**Blood Bank Inventory Manager**

By: Erich Elshoff (Junior, IT)

Jacob Stade (Senior, CS)

Oakland University

CSI 3450: Database Design and Implementation

Presentation Link: <https://www.youtube.com/watch?v=j_YjJKG949Y>

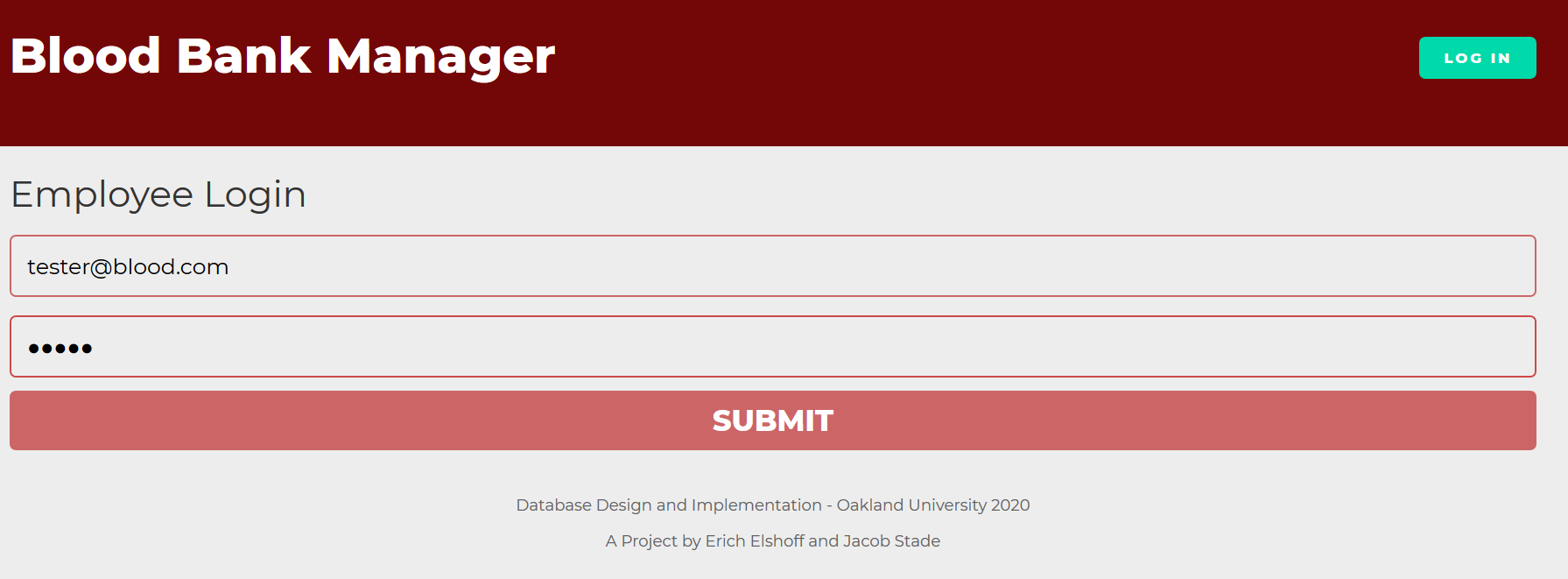
**1. User Manual**

The home page is where the user (who for this project is an employee of the blood bank) will be directed to when they enter localhost/BloodBank into the web browser. From here the user should click the login button to be directed to the login page. Any attempt to click on the manage, register, or donate buttons will cause the user to be redirected back to this page, as they require a logged in user to be accessed.

**Fig1. Home Page - localhost/BloodBank**

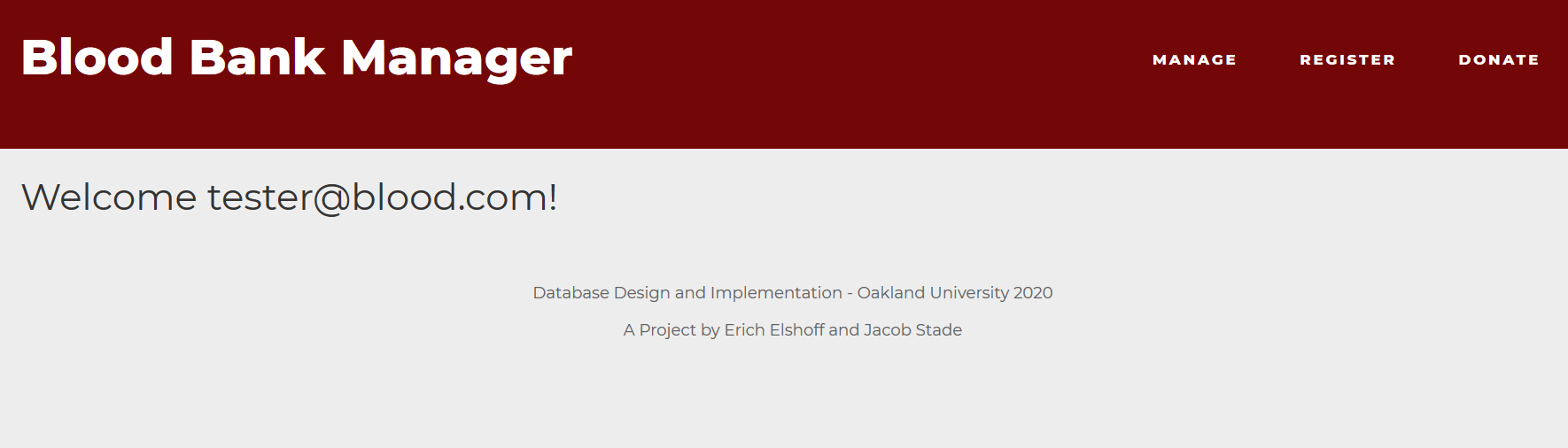


**Fig 2. Login Page**

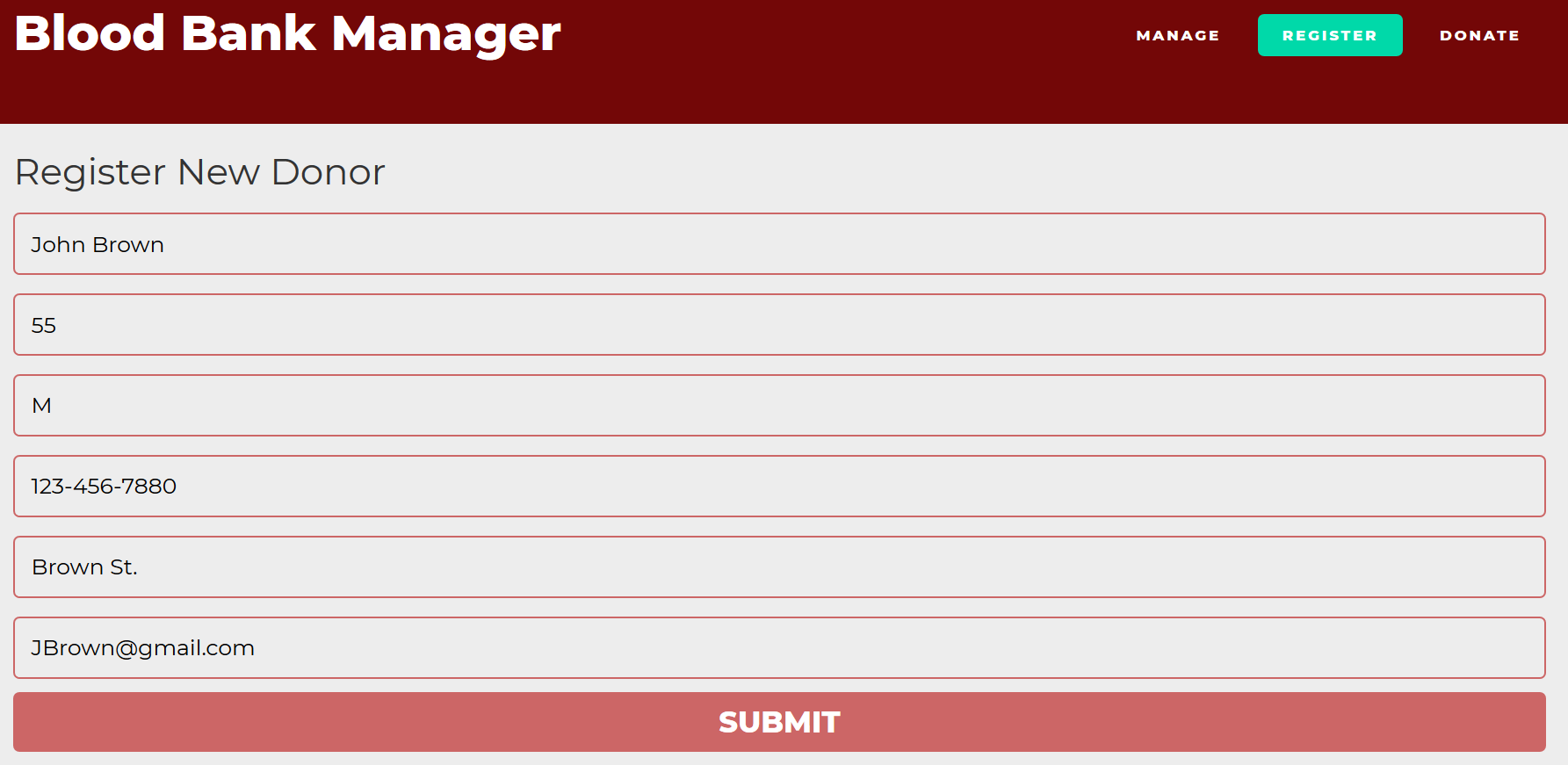


The employee will enter their credentials from this page to access the rest of the services. These credentials will be provided to them by an administrator so no registration button or function is necessary at the time. Incorrectly entered credentials will prompt an “incorrect email or password message”. Otherwise, the user will be directed to the welcome page seen in figure 2.1.

**Fig. 2.1 Welcome Page**

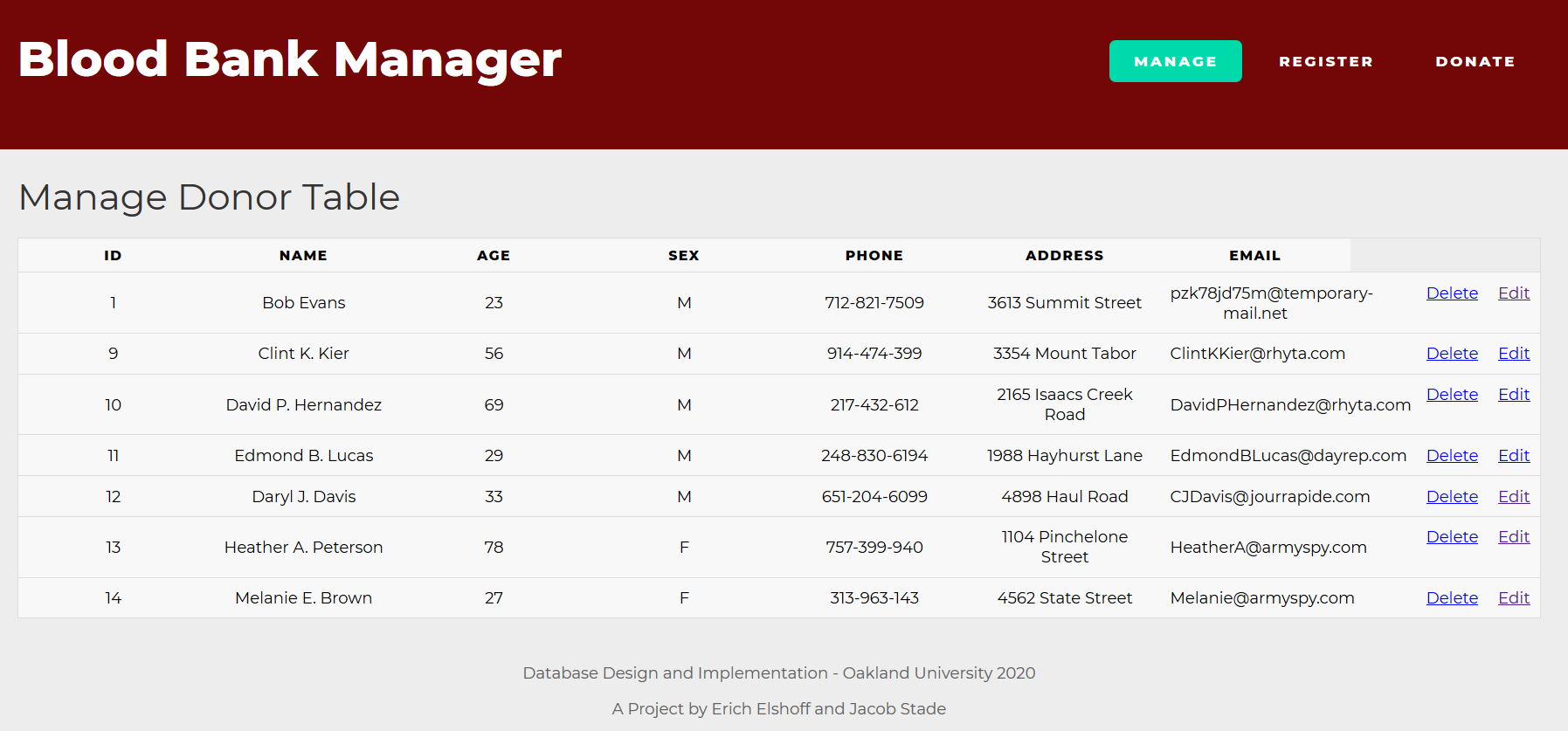


**Fig.3 Register Page**



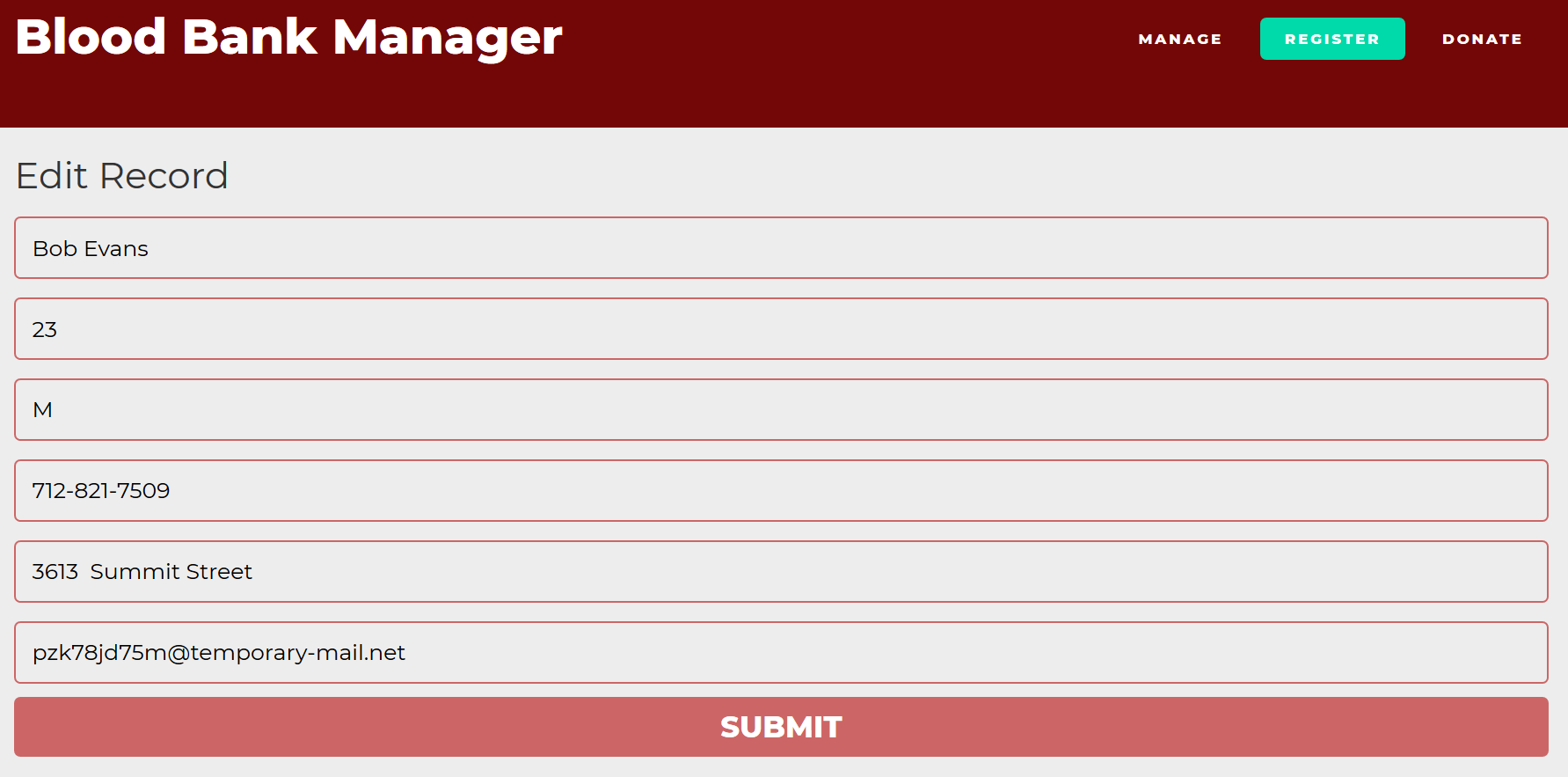
Employees can use the form on this page to enter data about blood donors into the database. The donate page works the same, but is used for entering data about the actual blood.

**Fig 4. Manage Page**



This page pulls the information from the DONOR table in the database and displays it in a table. The delete and edit buttons can be used to modify the table.

**Fig 4.1 Edit Page**



Clicking the edit button will direct you to the editor, which displays the values of the selected row in a form. Modifying the form entries will update the data in the database when the submit button is clicked.

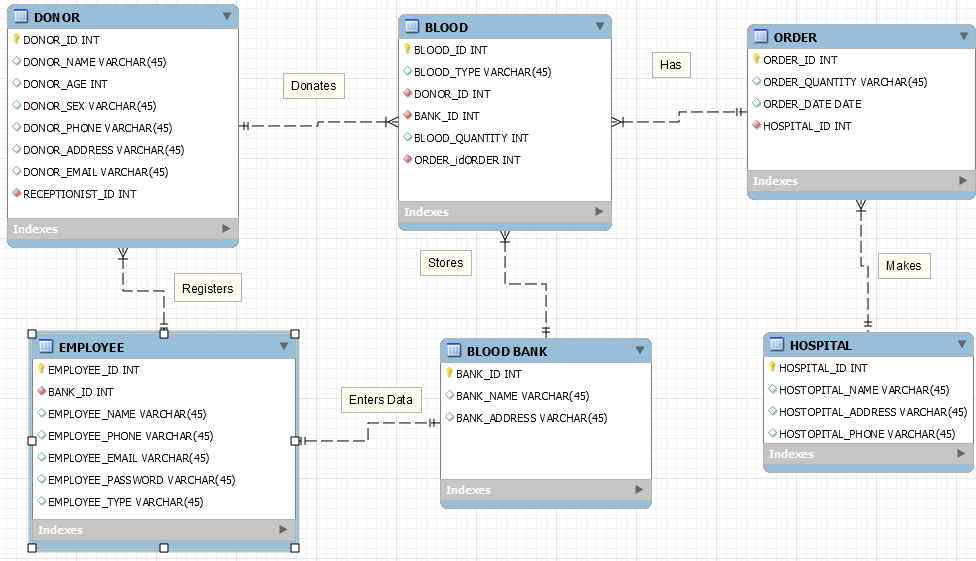
**2.1 Problem Statement:**

We propose to build a Blood Donation web application that will rely on a relational database system. This application will ease the process of gathering and storing donor blood information and will help in the management of inventories at blood banks.

**2.2 System Requirements:**

This system is designed to be used by administrators at blood banks to record and store donor information in a database. Donors must register with the receptionist, who will register their information into the database. Administrators will be able to add, delete, and modify database entries as they see fit. It is important that administrators are able to query the database to determine the status of blood bank inventories. The blood bank manager will also process orders for blood from hospitals and update the database accordingly.

**2.3 Conceptual Database Design:**

****

**2.4 Functional Requirements:**

All employee functions:

* All employees shall be able to login to the system
  + - Tables involved: EMPLOYEE
    - Input: EMPLOYEE\_EMAIL and EMPLOYEE\_PASSWORD
    - Output: redirects employee to appropriate page and gives administrators access to administrator pages

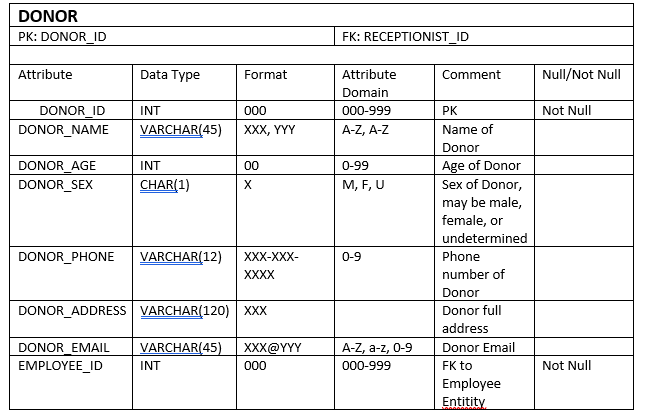
Receptionist Functions:

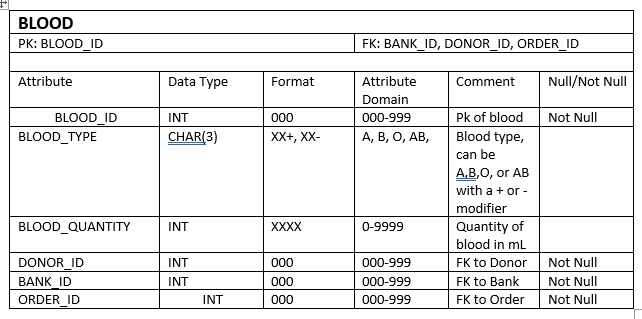
* The Receptionist shall be able to register donor information including …
  + Donor name, age, sex, address, phone, email
    - Tables involved: DONOR
    - Input: Employee inputs donor information into form which updates donor table
    - Output: Success message
  + Donor blood type, blood quantity
    - Tables involved: BLOOD
    - Input: Employee inputs blood information into form which updates BLOOD table
    - Output: Success message

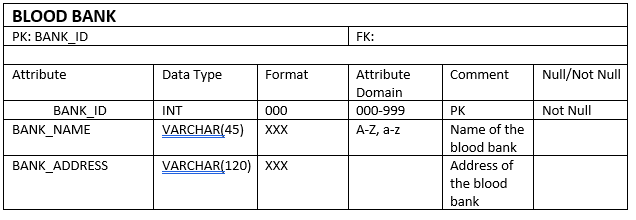
Administrative Functions:

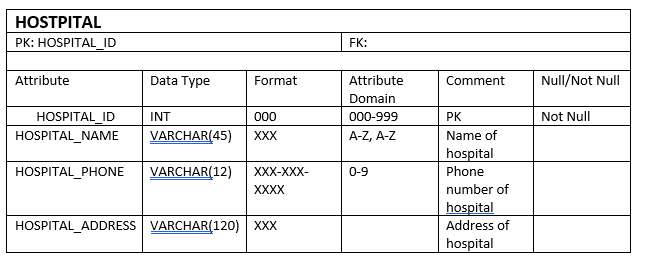
* Administrators shall be able to delete …
  + Donor personal information
    - Tables: DONOR
    - Input: Select a donor by DONOR\_ID from a given list and use the delete button
    - Output: Remove the selected table entry and display a success message.
  + Donor blood information
    - Tables: BLOOD
    - Input: Select a blood donation from a given list and use the delete button
    - Output: Remove the selected table entry and display a success message.
  + Blood bank orders
    - Tables: ORDERS
    - Input: Select an order from a given list and use the delete button
    - Output: Remove the selected table entry and display a success message.
* Administrators shall be able to update...
  + Donor personal information
    - Tables: DONOR
    - Input: Select a donor by DONOR\_ID from a given list and use the update button, which prompts a new form
    - Output: Override the selected table entry with information entered in the form and display a success message.
  + Donor blood information
    - Tables: BLOOD
    - Input: Select a blood donation from a given list and use the update button, which prompts a new form
    - Output: Override the selected table entry with information entered in the form and display a success message.
  + Blood bank orders
    - Tables: ORDERS
    - Input: Select an order from a given list and use the update button, which prompts a new form
    - Output: Override the selected table entry with information entered in the form and display a success message.
* Administrators shall be able to retrieve …
  + Donor personal information
    - Tables: DONOR
    - Input: A list of all Donors and their information or enter specific DONOR\_ID, DONOR\_NAME, DONOR\_PHONE
    - Output: List of donors that match the given attributes
  + Donor blood information
    - Tables: BLOOD, DONOR
    - Input: A list of all Blood donations or BLOOD\_TYPE, DONOR\_ID
    - Output: List of blood donations that match the given attributes
  + Hospital information
    - Tables: HOSPITAL
    - Input: A list of Hospitals and their information or HOSTPITAL\_ID, ORDER\_ID
    - Output: List of Hospitals and their orders

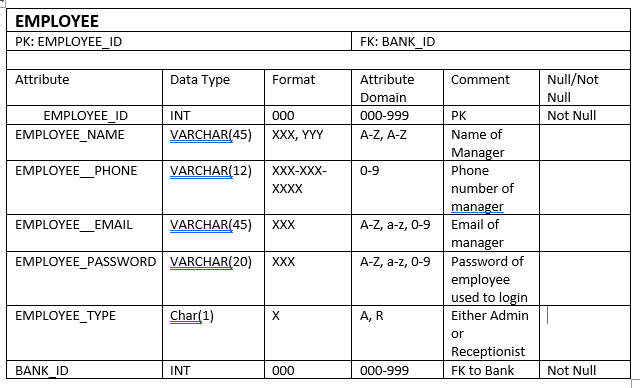
**2.5 Logical Design:**

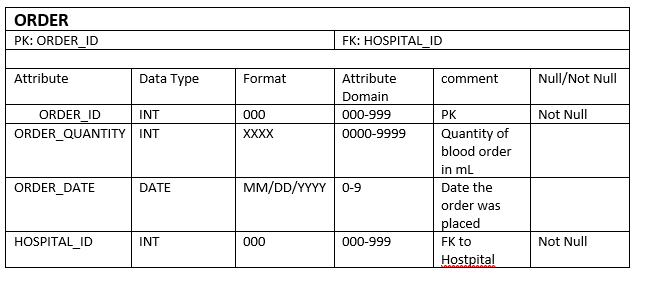












**2.6 Application Programs Design:**

**Login-**

When an employee fills in the email and password forms and presses the submit button, the data is sent to the login.php module which compares the data of the form to the data that exists in the database. To ensure there is some basic security, the login.php code will check to make sure that both the fields are filled out. After the login data is confirmed to be accurate, the program will save the session id as logged in, which will provide access to the other pages of the web app.

Pseudo code: from login.html -

Html form

<form action="login.php" method="post">

<input name="email" type="text" class="feedback-input" placeholder="E-Mail" />

<input name="password" type="password" class="feedback-input" placeholder="Password" />

<input type="submit" value="SUBMIT" /> </form>

When the submit button is pressed the data of the form is sent to the page login.php

<form action="login.php" method="post">

Get values from form and compare to database

if ( $stmt = $conn->prepare( 'SELECT EMPLOYEE\_ID, EMPLOYEE\_PASSWORD FROM EMPLOYEE WHERE EMPLOYEE\_EMAIL = ?' ) ) {

$stmt->bind\_param( 's', $\_POST['email'] );

$stmt->execute();

Store the result so we can check if the account exists in the database.

$stmt->store\_result();

if ( $stmt->num\_rows > 0 ) {

$stmt->bind\_result( $id, $password );

$stmt->fetch();

Account exists, now we verify the password.

if ( $\_POST['password'] === $password )

Using the login credentials all pages besides index.html will start with the following php code to check if they are first logged in, then if they aren't, they are redirected to the index page.

session\_start();..

if (!isset($\_SESSION['loggedin'])) {

echo 'Please Login to view this page';

header('Location: index.html');

**Receptionist Donor Registration-**

The register.php page will take data from a form and pass it to the insert.php page. The insert.php page will first check to make sure that no fields have been left empty, then it will write the data to the database. Otherwise, it will prompt an error message if the fields are empty.

Pseudo Code: Register.php / Donate.php → insert.php

First variables are assigned to the data entered into the forms.

$name = filter\_input( INPUT\_POST, 'name' );

$age = filter\_input( INPUT\_POST, 'age' );

$sex = filter\_input( INPUT\_POST, 'sex' );

$phone = filter\_input( INPUT\_POST, 'phone' );

$address = filter\_input( INPUT\_POST, 'address' );

$email = filter\_input( INPUT\_POST, 'email' );

Check if any fields are empty.

if ( !empty( $name ) ) {

else {

echo "Name should not be empty";

die();

Then insert it into the database

$sql = "INSERT INTO donor (`DONOR\_NAME`, `DONOR\_AGE`, `DONOR\_SEX`, `DONOR\_PHONE`, `DONOR\_ADDRESS`, `DONOR\_EMAIL`, `EMPLOYEE\_ID`) VALUES ('$name', '$age', '$sex', '$phone', '$address', '$email', '1')";

Other insertable functions that act the same are inserts into the BLOOD table and ORDER table.

**Retrieval Functions**

Admins will be able to retrieve information about donors, blood donations, orders, and hospitals. On the admin panel there will be buttons for these tables which, when clicked, will display the selected table. Currently only the donor retrieval call is functionion, but future iterations would be able to call all tables. The manage.php page will select entries from the DONOR table and put it in a html table. This table also includes edit and delete buttons to call those functions on the selected row.

Pseudo-code for selecting DONOR table: from manage.php

First, select from the donor page and assign it to a result variable.

$sql = "SELECT `DONOR\_ID`, `DONOR\_NAME`, `DONOR\_AGE`, `DONOR\_SEX`, `DONOR\_PHONE`, `DONOR\_ADDRESS`, `DONOR\_EMAIL` FROM `donor`";

$result = $conn->query($sql );

Then, run a loop to fetch results until there are no more rows in the database.

if ($result->num\_rows > 0) {

echo "<table><tr><th>ID</th><th>Name</th><th>AGE</th><th>SEX</th><th>PHONE</th><th>ADDRESS</th><th>EMAIL</th></tr>";

// output data of each row

while($row = $result->fetch\_assoc()) {

echo "<tr>";

echo "<td>".$row["DONOR\_ID"]."</td>";

echo "<td>".$row["DONOR\_NAME"]."</td>";

echo "<td>".$row["DONOR\_AGE"]."</td>";

echo "<td>".$row["DONOR\_SEX"]."</td>";

echo "<td>".$row["DONOR\_PHONE"]."</td>";

echo "<td>".$row["DONOR\_ADDRESS"]."</td>";

echo "<td>".$row["DONOR\_EMAIL"]."</td>";

echo "<td style='float: right;'><a href='edit.php?id=".$row['DONOR\_ID']."'>Edit</a></td>";

echo "<td style='float: right;'><a href='delete.php?id=".$row['DONOR\_ID']."'>Delete</a></td>";

echo "</tr>";

}

echo "</table>";

} else {

echo "0 results";

**Admin Delete Function:**

Admins can delete a row of data in the DONOR table by pressing the delete button on the corresponding DONOR\_ID. This sends the DONOR\_ID of the row that correspond to the button you pressed and sends that data to the delete.php module.

Pseudo code: manage.php → delete.php

Code on the manage.php form that displays the delete button and parses the corresponding DONOR\_ID to the delete.php file.

echo "<td style='float: right;'><a href='delete.php?id=".$row['DONOR\_ID']."'>Delete</a></td>";

Assigns the id from the delete button row to a variable.

$id = $\_GET['id'];

Then the row is deleted from the database.

$sql = "DELETE FROM DONOR WHERE DONOR\_ID = $id";

If the deletion is successful the user is redirect to the manage.php page, otherwise an error message is displayed.

if (mysqli\_query($conn, $sql)) {

mysqli\_close($conn);

header('Location: manage.php');

exit;

} else {

echo "Error deleting record";

**Admin Update Function:**

Admins can select the edit button on the admin panel which will override the information in that row with the new information entered in the form.

Pseudo-code: manage.php → edit.php → update.php

First the DONOR\_ID from the manage.php form corresponding to the row where the edit button was pressed is passed to a variable on the edit.php page.

$id = $\_GET['id'];

Then the data from that DONOR\_ID is selected from the database.

$sql = "SELECT \* from DONOR where DONOR\_ID='$id'";

Then assign those values from the database to variables.

$row = $result->fetch\_assoc();

$name = $row["DONOR\_NAME"];

$age = $row["DONOR\_AGE"];

$sex = $row["DONOR\_SEX"];

$phone = $row["DONOR\_PHONE"];

$address = $row["DONOR\_ADDRESS"];

$email = $row["DONOR\_EMAIL"];

Then display a form with the values from the database as placeholders.

echo "<form action='update.php' method='POST'>

<input type='hidden' name='id' value='$id'>

<input name='name' type='text' class='feedback-input' placeholder='Name' value='$name' />

<input name='age' type='text' class='feedback-input' placeholder='Age' value='$age'/>

<input name='sex' type='text' class='feedback-input' placeholder='Sex' value='$sex'/>

<input name='phone' type='text' class='feedback-input' placeholder='Phone' value='$phone'/>

<input name='address' type='text' class='feedback-input' placeholder='Address' value='$address'/>

<input name='email' type='text' class='feedback-input' placeholder='Email' value='$email'/>

<input type='submit' value='SUBMIT' />

</form>";

Finally, send the entered values to the update.php page, where the new values in the form are updated in the database.

$sql = "update DONOR set DONOR\_NAME='$name', DONOR\_AGE='$age', DONOR\_SEX='$sex', DONOR\_PHONE='$phone', DONOR\_ADDRESS='$address', DONOR\_EMAIL='$email' where DONOR\_ID='$id'";

And send the user back to the manage.php page if successful or show an error.

echo "Records updated: ".$id."-".$name."-".$age."-".$sex;

} else {

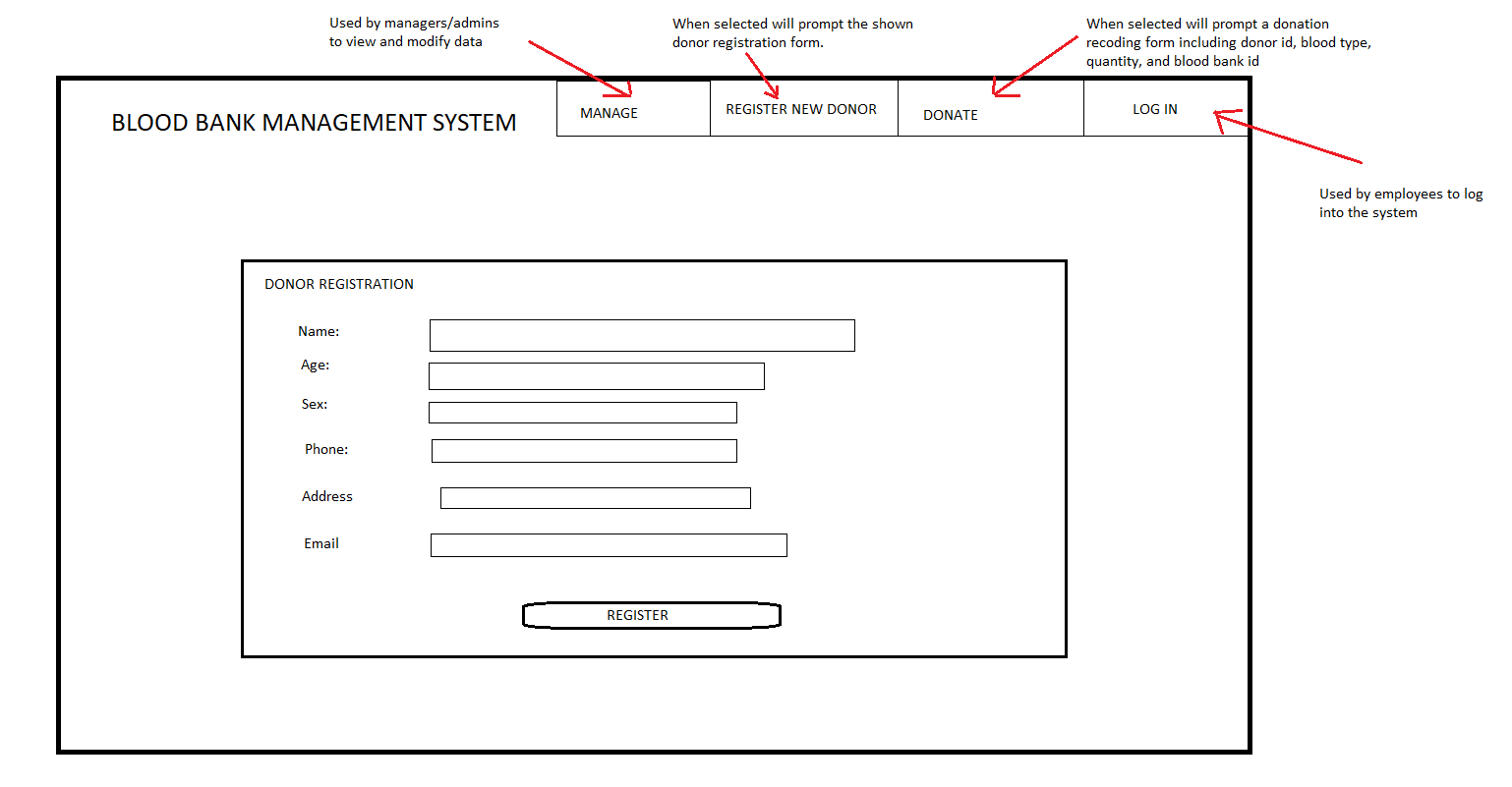
echo "Error: ".$sql."<br>".$conn->error;

}

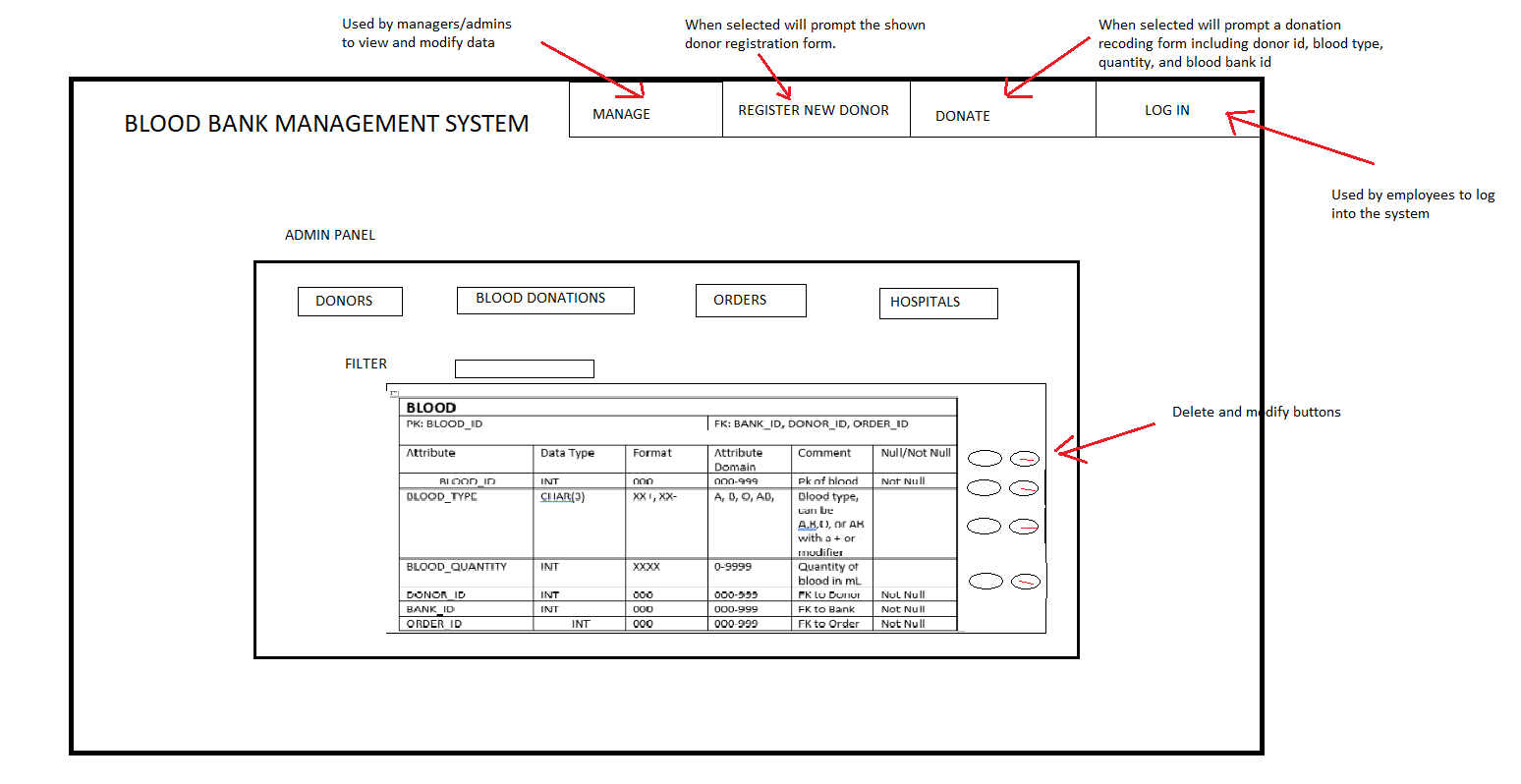
header('Location: manage.php');

**2.7 UI Design:**

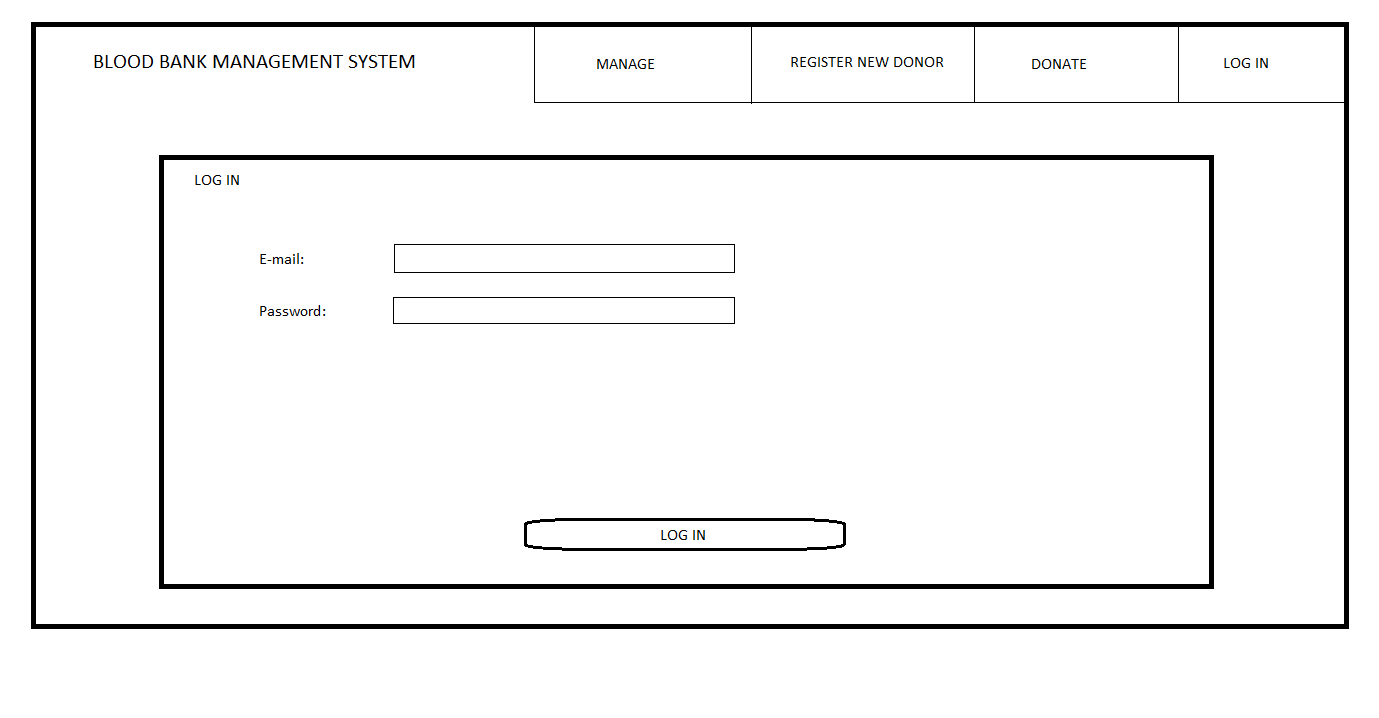
Receptionist View



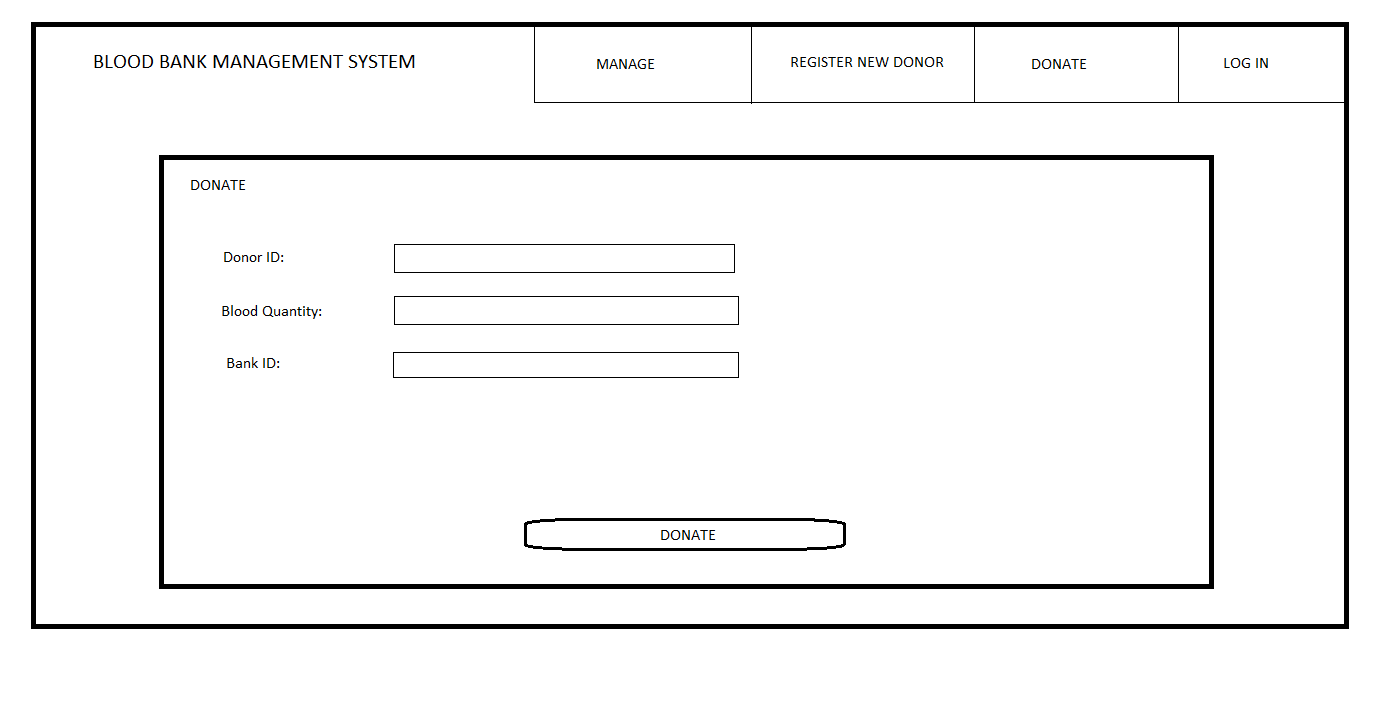
Admin panel



LOG IN selected



DONATE selected



**2.8 Implementation and Testing Plan:**

**Login-**

The login function is to be implemented along with the admin panel, because on successful login you are taken to the admin panel.

This feature will be implemented by Jacob.

To test the login function, login with an Email and password that is recognized by the system. Then attempt to login again with an Email and password combination that is not recognized by the system. The first login attempt should result in being taken to the admin panel. The second attempt should result in a failed attempt and redirect back to the index.

**Donor Registration-**

This feature will be implemented by Jacob.

To test the donor registration function, create and access a new donor record. Then attempt to create a record that is missing required data. The first record will be created and accessed with no issues. The second record will not be created and will ask for the required inputs until all of the form is filled.

**Donate-**

The blood donation function is to be implemented after the donor registration function, because the blood donation function requires a donor ID to exist to be usable.

This feature will be implemented by Jacob.

To test the blood donation function, create and access a new blood donation record. Then attempt to create a record that is missing required data. The first record will be created and accessed with on issues. The second record will not be created and will ask for the required inputs until all of the form is filled.

**Retrieval Functions-**

The retrieval function is to be implemented after the login function, because the retrieval function requires the login function to be usable.

This feature will be implemented by Erich.

To test the retrieval functions, login as an admin and try to retrieve whole tables, groups of records based on components, and single records. The result of each retrieval will contain only the information requested.

**Update Functions-**

The update function is to be implemented after the retrieval function, because the update function requires the retrieval function for testing.

This feature will be implemented by Erich.

To test the update functions, update and then retrieve a record from each table that has an update function. The result of each retrieval will contain the updated record.

**Delete Function-**

The delete function is to be implemented after the retrieval function, because the delete function requires the retrieval function for testing.

This feature will be implemented by Erich.

To test the delete function, delete an existing record and attempt to access it. The expected result is to get no results for a record that does not exist.

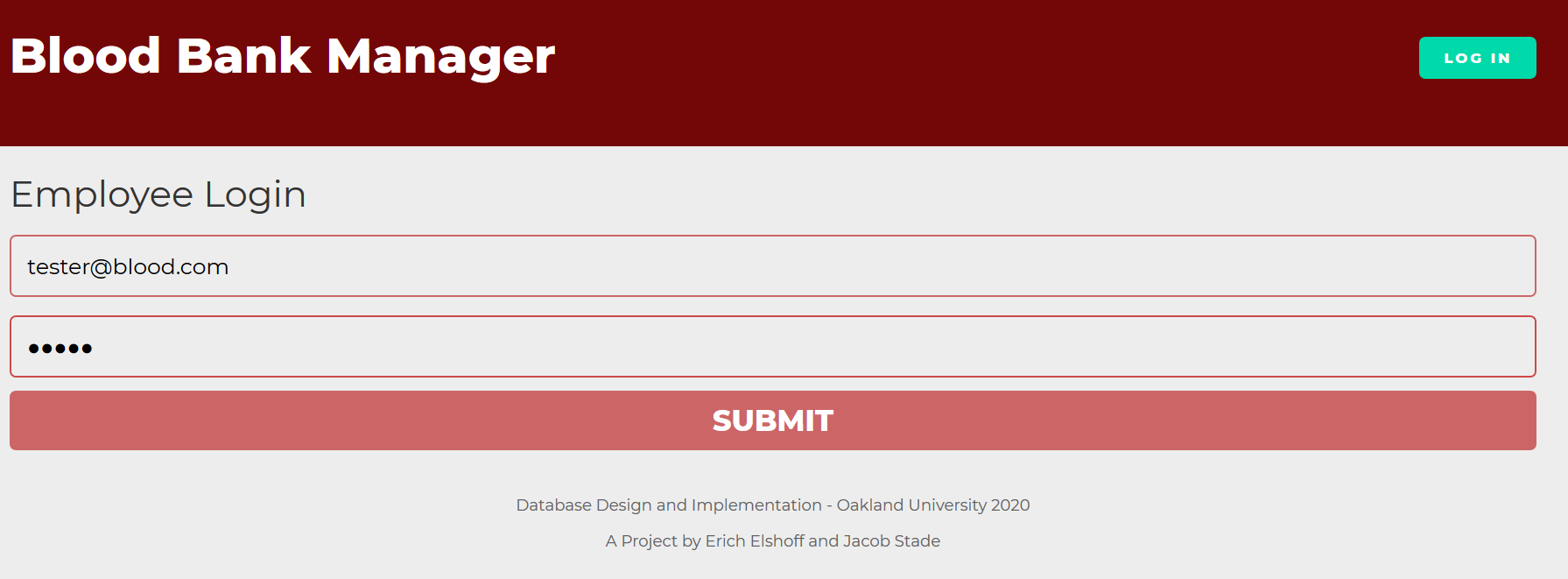
| Phase | Task | Erich | Jacob |
| --- | --- | --- | --- |
| Phase 1 | Time in Hours |  |  |
|  | Problem Statement | 1 | 1 |
|  | System Requirements | 2 | 2 |
|  | Conceptual Design | 3 | 3 |
|  | Functional Requirements | 3 | 3 |
|  | Total Hours: | 9 | 9 |
|  |  |  |  |
| Phase 2 |  |  |  |
|  | Logical Design | 5 | 5 |
|  | Application Design | 7 | 7 |
|  | User Interface | 3 | 3 |
|  | Implementation and Testing | 5 | 5 |
|  | Total Hours: | 20 | 20 |
|  |  |  |  |
| Phase 3 |  |  |  |
|  | User Manual | 4 | 4 |
|  |  |  |  |
|  |  |  |  |

**2.9 Code Listing:**

<https://github.com/erichelshoff/BloodBankDB>

**2.10 Sample Output:**

Startup Page:

Login Page:

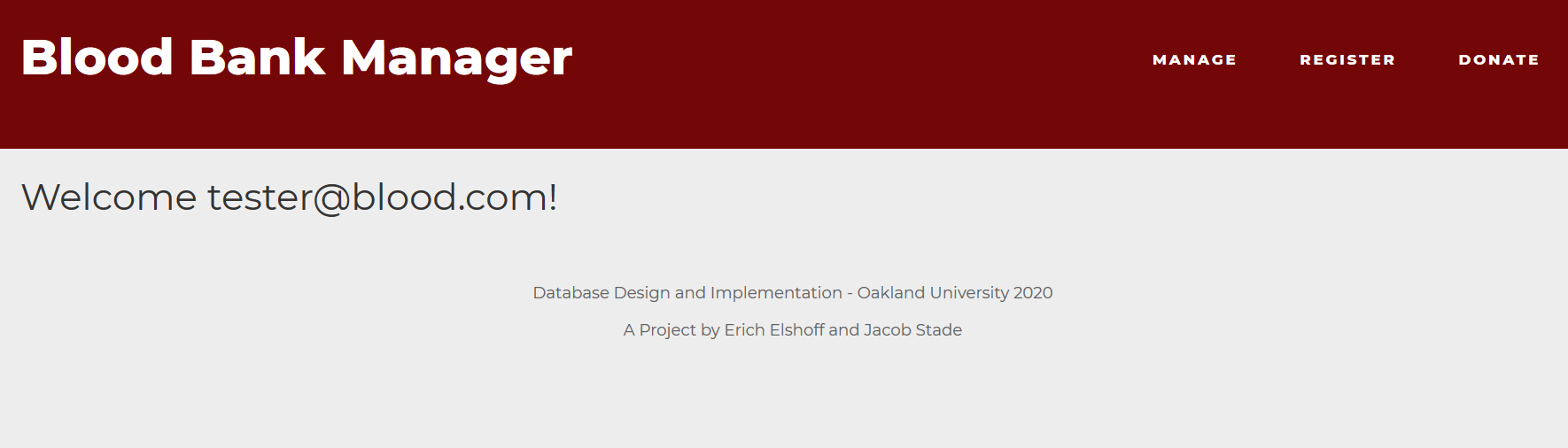
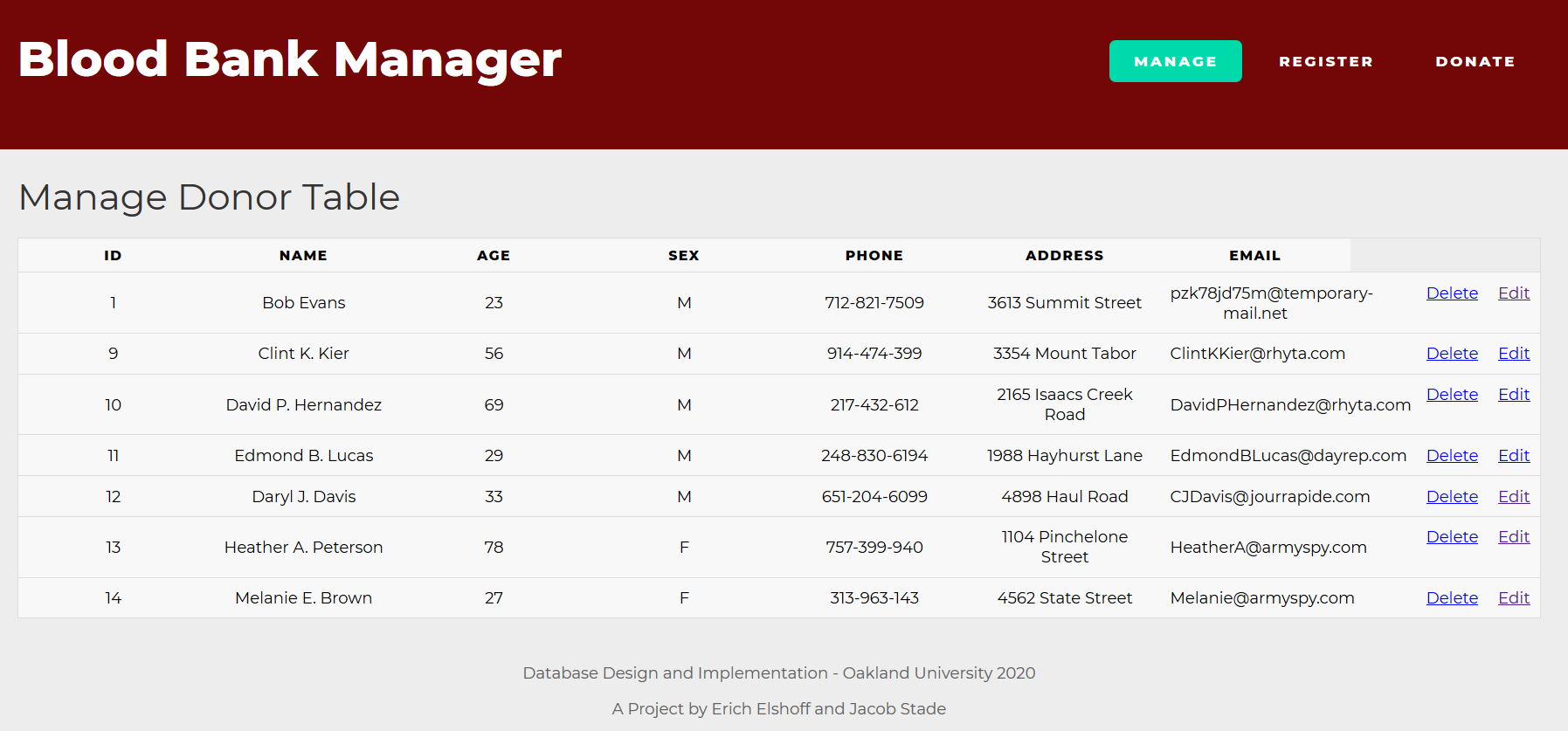
Login Confirmation:

Table Containing Registered Donors: 

Form for Registering New Donors: