CEN 4010 SPRING 2023

Overview: A website that users can used to save images from the web and put tags on them so they're easily retrievable

Milestone 4 Beta Launch and Reviews

Team 3

Aira Torres – Scrum Master, Backend Developer	airatorres2020@fau.edu
Daniel Rovira- Frontend Developer	drovira2020@fau.edu
George Gordon – Backend Developer	ggordon2019@fau.edu
Tyler Fasoli- Frontend Developer	tfasoli2020@fau.edu
Eric Smith – Repository Maintainer, Backend Developer, Product Owner	ericsmith2019@fau.edu

April 14, 2023

Revision History:

Product Summary:

Product name: Overview

With Overview, we want our users to be satisfied with saving, organizing, and viewing their photographs in one location. Access to searchable images that can be labeled will be helpful for people who love saving memories, taking funny pictures, or who need to keep images organized. With this product, our users can expect to utilize three main functions to store their photographs within our database safely. After entering our site, users must expect to see our “save new image” button, where they can paste their image URLs into the pop-up bar section to submit their image URLs. With this, they may add tags that commas can separate from differentiating the desired tags and then can submit their photo. Now the newly submitted photo will be on display, allowing users to select them or search for the image using our search bar feature. The search bar can locate the photos by URL or by the tag in which they would appear. These tags are also on display and can be selected anytime using a slider feature on the left side of the webpage. Users can also modify images by deleting them or logging out after they are done. Ultimately, with this process, users can utilize Overview to store, organize, and new their precious images all in one convenient area.

Usability Test Plan:

Test Objectives:

The staff at Overview perform user research to gain a better understanding of our product along with the users themselves. For this milestone, our main focus was placed on the backend development of creating an efficient and easy to use interface for creating an Overview account and producing an interface that allows users to save their image and create tags for their specific image to be stored into our application. For the usability test plan, our main testing objective focused on two specific things:

1. Whether our application was able to functionally create/link an account onto our program
2. If our application was able to save and tag an image if a proper url was given by the user, and if an error message was displayed when an improper image url was provided.

With the focus on the overall functionality of our intended purpose for our application, we specifically looked for instances where the program was properly executed with a proper image url and a specified tag for the image, while also looking to see if our image was able to be properly and securely stored in our photo bank. We also intended to find examples where our program had either a time where it was not properly executed or a time where there was an image url that did not meet our requirements in order to display our intended error message for the user.

<u>Test plan:</u>

For our test plan, we as a team decided that the main focus of our test should be on our error messages, our direct implementation of the photo provided by the user into our photo bank, and to investigate different scenarios in which our program may not work to our fullest intent, whether that be not implementing the image, not reading the user provided image url correctly, or not displaying the error message when an incorrect image url was provided by the user.

Starting point:

Our starting point of this test plan indicated that we would test the entire process of actually saving and tagging an image to test Overview's functionality aspect. This means that we went through the process of either creating an account or logging into a previously existing account, to then viewing our photo storage of this account, to then actually implementing an image by means of a url provided by the user and a tag set by the user as well. Finally, we would also test to see if our tag of the image was properly stored within the photo bank inside of Overview by using our "find images with a given tag" search bar within Overview.

Task to be accomplished:

There are a few intended tasks to be accomplished during this testing plan. We desire to have a functional and easy process to log onto the account that will provide us access to viewing the images stored there. We also are testing for the functionality of our add new image feature, in which we test to see if the image url given by the user is a valid url (displaying the valid image url message to the user) or if the image url is invalid (displaying the invalid image url message to the user). Lastly, after saving a valid image url with specific tags given by the user, we are testing to see if Overview has stored these images correctly and is able to bring up these images properly when the user searches for the specific tag. The intended user of these tests will be carried out from the user's perspective, and not Team 3's perspective. We are looking to experience our app from a consumer's viewpoint and not a developer's viewpoint here.

Completion criteria:

-If this test runs correctly, Overview will allow for the user to log in to our application with either an existing account or to create a new account.

-Upon logging in with their respective account, the next test would be to click on the save new image button and attempt to save a new image via url. If this test succeeds, after entering a proper image url, a green box will be displayed letting the user know that this is a valid image url to be saved into our application, and the specified tags box is displayed for the user to enter the tags. If the image url provided by the user is not a valid image url, the test would be considered complete if the red box is displayed with the error message of an invalid image url provided until a valid image url is provided.

-After saving the image via the url, the last test is to see if the app has successfully stored and documented these images under the user's account. For this, we are searching up the image tag that was previously provided by the user when saving the image, and checking to see if the desired image is brought up in our database. This test would be considered complete if the test image was displayed after searching up the proper image tag of the image, and is swiftly and clearly displayed upon search.

URL of the system to be tested:

<u>Questionnaire form:</u>

1. The system has reacted well to the user logging into their database and has allowed the user access to their secure database for imminent image saving and browsing

-Very unsatisfied -Unsatisfied -Neutral -Satisfied -Very satisfied

2. After accessing their database of photos, the user is able to save their desired image from a proper image url and has the option to tag their images

-Very unsatisfied -Unsatisfied -Neutral -Satisfied -Very satisfied

3. After saving the image and tagging the image, the user is able to search up the specified image tag and the image will appear for them

-Very unsatisfied -Unsatisfied -Neutral -Satisfied -Very satisfied

QA Test Plan:

Test Objectives:

Specifically for this test, we will be focusing a little bit more directly on the usage, functionality, and effectiveness of our save and tag image function. This is a little bit more precise than our test plan test objectives, because we are actually implementing test cases on our specific feature of saving and tagging images. In these test cases that are created from the user's perspective, we will be testing to see if our system is able to efficiently save an image, allow the user to tag this specific image with as many tags as they wish, and then later access these images through the image tags that were previously provided by the user. In these tests, we are looking for instances of effective image url placement and saving, instances of ineffective image saving regardless of the reason (invalid image url, incorrect placement of the image, image not being saved, etc.), and instances of searching up the tags and a proper display of the image.

Hardware and software setup:

Breaking down the overall testing environment of this feature testing environment, there is a mix of hardware and software setup required for this test to be operational. First, in terms of hardware, we must take into account that it is necessary to have a computer and a keyboard in order to actually access our program and interact with it as well. Our specific hardware operating systems that we support include Windows, OS X, and Linux. For the software aspect, this is the more relevant aspect of our feature, where the user must be able to access and interact with our program through our specified html link. This is done through our javascript and html coding on the backend of our team. The feature we are testing, image saving, is made possible through our backend coding product that allows the user through software aspects to enter an image url and then save this image url if it is a valid url. Overview itself is a software aspect, but our features run deeper with their functionality through our coding products. In this case, we are not necessarily testing the hardware aspect of our testing environment. We are testing our software capabilities through the hardware incorporations we have added to see if our features are fully functional or not. As stated previously in our non-functional requirements list, the type of software that our program is capable to be brought upon in terms of browsers is Google Chrome and Internet Explorer for the specific case that JavaScript is not installed on the browser.

Feature to be tested:

There are two features that fall under the same category and are necessary for dual-testing in this instance, which are the photo image saving feature (which includes the photo tagging aspect) and the proper image tagging feature where the user is able to search up the desired tag of the image in order to access the image and see if the image has saved properly. Both of these are necessary to be tested together due to the fact that after saving an image, to see if the image has been saved properly we must check to see if the specified tags will bring up the specific image. In this case, we are solely testing one image at a time under a specific tag in order to maximize our testing opportunities and usability scale. If our program does not work properly with only one image under a specific tag, then there is no point in checking for multiple images under a single tag until a later date.

Actual Test Cases:

Test #	Test Title	Test Description	Test input	Expected Correct Output	Test Results per browser (Internet Explorer & Google Chrome)
#1	Image of Tiger	For the first test case, we tested this image of a tiger under one of our test accounts to see if the image would be properly saved and I provided the tag of "tiger" to see if the image had been stored properly	https://images.unsplash.com/photo-1533450718592-29d45635f0a9?ixlib=rb-4.0.3&ixid=MnwxMjA3fDB8MHxzZWFyY2h8MXx8anBnfGVufDB8fDB8fA%3D%3D&auto=format&fit=crop&w=900&q=60	Image was saved and properly displayed when "tiger" was searched in the tag search bar	Internet Explorer: PASS Google Chrome: PASS
#2	Image of Turtle	For this test case, we tested this image of a turtle under one of our test accounts to see if the image would be properly saved and I provided the tag of "turtle" to see if the image had been stored	https://images.unsplash.com/photo-1547446692-18fa77522764?ixlib=rb-4.0.3&ixid=MnwxMjA3fDB8MHxjb2xsZWN0aW9uLXBhZ2V8MTZ8NTAzNTU0Nnx8ZW58MHx8fHw%3D&auto=format&fit=	Image was saved and properly displayed when "turtle" was searched in the tag search bar	Internet Explorer: PASS Google Chrome: PASS

		properly	crop&w=900&q=60		
#3	Image of golf course	For this test case, we tested this image of a tiger under one of our test accounts to see if the image would be properly saved and I provided the tag of "golf" to see if the image had been stored properly	https://images.unsplash.com/photo-1443706340763-4b60757a36ce?ixlib=rb-4.0.3&ixid=MnwxMjA3fDB8MHxzZWFyY2h8MTh8fGdvbGZ8ZW58MHx8MHx8&auto=format&fit=crop&w=900&q=60	Image was saved and properly displayed when "turtle" was searched in the tag search bar	Internet Explorer: PASS Google Chrome: PASS

Code Review:

CODING STYLE:

For our beta, we've implemented snake_case for all the functions and variables throughout and also used an imperative programming approach. Imperative programming specifies the required steps necessary to accomplish a goal within the code.

EXAMPLE CODE:

Some of the important code regarding functionality of our program are as follows,

```
function set_thumbnail_callbacks_and_populate_tag_list(tags_arr)
{
    var thumbnails = $(".card-img-top");
    thumbnails.each(function(index)
    {
        let thumbnail_manage_button = $("#thumb-manage-button-id-" + index.toString());
        let thumbnail = $("#thumb-img-id-" + index.toString());
        let thumbnail_image_source_url = thumbnail.attr("src");

        thumbnail.off().on("click", function(event)
        {
            console.log(index);
            let taglist = $("#the-tag-list");
            taglist.html("");
            for(let i = 0; i < tags_arr[index].length; i++)
            {
                taglist.append(
                  `
                      <li class="list-group-item">${tags_arr[index][i]}</li>
                  `
                );
            }
        });
```

The code above is used in our program to display the different tags available when searching for your desired image. Gives you a way to easily search and sort images based on a tag such as "Cool" or "Space" etc.

```
function show_thumbnails(images_arr, tags_arr)
{
    var grid = $("#thumbnail-grid");
    grid.html("");
    for (var i = 0; i < images_arr.length; i++)
    {
        var grid_id = i.toString();
        grid.append(`
          <div class="col-lg-3 col-md-4 col-sm-6 mb-4">
            <div class="card">
              <img src="${images_arr[i]}" class="card-img-top" id="thumb-img-id-${grid_id}" alt="...">
              <div class="card-body">
                <button type="button" class="btn btn-outline-danger btn-sm manage-image-button" data-bs-toggle="modal" data-bs-target="#manage-image-modal" id="thumb-manage-button-id-${grid_id}">Delete</button>
              </div>
            </div>
          </div>`);
    }
    set_thumbnail_callbacks_and_populate_tag_list(tags_arr);
}
```

This set of code allows us to place all the images throughout our beta onto a grid rather than being placed randomly. Allows for a much cleaner look and easy accessibility when gathering and searching for different images.

Self-check on best practices for security:

- Major Assets
  The only major asset our site protects is the user's password. The images the user saves are open to public

- Encrypt Password
  In create-account.php we hash the user password in line 13.

- Confirm Input Data Validation
  The only input we validate is the image url and the validation is handled in input-validator.js

Self-check: Adherence to original Non-functional specs:

List of non-functional requirements

Performance Requirements :

1. Responsiveness: The system will also be responsive, operating on various monitor sizes, ranging from 10” netbooks to 24” desktop monitors. It will also be responsive with a wide variety of resolutions, from 1024 x 480 through 1920 x 1080. [DONE]

2. Test Requirements: the test requirements for performance will include an expected load test as well as testing on all of the functional specs listed and their speed per transaction. [DONE]

3. Number of instances: The project should be able to withhold at least 50 instances of the site at a time. And the speed of each instance will process 10-50 transactions per second within a 20-100 millisecond response time between each transaction. [DONE]

4. Bug Count: The project will have no more than 6 bugs during the Beta testing phase and no more than 3 bugs can remain within the system after the final release.[ON TRACK]

5. Execution Speed: Execution speed of the initial home page on a high-speed internet connection should load within 100-200 milliseconds, depending on the current cycle time. [DONE]

6. Emphasis on usability, should be easier to add images when compared to using a native social media platform for image saving such as Pinterest. [DONE]

Ease of Use:

1. Training Time: Training time should be as much as it takes for users to learn [ON TRACK]

2. Accessibility: user interface items should be distinct and easy to discover [ON TRACK]

Interoperability Requirements:

1. Browser Compatibility: The system will be a web-based web app that will operate on at least two major browsers focusing on Google Chrome and Internet Explorer provide alternatives if the browser does not have JavaScript installed on it. [DONE]

2. Computer and OS Compatibility: The system will operate on various types of operating systems, including Windows, OS X, and Linux. It will also operate on any type. [ON TRACK]
of computer which can run a browser that is supported. [DONE]

Fault Tolerance:

1. Exception Handling: There will be exception handling provided in all situations

where an exception could occur. This will provide the user with an explanation as to why an exception occurred and give them a chance to either input the correct answer or they will be taken back to the home page. [DONE]

Security Requirements:

1. Login/Password System: Our system will have a login/password system to maintain the preferences, ratings, and reviews of our site's visitors and locals. This implementation will also require password confirmation upon creation. We will also ask the user for a security question that we will store along with the answer and if the user forgets their password, they will be able to retrieve it by providing the answer. [DONE]
2. Encryption: The website will not be encrypted as there will be microtransactions or requirements for personal information, such as credit card numbers, will occur. [DONE]

3. Security requirements: no personal data associated with the user, or their account should be stored. All relevant data retrieved via an API must only be used for the operation that the user has initiated; there will be no storage of any user-specific data. [DONE]

Availability Requirements:

1. Times of Accessibility: the system will run 24 hours a day, 7 days a week unless for bug fixes or maintenance times. As long as the server is available the website will continue to run. [ON TRACK]

2. Maintenance: fixes or maintenance will be fixed as soon as possible and will be displayed in the event that the website is down. [DONE]