**IMPORTANT LINKS:**

**Github:** [**https://github.com/jdwdm3/SWE2016**](https://github.com/jdwdm3/SWE2016)

**Proof of Azure Setup:** jdwswe.centralus.cloudapp.azure.com

**Software Sharks (Group 8)**

**Schedule Shark**

**Requirements Analysis**

**Authors:**

Kaitlin Anderson

Jeremy Warden

Josh Lewis

Han Chen

03/27/2016

**SPRINT 1**

**Version: 0.02**

**Table of Contents**

The Problem……………………………………………………3

User and System Requirements………………………………...4

Functional and Non-Functional Requirements…………………6

Use Case Diagram……………………………………………...8

Activity Diagram……………………………………………….9

ERD…………………………………………………………...10

DDL…………………………………………………………...11

User Interface……….…………………………………………13

Testing…………………………………………………………20

Change Log…..………………………………………………..22

Glossary….………………………..…………………………..23

**Group 8 (Software Sharks)**

Requirements Analysis

**-- PROBLEM --**

During the last few years, multiple members of our group have experienced poor scheduling techniques used for our various places of employment. It is a rather daunting task to balance everything that pertains to each employee such as availability and requesting off in order to formulate an accurate schedule. Additionally, to add an extra headache to the scheduling manager, they also have to create multiple schedules for various positions that are held. Managers already have enough work to attend to with daily problems that occur at the workplace, so we believe we can alleviate some of that stress with the implementation of Schedule Shark!

**- REQUIREMENTS ANALYSIS -**

**User Requirements**

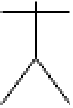
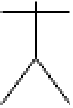
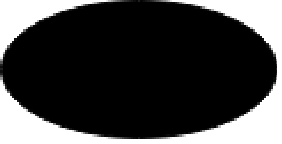
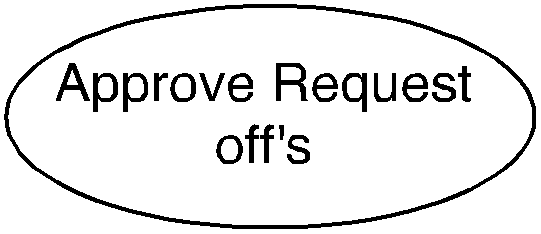
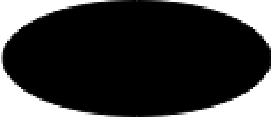
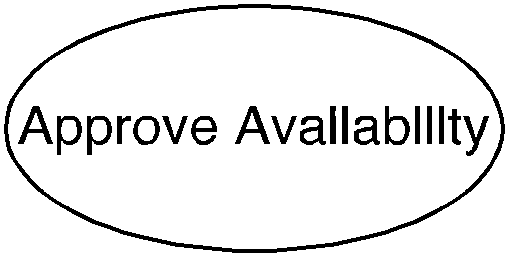
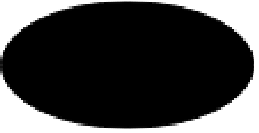
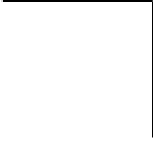
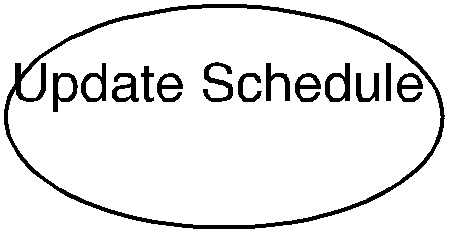
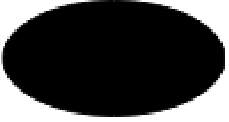
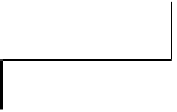
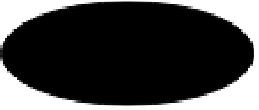
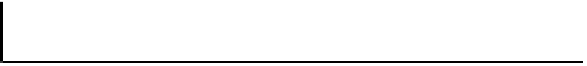
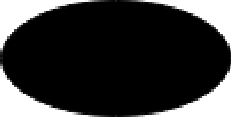
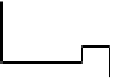
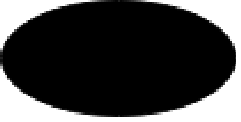
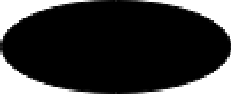
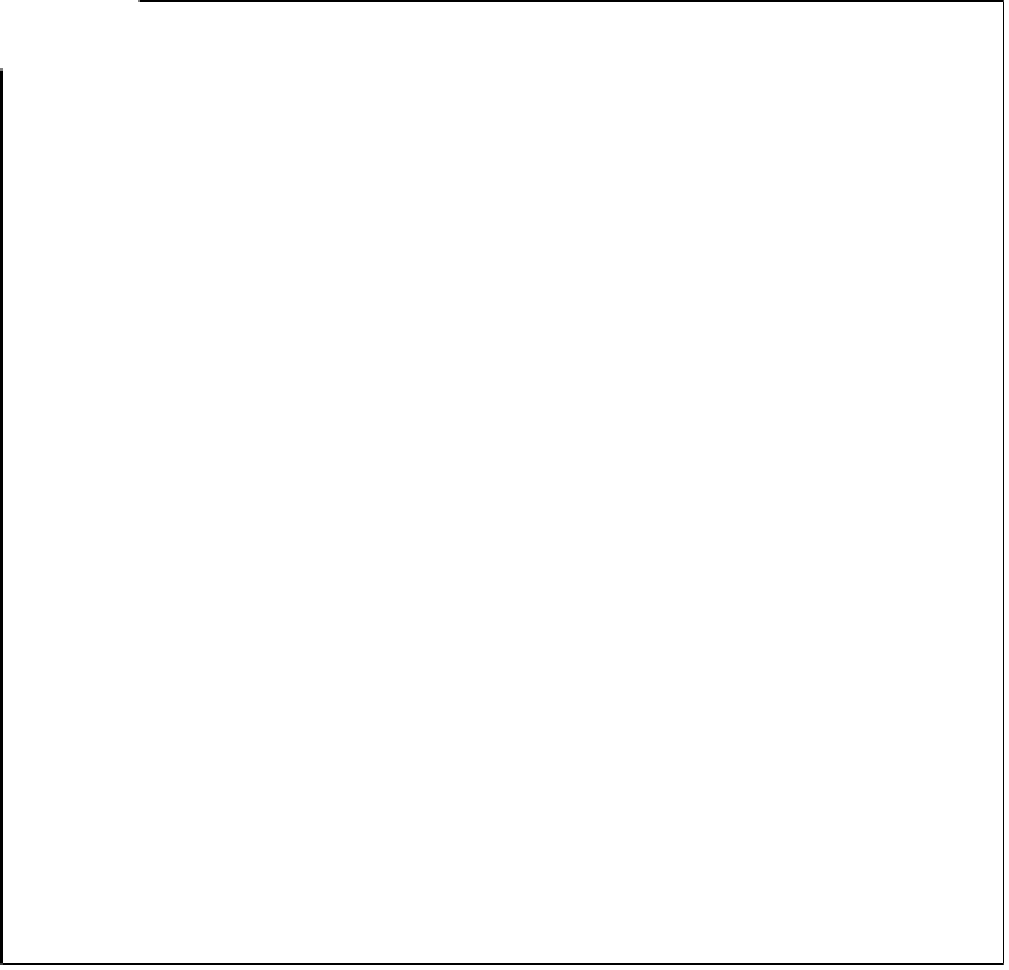
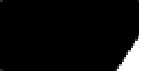
* User Login
  + Employee
    - Server
    - Bartender
    - Busser
    - Food-Runner
    - Cashier
    - Hostess/Host
    - Supervisor
    - Manager
* Give Availability
  + Employee
    - Days of availability
    - Time of availability
* Request Time off
  + Employee
* Accept Request off
  + Manager
* View Schedule
* Edit Schedule
  + Manager
* Contact other employees
  + Similar employees & Managers
* Registration approval
  + Manager
* **System Requirements –**
* **LAMP STACK (STORED ON AZURE)**
  + Linux
    - Virtual Machine is powered by Linux, creating a safe environment for us to utilize the resources necessary to run our application
  + Apache
    - Web server where we will be hosting our web application
  + MySQL Database
    - Our web based application will be database driven, using user data in order to function properly
  + Python/PHP
    - We will be communicating between our controller and our model with a server side scripting language
* **Functional Requirements -**
* **User Login**
  + On correct input, advances user to site.
  + On incorrect input, allows user to try again or change password.
* **Give Availability**
  + Store Employees availability
  + Use information for generating Schedule
  + Edit availability
* **Request Time off**
  + Send request off dates to manager in order for approval
  + Store date on approval
  + Use information for generating schedule
* **View Schedule**
  + Each type of employee will have ability to view the corresponding schedule
* **Edit Schedule**
  + Managers should be able to make changes to the schedule
* **Contact Other Employees**
  + Managers should be able to mass e-mail employees.
  + Employees should be able to contact similar employees, as well as their manager.

**Non-Functional Requirements**

**Primary: Josh Lewis**

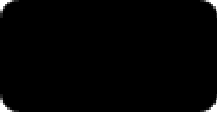
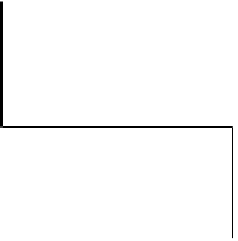
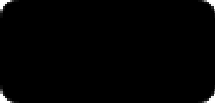
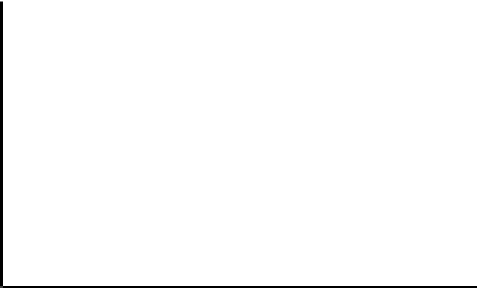
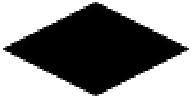
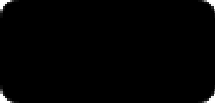
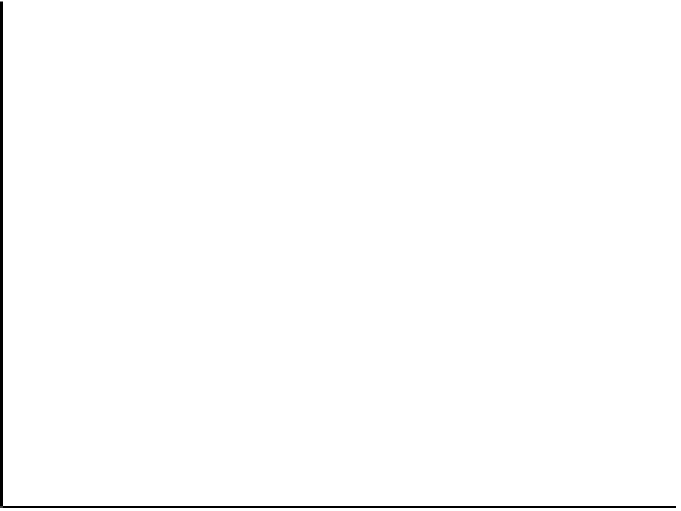
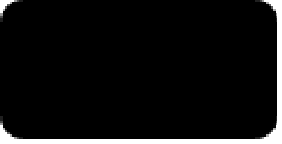
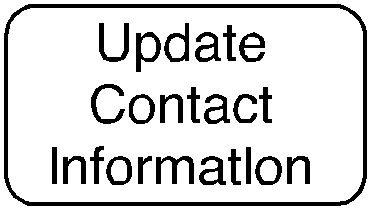
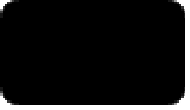
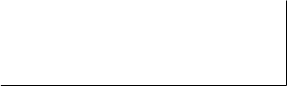
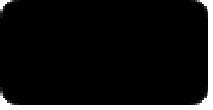
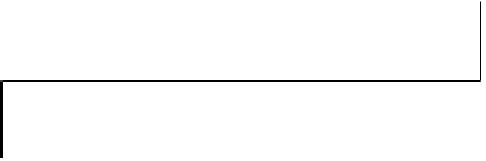
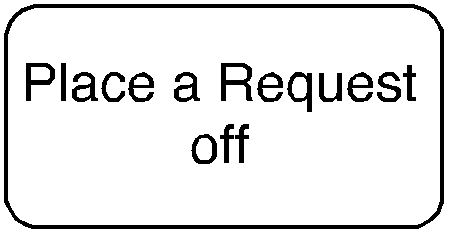
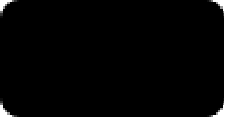
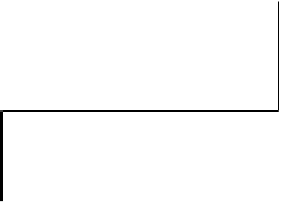
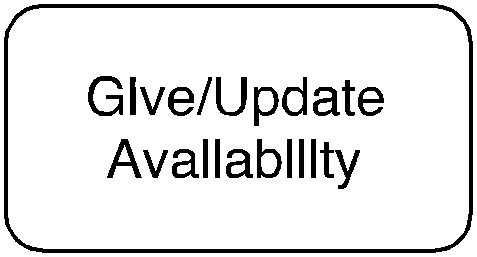
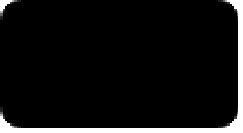
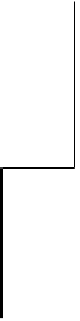
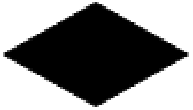
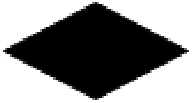
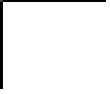
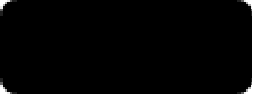
**Secondary: Han Chen**

* The program should be stable; it should have an incredibly low failure rate.
* The program should be fast and efficient, responding in under a minute to queries and requests.
* The program should recover gracefully from incorrect inputs and from system outages.
* Database should be able to handle large amount of data and simultaneous requests.
* System should be secure, not just anybody can register for an account in the system, employees must be invited.
* Should work on multiple web platforms including, IOS and Android web browsers.
* Use Case Diagram -



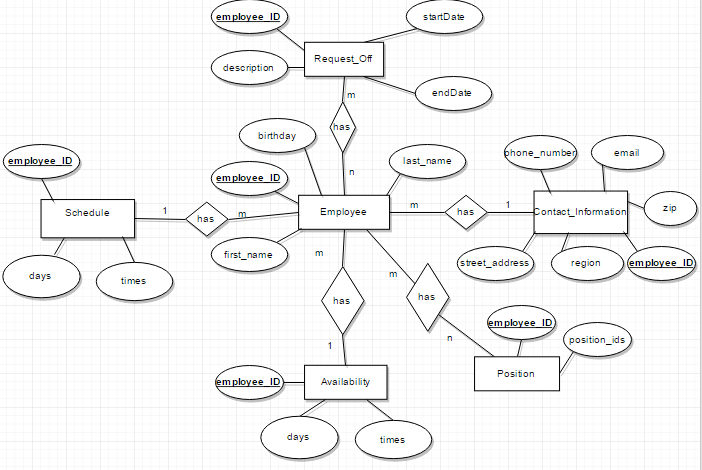
Employee

Manager



* Activity Diagram -

- Entity Relationship Diagram -



* **Data Definition Language** (DDL) -

CREATE TABLE Employee

(

employee\_ID int,

first\_name varchar(25),

last\_name varchar(25),

birthday Date,

PRIMARY KEY (employee\_ID)

);

INSERT INTO Employee (0,Jeremy,Warden,);

CREATE TABLE Contact\_Information

(

employee\_ID int,

phone\_number varchar(10),

email varchar(50),

zip varchar(5),

region varchar (2),

PRIMARY KEY (employee\_ID),

FOREIGN KEY (employee\_ID) REFERENCES Employee(employee\_ID)

);

CREATE TABLE Position

(

employee\_ID int,

position\_IDs varchar(10),

PRIMARY KEY (employee\_ID),

FOREIGN KEY (employee\_ID) REFERENCES Employee(employee\_ID)

);

CREATE TABLE Availability

(

employee\_ID int,

days varchar(10),

times time,

PRIMARY KEY (employee\_ID),

FOREIGN KEY (employee\_ID) REFERENCES Employee(employee\_ID)

);

CREATE TABLE Request\_Off

(

employee\_ID int,

description varchar(100),

startDate Date,

endDate Date,

PRIMARY KEY (employee\_ID),

FOREIGN KEY (employee\_ID) REFERENCES Employee(employee\_ID)

);

CREATE TABLE Schedule

(

employee\_ID int,

days varchar(10),

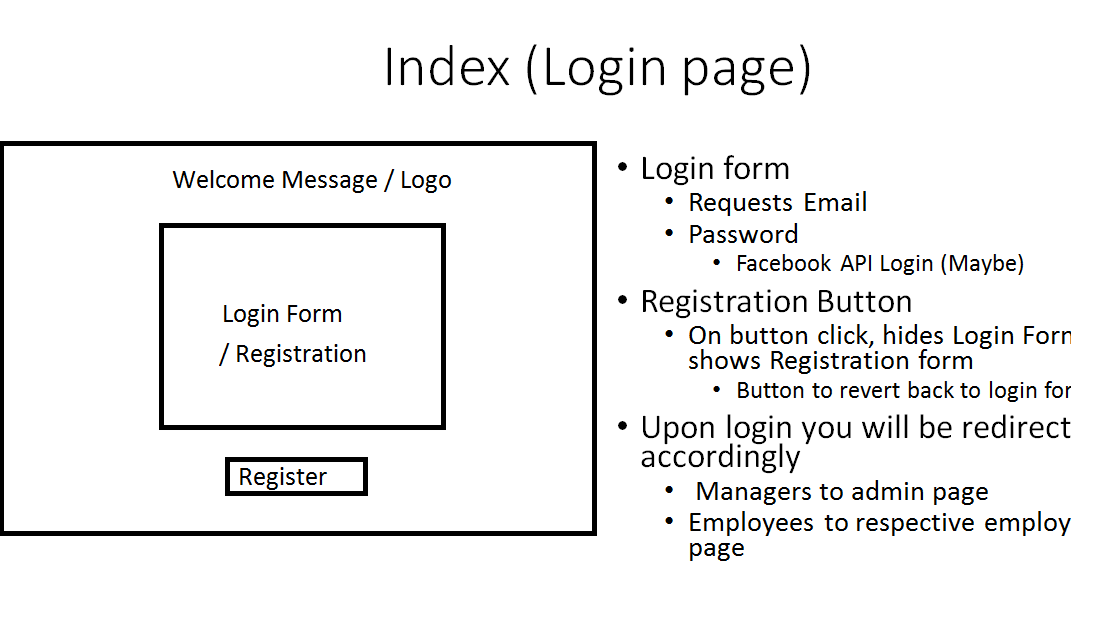
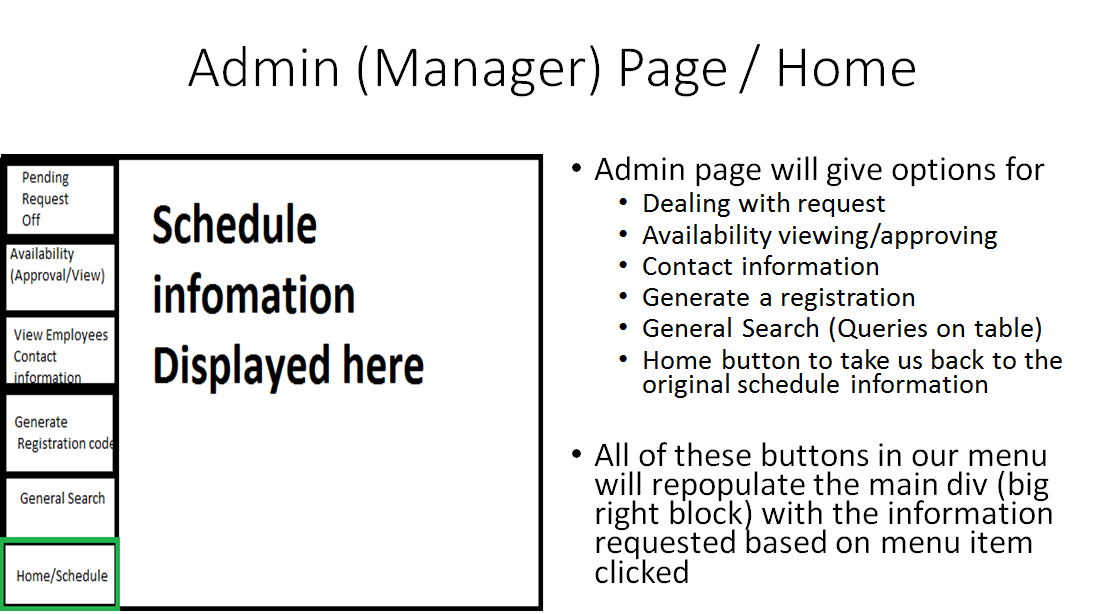
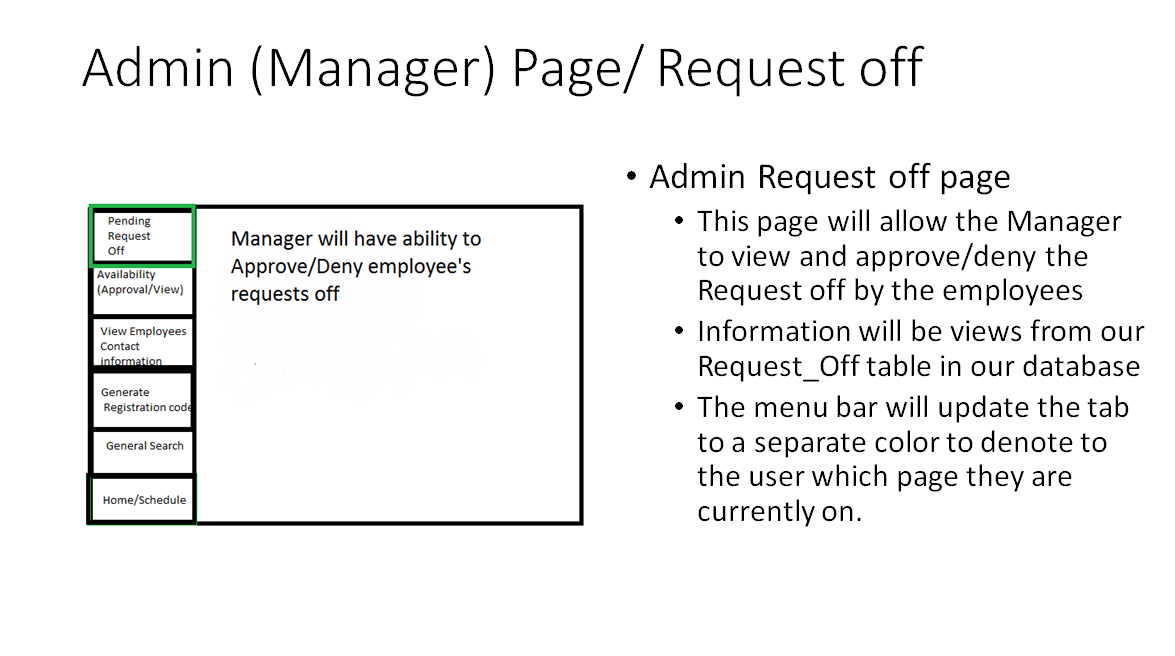
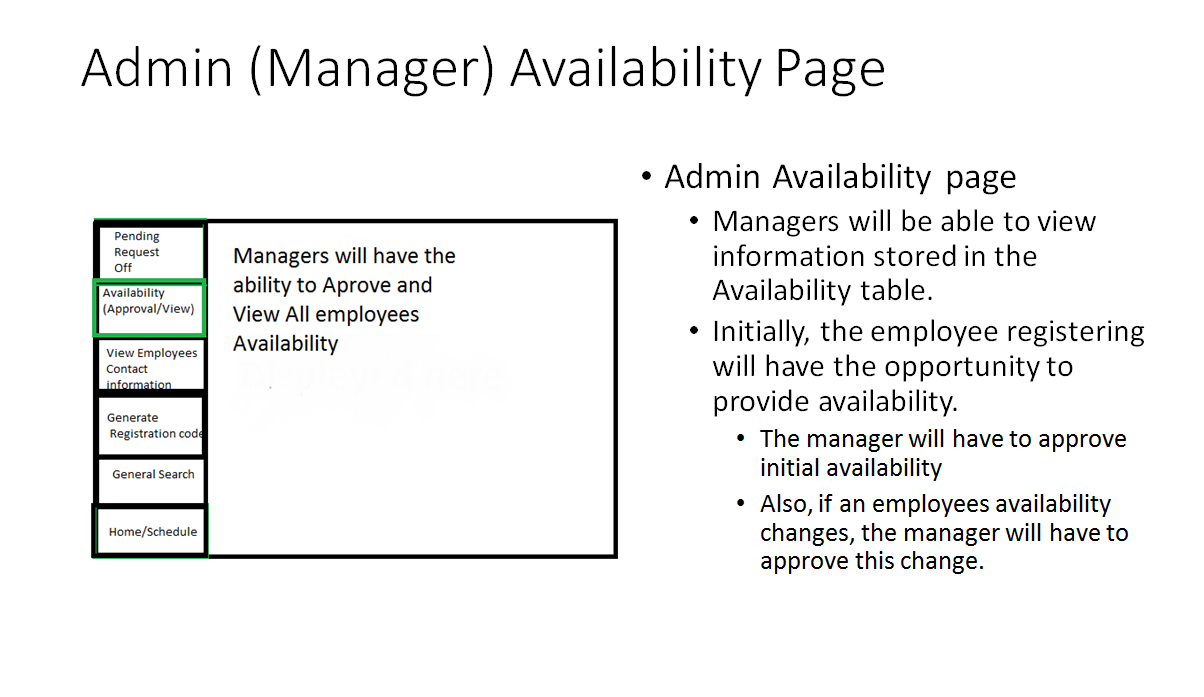
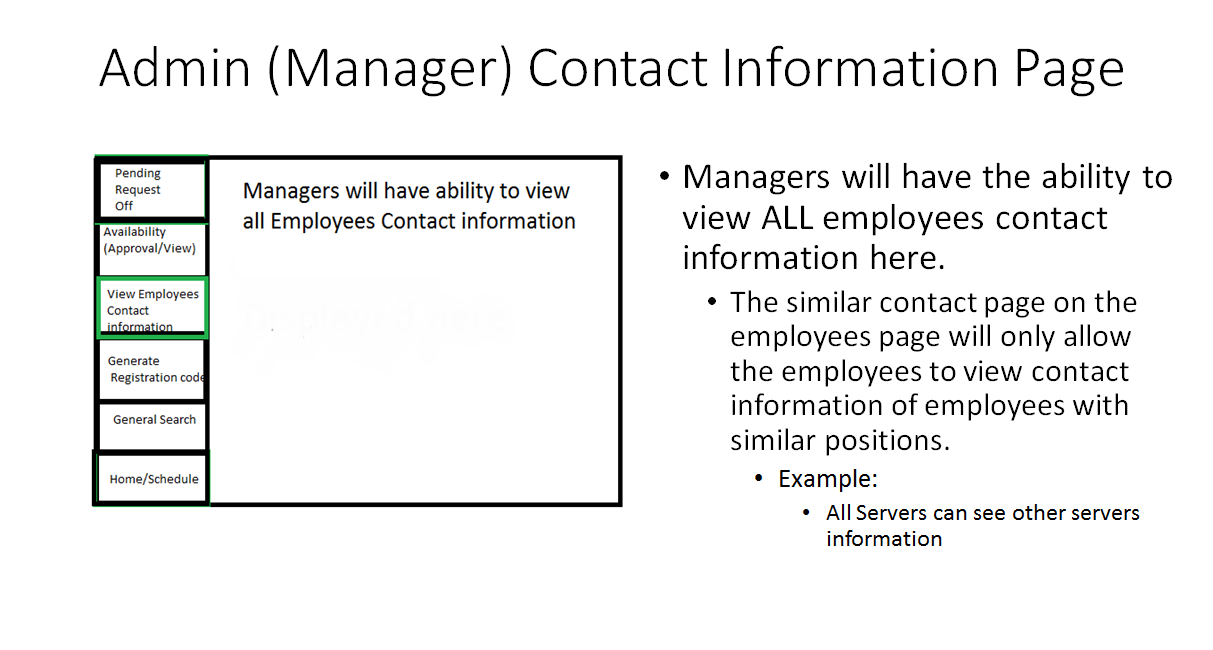
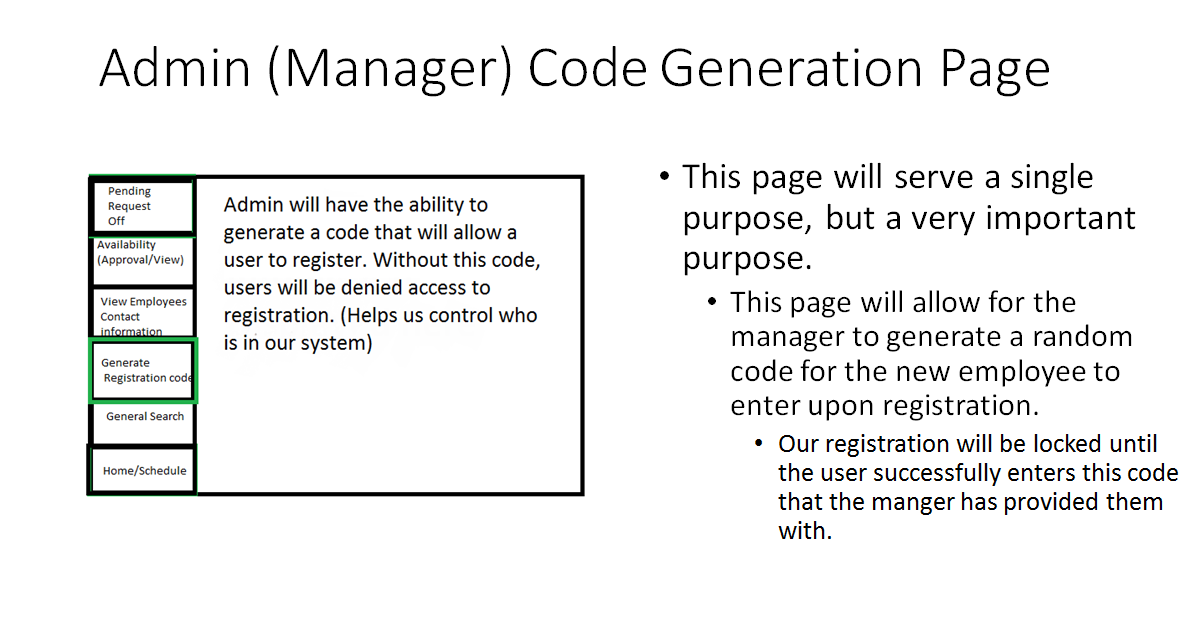
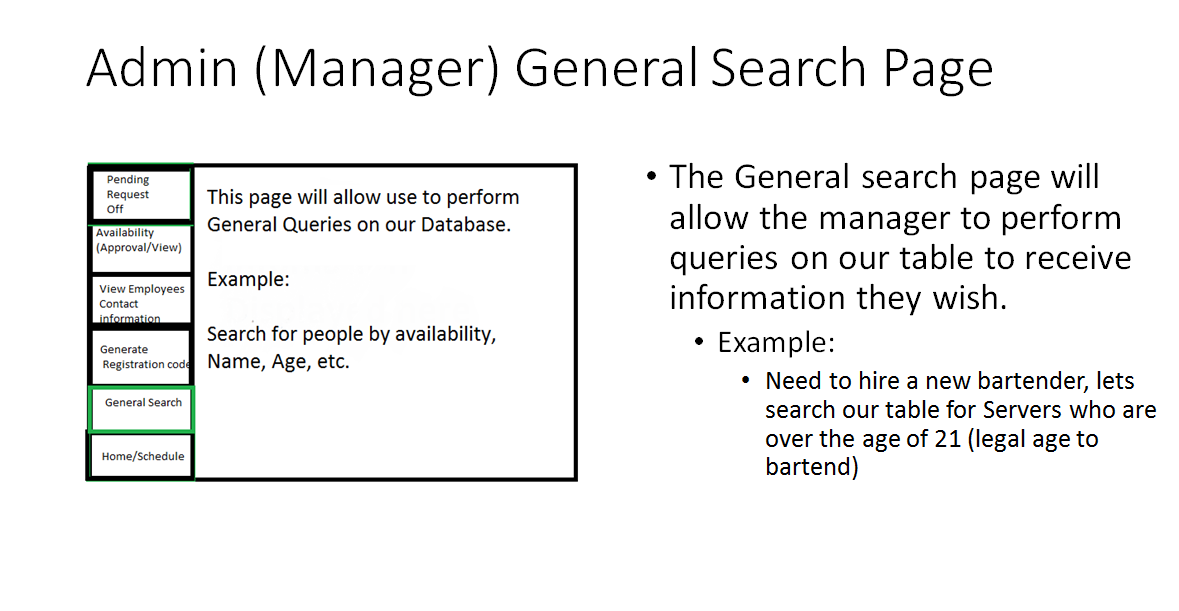
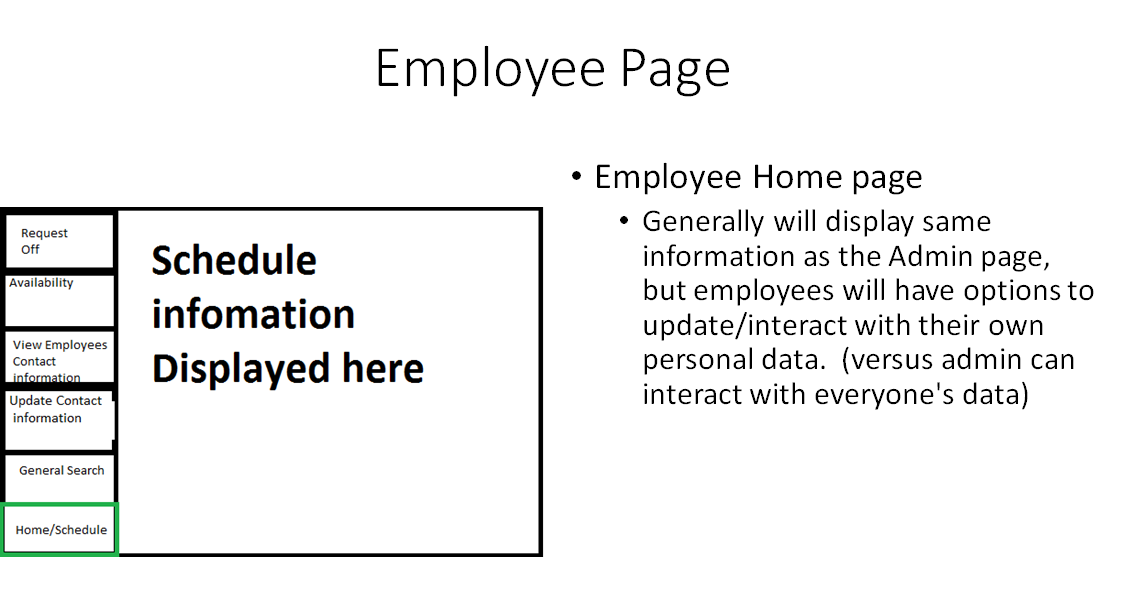
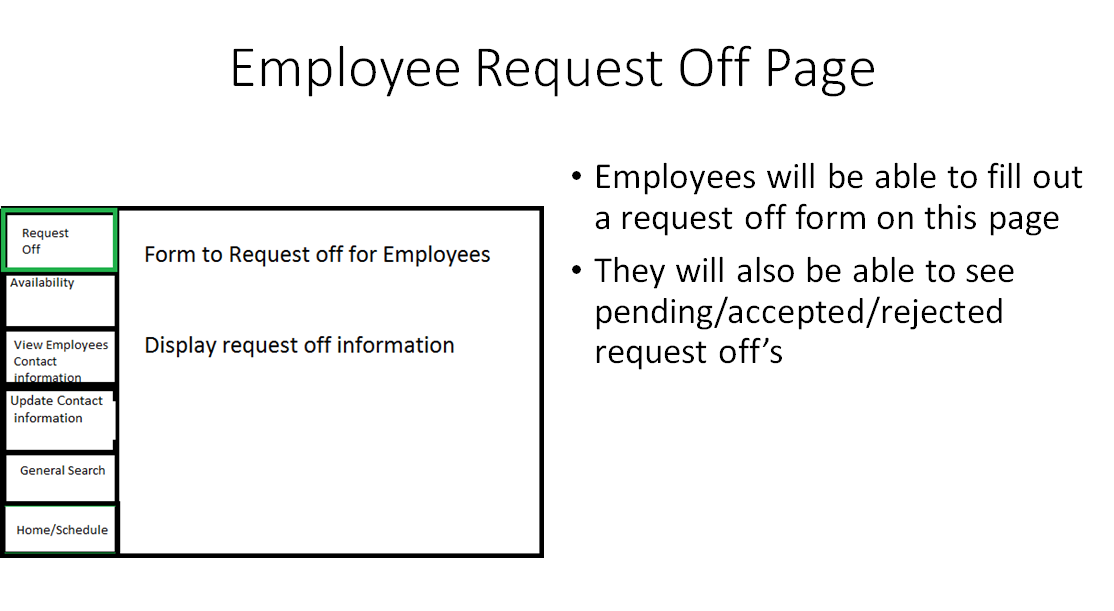
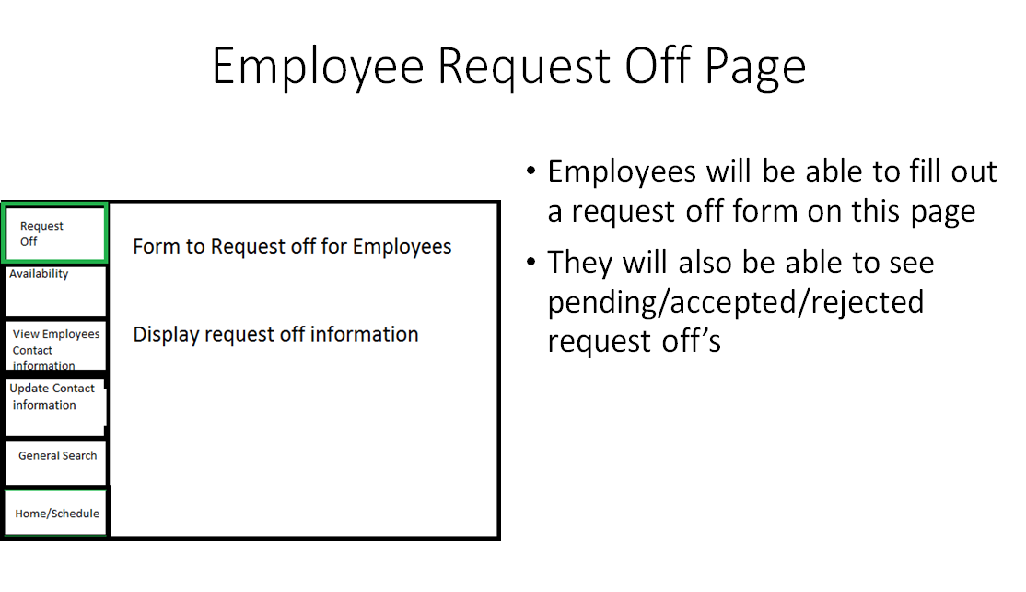
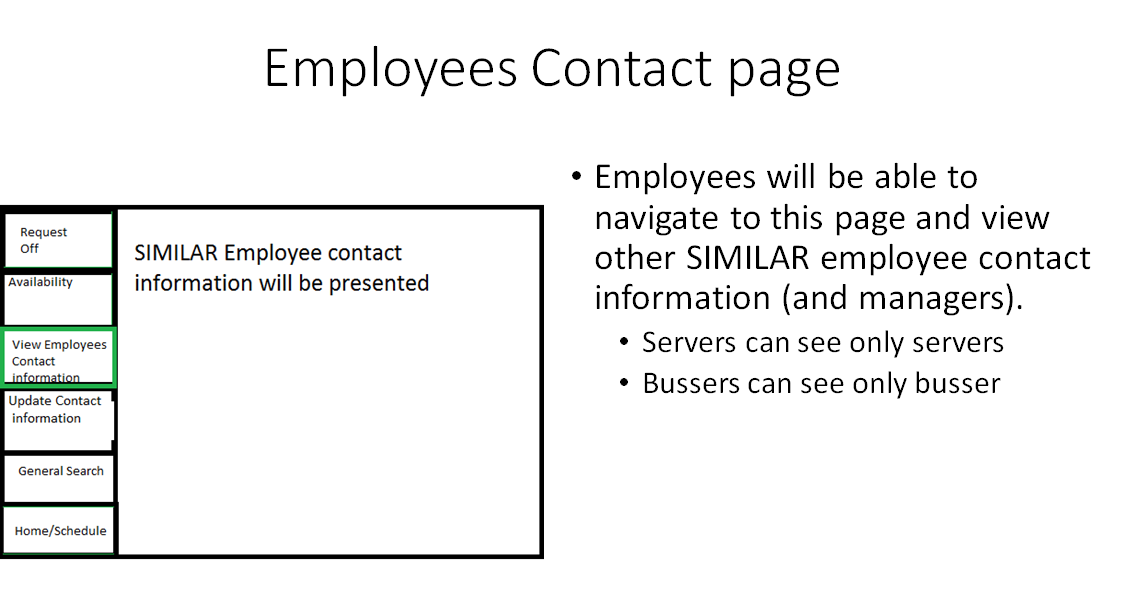
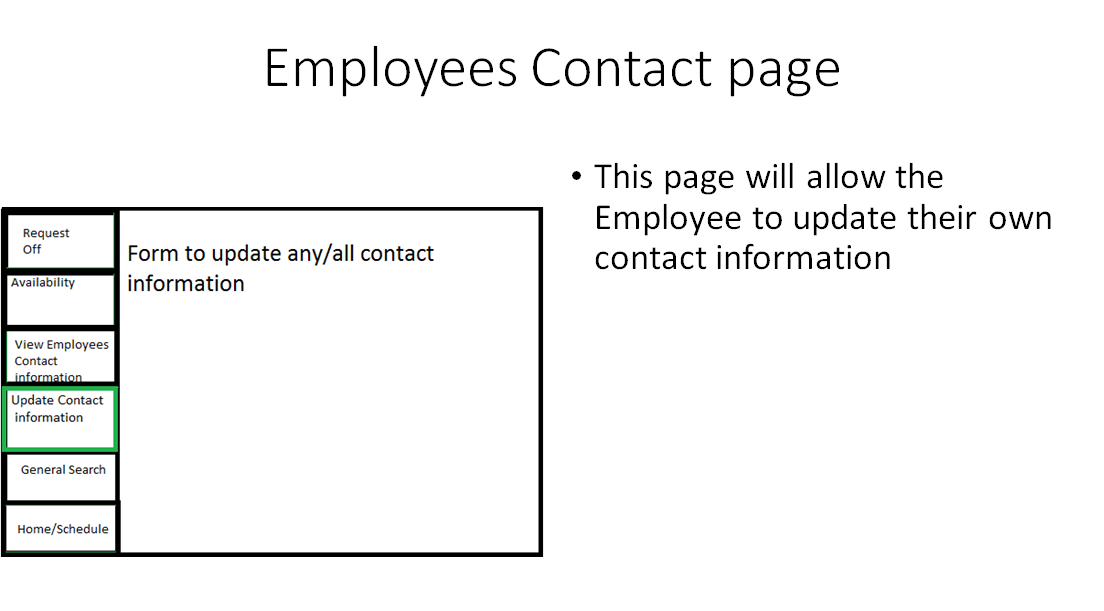
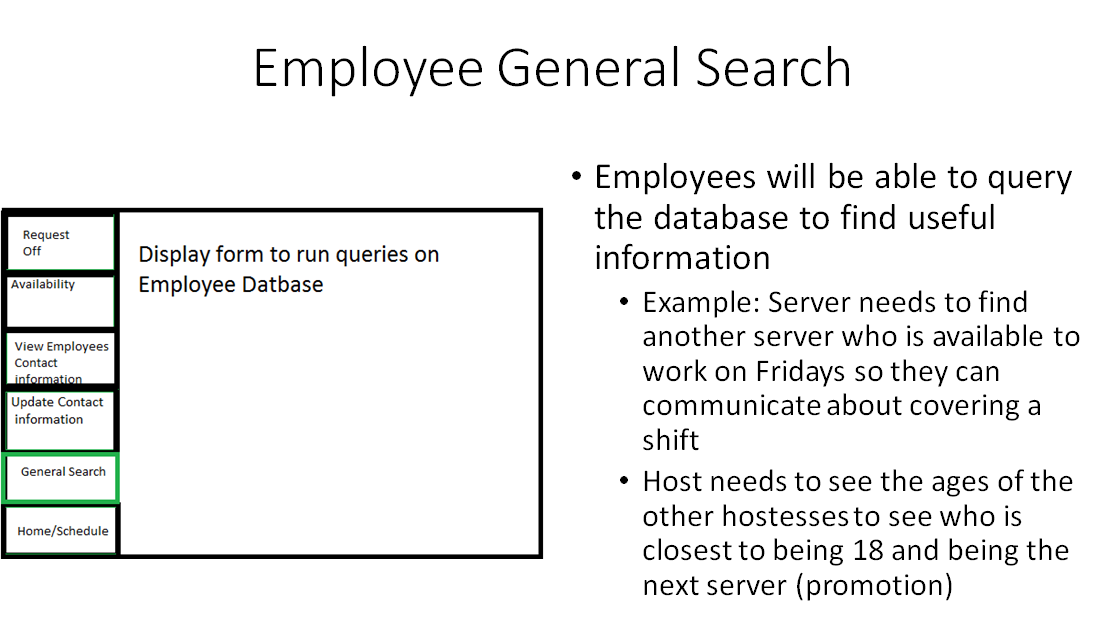
times time,

PRIMARY KEY (employee\_ID),

FOREIGN KEY (employee\_ID) REFERENCES Employee(employee\_ID)

);

* User Interface Design -

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 

* Testing -

**User Acceptance Testing: (VERIFICATION)**

This step of testing will take place once we have a physical user interface to interact with. For this portion, we request that you take a look at the general sketches and description of each page and give us feedback on the general design. Following our User Requirements (ABOVE IN THIS DOCMENT), we feel we have successfully addressed every one of our requirements. This design will enable us to allow for login and registration, ability to give availability, ability to request off, and update any personal information that the user gives us during the registration process. We will address User Acceptance testing in much greater detail as we begin to develop the physical UI.

**Unit Testing: (Plenty more to come as we get further along)**

**(VALIDATION)**

1. **Test to ensure registration form Data successfully is stored into our Database**
2. **Test to ensure that the availability can be updated by a user, viewed by a manger, and approved and stored back as the default availability for the user who requested the update**
3. **Test to ensure SERVERS can only view SERVER and MANAGER contact information (not able to view other employee’s info)**
4. **Test to ensure there are NO SQL INJECTION Vulnerabilities**
5. **Test to ensure we can successfully run an array of queries on our Database (General Search page) and return valid results**
   1. **NO SQL INJECTIONS**
   2. **PREPARED STATEMENTS**
6. **Test to ensure the Schedule is filled out completely**

**Regression Testing: (VALIDATION)**

Regression testing will occur with each step we complete from this point on. We have now successfully deployed our Database, and within the next week we will be building our web interface to interact with users and store vital information necessary to generate schedules for our employees. Every piece of information we collect, whether it is at the registration phase or updated information later on (example: updating availability) we will be ensuring that with every functional piece we add to our web application, we will check our database to ensure no data was stored improperly. If every step allows our data to remain useful, our scheduling algorithm will work just fine.

**Integration Testing: (VALIDATION)**

**We are creating a LAMP stack in order to run our application:**

1. **We have started by creating a LINUX virtual machine**
2. **Next, we integrated an APACHE WEB SERVER onto our VM**
3. **Next, we integrated MYSQL Database that we are using on our APACHE WEB SERVER**
4. **Finally, we have integrated PHP in order to communicate between our WEB SERVER (Client) and our MYSQL DATABASE (Server)**

**We will be testing with every piece of our UI that we create that we don’t break anything that already is working. If we find a bug, we will revert and start from scratch on the current problem we are trying to solve.**

Change Log

|  |  |  |  |
| --- | --- | --- | --- |
| # | Date | By | Description |
| 01 | 03/10/2016 | All | Sprint one meeting: decide how tasks are divvied up |
| 02 | 03/15/2016 | Kaitlin Anderson &  Jeremy Warden | Create user requirements |
| 03 | 03/15/2016 | Jeremy Warden &  Kaitlin Anderson | Create system requirements |
| 04 | 03/15/2016 | Josh Lewis &  Han Chen | Create functional requirements |
| 05 | 03/15/2016 | Han Chen &  Josh Lewis | Create non-functional requirements |
| 06 | 03/15/2016 | Jeremy Warden | Cerate DDL, User Case |
| 07 | 03/15/2016 | Kaitlin Anderson | Create ERD |
| 08 | 03/17/2016 | Josh Lewis | Integrate the documents and diagrams, create table of contents |
| 09 | 03/17/2016 | Han Chen | Create change log and glossary |
| 10 | 3/27/16 | Jeremy Warden | **Finalized**: ERD, DDL, UI  **Updated**: Table of contents, Glossary, Testing Scenarios. (SPRINT 1) |

**-- GLOSSARY –**

**Schedule**

A list of employees, and associated information, for example, position, working time, responsibilities for a given time period.

**User Requirements**

What the users expect the software to be able to do. The user requirements can be used as a guide to planning cost, timetables, milestones, testing, etc.

**System Requirements**

In order to work efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as system requirements and are often used as a guideline as opposed to an absolute rule.

**Functional Requirements**

It essentially specifies what the system should do. It specifies a behavior or function, for example, display the name, available time and edit the employees’ information, etc.

**Non-functional Requirements**

It essentially specifies how the system should behave and that it is a constraint upon the systems behavior. One could also think of non-functional requirements as quality attributes for of a system.

**Entity Relationship Diagram (ERD)**

Dealing with scheduling, involves a lot of data. We use this ERD diagram in order to give a pictorial representation of how our data will be stored. ERD’s use relationships between the data in order to store it more accurately and more clean.

**Data Definition Language (DDL)**

We take the ERD diagram and create a series of CREATE TABLE statements in SQL that allow our design to come to life on our Linux Virtual Machine.

**User Interface (UI)**

It is essential to the process that we show a rough draft of interface that our user will visually see. This allows us to get your approval on the scheme and also discuss within our group what layout will be the most effective for our application.