite of understanding how to code in common web technologies such as HTML, CSS, or JavaScript. The system caters to users ranging from little-to-no experience with web development, all the way to seasoned experts who are looking for a simple UI for quickly establishing a personal web presence. System access will be restricted through a basic system security, which will be implemented through the use of a user login system. Users will be able to upload photo or video content. Overall, this system is meant to make the task of creating and publishing a web page as efficient of an experience as possible, through the use of a drag-and-drop template design system.

1.5 References

1.5.1 Initial System premise

Initial system premise (and contract specification) based on project statement as provided by the COSC 4F00 course professor Dave Bockus.

Found Here:

<https://github.com/DrayWilliams1/Project Wixs/blob/master/Project%20Statement.pdf>

1.5.2 Initial CMS architecture

Initial CMS architecture page concepts, and feature set inspired by other popular competitors such as WordPress, Wix and Bolt CMS.

Found Here:

<https://wordpress.com/> (WordPress website)

<https://www.wix.com/> (Wix website)

<https://bolt.cm/> (Bolt CMS website)

1.5.3 Password strength

Criteria for password creation based on Lafayette College's standards for a strong password.

Found Here:

<https://its.lafayette.edu/policies/strongpasswords/>

1.5.4 Minimum software versioning

Information on minimum software versioning referenced in section 4.3 is obtained from a PHP info file hosted on a shared sandcastle group account.

Found Here:

<http://cosc.brocku.ca/~c4f00g02/projectWixs/phpinfo.php>

1.5.5 Branding Standards

Front-end system design will adhere to a project-specific style guide regarding colour schemes, spacing, and typography. Obtained from style guide in section 6.1

1.5.6 Coding Standards

System will adhere to programming standards agreed upon by the W3C.

Found Here:

<https://www.w3.org/standards/>

1.5.7 Accessibility Standards

System will adhere to accessibility standards as agreed upon by the W3C.

Found Here:

<https://www.w3.org/standards/webdesign/accessibility>

Section 2 - Overall Description

This section provides a high-level overview of the system. Mainly detailing what Project Wix offers and who the system is intended for.

2.1 Product Perspective

To help understand why Content Management Systems (CMS) like Project Wixs are beneficial to organizations, it is important to understand what CMS' are and what they do. In a study discussing CMS' and the organization's that adapt them, it was noted that,

Some form of content management (CM) process or system is becoming essential for all organisations with a significant Web presence as the amount of digital content continues to proliferate... Software product companies have moved to address this need and call their offerings content management systems. Such systems help a business to set-up and organise their Website(s), so that the Web sites can grow and change rapidly while maintaining high quality¹

CMS' such as Project Wixs offer an easy way for companies to develop their online presence in an increasingly digital world. The exact implementation is flexible - whether an organization needs a public site to help build their web presence, or a more interorganizational site to help share data between employees - CMS' provides the tools to make these a reality. Project Wixs delivers this, while also allowing for the creation of clean, efficient websites with no prior knowledge of HTML or other web design experience.

Project Wixs is a new CMS, conceived to serve as a competitor to existing web development apps such as Wix and Wordpress. Being a new product, Project Wixs does not aim to succeed or carry over content from any existing CMS, but serve as a new, better replacement to any organization's current system. Being web-based, the only requirement for an organization to set it up would be a server with capabilities similar to Sandcastle, and Internet-connected devices for users to access the site from (See Figure B-1: Basic System Architecture).

2.2 Product Features

As a Content Management System, Project Wixs will provide the adapting organization with all the tools necessary to begin creating sites and content to serve organizational goals. This begins with providing secure, controlled access to the site editing capabilities, which Project Wixs provides through standard account management. When first setup, an initial administrator account will be available that can manage the system's user base, and choose other accounts with which to share administrative privileges. Other users can then apply to register by going

¹ "Understanding Web content management systems: evolution" 29 Apr. 2018, https://www.researchgate.net/publication/220672404_Understanding_Web_content_management_systems_evolution_lifecycle_and_market. Accessed 4 Feb. 2020.

through the main Project Wixs home page and, upon account creation, can then begin editing site templates.

All non-administrative accounts are regular user accounts, with each account being able to edit and create four user templates as well as host a private collection of web media content (images) that can be shared on the site. To customize these sites, Project Wixs features a drag-and-drop editor that will allow users to create web templates with little to no knowledge of HTML. This will include standard HTML components such as headers, text, links, etc. as well as the aforementioned media content that can all be organized and arranged in a visual interface. For more experienced users, an option to switch to a plaintext HTML editor will be included as well.

Finally, Project Wixs will feature a Help section that will help guide new users on how to use the Project Wixs editor. When a new user first logs in, there will be a recommendation for the user to visit the help page. Access to the Help section will also be available through a link on the footer of all pages of the Project Wixs site. This Help section will contain text instructions, helpful links, and embedded video tutorials to help the user familiarize themselves with Project Wixs' capabilities.

2.3 User Classes and Characteristics

There are two specific ways of classifying user's we wish to address. The first is by the user's skill level, which is an important element to consider for what is the CMS' most central feature; its website editor. The other is a more general look across the entire site's userbase, comparing and contrasting their access and privileges.

2.3.1 Classification by Skill Level

The Project Wixs site editor has options for users of all skill levels. However, certain features of Project Wixs may or may not be relevant to the user's skill's level, which we have classified below. Overall, it *will* favour Beginner and Intermediate users with limited to no HTML experience (which is common for those outside the fields of Computer Science). The drag-and-drop functionality lends itself better to these low-level users, rather than high-level users who want greater freedom of customization.

Beginner

Beginner users are those with little to no experience with HTML or web design. Overall they are likely not very technologically literate in general, and may be intimidated by any form of plain text forms of code. Project Wixs seeks to appeal to them by allowing them not to have to touch HTML code at all, and to work exclusively within the visual drag-and-drop editor. In addition, the help pages are easily accessible and provide detailed, beginner-friendly tutorials in both video and text format.

Intermediate

Intermediate users would be those with under 5 years of experience with HTML and/or web design. They may feel confident using HTML, but could appreciate the convenience that the drag-and-drop editor provides and use a mix of both.

Expert

Expert users have over 5 years of experience with HTML/web design, and may not want to even bother with learning the drag-and-drop editor. They will likely be put off with the extra features or restraints put upon the editor, and would want full customization of the webpage. This should only require one click to move to editing plaintext HTML. This should be an easy enough switch to do, and once they are in the plaintext editor they will not need to bother with any of the extra editor features.

2.3.2 Classification by Privileges

Visitor

Any user to the site with no account registration. They can apply to become a user from the Project Wixs homepage. Otherwise they are able to view any site they know the address to. These users are more interested in the end-user site's than they are with the CMS, and are of low priority.

Editor

The editor class encompasses all the classes by skill level described in the previous classification. Editors are users that have access to edit their own webpage and media content library. These users have a wide range of needs, and the previous classification goes into detail on how each class is serviced.

Administrator

The admin class shares all the same functionality of editors, but with the addition of account management. This encompasses an additional page listing all the accounts and site statistics. While the page is still organized and clearly mapped, we expect the admin to be more comfortable with back-end elements and be able to deal with the large amount of information presented here.

2.4 Operating Environment

The equipment available within the Department of Computer Science at Brock University will be used to host the Project Wixs system, and as such, will operate on the x86_64 Red Hat Linux operating system. PostgreSQL is a powerful, open source object-relational database system and will be used as the database system for Project Wixs. The powerful server scripting language PHP (Hypertext Preprocessor) will be used to create user's dynamic webpage content and work hand-in-hand with PostgreSQL by actively modifying files stored in its database. For all front-end aspects, React.js will be used. React.js excels specifically in building user interfaces for single page applications, which is precisely the purpose of this project, and is therefore an ideal choice.

2.5 Design/Implementation Constraints

In order for user's webpages to load efficiently, Project Wixs will have a maximum allowable photo size of 10 MB. In most cases, this file size assumes responsibility of the designer to use a combination of reducing file size, reducing resolution in dpi, and image compression on their image before attempting to upload. Project Wixs only assumes responsibility for failure to load MP4 and WebM video file formats, which are both compatible with the most common browsers: Chrome and Firefox. These limits on photo file sizes ensure that web pages run and load effectively as this is a common problem for beginner designers with little knowledge about image optimization for the web. This also ensures that users do not try to upload incredibly large file sizes in an attempt to disrupt Project Wixs' services.

2.6 User Documentation

Users will be offered an array of tutorials and helpful documentation, including this Software Requirements Specification document. In addition to this document, short YouTube tutorial videos will be created and linked with topics including, but not limited to: user account creation and login, template creation, and editing user templates. Text-based tutorials and guides will also be provided. On user account creation, users will be prompted to click on a pop-up dialogue box that will link to the Help page which contains all the help resources. This dialogue box may be closed and will not popup again, but the resources may be accessed again at any time through the Help button, which stays persistent in the footer throughout the whole site.

2.7 Assumption and Dependencies

Project Wixs assumes that all users are not visually impaired or impaired in any other form that would reasonably render our content management system unusable or extremely difficult to navigate. Our system will currently not implement any features for the use by those visually, or similarly impaired individuals such as auditory prompts. These individuals are of the extreme minority of the targeted user base and implementing such features would require considerable time and resources currently unavailable within the project time and resource constraints.

Section 3 - System Features

This section lists and gives an in depth description of the macro features/processes of the Project Wixs system. Within each listed feature can be found a description and grading of priority, expected stimulus and responses when engaging the feature, and the functional requirements the feature is dependent upon for successful execution.

3.1 Persistent page features

3.1.1 - Description and Priority

- Working left to right, every page consists of a primary header spanning the top of the page. This header persists throughout every webpage and contains buttons

to transverse to the home page and dashboard; with an additional button to login or logout respectively (depending if the user is currently logged in or not). This is demonstrated in (Figure B-6).

- In addition, every page also displays a footer at the bottom also demonstrated in (Figure B-6). This footer contains copyright information followed by a help button which brings the user to the help page.
- **Priority - Medium:** As the buttons included are not critical for system performance, but simply help the user traverse through the site.

3.1.2 - Stimulus/Response Sequences

- Button presses would take the user to their desired location. As the footer and header is static throughout every page, these buttons can be pressed on every page.

3.1.3 - Functional Requirements

- N/A

3.2 Landing Page/Product Demo

3.2.1 - Description and Priority

- Upon arrival to site, the user is met with the landing page. The landing page presents our logo front and center, below the header. Below the logo, visitors can find 3 examples of features Figure B-6. The page provides basic high-level information about key system features and provides some interaction for those wishing to visit the website without fully engaging with the application
- **Priority - Medium:** Presentation is key and this page provides a quick way of displaying the system in a clean manner without much work required on the users end. Not vitally necessary to system function, however, we see it as important for a complete overall system.

3.2.2 - Stimulus/Response Sequences

- Visiting the website URL (to be decided)

3.2.3 - Functional Requirements

- N/A

3.3 User Registration/Login

3.3.1 - Description and Priority

- Navigating to the login page by means of the login button on the top right of the header presents the user with a screen to login to their account (Figure B-8). This screen contains two fields for input; an email/username followed by a password field. Ideally, there would also be a button underneath to register for a new account.

- The register page (Figure B-8) looks similar to the login page but would have two additional fields. This time, there would be independent fields for username and email. In addition, there would also be an input field for the user's password and another input field to ensure the previously written password is typed the same.
- **Priority - High:** As the majority of system features rely on a user interacting with their own individual templates, it's essential that the system recognizes users individually and serves them the right content.

3.3.2 - Stimulus/Response Sequences

- In terms of the login page, after the login button is pressed and a user is verified, they are brought to the user's unique template selection screen (Figure B-5).
- Similarly, with the register page, the register button would confirm the user input, create the account, and redirect to the template selection screen (Figure B-5).

3.3.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.2 - Validate/Sanitize Inputs:** Inputs obtained from the login and registration forms are trimmed of whitespace, checked for null values, and sanitized of any malicious code found within the strings.
- **4.3 - Create User:** After registration, input data is sent to the back-end so a user can be created using the specified details
- **4.4 - Retrieve (Get) User:** During login, the user is obtained (if the email matches) so that the password can be verified.

3.4 User Template Selection (Dashboard)

3.4.1 - Description and Priority

- After a user is verified through the login page they are redirected to their own template selection screen, otherwise known as the dashboard. (Figure B-5). This screen greets the user by name and displays their customizable templates. Each template block provides a field to edit the template name followed by three buttons below to edit said template, delete said template, or set the template as active.
- Active templates will be highlighted such that the user is able to visually recognize that the template is active. Any active templates will have a green border, along with a subtitle " (Active) " beside the template name in a muted font style. Active templates will not have the "Set Active" button.
- An admin button is included in the top right of the screen below the header if the current logged in user is an admin. This button brings the admin to the Admin tools page (Figure B-4).
- **Priority - High:** As the main divider between users and their templates, this screen is important as it's the gateway to the user's content.

3.4.2 - Stimulus/Response Sequences

- Each edit button would bring the user to the template editor page for said template (Figure B-7).
- Delete buttons would pull up a prompt which would require further input to delete said template.

3.4.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.4 - Retrieve (Get) User:** To retrieve the user whose templates must be loaded.
- **4.11 - Retrieve (Get) Template:** When an individual template is selected, its unique identifying value must be sent in the URL to the editor.
- **4.14 - List User Templates:** All templates associated with the user are listed.

3.5 Template Creation/Edit

3.5.1 - Description and Priority

- Once a user selects a template to edit from the template selection screen (Figure B-5), they are brought to the template editor page for said template (Figure B-7). On this page, the current template is shown as a preview, along with a component library on the right which allows the user to drag and drop components into the preview window.
- **Priority - High:** Arguably the highest priority in terms of utility, this screen is important as it's the main feature for users.

3.5.2 - Stimulus/Response Sequences

- Dragging and dropping a component from the item pallet on the right to the preview on the left would place said component where the user dropped it.
- The preview will confirm the layout and media chosen upon placement within that area.
- Once the user is content with their preview they are able to publish their site by means of the publish button on the left side of the detail editor header.

3.5.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.10 - Create Template:** Required in order to facilitate the creation of a template within the database.
- **4.12 - Update Template:** Performed upon alteration of template's grid/data layout
- **4.19 - List User Content:** In order to list user content within the media component modification area.

3.6 Template Auto Load/Save

3.6.1 - Description and Priority

- As the user edits and changes their template the template should be automatically saved to the database. This ensures that they do not accidentally make changes and forget to save them, or lose work in the event of a computer crash etc.
- **Priority - Medium:** This is very important for situations where not having autosave could cause the user to lose work, however, for most users they will not notice it happening in the background.

3.6.2 - Stimulus/Response Sequences

- No direct user interaction with this system. Whenever a user makes a change to their template the data should be automatically saved to the database under their account. This will trigger on layout updates, data/ content updates and finally on style updates. Whenever the autosave is running in the background a discrete spinner will be displayed in the upper right corner of the website to indicate the activity.

3.6.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.2 - Validate/Sanitize Inputs:** To validate template grid engine data and strip of any malicious or null-valued code
- **4.11 - Retrieve (Get) Template:** Upon template automatic loading, the template to be retrieved must be specified first.

3.7 Help Section and video guides

3.7.1 - Description and Priority

- To help with user interaction, the help page (Figure B-4) acts as a tool for those who require help with formatting and creation of their site. Ideally, a tutorial video will be embedded onto the page providing a visual guide on features. In addition, a text guide will be available beside the video, along with additional helpful links below the video.
- **Priority - Low:** Help pages serve to assist. Although it would be nice to have, it's not a critical component for user use, it serves as an aid for those in trouble.

3.7.2 - Stimulus/Response Sequences

- The help page can be reached by pressing the help button in the footer. Which would bring them to (Figure B-4).

3.7.3 - Functional Requirements

- N/A

3.8 Publish User Template

3.8.1 - Description and Priority

- Publishes the user's template as a fully-functional, active webpage
- **Priority - High:** Without this feature being functional, Project Wixs would fail on delivering its purpose to its user base.

3.8.2 - Stimulus/Response Sequences

- A Publish button should be available outside of the template editor, on the template selection screen (See Figure B-5), as well as within the editor itself (See Figure B-7).

3.8.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.4 - Retrieve (Get) User:** In order to get the user who will be referenced when updating the necessary templates.
- **4.12 - Update Template:** To update the selected template and set its 'active' flag to 'true' and all other templates associated with that user to inactive.

3.9 Delete User Template

3.9.1 - Description and Priority

- From the template selection page (Figure B-5), users will be able to delete existing templates as they see fit.
- **Priority - Medium:** it lies below *Template Creation/Edit* in terms of importance, but is still necessary to keep the template selection page as clean as possible.

3.9.2 - Stimulus/Response Sequences

- Templates may be deleted by pressing a "Delete Template" button underneath the desired template.

3.9.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.13 - Delete Template:** To facilitate the removal of the template from the system.

3.10 View Live Template

3.10.1 - Description and Priority

- A user's currently active template will be viewable to anyone with the Shareable live template link (which a user can find on their dashboard)

- **Priority - High:** Being unable to share or view the templates user's created would defeat the entire purpose of the site

3.10.2 - Stimulus/Response Sequences

- While editing, any changes made by the user are saved to the database
- Users can set any created template to be active from their dashboard. Upon selection, a Shareable live template link will be generated and displayed to the user.
- The aforementioned link will generate a publically viewable page based on the latest saved version of the user's current active template.

3.10.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.4 - Retrieve (Get) User:** To get the user from the URL so they active template can be retrieved.
- **4.15 - Get User Active Template:** To get the one template associated with the user which is flagged as active (published).
- **4.17 - Retrieve (Get) Content:** To get the individual content to be used within the templates grid engine.

3.11 Load site statistics (Admin)

3.11.1 - Description and Priority

- A page for system administrators allowing for the viewing of current active users and any other relevant statistics.
- **Priority - Low:** Statistics will be useful for administration, but not critical to the core Wixs experience.

3.11.2 - Stimulus/Response Sequences

- Templates may be deleted by pressing a "Delete Template" button underneath the desired template.

3.11.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.4 - Retrieve (Get) User:** To get the user accessing the web page so that they can then be checked for admin permissions.
- **4.5 - Update User:** When the admin updates another user's admin capabilities
- **4.6 - Delete User:** When the admin deletes a user's profile.
- **4.8 - Check if User is Admin:** Users only granted access to this page if they have been verified as administrators.
- **4.9 - List Users:** For listing the users within the 'user base' table.

3.12 User Content Management

3.12.1 - Description and Priority

- From the upload page, user's can see, add, and delete pictures they wish to be displayed on their site.
- **Priority - High:** Being able to upload and display media components is a key part to creating any professional site, and will be a high expectation from users.

3.12.2 - Stimulus/Response Sequences

- Users can select a file to upload from their Upload page.
- A PHP script will confirm the validity (file type, size) and save the image onto the database
- From the upload page any previously uploaded images will be displayed along with an option to delete them if desired.

3.12.3 - Functional Requirements

- **4.1 - Connect to Postgres Database:** In order to access necessary data stored in the database.
- **4.4 - Retrieve (Get) User:** In order to get the user associated with the content to be loaded.
- **4.16 - Upload (Create) Content:** When upload is clicked, the system relies on the process associated with creating content.
- **4.19 - List User Content:** For listing the associated user's content on the content management page.
- **4.18 - Delete Content:** For deleting the user's content.

Section 4 - Functional Requirements

This section describes the micro processes necessary to facilitate the features introduced in Section 3. Here, each process is provided a description of its function, the additional feature(s) it is dependent on to function, the inputs we intend it to receive, the outputs we intend it to produce, and the additional system features or processes affecting during the execution of that process.

4.1 Connect to Postgres Database

4.1.1 Description

- This refers to the system's ability to store data within the Postgres database contained on the Sandcastle server. This is an integral process acting as the backbone for nearly all user-based transactions within the application. After data is sent from the front-end to an exposed PHP file. The file will access secret credentials in order to open a connection with the Postgres database.

4.1.2 Depends On

- Function depends on the asynchronous request made to a PHP endpoint (See 4.20), which will then activate the file's retrieval of credentials and connection to the database.

4.1.3 Anticipated Inputs

- The process expects the credentials necessary to connect to the Postgres database, including the hostname, database name, port number, username, and password.

4.1.4 Anticipated Outputs

- Upon successful connection to the database, the corresponding PHP file's logic will execute, otherwise, an error message will be produced.

4.1.5 Anticipated Additional Effects

- Front-end will display a browser alert based on the response given from a successful/unsuccessful connection to the database

4.2 Validate/Sanitize Inputs

4.2.1 Description

- This is the process of checking all text inputted from users to ensure no script injections, database queries, or other malicious attacks are being attempted by a user.

4.2.2 Depends On

- Function depends on PHP scripts to 'clean' input. The recurring validInputs() function across various PHP scripts handling user input cleans the user's input by checking if empty, trimming of whitespace, using PHP built-in format checker (eg. checks if an email follows the email format), and 'sanitize' filters that removes special characters that could be read in as scripts. Also relies on the information being asynchronously communicated to a PHP file (4.20).

4.2.3 Anticipated Inputs

- Any text input (possibly malicious) from a user.

4.2.4 Anticipated Outputs

- The website functions as normally, with the next page being loaded as normally with any potential scripting attempts invalidated. If inputs are checked and deemed invalid, the PHP operation will not execute and the necessary response message will be sent.

4.2.5 Anticipated Additional Effects

- N/A

4.3 Create User

4.3.1 Description

- This process occurs whenever a user accesses, fills out, and submits the form on the RegisterPage.js page. This runs the addUser.php script, which will attempt to create an entry with the provided info in the users table.

4.3.2 Depends On

- This process requires connection to the database (4.1) to complete. Registration will be rejected if 15 accounts already exist on the system, as well as if a user is already registered with an email address. Also relies on the information being asynchronously communicated to a PHP file (4.20)

4.3.3 Anticipated Inputs

- User email, first name, last name, and password.

4.3.4 Anticipated Outputs

- Will return an appropriate message if limitations outlined in (4.3.2) are reached. Will successfully create an entry otherwise, invalidating any malicious scripts if detected (4.2).

4.3.5 Anticipated Additional Effects

- Users attempting to access the register page while logged in are redirected to their dashboard.
- The user's tuple will be added to the database.

4.4 Retrieve (Get) User

4.4.1 Description

- The retrieval of users is a highly important process for linking system transactions to an individual user. This process is queried whenever the website is launched and the user credentials are checked against stored cookies. On each page load, the process of retrieving a user takes place.

4.4.2 Depends On

- Process depends on a successful connection to the Postgres database. Considering this process relies on the cookies retrieved from the user's browser it also relies on the information being asynchronously communicated to a PHP file (4.20).

4.4.3 Anticipated Inputs

- User's email

4.4.4 Anticipated Outputs

- The user object (containing selected user's details) retrieved from the database

4.4.5 Anticipated Additional Effects

- N/A

4.5 Update User

4.5.1 Description

- The process of updating a user takes place on the admin-accessible screen. Here an admin can choose to set or unset another user's admin permissions.

4.5.2 Depends On

- For the front-to-back communication this relies on both (4.1) and (4.20).

4.5.3 Anticipated Inputs

- Setting as admin: The email of the user to be altered and true for the 'setAdmin' flag
- Unsetting as admin: The email of the user to be altered and false for the 'setAdmin' flag.

4.5.4 Anticipated Outputs

- A response message indicating that the user has been successfully updated based on the action provided.

4.5.5 Anticipated Additional Effects

- Updated users will now be able to access the admin page upon next page reload.

4.6 Delete User

4.6.1 Description

- As part of admin options, admins may delete users from the Project Wixs database at their discretion.

4.6.2 Depends On

- The ability to delete a user will be possible only if the user is a verified admin through functional requirement 4.8, if the list of users is successfully displayed through functional requirement 4.9, and if the Project Wixs database connection is active and functional.

4.6.3 Anticipated Inputs

- An admin must navigate to the “Admin Options” page, locate the user they wish to delete and then press the “delete user” button next to that user.

4.6.4 Anticipated Outputs

- If the deletion is successful, a pop-up dialogue box will appear on screen stating which user was deleted from the database and will prompt the user to dismiss the dialogue box, to ensure that they read and confirmed that they deleted the intended user.

4.6.5 Anticipated Additional Effects

- The “Project Wixs Userbase” table dynamically updates upon user deletion and so after a deletion, the page will automatically reload with the updated current user list.
- The user’s tuple will be removed from the database.

4.7 Logout User

4.7.1 Description

- When a user selects the ‘logout’ option within the nav bar, they will be unable to access the protected routes within the website upon page reload.

4.7.2 Depends On

- For proper database communication, the process will rely on 4.1 and 4.20.

4.7.3 Anticipated Inputs

- The user’s email.

4.7.4 Anticipated Outputs

- User’s session_id is updated to be empty within the database.

4.7.5 Anticipated Additional Effects

- Cookies storing user information (user’s email and session_id) are deleted.

4.8 Check if User is Admin

4.8.1 Description

- Must fulfil requirement of checking if a logged-in user is an admin or not in order to facilitate application options and features exclusive to admins. These admin features include: a unique Admin Options page of the website where an admin can grant/revoke admin privileges of other users as well as delete users.

4.8.2 Depends On

- The ability to check if a user is an admin is dependent on if the user has created an account or not. The PHP file to check if a user is an admin gets passed the stored cookie, "user" which corresponds to the user's email address that they used to create an account with Project Wixs. Therefore, if a user has not created an account, there will be no cookie data to pass.

4.8.3 Anticipated Inputs

- Stored cookie data from Project Wixs is required to search for the user in the database, and retrieve whether or not the user has admin access. Therefore, all that is required from the user is that they must create a valid account before being able to move forward with the admin check.

4.8.4 Anticipated Outputs

- No visual feedback such as a pop-up dialogue box will be received in either case of a user being an admin or not. The admin check is intended to be discreet for users, so that they are not aware of the existence of the additional admin options, and for admins, so that they can enjoy the subtle viewing experience that Project Wixs provides.

4.8.5 Anticipated Additional Effects

- If the user is found to be an admin, and they are signed in to their Project Wixs account, they will view a modified navigation bar that contains an "Admin Options" page where the user can access admin options.
- Otherwise, if the user is found to not be an admin, they will view the regular user site while signed in where they can create, publish and edit web pages.

4.9 List Users

4.9.1 Description

- When a user is a verified admin, they can view a table of Project Wixs users that updates in real-time and is editable by the admin.

4.9.2 Depends On

- A working connection to the Project Wixs database is required. The success of functional requirement 4.8 is required in order to verify that the signed-in user is a verified admin. Also relies on the information being asynchronously communicated to a PHP file (4.20)

4.9.3 Anticipated Inputs

- A user must have created, and be signed into their admin account in order to see the list of users. The cookie data for the user is checked with functional requirement 4.8 to verify if they are an admin. Only if the user data inputted for the admin check is verified, will all user information be pulled through database requests and dynamically inputted into the user table.

4.9.4 Anticipated Outputs

- If a user is an admin, the “Project Wixs Userbase” table will display on the “Admin Options” page. Options for each user in the database should be: “set admin” which grants a user admin privileges, “unset admin” which revokes a user’s admin privileges, and “delete user” which will delete a user entirely, including all associated data such as uploaded files and templates. Otherwise, if the user is not an admin, they will not have access to either table nor page.

4.9.5 Anticipated Additional Effects

- N/A

4.10 Create Template

4.10.1 Description

- This function deals with the process of accepting a custom template name from the user’s dashboard screen and creating a new template. The user can create up to 4 new templates. The newly created template is initialized with a tutorial template showing off how to make use of the editor

4.10.2 Depends On

- Communication with the database, facilitated by 4.1 and 4.2. Also depends on retrieving the user in order to identify the creator of the template (4.4)

4.10.3 Anticipated Inputs

- The creating user’s email, the custom template name, and a hard-coded tutorial template.

4.10.4 Anticipated Outputs

- A response message, identifying the success or failure of the process.

4.10.5 Anticipated Additional Effects

- The template tuple will be added to the database.
- The user’s dashboard will populate with the newly created template upon page reload.

4.11 Retrieve (Get) Template

4.11.1 Description

- Throughout various points of the site, a user’s template(s) may be requested for editing or viewing

4.11.2 Depends On

- When editing or viewing templates on the dashboards, only templates that are associated with the current user (identified by owner_email on the database) are loaded. This of course also assumes, that the user is logged in
- When viewing a template set active (accessed by the Shareable live template link) only a connection to the database is needed to retrieve the publically viewable site.

4.11.3 Anticipated Inputs

- The only input for templates to edit would be a session ID to identify the user. No input is needed for the shareable live template.

4.11.4 Anticipated Outputs

- Templates are stored as JSON strings; upon retrieving a template these strings are decoded and the user's view populated with elements as specified.

4.11.5 Anticipated Additional Effects

- N/A

4.12 Update Template

4.12.1 Description

- While in the editor, any changes made to the template are saved inconspicuously in the background.

4.12.2 Depends On

- When first loaded, the template is verified to be associated with the current user (identified by owner_email on the database). This of course also assumes that the user is logged in.

4.12.3 Anticipated Inputs

- When making any change to the template, the applyChange() function is called in the editor. This 'stringifies' the current template's layout into a JSON string, and executes a PHP script to update the database.

4.12.4 Anticipated Outputs

- No visible output is presented to the user to keep the saving seamless. The next time the template is retrieved. the changes will be visible.

4.12.5 Anticipated Additional Effects

- N/A

4.13 Delete Template

4.13.1 Description

- Templates the user no longer needs can be deleted from the dashboard page.

4.13.2 Depends On

- A user has to be logged in (and therefore a valid database connection) to view and select a template to delete.

4.13.3 Anticipated Inputs

- Selecting the delete button will call the delTemplate() function, passing the id to a PHP script that drops the selected template.

4.13.4 Anticipated Outputs

- If successful, an alert message informing the user will appear, and the page will be refreshed showing the updated listing of templates.
- Otherwise, an error message is displayed in an alert to the user.

4.13.5 Anticipated Additional Effects

- Deleting an active template will *not* select another remaining template as active. The user will simply have no active templates, and attempting to the live template link will tell the visitor this.

4.14 List User Templates

4.14.1 Description

- On the user's dashboard, all the user's current templates are fetched and displayed in an organized manner to be accessed and modified.

4.14.2 Depends On

- A user has to be logged in (and therefore a valid database connection) to view their dashboard.

4.14.3 Anticipated Inputs

- No input is needed from the user aside from simply navigating to their dashboard.

4.14.4 Anticipated Outputs

- Upon loading, a PHP script queries the database to return a list of templates associated with the current user. These are mapped to CustomCard elements that display the name, last update, and the operations a user can perform (set active, edit, and delete) as buttons.

4.14.5 Anticipated Additional Effects

- N/A

4.15 Get User Active Template

4.15.1 Description

- This process retrieves the single template owned by a user whose 'isActive' flag has been set to true. When a template is set as active on the user's dashboard, all other templates owned by the user will be set as inactive, guaranteeing that only one template will be active at a time. This allows for the public viewing (and sharing) of a link containing each user's live template. If one exists, the live page will display it, otherwise, a message will indicate they do not have one active.

4.15.2 Depends On

- A successful connection to the database (4.1) and the user credentials to be obtained from the URL so it can be asynchronously sent to the server for a response (4.20).

4.15.3 Anticipated Inputs

- The user's email (unique identifier), obtained from the URL

4.15.4 Anticipated Outputs

- The template object is returned and the layout data is loaded into the live grid engine to be displayed.

4.15.5 Anticipated Additional Effects

- N/A

4.16 Upload (Create) Content

4.16.1 Description

- Allows for the adding of user content to the database and storage of the uploaded file within Sandcastle's file system. This process can only be performed from the user's 'Upload' page. The content is limited to a max size of 10MB.

4.16.2 Depends On

- A successful connection to the database (4.1) and the uploaded file to be asynchronously sent to the backend (4.20). Also relies on knowing the author of the transaction taking place (4.4) in order to link the content.

4.16.3 Anticipated Inputs

- The user's email and the file to upload.

4.16.4 Anticipated Outputs

- A response message is produced confirming that the content has either been uploaded or if failure has occurred.

4.16.5 Anticipated Additional Effects

- The content is added to the users collection and viewable in either the 'Upload' page or within the template edit area.

4.17 Retrieve (Get) Content

4.17.1 Description

- Retrieves an individual content based on the provided content_id. This is mainly used in the template editor and live template display pages for individual image components that reference them.

4.17.2 Depends On

- A successful connection to the database and transmission of data from client to server (4.1) and (4.20).

4.17.3 Anticipated Inputs

- The content's content_id.

4.17.4 Anticipated Outputs

- The content is returned as an object containing all its details obtained from the postgres database. The component can make use of this data to get the image by making reference of the images storage location within the server so it can be displayed.
- If operation fails, an error message will be produced for the console.

4.17.5 Anticipated Additional Effects

- The content will be displayed in its corresponding location within the editor or live grid engine.

4.18 Delete Content

4.18.1 Description

- This process deals with the removal of a user's content. It is only triggered when the user is on their 'Upload' page and selects the 'Delete' button that is generated for each content's preview card.

4.18.2 Depends On

- Depends on a successful connection to the database and transfer of data to the back-end (4.1) and (4.20). Also depends on the initial listing of user content so that there is the option to delete it (4.19).

4.18.3 Anticipated Inputs

- The content_id of the content to be deleted.

4.18.4 Anticipated Outputs

- Response of whether the operation was a success or not.

4.18.5 Anticipated Additional Effects

- The content's tuple is deleted from the database.
- Upon page reload the content no longer appears on the user's 'Upload' page.

4.19 List User Content

4.19.1 Description

- This process allows the user to view their uploaded content on the 'Upload' page of the website. The user's content is gathered so they can be displayed within individual cards with descriptions of the content's file name and a button allowing the deletion of content.

4.19.2 Depends On

- Successful communication with the Postgres database (4.1) and the asynchronous request of the content (4.20). Also depends on retrieving the user whose content must be displayed (4.4).

4.19.3 Anticipated Inputs

- The user's email: for content retrieval

4.19.4 Anticipated Outputs

- An array of all content objects contained within the database that belong to the user

4.19.5 Anticipated Additional Effects

- The content will be rendered and displayed on the 'Upload' page upon page reload.

4.20 HTTP AJAX Data Transfer

4.20.1 Description

- This process serves to be the main architecture the front-end and back-end communication is built upon. The front-end will send any data obtained from input forms

asynchronously, using a React helper called Axios, to a PHP endpoint which will then deal with the data. Overall, the process takes care of connecting the client-facing operations with the server-facing operations. Data is sent (via Axios) using either GET or POST requests and the necessary data is included in the HTTPs body parameters. In the PHP file, the data is individually obtained from the request parameters and used to perform operations within the database.

4.20.2 Depends On

- Depends on any other process listed within this section that requires the transferring of data. (4.3 - 4.19)

4.20.3 Anticipated Inputs

- Axios requires a few details in order to work: the request type (eg. GET, POST, etc.) and the data to be sent in the parameters (eg. username, template_id, content_id, etc.)

4.20.4 Anticipated Outputs

- The response object from the PHP file which can be used to navigate the webpage after, create an alert, or alter state variables (React state variables).

4.20.5 Anticipated Additional Effects

- Various possible additional effects depending on the root user process that invoked this system process.

Section 5 - External Interface Requirements

This section describes the various interfaces necessary for cohesive communication within different layers of the overall system architecture.

5.1 User Interfaces

Although implementation for Project Wixs has not officially commenced, the system's front-end will still follow generally defined guidelines. Concept images for basic page layout can be found in Appendix B under Concept Page Layouts and a styling guide dictating colour palette, typography, and spacing can be found in Section 6. Majority of the navigable web pages present in the system will include shared components such as the project logo, header, and footer areas in order to maintain consistency. Based on HCI concepts, all inputs and operations will be designed to provide users with accurate feedback on what just occurred. Where possible, tooltips (when hovering over links) will be implemented to ensure ease of use. Body text will always be at an adequate contrast to the background of its respective container for a better viewing experience. Currently, there are no planned keyboard shortcuts for use within the system. The webpage will not make use of third-party ads or tracking systems at this time.

5.2 Hardware Interfaces

Project Wixs' front-end interface will be viewable from any device able to use an internet browser, however, the interface design will be best optimized for large-screened devices like Laptops and Desktop computers. With the system being hosted on Brock University's Sandcastle, this server will be the main hardware component controlling server-side operation of the database and hosting of system-relevant files. Beyond the aforementioned devices, the system does not interact with other forms of hardware.

5.3 Software Interfaces

In order for the front-end interface to display and operate as intended, JavaScript must be enabled within the client browser. Depending on the convenience of operations, small helper components may be imported (and adequately attributed) for the lower-functioning tasks of the system. For example, to facilitate the use of the AJAX technique, Axios² may be used alongside React.js to enhance the process. Time permitting, Google's reCaptcha may be introduced to better the security of the login system.

The system will operate on the following back-end technologies:

- Red Hat Linux
- PHP v5.4.16 (minimum)
- PostgreSQL v9.2.24 (minimum)

The system will operate on the following front-end technologies:

- HTML - DOM HTML support enabled
- CSS/SASS
- JavaScript
- React.js

5.4 Communication Interfaces

The system will make use of various standards and protocols in order to ensure accurate delivery and processing of information. Being hosted on the web, Project Wixs will follow HTTP and TCP/IP protocols for data transfer. To facilitate the generation of dynamic web pages, PHP will be utilized through the CGI protocol. HTTP interactions will be handled with AJAX technique for asynchronous data exchange. In order to connect to the Brock Sandcastle server, the system (and the team) will make use of the SSH protocol to allow for secure client-side access to the server over an unsecured network. In order to communicate with the database, queries will be formatted to follow PostgreSQL syntax. PHP will communicate with Postgres using the PDO module. User passwords will be MD5 hashed before transfer to the database.

Section 6 - Other Nonfunctional Requirements

² "axios - npm." 22 Jan. 2020, <https://www.npmjs.com/package/axios>. Accessed 4 Feb. 2020.

This section describes the aspects of the system which define how the system will go about performing the features detailed in Section 3 (System Features). Here constraints, limitations, and assertions are discussed in order to validate any features discussed.

6.1 Performance Requirements

In order to facilitate timely page loads and data transfer, the system's front-end architecture will make use of the AJAX technique for asynchronous client and server-side communications; this is crucial for system response without page refreshes and dynamically displaying data. No process should take over a minute (60 seconds) of time to complete, whether displaying, saving, or retrieving system data. Any tutorials or user-guidelines necessary to facilitate the use and mastery of the published system will be available on the Help Page (See Section 3.6).

6.2 Safety Requirements

In order to keep the system contained and with a minimal impact on host servers, quantifiable limits will be put in place for various different features. The file size of content uploads will be restricted to 10MB for photos. The system will not allow for the uploading of videos due to the possible vulnerabilities for file uploads that it introduces (in addition to reduced site speed performance). Instead, any videos used on the website are referenced and accessed from external video-hosting websites like YouTube or Vimeo. Up to 15 accounts may be registered within the system at any time (with one being reserved for an administrative account). Each of these user accounts may have up to four (4) custom templates stored at any time. For a documented safe intended use of the system, a tutorial/walkthrough will be provided (See Section 3.6).

6.3 Security Requirements

In general, system front-end access will be limited (with necessary page redirects) to those who have a registered account in the database. In order to maintain user privacy and security, a number of measures are being taken. User uploaded files will be matched with a unique identifier to them and only viewable (or usable in templates) solely by themselves. Upon account creation, user passwords will be hashed and stored in the database so that those with direct database access cannot obtain the password. The system will also require that user-created passwords, first meet a specified criteria to ensure password safety (See Section 1.5). All forms containing input fields will undergo a multi-staged validation process before the inputted data can interact with the database or any other state-altering systems. In this validation process, inputs are first checked to ensure the necessary inputs have been supplied, trimmed of whitespace, checked for proper formatting (eg. whether an email, URL, or of another special format), and Authenticated routes for site data display and user account administration will be available only to accounts with admin permissions.

6.4 Software Quality Attributes

6.4.1 Maintainability

A set team member or two will be in charge of ensuring files are up to standard, however, for the duration of this project the system will be rotationally maintained (whether it be commenting, bug fixes, or algorithm optimization) by all members of the Project Wixs team as necessary. Where possible, all source code files will contain a comment block including project name, description of file processes, and system version number. To keep a simple architecture, the system will run on as minimal an amount of files as possible, while maintaining functionality. In addition, multiple iterations of the files controlling system design will exist within a shared GitHub repository.

6.4.2 Availability

The system is available to anybody with an internet connection and with a firewall configuration allowing the viewing and execution of files on Brock's Sandcastle server. Considering this is meant to act as a personal website for most, the system will be expected to display 24/7. As the system will be hosted on Brock's Server, its uptime (and overall availability) will mirror that of Sandcastle's.

6.4.3 Reliability

As mentioned in Section 5.2, various validity checks will be performed on user input to ensure it is of an adequate size, representation, and reasonability. Exception handling will be implemented for possible null values, invalid data or overflows.

6.4.4 Portability

The systems software will have a relative ease when it comes to portability as it will make use of languages with a wide coverage of support across platforms, hosting environments, and compilers. 0% of the components in the system will be dependant on the host server being used (in this case, Sandcastle). All languages being used for the front-end and back-end are widely used and highly portable. Project Wixs' main front-end language, React.js, is one of the more recently introduced languages. Based on various statistics³, it is clear React.js is one of the most popular and well supported front-end javascript frameworks to use right now. The main back-end language, PHP, has been a staple of server-side programming for decades. It is currently maintained that PHP is used by 79% of websites with a known server-side language⁴.

Section 7 - Change Management Process

This section on the change management process goes into finer detail on how the Project Wix team handles the change management process to be used to identify, log, evaluate, and update

³ "2019 Stats on Top JS Frameworks: React, Angular ... - Tecla.io." 10 Jun. 2019, <https://www.tecla.io/blog/2019-stats-on-top-js-frameworks-react-angular-and-vue/>. Accessed 6 Feb. 2020.

⁴ "Usage Statistics and Market Share of PHP for ... - W3Techs." <https://w3techs.com/technologies/details/pl-php>. Accessed 6 Feb. 2020.

the SRS to reflect changes in project scope and requirements. When undergoing a system change, the Project Wixs team follows a structured process in order to ensure any alterations are handled in as efficient of a manner as possible. When a change is implemented, the team will be sure to maintain an updated source code, repository, message board, and SRS. Below the tools used to aid in the management of system changes are described:

7.1 Discord

Within the Project Wixs Discord, the team is able to handle most of the detailed communication necessary to initiate, process, and finalize a system change. On an ongoing basis, new ideas are internally formulated and vetted for feasibility then delegated to members based on experience, schedule, and necessity. Ultimately, Discord is the main message board used to maintain a cohesive flow of communication within the team.

7.2 GitHub

GitHub acts as the team's main repository for the source code, issue management, and task delegation. Internally, when it is decided a change is needed to be made, the issue will be created and described in as thorough detail as possible to ensure no information can be misinterpreted. The issue is then delegated to member(s) based on previous communications within the aforementioned message board (See Section 7.1). When an issue is being worked on, it will be tested and updated within a separate 'development' branch before being reviewed by members involved in the change, and merged with the 'master' branch upon approval. Also a project roadmap is used to better visualize the current progress of issues within a project. External users of the system (non-team members) who wish to see a change may visit the repository to fork, branch from, or produce a new issue for the Project Wixs team to review.

7.3 Google Drive

The Google Drive allows the team to store any files necessary for recurring usage by Project Wixs team members. Most importantly, here is where the SRS is stored and regularly updated within a Google Doc. The benefit of documenting changes here is that members can concurrently edit, view, and comment on changes being made by other team members. After a change has been discussed within the message board then completed within the GitHub repository, the change will be reflected within the SRS as a final statement reflecting the details of the change that has been made.

Section 8 - Legal, Copyright, and Other Notices

8.1 MIT License

Description

The Project Wixs system is licensed under the MIT License, meaning permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without


limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions. For the full description of the license and its limitations, visit: <https://github.com/DrayWilliams1/ProjectWixs/blob/master/LICENSE.md>

Section 9 - Other Requirements


9.1 Style Guide

Style Guide
ProjectWixs


Logo Evolution




v1.0




v2.0




v3.0



v4.0



v5.0



v6.0 (Final)

Colour Palette

	Hex: #2f4858 Name: Primary 1
	Hex: #33658a Name: Primary 2
	Hex: #f6ae2d Name: Accent 1
	Hex: #f26419 Name: Accent 2
	Hex: #55dde0 Name: Accent 3
	Hex: #202020 Name: Black
	Hex: #464646 Name: Grey

Typography

Font-Family: Open-Sans

Header 1

28pt - #202020 (Black)

Header 2

22pt - #202020 (Black)


Header 3

18pt - #464646 (Grey)

Header 4

12pt - #464646 (Grey)

Element Spacing



Elements such as the **ProjectWixs** logo should have margins of no less than a third of the height and width.

Appendices

Appendix A - Glossary

Acronyms/Abbreviations

AJAX: Asynchronous JavaScript and XML

CSS: Cascading Style Sheets

CGI: Common Gateway Interface

CMS: Content Management System

DOM (Browser): Document Object Model

Doc: Document

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

IE: Internet Explorer

MB: Megabyte (~1,000,000 bytes)

MIT: Massachusetts Institute of Technology

Postgres: PostgreSQL relational database

PHP: Hypertext Preprocessor

Repo: Repository

REST: REpresentational State Transfer

SASS: Syntactically Awesome Style Sheets.

SRS: System Requirements Specification *or* Software Requirements Specification (identical meaning in this document's context)

SSH: Secure Shell

SQL: Structured Query Language

UI: User Interface

UX: User Experience

W3C: World Wide Web Consortium

x86-64: the 64-bit version of the x86 instruction set.

Definitions

Back-End: Aspects of the system regarding the server-side operations and architecture. Mainly referring to the sandcastle, PHP scripts, and interaction with the Postgres database.

Branch (GitHub): A version of a given GitHub repository allowing for changes without affecting the branch it was derived from (typically derived from the repo's "master" branch).

Chromium: An open source web browsing software by Google.⁵

Discord: VoIP application providing various channels of communication between individuals, a group, or a community, including video, audio, images, and text.

Encrypted/Encryption: is a process that encodes a message or file so that it can be only be read by certain people.⁶

Firewall: A part of a computer system or network that is designed to block unauthorized access by monitoring inward and outward traffic.

Fork (GitHub): A fork is a copy of a GitHub repository that allows users to freely experiment with changes without affecting the original project.

Framework (Software): A packaged abstraction of another software's functionality, allowing for the introduction of additional/altered features while maintaining core functionality of the original.

Front-End: Aspects of the system regarding the client-side operations and architecture. Mainly referring to React.js (a JavaScript framework) files, HTML, and CSS/SASS.

⁵ "Chromium - The Chromium Projects." <https://www.chromium.org/Home>. Accessed 29 Jan. 2020.

⁶ "What is Encryption & How Does It Work? - Search Encrypt" <https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537>. Accessed 13 Apr. 2020.

Google: an internet search engine founded in 1998 by Sergey Brin and Larry Page that searches the World Wide Web for particular information specified in a textual web search query.

Linux: the most-used open source operating system.

Repository (GitHub): A directory or storage space that can hold GitHub projects including text, image, and code files.

RESTful (programming): Web services that conform to the REST architectural style are called RESTful Web Services. Representational State Transfer (REST) is a software architectural programming style that provides interoperability between computer systems on the Internet. When HTTP is used, it makes use of GET, POST, PUT, and DELETE for lookups, mutation, creation, and deletion respectively.

Sanitize/Sanitizing: The process of cleansing and scrubbing of user input to prevent it from jumping the fence and exploiting security holes.⁷

Sandcastle: Brock University Computer Science department's primary hosting server.⁸

SASS: By the official website's definition, SASS is "the most mature, stable, and powerful professional grade CSS extension language in the world."⁹

Whitespace: Leading or trailing space characters in an input accepting text.

⁷ "Prevent Web Attacks Using Input Sanitization - eSecurity Planet." 26 Oct. 2012, <https://www.esecurityplanet.com/browser-security/prevent-web-attacks-using-input-sanitization.html>. Accessed 28 Jan. 2020.

⁸ "HOW-TO Index - Computer Science - Brock University." <https://www.cosc.brocku.ca/help/howto>. Accessed 6 Feb. 2020.

⁹ "Sass: Syntactically Awesome Style Sheets." <https://sass-lang.com/>. Accessed 13 Apr. 2020.

Appendix B - Analysis Models

System Architecture

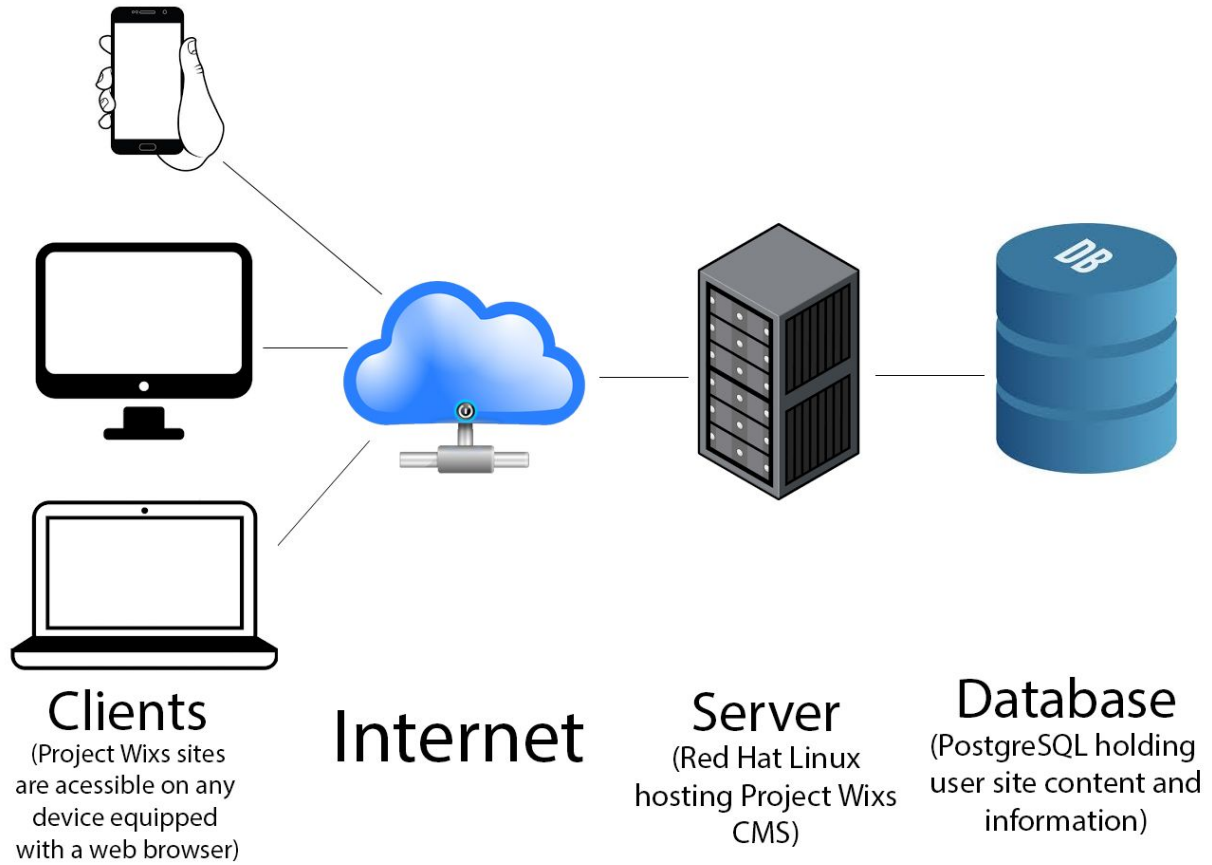


Figure B-1: Basic System Architecture

ER Model Diagram

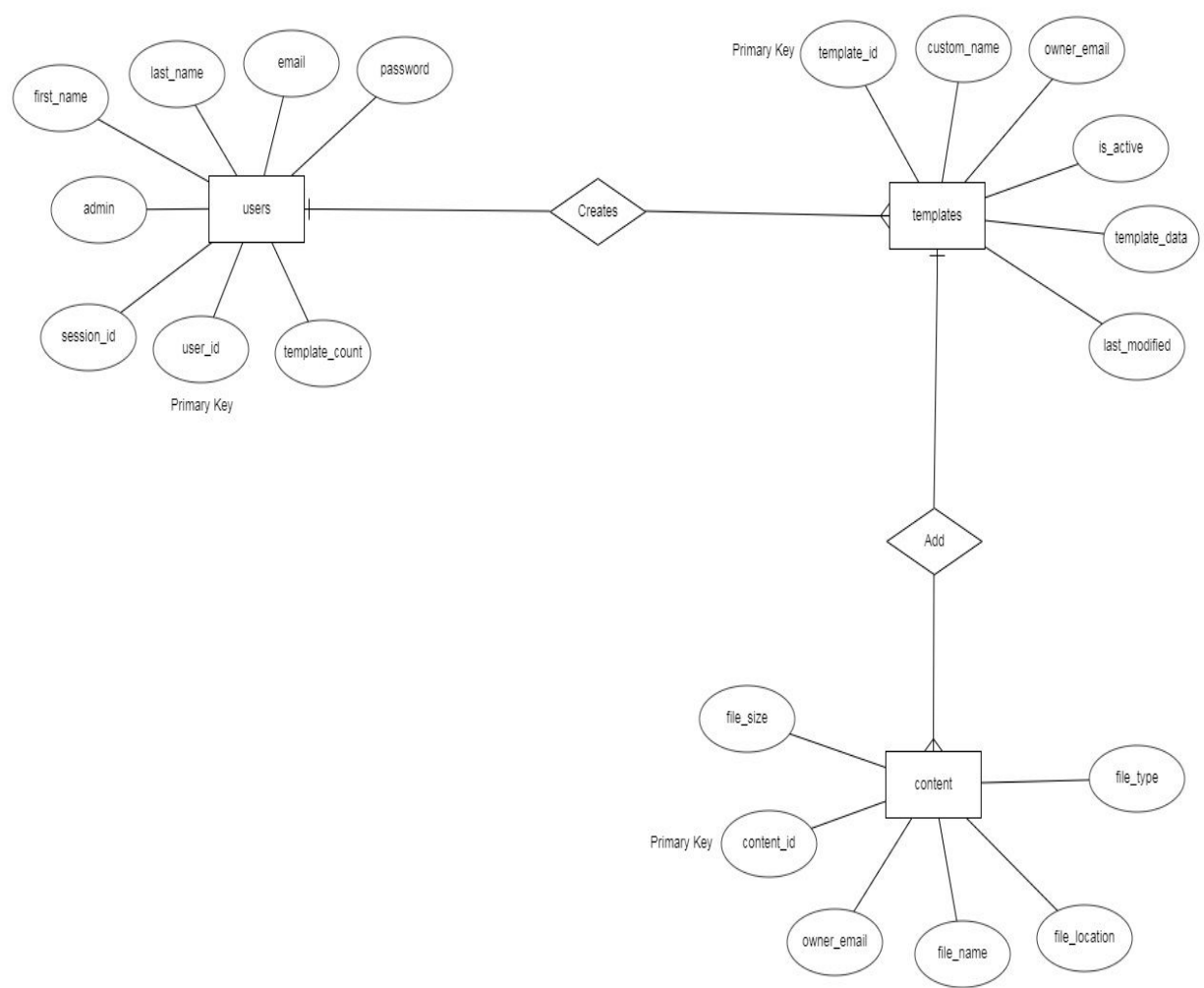


Figure B-2: Project Wixs ER Diagram

Use Case Diagrams

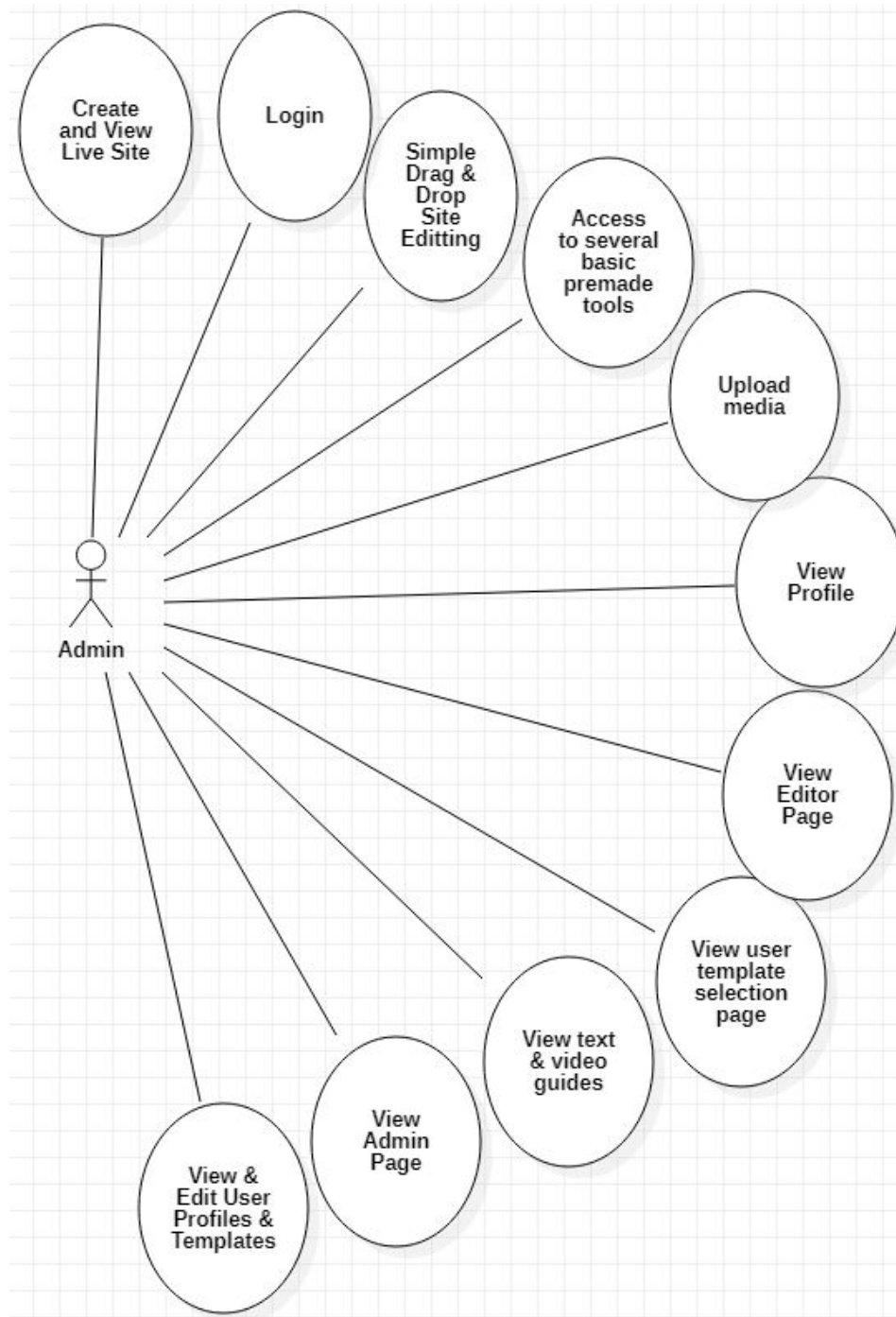


Figure B-4: Use Case Diagram for an Admin

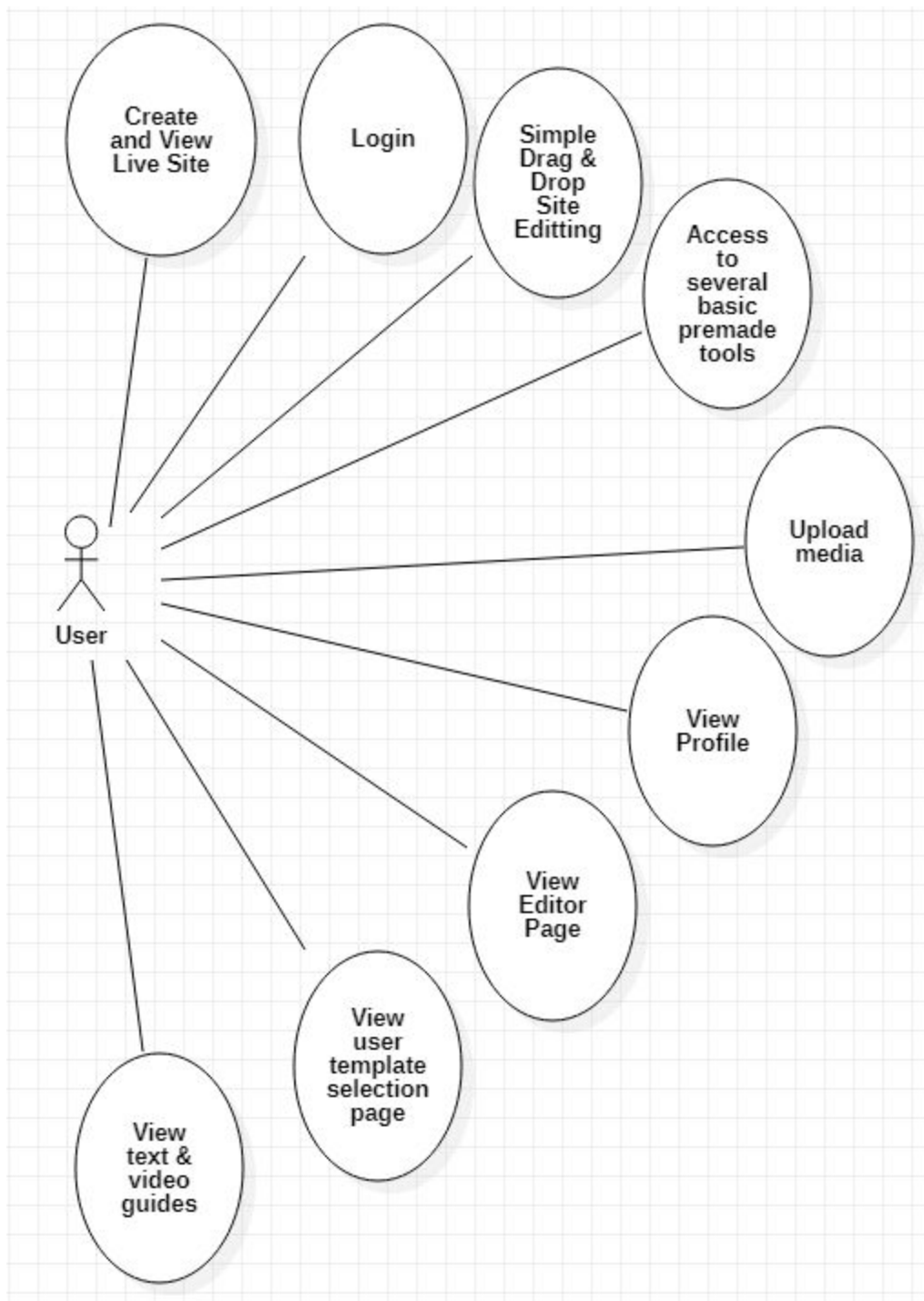


Figure B-5: Use Case Diagram for a Regular User

Concept Page Layouts

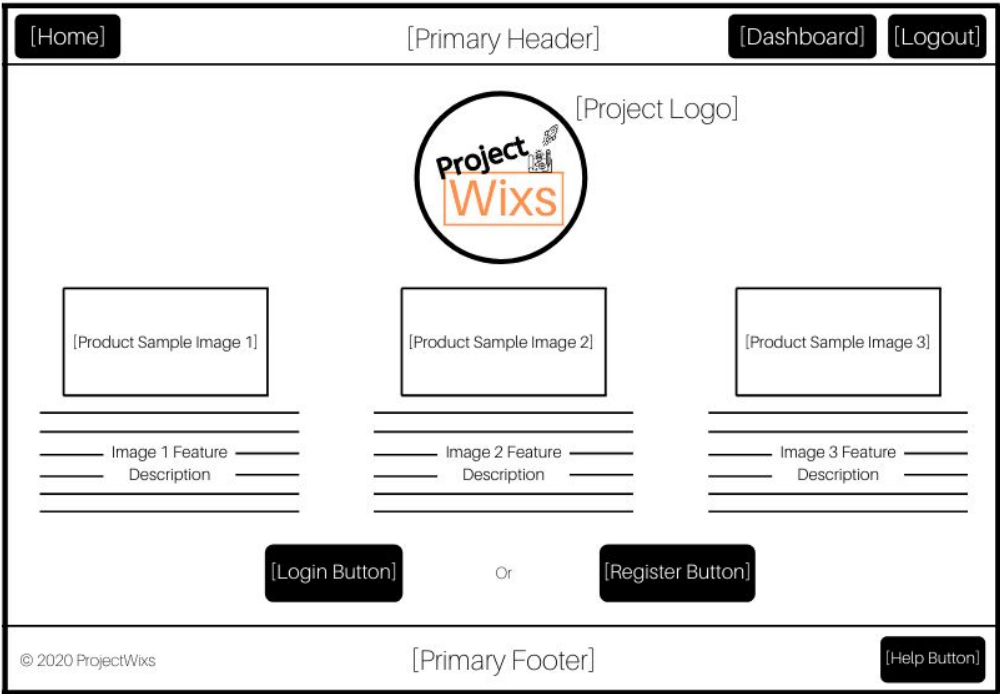


Figure B-6: Concept Landing Page

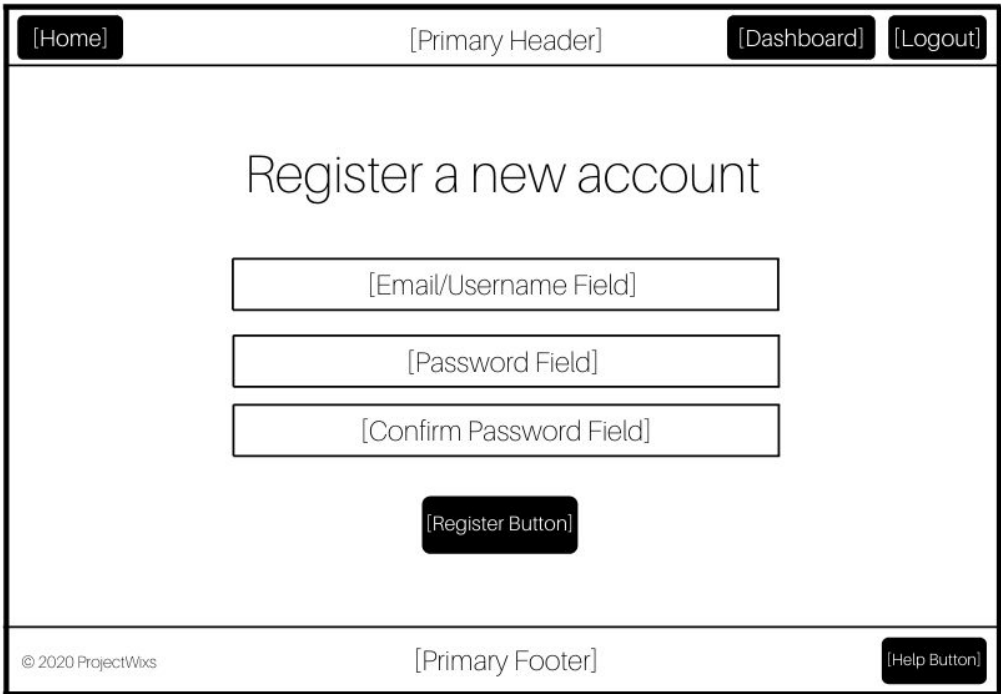
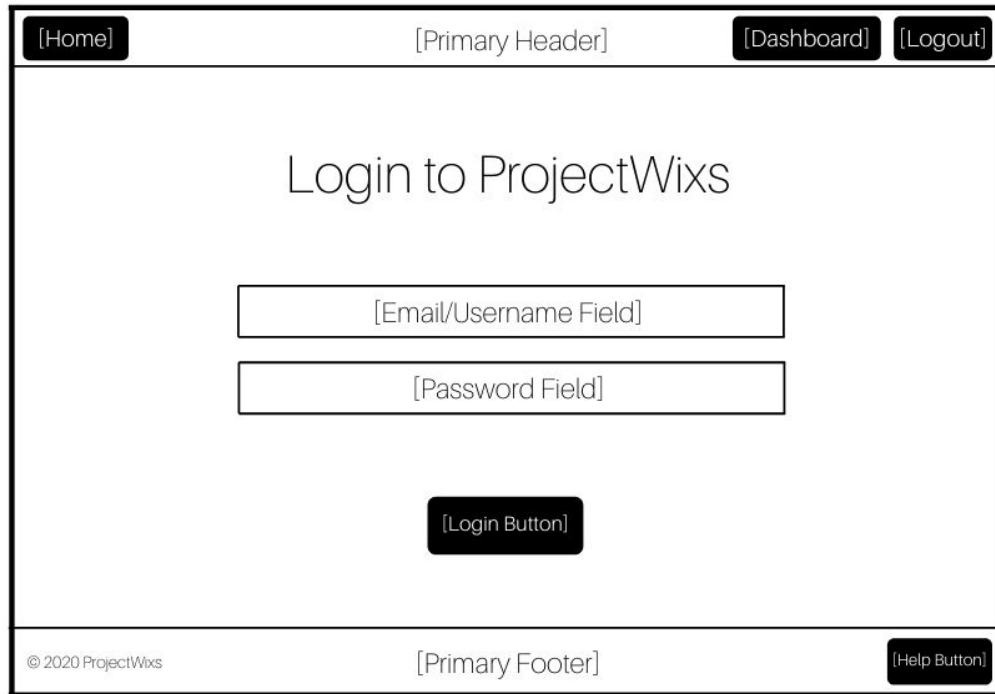
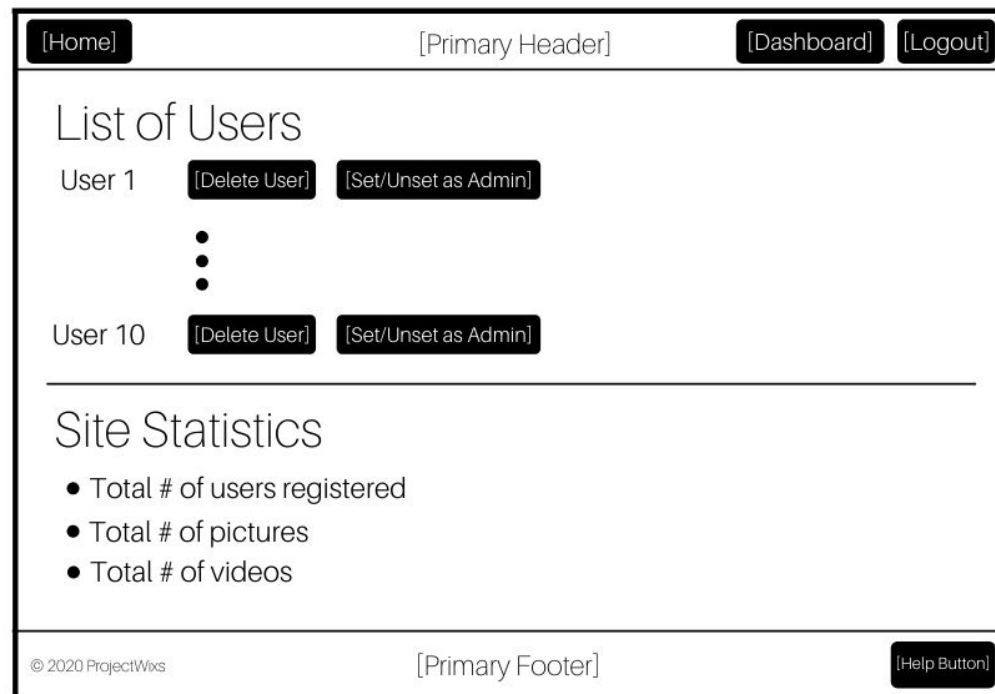


Figure B-7: Concept Registration Page



The wireframe for the login page features a header with three elements: a [Home] button on the left, a [Primary Header] label in the center, and [Dashboard] and [Logout] buttons on the right. The main content area is titled "Login to ProjectWixs" and contains two input fields: "[Email/Username Field]" and "[Password Field]". Below these fields is a [Login Button]. The footer consists of a copyright notice "© 2020 ProjectWixs" on the left, a [Primary Footer] label in the center, and a [Help Button] on the right.

Figure B-8: Concept Login Page



The wireframe for the admin tools page has a header identical to the login page. The main content area is titled "List of Users" and displays a list of users. Each user entry includes a user identifier (e.g., "User 1", "User 10"), a vertical ellipsis indicating more options, and two action buttons: "[Delete User]" and "[Set/Unset as Admin]". Below the user list is a section titled "Site Statistics" which contains a bulleted list: "Total # of users registered", "Total # of pictures", and "Total # of videos". The footer is identical to the login page, showing the copyright notice, primary footer label, and help button.

Figure B-9: Concept Admin Tools Page

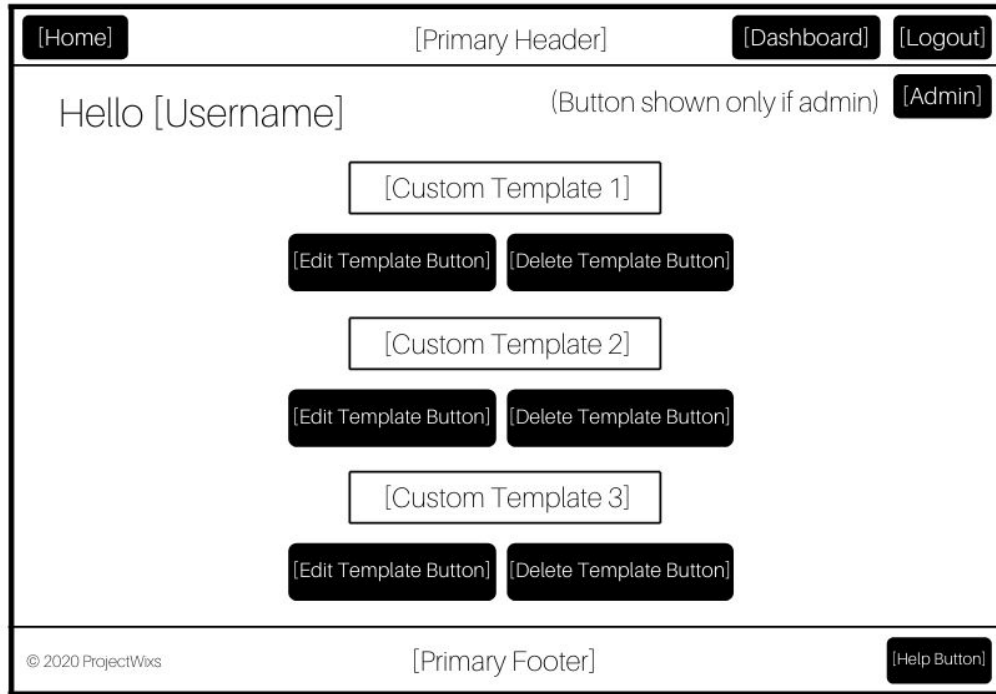


Figure B-10: Concept Template Selection Page

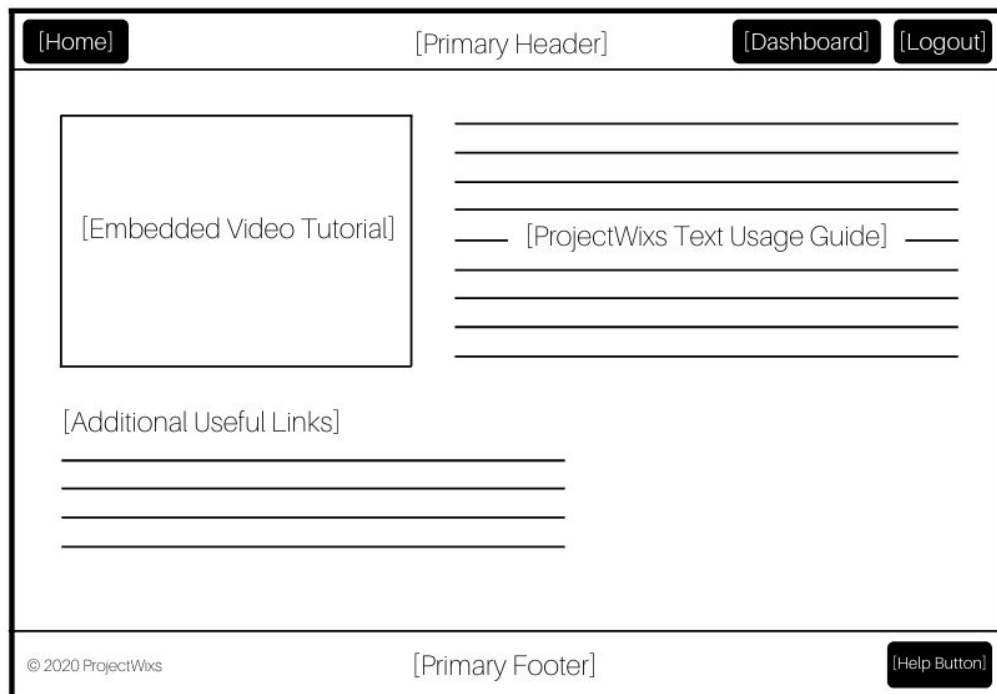


Figure B-11: Concept Help Page

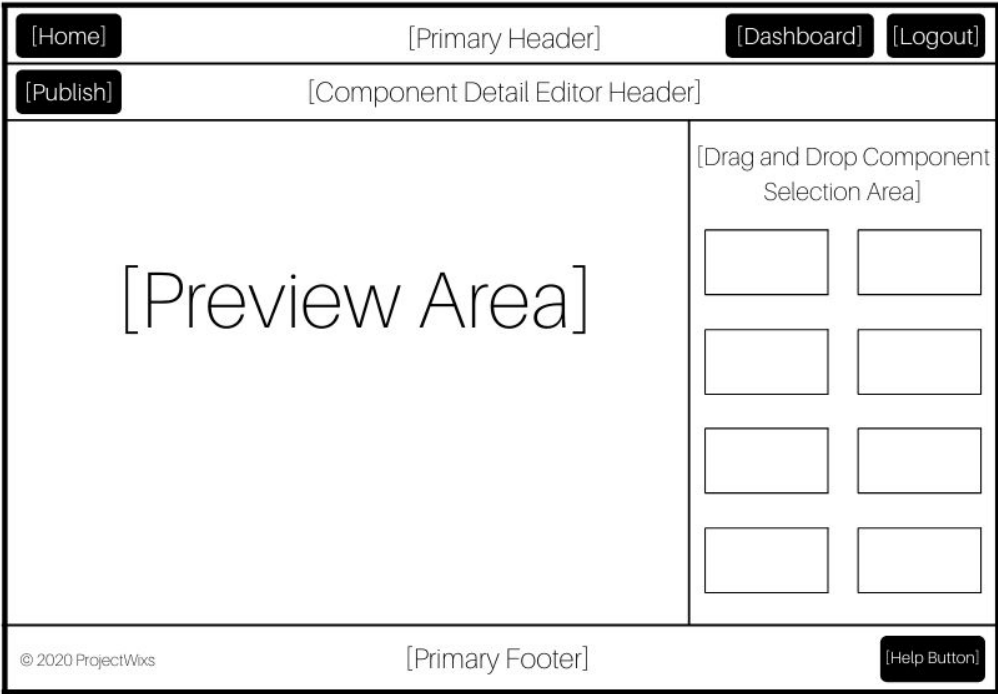


Figure B-12: Concept Template Editor Page

Sequence Diagrams

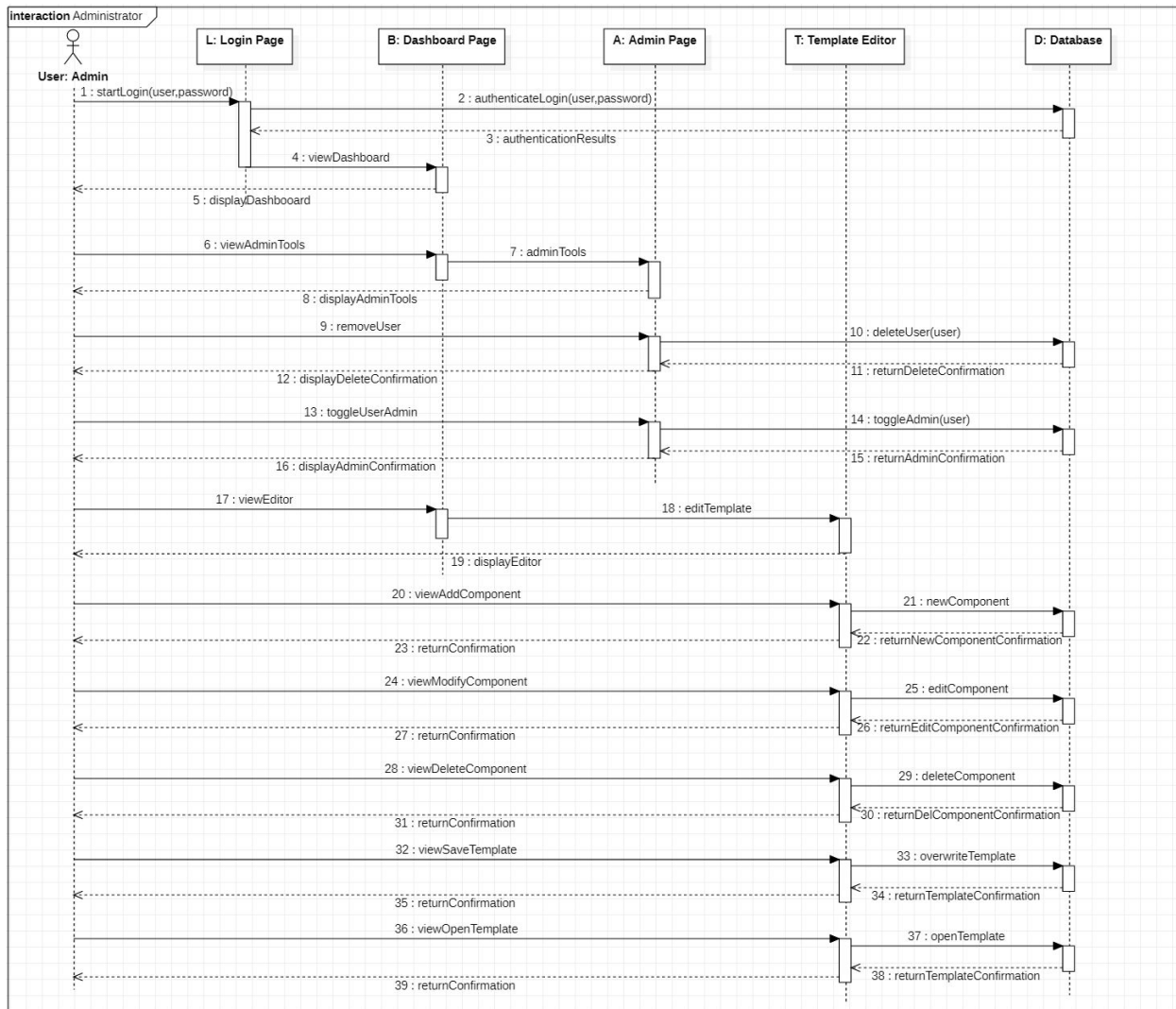


Figure B-13: Administrator Sequence Diagram

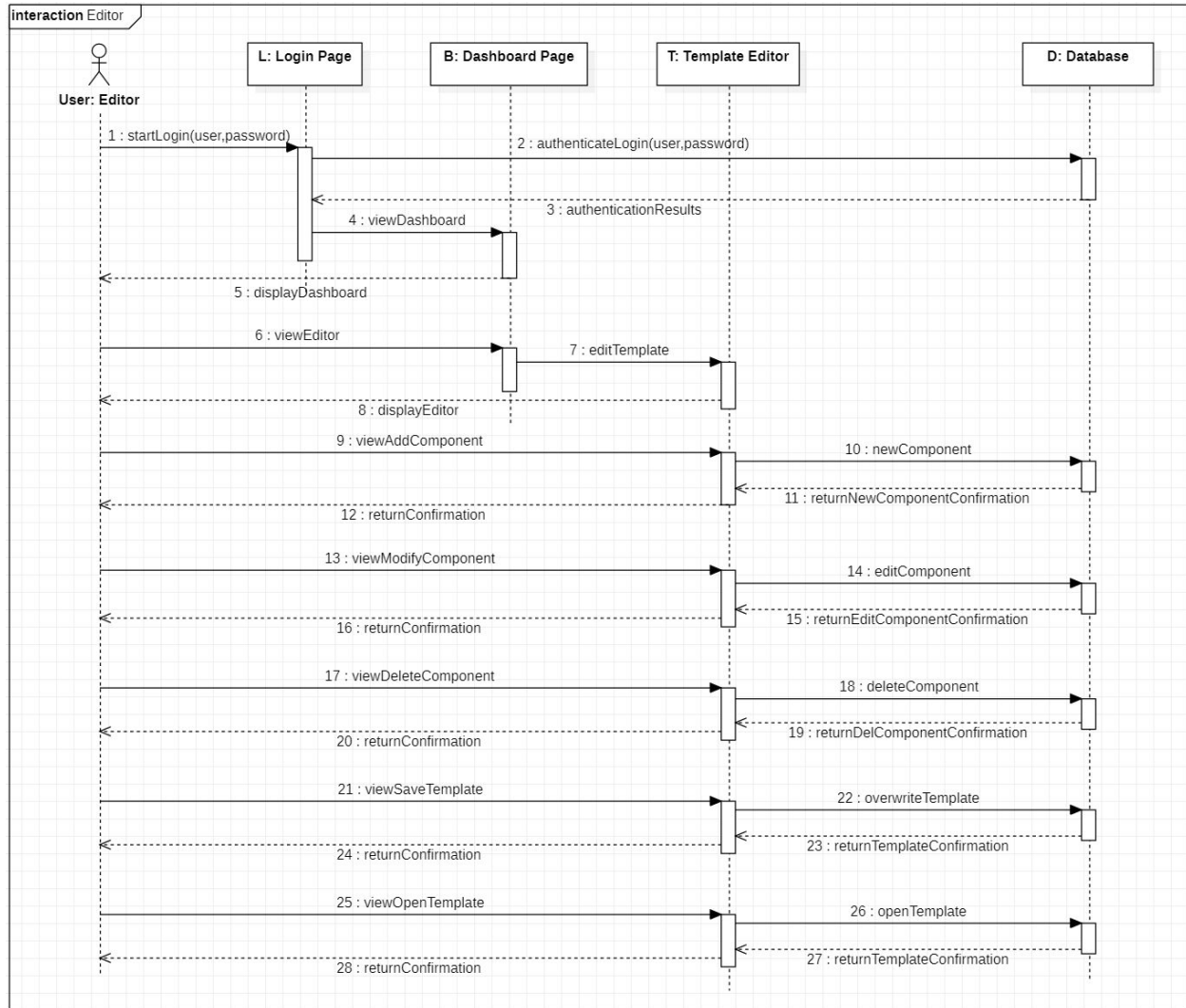


Figure B-14: Editor Sequence Diagram

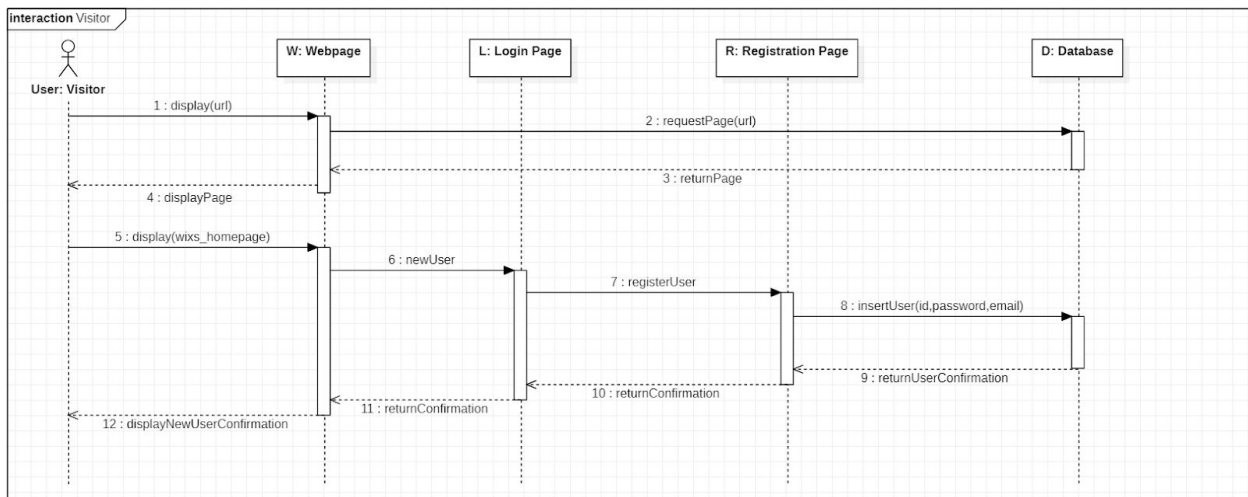


Figure B-15: Visitor Sequence Diagram

Appendix C - Issues List

- I. As of the current version of this document, limits on accounts to be created and content upload file sizes are general guidelines. The Project Wixs team is limiting the numbers to what we feel are adequate for basic testing for how we intend the system to be used. As implementation continues, limits can adjust to reflect changes. Some set limits will not reflect the actual strength/durability of the system, rather, they exist to define the user experience within a sandbox to reduce possible system-breaking use.