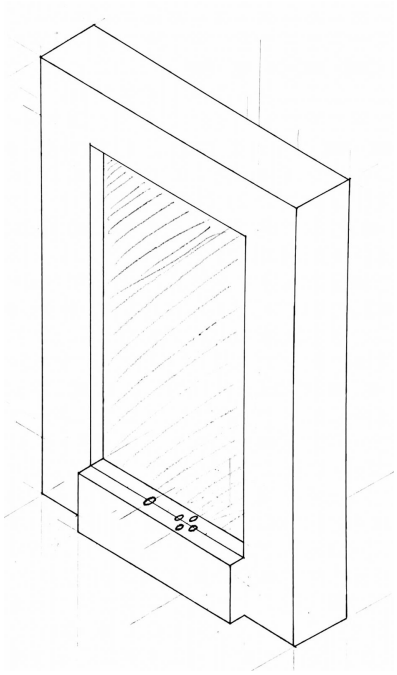
Northwestern | McCORMICK SCHOOL OF ENGINEERING

Department of Electrical Engineering and Computer Science
EECS 347 – Microprocessor System Projects
Spring 2017



Personalizable Display Frame ver 1.0
Final Report and User Guide

Names: Alex Gangwish, Karan Shah
Submitted to: Professor Henschen
Report Due: 6/9/2017

Table of Contents

Executive Summary

Personalizable Display Frame is a remotely managed digital decoration solution designed to help users personalize shared spaces such as offices or hotel rooms. It features two major components: a physical display frame with a screen that can show users' photographs or calendars, and a cloud server and database that manages the overall system and hosts the applications that allow users to interface with the system.

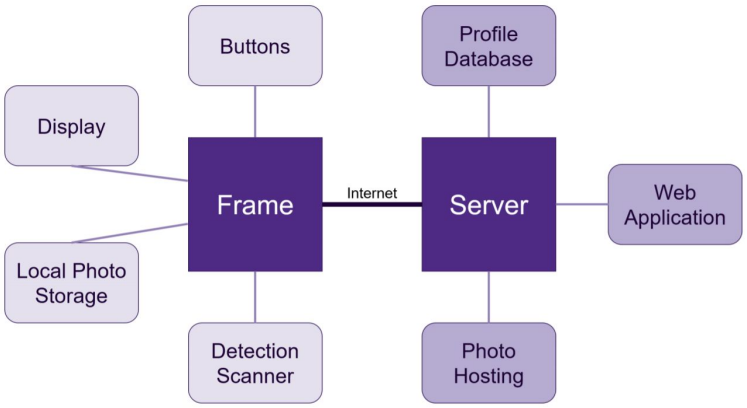


Figure 1: System Block Diagram

This report details our design process and describes our implementation, plus potential alternative implementations, of the following things:

- Frame Construction
  - Raspberry Pi 3 Setup
  - Button Mount
  - Python Scripting
- Server Setup
  - Elastic Cloud Compute server
  - Relational Database Service database
- Web Application
  - HTML, CSS, and JavaScript
  - SQL
  - PHP

In addition to this, Appendix A features a user guide, which explains to both primary users (office workers, hotel guests, etc.) and secondary users (frame owners and managers) how they can interact with our system.

Design Overview

This report details the design and implementation of Personalizable Display Frame, a two-quarter endeavor for EECS 347: Microprocessor System Projects at Northwestern University. Personalizable Display Frame is a system that allows for remote management of a decorative digital display, designed for shared environments like an office or a hotel room. The key feature of the system is that the display is controlled entirely by a remote server, allowing frame owners to manage which user's photos can be shown on the display from anywhere they can access a web browser.

The Personalizable Display Frame design has two main components: the physical frame itself and the remote server that hosts the controlling web application. The server and its associated database act as the master for the overall system by verifying users and controlling which frames get which data. The frames act as slaves, only broadcasting their identity and then receiving data selected by the server. Further details on the implementation of these components are contained in this section. Figure 2 shows the Personalizable display frame in action alongside the management web application at the 2017 Spring Internet of Things Expo at Northwestern University.

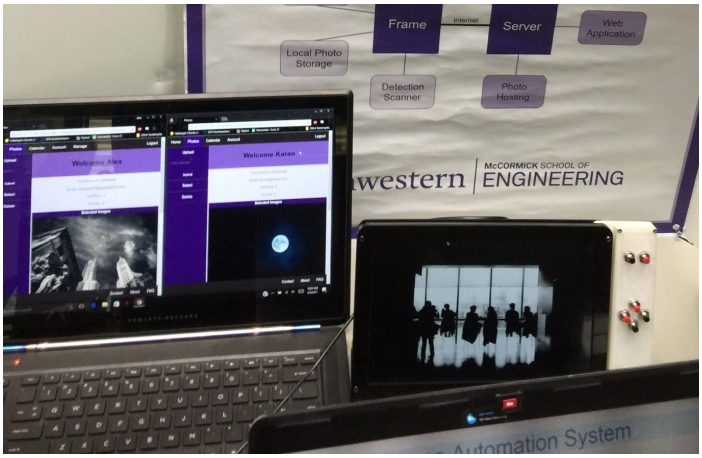


Figure 2: Personalizable Display Frame in action

Frame Construction

The client side of our project was based on interfacing the Raspberry Pi with external buttons to give the user to the chance to change the information displayed on the frame. The frame features a forward facing screen, a Raspberry Pi 3 mounted on the back, and a sidebar button attachment that houses the control button interface.

Raspberry Pi 3

The Raspberry Pi acts as a client in our system, querying the server for information and displaying the data sent back from the server through an HTML web browser. This is displayed on the Personalizable Display Frame screen through an HDMI interface that carries the output from the Raspberry Pi to the input of the screen.

Buttons

The buttons are connected GPIOs of the Raspberry Pi. Four of these buttons act like arrow keys which help the user navigate between their pictures (left and right arrows), or switch the display to calendar mode and back (up and down arrows). Two additional buttons act to mimic the "Delete" and "Enter" keyboard keys.

The enclosure for the button is shown in Figure 3. One end of the enclosure is attached to the enclosure of the screen for providing additional support to the user when pressing the button. The enclosure was 3-D printed and the design can be changed depending on user preference.

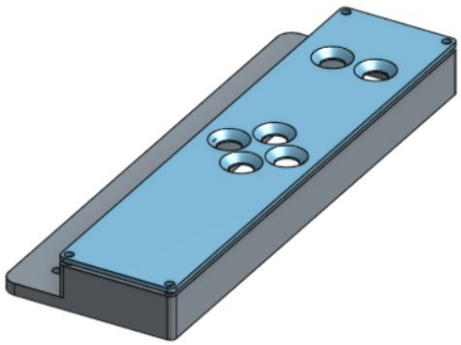


Figure 3: Button Enclosure

Python Script

A Python script is used for configuring the GPIOs to simulate a keyboard press. The GPIO waits to be interrupted by a button press and then outputs a predefined key. We utilized the Python uinput module to emulate keypresses when a user presses a button on the frame. The Python script starts running as soon as the Raspberry Pi is booted.

Server Setup

We implemented our server for the Personalizable Display Frame system as an Amazon Web Services (AWS) cloud server. There were a few main reasons for this decision. First, it would be

a good way to allow the system to easily scale when the product moves past the prototype stage--simply deploy the application the exact same way, except pay for more storage and better load handling (or pay Amazon to manage it themselves). Second, an AWS server can be configured to run essentially the same as many kinds of servers, allowing us to create a cloud server that was quite simple for us to learn to work with.

Elastic Cloud Compute Server

The server itself is hosted using an instance on the AWS Elastic Cloud Compute (EC2) service. It is a Linux server, accessible by SSH (we used the SSH client Putty to access our server on a Windows PC). If a developer has a valid security key they can SSH into the server and interact with it using the Linux command line (note that there are existing softwares that allow developers to interface with a cloud server using a GUI instead, however we did not utilize any in the course of this project).

The server is implemented as an Apache HTTP Server. This means that the basic server setup is the same as if it were an Apache server hosted on any device like a local machine, Raspberry Pi, etc. This would allow our server to be deployed on many different potential devices with little to no changes required to the software itself.

Relational Database Service Database

Linked to our EC2 server is a database hosted using the AWS Relational Database Service (RDS). Our database is setup to interface with SQL queries, a very common method for implementing a relational management database. Much like our cloud server, the methods for interacting with this database are much the same as they would be for interacting with a similar database hosted on a local machine or a different cloud service, providing good portability to the design.

For a more detailed guide on how to set up these instances see Appendix B.

Web Application Development

We designed a web application to run on our cloud server that would be in charge of managing user profiles, frame linkages, and data storage management. The client side interface was created using HTML, CSS, and JavaScript, which should allow it to be supported by all web browsers. The server side code was written using PHP scripts and SQL queries--note that this requires the EC2 server to have PHP installed and the RDS database to have SQL installed.

HTML, CSS, and JavaScript

HTML, CSS, and JavaScript are used to make up the client side interface of the web application (i.e. what the primary and secondary users would see when they access our product). HTML is used to structure the web page, CSS is used to add specific styling, and JavaScript is used to handle any necessary client side scripting.

HTML and CSS are used in close conjunction to produce the visuals that users will actually see when they use our application. For our application, we used CSS to add styling to the <header> and <footer> html tags, allowing us to create top and bottom fixed navigation menus and make our app seem more like an application and less like a blog or other simple web page. An example of this styling can be seen below in Figure 4. This basic layout, once established, was easy to replicate, allowing us to quickly add more pages in the exact same style. Note that the HTML and CSS and easily be adjusted to produce a variety of different outputs and effects depending on the specific requirements, color schemes, or branding of the developers.

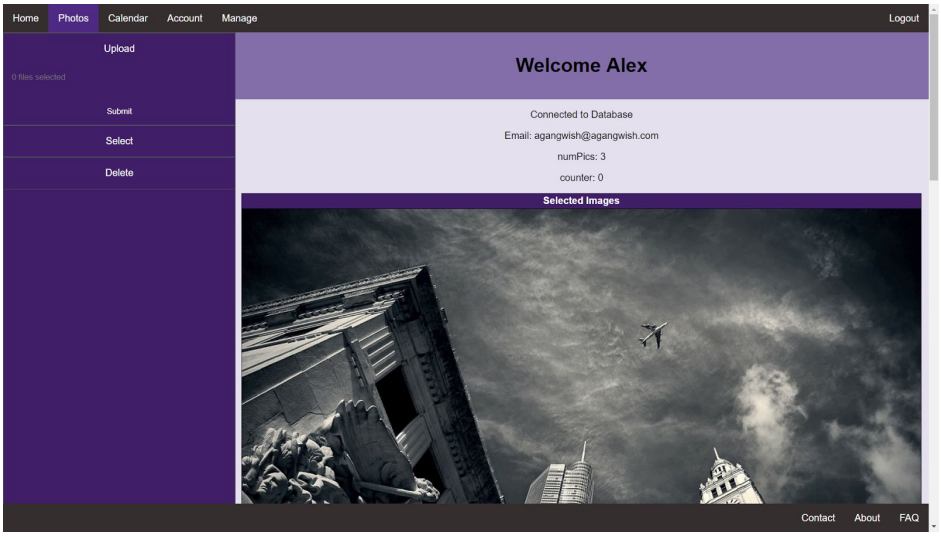


Figure 4: Our general layout

We used this general layout to create pages for managing photographs, managing user account information, managing frames and frame linkages, and general informational pages. By copying pages and then editing only the desired elements on the page, it is easy to ensure that the overall layout and design stays consistent. This technique can be used to create any number of pages as required by the developers to add additional features or information.

In addition to the general web application layout, we also created an HTML page designed only to serve images to frames. The only visual structure required on this page is an image frame,

with CSS styling to scale the image to 100% height and width of the screen automatically. This page also contains invisible links that can be used to navigate to different images or pages altogether.

The crucial piece of JavaScript for this web application was a listener tied to the invisible links on the slideshow page. This listener runs while the web page is open, and navigates to different links depending on the keyboard input it hears. In our application there are 6 invisible links with each one linked to one of the up arrow, down arrow, left arrow, right arrow, enter, or delete keys. When the slideshow page is loaded on a computer browser (for testing), pressing any of these keys will prompt the page to reload with new information (see the PHP section for more information on how this is processed). When the slideshow page is loaded on the frame (as intended for deployment), the Python script on the Raspberry Pi 3 should translate the buttons on the frame to simulated keypresses mirroring these particular keys to achieve the same effect.

SQL

SQL was the language with which we interfaced with our database. By sending specifically worded queries we were able to create tables of data, insert data into those tables, edit that data, and look up the data as needed in our application.

Using SQL CREATE statements we created database tables for users, managers, images, calendar files, and frames. Each of these tables has a set of column names and a set of associated data types representing the type of information they hold. A example of the basic types of information needed for the web application is contained in Table 1.

Table	**Column Name**	**Datatype**
Users	email	VARCHAR
	password	VARCHAR
Managers	email	VARCHAR
Frames	id	INT
	email	VARCHAR
	manage_email	VARCHAR
Photos	image	BLOB

	email	VARCHAR
Calendar	calendar	BLOB
	email	VARCHAR

Table 1: Example database setup

Using a setup such as the one in Table 1 gives you the ability to match a variety of components together. For example, when a user adds a photo or calendar, that file is uploaded to the corresponding table and stored with the user's email--later, these files can be retrieved for that specific user by querying for files with their email. The same holds true for frames and secondary users--when a secondary user (manager) registers their frame, it will be inserted into the Frames table with the manage_email set to match the manager's email. When the manager wants to access their frames via the web application, the frames can be found by searching for frames with their email. When the manager links a primary user to their frame, the frames email field is updated to match the primary user's email, allowing the application to serve that frame the correct images.

SQL INSERT and UPDATE statements allow our web application to add and edit database information. SELECT statements can be used to selectively retrieve entries from the database, allowing the application to find users according to one piece of user information and retrieve all of said user's information. These SQL queries are sent using PHP scripts, and their responses are also handled by PHP scripts (see the next section for a more detailed discussion on how PHP is used in our system).

PHP

To manage our server-side scripting we utilized PHP. There are a few reasons to handle scripting on the server side and not the client side, with the main reason being security (your code does not need to be downloaded and run by the client's browser). PHP is fairly simple to get started with--simply change the file extensions of the web applications HTML pages from .html to .php. When a user attempts to access the web application, navigating to the public URL will cause Apache to load the main index.php page--using the .php extension instead of .html will tell the server to first process anything on the page enclosed in <?php > brackets before serving the page to the user's browser as an HTML file.

PHP allows a great deal of processing to be added to the application. In our case, the primary use was to manage our registration and login system. Whenever a user attempts to access a page in our application, a PHP header at the top of the page first checks their credentials to see if they

have logged in or not. If they have not, they are redirected to the login screen. If they have, they are allowed to continue to their content.

When a user attempts to log in, they are directed to an invisible login-process page that runs a PHP script to determine whether they have good credentials or not (by querying the database with SQL strings). If they do, they are directed to their homepage. If they do not, they are sent back to the login screen with an error message. Once a user is successfully logged in, a User object is created for them (this class can be stored in a separate user-class php file). This object is carried with them through the application until they log out, and keeps track of their basic user information and tells the pages that they are logged in. Once the user logs out, this object is destroyed and the user will need to enter their login credentials again to continue using the application. Note that much of the login and verification logic comes from a textbook tutorial on creating a members only website (see Suehring in References), as we were learning this material on the fly as we built our prototype.

Another key way that PHP is used in our application is to echo different HTML statements in loops or conditionals. This allows us to change the display that users will see as a result of scripting we do on the back end. For example, using this technique allows us to check if a user is registered as a “manager” or not--if they are, we can use PHP to echo the “Manage” button onto their homepage; if they are not then we can do nothing and the button will not show up for them.

PHP is used on the slideshow page to retrieve and display the appropriate images for a particular frame. Each frame comes preloaded with it’s own unique ID. A startup script on the Raspberry Pi 3 opens the frame’s web browser and navigates it to the correct page with it’s ID appended (e.g. example-url.com/slideshow.php?id=xxxxxxxxxx). The slideshow.php page can access this id field in the URL using a $_GET request. Once the server has retrieved the frame’s id in this manner, it can query the database to find if there is a user’s email associated with said frame (as a result of a manager linking a user). If there is, the server checks if the user has any uploaded photos and retrieves one if so. If there is no user or no photos, the server can instead select a default image to display. Using this technique allows all linkages to be controlled by the server and thus controlled by the web application--no interfacing with the frame (other than powering it on) should be necessary to pair the frame with appropriate users.

Finally, PHP is also used in conjunction with our JavaScript listener to control navigation of the slideshow. The invisible links on the slideshow page are each of the form example-url.com/slideshow.php?id=xxxxxxxxxx&next=left, with the “left” part replaced by “right”, “up”, “down”, “delete”, or “enter” as appropriate. This means that when a user pushes a button the frame, the same slideshow page is reloaded with the appropriate “next” field--this

field is used to tell the server what to show on the frame (e.g. “left” = load previous picture, “right” = load next picture, etc.).

PHP can be used in similar ways to add a great deal of different features to the web application, depending on the developer’s needs.

Future Steps

The biggest next step for this project is refinement of our web application. Although we were able to create a functional prototype, there are plenty of areas that could be made more efficient or where more features could be added. Much of these issues stem from the rapid manner in which we developed the application (coupled with our lack of experience in web development), but could be avoided with more careful planning. For example, our application uses a table to hold user photos that contains five columns for different images (for the five images the user can select for display at a time). However, after developing more of our application we decided that this is a somewhat troublesome way to do this--it would be better to have each entry contain only one image and one associated email, and then search for multiple rows instead of finding one row with all five images. Small improvements like this to the structure of the application would provide a cleaner user experience and more efficient code.

Another major future step would be the addition of calendar functionality. While the basic framework for calendar support is already in place, we currently do not have a page capable of actually serving the calendar file in a reasonable and pleasing display. As of right now, attempting to navigate to the calendar page from the slideshow page when using the frame just displays a placeholder “CALENDAR PAGE” string--it would be fairly trivial to plug in a calendar HTML page in place of this.

One major component to our system that we did not have time to develop but would like to see added is linking with the frame on-location (i.e. in the room using the frame hardware, and not relying on a manager to link users with the frame). The two big ideas we had for this were to use a QR scanner (manually link) or an RFID scanner (automatically link). With the QR scanner, the frames display would default to showing a QR phone when unpaired with a user. A primary user could then scan the QR code with their phone, which would pull up a page of the web application in their browser asking them for their login credentials. If they successfully login, the server links them to the frame and their images can be displayed.

The same principle holds for the RFID scanner, except the primary user would not have to interact with the frame at all. Instead, the scanner hardware would detect the user upon entering the same room as the frame (perhaps using an RFID tag in the user’s key fob or ID card). Upon

detection, the frame would send the user's ID to the server. The server would check if the user was allowed to use that frame and if they have photos, and return a photo for display if so.

Conclusion

Our final product has some limitations compared to the design we originally set out to build (most notably the lack of calendar functionality and the lack of local linking by primary users). However, we were very satisfied with the prototype that we did manage to complete. We were able to demonstrate a cohesive user application for both primary and secondary users, and successful control of frame output via a central server- the most crucial tenets of our design. With more development time, it should not be very difficult to add the missing functionality to the existing infrastructure.

References

1. Suehring, Steven and Valade, Janet. *Php, Mysql, Javascript & Html5 All-in-one for Dummies*. Hoboken, NJ: Wiley, 2012. Print.

Appendix A: User Guide

This appendix outlines the information that would need to be provided in a user guide to accompany this product. Please note that through the course of this appendix we will refer to the web application's address as example-url.com--in actual deployment this would be where the official domain name for the application would go.

Primary Users

Primary users are users who utilize the spaces in which the Personalizable Display Frame is designed to go (e.g. offices, hotels, etc.). In the current iteration of this product, the only interactions that concern these users are creating and accessing their accounts, uploading files, and viewing files (they rely on secondary users to link their files to frames for them).

Create An Account

These instructions detail how to create an account if you do not already have one.

1. Go to www.example-url.com
    - You will be directed to the login page
2. Click the button in the top right corner that says "No Account? Register"
    - You will be directed the registration page
3. Fill in the registration form and click "Submit"
    - If you have successfully filled out the form, you will be directed back to the login page
    - If there are errors in your form you will be directed back to the registration page

Access Your Account

These instructions detail how to access your account if you have already registered one.

1. Go to www.example-url.com
    - You will be directed to the login page
2. Enter your email and password in the appropriate fields and click "Submit"
    - If you enter valid credentials, you will be directed to your home page
    - If you do not enter valid credentials, you will be directed back to the login page
3. Navigate through the application
    - The "Photos" tab allows you to upload, view, and delete your photos
    - The "Calendar" tab allows you to upload, view, and delete your calendar
    - The "Account" tab allows you to edit or delete your account

Upload/Edit Files

These instructions detail how to add files to your account for display on a frame.

1. Go to www.example-url.com
    - If you are already logged in you will be directed to your home page
    - If you are not logged in you will be directed to the login page
        i. See section “Access Your Account” for information on how to log in
        ii. Log in and access your home page
2. Click the “Photos” tab
    - You will be directed to the “Photos” page
3. Click the “Upload” button
    - A menu will appear asking you to select an image file from your local directory
        i. Choose a file and click “Submit”
        ii. The page will reload with your new image displayed in the “Selected Images” table
4. Click the “Delete” button
    - You will be navigated to the delete-photos page
        i. Click the “Delete?” button next to all the photos you wish to delete
        ii. Click the “Submit” button
        iii. The page will reload without the delete images--they have been removed from the database entirely

Note that the process for uploading and editing calendar files is exactly the same, except for Step 2 requires you to instead click the “Calendar” tab.

View Your Files on the Display Frame

These instructions detail how to interact with the physical display frame to view your files. Note that in this iteration, the frame must already be linked with your user profile by its manager in order for your data to correctly appear.

- **Default:** The default mode for the display frame upon linking with a user profile and turning on is to show one of the user’s associated photos. If the user has no photos on their profile, the display will show a default image.
    - Pressing the Left button will cause the previous image in the user’s image queue to be displayed
    - Pressing the Right button will cause the next image in the user’s image queue to be displayed

    - If there is only one image associated with the user's profile, the left and right arrows will have no effect.
    - Pressing the Up button will cause the frame to enter Calendar mode
        - Note that currently Calendar mode holds only a placeholder text in place of the actual calendar display, which will be added in future iterations.
- **Calendar:** This section describes how the calendar feature will be implemented in the next version of the product. Upon entering Calendar mode from Default mode, the display will show either the user's monthly calendar, or a default screen instructing them how to add a calendar file.
    - Pressing either the left and right buttons will navigate the calendar display between monthly, weekly, and daily view mode
    - Pressing the down button will return the screen to Default mode

Note that future iterations of this product will feature a more comprehensive menu system that more fully takes advantage of the six available navigation buttons on the frame.

Secondary Users

Create An Account

Secondary users create an account in the same way as primary users (see previous section). The only difference is that secondary users should select "Manager" under the "User Type" field in the registration form. Note that in this iteration there are no extra checks--any user can register as a manager. Future iterations will resolve this issue by requiring managers to register their first frame at the same time they make their account, requiring them to have already purchased a frame to be able to register as a manager.

Register Frames

This section details how to register frames under your management account.

1. Go to www.example-url.com
    - If you are logged in, you will be directed to your home page
    - If you are not logged in, you will be directed to the login page
        i. Log in to your account. See "Primary Users-Access Your Account" for information on how to log in.
        ii. You will be directed to your home page
2. Click on the "Manage" tab (this tab will only be visible if you have successfully registered as a manager account)
    - You will be directed to the "Manage" page

3. Click on the “Add Frames” button
    - You will be directed to the “Add Frames” page
4. Enter the ID of the frame you wish to add
    - The frame will be added to the database with your email listed as the managing email

Note that in this iteration there are no additional checks when you attempt to add a frame. Future versions will maintain a database of produced frames--when a new frame is registered, the server will compare its ID to see if it is in the list of produced frames. If it is, the frame can be registered to the manager. If it is not, an error message is instead produced telling the manager that their frame ID is invalid. Additionally, future versions should require some second factor of authentication when registering a frame beyond the ID number, to prevent malicious managers from attempting to register frames they do not actually own.

Set Up Frame

This section details how to set up the physical Personalizable Display Frame itself.

1. Plug the Personalizable Display Frame into an appropriate wall outlet near your desired location.
    a. The screen should load up immediately
2. The frame should automatically navigate it’s browser to www.example-url.com/slideshow.php?id=xxxxxxxxxx and display the appropriate information--setup is complete!

Link Users

This section details how to tell your frames what data they should display.

1. Go to www.example-url.com
    - If you are logged in you will be directed to your home page
    - If you are not logged in you will be directed to the login page
        i. Log in to access your account. See section “Primary Users-Access Your Account” for more information.
        ii. You will be directed to your home page
2. Click on the “Manage” tab
    - You will be directed to the “Manage” page
3. Click the “Link Users” button
    - You will be directed to the “Link Users” page
4. Enter a user’s email and the frame you wish to link the user with and click “Submit”
    - If the server finds the frame in the database, it will add the user’s email to the table entry. The frame with that ID will now display that user’s pictures.

Note that this iteration does not do additional verification when you attempt to link users. Future versions will ensure that the users exists in the database before linking them. Additionally, future versions will preemptively search for all frames that list you as the managing email and display them in a list--you will then be able to click on a frame to manage it instead of entering its ID. Another future feature would be to display a list of available primary users by first and last name that you could click on to associate with a frame, instead of entering the email manually. A search function could also be implemented here to increase ease of use.

Appendix B: Amazon Web Services Setup Guide

This appendix contains a more detailed explanation on how to setup and deploy the web application on an Amazon Web Services (AWS) cloud server. Note that an official Amazon tutorial on how to setup a Free-Tier cloud server can be found at these links:

- https://aws.amazon.com/ec2/getting-started/
- http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.MySQL.html

This guide will be a summary of the relevant features from these tutorials.

Elastic Cloud Compute

1. Setup and log into an AWS account
2. Navigate to the Elastic Cloud Compute (EC2) dashboard and select “Launch Instance”
3. Configure your instance
    a. Choose the machine type- we chose a Linux machine
    b. Choose the instance type- we chose t2.micro, a Free-Tier level processor
    c. Choose the security group- use the same group for the EC2 instance and the Relational Database Service (RDS) instance
4. Launch the instance and create a permission key by selecting “Create a new key pair”
5. Connect using Putty on Windows
    a. Download Putty if not already installed
    b. Obtain the following information about your instance:
        i. Instance ID- found in the “Instance ID” column on the EC2 dashboard
        ii. Public DNS- “Public DNS (IPv4)” column on the EC2 dashboard
    c. Enable SSH traffic by clicking the security group from the EC2 dashboard and then clicking “Add Rule->SSH->Anywhere”
    d. Use Puttygen (installed with Putty) to convert the permission key file
    e. Start Putty
        i. Select “Session” from the “Category” panel
            1. In the “Host Name” box enter “ec2-user@public_dns_name
            2. “Connection Type” should be set to “SSH”
            3. “Port” should be 22
            4. Click “Connection->SSH->Auth” and browse for the converted permission key
            5. (Optional) Enter a name and save the session for easier future use
        ii. Click “Open” to start the SSH session--you should now have access to the server using the Linux command line

Relational Database Service

1. Setup and log into an AWS account
2. Navigate to the Relational Database Service (RDS) dashboard
3. Click “Launch DB Instance” and enter the following details:
    a. “Select Engine” - mySQL
    b. “License Model” - default (general-public-license)
    c. “DB Engine Version” - default
    d. “DB Instance Class” - choose a size that fits your needs
    e. The other settings can easily be changed to suit your needs without affecting the rest of this process
    f. Navigate to “Configure Advanced Settings” - most of these setting do not need to be changed, but be sure to set the same security group as your EC2 instance
4. Launch the Instance - you should now be able to use mySQL queries to access the database upon successfully establishing connection