**Mini WKU Wiki**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Milestone 4

CS 360 (Section 2)

**Client:**

Manh Do: manh.do226@topper.wku.edu

**Team:**

Steven Booth: steven.booth527@topper.wku.edu

Peter Harlan: peter.harlan996@topper.wku.edu

Daniel Monterrosas: daniel.monterrosas118@topper.wku.edu

Michelle Schapmire: [michelle.schapmire782@topper.wku.edu](mailto:michelle.schapmire782@topper.wku.edu)

Zach Wathen: zachary.wathen003@topper.wku.edu

Date: December 1, 2017

**Table of Contents**

1. Executive Summary 1
2. Project Overview and Goals 1
3. Project Overview
4. Our Client
5. Preliminary Requirement Analysis 1
6. Business Considerations
7. General Objectives
8. Business Objectives
9. Project Scope 2
   1. Included Features
   2. Excluded Features
10. Benefits 3
    1. Information About Campus
    2. Archive of Information
11. Technical Resources (Hardware/Software) 4
    1. Hardware
    2. Software
12. Hardware Architecture Raspberry PI 5
13. Planning and Resources (The Group’s Approach to the Project) 5
    1. Meeting Times
    2. Process to be Followed
    3. Outline Plan
    4. Milestone 1
    5. Milestone 2
    6. Milestone 3
    7. Milestone 4
14. Documents and Presentations 8
    1. Weekly Reports
    2. Periodic Presentations
    3. Feasibility Study
    4. Development Study
    5. Milestone 3 Analysis
15. Technical Deliverables 9
    1. Database
    2. Product Features
    3. Source Code
    4. Docker Containers
16. Technical Feasibility 9
    1. Security
    2. Administrative Interface
    3. Account Privileges
17. Schedule Feasibility 10
    1. Estimated Time to Complete Each Task
    2. Milestone 2 Development Plan
    3. Milestone 3 Development Plan
    4. Milestone 3 Development Plan
    5. Activity Graph for Completing Tasks
    6. Activity Graph for Milestone 2
    7. Activity Graph for Milestone 3
    8. Activity Graph for Milestone 4
18. Project Visibility 20
    1. Communication
    2. Intermediate Deliverables and Presentations
19. Initial Risk Analysis 21
    1. Risk Analysis Update
    2. Emergent Risks/Problems
20. Risk Monitoring 24
21. Software Requirement Specification 24
    1. Purpose
    2. Definitions, Acronyms, and Abbreviations
22. General Product Description 25
    1. Product Perspective
    2. Product Functions
    3. Future Proposed Functions
    4. User Characteristics
    5. General Constraints
    6. Assumptions and Dependencies
23. User Interface Interactions Use Cases 27
    1. Flow of Events: Public User
    2. Flow of Events: Registered User
    3. Flow of Events: Admin User
24. Scenarios 28
    1. Scenario 1: User Creating an Account
    2. Scenario 2: User Login and Searches for a Page
    3. Scenario 3: The Register User Creates a Page
25. User Interface Requirements 30
26. Functional Requirements 31
    1. Final Sprint Requirements
    2. Process Requirements
27. Software Process Model 35
28. Nonfunctional Requirements 35
29. Software System Performance Requirements 36
    1. Subsystem Performance Objectives
    2. Subsystem Expectations
    3. Application Workload Mix
30. Bottlenecks 38
    1. Hardware
    2. Software
31. Security Requirements 39
32. Data Dictionary Design 40
33. Pseudocode Design 40
34. Class Diagrams Design 41
35. System Boundaries Design 41
    1. Physical Boundaries
    2. Logical Boundaries
36. Software Architecture Design 42
37. Software Engineering Performance Design 43
    1. Development Decisions
38. Security Design 43
39. Implementation 44
    1. Frontend: Creating the Interface
    2. Backend: Database Creation
    3. Backend: Adding Functionality
    4. Docker: Creating and Packaging the Server
40. Prototype Approach 46
    1. Plan
    2. Documentation
41. Security Implementation 46
    1. Hash Password for Registration
42. Verification and Validation (Review) 47
    1. Unit Testing
    2. Integration Testing
    3. System Testing
    4. Acceptance Testing
    5. Delivery
43. Software Performance Validation 51
    1. File I/O: Docker and Base OS Environment
    2. CPU: Docker and Base OS Environment
    3. Database Service: Base OS Environment
    4. Network
44. Security Verification 61
    1. Security Question
45. Operation and Maintenance (Release) 61
    1. Operation
    2. Maintenance
46. Requirement Conclusion 62
47. Appendix 62
    1. PI Setup
    2. Remote Access PI via PuTTY(SSH)
    3. Viewing Source Code Directly on PI
    4. Build Instructions
    5. Remote Login Instructions
    6. Docker Setup
    7. Github Setup
    8. Source Code

**Executive Summary**

The following report was created with the purpose of detailing the feasibility and requirements of cloning the Wikipedia website on a raspberry pi. A team of five students will be undergo the project with their primary client being Manh Do. The Wikipedia clone will adhere to the specifications of Mr. Do and include the core features of Wikipedia: web based development of content, editing of content, dynamic generation of new web content, and generation of individual user accounts. The theme of Wikipedia clone will be based on Western Kentucky University (WKU) and will include pages on the school’s history, athletics, buildings, etc. This report details the feasibility of completing the task within the time constraint of the semester, as well as the requirements associated with the project.

**Project Overview and Goals**

*Project Overview*

Wikipedia is an online encyclopedia with articles created and edited by volunteers around the world. Wikipedia describes itself as the free web-based encyclopedia that anyone can edit. The accessibility and ease of use of the website are integral elements of its growth and popularity. These elements will shape how each feature is developed for the final product. The goal is to create an accessible web-based encyclopedia. The mini WKU themed Wikipedia will be easy to use, which will attract users to create, edit, and explore articles.

*Our Client*

Our team will be working for a client, Mahn Do, who will direct our building of the mini Wiki from start to finish. Mr. Do will be briefed weekly on the team's development progress, and will be notified and inquired about all major decisions dealing with the Wiki clone project. The Wiki will be built to the specifications and liking of Mr. Do throughout the entire developmental process.

**Preliminary Requirement Analysis**

*Business Considerations*

The team will not have to plan an extensive budget for this project, but must consider some simple business matters. In order to initiate the project, the team must purchase a Raspberry Pi 3 and the Docker textbook. For one, purchasing the Raspberry Pi 3 was a cost divided amongst all the group since all team members will be equally dependent on it. On the other hand, only one team member needed to purchase the Docker textbook for the whole group, and this person will have the textbook to resell or keep after the end of this class. It is unlikely the team will need to make any additional purchases beyond these.

*General Objectives*

The main goal of this project is to create a scaled down version of Wikipedia (web based encyclopedia). The niche Wikipedia will provide information about key structures on WKU's campus such as: academic buildings, residential building, and resource building. Users will be able to add/edit pages and content on the wiki.

*Business Objective*

There are two different goals for our mini WKU Wiki. The primary goal is to provide students at WKU information about different buildings around campus. The Wiki will allow students to easily navigate to the resources they need. The Wiki will show the student resources available in specific buildings such as tutor labs, the financial aid office, computer labs, and counseling services. This would be an extremely useful tool for newer students at Western who are looking for specific things on campus but don't know where to find them. The second goal of the mini Wiki is to carry down the tradition/information about building from generation to generation, documenting information now that may one day be of historical interest. The public will be given the ability to learn about the culture of WKU through the information on the mini Wiki.

**Project Scope**

The task at hand is to create a smaller version of Wikipedia that is focused solely on providing information about Western Kentucky University. The product that our developmental team will be producing will give users the ability to search for, read, and edit articles pertaining to WKU. Unlike Wikipedia, this version of Wiki will be themed and limited to information about WKU, however, like Wikipedia, users will be able to edit page information and create new pages with new subjects, search pages and easily navigate the website to find specific information they make be looking for, and create login accounts from which to make these edits. We are creating this website for the purpose of helping new Western students learn more about the University, (about campus, resources, faculty, and more,) and to be a sort of easy compilation of WKU information for anyone else who may be interested, such as alumni or students searching for colleges.

During the last sprint, the features left to implement were the search bar function and adding the ability to edit already existing articles. Completing the features specified for the current sprint will fulfill all the included feature requirements decided upon by the development team and client. *Restoring Page Backups* has moved from being an included feature to excluded because of time constraint. A list below can be seen of all features included in the scope of the project as features from the official Wikipedia that will be excluded.

*Included Features*

User Accounts/Levels – There will be two different types of users (registered and public) in the niche Wiki. Users who are registered will be able to create new pages, edit/add information to previous pages. Public, can only search for an article, view an article, and register.

Page Creating/Editing – When using the mini wiki, users should be able to easily create and modify pages. There will be a simple way for users to add and edit pages from the website. There are no plans currently for an editor mode because of difficulty and time restraints, but there will be some way for users to do it from the page, and hopefully the mini WKU Wiki will support easy access for user to create and edit different pages.

Easy Search – Users will be able to search the wiki using keywords to find whatever pages it is they are looking for, making it easy to navigate the wiki. There will be a search bar visible on every page, which will allow the user to quickly find needed information and jump pages at any time. A list of pages may form as search results based on relevance by user's inputted keywords. This functionality will help the user quickly locate the desired information

*Excluded Features*

Feature Requests – Users for the mini WKU Wiki will not be able to request for additional features directly through the mini WKU Wiki. Furthermore, users will cannot provide positive or negative feedback about the functionality of the wiki through the website.

Donations Page – The mini WKU Wiki will not support a donation page. All three tiers of users (Admin, members, and public) will not have the ability to donate directly through the webpage.

Geo-notice – The user’s physical location (longitude and latitude of the user) will not be calculated. The mini wiki will not tell the user how close he/she is to a certain building on WKU’s campus. In addition, the user will not be able to pull up the longitude and latitude information on any of the buildings.

Video – All the user tier (admin, member, and public) will not have the ability to publish/upload video content to any of the webpages. Playback and support of the video content will not be available for the niche WKU wiki.

Languages – The mini WKU Wiki will not support any language besides English as the main webpage content; English will be the main language on the wiki. Users will not be able to translate pages to other languages (example: the wiki will not support two pages of the exact building in two different languages). However, the WKU Wiki will allow users to add foreign languages to pages (example: users can add the Chinese character of Confucius Institute’s wiki page).

Restoring Page Backups – When an admin level user sees that a member has continuously added incorrect information or deletes pertinent information, they may wish to pull a backup of the page before the rouge member made their bad edits. Therefore, there will be a function that backups and stores a page after each editing session, as well as a function to recover one of these backups if needed

**Benefits**

This project has the potential to be an extremely useful tool to an array of different people, from current WKU students wanting to learn about campus to someone who is considering attending WKU but first wants to learn more about Western. Really there are any number of reasons that someone might use our mini Wikipedia, meaning there are lots of benefits to creating it.

*Information About Campus*

For one, a mini WKU Wiki would be very helpful for finding different buildings and resources on campus. Having an easy access, easily navigable website to help you find where your advisor's office probably is or where different classes are would be a big tool for newer students who do not know much about the university yet. It could just as well be a handy tool for students who have been going to WKU for a while, wondering what's new on the ever-under-construction campus, or what time different on campus restaurants close, or when parking is free in certain areas. The benefits of having all of this information in one place and easy to search through are huge.

*Archive of Information*

This project will essentially be the one individual easy to navigate website with all the information anyone could be looking for about WKU athletics, clubs, faculty, locations, and more. Everything will be in one place, but organized in a way so that it will be easy to find what one is looking for. Also, this will give everyone an outlet to post more helpful information that may not be on the site already, allowing it to grow and be filled with useful information from a variety of sources. This eliminates the complex navigation of websites like the University's, with dozens more purposes and links, or jumping around then to a sports website to get statistics and information about athletics. The sole purpose of the wiki is to have an easily accessible mass of information, unlike other websites providing links and student resources, making it the best place to find what one is looking for quickly.

**Technical Resources (Hardware/Software)**

*Hardware*

Raspberry PI 3 – The workspace environment for this project is going to be ran on this device. This device will hold the source for our website in its entirety as well as all the backend database data. Using this device as well as a few other tools, we'll be able to remotely connect to the backend of our website from our personal devices, to make uploads and modifications.

*Software*

LAMP (Linux, Apache, MySQL and PHP) – This package is the core backbone of running our website as managing and connecting to our databases.

MySQL – We will be using MySQL Prepared statements to make queries to our database, these statements are secure as they prevent MySQL injection.

PHP – We are going to use PHP to handle the results of queries, and piece together HTML based on those results. We will also have PHP classes that will hold static variables that are needed across multiple files, such as DB\_USERNAME, DB\_PASSWORD as well as DB\_HOSTNAME. These names will never change, but will need to be frequently accessed from a connection.php script.

Bootstrap – This is a front-end component library. With this library, we will gain access to a responsive grid system that is capable of changing views based on the screen size of the visitor's device as well as many prebuilt components.

GitHub – We will use the GitHub web service to backup all documentation as well as the source code for the site itself.

phpMyAdmin – This is a back-end database interface that will allow us to easily access our MYSQL database to make any kind of modifications to the architecture of the database, as well as to make manual queries directly to the database.

Docker -Docker is a tool used to package applications as well as all of the libraries and dependencies needed to run the application inside something referred to as a container. Three Docker containers will be developed to contain the web interface, the database and the file repository. Docker will also be used as the packaging tool to deliver the final product.

Docker Cloud - Docker Cloud is a cloud registry to create and share Docker images. The repository can be shared with a team or open to the public. This is where the images for the project will be stored.

Hardware Architecture Raspberry Pi

The hardware of a raspberry pi 3 is relatively simple as its designed to be so. The PI's hardware components include:

HDMI Port – Used for displaying the system's graphical interface, as HDMI is one of the most common outputs for graphics.

Wifi – Built in wifi to allow for mobility of the pi itself, while maintaining the ability to run the server from any wifi service with the appropriate open ports.

Ethernet – The PI comes with a ethernet jack that allows for a more stable/consistent connection. This is optimal for running a server, as it prevents even more latency between the pi and the router over wifi.

Audio Jack – Comes with an audio jack, that may only be useful in certain mobile situations to not disturb others

SD Card Port – The PI relies on SD cards as its method of storage. SD cards are relatively quick in comparison to other storage methods, and are compact enough to function well with the mobility offered by a PI.

Micro USB Power – Relies on Micro USB for power, the universal availability of micro USB provides the PI with the ability of being powered from a multitude of different accessories.

USB Ports – The PI 3 comes with 4 available USB ports, these ports allow for the connection of a majority of USB devices, such as storage, keyboards and mice.

Quad Core Processor – Provides the small device with enough power to run most functions that are expected of the device.

The PI's hardware design was geared towards simplicity and mobility. You would not use a single PI to operate a traffic intensive service, but with this being a smaller project, the PI is more than what we require.

**Planning and Resources (The Group’s Approach to the Project)**

*Meeting Times*

For the first milestone, the team was divided into two subgroups. Group A is the front-end team and Group B is the back-end team. Each group will meet once weekly on their own in addition a meeting of all team members which will include both groups. Every Friday all team members will meet with the client for discussion and feedback as well as a virtual check in will occur at the end of each week to discuss progress, questions, and goals. The first chart below (Figure 1) is outlining the preliminary weekly meeting plan created during the first feasibility study.

|  |  |  |  |
| --- | --- | --- | --- |
| Meeting Time | Meeting Type | Group A | Group B |
| Monday 3-4pm | In Person | Yes | No |
| Tuesday 4-5:30pm | In Person | Yes | Yes |
| Wednesday 3-4pm | In Person | No | Yes |
| Friday 11:30-12:25pm | In Person  Client Meeting | Yes | Yes |
| Friday 3-4pm | Virtual | Yes | Yes |

**Figure 1** Time Table for Milestone 1

After the first milestone, a new weekly meeting plan was created to accommodate the changing schedules and needs of the team members as well as to increase productivity. Since this schedule works well for all the team members, it remains unchanged for milestone three. The number of virtual meetings in the new schedule has increased. Virtual meetings made it easier for groups to have meetings and increased communication. Meeting in person on campus was not always feasible as each team member have such varying schedules. The small group meeting times are flexible but they must either meet in person or virtually that day. A meeting of the whole team, in person, will still be held as scheduled before. Meeting in person at least twice the week is the goal. The updated meeting schedule can be seen below (Figure 2).

|  |  |  |  |
| --- | --- | --- | --- |
| Meeting Time | Meeting Type | Group A | Group B |
| Monday 6-8:30pm | In Person/Virtual | Yes | No |
| Tuesday 4-7:00pm | In Person | Yes | Yes |
| Wednesday  7-9:30pm | In Person/Virtual | No | Yes |
| Thursday 7-9:00pm | Virtual | Yes | Yes |
| Friday  11:30-12:25pm | In Person  Client Meeting | Yes | Yes |

**Figure 2** Time Table for Milestone 2 & 3

For this sprint, the team was divided into two new subgroups. One group is the testing team. They are in charge of creating and performing software tests. The other group is the development team and they are tasked to finish the development of the software. The subgroups are scheduled to meet separately on Mondays and Thursdays either in person on campus or virtually through GroupMe. On Tuesday, we will continue to have a full team meeting, in person on campus, as schedules for other milestones have specified. On Thursday nights, a short virtual meeting is held for members to check in with progress and questions. On Friday, the whole team is required to meet with our client Mahn Do. The updated meeting schedule can be seen below (Figure 3).

|  |  |  |  |
| --- | --- | --- | --- |
| Meeting Time | Meeting Type | Subgroup Meeting | Team Meeting |
| Monday 6-8:30pm | In Person/Virtual | Yes | No |
| Tuesday 4-7:00pm | In Person | No | Yes |
| Thursday 6-8:30pm | In Person/Virtual | Yes | No |
| Thursday 8:30-9pm | Virtual | No | Yes |
| Friday  11:30-12:25pm | In Person/Client Meeting | No | Yes |

**Figure 3** Time Table for Milestone 4

*Process to be Followed*

To complete this project, the development team decided on using an agile method. The development process will use concepts from both the iterative refinement approach as well as the incremental development approach to achieve completion. The project has been broken into four development phases or "sprints" with the Milestone dates marking the end of each sprint. During each developmental sprint, a set of features will be chosen to design and implement into a sprint specific prototype to present to the client. At the end of every sprint, the current prototype features will be reviewed and revised as the features to be added for the next sprint's prototype begins their development. The project manager, Michelle Schapmire, will be in charge of finalizing the sprint plans with the approval of the client.

*Outline Plan*

The project development plan is below, outlining the principal activities and milestones during different phases of the project. At each milestone, the client will be presented with the most current prototype of the website as well as a report on the development process for evaluation and feedback.

*Milestone 1 (September 8, 2017)*

Feasibility Study – An analysis on the feasibility of the project will be documented and presented to the client. This will include a preliminary requirements analysis, suggested deliverables, an outline of the development process, a visibility plan, a risk analysis, as well as any other technical requirements.

Principal Activities – The selection and setup of the tools to be utilized during the project will be completed including the development environment on the raspberry pi, a GitHub repository for technical files and data, a One Drive folder for documentation storage, as well as the setup of other libraries or frameworks needed to build the project.

*Milestone 2 (October 4, 2017)*

Development Study– A formal report will be generated providing information on the development process up to the second milestone. This report will include an updated feasibility analysis, an updated development plan, a requirements specification, and details about the steps taken during this phase of development.

Principal Activities – An initial prototype is to be created to present to the client for criticism and feedback. This prototype will include a homepage, an article page mock up, a preliminary database, along with user registration and login functionality. The first Docker container holding the PHP-Apache image will be set up. Furthermore, the documentation from Milestone 1 will be expanded to 30 pages to include the required documentation on the requirements; requirement documentation entails details on the development aspect of the project.

*Milestone 3 (November 3, 2017)*

Milestone 3 Report – A formal report on the progress since milestone 2 will be documented and presented to the client. This will include any changes made to the prototype to meet the requirements and feedback from the client, what still needs to be completed and an updated schedule. This report will also contain a report on security requirements and features as well as documentation pertaining to software performance engineering.

Principal Activities – A revised prototype is to be completed to present to the client for further feedback. This will include a revised homepage and article page with the addition of functionality for users to create and edit articles. An administrative page will be created to view how many users are registered and the most popular page on the website. Tests will be ran on the subsystem and software to test its performance.

*Milestone 4 (December 1, 2017)*

Final Report – A handover package will be created to be given to the client. This will include all information from previous reports as well as information on the progress since milestone 3 and finalized information about the project's completion. Furthermore, the mini WKU Wiki will be delivered to the client.

Principal Activities – The search for an article feature is to be completed and added to the final product as well as the edit page feature not completed last milestone. A final working prototype is to be completed two weeks before this milestone for final testing and revisions.

**Documents and Presentations**

*Weekly Reports*

Each week a report on what has been accomplished that week as well as goals for the following will be delivered to the client.

*Periodic Presentations*

During development process, a demonstration on the current working features of the product will be presented to the client as new features are added to the working prototype.

*Feasibility Study*

A formal document will be prepared including a preliminary requirements analysis, suggested deliverables, an outline of the development process, a visibility plan, a risk analysis, as well as any other technical requirements.

Based on the information our team has collected about the feasibility of developing and running a Wikipedia clone on a Raspberry Pi 3, our team finds that it is indeed feasible in respect to the amount of time our team members have to contribute to the development of this project, the technical requirements to run the server, as well as the amount of knowledge we possess and ability to learn as a team. The conclusion of this study is that we're willing and able to complete this project in a timely, well organized and professional manner.

*Development Study*

A formal report will be generated providing information on the development process up to the second milestone. This report will include an updated feasibility analysis, an updated development plan, a requirements specification, and details about the steps taken during this phase of development.

Milestone 3 Analysis

A formal report will be prepared providing information on the development process up to the third milestone as well as a security analysis and a software performance analysis. This document will include sections from past documentation as well as updates to the sections.

**Technical Deliverables**

*Database*

A database schema will be given that demonstrates the design and how everything is interconnected and stored in the database. The schema will entail the use of UML diagrams.

*Product Features*

User account registration and login, administrative user privileges, manipulation of information directly through the web browser, creation of new articles, editing of existing articles, and an article search engine will all be added to the final product.

*Source Code*

Source code will be provided throughout the development of the project in order to demonstrate progress as well as implementation. Comments will be made throughout the development process to document the functionalities of the code implementation.

*Docker Containers*

Docker containers will be built containing all of the files as well as the environment needed to run files. There will be three different containers. One will contain the web interface, another will contain the database, and the other will hold the file repository. The final containers will be linked together and delivered to the Client.

**Technical Feasibility**

In all reality, the raspberry pi 3 is an inexpensive route to take to grant a group the ability to host a small server. The pi isn't capable of running a massive number of queries due to its memory allocation limitations. The network the pi is being hosted on is also not meant to withstand a major amount of web traffic, it's running on a home Wi-Fi. Given these limitations, the pi is still feasible to run a small class assignment on. The pi, as well as the internet it's being hosted on, is able to maintain connections to the members of the group, as well as store the small amount of data that will be required. A few things to consider in the aspect of technical feasibility:

*Securit*y

In order to maintain a secure environment, we must secure the data we store as well as prevent unauthorized access to the data that is stored. In the aspect of securing the currently stored data that may be deemed as sensitive information (email, password), we can use simple hashing methods. In terms of hashing, we will consider looking into SHA1 hashing, but may just stick with RIPEMD-160 hashing as the security required for this project isn't too extensive. To prevent un-hashed data that is stored in the database from being illegitimately obtained via MySQL injection, we will use PHP prepared statements. This allows us to check the post variables incoming to the script and bind them to the SQL statement. Also, it's important to note that we decided to pass data to PHP files via POST method rather than the GET method which would display the desired variable name in the URL, which could allow a user to flood the database.

*Administrative Interface*

Given the project is going to be for the most part accessed by the group members and our client, we expect that pages created by the users will be appropriate and not contain any kinds of profane content. Even though, the page will be accessible to the public, so we will need to have an interface that allows an admin user with special privileges to remove/edit pages regardless if they're the page's creator.

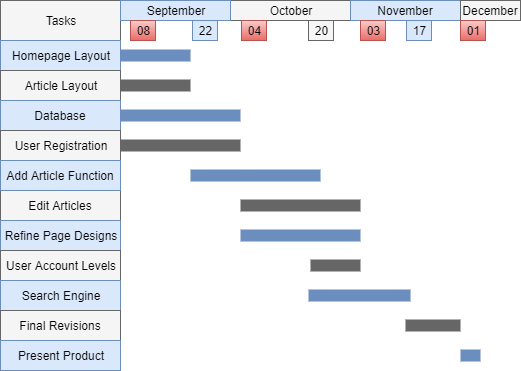
*Account Privileges*

We will need to establish multiple account security clearance levels that allow special users to make administrative decisions in the aspects of a page's validity to be posted on our site.

**Schedule Feasibility**

*Estimated Time to Complete Each Task*

The following chart (Figure 4) illustrates the schedule for the development of each feature to be included in the final product. Each bar represents the expected dates and duration that the corresponding task will take to complete. The dates in red represent each milestone and all other outlined dates line up with client meetings. At the beginning of every milestone a new phase specific Gantt chart will be created specifying the schedule of tasks to be completed to reach the goals required by the next milestone.



**Figure 4** Schedule Table

*Milestone 2 Development Plan*

To complete the goals laid out for Milestone 2, the requirements were broken down into a series of tasks. The process closely follows the development plan laid out in the *Planning and Resources* section in the Feasibility Report from the first milestone. Each task was assigned a start date and a specific number of days to complete the tasks. These values were laid out into the table in excel as seen below. This table was also used as way to increase progress visibility. Rows are added to the table to track the completion of the tasks. These rows include days completed, days remaining, and the percentage of completion. After every team meeting, both in person and virtual, the last three rows of the chart are to be updated based on the progress that has been made. The tasks and plan of this Milestone can be seen below as well as in the Gantt chart following the schedule table. The tasks are split into two groupings. The first group are system tasks. The second group, which starts with the "define requirements for current sprint" tasks, are the deliverables needed for Milestone 2 (grouping two is denoted by an “\*”) (Figure 5).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Task Name | Start Date (2017) | End Date (2017) | Duration (Days) | Days Complete | Days Remaining | Percent Complete |
| Get pi Remotely Accessible | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Preliminary Database Schema | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Login Function Design | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Registration Function Design | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Home Page Design | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Registration Page Design | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Set Up Docker | Sept 15 | Sept 22 | 7 | 7.00 | 0.00 | 100% |
| Create Home Page Interface | Sept 15 | Sept 22 | 7 | 7.00 | 0.00 | 100% |
| Create Registration Page Interface | Sept 15 | Sept 22 | 7 | 7.00 | 0.00 | 100% |
| Create User Tables in mySQL | Sept 15 | Sept 22 | 7 | 7.00 | 0.00 | 100% |
| Add Login Functionality | Sept 22 | Sept 29 | 7 | 7.00 | 0.00 | 100% |
| Add Registration Functionality | Sept 22 | Sept 29 | 7 | 7.00 | 0.00 | 100% |
| Research Create Page Feature | Sept 29 | Oct 4 | 5 | 5.00 | 0.00 | 100% |
| Define Requirements for Current Sprint\* | Sept 8 | Sept 15 | 7 | 7.00 | 0.00 | 100% |
| Create Models for the System Requirements\* | Sept 15 | Sept 25 | 10 | 10.00 | 0.00 | 100% |
| Create Milestone 2 Report Outline\* | Sept 15 | Sept 20 | 5 | 5.00 | 0.00 | 100% |
| Create Milestone 2 Report Draft\* | Sept 20 | Sept 27 | 7 | 7.00 | 0.00 | 100% |
| Create Final Milestone 2 Report\* | Sept 27 | Oct 3 | 6 | 6.00 | 0.00 | 100% |
| Create Presentation for Client\* | Sept 27 | Oct 3 | 6 | 6.00 | 0.00 | 100% |
| Deliver Report and Presentation to Client\* | Oct 4 | Oct 5 | 1 | 1.00 | 0.00 | 100% |

**Figure 5** Milestone 2 Schedule Table

The Gantt chart below is generated on Excel based on the values in the table shown above. This provided a visual guide to track progress as the bars colors would fill with a darker color as the percent complete column increased. A version of the Gantt chart with an example of some incomplete tasks is shown below (Figure 6). By using the Gantt chart, the members in team two were able to plan out and keep track of their progress. This helped team two stay on track and fulfil the desired requirements for milestone two. The Gantt chart is an excellent tool for team two to visualize their deadlines as well as their goals. As one can summarize from the Gantt chart, the main goal of milestone two is to implement the homepage interface as well as create the foundation for the database.

**Figure 6** Milestone 2 Gantt Chart

*Milestone 3 Development Plan*

To complete the goals laid out for Milestone 3, the requirements were broken down into a series of tasks. The process closely follows the development plan laid out in the *Planning and Resources* section in the Feasibility Report from the first milestone. There is a deviation from the plan that occurred because of issues that arose from the Raspberry Pi. Also, the team and client decided to push the development of the admin level user features to the next milestone after the search function is developed. These issues will be discussed in the *Risk Management* sections of this document. Each task was assigned a start date and a specific number of days to complete the tasks and entered into an excel chart similar to the one above (Figure 6). A Gantt chart is then generated based on the values in the excel chart. Below is the development plan for Milestone 3 to complete the requirements agreed upon by the client and team (Figure 7).

**Figure 7** Milestone 3 Gantt Chart with Delayed Schedule

*Milestone 4 Development Plan*

The schedule of tasks to be completed during this final milestone can be found below (Figure 8). This schedule varies from the development plan laid out in the *Planning and Resources* from Milestone 1. The edit page function had to be pushed back to after we implement the search function. The administrative user features will not be implemented due to time constraints.

**Figure 8** Milestone 4 Gantt Chart

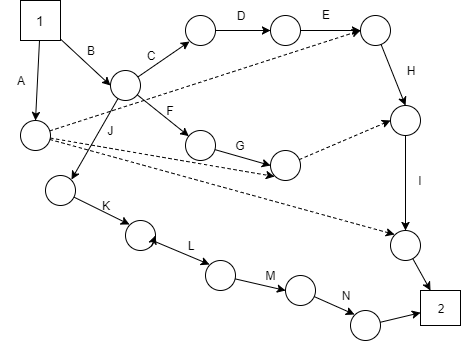
*Activity Graph for Completing Ta*sks

An activity graph is used to estimate the flow of the development process. They are used to estimate when tasks can begin and when they can be completed. It is a useful tool to visually layout which tasks can be completed in parallel or completed sequentially. There are 4 symbols used in an activity chart. The squares represent a milestone. The square labelled 1 represents the first milestone and the square labelled 2 represents the second milestone. The solid arrow lines represent a task and the dotted arrow lines represent a task dependency. The circle marks events and can be thought of the task completion. A table is provided with each activity graph with as a key for the activity graph. The following activity graphs will provide the key first and then the graph itself (Figure 9-14).

*Activity Graph for Milestone 2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Label | Description | Personnel | Required Resources | Task dependency | Hours Est. |
| *A* | Set up remote access on the pi | Zach | Raspberry Pi  SSH access  PutTy |  | 4 |
| *B* | Define requirements for sprint | Team  Client |  |  | 2 |
| *C* | Plan Timeline | Michelle | Team input | B | 2 |
| *D* | Design home page and registration page | Peter  Steven |  |  | 5 |
| *E* | Create home page and registration page | Peter  Steven | Html, CSS, JavaScript, bootstrap | A, D | 10 |
| *F* | Create a database design | Daniel  Zach |  |  | 2 |
| *G* | Create database and accounts table | Daniel | Phpmyadmin,  MySQL | A | 2 |
| *H* | Add login and registration functionality | Zach  Steven | PHP, Apache, MySQL, JavaScript | D, G | 10 |
| *I* | Set up Docker and package the web interface | Michelle | Docker,  Docker Cloud,  SSH | A, H | 10 |
| *J* | Create models for system requirements | Michelle  Daniel |  | B | 3 |
| *K* | Create outline for milestone 2 documentation | Peter  Michelle |  |  | 2 |
| *L* | Create 25-page draft of documentation | Team |  | K | 10 |
| *M* | Create final report for milestone 2 | Team |  | L | 8 |
| *N* | Create presentation | Team |  | M | 2 |

**Figure 9** Activity Key Chart

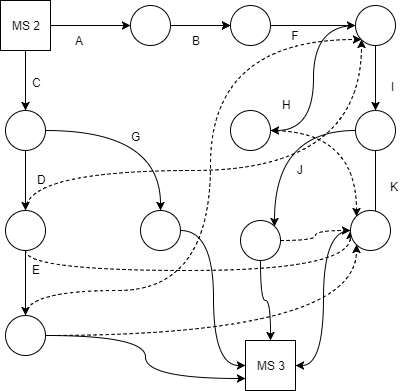


**Figure 10** Activity Flow Chart

*Activity Graph for Milestone 3*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Label | Description | Personnel | Required Resources | Task Dependency | Hours Est. |
| A | Link database container to webserver | Michelle | Docker, PutTy |  | 3 |
| B | Define performance/security requirements | Michelle, Steven, Daniel |  |  | 2 |
| C | Design Create/Edit Page function | Zach, Peter, Steven |  |  | 3 |
| D | Implement Create Page | Zach, Daniel, Peter | Apache, MySQL, PHP, Bootstrap | C | 10 |
| E | Implement Edit Page | Zach, Peter | Apache, MySQL, PHP, Bootstrap | D | 6 |
| F | Run performance tests | Michelle | Docker, MySQL, Raspbian | D, E | 5 |
| G | Design Article Page | Peter | Bootstrap |  | 3 |
| H | Create Admin Page | Peter, Zach, Daniel |  |  | 6 |
| I | Create milestone 3 draft | Everyone |  | B, F | 8 |
| J | Create final report | Everyone |  | I | 6 |
| K | Create presentation for client | Everyone |  | J, D, E, H | 4 |

**Figure 11** Milestone 3 Activity Key Chart

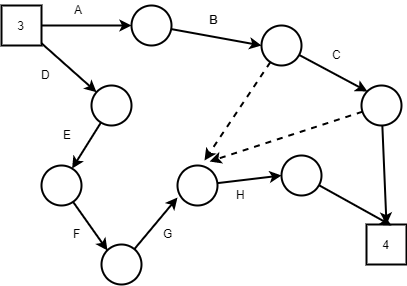


**Figure 12** Milestone 3 Activity Graph

*Activity Graph for Milestone 4*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Label | Description | Personnel | Required Resources | Task Dependency | Hours Est. |
| A | Design the search function | Zach, Peter, Michelle |  |  | 3 |
| B | Implement the search function | Zach, Peter | Apache, MySQL, PHP, Bootstrap | A | 8 |
| C | Implement the edit page function | Zach | Apache, MySQL, PHP, Bootstrap |  | 4 |
| D | Define reliability requirements | Daniel, Steven |  |  | 2 |
| E | Create software tests | Daniel, Steven |  |  | 3 |
| F | Create final report draft | Everyone |  | D, E | 4 |
| G | Run software tests | Daniel, Steven |  | B, C | 7 |
| H | Create final report and presentation | Everyone |  |  | 5 |

**Figure 13** Milestone 4 Activity Key Chart



**Figure 14** Milestone 4 Activity Graph

***Project Visibility***

The team will attempt to make the progress of the development as visible as possible in order to ensure that It meets the specifications provided by the client. If there are any parts that do not adhere to the client’s parameters, they can be caught early and corrected accordingly. To make certain that the developmental process is visible the team will utilize the various methods described below.

*Communication*

Client - Communication with the client is vital to the development process in order to make sure that the team is on the right tracks to completing the project on time and to the specification of the client. Weekly meetings with the client will be attended every Friday to discuss progress and for the purpose of feedback. Communication through email will also be utilized when needed for any questions or concerns that the team or client may have.

Team - The team itself will also schedule and attend weekly meetings to make sure that every member is caught up on what needs to be finished for the week and to discuss/brainstorm any ideas they may have that may be beneficial. Apart from face to face meetings the team will also have a constant line of communication available through Group-me for the purpose of general questions or comments about the project. Furthermore, the team will use OneDrive to share Word documents, images, video and PowerPoint presentations. The different tools that the team will uses will allow them to communicate effectively and efficiently.

*Intermediate Deliverables and Presentations*

Demonstrations – Live demonstrations will be shown to the client during the weekly meetings as deemed necessary to show progress and receive feedback. Furthermore, the team members will record video demonstration for the weekly client meeting. The video demonstrations will be used if the live functions of the project does not load or is not in a stable state.

Presentations – Slideshow presentations will be presented periodically to show progress and design/screen layout. The PowerPoint slides will help the team communicate efficiently and effectively with the client by confirming his request. Presentations will allow the client to see what the team has accomplished. In addition, the slides will give the client a clear idea if his request are fulfilled accordingly.

Client Presentations– Documentation containing weekly progress will be turned into the client every Friday. This will include what the team accomplished, how they accomplished it, and what they did not accomplish that they had planned to. These will allow the client to keep track of the team's weekly progress.

**Initial Risk Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Category | Risk Level  (based on loss) | Likelihood | Impact |
| Lack of Time | Known-Known | Medium | Unlikely | Missed delivery dates |
| Lack of Experience | Known-Known | Low | Likely | Software Quality will not be good |
| New to Collaborative Development | Known-Known | Medium | Likely | Missed delivery dates |
| Loss of a Team Member | Unknown-Known | High | Unlikely | Development will slow down |
| Poor Communication | Unknown-Known | Medium | Likely | Confusion on task goals |
| Pi Failure | Known-unknown | High | Unlikely | Loss of progress |

**Figure 15** Risk Analysis

There are a number of risks for our team to consider when taking on this project, as with any large project. In order to plan effectively and ensure this project's success, we must take a good, realistic look at the risks involved and the possible difficulties we may encounter along the way. We quickly discovered there are multiple things that may hinder us a bit in the coming weeks, but have done our best to create a list and a game plan for how we will overcome or work around these problems (Figure 15).

Lack of Time –Our most prominent issue will likely be time. Each member of our team are full time students and many are working and/or involved in extracurricular activities, so it makes sense that we will be very busy both individually and as a group working on this project. Thus far, our team has effectively planned meeting times and divided work up amongst ourselves in such a way that we should have time to build what our client is looking for in the time allotted. However, if in the coming weeks we are to find that we are behind our desired schedule, we will be ready to assemble and make changes accordingly. This may mean modified or extra meeting times, rearranging team member's assignments to assist others (for example; if the back-end developers are on time and the front-end developer is behind, one back-end developer may switch to help), or cutting down on the more non-essential functions or graphics of our project to ensure completion overall, even if not to the quality we had originally wanted.

Lack of experience -Another issue to consider is our team's overall experience. There is an imbalance within the team when it comes to knowledge of/experience with different programming languages and the web design process. However, each member of our team is very competent and smart, meaning each of us can learn quickly and understand the concepts behind everything we are doing. Every member of the team has a fairly strong background in computer science as most are 3rd year CS students or beyond, so learning and working with new things is fun and desirable. Also, after gauging one another's limitations and specialties, it appears that where one member's experience falls short, another member's skills pick up there, so the imbalance works quite well and our team is organized accordingly. Nonetheless, it is sensible to expect that as a team we may still have problems along the way with our shortcomings in experience, so we have devised some little plans to handle such an occurrence. For one, we will attempt to learn from an outside source how to solve a problem that none of our group has prior experience with. This may mean searching the internet and watching tutorials, or consulting someone we know who does have experience, such as a CS professor, or the CS tutor in the tutoring lab. Also, if needed we may get together and decide we need to make a change to our overall plan or go about solving a certain problem in a completely different way.

New to Collaborative Development - Our team is completely new to collaborative web development. Working as a team throughout the entire semester may prove difficult, and as pretty much all of us are used to building exactly the projects we want exactly the way we want to do them, working in a team will mean possibly having disagreements and therefore limitations to just having our usual full creative liberties. Our team will have to learn to work together to be able to settle and negotiate disagreements, even over simple additions, and make little sacrifices of effort and time and ideas to accommodate the team's goal. The key to success on the project is effective collaboration inside our development team, which we realize, and are confident we can make happen.

Loss of a Team member *–* If we were to lose a team member our development process would suffer greatly. Since we have already divided up roles in such a way that everyone in our group has a main task that they focus on, the loss of a group member would create a hole that would need to be filled. Every member in our group would have to contribute to the missing team members role in order to complete the project. This would increase our work load and lead to a slower development process. However, this risk is unlikely to happen since we have already passed the date to drop out of the class and our every member of the group seems to be invested in the class.

Poor Communication –Poor communication among our team would result in the confusion of our goals. If we do not clearly indicate what needs to be accomplished and who is responsible for it, the task may not be completed or it may be completed twice which would just waste our time. In order to avoid such situations, we will always have a line of communication available through GroupMe so that anyone in our group can ask questions if they are unsure of their responsibilities or if they need help with completing their task. We have also scheduled face to face meetings to better communicate our progress, divide tasks, and answer any questions that we may have about our project.

*Risk Analysis Update*

As of right now the biggest risk being faced by our team appears to be our lack of experience with the software we are using, in particular docker. Currently we have the web interface in a container but are unsure where to go from there. We will have to do more research on docker to make sure that we know what we are doing and prevent this from becoming a larger problem.

*Emergent Risks/Problems*

Remote Connection to Pi - A risk that our team did not initially predict involved the actual hardware itself. In order to establish remote access to the pi a better router had to be purchased. Because of this, a remote connection could not be established for about two weeks. This set us back slightly but we were able to recover due to our time allocations and management.

Pi Resets - Another potentially threatening risk involves connecting to the pi, at times a connection cannot be established and the pi has to be reset in order to fix this. Other times the pi's password will change for no apparent reason. We are currently unaware of what is causing this so it could potentially set us back a bit if we do not figure out the reason behind this and fix it. In the event of similar mishaps our team should consider allotting more time for debugging and testing to prevent occurrences like this from happening.

SD Card Corruption - A major risk involving the hardware occurred in which the SD card became corrupted, we had to get a new card and reinstall everything that was on the old one. All of our files were backed up on git hub so this only delayed us about a day as it did take some time to reinstall everything. If we had not had all of our files backed up this could have potentially been a large setback that would have resulted in recreating all the lost html, php, CSS, and JavaScript files.

As for the risks, we initially predicted none have really been a major threat though we are new to the collaboration process our team works really well together and our time management has helped overcome any risks that would result in missed delivery dates.

**Risk Monitoring**

Since our team has identified the risks associated with the project, we are prepared to take action to minimize the threat of any of these risks from potentially causing the team any difficulty with completing our set goals. The milestones that we have set will allow us to keep track of our progress and help prevent any risks that may arise as a result of time management. Making our project as visible as possible by providing documentation, giving presentations, supplying source code, and through meetings with the client will allow us to minimize the any functionality risks, that is, risks having to do with how the system works. If the client does not like how we implemented or how something works then we have not accomplished our goal. If the client is able to see as much of our development process as possible, he will be able to notify us if we do anything that he does not like. Resource risks (risks involving the technologies we use) may arise due to lack of money, fortunately for us every member in our group has laptop available and we have already purchased a raspberry pi. We are also using software which does not cost us anything to keep costs at a minimum. There is also always the probability of hardware failure, system crashes, bugs in the code as well as the fact that our Wikipedia may work slightly different depending on the internet browser that is being used. To minimize the damage these risks may cause we will have to test our project frequently in order to catch any bugs or mistakes that we have made and fix them accordingly.

In the weekly meetings, the team will consider risks that may occur or subside. In addition, the team will concentrate on how to prevent risks from occurring. Furthermore, the team will meet with the client to discuss how different risks may negatively impact the features that the client has requested.

 **Software Requirements Specification**

*Purpose*

The purpose of the requirement specification is to address the requirements associated with development. To meet the client’s requirements, team two will use the software requirements specification to address functional and nonfunctional requirements associated with the project. In the end, the client will be delivered a project developed by the required specification.

*Definitions, Acronyms, and Abbreviations*

This subsection provides the definitions of all terms, acronyms, and abbreviation required to interpret the SRS. Additional information about the topics may be referenced in the appendix in the SRS or by with other documents.

WKU – Western Kentucky University, a university located in Bowling Green, KY which is the central topic of our version of Wikipedia from which all the pages created will follow the theme of.

LAMP - Linux, Apache, MySQL, and PHP... A software bundle that will be used by team two to develop the mini WKU wiki. Functionalities include, server, scripting functionalities, and database.

Docker– An opened platform that allows the team members to work alongside each other without having to worry about cross-platform problems associated with the project development.

Github – A repository that allows the team members to commit their work. This repository will back up the software system.

**General Product Description**

Product Perspective

To put the mini WKU Wiki product into perspective, it would be beneficial to first look to look at similar projects. The WKU Wiki is based off of the format and functionality of the popular online encyclopedia, Wikipedia. This product, however, will be much smaller than Wikipedia and built for a miniscule number of users compared to the target audience of Wikipedia. The WKU Wiki will have similar format to Wikipedia, and the search function, layout, page creation function, and purpose will be basically the same, but on a smaller scale.

While Wikipedia is built to provide a huge amount of information about all subjects to anyone who wants to use it, the WKU Wiki will provide a large deal of information strictly about Western Kentucky University and subjects within that general theme. The WKU Wiki product will still be a compilation of information with the purpose of providing users the ability to search through and read it, as well as a hub for users to collaborate and add new information where the Wiki leaves off. Simply put, the WKU Wiki will be a smaller scale, Western themed, version of Wikipedia.

Product Functions

Current Sprint Functions – Currently, the mini WKU Wiki has a home page with a nonfunctional search option and a nearly working log in and sign up function. Our team is in the final stages of implementing the sign up and log in functions, of which the pages and database for are already created. The sign up will allow users to create a password protected account that they will later be able to use the log in function to access with their specific credentials, and use the Wiki under that log in.

Future Proposed Functions – In the future, the WKU Wiki will have several user functions that are not yet implemented in this current version of the Wiki. For one, there will be a search function allowing users to easily scry through the different pages and quickly find what they are searching for more exactly by using keywords. Secondly, the WKU Wiki will have different user account levels, ranging from visitor to admin ranks; visitors will be able to search through the pages and view all the information, regular users will be able to edit and add pages on the Wiki, and admin level users will be able to roll back the changes made by regular users if necessary and confiscate their editing privileges should an abusive user be discovered. Finally, the Wiki will have a roll back function for admins, which allow old versions of pages to be reused should a more recent version have "bad" edits made, and there will be add and edit page functions available for all non-visitor level users.

User Characteristics

The target audience and therefore expected users of this product will be present and upcoming WKU students, faculty, staff, and alumni. Hopefully, the WKU Wiki will be a useful tool for anyone who wishes to learn more about the University for any reason, whether it be upcoming students who want to find out about the location or purpose of the different resources and offices available to them, or a WKU alumnus who wants to look back and see how much has developed and changed at WKU since they graduated. The users of this product will vary in age, though most likely there will be few, if any, users under the age of 16 with any reason to use the WKU Wiki.

Bearing this expected audience in mind, the WKU Wiki should be created in such a way that any user with some experience using the internet should be able to easily navigate the website and be able to find any information they might be looking for. For the regular user, who can create and edit pages, the process should remain simple to do these operations in order for the Wiki to grow and flourish with the collaborative effort of many users. The only users which may be regarded with a higher standard of expectations would be the admin level users. Admins should probably have more than the average experience with using the WKU Wiki and computer systems, since an admin level user's mistake could actually be extremely detrimental and destructive to the website.

General Constraints

The largest constraint at the moment with developing our product is time, as each member of the development team is a full-time student and a few are also part-time workers, with lots of other extra-curricular activities amongst them. The time deadline for this project is the end of the semester, and with several other homework projects and extra-curricular commitments also on the plate of the development team's members, the time constraints are a definite risk to consider when working on the Wiki.

Another constraint with the WKU Wiki project is the limitations of the Raspberry PI system from which our website is presently hosted. The PI should have enough RAM and storage space to run the website and be accessed by one to three people at a time, but anything truly extensive will be too heavy for the PI to handle, so this must be kept in mind while implementing different functions. For example, creating a search function with a good runtime and light complexity are important, since the PI may not handle several linear searches back to back very well when there is eventually a significant amount of content pages added to search through.

*Assumptions and Dependencies*

As of now, the project is highly dependent on a few things. One thing being that the Raspberry PI will be our functional hardware that will host our WKU Wiki, and it will have enough space and memory to do so. Should there be a problem with the PI or some unexpected restraints due to the chosen hardware, changes will need to be made to either the hardware or software. Our WKU Wiki will also be developed with the assumption that all the needed development software we are using will be compatible with the Linux OS on the Raspberry PI. Some smaller things we will be dependent upon are a good internet connection to access and develop the Wiki on the Raspberry PI, bug-less functionality with the programming languages we are using (meaning the language itself must not have bugs, even if what we code may), and each other, meaning the other members of the team, to do their designated part in a timely manner.

**User Interface Interactions Use Case**

The use case diagram below depicts three actors the public user, the Registered user, and the Admin user. The lines connecting the actors to the use cases indicate which function is associated with which actor. Here the public user is able to register, search for an article and read an article. The Registered user is able to login, search for an article, read an article, create an article, and edit an article. Finally, the Admin user is able to do everything the Registered user is able to do as well as recover backup content and remove articles. The flow of events can be seen below.

*Flow of Events: Public User*

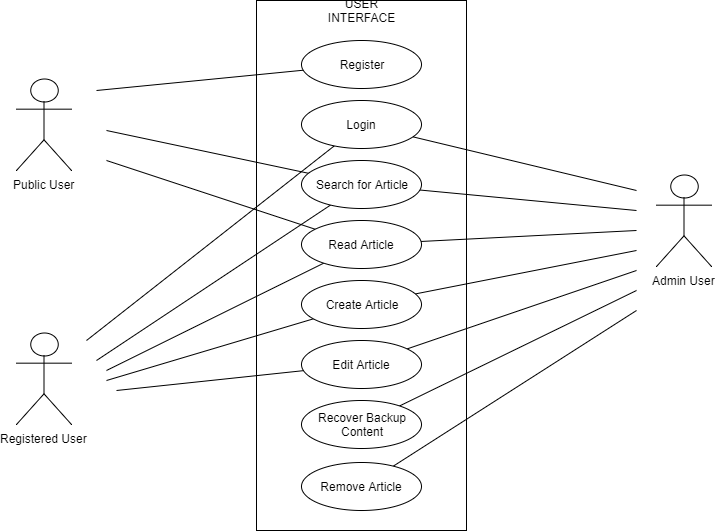
The public user connects to the user interface. The public user enters a search query into a search bar to find an article. The article page is then loaded into the user interface. The public user can also register by selecting the sign up option on the interface. This brings them to the registration page in the interface. From here, they can input their information and become registered users.

*Flow of Events: Registered User*

The registered user connects to the user interface and logs in. The user now has the option to create an article page. The option is selected in the interface and they are brought to the article creation user interface. They can also search for articles as public users can. From there, they can choose to edit the article.

*Flow of Events: Admin User*

The admin user will follow the same flow of events as the registered user. One difference is once they find an article with changes they find is incorrect or misleading, they have the option to recover back up content or remove the article all together through the user interface (Figure 16).



**Figure 16** Use Case

**Scenarios**

*Scenario 1: User Creating an Account*

Initial Assumptions – The user creating the account is not already logged in. The user creating the account is using a computer with an internet connection.

Normal – The user chooses to create an account. From the home page the user clicks on the drop-down menu button in the top right corner, this bring down some options one of them being sign up. The user clicks on the sign-up option and a form appears in which the user enters the appropriate information. Once the user has filled out the form, he/she clicks submit.

What Can Go Wrong – The user may try to create an account with an email that is already in use. If this is the case then the system will not except the information and will display a message informing the user that the email they entered is already in use.

The user did not completely fill out the form but still attempted to submit it. In this case the system will display a message informing the user that there is information missing on the form and the user will be able to enter the missing information or change the information he/she previously had.

System state on completion – The user is now logged in and his information has been stored in the database. The user can now create pages and edit articles once they are logged in.

*Scenario 2: User Logins and Searches for a Page*

Initial Assumptions – The user has already successfully created an account and is using a computer with internet access.

Normal – At the home page the user clicks on the drop-down menu at the top right of the screen. This will display a small menu in which the user can enter his/her email and password in order to login. Once the user has entered his/her information they will then click the login button right below the form in which they entered their information. The user will now be at the home page and will enter their search terms in the search bar. Once they press enter a list of the titles of articles will appear based on the keywords. The user will then click on the article that he/she wishes to go to.

What Can Go Wrong – The user's information did not match any information in the database. In this case the system will display a message notifying the user that the information he/she entered did not match any in the database. The user will then be able to enter different information or create a new account.

The users search terms are too obscure/incoherent to result in any article being found. In this case the system will just display a message stating that no results were found as well as some advice to try different search terms.

System State on Completion – The user is still logged in nothing else has changed.

*Scenario 3: The Register User Creates a Page*

Initial Assumptions – The user has already successfully created an account and is using a computer with internet access.

Normal – From the home page the user clicks on the drop-down menu at the top right of the screen. Here the user will see a button named "Create Article" once the user clicks on this button they will be redirected to a different page in which the user will be able to enter general information about the article in question. The user will then submit this information. The user will then be redirected to a page where he can directly add information such as text, images, and links. Once the user is satisfied with the his/her article they will click a save button.

What can go wrong – The user attempts to create a page with a title that is already in use. In this case the system will ask the user to be more specific/add a subcategory for the title to distinguish the article from others. The system will also display other articles with the same name as the one the user was trying to create so that the user can see the if he is attempting to make an article that has already been created.

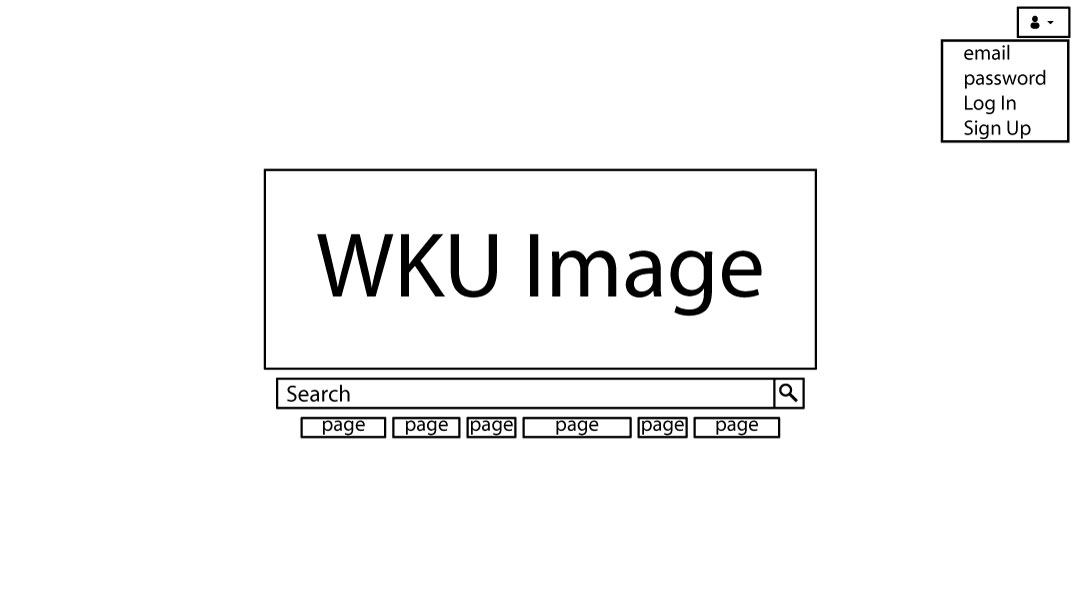
The user attempts to submit an incomplete form while entering general information about the article that is being created. If this happens the system will not accept the information and will prompt the user to fill out all required fields in order to proceed.

System State on Completion – The user will still be logged in and the information about the page and its creation will be stored in the database.

**User Interface Requirements**

Nonregistered users are those who have not created an account, registered users are those who have created an account and are logged in, and administrators will consist of our development team. Each of these users will have access to two types of pages the Main Page which will consist of a search bar and a drop-down menu from here users will be able to search for a page, create an account, or log in. The other page will be the article page these pages will have the information about the subject itself. From the home page, users will be able to login, create an account, or search for articles. These are the user interface requirements for the homepage.

Figure 17 is an example of defining the user interface requirements. for the landing page. The figure is used to help define the user interface requirements.

****

**Figure 17** Homepage Wireframe Interface

There are diverse types of user interface in team two’s project. The most popular interface is the homepage interface. A wireframe was created to help define the user interface requirements (Figure 17). The user interface requirement for the homepage needs allow the user to login/logout, create a new account, search, and explore. Furthermore, the interface needs to be clean and easy to use. Using the wireframe, team two was able to fulfill each of these requirements. The login/logout and create a new account feature is located at the top left-hand corner of the page stored under the user icon. By storing the components under the icon allows make the interface consolidated. Furthermore, the search function (the most important function) is located at the center of the page to draw the user’s attention. Without having to try to find the main feature of the page, the search page is centrally located, thus provides easy access. Lastly, in the wireframe, under the search box, there are buttons that shows the most popular pages to encourage the user to explore and learn more about WKU. This component is properly aligned to help with organization, thus making it clean and easy to use.

****

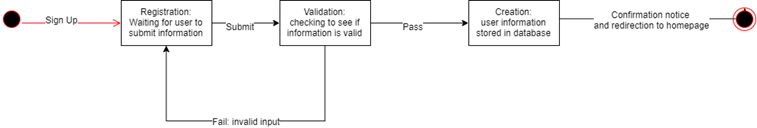
**Figure 18** Actual Homepage Interface

Figure 18 shows the actual implementation of the homepage with the guidance from the homepage wireframe. Using the wireframe to address the user interface requirements, team two was able to create a well-designed page that fulfills the user’s interface requirements.

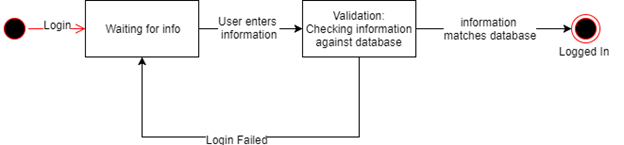
**Functional Requirements**

Functional Requirements describe the functions that the system should be able to perform. Our system will be able to perform the following functions:

*Past Sprint Requirements*

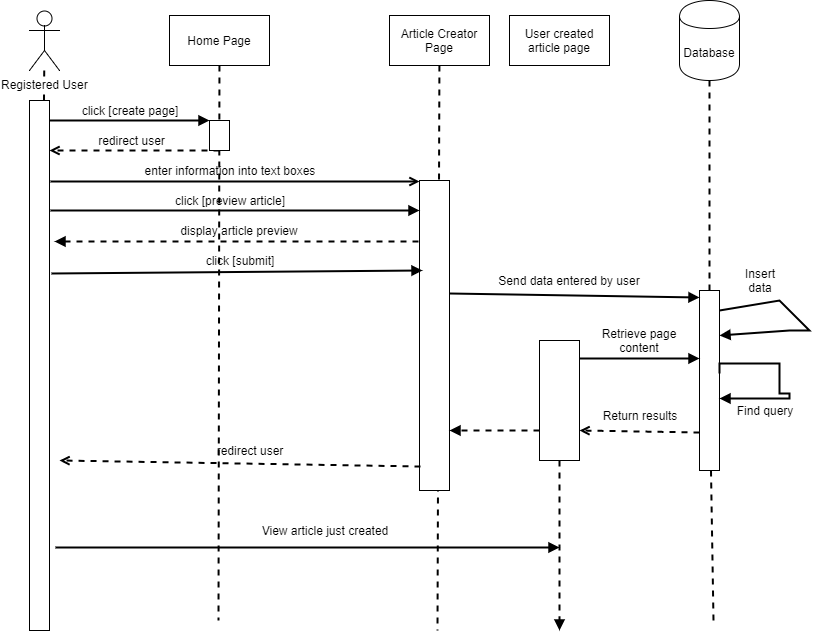
**Figure 19** Account Creation

Account Creation – The above diagram (Figure 19) shows the state diagram associated with creating an account. The user begins by clicking the sign-up button in the main menu. Once the user has filled out the form he will press the submit button and if the information is filled out correctly, then the information will be stored in the database and a message will be displayed notifying the user that their account has been created (Figure 20).



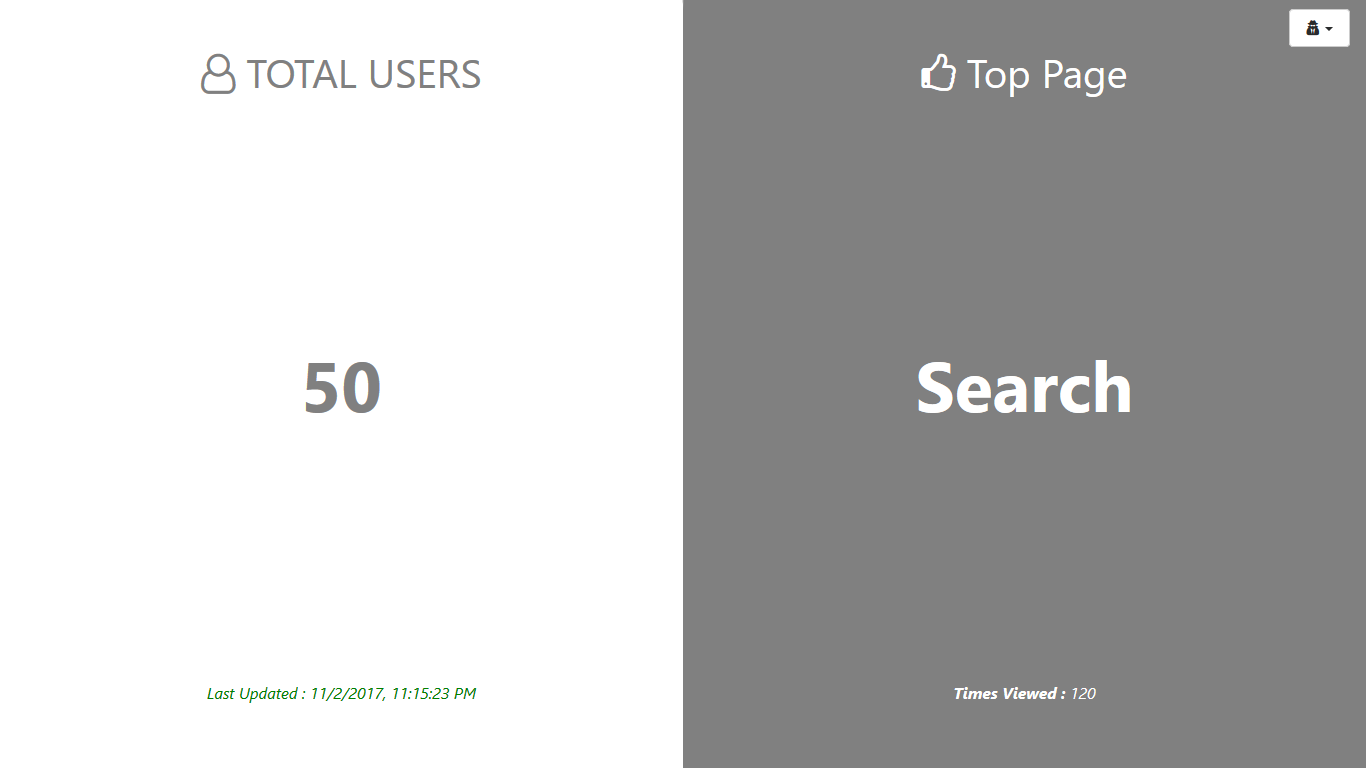
**Figure 20** Login

Log In – The diagram above shows the state diagram associated with logging into an account. The user begins by entering his email and password in the appropriate forms and clicking submit. Our system then verifies this input against our database, if they match then they will be logged in. If no match is found, then the user will be notified and asked to renter valid information.



**Figure 21** Create Page Sequence

Create Page – The diagram above, Figure 21, is a sequence diagram. It is used to model the interactions between the actors and the objects within a system. This diagram models the interactions and sequence of events between the registered user and the system. The user will have to first go to the homepage and select the create page option. From there they will be redirected to the article page creator page. They will then have to input the data into given text boxes and hit submit. The data entered by the user is then sent and stored into the database. The website then redirects the user to the new page they just created by pulling the data they entered from the database and displaying it onto the web browser.



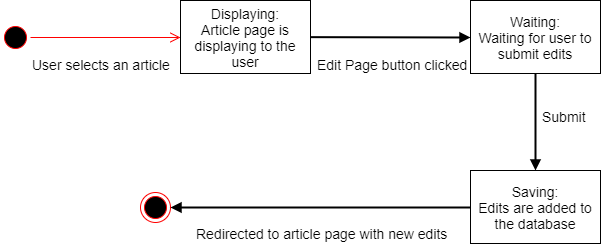
**Figure 22** Performance Monitor Screenshot

Performance Monitor – A web-based administrative interface was created to display the number of total users as well as the most viewed page on the website. This page provides this information in real time (Figure 22).

*Final Sprint Requirements*

The requirements listed below are agreed upon by the team and client to develop during the final sprint. This includes allowing users to search for articles as well as editing them.

Edit Page – Users who have created an account and are logged in will be able to make changes to existing pages. The diagram above shows the state diagram associated with the edit page feature. The logged in user will first direct their web browser to an article page of their choice. If the user decides to edit the article, they click the edit article button. This will redirect them to the page editor for that article. The editor page will then wait for the user to submit their edits. Once submitted, the changes are saved to the database and the user is redirected back to the article viewing page including the new changes (Figure 23).



**Figure 23** Edit Page State Diagram



**Figure 24** Search Bar on Homepage

Search – The search function will take in the users input which should consist of the name of the article they want to find, or words related to it. The system will then provide pages that most closely relate to the users input. In order to implement this, we will have to determine how to best implement the search function and have pages available to search for (Figure 24).

*Process Requirements*

Docker –Docker containers will be used as the web and database server, testing environment, and as the packaging tool to deliver the final product. Currently, one image is in use which contains an PHP image packaged with the Apache web server. This container is meant to hold the web interface of the project. Two more containers will be developed. One will contain the database and the other will contain the file repository. Currently this container will use the localhost MySQL database until the database image is developed. The setup and installation of Docker can be found in the appendix as well as instructions on how we implemented the container and how to connect to the Docker container.

Docker Cloud –Docker Cloud will be used as a cloud repository for the Docker images. New builds of the images when code is updated or added will be pushed into this repository. This will allow access to the containers remotely as well as allow us to access past builds. Docker Cloud will also give us the ability to automate builds from external repositories once that option is set up. This will speed up the development process. The setup of Docker Cloud can be found in the appendix.

Github –Github is now an integral part of development process because of the way the Dockerfile for the LAMP image is setup. All changes will now be pushed to Github for testing in the development environment. To deploy changes, the container will have to restart with every change. The project can be viewed in the web browser at the url: 104.145.83.147/html. Github will also serve to increase progress visibility since the whole team can access and view all of the main source code. The setup of the team’s Github can be found in the appendix. The repository can be viewed at the url: <https://github.com/michelleschap>.  
**Software Process Model**

The software process that our team used during this project can best be modeled by the Iterative Approach. This is particularly standard, because it is a simple and very common approach to go about process development. A lot of the process was actually done and planned out before we even knew exactly what the "Iterative Approach" ever was, and only after learning about models did we realize this was the best model to display the process that we followed during development.

**Nonfunctional Requirements**

A nonfunctional Requirement describes how the system should be. This is criteria that can be used to judge the system. The following are a few nonfunctional requirements we plan to implement.

Compatibility – The website will look good on devices of varied sizes. The code we have created allows the website to conform to the size of the screen and look decent on the numerous sizes available. So, it will look good on a mobile device as well as on a pc.

Maintainability – Our system will be, for the most part, maintained by the users. This will lessen the work load on us (the developing team) as the users themselves will be able create and add information to the system.

Security – Every user who creates an account will have to create a password, this password will be hashed and then stored in the database. In the case that someone gets access to the database, rather than having access to the passwords and being able to login as any user they will only have access to the hashed password which will not enable them to do anything.

Usability - We have implemented an easy to use interface so that anyone can easily utilize our system. Our system will mostly consist of a main page and the article pages. Users will also be able to contribute by creating an account but will not be obligated to do so.

Reliability: Probability of Failure on Demand (POFOD) – This is a metric that measures the probability that a demand for service from a system will result in a system failure. To obtain the data, we will count the number of transactions that is performed before a system failure. Let x be the number of transactions before a system failure, we will perform the equation 1/x to calculate the POFOD.

Reliability: Rate of Occurrence of Failures (ROCOF) – This metric measures the rate of failure in the system. We can measure the ROCOF by dividing the number of failures by the total number of operations.

Reliability: Mean Time Between Failure (MTBF) – This metric measures the amount of mean time between failures. To calculate the MTBF, we will have to log the average time between each failure.

Reliability: Mean Time to Repair (MTTR) – This metric measures the amount of time to repair a system failure. To calculate this, we will have the keep track on how long it will take us to correct a failure.

Reliability: Availability – Availability is a metric that measure the MTBF/ (MTBF+MTTR). In essence, it measures the average amount of time that a system is able to respond to a client’s request.

Reliability – Overall reliability is measured by the MTBF/(1+MTBF). In summary, it measures the probability of failure free system for a specified period.

**Software System Performance Requirements**

*Subsystem Performance Objectives*

The following performance requirements are based on the current benchmark measurements taken from testing the current subsystem. The subsystem's performance expectations can be found below in Figure 25. These performance requirements are to be used as a proof of concept since the Raspberry Pi can only handle a fraction of the traffic and users desired in a final released product. These requirements are meant to be used to curve the development in a way to help optimize the overall systems performance and make the system more scalable.

|  |  |
| --- | --- |
| Component | Performance Objective |
| Database | Handle 200 transactions per second |
| File I/O | Process 5,000 kb/sec |
| CPU | Execute 10,000 events under 300 seconds |
| Network | Ping under 100ms |

**Figure 25** Subsystem Performance Objectives

*Subsystem Expectations*

The table below (Figure 26) lays out the subsystem performance expectations from the current hardware inside the Raspberry Pi. These values were taken into account during our performance engineering analysis.

|  |  |  |
| --- | --- | --- |
| Subsystem Type | Hardware | Performance Expectations |
| CPU | 4x ARM Cortex-A53 | Minimum speed 700MHz |
|  |  | Maximum speed 1.2GHz |
|  |  | L1 Cache 8-64KiB |
|  |  | L2 Cache 128KiB-2MiB |
| Network | BCM43438 wireless "combo" chip | Dual-band 2.4 GHz and 5 GHz IEEE 802 |
| File I/O | 1GB LPDDR2 | Internal Access Rate 200MHz |
|  |  | I/O Bus Clock Frequency 400MHz |
|  |  | Data Transfer Rate 800MT/s |

**Figure 26** Subsystem Performance Expectations

*Application Workload Mix*

**Figure 27** Application Workload Mix

The pie chart above (Figure 27) represents the estimated ratios of our application workload. The majority of users will use this application to search for the page relevant to the information they are interested in and then viewing said page especially since this is the only function available to all users, including unregistered users. 60% of the application workload is estimated to belong to the search function because of this. Another 15% will be spent on registered user's logging in. It is estimated that a majority of users will not create an account since most users will use this to read articles rather than create or edit them which is why we only estimate 15% of the workload to belong to logging in and 5% to user account creation. We feel that creating and editing a page will be split. When the application is released many articles will be created but the number of available new articles will dwindle down quickly since this only serves to cover information about WKU. A lot of edits and editions will be made to articles after the initial release of the application. After the WKU wiki has been populated with articles, the usage of the create page and edit page functions will drop significantly so we awarded them 20% of the application workload mix.

**Figure 28** System Workload Mix

The pie chart above (Figure 28) represents the system workload mix. Shown in the pie chart corresponding to the application workload mix, a majority will be searching for articles to view and logging in. Because of this we estimate that 30% of the system's workload will belong to reading data from the database. Since a small portion of users will be creating and editing articles, we estimate that only 10% of the workload will belong to writing data to the database. A large portion of the workload will be communicating with the network and server through http to communicate with the users. We estimate 60% of the workload will belong to this splitting it in half for http requests and http responses.

**Bottlenecks**

Bottlenecks are areas that constrict flow of data, these include both hardware and software. In order to for our system to perform to the expectations we have set in the previous sections we will have to take these areas into consideration and attempt to optimize them. We are for the most part confined to using a raspberry pi for this project so most of our optimization will come from adjusting the code to minimize the impact of these bottlenecks

*Hardware*

Since we are using a raspberry pi 3 our system will be limited by the hardware of the pi. The pi’s CPU is a quad-core ARM Cortex-A53 which has a max speed of 1.2 GHz. It has a 1GB LPDDR2 with an internal access rate of 200MHz and a data transfer rate of 800 MT/s. That means that the performance of the system is limited to these standards.

*Software*

Hot spots occur where the probability of a problem occurring is high and the impact of the problem is great. Since our system is centered around finding relevant information by searching the database, one of the features (a hot spot) that will be used most often will be the search feature. Because this feature will be used often it will play a major role in demonstrating the performance of the system. For that reason, we will need to pay special attention to the processes that make up this function and optimize them to be fast enoughto perform to the user’s expectations.

The search function consists of the user sending search terms to the webpage, the webpage will then form an SQL query that will be sent to the database, the database will then return the results of the SQL query to the website which will sort the results according to how relevant it is and display them for the user. In order for the search to be more efficient we have added tags to each article. These are just words that can be used to broadly describe or are associated to the article itself, for example an article about Cherry Hall may have the tags “buildings”, “English”, “History”, and “Philosophy”. This will allow the search function to prioritize these words when sorting the articles and determining which articles the user is most likely looking for. If the system does not return any results quickly enough the user may become agitated and is likely to not continue to use our system.

**Security Requirements**

Identification *–* The system will identify the user during login. If the user does not login then the system will restrict the user to only the basic functions of the system.

Authentication *–* Users will be identified during login based on the email that they have provided and the password that they have chosen. They will submit these to the website and then it will be compared to the database to make sure that they are in the database.

Authorization *–* There will be three different types of users, non-registered users, registered users, and admin users. Non-registered users will only be allowed to create an account and search/browse articles. Registered users will be allowed to search/browse articles create articles, and edit articles. Admin users will be given special privileges such as access to a monitoring page, deleting articles, and blocking users.

Immunity *–* To protect the system from viruses, worms, and similar threats we will install anti-virus software.

Integrity *–* In order to prevent losing important files due to corruption we will regularly back-up the data so that in the case of a loss we will be able to just replace it with a back-up.

Intrusion *–* We will consider using a Network-based Intrusion Detection System (NIDS) such as Snort to detect attacks on the system.

Privacy *–* Private information such as passwords will be hashed and then stored in the database this will prevent anyone from knowing the actual passwords of the users. As a second line of defense the database is only accessible to those who have the correct password. So even if someone were to get access to the database they would not acquire any confidential information.



**Data Dictionary Design**

The design of the Data Dictionary we will be using is a simple chart that shows each of the names of the essential variables, lists the function of that variable, the page(s) that this variable is used, and gives a short description. The full data dictionary is much larger and would be infeasible to put here, as the project is very large, but this example and sample shows the format of the dictionary. See Figure 29 for an example of this.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Dictionary | | | |
| Name | Type | Where Used | Description |
| $loginSuccessful | Boolean | login.php | Returns true if login succeeds |
| $accountLevel | Integer | login.php + | Variable that holds account level (0,1,2) |
| $name | String | login.php + | String representing username |
| $userID | Integer | login.php + | The hidden ID of a user |
| $password | String | login.php + | String holding temp password to pass & check |
| $accExists | Boolean | login.php + | Returns true of the account exists |
| $email | String | register.php | String that holds email address |
| $fname | String | register.php | String that holds first name of user |
| $lname | String | register.php | String that holds last name of user |

**Figure 29** Data Dictionary Example

**Pseudocode Design**

One of the core functions of our site as a Wikipedia clone is the ability to create pages as a normal user, to accomplish this we follow a simple path to processing the data and storing it in our database, the Pseudocode for that specific function is below, full source code can be found in the appendix. Figure 30 shows the pseudocode for the create page function.

|  |
| --- |
| JS  //Get all field values - each  Var … = document.getElementById(...);  //Make sure not empty  If value.length is not 0 {  //Nested for each element  Ok to submit values to php  }  PhP  If dbConnection is true {  Contact database and insert data from the fields  If it was a success{  Echo success result – 1 back to ajax call  }  } |

**Figure 30** Pseudocode Create Page Function

**Class Diagrams Design**

For our project thus far, we currently do not have any classes or any foreseeable need for classes. As the project develops and we continue to build the Wiki, it is possible that this situation will change and we will add classes, providing the needed updates here. At the moment though, there are no plans and no need of doing such.

**System Boundaries Design**

Our mini WKU Wiki is hosted on a small Raspberry PI 3, meaning that there are obviously some boundaries and limitations to the system, both on a physical and logical level. Our team has realized over the past several weeks while working on the Wiki that this is true, hitting a number of unforeseen walls and unexpected hindrances to a completed project. These obstacles can be most easily overcome once recognized and listed.  
*Physical Boundaries*

A few of the physical boundaries that the PI has are not necessarily any that we will likely hit any time soon. For instance, the storage limit for the PI is 32 Gigabytes, a boundary that would take some serious effort to reach, and would not be the first one reached. The RAM for the PI maxes out at 1 Gig, a much easier limit to max out, especially if multiple users are on simultaneously using functions that require a fair amount of RAM, the search function for example. Another limitation that could actually be an issue is the network speed where the PI is hosted, which is not really a limitation of the PI itself, but does factor into the functionality of the system. The bandwidth of the PI is satisfactory, but could easily be for less so if several users are online at the same time. Also, the internet where the server is hosted is not the most dependable ever, having gone out a few times since the start of the project. Finally, the PI is known to get fairly hot after running non-stop for weeks, and that could really become an issue of there is any damage done to the server.

*Logical Boundaries*

The logical boundaries for this website are primarily seen because of the project requirements. For example, our project requires the use of no less than 3 docker containers, which in itself is a boundary we cannot undermine. Also, there are few, but some, a rather small number of, limitations to various coding languages and software components which we are using, some of which have been causing errors and bugs. To name one, the IDE Zach has been using to code php with does not give adequate error messages, meaning he is often presented with a problem but no indication of how to fix it. This limitation of the IDE is a problem considering how much time it takes to fix something that could be really rather simple.

**Software Architecture Design**



**Figure 31** Data Movement Through System Components

In order to create a system, team two used an UML to create the software architecture design. In Figure 31, one can see that there are several components to the software architecture design. The figure shows how the user will interact with the software and how the software will interact with itself. From the web browser (user), communication will be transferred to HTTP (Hypertext Transfer Protocol). The HTTP will handle communication between the user's web browser and our web server which is contained within a Docker container. This web server container is bound and communicates with the local host (the Raspberry Pi) on port 80 and this container is also linked to another container holding a database. This container is bound to the local host (the Raspberry Pi) on port 3306. The web server container will pull its web files from a Docker container holding a file repository. This file repository container has not been implemented yet.

**Software Engineering Performance Design**

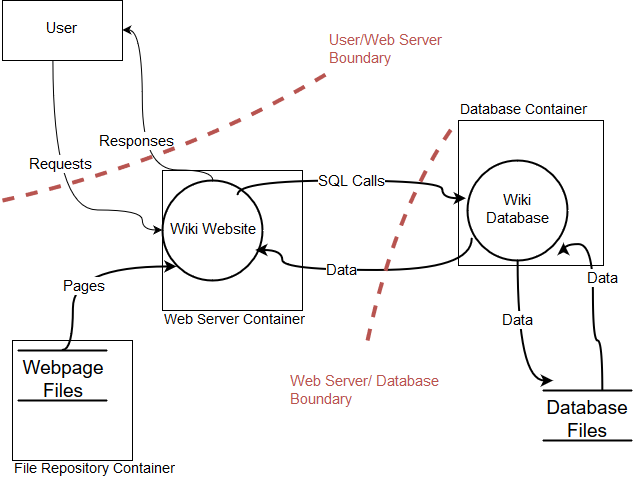
*Development Decisions*

Throughout the development of our project, we've used multiple languages such as HTML, PHP, JavaScript as well as CSS. We decided to use these specific languages due to their universal compatibility in relation to each other as well as the history of extensive testing endured by each of the languages. When using these languages to perform a specific database task, there is a common method we use to make these database tasks, that being using java script ajax call to a php file and obtaining the echo result from the database query result and determining rather the call was successful and display information based on that result in the form of html. All the data for our project is stored in the encrypted MySQL database. Some of the data in this database is even hashed out for added security.

When elaborating on the frequency of high level calls we must discuss all the calls that take place for multiple tasks. These calls include ensuring that an active database connection exists in the dbconnection file as well as the calls that fetch the php session variables to be delivered to CSS files. These calls are executed quite frequently, as almost every database connection will interact with these methods. The calls to the low level library routines are far more common as the methods of these libraries are often determined to be the most efficient way of accomplishing a task. For example, calls to execute a MySQL query all use the same functions uniformly across all of our project.

**Security Design**

In order for a user to gain access to any private information they would have to get through the security barriers of the website. Without being logged in the user will only be allowed to search/view pages or create an account. The only threat here being SQL injections which will be deterred by the use of SQL prepared statements. In order for a user to gain access to an admin account they will need to know their password which will only be known by the user themselves since the password is hashed and then stored in the database. To gain access to the database the user will need to enter the correct password which is currently not stored anywhere (Figure 32). The figure show how SQL injections will be prevented, thus increasing security to our website. Preventing unwanted access to information will attribute to accountability. Having high accountability on the Mini WKU wiki means that team two will know what who has access to certain type of information. The security one can see that there is a security boundary between the user and the web interface. Furthermore, one can see that there is a trust boundary between the web interface and the database. These two trust boundaries will help increase security to the Mini WKU Wiki.



**Figure 32** Security Threat Data Flow Diagram

\

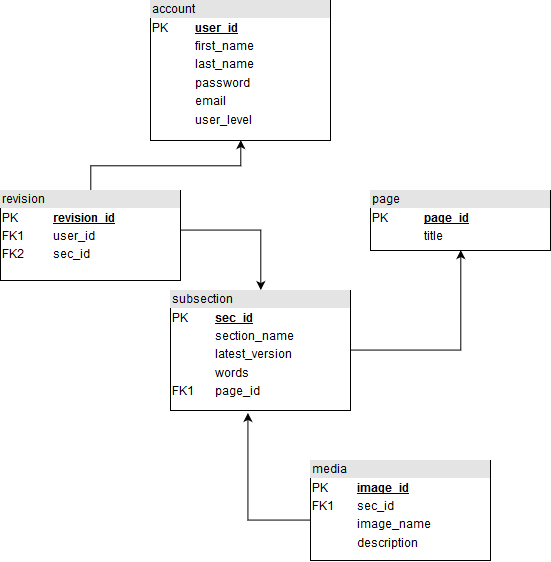
**Implementation**

*Frontend: Creating the Interface*

The frontend interface for the mini wiki uses HTML, CSS, and JavaScript. In addition, it uses the framework of Bootstrap. In order to use Bootstrap, the CSS and JavaScript files are linked in each HTML document in the mini wiki project. The CSS is linked towards the end of the header while the JavaScript is linked towards the end of the body of the HTML file. CSS and JavaScript provide functionality and styling to each of the webpages.

*Backend: Database Creation*

In order to create the database on the pi, a remote connection had to be established. Once phpMyAdmin was installed on the pi the database could be made on a different device by specifying the ip adress of the pi. To access the pi and create the database the ip address and phpMyAdmin could be typed into any address bar with internet connection (104.145.83.147/phpmyadmin). From here mySQL was used to create the database itself. The database is organized into five tables consisting of account, media, page, revision, and subsection. Figure 33 illustrates how the tables are organized and connected to each other.



**Figure 33** Table Organization

*Backend: Adding Functionality*

To add functionality, PHP and JavaScript were utilized to create scripts to connect the front end, apache server, and database. PHP MySQL calls were used to send queries to the database through prepared statements. PHP was also used in conjunction with apache to connect the files, web server, and MySQL server. JavaScript was used as a tool to create dynamic content. The front-end files are then combined with or call functions from the back end scripts created to create a functional version of the website. The source code can be found in the appendix.

*Docker: Creating and Packaging the Server*

To connect the web files and database to an Apache web server, a container was created that runs PHP in conjuction with an Apache httpd. A Dockerfile was created that pulls an image that contains PHP version 7.0 as well as an Apache server. This Dockerfile, as well as all of the front end/server scripting files, are packaged into an image and pushed into the team's image repository on Docker Cloud. A new image is pushed every time code is updated or added. The latest image in the repository can be tested by pulling and running the latest container and binding the Apache server to the localhost's port 80. Currently, the Apache server will have to connect to the local database server (MySQL) hosted on the Pi. In the next prototype implementation, a container holding the database will be developed and linked to the current image. Details on the setup and build instructions of Docker and Docker Cloud can be found in the appendix.

**Prototype Approach**

Our prototyping plan for this project follows the schedule and format required of the Computer Science 360 class, having demonstrations and documentation provided during key dates where we should meet all goals of a certain "Milestone." This approach includes reports of progress to a client, a presentation to the class explaining what we've done in the Milestone, a demonstration of the current website, the website version itself, and the Milestone report.

*Plan*

The schedule shown in Figure 3 lists the Milestone dates, by which point we plan to have the parts listed on the schedule fully implemented and ready to present as a prototype. Each prototype version of the website will be the current version, meaning that each version is not saved somewhere as a separate prototype, but that every current version is the latest version and a just prototype of the final completed version of the project. The changes to every latest version are pushed to github to be backed up immediately after changes and updates are made.

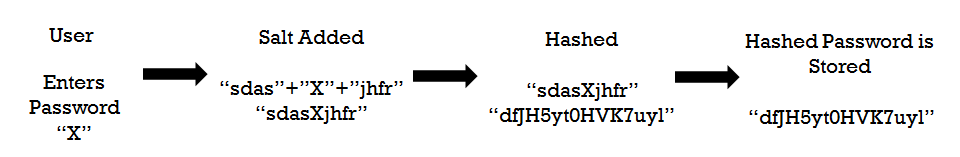
*Documentation*  
 Each Milestone date a completed Milestone report (the report you are presently reading) is completed with the sections that are mandatorily required for the class. These sections are continuously more in depth every Milestone, and cover different parts of the software our team is developing. Also, weekly, our team presents a weekly report of our progress to the client of our team, Mahn Do. These weekly reports show what was completed and what was planned to be completed in the previous week, as well as the plan for the next week.

**Security Implementation**

*Hash Password for Registration*

In order to secure our website user's login credentials, we created a function which hashes the password with salt upon user registration. This means that when the user submits their username and password, the password will be modified based on a highly secure, highly complex hash key that the users cannot see and stored. The "salt" is an additional complexity, two random strings of characters stored safely in the system beyond access of any user which are concatenated to the front and back of the password before it is hashed and the hashed password saved. After the salt is added, the password is scrambled and rearranged (hashed) through an offsite hash key that is virtually impossible to crack. The hashed password is the only version saved to the database, so, even if the admin user wanted to retrieve the password it would be impossible, meaning it cannot be hacked either (Figure 34).

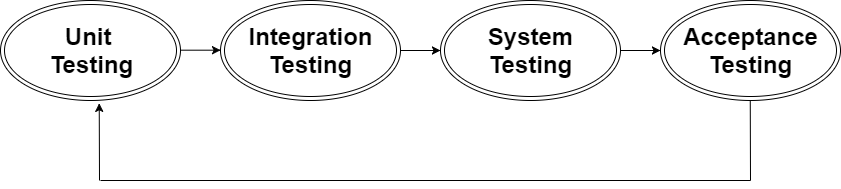
When the user logs on again, the password they enter has the salt added and is hashed through the same key again, then the 2 hashed passwords compared. If the password is correct, they should match, and the user will be logged on and given access to their account. This method of storing passwords is one of the safest possible ways and gives no one the actual password.



**Figure 34** Security Password Implementation



**Verification and Validation (Review)**

**Figure 35** Testing Cycle

Verification and validation allows the team members on team two to make sure they are creating a product that conforms to the system specification and meets the requirements of the client. To make sure that the system meets the client’s expectations, the group will go through a series of testing process. Throughout the verification and validation process, the group members goes over project functional and nonfunctional requirements. In addition, test cases will be generated based on the requirements. Thirdly, the group will execute the system functions based on the test case to make sure they abide by the requirements. Overall, to address verification and validation, tea two iterates through a review cycle. The cycle includes unit test, integration test, system test, and acceptance testing (Figure 35).

*Unit Testing*

Throughout the unit texting individuals components of the mini wiki will be tested against the design in the documentation, which includes functional and nonfunctional requirements. For example, one of the functional requirement on the mini WKU wiki is to allow members to create and modify pages. In order to test the functionality of this requirement, team two will have to set up a member level account to test to see if user will be able to create and modify pages. Furthermore, one of the nonfunctional requirement is to make sure that users are able to access the webpage within a reasonable amount of time. To address this, team two will have try to access the website through the client side to make sure this nonfunctional requirement is met (Figure 36).

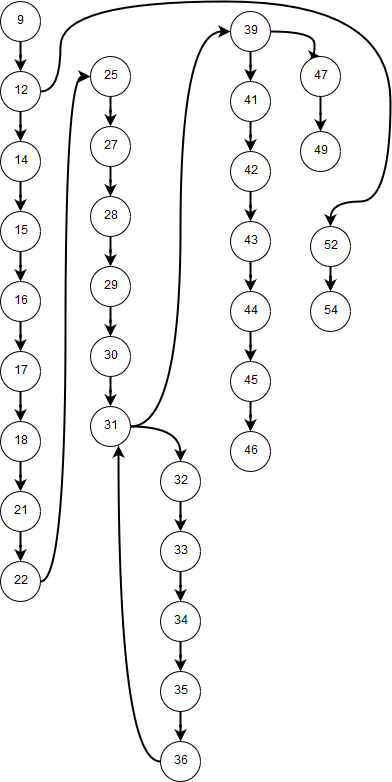
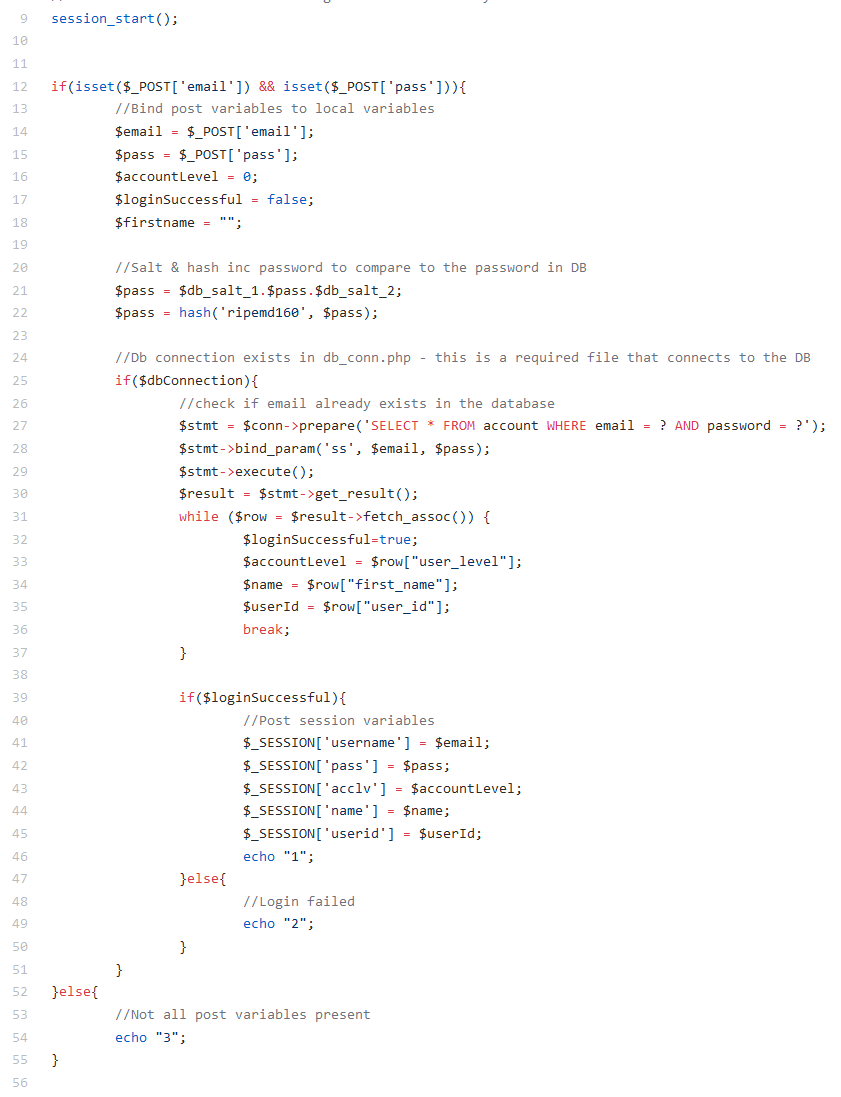
|  |  |  |  |
| --- | --- | --- | --- |
| **Unit Test Cases** | | | |
| **ID** | **Test Cases** | **Relevant Input Values** | **Expected Output** |
| 1 | **Test Procedure:**  Login with an account that has already been created.  **email:** abc@gmail.com  **Pasword:** 123456  **Expected results:**  Invalid login combination message will appear for wrong email/password combination. | Email: abc@gmail.com  PW: 123456  Email: abc@gmail.com  PW: 1234567  Email: ab@gmail.com  PW: 123456 | Successful login  Invalid login combination  Invalid login Combination |
| 2 | **Test Procedure:**  Creating accounts with different emails, same emails, and same names and different emails.  **Pasword:** abcdef  **Expected results:**  If an account is created with the same email a message should notify the user that the email they used is already associated with an account. If an accounts are created with the same names and/or passwords but different emails it should accept the accounts. | First name: abc  Last name: def  Email: [abc@email.com](mailto:abc@email.com)  PW: abcdef  First name: abc  Last name: def  Email: [abc@email.com](mailto:abc@email.com)  PW: abcdef  First name: abc  Last name: def  Email: [abcdef@email.com](mailto:abcdef@email.com)  PW: abcdef | Successful account creation  Unsuccessful creation (email already in use)  Successful account creation |

**Figure 36** Unit Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Unit Test Results** | | | | |
| **ID** | **Test Cases** | **Pass/Fail** | **Tested By** | **Date Tested** |
| 1 | **Test Procedure:**  Login with a user account  **email:** abc@gmail.com  **Pasword:** 123456  **Expected results:**  Invalid login combination message will appear for wrong email/password combination. | Pass | Daniel Monterrosas | 11/15/2017 |
| 2 | **Test Procedure:**  Creating accounts with different emails, same emails, and same names and different emails.  **Pasword:** abcdef  **Expected results:**  If an account is created with the same email a message should notify the user that the email they used is already associated with an account. If an accounts are created with the same names and/or passwords but different emails it should accept the accounts. | pass | Daniel Monterrosas | 11/17/2017 |

**Figure 37** Unit Test Results

The tables above (Figure 37) depict the tests used to determine whether or not the separate functions worked reliably or not. The First test determines whether the login function works as intended. In this case it does as it only allowed the user to login when they entered the correct credentials. When the user entered the wrong credentials an error message was displayed and the user was not able to log in, this is exactly what we intended to happen so the system passed the test.



**Figure 38** Flow Diagram for the Login Feature and Associated Code

From the flow diagram depicted above (Figure 38) we were able to create test cases for the login function. The flow diagram allows use to see how many different outcomes are possible for the login function the important parts in the flow diagram, the forks, represent where there are more than one possible outcome. For example, at line 12 if the system does not receive an email and password variable, it will just skip to line 52 and bypass the majority of the code as it will not be utilized. Different types of path allow for different length of code processes. In conclusion, one can see that there are numerous paths in our system. This will result in the possibility for paths to have different lengths. Conditions statements such as while, for, and if, will attribute to a split or loop in the flow of the system processes.

*Integration Testing:*

In the project there will be some components that relies on the functionalities of other components. Group two will perform integration testing in order to validate the functionalities of dependent requirements. For example, the mini WKU wiki will require the use of MySQL database (Component A) and front-end web interface (Component B.) Group two will test to that the front-end web interface will be able to correctly fetch the require data for a particular web page. Furthermore, group two will make sure that the web interface will reliably push the user edits to the database. In both of these situations, team two will verify and validate the integration testing in this process.

*System Testing:*

After all of the working components are developed, group two will perform a system test. This test will entail the verification and validation of the completed system as a whole. All the functionalities will be tested to make sure it meets the specified functional and nonfunctional requirements. Since group two will be using the iterative approach to software development, the system test will be conducted many times throughout the life of the project. The system test will be performed at the review stage of each sprint; cumulative testing will be conducted to make sure the system meets the functional and nonfunctional requirements.

*Acceptance Testing:*

Team two will perform an acceptance test. This test will entail a test case which will be used to validate the system functions for the client. The acceptance test will be performed after the system testing phase during the review section of the sprint. The client and the members on team two will review the completed functional and nonfunctional requirements that implemented on the min WKU wiki with the specification of the client. The review will result in either a repeat of the testing cycle if the requirements are not fulfilled or be pushed to the next phase, delivery (Figure 39).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Functional Requirements | | | | |
| **Function being tested** | **Action To perform** | **Present?** | **Ease of action**  **(1-10)** | **Comments/Suggestions** |
| Account Creation | Create an account | Yes  No |  |  |
| Login | Login with an account | Yes  No |  |  |
| Create Page | Create a page | Yes  No |  |  |
| Edit Page | Edit your page | Yes  No |  |  |
| Search | Search for something | Yes  No |  |  |

**Figure 39** Form Used to Make Sure all Functional Requirements are Met

*Delivery:*

If the acceptance test between the client and members on group two is successful, the testing cycle will be pushed to the final stage, delivery; otherwise, the system will iterate again through the testing cycle. In the delivery stage, the final product (mini WKU wiki) will be released.

**Software Performance Validation**

As a part of the validation step, our system needs to be benchmark tested to see if it meets both the hardware and software requirements. The scope of the benchmark testing process will entail the testing of File I/O performance, CPU performance, and Database service performance. Using Sysbench, team two will implement the benchmark testing process for File I/O, and CPU service performance on both the base OS and in the Docker Container. Furthermore, team two will implement a performance test for the database in the base OS environment. In addition, the network test will be performed through a small network of bots from around the world that pings the PI.

*File I/O: Docker and Base OS Environment*

Testing Procedures – In order to test the Disk I/O, we used sysbench and pass in some parameters. To test the CPU, we used sysbench to calculate the amount of time in seconds it took process different sizes of files (using {128 MB, 256 MB, 512 MB, 1 GB, 2 GB, 4 GB, 8GB}). We performed a total of 21 different test (three tests per file size) and took the overall average per file size in both of the environments (Docker and Base OS) .

Test Input – To perform the test, we wrote a little script for bash to execute. The following script is used to calculate the amount of time for the pi to process the different size of files for both of the environments (Figure 40).

|  |
| --- |
| *for i in 128MB 256MB 512MB 1GB 2GB 4GB 8GB*  *do*  *sysbench --test=fileio --file-total-size=$i prepare*  *sysbench --test=fileio --file-total-size=$i --file-test-mode=rndrw --init-rng=on --max-time=300 --max-requests=0 run > test$i.txt*  *sysbench --test=fileio --file-total-size=$i --file-test-mode=rndrw --init-rng=on --max-time=300 --max-requests=0 run >> test$i.txt*  *sysbench --test=fileio --file-total-size=$i --file-test-mode=rndrw --init-rng=on --max-time=300 --max-requests=0 run >> test$i.txt*  *sysbench --test=fileio --file-total-size=$i cleanup*  *done* |

**Figure 40** File I/O Input Test Script

Test Output: – Figure 41 shows an example test from the File I/O output in the Docker environment for a 1 Gb test. Although the test for the base OS is in a different environment the overall output reflects the output from the Docker environment.

|  |
| --- |
| Operations performed: 52800 Read, 35200 Write, 112551 Other = 200551 Total  Read 825Mb Written 550Mb Total transferred 1.3428Gb (4.5813Mb/sec)  293.20 Requests/sec executed  Test execution summary:  total time: 300.1334s  total number of events: 88000  total time taken by event execution: 36.4240  per-request statistics:  min: 0.02ms  avg: 0.41ms  max: 33.47ms  approx. 95 percentile: 1.39ms    Threads fairness:  events (avg/stddev): 88000.0000/0.00  execution time (avg/stddev): 36.4240/0.00    sysbench 0.4.12: multi-threaded system evaluation benchmark    Running the test with following options:  Number of threads: 1  Initializing random number generator from timer.    Extra file open flags: 0  128 files, 8Mb each  1Gb total file size  Block size 16Kb  Number of random requests for random IO: 0  Read/Write ratio for combined random IO test: 1.50  Periodic FSYNC enabled, calling fsync() each 100 requests.  Calling fsync() at the end of test, Enabled.  Using synchronous I/O mode  Doing random r/w test  Threads started!  Time limit exceeded, exiting...  Done. |

**Figure 41** Example of Docker File I/O Calculation Output for 1 Gb Test

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **File Size** | **Test** | **Docker Time (Kb/s)** | **Docker Average (Kb/s)** | **Base OS Time (Kb/s)** | **Base OS Average (Kb/s)** |
| **128 MB** | a | 10146.00 | 10122.9 | 11615.00 | **11752** |
|  | b | 9724.70 |  | 12172.00 |  |
|  | c | 10498.00 |  | 11469.00 |  |
| **256 MB** | a | 3537.40 | 2411.0667 | 11022.00 | **10914** |
|  | b | 1863.60 |  | 10757.00 |  |
|  | c | 1832.20 |  | 10963.00 |  |
| **512 MB** | a | 4368.80 | 3765.8 | 9924.40 | **10285.8** |
|  | b | 3400.30 |  | 10641.00 |  |
|  | c | 3528.30 |  | 10292.00 |  |
| **1 GB** | a | 4581.30 | 4150.0667 | 7800.90 | **7881.1** |
|  | b | 3999.90 |  | 7681.80 |  |
|  | c | 3869.00 |  | 8160.60 |  |
| **2 GB** | a | 4975.50 | 4764.5333 | 7035.80 | **6817.5333** |
|  | b | 4975.50 |  | 6714.00 |  |
|  | c | 4342.60 |  | 6702.80 |  |
| **4 GB** | a | 5326.30 | 4914.2667 | 6546.00 | **6451.6667** |
|  | b | 4718.60 |  | 6351.50 |  |
|  | c | 4697.90 |  | 6457.50 |  |
| **8 GB** | a | 3599.10 | 3465.4667 | 6262.40 | **6135.3** |
|  | b | 3334.30 |  | 6087.80 |  |
|  | c | 3463.00 |  | 6055.70 |  |

**Figure 42** Dockerand Base OS File I/O Output Test Summary

After running three tests per thread per environment, below is the table for the overall File I/O output for both of the Docker and base OS environment (Figure 42).

After the average was calculated, it was plotted using a line graph to indicate the output for the File I/O test in the Docker and base OS environment (Figure 43). The output results indicate that running the test within the Docker environment results in a decrease in performance rate. This can be expected since the PI has to use additional resources to support the Docker environment. Furthermore, in general the chart indicates that there is a negative linear relationship between the file size and average execution rate. To be specific, there steep decrease in average execution rate for file sizes greater than 512MB in the base OS environment; this is to be expected since the ram size of the Raspberry Pi only has 1 GB of RAM. The sharp decrease indicates that larger file are stored on the MicroSD instead of the RAM (getting information from RAM is much quicker than from the MicroSD). On the other hand, there is a sharp decrease in average execution rate for file size greater than and equals to 256MB in the Docker environment. Since the PI has to support addition resources for the Docker container, less ram is available, which results in the sharp decrease of average execution rate at a much smaller file size when compared to that in the base OS environment. In the end, the graph shows that the average time to process a small file that can be stored in the RAM is significantly faster than processing file from the disk.

**Figure 43** Docker File I/O Average Execution Rate

*CPU: Docker and Base OS Environment*

Testing Procedures – In order to test the CPU, we used sysbench and pass in some parameters. To test the CPU, we used sysbench to calculate the amount of time in seconds it took different thread count (using {1,2,4,8,16,32}) to calculate 15,000 prime numbers. We performed a total of 18 different test (three tests per thread count) and took the overall average per thread in both environments (Docker and base OS).

Test Input– To perform the test, we wrote a little script for bash to execute. The following script is used to calculate the amount of time for different number of threads to calculate 15,000 prime numbers (Figure 44).

|  |
| --- |
| *for j in 1 2 4 8 16 32*  *do*  *sysbench --num-threads=$j --test=cpu --cpu-max-prime=15000 run> cputest$j.txt*  *sysbench --num-threads=$j --test=cpu --cpu-max-prime=15000 run >> cputest$j.txt*  *sysbench --num-threads=$j --test=cpu --cpu-max-prime=15000 run >> cputest$j.txt*  *done* |

**Figure 44** CPU Input Test Script

Test Output – Below is an example of the output data for the CPU test for 32 threads calculating 15,000 prime numbers in the Docker environment (Figure 45). Once again, the overall output format for the base OS environment reflects that of the Docker environment; only the specific data (i.e. total process time) changes.

|  |
| --- |
| Maximum prime number checked in CPU test: 15000  Test execution summary:  total time: 282.8320s  total number of events: 10000  total time taken by event execution: 9041.0943  per-request statistics:  min: 113.43ms  avg: 904.11ms  max: 1114.74ms  approx. 95 percentile: 964.41ms  Threads fairness:  events (avg/stddev): 312.5000/2.52  execution time (avg/stddev): 282.5342/0.16  sysbench 0.4.12: multi-threaded system evaluation benchmark  Running the test with following options:  Number of threads: 32  Doing CPU performance benchmark  Threads started!  Done.  Maximum prime number checked in CPU test: 15000  Test execution summary:  total time: 291.1831s  total number of events: 10000  total time taken by event execution: 9308.6375  per-request statistics:  min: 113.00ms  avg: 930.86ms  max: 1120.25ms  approx. 95 percentile: 983.35ms  Threads fairness:  events (avg/stddev): 312.5000/2.89  execution time (avg/stddev): 290.8949/0.18  sysbench 0.4.12: multi-threaded system evaluation benchmark  Running the test with following options:  Number of threads: 32  Doing CPU performance benchmark  Threads started!  Done. |

**Figure 45** Docker CPU Calculation Output Example for 32 Threads

After running three tests per thread, below is the table for the output in both the Docker and base OS environment (Figure 46).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of Threads** | **Test** | **Docker Time (s)** | **Docker Average (s)** | **Base OS Time (s)** | **Base OS Average (s)** |
| **1** | a | 704.8941 | **702.6576** | 245.8185 | **246.4031** |
|  | b | 697.4235 |  | 247.6008 |  |
|  | c | 705.6553 |  | 245.7899 |  |
| **2** | a | 357.5556 | **356.7329** | 123.2778 | **123.3406** |
|  | b | 356.3991 |  | 123.2224 |  |
|  | c | 356.244 |  | 123.5217 |  |
| **4** | a | 266.5773 | **276.3085** | 84.0048 | **90.7001** |
|  | b | 282.1939 |  | 92.2767 |  |
|  | c | 280.1544 |  | 95.8189 |  |
| **8** | a | 283.1337 | **283.9275** | 98.2999 | **99.9526** |
|  | b | 283.6378 |  | 100.1807 |  |
|  | c | 285.0111 |  | 101.3771 |  |
| **16** | a | 284.7773 | **286.3180** | 102.1921 | **102.4749** |
|  | b | 289.2878 |  | 102.9332 |  |
|  | c | 284.889 |  | 102.2993 |  |
| **32** | a | 282.832 | **286.5714** | 102.8459 | **102.6792** |
|  | b | 291.1831 |  | 102.4663 |  |
|  | c | 285.699 |  | 102.7255 |  |

**Figure 45** Docker CPU Output Test Summary

After the average was calculated in the Docker environment, it was plotted using a line graph (Figure 47). In both environments, the trend is linear for threads count greater than 4. This is to be expected since the Raspberry PI has a Quad-Core processer. Furthermore, by looking at the graph, one can see that there the average total process time for executions in the Docker Environment is greater than that of the Base OS Environment. Once again, this is to be expected because the Raspberry PI has to support the Docker Environment as well as perform the same calculations found in the base OS environment. In summary, the process tends to be quicker on the base OS environment. In addition, the process time decrease from 1 thread count to 4 thread count but starts to have a linear relationship for all process greater than 4 thread count in both of the environments.

**Figure 47** Docker Thread Count to Average Total Time

*Database Service: Base OS Environment*

Testing Procedures – In order to test the database, we used sysbench and pass in some parameters. To test the database, we used sysbench to calculate the transactions per second for each thread values (using {1,2,4,8,16,32} thread values) and a database table size of 1,000,000. We performed a total of 18 different test (three tests per thread count) and took the overall average per thread in the base OS environment.

Test Input– To perform the test, we wrote a little script for bash to execute. The following script is used to calculate the transaction per second for each thread values {1,2,4,8,16,32} using a table size of 1,000,000 (Figure 48).

|  |
| --- |
| for x in 1 2 4 8 16 32  do  sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root --mysql-password=wku2000 --db-driver=mysql prepare    sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root --mysql-password=yourrootsqlpassword --max-time=60 --oltp-read-only=on --max-requests=0 --num-threads=$x run > sqltest$x.txt  s  ysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root --mysql-password=yourrootsqlpassword --max-time=60 --oltp-read-only=on --max-requests=0 --num-threads=$x run >> sqltest$x.txt    sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root --mysql-password=yourrootsqlpassword --max-time=60 --oltp-read-only=on --max-requests=0 --num-threads=$x run >> sqltest$x.txt    sysbench --test=oltp --mysql-db=test --mysql-user=root --mysql-password=wku2000 --db-driver=mysql cleanup  done |

**Figure 48** Base OS Database Input Test Script

Test Output – Below is an example of the output data for the CPU test for 32 threads using a table size of 1,000,000 in the base OS environment (Figure 49).

|  |
| --- |
| OLTP test statistics:  queries performed:  read: 153328  write: 0  other: 21904  total: 175232  transactions: 10952 (182.30 per sec.)  deadlocks: 0 (0.00 per sec.)  read/write requests: 153328 (2552.15 per sec.)  other operations: 21904 (364.59 per sec.)  Test execution summary:  total time: 60.0780s  total number of events: 10952  total time taken by event execution: 1921.0467  per-request statistics:  min: 11.20ms  avg: 175.41ms  max: 637.70ms  approx. 95 percentile: 252.26ms  Threads fairness:  events (avg/stddev): 342.2500/3.63  execution time (avg/stddev): 60.0327/0.02  sysbench 0.4.12: multi-threaded system evaluation benchmark  Running the test with following options:  Number of threads: 32  Doing OLTP test.  Running mixed OLTP test  Doing read-only test  Using Special distribution (12 iterations, 1 pct of values are returned in 75 pct cases)  Using "BEGIN" for starting transactions  Using auto\_inc on the id column  Threads started!  Time limit exceeded, exiting...  (last message repeated 31 times)  Done. |

**Figure 49** Database Calculation Output Example

After running three tests per thread, below is the table for database output for in the base OS environment (Figure 50).

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Threads** | **Test** | **Transactions per Sec.** | **Average** |
| **1** | a | 99.74 | **97.58** |
|  | b | 96.63 |  |
|  | c | 96.37 |  |
| **2** | a | 174.09 | **171.87** |
|  | b | 172.64 |  |
|  | c | 168.88 |  |
| **4** | a | 233.64 | **212.61** |
|  | b | 205.91 |  |
|  | c | 198.28 |  |
| **8** | a | 206.65 | **190.08** |
|  | b | 185.19 |  |
|  | c | 178.4 |  |
| **16** | a | 182.3 | **168.98** |
|  | b | 164.29 |  |
|  | c | 160.35 |  |
| **32** | a | 189.35 | **175.7966667** |
|  | b | 171.47 |  |
|  | c | 166.57 |  |

**Figure 50** Database Output Test Summary in Base OS Environment

After the average transaction was calculated in the base OS environment, it was plotted using a line graph (Figure 51). The trend indicates a positive linear relationship between the transaction per second and the number of thread counts from 1-4. There is a negative relationship of transactions for thread count from 8 threads to 16 threads. Thirdly, there is roughly a neutral linear for thread count of 32. Once again, this is to be expected since the Raspberry PI has a Quad-Core processer; process that uses threads greater than 4 is expected to decrease then even out. In summary, using 4 treads returns the best transactions per second.

**Figure 51** Base OS Thread Count and Transactions Per Second

*Network*

Testing Procedures – In order to test the network speed of the mini WKU Wiki website, which is being hosted on the Raspberry PI, several speed test were conducted to ping the site's IP from different places around the globe (using different nodes on a small bot net and remote accessing them.)

Test Input – Using the command terminal, the PI’s IP address was ping. 8 different 32-byte pings were performed from each location in which the round trip was recorded then averaged. Furthermore, from each location, the complete Index page for the mini WKU Wiki was downloaded to calculate the load time.

Test Output – Calculation from the three locations were recorded. The max ping indicates the longest wait ping time that occurred in all the 8 testes. The average ping time is the total average of the ping speed from all 8 testes. Furthermore, the max load time indicates the long time that occurred in all 8 testes to download a website completely. Lastly, the average load time is calculated by averaging the total download time for all 8 tests per location (Figure 52).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Location | Max Ping | Average Ping | Max Load Time (207.6KB) | Average Load Time (207.6kB) |
| WKU | 14ms | 12ms | 223ms | 213ms |
| New York | 17ms | 15ms | 401ms | 322ms |
| California | 22ms | 21ms | 579ms | 556ms |
| Sydney, Australia | 34ms | 31ms | 1.43s | 1.32s |

**Figure 52** Network Ping Summary (Remotely access the nodes will have slightly lowered the download speed from these locations.)

Analysis – As far as network performance goes, the PI is operating at an excellent speed. Due to the minimal traffic, the website is running as quickly as you would expect. While the PI is up, the response time is very fast, so the network performance speed overall certainly **exceeds expectations** considering where the PI is located.

Improvement – One way that the network response time could be improved is by cutting down on the unnecessary libraries that are being downloaded. The **performance requirements** are all met for speed, and the issues are minimal, but 82.8kB (roughly 40%) of the total 207.6kB website is being used for special font libraries. We are using the special fonts in some places, but perhaps there is a way to minimize the amount of space that is taking up by isolating the specific ones we are using to a library.

**Security Verification**

*Security Questions*

One method of user verification that will be implemented are security questions. These will be three short answer questions that the website will prompt the user with when they first register an account. The user will be able to choose from a list of different questions to answer, and then enter an answer. At distant intervals, or perhaps when successfully logging on (entering correct password) from a new visiting IP address, the user will again be prompted with the security questions and must present answers that are the same as the ones they originally used. They will also be used if the user attempts to changed their password or claims that they forgot it. These security questions are thus as important as the passwords themselves, so will also be hashed with salt and stored securely on the server. Behind the hashed password, this will be the final method of secure user verification.

**Operation and Maintenance (Release)**

*Operation*

Currently, the PI is operational with all the libraries functioning that we currently require. As we progress, the PI's operations will be expanded to meet our current requirements from our client as well as any future requirements once requested.

*Maintenance*

As the PI's operational status should be online as much as possible, maintenance to the PI will be conducted at off hours if it requires the service to be down. This maintenance could include making backups of the current database, backups of the server scripts and even physical maintenance such as dusting out the case.

**Requirement Conclusion**

Team two will use the iteration process to for the to develop the mini WKU Wiki. Functional and nonfunctional requirements of the system will be review with the client to make sure they are fulfilled. If the requirements are not addressed, team two will go through the iterative cycle to address the client’s requests. The testing cycle will be performed at the review stage (last state of the iterative process) of the iterative to determine if the group will have to rework the system, progress to the next development phase, or deliver the system to the client.

**Appendix**

*PI Setup*

1) The first thing we did to setup the PI was to secure it in its compact case, this came with the starter kit linked on the course website.

2) Following that, we plugged in all the components to set it up, such as a mouse and a keyboard.

3) Setup the default operating system - Raspbian.

4) Setup the LAMP Server via command line and ran a restart on the pi.

5) Installed phpMyAdmin via command line.

6) Following all those steps, we needed to run a test on the apache server, to see if we could reach the localhost default index.html. We were able to access the default index.html but were unable at the time to create files in the "www" directory as the default PI user. After an hour or so toying around with it, we came across the solution. By default, the www folder belongs to the raspberry pi user 'root'. Any attempts to modify/add/remove files in that folder or any of its directories would be blocked if you're not logged in as root, even if you're logged in a user with root access. The solution was to set the owner of the www folder and any of its directories to the default pi user, and grant the default pi user default root access/super user access via command line using the 'chown' command. Once finished with these steps, we were able to replace the default index.html with index.php and test a hello world echo PHP script.

*Remote Access PI via PuTTY(SSH)*

In order for us all to have access to the PI located at an off-campus apartment, we had to do multiple tasks as well as follow a multitude of guides. The first step we took was to test the connection to the pi from a windows computer on the same network. To accomplish this, we found the pi's local IP address and used PuTTY to connect to it. At first, the connection failed due to what it claimed to be "authentication issues" which came out to be the fact that by default, the pi's SSH access is disabled in the preferences. After enabling SSH on the PI and running a restart on it, the PuTTY connection was successful. Using PuTTY we now have access to the PI's command line. At this point, the PI was moved to a more suitable location in the apartment that will allow it to run free from disturbances, using only the power supply and an internet connection. Given the fact that creating directories and files from the pi command line can be a task in its own, we decided to also use window's remote desktop connection. This will allow us to control the pi itself from our windows machines, using the PI's default operating system interface. At this point, we have access to the command line as well as the pi's interface which allows us to create files, manage directories, manage the database via phpMyAdmin as well as to test files straight from the PI via localhost.

*Viewing Source Code Directly on PI*

All of our code is being ran from the pi, and is located in the "/var/www/html" directory. To test our site, simply open a browser from the PI and type localhost. This will bring you to the default index.php file. Later on with the project we hope to open the pi's html folder to the web so you can just use the pi's IP address to test the website itself instead of using localhost.

*Build Instructions*

To build our source code on a local machine, download our source code and place everything inside the 'html' folder into the root of your apache server (Ex: xampp/htdocs). Start the apache and mysql server. Navigate to a browser and type "localhost". This will pull up the home page.

*Remote Login Instructions*

Putty : To connect via putty to our command line, open putty and under hostname enter '104.145.83.147' with port 22. This will pull up a login prompt, login as user: pi with password 'wku##2000'. From here, the command line should load up and be fully functional to you remotely.

Remote Connection : On any windows device, go to the search bar and type 'Remote Desktop Connection'. This will pull up a window asking for the hostname. Type '104.145.83.147'. Hit connect, clicking yes to any prompts messaging you about security. Xrdp will pull up at this point, login using user 'pi' and password 'wku##2000'.

Live login : To connect to the website, you can use a premade account we have setup. Username/email : [test@gmail.com](mailto:test@gmail.com) password is 'password'.

PhpMyAdmin Login : (104.145.83.147/phpmyadmin) username : root, password 'wku##2000'

*Docker Setup*

1) To download and install Docker onto the Pi, the command below was entered into the shell.

|  |
| --- |
| curl -sSL https://get.docker.com | sh |

2) A Dockerfile was then created that pulls from Docker's official library a PHP image packaged with an Apache server. This file is created and located in the directory containing all of the PHP and front end code. The Dockerfile can be seen below. This was used to build an image containing the web server and web files.

|  |
| --- |
| FROM php:7.0-apache  COPY . /var/www/html/ |

3) The next step is to build the MySQL database. The following command was used to build the database container. A sql dump file was used to import the data.

|  |
| --- |
| sudo docker run --name sqlserver -d -p 3306:3306 -v /home/pi/.local/share/mysql:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=wku2000 tobi312/rpi-mysql |

4) To access the containers from Docker Hub. use the following commands.

|  |
| --- |
| docker pull mschap09/wikiwebsite  docker pull mschap09/sqlserver |

5) To run the website the following commands should get the containers linked and running.

|  |
| --- |
| sudo docker run --name sqlserver -d -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=wku2000 mschap09/sqlserver  sudo docker run -it --link sqlserver --name wikiwebsite -d -p 80:80 mschap09/wikiwebsite |

6) The deployment of the containers and its content can then be viewed in the web browser by going to the url: 104.145.83.147

*Github Setup*

1) To setup a Github repository for the team, one was created on a personal account of a team member who already had an account. The Github repository is created under the user account *michelleschap* and the repository is named *WikiClone*.

2) The local repository is located within the team's shared OneDrive folder. All team members are listed as collaborators for the repository so that all will have access to the project files.

3) The html folder located locally on the Raspberry Pi found under the path *var/www/html* was moved into the repository and holds all the technical files needed for the website. The project was first being developed on the Pi's system moving files through Filezilla but now will be developed in containers using Github.

*Source Code*

/VARS/vars\_gen.php

This file currently contains the salted characters we use for hashing.

|  |
| --- |
| <?php  $db\_salt\_1 = "aarh8as73akdgja9s";  $db\_salt\_2 = "zdfh98dw4ykfhsi8hs";  ?> |

/VARS/vars\_dbconnection.php

Contains the variables we use to execute sql statements.

|  |
| --- |
| <?php  $db\_hostname = "localhost";  $db\_username = "dbcontroller";  $db\_password = "wku2000";  $db\_database = "wikipedia";  ?> |

/PHP/sAuth/db\_conn.php

This file is what we use to make a MySQL connection to the database, using the variables from vars\_dbconnection.php. This file will be required throughout nearly every php file on our site that makes changes in the MySQL database. If the connection is a success, it sets a variable to true that is what we use to establish whether or not we're connected.

|  |
| --- |
| <?php  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_dbconnection.php";  $dbConnection = false;  // Create connection  $conn = new mysqli($db\_hostname, $db\_username, $db\_password, $db\_database);  // Check connection  if ($conn->connect\_error) {  die("Connection failed: " . $conn->connect\_error);  }else{  $dbConnection = true;  }  ?> |

/PHP/sAuth/destroySession.php

This file is what kills the cookies as well as the current session that's holding variables. This is used whenever the user logs out and we don't need to carry their information anymore.

|  |
| --- |
| <?php  session\_start();  session\_unset();  session\_destroy();  $sess\_username = null;  $sess\_pass = null;  $sess\_acclv = null;  $sess\_name = null;  if (ini\_get("session.use\_cookies")) {  $params = session\_get\_cookie\_params();  setcookie(session\_name(), '', time() - 42000,  $params["path"], $params["domain"],  $params["secure"], $params["httponly"]  );  }  ?> |

/PHP/sAuth/getcurrentSess.php

This file delivers the current session variables using a delimited echo to the javascript functions that need that information.

|  |
| --- |
| <?php  //Start a session that can hold login variables if they are successful.  session\_start();    $sess\_username = $\_SESSION['username'];  $sess\_pass = $\_SESSION['pass'];  $sess\_acclv = $\_SESSION['acclv'];  $sess\_name = $\_SESSION['name'];    //Use '~~' as a delimiter to deliver the session variables  echo $sess\_username."~~".$sess\_pass."~~".$sess\_acclv."~~".$sess\_name;  ?> |

/PHP/register.php

This file is the core database file for the registration function, calling on db\_conn.php as well as vars\_gen.php, this file first checks that the post variables delivered to it exist. Next, it binds the post variables to local variables to allow for easy access to the values. Then it checks that we have a database connection then runs a SQL statement verifying that the requested registration email doesn't already have an account associated with it in the database, if it does, it returns an error. If it's been established that the email doesn't already exist, it salts the password using random characters stored in vars\_gen.php. After salting the values that are sensitive, we use the hashing method 'ripemd160' on the password. Next, it runs a prepared statement that inserts the first name, last name, password and email into the account database. With this being a success, it stores the recently registered values as session variables to allow ease of access throughout the site. This script then echos back a response of '1' to the ajax call which understands that as a successful registration and redirects the user back to the home page, which on its load, checks for the session variables that were just set and automatically logs the user in.

|  |
| --- |
| <?php  error\_reporting(E\_ALL);  ini\_set('display\_errors', 1);    require\_once $\_SERVER['DOCUMENT\_ROOT']."/PHP/sAuth/db\_conn.php";  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_gen.php";    //Start a session that can hold login variables if they are successful.  session\_start();    /\*echo "<!DOCTYPE html>  <html>  <head>  <title>Register</title>  </head>  <body>  <form method='post' action='register.php'>  Firstname:<br>  <input type='text' name='fname'><br>  Lastname:<br>  <input type='text' name='lname'><br>  Email:<br>  <input type='text' name='email'><br>  Password:<br>  <input type='text' name='password'><br>  <input type='submit' value='Register'>  </form>    </body>  </html>"\*/;  if(isset($\_POST['fname']) && isset($\_POST['lname']) && isset($\_POST['email']) && isset($\_POST['password'])){  //Bind post variables to local variables  $fname = $\_POST['fname'];  $lname = $\_POST['lname'];  $email = $\_POST['email'];  $password = $\_POST['password'];  $accExists = false;  $accountLevel = 0;    //Db connection exists in db\_conn.php - this is a required file that connects to the DB  if($dbConnection){  //check if email already exists in the database  $stmt = $conn->prepare('SELECT \* FROM account WHERE email = ?');  $stmt->bind\_param('s', $email);  $stmt->execute();  $result = $stmt->get\_result();  while ($row = $result->fetch\_assoc()) {  $accExists=true;  break;  }  if($accExists){  Echo "2";  }else{  //Create the account  //Salt the password  $password = $db\_salt\_1.$password.$db\_salt\_2;  //RIPEMD Hash  $password = hash('ripemd160', $password);  //Insert into DB  //prepare and bind  $stmt = $conn->prepare("INSERT INTO account (first\_name, last\_name, email, user\_level, password) VALUES (?, ?, ?, ?, ?)");  $stmt->bind\_param("sssss", $fname, $lname, $email, $accountLevel, $password);  //execute  $stmt->execute();      //Get User ID from the db so we can auto login and store the id for page creation  //in the session variables.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  $stmt = $conn->prepare('SELECT \* FROM account WHERE email = ? AND password = ?');  $stmt->bind\_param('ss', $email, $password);  $stmt->execute();    $result = $stmt->get\_result();  while ($row = $result->fetch\_assoc()) {  $user\_id = $row["user\_id"];  break;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  $user\_id = "";      $\_SESSION['userid'] = $user\_id;  $\_SESSION['username'] = $email;  $\_SESSION['pass'] = $password;  $\_SESSION['acclv'] = $accountLevel;  $\_SESSION['name'] = $fname;    echo "1";  }  }  }else{  echo "3";  }    ?> |

/PHP/login.php

This file calls up the required files db\_conn.php as well as vars\_gen.php. It then starts a session allowing it to use the session variables. It then checks if the variables in the post consist of the required information (email and pass). If it contains the required post information, it stores the post variables as local variables for ease of access. Next, it takes the supplied password and salts it with the same salted characters we used during registration. It then hashes this password. Next, it executes a prepared statement that queries the database for a record in the account table that has both the same email and hashed password as the ones supplied in the login attempt. If it does, it uses the result associative array and stores the user\_level and first\_name. Then it runs an if statement that executes if the login was successful, in this if statement it sets the session variables that will need to be used later on throughout the site. If the login was successful, it then echos back a '1' to the ajax js call which the javascript file understands as a successful login.

|  |
| --- |
| <?php  error\_reporting(E\_ALL);  ini\_set('display\_errors', 1);    require\_once $\_SERVER['DOCUMENT\_ROOT']."/PHP/sAuth/db\_conn.php";  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_gen.php";    //Start a session that can hold login variables if they are successful.  session\_start();      if(isset($\_POST['email']) && isset($\_POST['pass'])){  //Bind post variables to local variables  $email = $\_POST['email'];  $pass = $\_POST['pass'];  $accountLevel = 0;  $loginSuccessful = false;  $firstname = "";    //Salt & hash inc password to compare to the password in DB  $pass = $db\_salt\_1.$pass.$db\_salt\_2;  $pass = hash('ripemd160', $pass);    //Db connection exists in db\_conn.php - this is a required file that connects to the DB  if($dbConnection){  //check if email already exists in the database  $stmt = $conn->prepare('SELECT \* FROM account WHERE email = ? AND password = ?');  $stmt->bind\_param('ss', $email, $pass);  $stmt->execute();  $result = $stmt->get\_result();  while ($row = $result->fetch\_assoc()) {  $loginSuccessful=true;  $accountLevel = $row["user\_level"];  $name = $row["first\_name"];  $userId = $row["user\_id"];  break;  }    if($loginSuccessful){  //Post session variables  $\_SESSION['username'] = $email;  $\_SESSION['pass'] = $pass;  $\_SESSION['acclv'] = $accountLevel;  $\_SESSION['name'] = $name;  $\_SESSION['userid'] = $userId;  echo "1";  }else{  //Login failed  echo "2";  }  }  }else{  //Not all post variables present  echo "3";  }    ?> |

PHP/getUserCount.php

This file is a simple SQL execution that gets the total amount of users stored in the account table

|  |
| --- |
| <?php  require\_once $\_SERVER['DOCUMENT\_ROOT']."/PHP/sAuth/db\_conn.php";  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_gen.php";    session\_start();    $count = 0;    if($dbConnection){  $fakeNum = 1;    //PREPARE  $stmt = $conn->prepare("SELECT COUNT(\*) FROM account WHERE 1 = ?");  //BIND  $stmt->bind\_param("i", $fakeNum);  //EXECUTE  $stmt->execute();    $col1 = null;  $stmt->bind\_result($col1);    while ($stmt->fetch()) {  $count = "{$col1}";  }    echo $count;  }  ?> |

PHP/getTopPage.php

This file simply runs a sql statement to fetch all the rows in the page table and decide which one has the highest number of views. This is used for information references on the admin page.

|  |
| --- |
| <?php  require\_once $\_SERVER['DOCUMENT\_ROOT']."/PHP/sAuth/db\_conn.php";  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_gen.php";    session\_start();    $count = 0;    if($dbConnection){  $fakeNum = 1;  //get page\_id so we can use to match it in the subsection table  $stmt = $conn->prepare('SELECT \* FROM page WHERE 1 = ?');  $stmt->bind\_param('i', $fakeNum);  $stmt->execute();    $top\_page\_name = "";  $top\_page\_views = 0;    $result = $stmt->get\_result();  while ($row = $result->fetch\_assoc()) {  $x = $row["views"];  if($x>$top\_page\_views){  $top\_page\_views = $x;  $top\_page\_name = $row["title"];  }  }    echo $top\_page\_views."~split~".$top\_page\_name;  }  ?> |

PHP/createpage.php

This page takes in the data from the create page html file and inserts it into the proper tables using a prepared statement. The first prepared statement stores the title, the second gets the page\_id from the table it just inserted data to and the third uses the id to match the information in the topics to the title.

|  |
| --- |
| <?php  require\_once $\_SERVER['DOCUMENT\_ROOT']."/PHP/sAuth/db\_conn.php";  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_gen.php";    error\_reporting(E\_ALL);  ini\_set('display\_errors', 1);    //Start a session that can hold session variables if they are successful.  session\_start();    if(isset($\_POST['pageSubject']) && isset($\_POST['pageSupportingTopic']) && isset($\_POST['pageSupportingContent']) && isset($\_POST['pageSupportingTags'])){  $page\_subject = $\_POST['pageSubject'];  $page\_topic = $\_POST['pageSupportingTopic'];  $page\_content = $\_POST['pageSupportingContent'];  $page\_tags = $\_POST['pageSupportingTags'];  $user\_id = "";  $page\_id = "";  if($dbConnection){  //The js already verifies the user is logged in, so we don't have to do it here too  //Get the user id from session variables  $user\_id = $\_SESSION['userid'];  if(isset($user\_id)){  //They're logged in    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Store information in the page table  //PREPARE  $stmt = $conn->prepare("INSERT INTO page (title, user\_id, tags) VALUES (?, ?, ?)");  //BIND  $stmt->bind\_param("sss", $page\_subject, $user\_id, $page\_tags);  //EXECUTE  $stmt->execute();  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //get page\_id so we can use to match it in the subsection table  $stmt = $conn->prepare('SELECT \* FROM page WHERE user\_id = ? AND title = ?');  $stmt->bind\_param('ss', $user\_id, $page\_subject);  $stmt->execute();    $result = $stmt->get\_result();  while ($row = $result->fetch\_assoc()) {  $loginSuccessful=true;  $page\_id = $row["page\_id"];  break;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Store information in subsection table  //version is 0 beacuse this is create page  $page\_version = "0";  //PREPARE  $stmt = $conn->prepare("INSERT INTO subsection (section\_name, version, words, page\_id) VALUES (?, ?, ?, ?)");  //BIND  $stmt->bind\_param("ssss", $page\_topic, $page\_version, $page\_content, $page\_id);  //EXECUTE  $stmt->execute();  //success  echo "1";    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  }else{  //echo "2";  }  }  } |

/viewPage.php

This page takes in the pageid as a global variable and gets the information from the database matching that page id and pieces it together to form the page content.

|  |
| --- |
| <?php  require\_once $\_SERVER['DOCUMENT\_ROOT']."/PHP/sAuth/db\_conn.php";  require\_once $\_SERVER['DOCUMENT\_ROOT']."/VARS/vars\_gen.php";    error\_reporting(E\_ALL);  ini\_set('display\_errors', 1);    //Start a session that can hold session variables if they are successful.  session\_start();    $pg\_id = 0;  $pg\_title = "";  $Body = "";    if($dbConnection){  if(isset($\_GET['pageID'])){  //Get page id from Get  $pg\_id = $\_GET["pageID"];    //Load and populate the page title  $stmt = $conn->prepare('SELECT \* FROM page WHERE page\_id = ?');  $stmt->bind\_param('i', $pg\_id);  $stmt->execute();  $result = $stmt->get\_result();  while ($row = $result->fetch\_assoc()) {  $pg\_title = $row["title"];  break;  }        //Load and populate the body  $stmt = $conn->prepare('SELECT \* FROM subsection WHERE page\_id = ?');  $stmt->bind\_param('i', $pg\_id);  $stmt->execute();  $result = $stmt->get\_result();  $count = 0;  while ($row = $result->fetch\_assoc()) {  $count++;  $value\_Title = $row["section\_name"];  $value\_Body = $row["words"];    $topic\_title = "<div class='card'>  <div class='card-header'>  <a data-toggle='collapse' href='#card".$count."' class='heading mb-0'>".$value\_Title."</a>  </div>  </div>";  $topic\_body = "<div id='card".$count."' class='container collapse'>  <p class='paragraph'>".$value\_Body."</p>  </div>";    //Add both to the generated body  $Body .= $topic\_title.$topic\_body;  }      //Increase Page Count  $stmt = $conn->prepare('UPDATE page SET views = views + 1 WHERE page\_id = ?');  $stmt->bind\_param('i', $pg\_id);  $stmt->execute();    }else{  Echo "No Valid Page ID";  }  }    echo"<!DOCTYPE html>  <html lang='en'>  <head>  <meta charset='utf-8'>  <meta http-equiv='X-UA-Compatible' content='IE=edge'>  <meta name='viewport' content='width=device-width, initial-scale=1'>  <!-- The above 3 meta tags \*must\* come first in the head; any other head content must come \*after\* these tags -->  <title>Helm Library</title>    <!-- Bootstrap -->  <link href='css/bootstrap.min.css' rel='stylesheet'>  <!--Create custom javascripts for this page - Includes login Ajax function-->  <script src='js/login.js'></script>  <script src='js/viewPage.js'></script>      </head>    <!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->  <!--onload calls the java script when the webpage is loaded-->  <body onload='loadFunctions()'>  <!--Custom CSS-->  <style type='text/css'>    /\*Removes underline and set color to link as black\*/  a:link {color: black; text-decoration: none} /\* unvisited link \*/  a:visited {color: black; text-decoration: none} /\* visited link \*/  a:hover {color: black; text-decoration: none} /\* mouse over link \*/  a:active {color: black; text-decoration: none}  .heading {  font-size: 32px;  font-style: italic;  display: block;  }  .paragraph{  text-align: justify;  /\*Indent paragraph text\*/  text-indent: 3em  }  </style>    <!--Navigation-->  <!--Add navbar-light bg-faded to make navbar visiable-->  <nav class='navbar align-items-end navbar-light bg-faded'>  <!-- navBar Name-->  <!-- <a class='navbar-brand' href='#'>WKU Wiki</a> -->    <div class='btn-group ml-auto'>  <!--Button for dropdown-->    <button type='button' class='btn btn-secondary dropdown-toggle' data-toggle='dropdown' aria-haspopup='true' aria-expanded='false' id='btnHolderName'><i class='fa fa-user-secret' aria-hidden='true'></i></button>  <div class='dropdown-menu dropdown-menu-right'>  <div class = 'mx-2' >  <input class='dropdown-item form-control' type='text' id='Email' placeholder='email'>  <input class='dropdown-item form-control' type='password' id='Password' placeholder='password' >  <button class='dropdown-item ' type='button' id='LogIn' onclick='submitLogin()'>Log In</button>  <button class='dropdown-item' type='button' id='SignUp' onclick='signUp()'>Sign Up</button>  <button class='dropdown-item' type='button' id='CreatePage' onclick='createPageLink()' style='display: none;'>Create Page</button>  <button class='dropdown-item example' type='button' id='SignOut' style='display: none;' onclick='logOut()'>Sign Out</button>  </div>  </div>  </div>  </nav>    <!--  !!!Notes!!!  Make sure you rename href in the header and the id in the body  both 'xxx' should be replaced  'Header Text' should be replaced by the name of the header    This is the heading  <div class='card'>  <div class='card-header'>  <a data-toggle='collapse' href='#xxx' class='heading mb-0'>Header Text</a>  </div>  </div>    Body Begins here  <div id='xxx' class='container collapse'>  <p class='paragraph'>Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of 'de Finibus Bonorum et Malorum' (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, 'Lorem ipsum dolor sit amet..', comes from a line in section 1.10.32.</p>  </div>  -->    <!--Topic header-->  <h1 id='pg\_title'><b>".$pg\_title."</b></h1>  <div class='container mt-3'>  </div>  ".$Body."    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->  <script src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script>  <!-- Include all compiled plugins (below), or include individual files as needed -->  <script src='js/bootstrap.min.js'></script>    </body>  </html>";  ?> |

js/login.js

This script handles all the javascript functions for the login page. Some of the functions include the submitLogin function that posts the data it takes from the form controls to the login.php file, the destroySession function that is called to log the user out. This function uses an ajax call to the destroySession.php file that closes the session and unsets all the variables. This script also includes multiple other functions that alter the current html of the page.

|  |
| --- |
| var glob\_firstname;  function submitLogin()  {  var failed = false;  var failedmessage = "";    var emailval = document.getElementById('Email').value;  var passval = document.getElementById('Password').value;    if(emailval.length==0){  failed = true;  failedmessage="The email field is empty";  }    if(passval.length==0){  failed = true;  failedmessage="The password field is empty";  }    if(failed==false){  //The fields are not empty -- submit to php login script.  //Create the xhttp request  var xhttp = new XMLHttpRequest();  var params = 'email='+emailval+'&pass='+passval;  xhttp.open('POST', '<http://104.145.83.147/PHP/login.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  if(this.responseText == "1"){  //1 Indicates a successful login  //Verify that the session variables were accepted  isLoggedIn();  }else if(this.responseText=="2"){  //2 Indicates incorrect login  alert("Invalid Login Combination!\nIf you've forgot your username and password, go to <http://104.145.83.147/PHP/accountforgot.php>");  }else if(this.responseText=="3"){  //Shouldn't reach here - failsafe  alert("Not all post vars collected");  }else{  alert(this.responseText);  }  }  };  xhttp.send(params);  }else{  alert(failedmessage);  }  }  function display(x)  {  x.style.display = 'block';  }  function hide(x)  {  x.style.display = 'none';  }  function isset(variable)  {  if(typeof(variable) != "undefined" && variable !== null && variable!="") {  return true;  }  }    function signUp()  {  location.href = "signUp.html";  }    function createPageLink()  {  location.href = "createPage.html";  }      //The onload functions  function loadFunctions()  {  getFocus();  if(isLoggedIn()){  isLoggedInModifyHomeDrop();  }else{  isLoggedOutModifyHomeDrop();  }  }    //IsLoggedIn  function isLoggedIn()  {  var response = "";  var sess\_email = "";  var sess\_pass = "";  var sess\_accLevel = "";  var sess\_name = "";    var xhttp = new XMLHttpRequest();  xhttp.open('POST', '<http://104.145.83.147/PHP/sAuth/getcurrentSess.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  response = this.responseText;  var res = response.split("~~");  sess\_email = res[0];  sess\_pass = res[1];  sess\_accLevel = res[2];  sess\_name = res[3];  glob\_firstname = res[3];  if (isset(sess\_email) && isset(sess\_pass) && isset(sess\_accLevel) && isset(sess\_name)){  isLoggedInModifyHomeDrop();  }else{  return false;  }  }  };  xhttp.send(null);  }    //Destroy Session  function destroySession()  {  var xhttp = new XMLHttpRequest();  xhttp.open('POST', '<http://104.145.83.147/PHP/sAuth/destroySession.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  }  };  xhttp.send(null);    } |

Js/register.js

This javascript file holds all the functions associated with the register page. Some of these functions include: checking the elements to make sure they're all filled out as required, as well as posting the values of those elements to the register.php file and handling the result echo from that file.

|  |
| --- |
| function submitReg()  {  var failed = false;  var failedmessage = "";    //get the values from the page  //var name = document.getElementById("txtName").value;  var fname = document.getElementById("firstName").value;  var lname = document.getElementById("lastName").value;  var email = document.getElementById("email").value;  var pass = document.getElementById("password").value;  var confirmpass = document.getElementById("confirmPassword").value;    if(fname.length==0){  failed = true;  failedmessage="The first name field is empty";  }    if(lname.length==0){  failed = true;  failedmessage="The last name field is empty";  }    if(email.length==0){  failed = true;  failedmessage="The email field is empty";  }    if(pass.length==0){  failed = true;  failedmessage="The password field is empty";  }    if(confirmpass.length==0){  failed = true;  failedmessage="The confirm password field is empty";  }    if(confirmpass != pass){  failed = true;  failedmessage="The password fields do not match up!";  }    if(failed==false){  //The fields are not empty -- submit to php registration script.  //Create the xhttp request  var xhttp = new XMLHttpRequest();  var params = 'fname='+fname+'&lname='+lname+'&email='+email+'&password='+pass;  xhttp.open('POST', '<http://104.145.83.147/PHP/register.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  if(this.responseText == "1"){  //1 Indicates a successful registration  //Redirect to homescreen  location.href = "<http://104.145.83.147/>";  }else if(this.responseText=="2"){  //2 Indicates that the email already exists  alert("There's an email already associated with this account! (Error: 1002")  }else if(this.reponseText=="3"){  //3 Indicates error occured.  alert("Unhandled Exception - Please try again later! (Error: 1003)")  }else{  //Shouldn't reach here - failsafe  alert("Unhandled Exception - Please try again later! (Error: case else - 1004+)" + this.responseText)  }  }  };  xhttp.send(params);  }else{  alert(failedmessage);  }  } |

js/viewPage.js

This file containts page specific modifications. May override some functions of the login.js file to better change the onload methods as well as other specific actions

|  |
| --- |
| //a method that focus the user's cuser to the search box  function getFocus()  {  document.search\_box.search.focus();  }    function signUp()  {  location.href = "signUp.html";  }    //Log in button toggle  function logInToggle()  {  var a = document.getElementById('LogIn');  var b = document.getElementById('LogOut');  display(a);  display(b);  }    //Function to toggle display of the buttons  function display(x)  {  if (x.style.display === 'none') {  x.style.display = 'block';  } else {  x.style.display = 'none';  }  }  function isLoggedInModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    display(x);  display(y);  hide(z);  hide(a);  hide(b);  hide(c);  d.innerText = glob\_firstname + " ";  }    function isLoggedOutModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    hide(x);  hide(y);  display(z);  display(a);  display(b);  display(c);  //Adds user icon on log in button  d.innerHTML = '<i class = "fa fa-user" aria-hidden="true"></i> ';  // d.innerText = "Account ";  }    //logout function  //@Override login.js  function logOut()  {  //Destroy Session Variables  destroySession();  isLoggedOutModifyHomeDrop();  }    //The onload functions  //@Override login.js  function loadFunctions()  {  if(isLoggedIn()){  isLoggedInModifyHomeDrop();  }else{  isLoggedOutModifyHomeDrop();  }  } |

js/loginpage.js

This file containts page specific modifications. May override some functions of the login.js file to better change the onload methods as well as other specific actions

|  |
| --- |
| //a method that focus the user's cuser to the search box  function getFocus()  {  document.search\_box.search.focus();  }    function isLoggedInModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    display(x);  display(y);  hide(z);  hide(a);  hide(b);  hide(c);  d.innerText = glob\_firstname + " ";  }    function isLoggedOutModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    hide(x);  hide(y);  display(z);  display(a);  display(b);  display(c);  //Adds user icon on log in button  d.innerHTML = '<i class = "fa fa-user" aria-hidden="true"></i> ';  // d.innerText = "Account ";  }    //logout function  function logOut()  {  //Destroy Session Variables  destroySession();  isLoggedOutModifyHomeDrop();  }    //View DSU  function viewDSU()  {  location.href = "<http://104.145.83.147/viewPage.php?pageID=5>";  } |

js/createpage.js

This file takes in all the information on the create page html file and processes it and posts it to the php file that handles the back end transaction with the database to store the information it passes. This page also handles many other small page modifications that are specific to this page.

|  |
| --- |
| function submitPage()  {  //run a double check to make sure they're currently logged in and haven't timed out  if(isLoggedIn()==false){  redirHome();  }else{  var failed = false;  var failedmessage = "";    var pageSubject = document.getElementById('pageSubject').value;  var pageSupportingTopic = document.getElementById('supportingTopic').value;  var pageSupportingContent = document.getElementById('supportingContent').value;  var pageSupportingTags = document.getElementById('contentTags').value;    if(pageSubject.length==0){  failed = true;  failedmessage="The Page Subject field is empty";  }    if(pageSupportingTopic.length==0){  failed = true;  failedmessage="The Supporting Topic field is empty";  }    if(pageSupportingContent.length==0){  failed = true;  failedmessage="The Supporting Content field is empty";  }    if(pageSupportingTags.length==0){  failed = true;  failedmessage="The Tags field is empty, users won't be able to find your page if there are no search tags! You're required to specify at least 1 tag.";  }    if(failed==false){  //The fields are not empty -- submit to create page script.  //Create the xhttp request  var xhttp = new XMLHttpRequest();  var params = 'pageSubject='+pageSubject+'&pageSupportingTopic='+pageSupportingTopic+'&pageSupportingContent='+pageSupportingContent+'&pageSupportingTags='+pageSupportingTags;  xhttp.open('POST', '<http://104.145.83.147/PHP/createpage.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  if(this.responseText == "1"){  alert("Page Created");  redirHome();  }else if(this.responseText=="2"){  alert(this.responseText);  }else{  alert(this.responseText);  }  }  };  xhttp.send(params);  }else{  alert(failedmessage);  }  }  }    //a method that focus the user's cuser to the search box  function getFocus()  {  var firstField = document.getElementById('pageSubject');  firstField.focus();  }    function isLoggedInModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    display(x);  hide(y);  hide(z);  hide(a);  hide(b);  hide(c);  d.innerText = glob\_firstname + " ";  }    function isLoggedOutModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    hide(x);  hide(y);  display(z);  display(a);  display(b);  display(c);  d.innerText = "Account ";  }    //logout function  function logOut()  {  //Destroy Session Variables  destroySession();  isLoggedOutModifyHomeDrop();  redirHome();  }    //User can no longer create pages because they've been logged out  function redirHome()  {  location.href = "index.html";  } |

js/adminStatsPage.js

This file runs a series of ajax calls to get the information about the current state of the service, such as the amount of users as well as the most viewed pages, also handles many other minor page specific modifications.

|  |
| --- |
| function isLoggedInModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    display(x);  display(y);  hide(z);  hide(a);  hide(b);  hide(c);  d.innerText = glob\_firstname + " ";  }    function isLoggedOutModifyHomeDrop()  {  var x = document.getElementById('SignOut');  var y = document.getElementById('CreatePage');  var z = document.getElementById('SignUp');  var a = document.getElementById('LogIn');  var b = document.getElementById('Email');  var c = document.getElementById('Password');  var d = document.getElementById('btnHolderName');    hide(x);  hide(y);  display(z);  display(a);  display(b);  display(c);  //Adds user icon on log in button  d.innerHTML = '<i class = "fa fa-user" aria-hidden="true"></i> ';  // d.innerText = "Account ";  }    //logout function  function logOut()  {  //Destroy Session Variables  destroySession();  isLoggedOutModifyHomeDrop();  }    //The onload functions  //@Override login.js  function loadFunctions()  {  if(isLoggedIn()){  isLoggedInModifyHomeDrop();  }else{  isLoggedOutModifyHomeDrop();  }    //Load Data for page  populatePage();  }    function populatePage()  {  var lblTotalUser = document.getElementById('totalUserCounter');  var lblLastUpdate = document.getElementById('lblLastUpdate');  var lblTopPage = document.getElementById('lblTopPage');  var lblPageViewCount = document.getElementById('lblPageViewCount');    //Ajax call to run php file that gets number of users in db  var xhttp = new XMLHttpRequest();  xhttp.open('POST', '<http://104.145.83.147/PHP/getUserCount.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  response = this.responseText;  lblTotalUser.innerText = response;  }  };  xhttp.send(null);    //Fill in Last Updated field - which is now  var date = new Date().toLocaleString();  lblLastUpdate.innerText = "Last Updated : " + date;    $topPageName = "";  $topPageViews = 0;    //Fill in top views section  var xhttp = new XMLHttpRequest();  xhttp.open('POST', '<http://104.145.83.147/PHP/getTopPage.php>', true);  xhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  response = this.responseText;  var arrayz = response.split("~split~")  $topPageViews = arrayz[0];  $topPageName = arrayz[1];    lblTopPage.innerText = $topPageName;  lblPageViewCount.innerText = "Times Viewed : "+$topPageViews;  }  };  xhttp.send(null);  } |