

IMPORTANT LINKS:

Github: <https://github.com/jdwdm3/SWE2016>

Login Page: jdwswe.centralus.cloudapp.azure.com

Registration Page: <http://jdwswe.centralus.cloudapp.azure.com/register.php?>

Registration Step 2 (Basic): http://jdwswe.centralus.cloudapp.azure.com/register_basic.php

Registration Step 3 (Contact): http://jdwswe.centralus.cloudapp.azure.com/register_contact.php

Registration Step 4(Availability): http://jdwswe.centralus.cloudapp.azure.com/register_availability.php

Confirming Registration: http://jdwswe.centralus.cloudapp.azure.com/registration_Check.php

Software Sharks (Group 8)

Schedule Shark

Requirements Analysis

Authors:

Kaitlin Anderson

Jeremy Warden

Josh Lewis

Han Chen

03/27/2016

SPRINT 3

Version: 0.04

Table of Contents

Overview.....	4
The Problem.....	4
Requirements.....	5
User Requirements.....	5
System Requirements.....	6
Functional.....	7
Non-Functional Requirements.....	8
System Design.....	9
Use Case Diagram of Whole System.....	9
Activity Diagram of Schedule Interaction.....	10
ERD of Database.....	11
Data Definition Language(DDL).....	12
Employee Table.....	12
Availability Table.....	12
Contact Information Table.....	13
Position Table.....	14
Request Off Table.....	15
Schedule Table.....	15
Registration Codes Table.....	16
Data Manipulation Language(DML).....	17
Employee Table.....	17
Availability Table.....	18
Contact Information Table.....	18
Position Table.....	20
Request Off Table.....	21
Schedule Table.....	22

Registration Codes Table.....	22
User Interface.....	23
Login/Registration.....	23
Employee Home.....	26
Employee Availability.....	27
Employee Contact.....	28
Employee Request Off.....	29
Manager Home.....	30
Manager Availability.....	31
Manager Contact.....	32
Manager Request Off.....	33
Testing.....	35
User Acceptance Testing.....	35
Unit Testing.....	35
Regression Testing.....	36
Integration Testing.....	36
Edge Case Testing.....	36
Change Log.....	39
Glossary.....	41

OVERVIEW

-- 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

-- USER REQUIREMENTS --

Primary: Kaitlin Anderson

Secondary: Jeremy Warden

- **Employees**
 - Server
 - Bartender
 - Busser
 - Food-Runner
 - Cashier
 - Hostess/Host
 - Supervisor
- **Managers**
 - Floor Manager
 - Branch Manager
- **Manager Requirements**
 - Edit Schedule
 - Swap shifts
 - Remove employees
 - Add employees
 - Change times of shifts
 - Generate Registration Code for New Employees
 - Accept Request Off from Employees
 - Accept Availability Changes from Employees
 - Contact All Employees
- **Employee Requirements**
 - Give Availability
 - Day of availability
 - Time of availability
 - Request Time Off
 - Edit Availability
 - Contact employees within the same job title
- **Manager and Employee Requirements**
 - Registration
 - User Login
 - View Schedule
 - General Search of Employees

-- SYSTEM REQUIREMENTS --

Primary: Jeremy Warden

Secondary: Kaitlin Anderson

- **Cloud Storage**
 - Azure
- **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 --

Primary: Han Chen

Secondary: Josh Lewis

- **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.
- **Adding New Employees**
 - Generate random code for employee to enter and gain access to system

-- NON-FUNCTIONAL REQUIREMENTS --

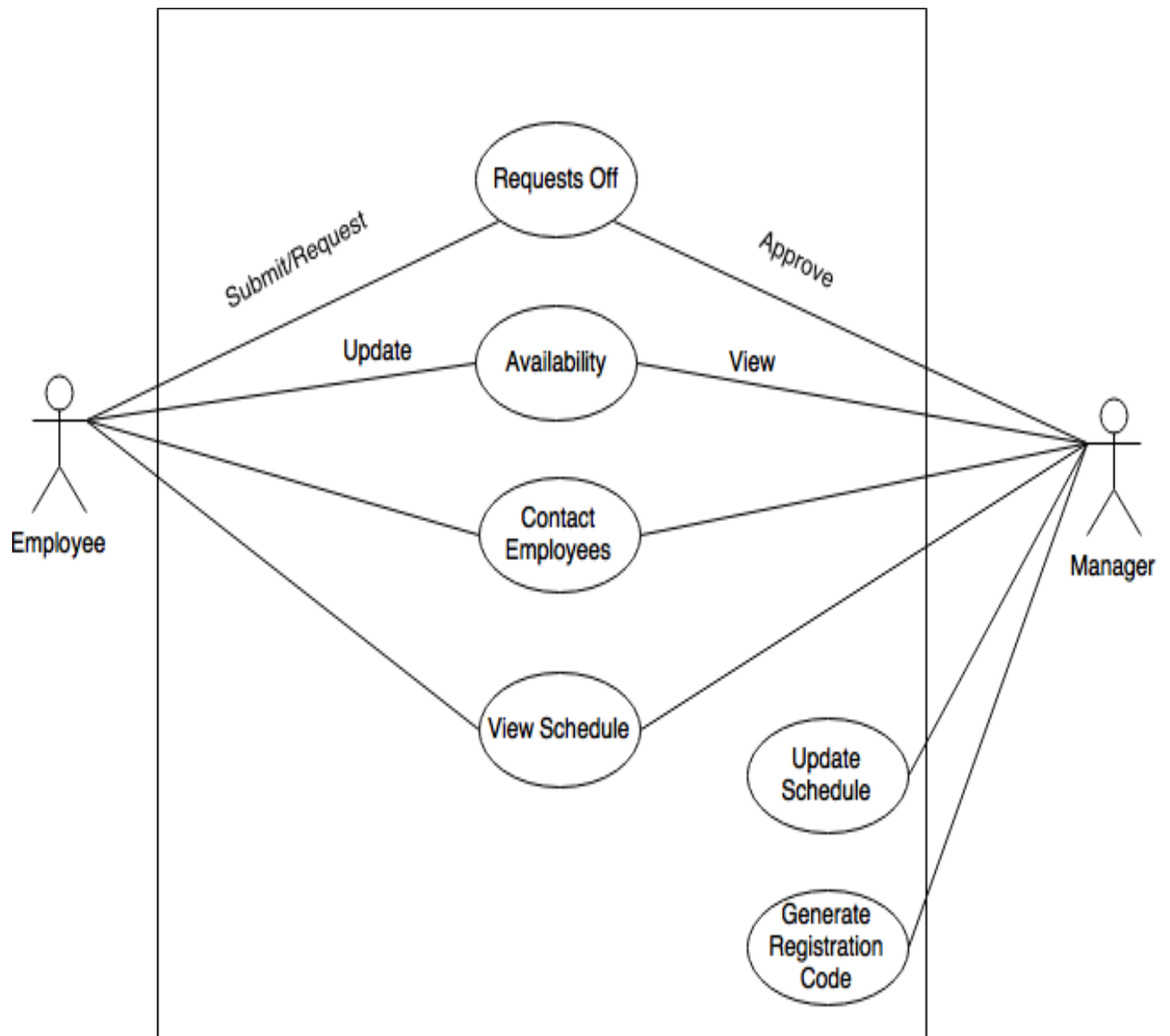
Primary: Josh Lewis

Secondary: Han Chen

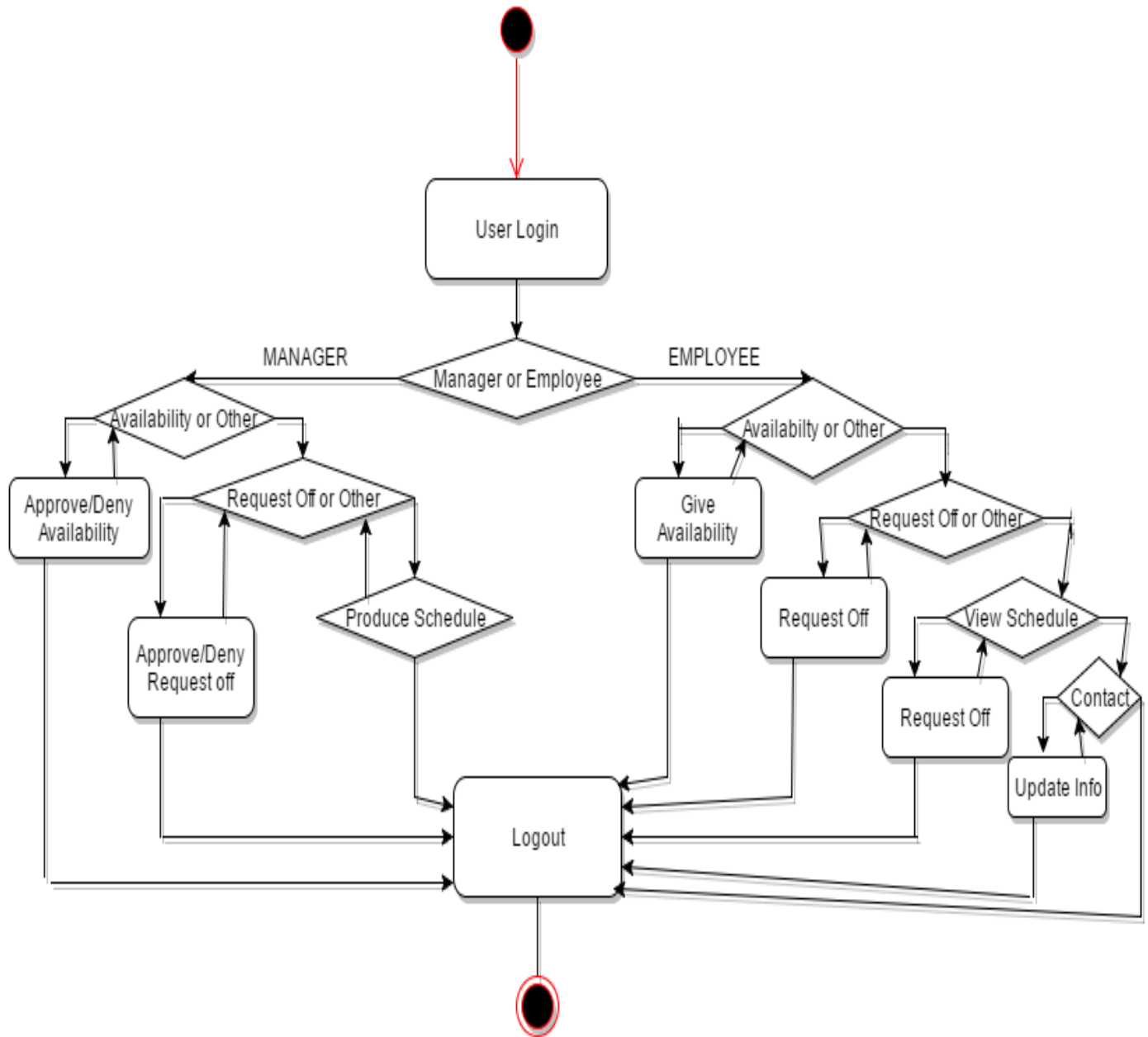
- **Stability**
 - The program should be stable; it should have a one percent failure rate.
- **Efficiency**
 - The program should be fast and efficient, responding in under a minute to queries and requests.
- **Recoverability**
 - The program should recover gracefully from incorrect inputs and from system outages.
- **Database**
 - Database should be able to handle large amount of data and simultaneous requests.
- **Security**
 - System should be secure, not just anybody can register for an account in the system, and employees must register with an access code generated by a manager.
- **Platforms**
 - Should work on multiple web platforms including, IOS and Android web browsers.

SYSTEM DESIGN

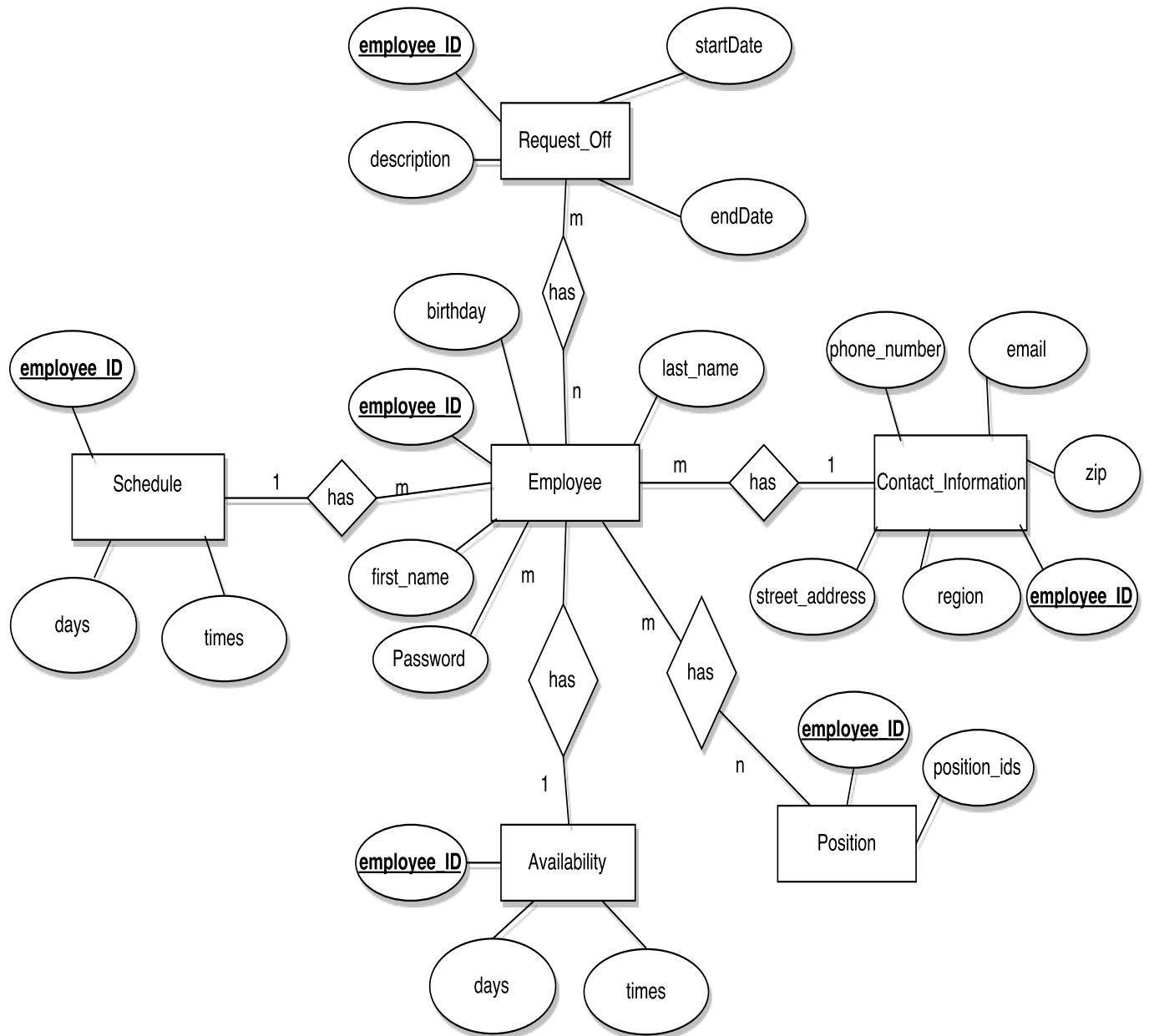
-- USE CASE DIAGRAM OF WHOLE SYSTEM --



-- ACTIVITY DIAGRAM OF SCHEDULE INTERACTION --



-- ERD OF DATABASE --



DATA DEFINITION LANGUAGE (DDL)

--EMPLOYEE --

```
CREATE TABLE Employee
```

```
(  
  employee_ID int,  
  first_name varchar(25),  
  last_name varchar(25),  
  birthday Date,  
  password varchar(25),  
  PRIMARY KEY (employee_ID)  
);
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (0,'Jeremy','Warden','1996-11-30', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (1,'Joe','Peshi','1948-06-23', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (2,'Han','Chen','1995-02-14', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (3,'Kaitlyn','Anderson','1994-04-19', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (4,'Josh','Lewis','1994-02-21', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (5,'Donkey','Kong','1963-12-24', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (6,'Remilia','Scarlet','1509-06-13', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (7,'Bruce','Wayne','1971-01-29', 'Hello123');
```

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)  
VALUES (8,'Deadpool','Awesomeguy','1948-09-19', 'Hello123');
```

--CONTACT INFORMATION--

```
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)
);
```

```
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (0,'5738765309','jeremy@whatever.com', '65202', 'MO');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (1,'5738490998','joe.peshi@ReallyFamous.com', '64908', 'IL');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (2,'4897684430','han.chen@whatevers.com', '65202', 'MO');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (3,'5734443344','Kaitlin@something.com', '65202', 'MO');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (4,'5737890020','josh@whatever.com', '65203', 'MO');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (5,'1113987654','KRoolDroolz@bananas.com', '12345', 'AR');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (6,'6669874636','remilia.scarlet@gmail.com', '48765', 'KS');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (7,'4389207839','gotham@batcave.com', '98766', 'NY');
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)
VALUES (8,'2874392938','deadpool@thebestemail.best', '48732', 'LA');
```

--POSITION--

```
CREATE TABLE Position
(
employee_ID int,
position_IDs varchar(10),
PRIMARY KEY (employee_ID),
```

```
FOREIGN KEY (employee_ID) REFERENCES Employee(employee_ID)
);
```

```
INSERT INTO Position (employee_ID, position_IDs) VALUES (0,'1');
INSERT INTO Position (employee_ID, position_IDs) VALUES (1,'2');
INSERT INTO Position (employee_ID, position_IDs) VALUES (2,'3');
INSERT INTO Position (employee_ID, position_IDs) VALUES (3,'4');
INSERT INTO Position (employee_ID, position_IDs) VALUES (4,'2');
INSERT INTO Position (employee_ID, position_IDs) VALUES (5,'5');
INSERT INTO Position (employee_ID, position_IDs) VALUES (6,'5');
INSERT INTO Position (employee_ID, position_IDs) VALUES (7,'3');
INSERT INTO Position (employee_ID, position_IDs) VALUES (8,'4');
```

--AVAILABILITY--

```
CREATE TABLE Availability
(
employee_ID int,
days varchar(10),
times time,
PRIMARY KEY (employee_ID),
FOREIGN KEY (employee_ID) REFERENCES Employee(employee_ID)
);
```

```
INSERT INTO Availability (employee_ID,days,times) VALUES
(0,'MTWTHFS','08:00:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
(1,'TWTHFS','12:00:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
(2,'MTWTHS','10:00:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
(3,'MTWTHFS','06:00:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
(4,'MTWFS','09:45:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
(5,'MTWTHFS','04:00:00');
```

```

INSERT INTO Availability (employee_ID,days,times) VALUES
    (6,'MTWTHFS','14:00:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
    (7,'MTWTHF','11:00:00');
INSERT INTO Availability (employee_ID,days,times) VALUES
    (8,'MTTHFS','10:30:00');

```

--REQUEST OFF--

```

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)
);

```

```

INSERT INTO Request_Off (employee_ID,description,startDate,endDate) VALUES
    (6,'Bored, need a break.','2016-06-19','2016-07-31');
INSERT INTO Request_Off (employee_ID,description,startDate,endDate) VALUES
    (3,'Visiting uncle in Zimbabwae.','2016-08-10','2016-08-21');
INSERT INTO Request_Off (employee_ID,description,startDate,endDate) VALUES
    (2,'Going to the moon for a short time.','2016-06-19','2016-06-24');
INSERT INTO Request_Off (employee_ID,description,startDate,endDate) VALUES
    (7,'The Joker got out of prison again.','2016-10-19','2016-10-22');
INSERT INTO Request_Off (employee_ID,description,startDate,endDate) VALUES
    (4,'Visiting uncle in Zimbabwae.','2016-12-19','2016-12-20');

```

--SCHEDULE--

```

CREATE TABLE Schedule
(
employee_ID int,
days varchar(10),
times time,

```

```
PRIMARY KEY (employee_ID),  
FOREIGN KEY (employee_ID) REFERENCES Employee(employee_ID)  
);
```

```
INSERT INTO Schedule (employee_ID,days,times) VALUES (0,'MTWTHF','09:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (1,'TWTTHFS','09:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (2,'MTWTHS','11:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (3,'MTWTHF','10:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (4,'MTWFS','09:30:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (5,'MTWTHS','09:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (6,'MTWTHF','14:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (7,'MTWTHF','11:00:00');  
INSERT INTO Schedule (employee_ID,days,times) VALUES (8,'MTTHF','11:00:00');
```

--REGISTRATION CODES--

```
CREATE TABLE Registration_Codes  
(  
code int,  
PRIMARY KEY(code)  
);
```

```
INSERT INTO Registration_Codes(code) VALUES ($code);
```


DATA MANIPULATION LANGUAGE(DML)

-- EMPLOYEE --

```
INSERT INTO Employee (employee_ID, first_name, last_name, birthday, password)
VALUES ($id, $FirstName, $LastName, $Birthdate, $Password);
```

```
SELECT * FROM Employee;
--Displays all employees
```

```
SELECT first_name , last_name FROM Employee WHERE birthday <=
DATE_SUB(curdate(),INTERVAL 21 YEAR);
--Returns all employees who are 21 or older
```

```
SELECT first_name,last_name FROM Employee WHERE employee_ID = $id;
--Displays an employee
```

```
DELETE FROM Employee WHERE employee_ID = $id;
--Deletes an employee based on id
```

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",
Request_Off.description AS "Description", Request_Off.startDate AS "Start",
Request_Off.endDate AS "End" FROM Request_Off JOIN Employee
USING(employee_ID);
--Finds all employees who have requested off
```

```
UPDATE Employee SET first_name = $fname, last_name = $lname WHERE
employee_ID = $id;
--This can be split into two separate queries if need be
```

```
UPDATE Employee SET birthday=$bday WHERE employee_ID = $id;
-- Just in case the employee makes a mistake when entering it
```

```
UPDATE Employee SET employee_ID = $newId WHERE employee_ID = $oldId;
```

```
SELECT * FROM Employee WHERE MONTH(birthday) AND DAY(birthday) =  
MONTH(curdate()) AND DAY(curdate());--returns anyone who has a birthday today
```

-- AVAILABILITY --

```
INSERT INTO Availability (employee_id,days,times) VALUES ($id,$days,$times);
```

```
SELECT * FROM Availability;
```

```
DELETE FROM Availability WHERE employee_ID = $id;  
--Deletes an availability listing based on employee id
```

```
UPDATE Availability SET days = $days, times = $times WHERE employee_ID = $id;  
--Changes the current availability to new days and times
```

```
UPDATE Availability SET employee_ID = $newId WHERE employee_ID = $oldID;  
--Sets an availability to a different employee id
```

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Availability.days AS "Days Available", Availability.times AS "Earliest Time Available"  
FROM Availability JOIN Employee USING(employee_ID);  
--Returns the availability of all employees
```

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Availability.days AS "Days Available", Availability.times AS "Earliest Time Available"  
FROM Availability JOIN Employee USING(employee_ID) WHERE days = $days;  
--Similar to last, but only searches for those open on the specified days
```

-- CONTACT INFORMATION--

```
INSERT INTO Contact_Information (employee_ID, phone_number, email, zip, region)  
VALUES ($id,$phone,$email, $zipcode, $region);
```

```
SELECT * FROM Contact_Information;
```

```
DELETE FROM Contact_Information WHERE employee_ID = $id;
```

--Deletes a contact information listing based on the employee id

```
UPDATE Contact_Information SET phone_number = $newPhone WHERE  
employee_ID = $id;
```

--Updates the phone number of the specified employee based on their id

```
UPDATE Contact_Information SET email = $newEmail WHERE employee_ID = $id;
```

--Updates the email belonging to the specified employee, based on id

```
UPDATE Contact_Information SET zip = $newZip , region = $newRegion WHERE  
employee_ID = $id;
```

--Updates the zip and the region fields based on employee id

--I separated most of these since a change in phone number doesn't necessarily herald a change in email or zip code. However a change in zipcode could mean a region change as well

```
UPDATE Contact_Information SET employee_ID = $newID WHERE employee_ID =  
$old_ID;
```

--Updates the employee_ID

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Contact_Information.phone_number AS "Phone", Contact_Information.email AS "E-  
mail", Contact_Information.zip AS "Zip Code", Contact_Information.region AS  
"State/Region" FROM Contact_Information JOIN Employee USING (employee_ID)  
WHERE employee_ID = $id;
```

--Returns contact information on an employee based off their employee id

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Contact_Information.email AS "E-mail" FROM Contact_Information JOIN Employee  
USING(employee_ID);
```

--Returns all emails for all employees

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Contact_Information.phone AS "Phone" FROM Contact_Information JOIN Employee  
USING(employee_ID);
```

--Returns all phone numbers for all employees

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Contact_Information.zip AS "Zip Code", Contact_Information.region AS "State/Region"  
FROM Contact_Information JOIN Employee USING(employee_ID);
```

--Returns all zip and region codes for all employees

```
SELECT Position.position_IDs AS "Position ID", Employee.first_name AS "First  
Name", Employee.last_name AS "Last Name", Contact_Information.phone_number AS  
"Phone", Contact_Information.email AS "E-mail", Contact_Information.zip AS "Zip  
Code", Contact_Information.region AS "State/Region" FROM Contact_Information  
JOIN Employee JOIN Position ON Contact_Information.employee_ID =  
Employee.employee_ID AND Contact_Information.employee_ID =  
Position.employee_ID AND Employee.employee_ID = Position.employee_ID ORDER  
BY Position.position_IDs ASC;
```

--Returns all employees contact info and orders them by their position

```
SELECT Position.position_IDs AS "Position ID", Employee.first_name AS "First  
Name", Employee.last_name AS "Last Name", Contact_Information.phone_number AS  
"Phone", Contact_Information.email AS "E-mail", Contact_Information.zip AS "Zip  
Code", Contact_Information.region AS "State/Region" FROM Contact_Information  
JOIN Employee JOIN Position ON Contact_Information.employee_ID =  
Employee.employee_ID AND Contact_Information.employee_ID =  
Position.employee_ID AND Employee.employee_ID = Position.employee_ID WHERE  
Position.position_IDs = $id ORDER BY Position.position_IDs ASC;
```

--Same as above but only returns a employees in a specific position

-- POSITION --

```
INSERT INTO Position (employee_ID, position_IDs) VALUES ($id,$position_ID);
```

```
SELECT * FROM Position;
```

```
DELETE FROM Position WHERE employee_ID = $id;
```

```
SELECT Position.position_IDs AS "Position ID", Employee.first_name AS "First  
Name", Employee.last_name AS "Last Name", Employee.employee_ID AS "Employee  
Id" FROM Employee JOIN Position USING(employee_ID) ORDER BY  
Position.position_IDs ASC;
```

--Returns all employees and orders them by their position

UPDATE Position SET position_IDs = \$newID WHERE employee_ID = \$id;

--Changes the position of an employee

UPDATE Position SET employee_ID = \$newID WHERE employee_ID = \$oldID;

--Changes the employee that is listed in this specific entry

-- REQUEST OFF --

INSERT INTO Request_Off (employee_ID,description,startDate,endDate) VALUES (\$id,\$description,\$start,\$end);

SELECT * FROM Request_Off;

Delete FROM Request_Off WHERE \$id = employee_ID;

--Deletes the request off based on an input id

Delete FROM Request_Off WHERE endDate < curDate();

--Deletes the request off if the end date has already passed

UPDATE Request_Off SET description = \$newDescription WHERE employee_id = \$id;

--Changes the description of a request off

UPDATE Request_Off SET startDate = \$start , endDate = \$end WHERE employee_id = \$id;

--Changes the start and end date of the request off

UPDATE Request_Off SET employee_ID = \$newID WHERE employee_ID = \$oldID;

--Changes which employee the request off belongs to

SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",
Request_Off.description AS "Description", Request_Off.startDate AS "Start",
Request_Off.endDate AS "End" FROM Employee JOIN Request_Off
USING(employee_ID) ORDER BY Request_Off.startDate ASC;

--Returns all current requests off, ordered by the start date

-- SCHEDULE --

```
INSERT INTO Schedule (employee_ID,days,times) VALUES ($id,$days,$times);
```

```
SELECT * FROM Schedule;
```

```
SELECT Employee.first_name AS "First Name", Employee.last_name AS "Last Name",  
Schedule.days AS "Days Available", Schedule.times AS "Start Time" FROM Schedule  
JOIN Employee USING(employee_ID);
```

--Shows all days that the employees have scheduled to work and what
times they start

```
DELETE FROM Schedule WHERE employee_ID = $id;
```

```
SELECT Position.position_IDs AS "Position ID", Employee.first_name AS "First  
Name", Employee.last_name AS "Last Name", Schedule.times AS "Start Time" FROM  
Schedule JOIN Employee JOIN Position ON Schedule.employee_ID =  
Employee.employee_ID AND Employee.employee_ID= Position.employee_ID AND  
Schedule.employee_ID = Position.employee_ID WHERE Schedule.days = $day ORDER  
BY Position.position_IDs ASC;
```

--Returns all employees able to work on the given days

```
SELECT Position.position_IDs AS "Position ID", Employee.first_name AS "First  
Name", Employee.last_name AS "Last Name", Schedule.times AS "Start Time" FROM  
Schedule JOIN Employee JOIN Position ON Schedule.employee_ID =  
Employee.employee_ID AND Employee.employee_ID= Position.employee_ID AND  
Schedule.employee_ID = Position.employee_ID WHERE Position.position_IDs =  
$posID ORDER BY Position.position_IDs ASC;
```

--Returns all employees of a position that can work on the given days

-- REGISTRATION CODES --

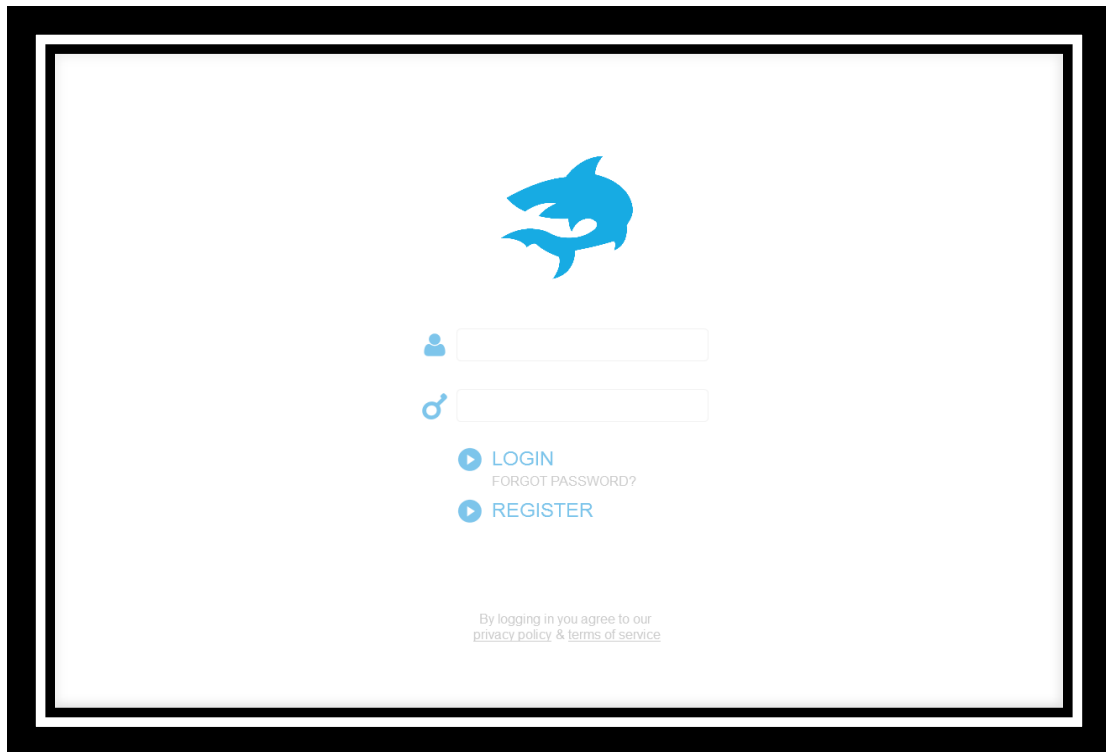
```
INSERT INTO Registration_Codes(code) VALUES ($code);
```

```
SELECT * FROM Registration_Codes WHERE code = $code;
```

```
DELETE FROM Registration_Codes WHERE code = $code
```

USER INTERFACE

--LOGIN/REGISTRATION--



Description:

Our Login/Registration page will be the index page to our website. In order to access our site you **MUST BE REGISTERED**, there is nothing about this site that should be public to people who are not employed.

Stub-Calls:

Register Function(s)

- 1.) Accepts Registration associative array:
 - a.) Package up all the information we need from our forms into organized data to be inserted into our database to register a new user
- 2.) Registration **Employee table** requires:
 - a.) Valid Employee First Name (Separate Function Validates)

- b.) Valid Employee Last Name (Same Function Validates as First Name)
 - c.) Valid Employee birthday (Separate Function Validates)
- 3.) Registration **Contact_Information table** requires:
 - a.) Valid Phone Number (Separate Function Validates)
 - b.) Valid Email Address (Separate Function Validates)
 - c.) Valid Zip-Code (Separate Function Validates)
- 4.) Registration **Position table** requires:
 - a.) Valid Registration Code Generated by Admin
 - i.) That registration code will give access to position_id
- 5.) Registration **Availability table** requires:
 - a.) Valid Days Available (Will be Options to Select, No Validation Required)
 - b.) Valid START TIMES for each DAY AVAILABLE

- 1.) Accepts initial Associate array
- 2.) Calls Validation checks listed above on each index of associative array that needs validation
- 3.) Returns TRUE on a successful registration and adding a new Employee to our Database

Bool Register (char RegisterInfo[]);

- 1.) Accepts either first or last name
- 2.) Checks each character to ensure it is a valid character
- 3.) Checks length of string to ensure it's less than or equal to 25
- 4.) Returns TRUE on Successful validation, FALSE on error

Bool NameValidate(String name)

- 1.) Accepts a Date (Users Birthday)
- 2.) Checks to make sure that the user's age matches requirements for position
 - a.) Servers must be ATLEAST 18
 - b.) Bartenders must be ATLEAST 21
 - c.) All others must be ATLEAST 16
- 3.) return TRUE on Success, FALSE on Failure

Bool BirthdayValidate(Date bday)

- 1.) Accepts Phone Number
- 2.) Ensures the number of digits is EXACTLY 10
- 3.) Ensures all the values are digits
- 4.) return TRUE on Success, FALSE on Failure

Bool PhoneNumberValidate(double phoneNumber)

- 1.) Accepts Email Address
- 2.) Ensures that the email has proper format
 - a.) Either lookup API for this
 - b.) Or check for “@” and “.com” to exists in the string
- 3.) Check to make sure email length is less than or equal to Max email length
- 4.) return TRUE on success, FALSE on failure

Bool EmailValidate(String emailAddress)

- 1.) Accepts a Zipcode
- 2.) Checks to make sure zip code has 5 digits exactly
- 3.) returns TRUE on Success, FALSE on failure

Bool ZipValidate(int zipcode)

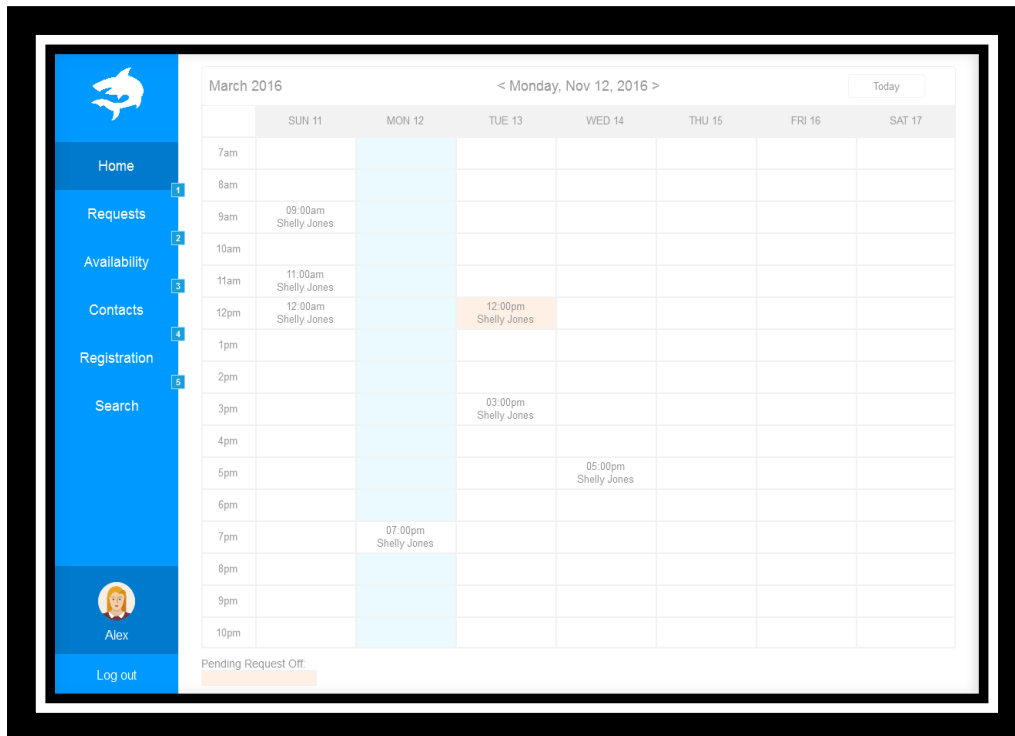
- 1.) A code is generated by our Manager that allows an employee to register AND links the employee to a position ID
- 2.) We check that the code to make sure it is accurate (Check against codes in Database)
- 3.) returns position ID if the code is valid, returns -1 if invalid code

int ValidationCode()

- 1.) Accepts email and password (both strings)
- 2.) Runs Query against database that receives the hashed password associated with the email
- 3.) We will hash the password that was inputted and compare it to the one from the Database
- 4.) On success, fill out all necessary Session Variables and allow login, return true
- 5.) On failure, return false

Bool Login(String email, String Password)

--EMPLOYEE HOME--



Description:

Upon login, our users will be able to view their schedule as their home page. This allows for a quick login, check schedule, and logout.

Stub-Calls:

- 1.) Accepts nothing, returns nothing
- 2.) Runs a strategic query on our Schedule table to print the information out in an organized fashion viewable for the employee

Void LoadSchedule()

- 1.) Logout Accepts nothing, and returns nothing
- 2.) This function will remove all SESSION variables we currently have present

Void Logout()

--EMPLOYEE AVAILABILITY--

	SUN 11	MON 12	TUE 13	WED 14	THU 15	FRI 16	SAT 17
7am							
8am							
9am	09:00am Shelly Jones						
10am							
11am	11:00am Shelly Jones						
12pm	12:00am Shelly Jones		12:00pm Shelly Jones				
1pm							
2pm							
3pm			03:00pm Shelly Jones				
4pm							
5pm				05:00pm Shelly Jones			
6pm							
7pm		07:00pm Shelly Jones					
8pm							
9pm							
10pm							

Description:

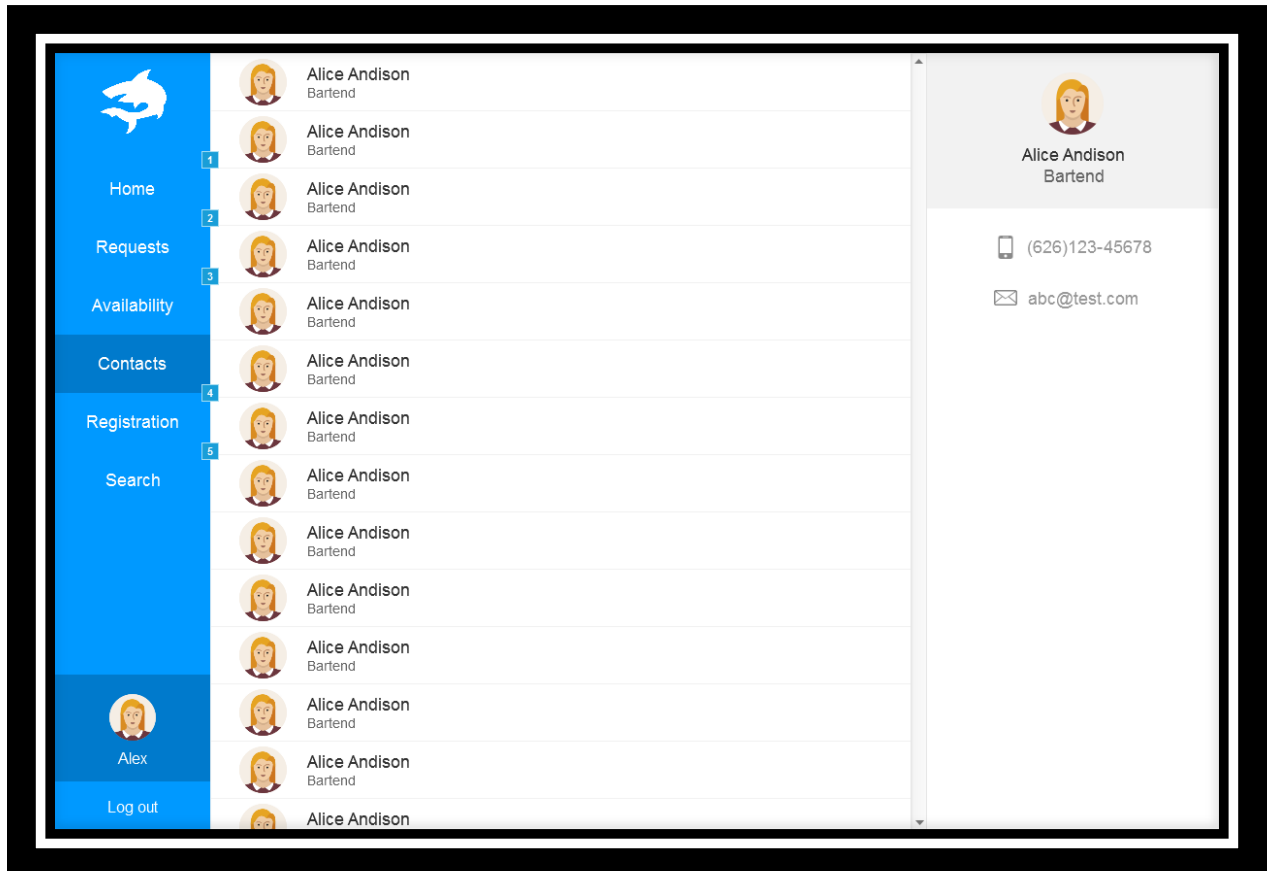
This page will allow our users to update their availability (Upon approval by a manager). It will be a simple form to fill out that will be extremely similar to the form that we use to get availability during registration.

Stub-Calls:

- 1.) This will accept the information from the update availability form
- 2.) Insert this information into our Availability table (UNNAPROVED)
- 3.) The manager will have to approve availability before changes take effect
- 4.) Return TRUE on success, FALSE on failure

Bool UpdateAvailability()

--EMPLOYEE CONTACT--



Description:

This page will allow our users to check on contact information for other employees. The information given will be limited to employees of a similar position ID. There is no reason to contact other employees for non-professional purposes, therefore we will not be providing unnecessary information.

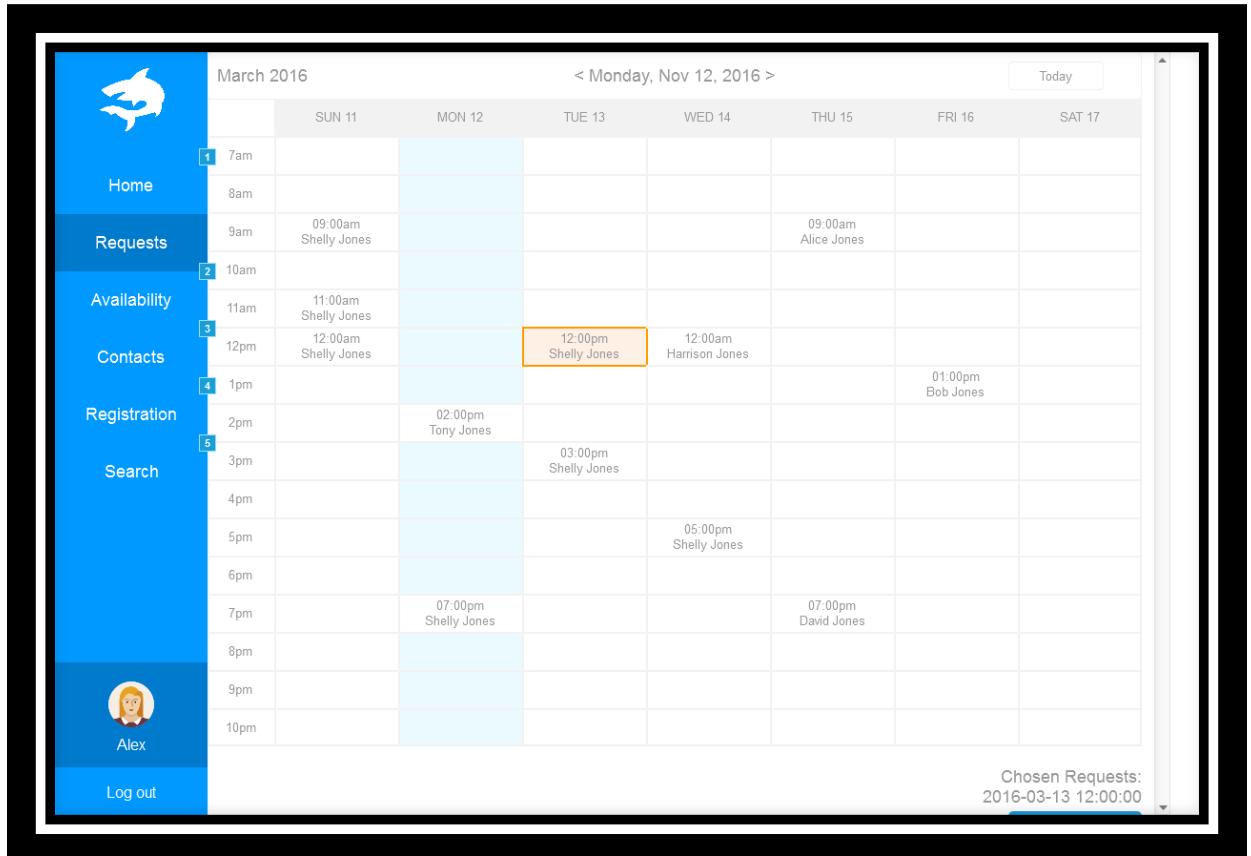
Stub-Calls:

1.) Accepts employees positionID (will be stored as a session variable for easy access)

2.) Displays the common PositionID contact information in a structured manor

Void DisplayContactInfo(int positionID)

--EMPLOYEE REQUEST OFF--



Description:

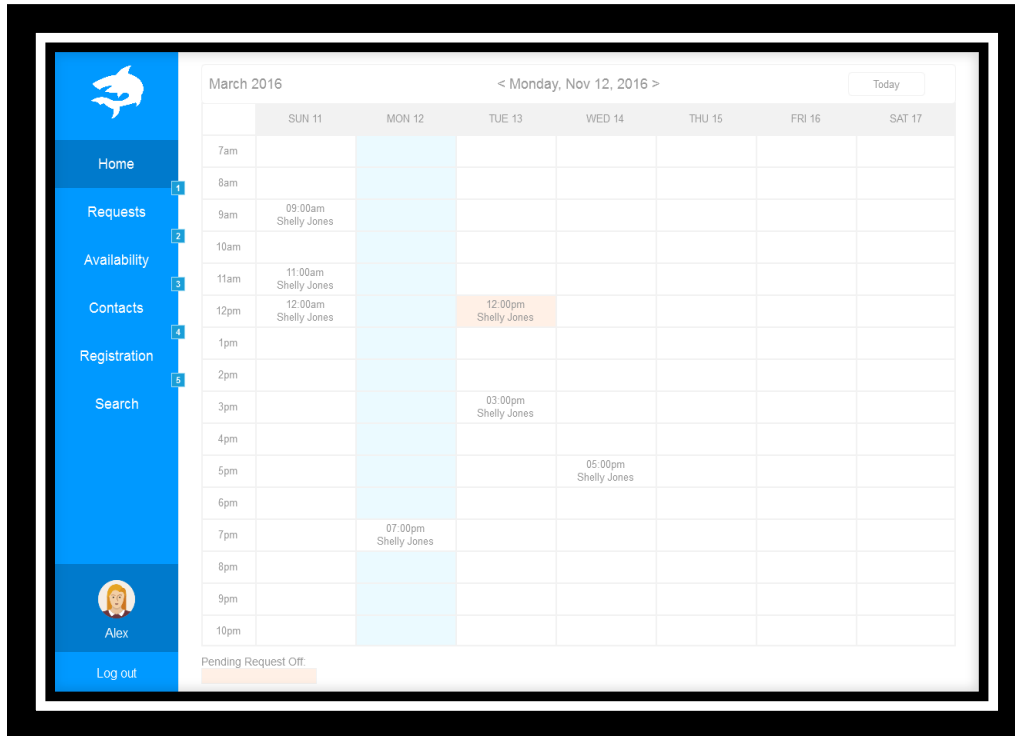
This page will allow our users to submit request off requests. A simple form will be present to allow the user give us information on the start date, and the end date of their request off. A text box will also be available for the employee to give a description as to why they are requesting off.

Stub-Calls:

- 1.) Accepts an associative array that stores the information with regards to request off
- 2.) Does minor error checking on dates to ensure that date is in the future
- 3.) On success, request off is stored in the Database to be approved/denied by the manager

Bool submitRequestOff(String requestOffInfo[])

--MANAGER HOME--



Description:

Upon login, our managers will be able to view the schedules for each type of employee in an organized manor.

Stub-Calls:

- 1.) Accepts positionID, returns nothing

2.) Runs a strategic query on our Schedule table to print the information out in an organized

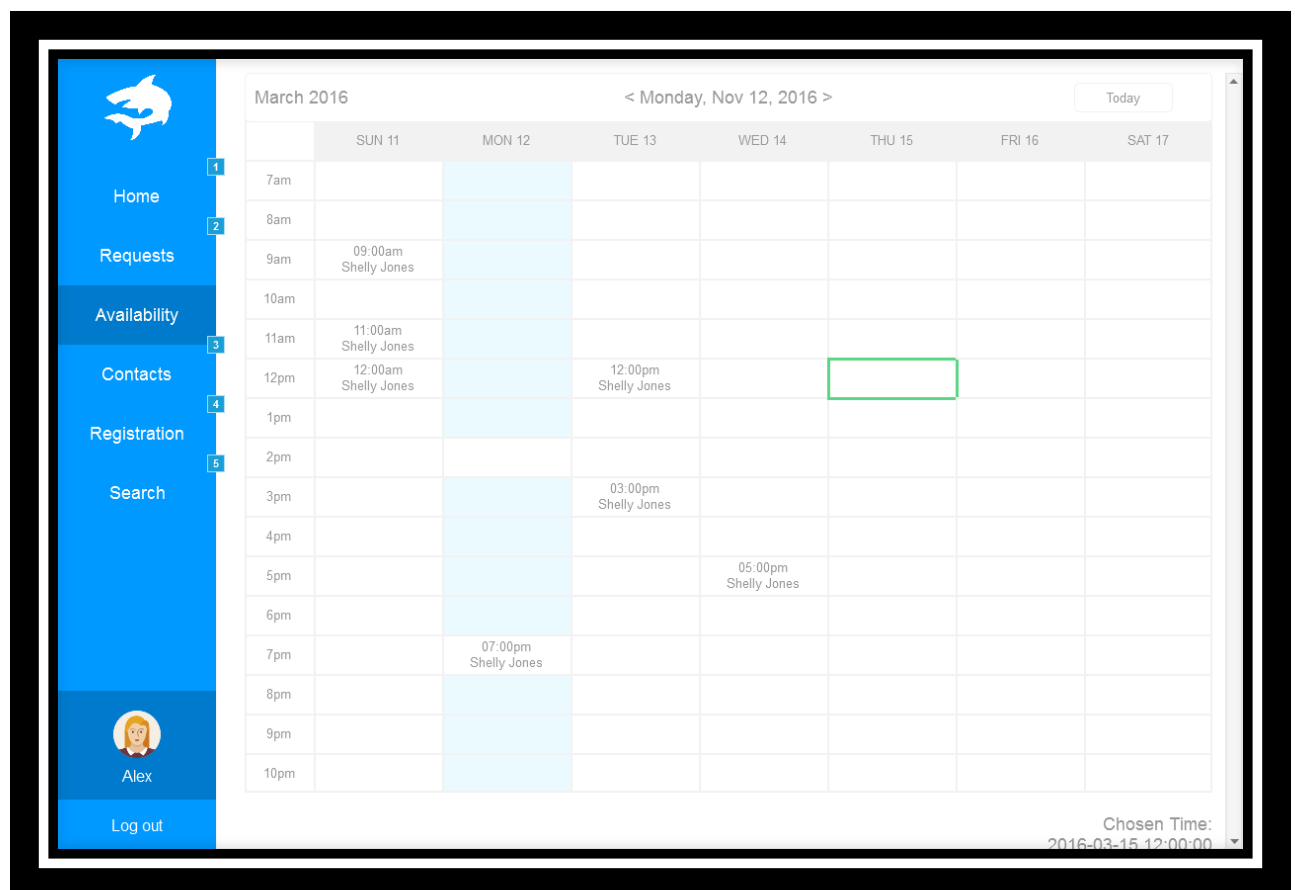
fashion viewable for the manager (ALL SCHEDULES ARE PRINTED)

Void LoadSchedule(int positionID)

1.) Logout Accepts nothing, and returns nothing

2.) This function will remove all SESSION variables we currently have present

--MANAGER AVAILABILITY--



Description:

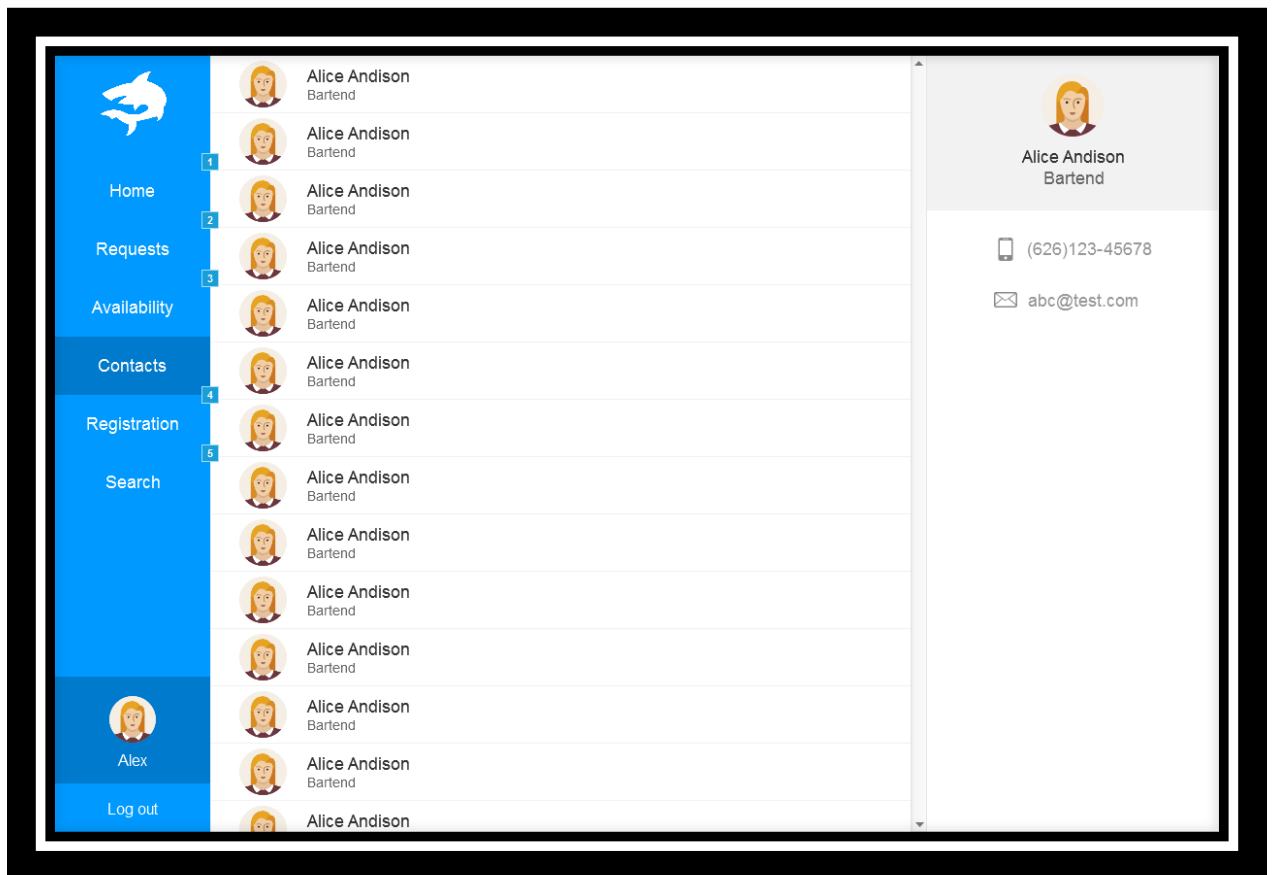
This page will allow our managers to approve/deny any pending changes to employees availability.

Stub-Calls:

- 1.) This function accepts the action (Approve or Deny) the employee ID to alter their availability, and an array storing the new availability
- 2.) If action is Deny, then we remove this request from the table, and return
- 3.) On Accept, Update table and return

Bool ManagerAvailability(String action, int employeeID, String availability[])

--MANAGER CONTACT--



Description:

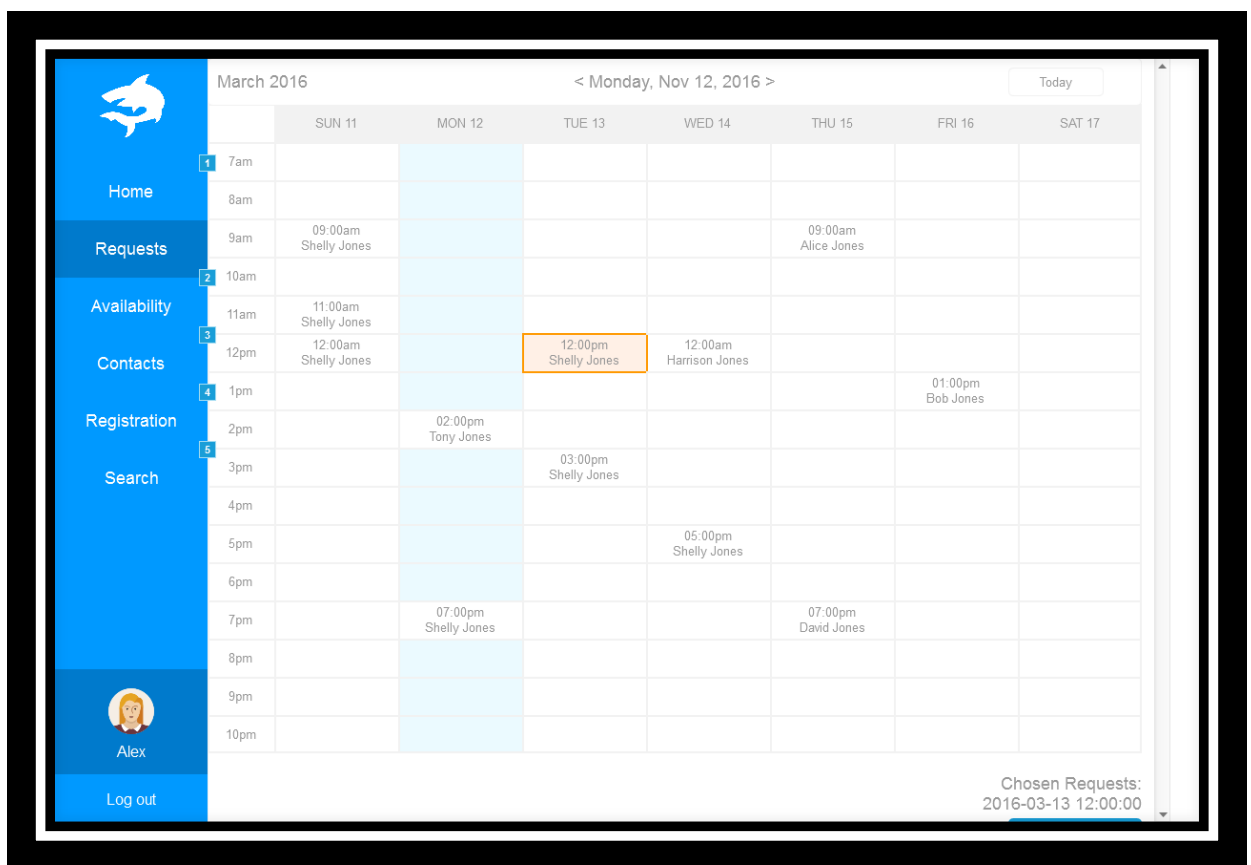
This page will allow our users to check on contact information for everyone. Managers will have the privilege to view ALL Employees contact information no matter what position they have.

Stub-Calls:

- 1.) Accepts employees positionID (will be stored as a session variable for easy access)
- 2.) Displays the common PositionID contact information in a structured manor

Void DisplayContactInfo(int positionID)

--MANAGER REQUEST OFF--



Description:

This page will allow our manager to approve or deny any request off

Stub-Calls:

- 1.) Accepts associative array with request off info and approve or deny
- 2.) Action is deny, we delete this request from our table
- 3.) Action is accept, we update information in Database to not allow that employee to // be scheduled during the time given.

Bool ActionRequestOff(String info[], String action)

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 DOCUMENT), 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
 - a. NO SQL INJECTIONS
 - b. 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.

--EDGE CASES--

- **Login Page:**
 1. Email
 - a. The user inputs something that is not an Email address (i.e. the string has not @ symbol).
 - b. The user inputs nothing in the field.
 - c. The user inputs an Email address that is not in the database.
 2. Password

- a. The user inputs nothing in the field.
 - b. The user inputs a password that does not match up with the password associated with the given Email from the Email field.
- **Code Registration:**
 - 1. Registration Code
 - a. The user inputs nothing into the field.
 - b. The user inputs a code that doesn't exist.
- **Register Basic:**
 - 1. First Name/Last Name
 - a. Can't have any numbers.
 - b. Can't contain any special characters.
 - c. Must be at least one character long, and no longer than twenty-five characters.
 - d. The user does not put any input in.
 - 2. Birthday
 - a. The user inputs any character that is not a number.
 - b. The user inputs nothing.
 - c. The user manually types in a date that hasn't passed yet.
 - d. The user inputs a date that is from less than 18 years ago.
 - 3. Password
 - a. The password is not at least eight characters.
 - b. The password does not include at least one capital letter.
 - c. The password does not include at least one special character.
 - d. The user inputs nothing for the password.
 - 4. Confirm Password
 - a. The password does not match the password field.
- **Register Contact:**
 - 1. Email
 - a. The entered e-mail address can't be validated.
 - b. The user doesn't enter an e-mail address.
 - c. The user enters something that not an email address (i.e. forgets the @ symbol).
 - 2. Confirm Email
 - a. The user leaves the space empty.
 - b. The entry does not match the input in the Email field.
 - 3. Cell Phone Number
 - a. The user doesn't input a phone number.
 - b. The user puts in a phone number that includes letters.
 - c. The user puts in a phone number that includes special characters.
 - 4. Zip Code
 - a. The user doesn't input anything.

- b. The user puts in a string that includes letters.
 - c. The user puts in a string that includes special characters.
 - d. The user inputs a string that is not 5 characters long.
- 5. State
 - a. The user puts in a state that doesn't contain the zip code they entered previously.
- **Register Availability**
 - 1. Availability
 - a. The user leaves all selections as "Not Available".

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	3/27/2016	Jeremy Warden	Finalized: ERD, DDL, UI Updated: Table of contents, Glossary,

			Testing Scenarios. (SPRINT 1)
11	4/10/2016	Kaitlin Anderson	Update Requirements Analysis Document
12	4/10/2016	Joshua Lewis	Create DML (SPRINT 2)
13	4/10/2016	Han Chen	Create UI (SPRINT 2)
14	4/10/2016	Jeremy Warden	Create stub calls (SPRINT 2)
15	4/10/2016	Jeremy Warden	Updated UI portion in document
16	4/10/2016	Kaitlin Anderson	Documentation of Sprint 2
17	4/17/2016	Jeremy Warden	Uploaded Login Page
18	4/17/2016	Han Chen	Uploaded Registration Pages (Steps 1 and 2)
19	4/17/2016	Kaitlin Anderson	Uploaded Registration Page (Step 3)
20	4/17/2016	Jeremy Warden	Uploaded Registration Page (Step 4)
21	4/17/2016	Joshua Lewis	Edge Case Testing
22	4/17/2016	Kaitlin Anderson	Documentation of Sprint 3

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.

LAMP Stack

Includes the four open-source components: the Linux operating system, the Apache HTTP Server, the MySQL relational database management system (RDBMS), and the PHP programming language.

Edge Case Testing

Testing for problems that occur at an extreme operating parameter