Pixels for Humanity

Min Woo Kim (mk4ed), Helen Shi (hs2fw), Kyle Cheng (kwc9ap), Ryan Pope (rcp3by), Surya Ambardar (sva4qf)

I - Project Information

For our final project for CS 4750, we created Pixels for Humanity, a web application and database which facilitates collection of nonprofit donations for a wide range of worthy causes while encouraging artistic creativity. The web application portion of our project begins as an entirely white 20 pixel by 20 pixel canvas. Users can interact with the canvas by selecting a pixel of their choice and changing its color to a different one, from a list of sixteen hues we provide. The coloring of a pixel represents a donation to a charity - donors can pick a dollar amount to contribute, and for each dollar, they can select a pixel and change its color from white to one of sixteen hues. Each pixel's color change represents a donation to a nonprofit organization of the users' choice. Users' color changes to the canvas are saved when the donation is processed through our own mock payment system. This project models a world where donations are not just fulfilling, but also can be a way to communicate an image or message. By donating and collaborating with friends, users can turn their donations into a picture or spell out a message on the blank canvas. Pixels for Humanity will accept donations in any dollar amount, and donors can allocate varying percentages of their donations to different nonprofits from our approved list. We were inspired by Alex Tew's wildly successful 2005 project titled "Million Dollar Homepage," where Tew sold rights to pixels on a blank canvas to fund his education. By revamping this idea into an interactive and engaging spin on nonprofit donation, we hope to raise money and bring awareness to a wide swath of worthy causes from world hunger to environmental protections, both locally in Charlottesville and on a national scale. Representing a diverse variety of causes was important to us because we wanted to highlight charities which have not received significant media attention, shining a spotlight on the many important causes which have not gone viral. The system will use databases to record donors, donation amounts, available colors, the charities they are linked with, and store a map of each pixel's current status. Any user on the internet can use this application to donate to causes they find important, while also creating messages and pictures on the canvas which express support for their charities.

The requirements for this project all revolve around ensuring its ability to solicit donations while allowing artistic expression. The primary requirements for the web application are:

- Allow new users to register with a username, password, and email address
- Allow previous users to log into the application using their username and password
- Display a canvas of pixels to users who are logged in
- Allow donors to select a particular pixel or pixels they wish to change the color of
- Allow donors to select a color of their choice from a dropdown menu of 16 colors

- Allow donors to select a charity of their choice from a dropdown menu of charities
- Allow donors to make multiple donations for different pixels in one checkout
- Display a running total of purchase info for users
- Provide a internal mock payment system for donors to submit their donations
- Allow donors to submit feedback on the web application so we know how to improve it in the future
- Allow users to view relevant pixel information by hovering over them
- Allow users to filter their purchases dynamically based on charity name
- Allow donors to view and manage their past feedback submissions
- Allow users to view a profile page with their account information
- Allow users to be marked as charity admins, giving them the ability to maintain and export basic information about their donors (email addresses, pixels donated, etc.)
- Allow charity admins to export all user's emails and total donation amounts who donated to their charity in CSV format
- Ensure that queries which require dynamic input are implemented via prepared statements (no inline string concatenation)
- Allow users to be given special admin privileges via user roles in the database, allowing them to view and delete users' feedback as well as approve new charity requests
    - There are three user roles in the database:
        - hs2fw_a -> solely for admins of charities to manage and view donators'
        - hs2fw_b -> for users to donate to the system
        - hs2fw_c -> solely for login/registration
- Allow users to log out of the application

The primary requirements for the database are:

- Maintain a record of users' usernames, passwords, and email addresses for web application login
- Maintain a record of users who can act as charity administrators
- Maintain a record of pixels' x and y locations in the canvas
- Maintain a record of pixels' current colors
- Maintain a record of pixels' most recent purchasers
- Maintain a list of colors donors can select from
- Maintain a list of charities donors can select from
- Maintain a record of donations which have been made
- Maintain a record of donors who have contributed
- Maintain a record of user-submitted feedbacks
- Ensure limited privileges for users in queries from the application via userrole enforcement

Other desired features of our system are:

- Aesthetically pleasing look and feel throughout the web application
- Encouraging multiple donors to collaborate and coordinate to make larger messages and images on the canvas
- Encourage donations to underfunded, local, or lesser-known charities

Some examples of how Pixels for Humanity is intended to be used are:

- After learning about a local child with cancer, a group of students decide to donate to St. Jude's research hospital. They collaborate together to color the central section of the canvas light purple, signifying their support for cancer patients, and raising both awareness and funds.
- After a deadly natural disaster in India, an individual decides to donate money to the Red Cross to help with disaster relief. They color 12 squares of the canvas with orange, white and green rows to depict the Indian flag, encouraging other users who log on to the web application to contribute to the cause as well.

II - Design Process

When designing Pixels for Humanity, our design decisions focused on the languages we had prior experience with, or those best suited to accomplish our goal. For the front end design of the web application, we mainly relied on HTML, CSS, and JavaScript, because these languages are intended for web design and make building custom applications straightforward. This is a very common design decision; the vast majority of websites today employ these three languages as their building blocks. Like nearly all websites, we used HTML for structuring our web application, CSS to design and style it, and JavaScript to build front end controls. We additionally used Ajax to start the project because we were able to carry over some code from the previous assignments in the class.

A huge part of our project was deciding how to handle donations from consumers. We initially integrated a Paypal framework because of its built in sandbox testing environment. Since the sandbox environment has the same API interface as the live environment, we were able to simulate what using Paypal would look like for the consumer. As our site evolved, however, we wanted more control over the actions taken upon pixel purchase. We also decided that for our core concept, it would be easier to use our own mock payment system.

This database is intended to appeal to members of the general public who are looking to donate to a wide range of causes. Because the user experience is integral to the trust of those who use our interface, there are a couple of tenets we based our project on:

- Accountability and Protection: Recent political activities concerning big tech have been highlighting the issues in tech platform power dynamics. Accountability means impeding user exploitation, curtailing surveillance, and prioritizing consumer safety. We addressed this by implementing data protection (password hashing), enforcing user privileges, and administering user control on phpMyAdmin.

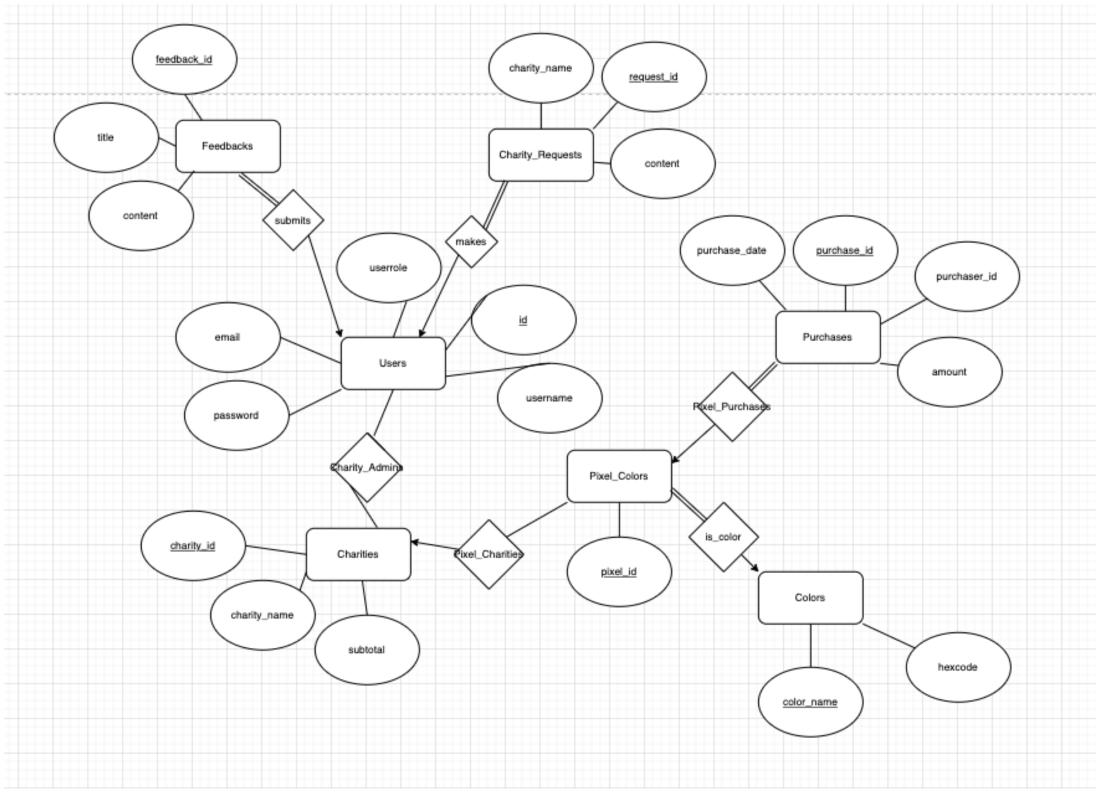
- Ongoing Testing: Since our product is geared towards a specific population, we must continuously change the product until a pleasurable user experience is established. Testing included SQL injection possibilities, as well as numerous different environments and users trying out the system. We also incorporated a system for website feedback so we can see what aspects of the site users like and dislike, guiding our future changes and patches to the application moving forward.

Database Schema

Charities(charity_id, charity_name, subtotal)
Charity_Admins(charity_id, admin_id)
Charity_Requests(request_id, user_id, charity_id, content)
Colors(color_name, hexcode)
Feedbacks(feedback_id, user_id, title, content)
Pixel_Charities(pixel_id, charity_id)
Pixel_Colors(pixel_id, color)
Pixel_Purchases(purchase_id, pixel_id)
Purchases(purchase_id, amount, purchaser_id, purchase_date)
Users(id, username, password, email, userrole)

More detailed information on the database schema, in addition to the triggers, constraints, and stored procedures we incorporated, can be found in hs2fw_FINALPROJECT.sql, which is attached with the rest of the project materials.

ER Diagram



Proof of 3NF for each table

To eliminate redundancies and ensure that there were no data conflicts in our tables, we normalized each relation in 3NF form. Below are the proofs of 3NF normalization for each table.

1. Charities(charity_id, charity_name, subtotal)

FD Set: {charity_id -> charity_name, subtotal}
Candidate key: charity_id
Check the FDs:

- charity_id -> charity_name: charity_id is a candidate key
- charity_id -> subtotal: charity_id is a candidate key

For table 1, no attributes on the right hand side of functional dependencies are transitively dependent on a candidate key, charity_id. Thus, this table is normalized in 3NF.

2. Charity_Admins(charity_id, admin_id)

For table 2, there are only two columns, so the attribute on the right hand side of the sole functional dependency cannot be transitively dependent on the candidate key, charity_id. Thus, this table is in 3NF.

3. Charity_Requests(request_id, user_id, charity_id, content)

FD Set: {request_id -> user_id, charity_id, content}
Candidate key: request_id

Check the FDs:

- request_id -> user_id: request_id is a candidate key
- request_id -> charity_id: request_id is a candidate key
- request_id -> content: request_id is a candidate key

For table 3, no attributes on the right hand side of functional dependencies are transitively dependent on the candidate key, request_id. Thus, this table is normalized in 3NF.

4. Colors(color_name, hexcode)

For table 4, there are only two columns, so the attribute on the right hand side of the sole functional dependency cannot be transitively dependent on the candidate key, color_name. Thus, this table is normalized in 3NF.

5. Feedbacks(feedback_id, user_id, title, content)

FD Set: {feedback_id -> username, title, content}
Candidate key: feedback_id
Check the FDs:

- feedback_id -> user_id: feedback_id is a candidate key
- feedback_id -> title: feedback_id is a candidate key
- feedback_id -> content: feedback_id is a candidate key

For table 5, no attributes on the right hand side of functional dependencies are transitively dependent on the candidate key, feedback_id. Thus, this table is normalized in 3NF.

6. Pixel_Charities(pixel_id, charity_id)

For table 6, there are only two columns, so the attribute on the right hand side of the sole functional dependency cannot be transitively dependent on the candidate key, pixel_id. Thus, this table is normalized in 3NF.

7. Pixel_Colors(pixel_id, color)

For table 7, there are only two columns, so the attribute on the right hand side of the sole functional dependency cannot be transitively dependent on the candidate key, pixel_id. Thus, this table is normalized in 3NF.

8. Pixel_Purchases(purchase_id, pixel_id)

For table 8, there are only two columns, so the attribute on the right hand side of the sole functional dependency cannot be transitively dependent on the candidate key, purchase_id. Thus, this table is normalized in 3NF.

9. Purchases(purchase_id, amount, purchaser_id, purchase_date)

FD Set: {purchase_id -> amount, purchaser_id, purchase_date}
Candidate key: purchase_id

Check the FDs:

- purchase_id -> amount: purchase_id is a candidate key
- purchase_id -> purchaser_id: purchase_id is a candidate key
- purchase_id -> purchase_date: purchase_id is a candidate key

For table 9, no attributes on the right hand side of functional dependencies are transitively dependent on the candidate key, purchase_id. Thus, this table is normalized in 3NF.

10. Users(<u>id</u>, username, password, email, userrole)

FD Set: {id -> username, password, email, userrole}
Candidate key: username
Check the FDs:

- id -> username: id is a candidate key
- id -> password: id is a candidate key
- id -> email: id is a candidate key
- id -> userrole: id is a candidate key

For table 10, no attributes on the right hand side of functional dependencies are transitively dependent on the candidate key, username. Thus, this table is normalized in 3NF.

Additional SQL Features

- Unique key constraints on Pixel_Purchases and Pixel_Charities ensure that there cannot be multiple entries for the same pixel
- Trigger ensures that there are no invalid pixel numbers in pixel_charities by raising an error for queries with an invalid pixel number (outside the range 0 to 399)
- Trigger ensures that there are no invalid pixel numbers in pixel_purchases by raising an error for queries with an invalid pixel number (outside the range 0 to 399)
- Created stored procedure RetrievePixelInfo() to allow a simple way to retrieve all pixel info, returning pixel id, color, charity_name, username, and purchase date
- More detailed information about the additional and advanced SQL features we incorporated can be found in the document hs2fw_FINALPROJECT.sql, which is attached with the rest of the project materials.

III - Evaluation of Product

While developing our database and application, we tested regularly throughout to ensure that everything was working as intended. Our tests were largely based on our intended requirements, so that we could ensure that all required functions were working ws we intended them to. For the web application, our testing procedures were:

- Testing that new users can register with new username, password, and email address by registering to the web application from multiple new accounts

- Testing that previous users can log into the application using their username and password by logging into the web application with existing credentials
- Testing that the web application displays a canvas of pixels to users who are logged in by logging into the application and checking the main page
- Testing that the canvas is initially entirely white by logging into the application before any changes are made
- Testing that users can select a particular pixel or pixels they wish to change the color of by navigating around the web application and clicking each pixel
- Testing that users can select a color of their choice from a dropdown menu of 16 colors by navigating around the dropdown menu and selecting each color
- Testing that users can select a charity of their choice from a dropdown menu of charities by navigating around the dropdown menu and selecting each charity
- Testing that users can make multiple donations for different pixels in one checkout by observing the running total of purchase info for users as we navigate through the page and select multiple pixels
- Testing the mock donation system, allowing users to purchase selected pixels, updating the pixel's representation in the database
- Testing that users can submit feedback on the web application by navigating to the submission form and submitting test feedbacks
- Testing that users can view and manage their past feedback submissions by navigating to the past feedback option and confirming that we can view old feedbacks
- Testing that users can view a profile page with their account information by navigating to the profile page button and confirming we can view profile information
- Testing that we can export of all user's emails who donated to their charity in CSV format and that exported emails and pixel counts are accurate
- Testing that users can view relevant pixel information by hovering over them and that the retrieved information is accurate
- Testing that users can filter their purchases dynamically based on charity name and that the sorting is accurate
- Testing that queries which require dynamic input are implemented via prepared statements (no inline string concatenation)
- Testing that users can be marked as charity admins, giving them the ability to maintain and export basic information about their donors (email addresses, pixels donated, etc.)
- Testing that users can be given special admin privileges via user roles in the database, allowing them to view and delete users' feedback as well as approve new charity requests
- Testing that users can log out of the application by navigating to the logout button and confirming it takes users back to the login page

For the database, our testing procedures were:

- Testing that the database in phpMyAdmin was maintaining a record of users' usernames, passwords, and email addresses for web application login by checking the users table before and and after registering a new user
- Testing that the database in phpMyAdmin was maintaining a record of pixels' current colors by checking the Pixel_colors table before and and after new donations and pixel color changes were made
- Testing that the database in phpMyAdmin was maintaining a record of pixels' most recent purchasers by checking the Pixel_Purchases table before and and after a new donation was made
- Testing that the database in phpMyAdmin was maintaining a list of colors donors can select from by checking the Colors table
- Testing that the database in phpMyAdmin was maintaining a list of charities donors can select from by checking the Charities table
- Testing that the database in phpMyAdmin was maintaining a record of donations and donors which have been made by checking the purchases table
- Testing that the database in phpMyAdmin was maintaining a record of user submitted feedbacks by checking the Feedbacks table

Sample Data from 'Charities' Table

```
VALUES - (charity_id, charity_name, subtotal)
(1, 'Salvation Army', 15),
(2, 'Meals on Wheels', 42),
(3, 'Rivanna Conservation Society', 12),
(4, 'NAACP', 37),
(5, 'The Innocence Project', 5),
...
(10, 'ASPCA', 40);
```

Sample Data from 'Colors' Table

```
VALUES (color_name, hexcode)
('black', '#1c1c1c'),
('brown', '#a26a3d'),
('cyan', '#00e6f2'),
('dark_green', '#00c100'),
('dark_grey', '#89888a'),
...
('white', '#ffffff');
```

Sample Queries

The most significant interaction between users and the database comes from buying a pixel and being able to update that pixel's representation in the database. To display the pixels when the web application is first loaded, the database is queried for an array containing a "pixel_id" that uniquely identifies each pixel, along with other information such as the color of the pixel.

```
SELECT pc.pixel_id, pc.color, c.hexcode, chars.charity_name,
username, purchase_date FROM `Pixel_Colors` pc
LEFT OUTER JOIN `Colors` c ON c.color_name = pc.color
LEFT OUTER JOIN `Pixel_Purchases` pp on pp.pixel_id =
pc.pixel_id
LEFT OUTER JOIN `Purchases` purch on purch.purchase_id =
pp.purchase_id
LEFT OUTER JOIN `Pixel_Charities` pchar on pchar.pixel_id =
pc.pixel_id
LEFT OUTER JOIN `Charities` chars on chars.charity_id =
pchar.charity_id
LEFT OUTER JOIN `Users` u on u.id = purch.purchaser_id ORDER BY
pixel_id ASC
```

Once a user registers for the website and is assigned a user_id, chooses a color and a charity, they can then purchase a pixel and its representation in the database will be updated accordingly. For example, if user "John Doe" with a user_id of 6, were to choose the color "sierra_orange" and the charity "Salvation Army (1)', and then buy pixel 376, the following queries would be ran:

```
1) UPDATE `Pixel_Colors` SET color= 'sienna_orange' WHERE
   pixel_id=376"
2) INSERT INTO 'Purchases' (amount, purchaser_id) VALUES (1,6)
3) INSERT INTO 'Pixel_Purchases' (purchase_id, pixel_id)
   VALUES (35, 376)
4) INSERT 'Pixel_Charities' (pixel_id, charity_id) VALUES
   (376, 1)
```