Design Specification Document

Project: PhotoChop

Developers:

Cameron Yen
Nick Graham
Tyler Woodfin
Simon Boerwinkle
Alex Salazar-Almaraz
Nathan Smith
Khaled Jabr
Jordan Nguyen
Tariq Broadnax

Document Versions

Version Number	Date	Authors
1.0	September 23, 2016	Nick Graham, Simon Boerwinkle
2.0	September 26, 2016	Nick Graham, Simon Boerwinkle, Tyler Woodfin, Alex Salazar-Almaraz, Khaled Jabr

Table of Contents

1. Overview

PhotoChop is an image processing application that finds the “most interesting area” of a photograph (referred to hereafter as MIA) and crops the remaining area. The app uses a high-level image detection framework called ImageProcessor to identify components of an image and an in-house algorithm to combine those components in interesting ways. The app will include a button that will take the user to a page with instructions on how to upload their images to Instagram.

2. User Stories

Jenny is a young, college-educated professional in her late twenties. She loves to travel and explore, whether it's weekend trips to scout out local flavor/festivals/art/nature, or saving up money and vacation time for longer trips around or out of the country. She considers herself an amateur photographer, despite her awful photography skills, and she wants her Instagram shots to look sophisticated and artistic. She's not just snapping selfies and food shots; she is trying to capture what catches her eye- she is trying to capture what makes travel important for her. However, she is often frustrated that her pictures don't do her experiences justice. Fortunately, Jenny has discovered PhotoChop. With just a few clicks, she is able to import any number of photos from her trip and quickly process them. Moreover, using the app’s Instagram Upload instructions, she is able to easily share them with her followers. PhotoChop allows Jenny to capture images much more quickly, now that she no longer has to manually crop out distractions.

Walter is a professional photographer who runs a small workshop on the side in order to inspire future photographers or journalists who want to enhance their articles with quality photos. Recently, a friend had told him about an app called PhotoChop that helps users edit their photos to remove image clutter. As a veteran photographer, he has very high standards for any editing tools used to process his photos, and he tends to minimize any use of automated image processing. Nonetheless, he decides to check out this new app and is greeted with a simple home screen asking the user to upload a photo. The option to use the rule of thirds for enhancing the photograph catches his eye. He decides to use some of the photographs taken by his students to see the results and was surprised to see that the resulting pictures align extremely well with this photography principle. Although most of the photographs weren't perfectly aligned, there is a definite improvement in the quality of the photos. Walter now introduces PhotoChop to his workshops to illustrate such a fundamental principle of photography. Even the older members of the workshop can use the app since the interface is simple enough for even a novice computer user.

3. Constraints

This application must:

Be written in C#

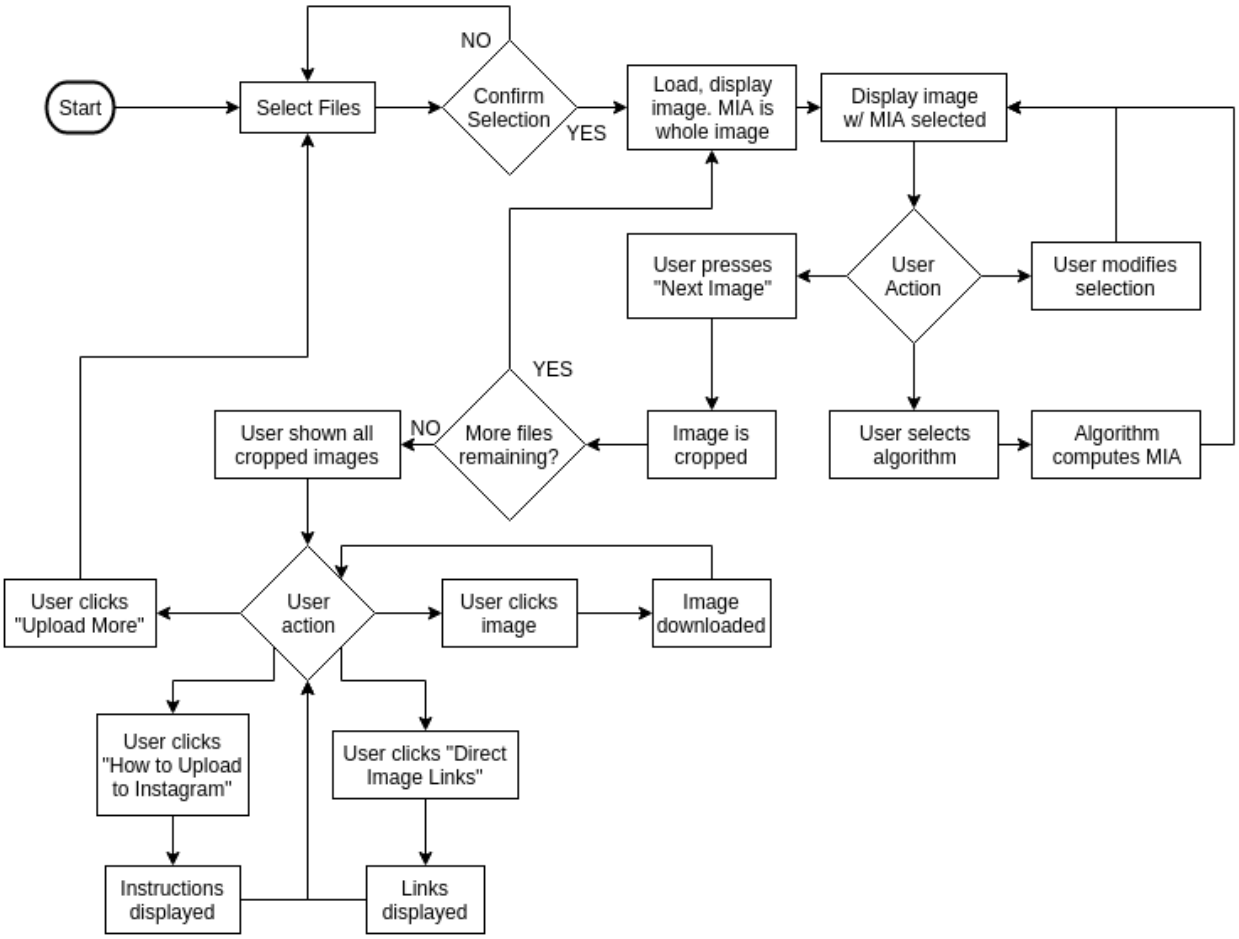
Operate at full functionality in the latest release versions of Google Chrome, Safari, and Firefox.

Non-functional requirements:

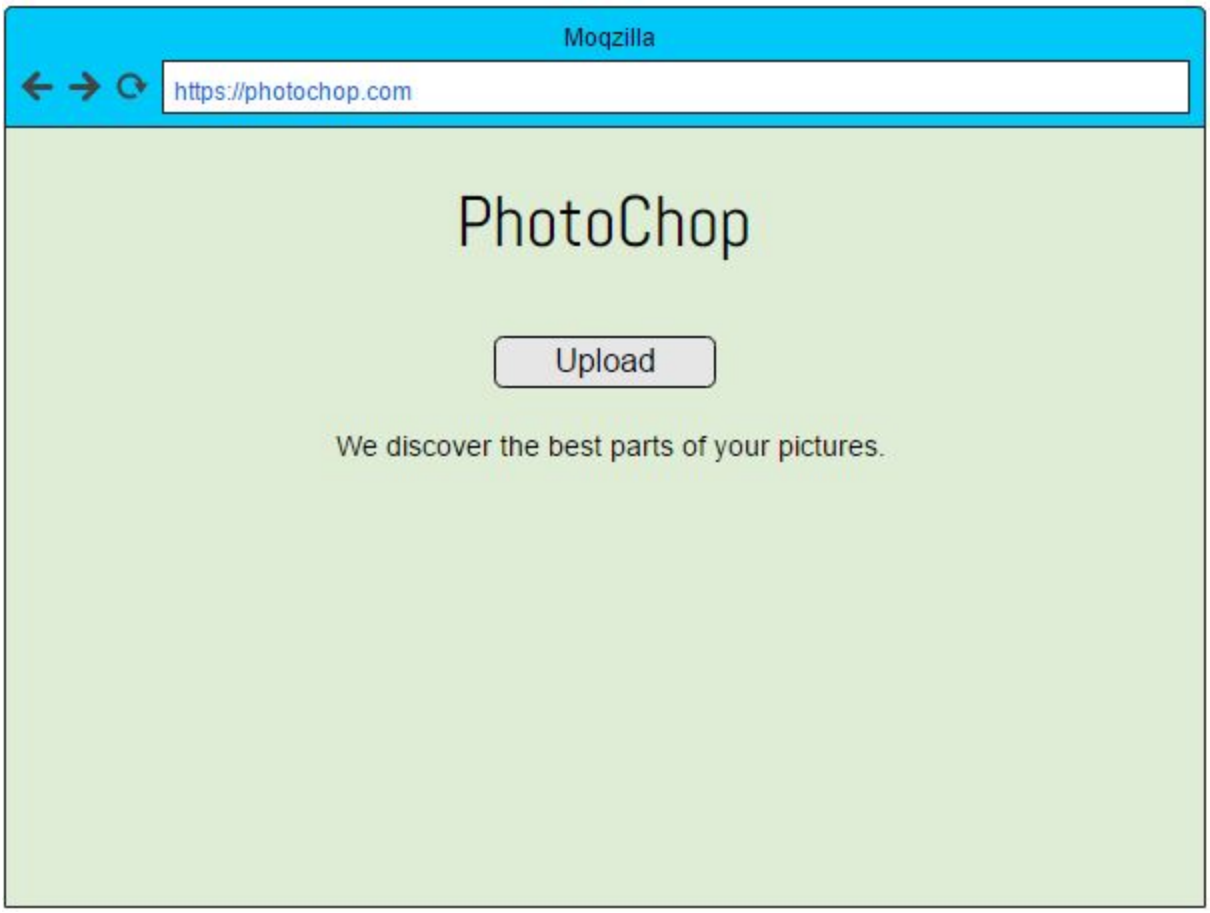
Only files with the extensions .jpg, .jpeg, .gif, and .png will be supported.

4. Flowcharts

The basic user flow through the app is visualized below:

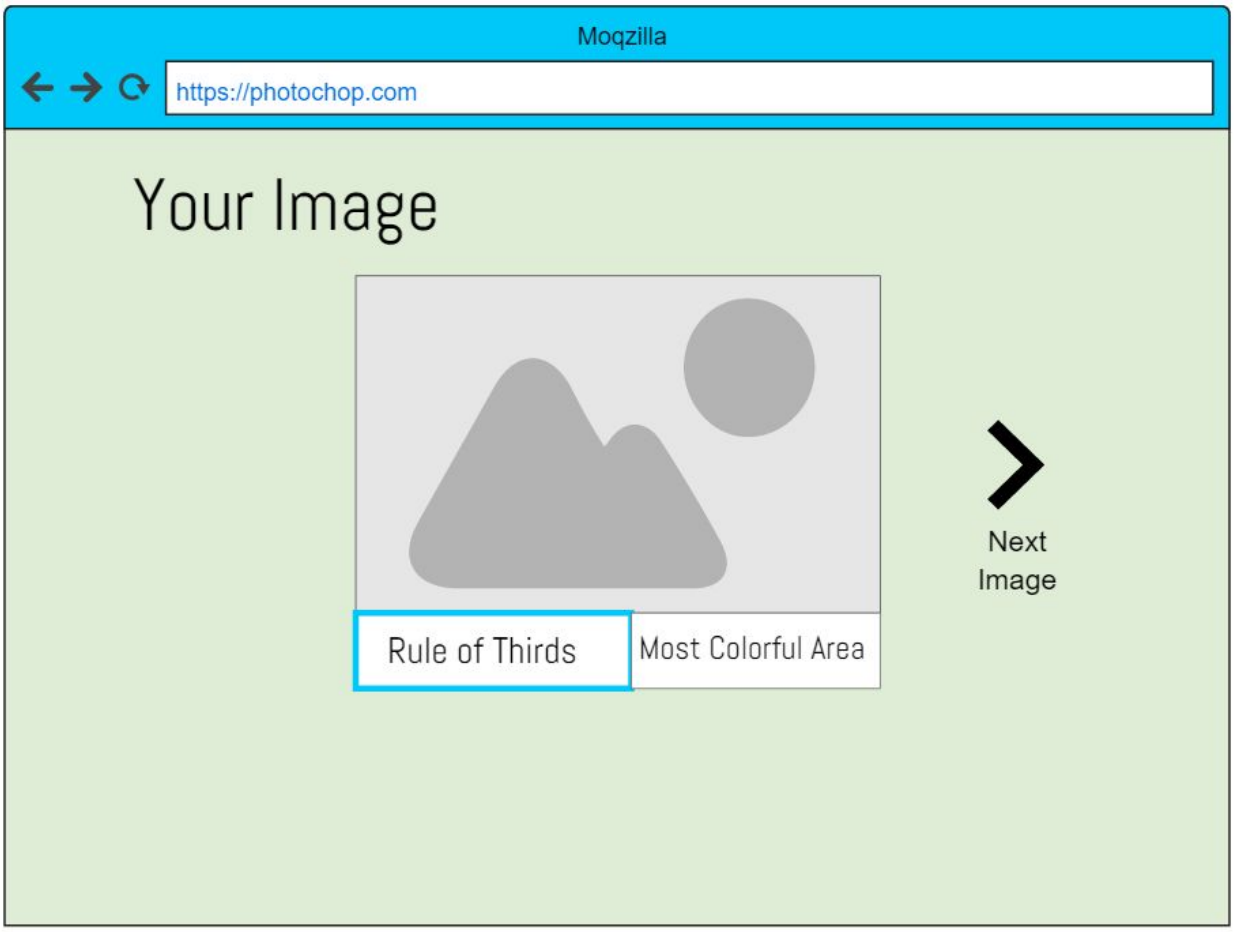


5. Screen by Screen Specification



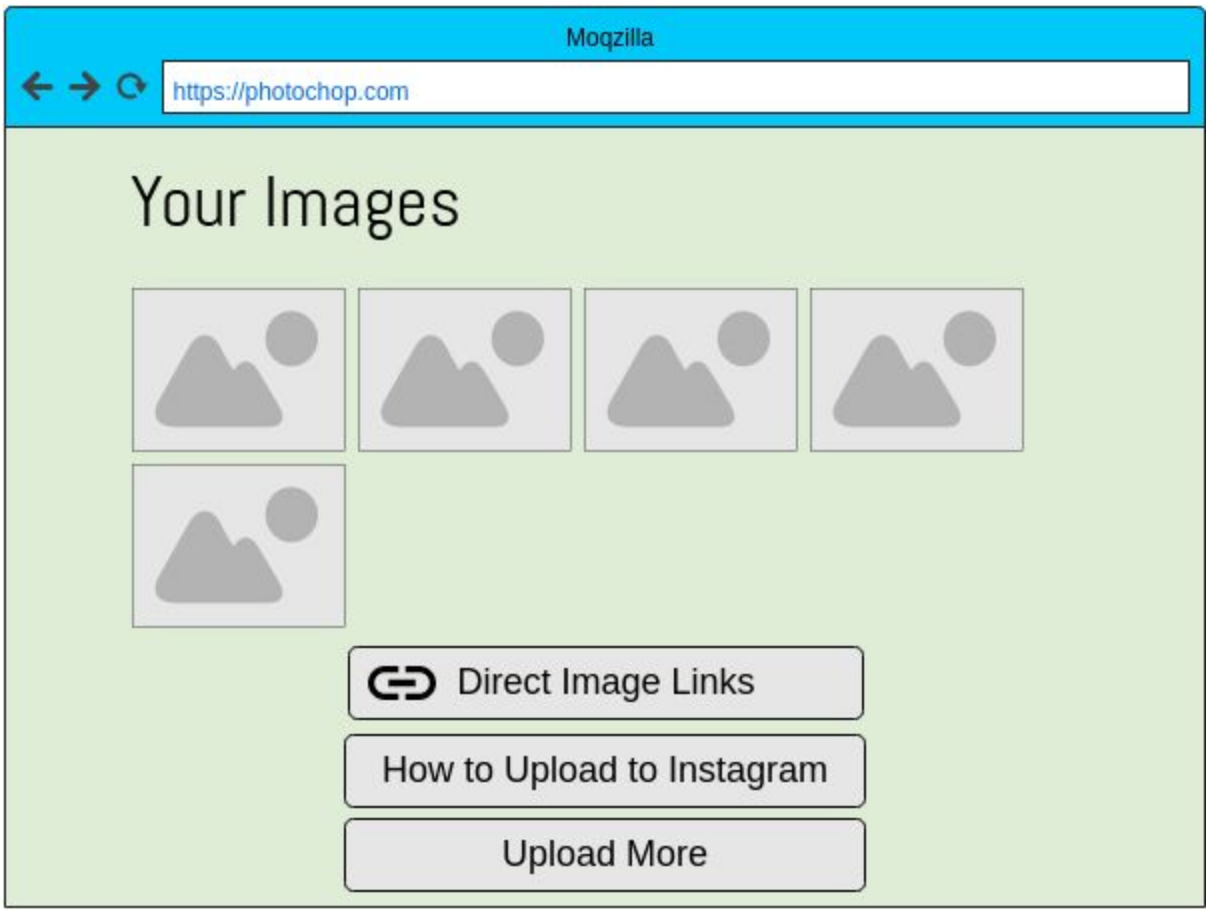
Home Screen:

The home screen is a simple landing page. It has one button labeled “Upload” that will open a file upload dialog box. It has the header dialog “PhotoChop” and the tag line “We discover the best parts of your pictures.” The upload dialog is where the images with the appropriate file extensions, i.e. “.png, .gif, .jpg, .jpeg”, will be selected by the user. The user will also be able to upload multiple images at once as listed in the requirements. Upon uploading the image/s the user will be taken to the post-upload screen.



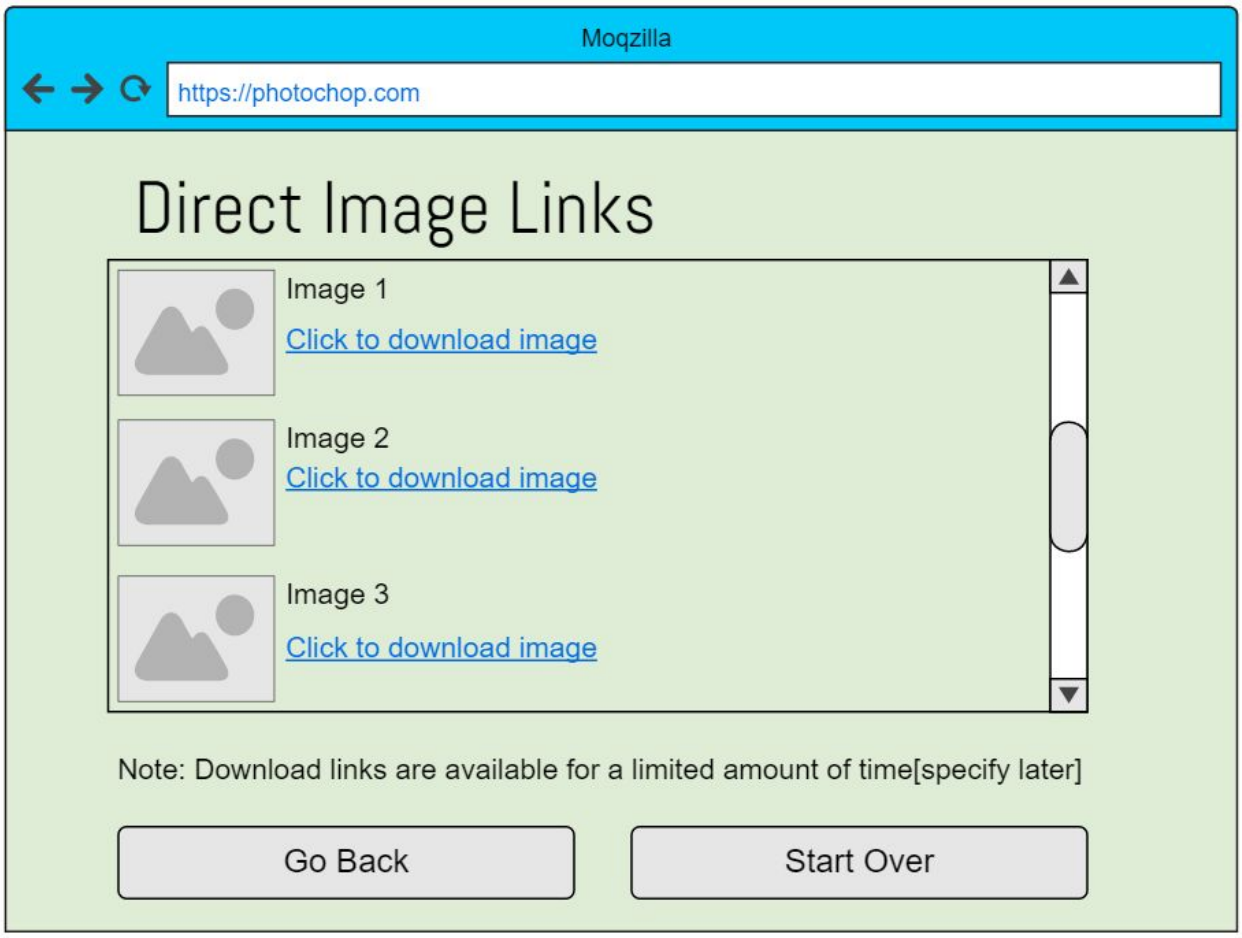
Post-Upload Screen:

This is the screen the user sees after selecting the images they want to upload. They will initially be displayed the first picture. The can select the algorithm to use, choosing between rule of thirds, and most colorful area. They can then modify the selection then modify the selected area to suit their specific needs. After they have selected their crop area they can click on the next image button to move to the next image. If there are no images remaining, they are taken to the download page.



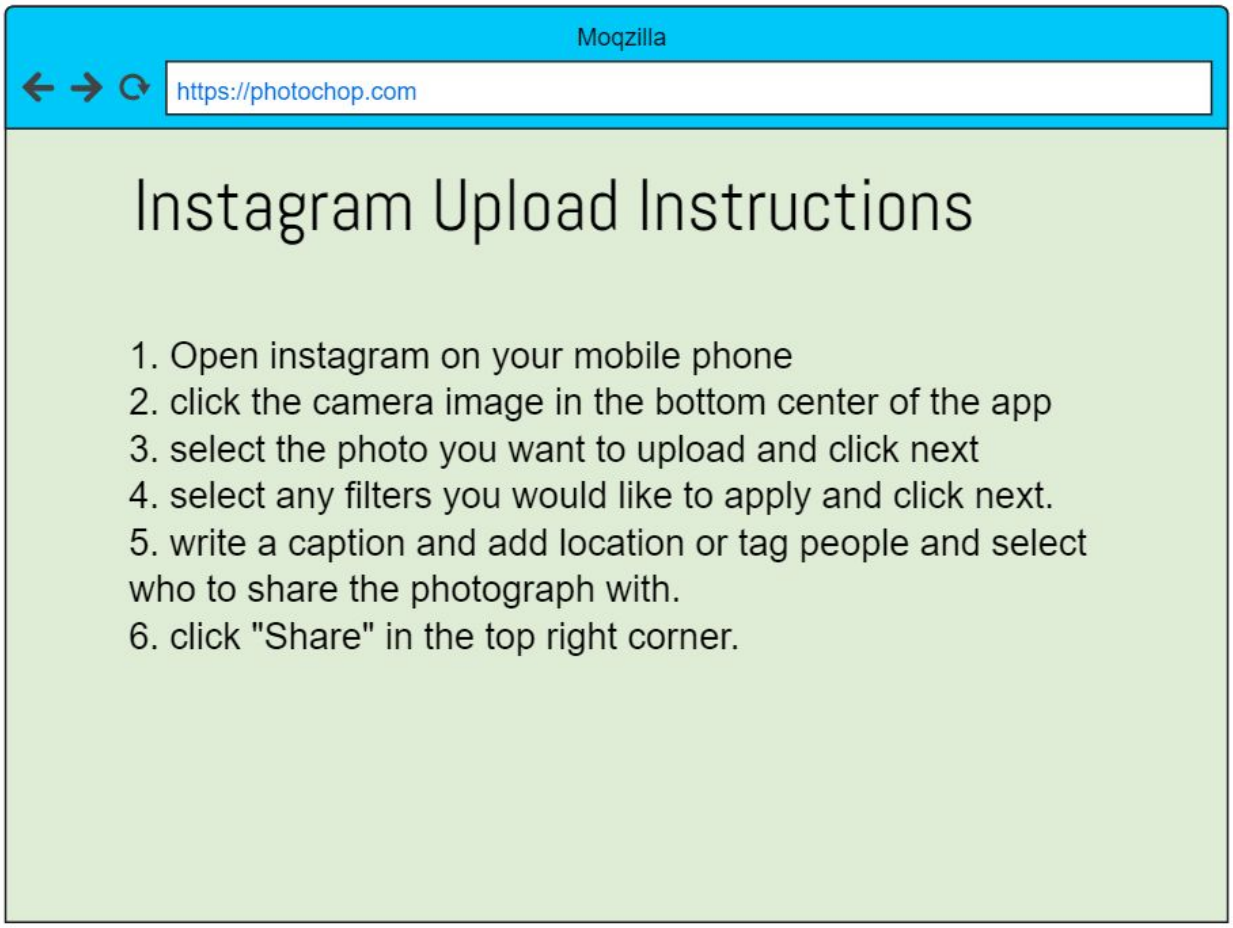
Download Screen:

The download screen shows all of the users uploaded images cropped to their liking. They are presented three buttons. The direct image links button will present them with download links for the images. The How to upload to instagram button takes them to a page that lists instructions on how to upload their cropped images to instagram. The upload more button will take them back to the home screen.



Direct Image Links Screen:

This is the screen user are directed to after clicking the “Direct Image links” button from the Download Screen. This allows users to download any number of cropped images the user has made through PhotoChop. The links will only be active for about 10 minutes and the images can be downloaded any number of times. Clicking on the “Go Back” button will redirect the users back to the “Download Screen” and clicking the “Start Over” button will take them to the homepage.



Instagram Directions Page:

This page is simply directions on how to upload the images to Instagram. The requirement was originally that the user be able to do this from the app, but it is impossible without partnering with Instagram and with Instagram's current API only not allowing 3rd-party application to directly upload images. We will instead provide clear instructions on how it can be achieved using Instagram's official application.

6. Component & Process Design

Front End:

The button for image uploads will be an input tag with the "type" attribute set to "file", the "accept" attribute set to ".png, .gif, .jpg, .jpeg", and the "multiple" property set. This is a fairly standard set of features that will work on all modern browsers.

The front end's functionality is largely defined by the flowchart (Section 4) and the permissible types of requests (Section 6). Error handling falls squarely into this section, however. When any

error is received from the server, a JS alert forwards the error message to the user. Then, if this error occurred…

- during the initial upload, no further action is taken, and the user stays on the home page.
- when requesting an image, the app behaves as if the user immediately presses “Next Image”.
- when requesting an MIA, the MIA is taken to be the whole image.
- when uploading an MIA, the user is prompted to retry. If they elect not to, the app proceeds to the next image.
- when downloading all cropped images, the user is prompted to retry. If they elect not to, the app returns to the home screen.

Back End:

The back end will be an ASP.NET web service. It will use existing .NET functionality to map requestable URLs to functions written by the project team. It shall store user sessions (as defined in Data Design -> Back End Data), and on each session creation event it shall iterate over the stored sessions and remove any which have expired.

Image processing will be done by the ImageProcessor library. There will be three distinct algorithms, two of which can be selected by a button on the MIA selection screen and one of which performs the actual cropping. These are detailed below:

func {int,int,int, int} getInterestingRectangle(image)

- Parameters:
  - image - takes in the uploaded image
- Returns a bounding rectangle of pixels around the most interesting area
  - Returned as a set of four numbers representing the {maxX, minX, maxY, minY}.
- Uses cropEntropy() from ImageProcessor to remove mundane parts of the image. Then it will use detectEdges() to find edges in the image. The result of detectEdges() will also be feed into cropEntropy() to crop out areas of the image that do not have edges. The final bounded rectangle returned will be the intersection of the bounding rectangles from the two processes.

func {int,int,int, int} ruleOfThirds(image)

- Parameters:
  - image - takes in the uploaded image
- Uses the output of the above algorithm, but returns a bounding rectangle such that the first result occupies the left one-third.

func image crop({int,int,int,int}, image)

- Parameters:
  - Bounding rectangle as {maxX, minX, maxY, minY} to be cropped to
  - image - takes in the uploaded image
- Receives an image and a bounding rectangle and returns that image cropped to that boundary.

Of course there will be images for which neither MIA detection algorithm yields acceptable results, but if one continually adds new algorithms, eventually the choice of algorithm is as daunting as the task of manually cropping. If the user is dissatisfied, they may manually adjust the cropping region.

Should a file which is not an image be sent to the server which is not an image, ImageProcessor will fail to parse the file. This will throw an exception, which is to be handled by removing the offending file from the list of files for that session. Any requests for out-of-bounds image indices (which the client is likely to generate in this case, as it is unaware of the removed file) will instead receive an error with the text “Requested image not found”.

7. Data Design

Data for the project can be broadly split into front-end data, transfer data, and back-end data.

Front End Data:

The webpage itself only needs to keep track of selected files, a session ID, and the content and MIA of the *current* image. The list of selected files is simply a list of strings, and the session ID is a string (discussed in more detail in the back end). The MIA is a set of 4 integers defining a rectangle, while the image is stored in the browser’s default manner for any displayed image.

Transfer Data:

The webpage will send information to the server as AJAX FormData objects, and all image uploads will be handled in the same way as normal file uploads. Should malicious users circumvent the file extension filter, the transfer occurs regardless - see Section 6 for further details on how this is handled. Images more than 15 MB will be rejected by the client, and additionally by the server should malicious users circumvent the JS file size limit. The server’s replies will be simple JSON objects, and when images need to be sent to the web page, a URL for downloading said image will be sent as part of the JSON. Any errors generated by the server will be transmitted as JSON objects with a single key (“error”) with the value equal to the error message. The following types of transfers may occur:

<u>Initial Upload</u>
Form contents: Multiple images
Expected reply: Session ID

<u>Request for Image</u>
Form contents: Session ID, image index
Expected reply: URL for image (only guaranteed valid until next Request for Image from this Session ID, or until Request All)

<u>Request for MIA</u>
Form contents: Session ID, image index, MIA algorithm index
Expected reply: Coordinates for MIA

MIA Upload

Form contents: Session ID, image index, MIA boundary coordinates

Expected reply: Empty JSON object. Server silently performs cropping computation before reply, to ensure synchronization.

Request All

Form contents: Session ID

Expected reply: URLs for all cropped images (only guaranteed valid for 20 minutes)

Back End Data:

The back end needs to keep track of data for multiple users. Sessions do not need to last past a restart of the service, nor do any joins or other complex operations need to be performed, so a full database is unnecessary. All that is required is a dictionary from Session IDs to Session objects, each of which contains a "last activity" date and an array of images. Below is detail for each type of data:

Session IDs: Strings of 20 characters, each of which comes from the standard set for base64 encoding: 0-9, a-z, A-Z, +, /. These shall be pseudorandomly generated to minimize the possibility of guessing another user's session.

"Last activity" dates will be stored as a DateTime object.

Images before processing: raw file data as provided by the user. Not to exceed 15 MB, as described in Transfer Data.

Images during processing: converted to ImageProcessor native format for processing

Images after processing: raw file data in a png format

8. Algorithm/Library Appendix

The following libraries / technologies will be used:

Front end:

- HTML
- CSS
- Bootstrap
- JavaScript
    - jQuery

Back end:

- ASP core - For web hosting
    - For demonstration purposes, the site will be hosted on a team member's machine. Deployment to the cloud is not considered part of development anymore than marketing.
- C#

- ImageProcessor
    - A .NET library for processing images