Project Iteration 3

CSE 6324: Advanced Topics in Software Engineering

Project Name : Shape Recipe

Team 1
Harsh Chaludia
Deep Patel

Abstract:

A. Planning:

1. Features
2. Planning
3. Similar related apps in the market
4. How we differ from similar apps in the market.
5. Five biggest risks in terms of time constrain.
6. Plans to deal with risks.

B. Specification & Design:

1. Inputs and Outputs
2. Data structures
3. Use-cases

C. Code & Tests:

1. Code snippets
2. Test cases
3. Instruction to compile and run the code
4. UI Layout

D. Customers & Users:

1. Who are the customers and their needs
2. Customer Feedback

E. Reference:

A. Planning

1. Features:

We load the image file with a resized factor typically smaller, to outline the shapes as well as digits with a good estimation. Next, we grayscale the image for recognizing the shapes, find contours then we grab the contours, and finally initialize the shape detector.py, and label the images. Whereas for recognition of number, we have trained a model from using 500 images for each digit from 0-9, which will help us identify the digits. After that, we create a JSON object with shapes and digits and dump it on the PHP page, or a server to decode it with an array. After that, we Map each object with skeleton elements like header, footer, cards, slider, etc. Finally, getting the JSON object we generate a website, or a react native app that renders the app both on IOS as well as android.

2. Planning:

We have decided to use EXPO which is a native app framework – opensource. This will help us to render the application to both in android as well as IOS. Secondly, we will be having an instruction manual for sketches/shapes as well as for digits to help in deciding the design for the website. Lastly, we will give the download/save button to download the source code for the final website generated.

3. Similar related apps in the market:

1. Fronty (https://fronty.com)

   Fronty is a paid application which takes an image as an input from the from user and converts that image into an HTML website and also gives the facility to make the website live. So it is required that the uploaded image is fully designed website photoshop image. The app then converts the image to HTML website. It provides the facility to manual change the content of the website before it is published. Finally, if the user is satisfied with the website converted, then he/she publishes the website which is also one of the feature of the app to publish directly from the app.

2. Sketch-code (https://github.com/ashnkumar/sketch-code)

   Sketch-code is an open source project Ashwin Kumar(https://github.com/ashnkumar) which converts the hand drawn wireframe of the website to generate HTML markup. So the app scans the image provided by the user and then with help of their AI model it identifies text and shapes in the image. Finally, once the shapes are recognized the shapes are converted into HTML elements like div and button.

4. How we differ from similar apps in the market:

1. Fronty (https://fronty.com)

   Fronty is a strong competitor for our app in the market. However, there are quite distinguishing features in our app which gives us the upper hand in comparison to fronty. Firstly, we have a manual of shapes that are associated with code, so user need not need a high photoshop image made by a professional UI/UX developer. Secondly, we

have 2 options to choose from for the final code either website or the mobile application UI.

2. Sketch-code (https://github.com/ashnkumar/sketch-code)

   Sketch-code basic idea seems to match with our overall idea of hand drawn images. However, Sketch-code app is still in research phase and doesn't give market useable product /fully developed website code. Moreover, it uses the AI model which separates every element in the image not just limited to proper shapes but to text and other uneven shapes while we are associating code with the shapes.

5. Five biggest risks in terms of time constraint:

Risk 1: Considering the edge detection method, it could be a tedious task if we draw with a doodle element where the sketch could slightly change the polygons. So, we give a probability of 70% for this scenario and 15 hour effect which gives the risk effect of 10.5 hours for this risk.

Risk 2: To render the application on React Native platform, we need to save the results/ outcome in an API call to save progress. So, we give a probability of 60% for this scenario and 12 hour effect which gives the risk effect of 7.2 hours for this risk.

Risk 3: Without knowing instructions to sketching, would confuse the users, and they might end up making an incorrect version of their design. So, we give a probability of 50% for this scenario and 8 hour effect which gives the risk effect of 4 hours for this risk.

Risk 4: We observed, we only took shapes as the method to draw elements like footer, header, etc. This limited us to include limited elements. So, we give a probability of 60% for this scenario and 15 hour effect which gives the risk effect of 9 hours for this risk.

Risk 5: For mobile, Xcode, and android studio is already into drag & drop. So, we give a probability of 40% for this scenario and 10 hour effect which gives the risk effect of 4 hours for this risk.

6. Plans to deal with risks:

Risk 1: To overcome this issue, we can ask the user to drag and drop shapes from the toolbar, to approximate the shapes.

Risk 2: We need to have a temporary database, and the ideal solution would be using NoSQL database.

Risk 3: We can make an instruction manual in a separate page on the website.

Risk 4: We decided to use more colors to also have styling of elements associated to geometric shapes. Also, we are planning to have numeric digits to have more choices for users.

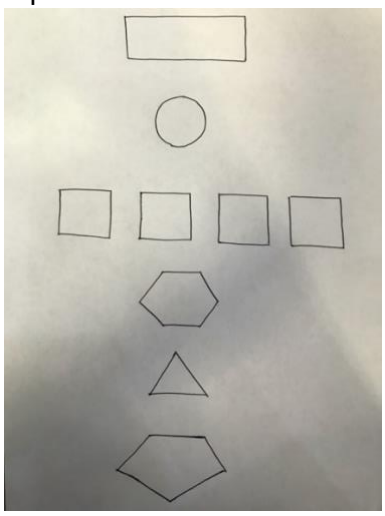
Risk 5: To avoid already existing features, and since we are using React Native, we will incorporate themes made by open source theme providers.

B. Specifications & Design

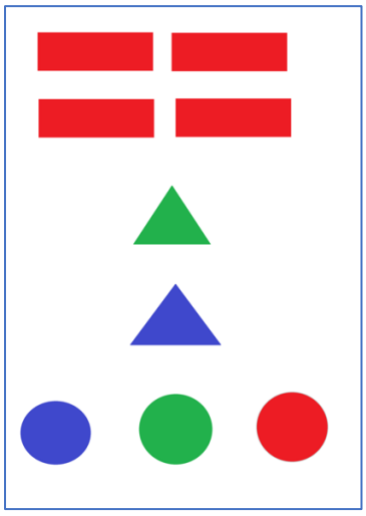
1. Inputs and Outputs:

The input for our app will be an image of either a hand-drawn sketch of shapes/digits or drawn using some drawing tools like paint or photoshop and the output will a website along with its source code.

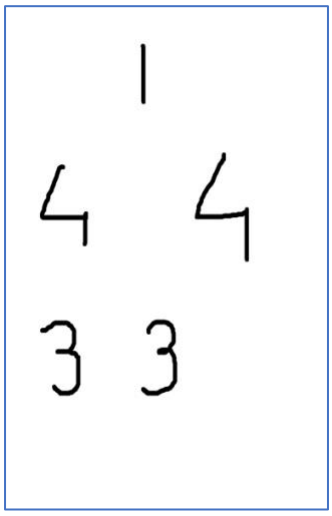
Input:



Hand drawn sketch

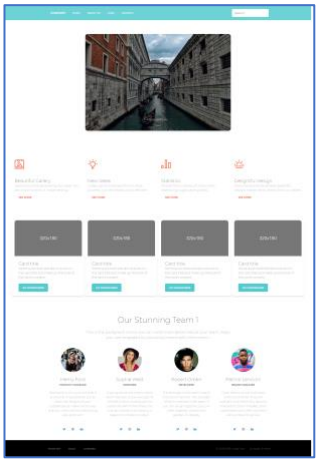


Sketch drawn in MS Paint



Sketch for digits

Output:



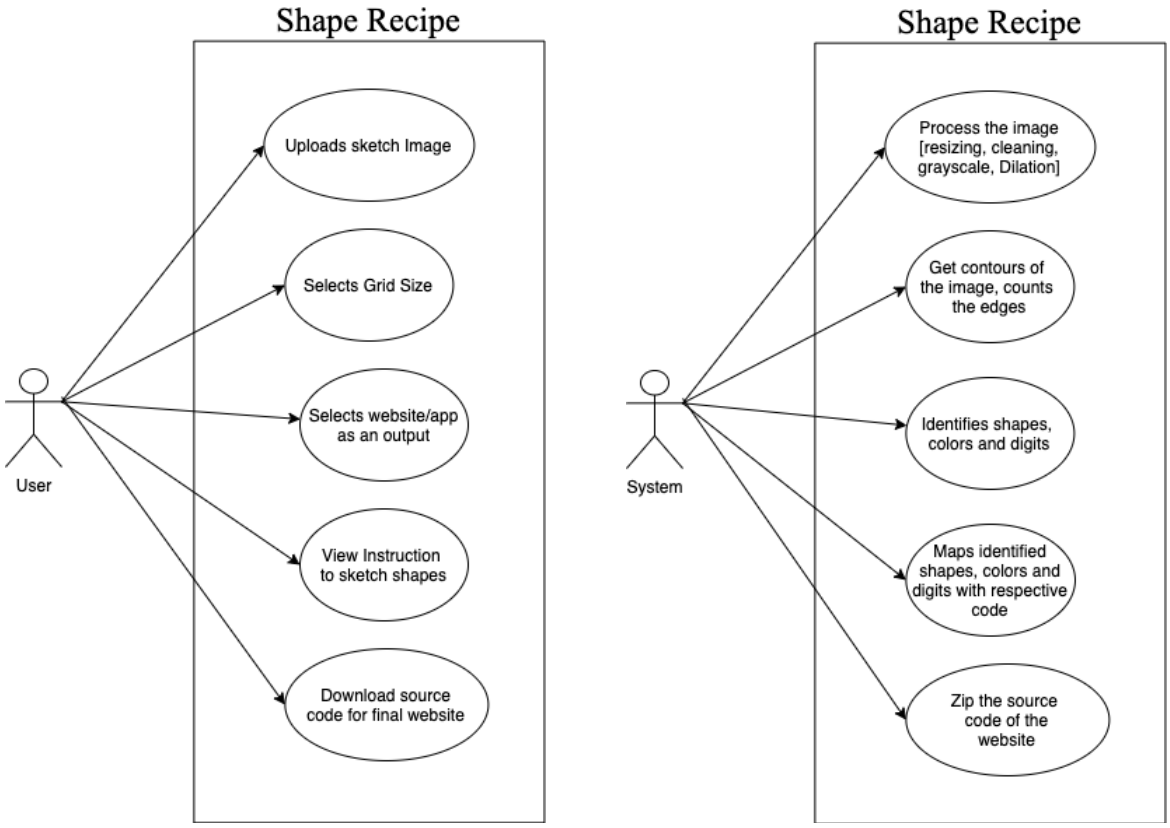
Website(Image clickable).

Mobile app(image clickable)

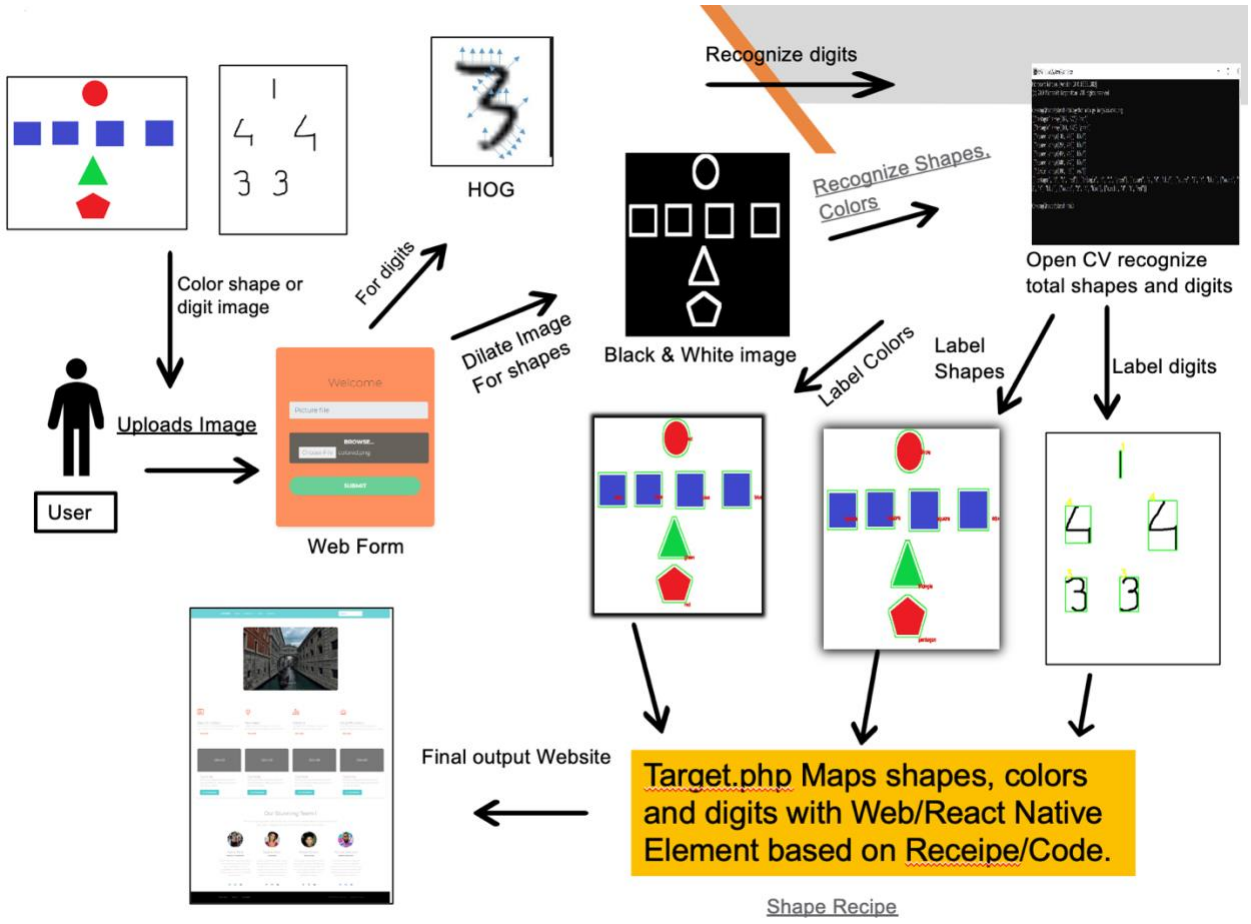
2. Data structures:

We are using Pandas Library for managing the data which we got after identifying either shapes, color, or numbers from the sketch image. We have 6x6, 5x5, 4x4 grid options for shapes as well as digits that can we drew i.e. 6 shapes at max in rows and columns for 6x6 and similarly for 5x5 and 4x4 grid. And once these values are identified correctly then they are managed in the matrix. In the end, code is appended iteratively in the final website which is mapped with the values of each cell in the matrix. The code will be stored in the JSON format and this JSON will be called every time we need to do the mapping of elements.

3. Use Cases:



Use case diagrams



As per the rubric, along with use case diagram we are also providing here screen shot transition diagram

C. Code & Tests

1. Code snippets:

Main Function – Detect Shapes, and Colors from the input image.

```
  # Read the image data from argument
image = cv2.imread(argv[1])
# Resize if it is necessary(faster but not accurate)
resized = imutils.resize(image, width=800)
ratio = image.shape[0] / float(resized.shape[0])
blurred = cv2.GaussianBlur(resized, (5, 5), 0)
gray = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)
lab = cv2.cvtColor(blurred, cv2.COLOR_BGR2LAB)
thresh = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)[1]
# Get the edge by canny effect
canny = cv2.Canny(gray.copy 80, 120)
# Set the kernel for dilation(make the white line thicker)
kernel = np.ones((3, 3), np.uint8)
dilation = cv2.dilate(canny.copy(), kernel, iterations=7)
cv2.imwrite('dilate.jpg', dilation)
# Get the contour from dilated picture, EXTERNAL contour used
cnts = cv2.findContours(dilation.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
cnts = imutils.grab_contours(cnts)
# ColorLabel
# initialize the ShapeDetector class
sd = ShapeDetector()
cl = ColorLabeler()
oracle = []
for c in cnts:
   try:
      # Do if the contourarea is larger than specific area
      if cv2.contourArea(c) > np.shape(image)[0] / 40 * np.shape(image)[1] / 40:
         # Get the moments of the shapes
         M = cv2.moments(c)
         cX = int(M['m10'] / M['m00'])
         cY = int(M['m01'] / M['m00'])
         # Detect the shape
         shape = sd.detect(c)
         color = cl.label(lab, c)
         # Put additional information in the image
         c = c.astype('float')
         c *= ratio
         c = c.astype('int')
         cv2.drawContours(image, [c], -1, (0, 255, 0), 2)
         cv2.putText(image,shape,(cX, cY),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0, 0, 255),2,)
         # Store the shape information to oracle
         oracle.append([shape, np.array([cX, cY]),color])
   except:
      continue
# Save the image file
cv2.imwrite('Shape_recognized.jpg', image)
```

```
alsoOracle = oracle
oracle = np.array(oracle)
#print(oracle)
# Create the json file to return the following value to PHP
# ["Type of shape", "RowID", "number of shape in the same row"]
jsonfile = get_block(image, oracle, (6, 6),alsoOracle)
print(jsonfile)
```

Get Block Function -- "Type of shape", "RowID", "Total shapes in row", ”Color”

```
def get_block(image, oracle, split, alsoOracle):
    # Get the distance of unit
    unit_X = np.shape(image)[0] / split[0]
    unit_Y = np.shape(image)[1] / split[1]
    # Get the row ID and return in data
    data = []
    for instance in oracle:
        row = math.floor(instance[1][1] / unit_X)
        data.append([instance[0], row])
    data = np.array(data)
    # Get the set of rowID
    index_full = set(data[:, 1])
    # Calculate the number of shapes in the same row
    post = []
    for index in index_full:
        sum = 0
        for i in range(0, len(data)):
            if data[i, 1] == index:
                sum += 1
        post.append([index, sum])
    post = np.array(post)
    # add the row which specifies the number of shape in the same rows
    data = np.hstack((data, np.zeros((np.shape(data)[0], 1))))
    for i in range(0, np.shape(post)[0]):
        for j in range(0, np.shape(data)[0]):
            if post[i, 0] == data[j, 1]:
                data[j, 2] = post[i, 1]
    # return json file to process in json
    data = data.tolist()
    for x in range(len(data)):
        data[x].append(alsoOracle[x][2])
    return json.dumps(data)
```

performRecognition Function – Detect numbers from the input image.

```
# Get the path of the training set
parser = ap.ArgumentParser()
parser.add_argument("-c", "--classiferPath", help="Path to Classifier File", required="True")
parser.add_argument("-i", "--image", help="Path to Image", required="True")
args = vars(parser.parse_args())

# Load the classifier
clf, pp = joblib.load(args["classiferPath"])
```

```
# Read the input image
im = cv2.imread(args["image"])

# Convert to grayscale and apply Gaussian filtering
im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im_gray = cv2.GaussianBlur(im_gray, (5, 5), 0)

# Threshold the image
ret, im_th = cv2.threshold(im_gray, 60, 255, cv2.THRESH_BINARY_INV)

# Find contours in the image
ctrs, hier = cv2.findContours(im_th.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Get rectangles contains each contour
rects = [cv2.boundingRect(ctr) for ctr in ctrs]


oracle = []
# For each rectangular region, calculate HOG features and predict
# the digit using Linear SVM.
for rect in rects:
    # Draw the rectangles
    cv2.rectangle(im, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 255, 0), 3)
    # Make the rectangular region around the digit
    leng = int(rect[3] * 1.6)
    pt1 = int(rect[1] + rect[3] // 2 - leng // 2)
    pt2 = int(rect[0] + rect[2] // 2 - leng // 2)
    roi = im_th[pt1:pt1+leng, pt2:pt2+leng]
    # Resize the image
    roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)
    roi = cv2.dilate(roi, (3, 3))
    # Calculate the HOG features
    roi_hog_fd = hog(roi, orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1), visualize=False)
    roi_hog_fd = pp.transform(np.array([roi_hog_fd], 'float64'))
    nbr = clf.predict(roi_hog_fd)
    cv2.putText(im, str(int(nbr[0])), (rect[0], rect[1]),cv2.FONT_HERSHEY_DUPLEX, 2, (0, 255, 255), 3)
```

```
    oracle.append([str(int(nbr[0])), np.array([rect[0], rect[1]])])

#print(oracle)
#cv2.namedWindow("Resulting Image with Rectangular ROIs", cv2.WINDOW_NORMAL)
#cv2.imshow("Resulting Image with Rectangular ROIs", im)
#cv2.waitKey()


cv2.imwrite('Shape_recognized.jpg', im)
alsoOracle = oracle
oracle = np.array(oracle)
```

Get Block Function for numbers -- "Digit", "RowID", "Total digits in row”

```
# Get the distance of unit
image = im
split = (6,6)

unit_X = np.shape(image)[0] / split[0]
unit_Y = np.shape(image)[1] / split[1]

# Get the row ID and return in data

data = []
for instance in oracle:
    row = math.floor(instance[1][1] / unit_X)
    data.append([instance[0], row])
data = np.array(data)

# Get the set of rowID

index_full = set(data[:, 1])

# Calculate the number of shapes in the same row

post = []
for index in index_full:
```

```
    sum = 0
    for i in range(0, len(data)):
        if data[i, 1] == index:
            sum += 1
    post.append([index, sum])
post = np.array(post)

# add the row which specifies the number of shape in the same rows

data = np.hstack((data, np.zeros((np.shape(data)[0], 1))))
for i in range(0, np.shape(post)[0]):
    for j in range(0, np.shape(data)[0]):
        if post[i, 0] == data[j, 1]:
            data[j, 2] = post[i, 1]

# return json file to process in json
print(json.dumps(data.tolist()))
```

Shape Detection Function – Using Ramer-Douglas-Puecker algorithm using approxPolyDp function from cv2, and edge detection method.

```python
# import OPENCV
import cv2

class ShapeDetector:
    # Nothing to initialize
    def __init__(self):
        pass

    # Method to detect the shape
    def detect(self, c):
        # Return "unidentified" if the shape doesn't match any shape
        shape = "unidentified"
        # Approximate the contours by polinomial shape
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.04 * peri, True)
        # Triangle
        if len(approx) == 3:
            shape = "triangle"
        # Rectangle
        elif len(approx) == 4:
            (x, y, w, h) = cv2.boundingRect(approx)
            por = w / float(h)
            # if the threshold satisfies==> square, if not rectangle
            shape = "square" if por >= 0.6 and por <= 1.4 else "rectangle"
        # Pentagon
        elif len(approx) == 5:
            shape = "pentagon"
        # Hexagon
        elif len(approx) == 6:
            shape = "hexagon"

        # if many edges than octagon --> Circle
        else:
            shape = "circle"

        # return the type of the shape
        return shape
```

Color Detection Function – Using Lab Color Space, by converting RGB values to (L*,a*,b*)

```
class ColorLabeler:
    def __init__(self):
        # initialize the colors dictionary, containing the color
        # name as the key and the RGB tuple as the value
        colors = OrderedDict({'red': (255, 0, 0), 'green': (0, 255, 0),'blue': (0, 0, 255)})
        # allocate memory for the L*a*b* image, then initialize
        # the color names list
        self.lab = np.zeros((len(colors), 1, 3), dtype='uint8')
        self.colorNames = []
        # loop over the colors dictionary
        for (i, (name, rgb)) in enumerate(colors.items()):
            # update the L*a*b* array and the color names list
            self.lab[i] = rgb
            self.colorNames.append(name)
        # convert the L*a*b* array from the RGB color space
        # to L*a*b*
        self.lab = cv2.cvtColor(self.lab, cv2.COLOR_RGB2LAB)
    def label(self, image, c):
        # construct a mask for the contour, then compute the
        # average L*a*b* value for the masked region
        mask = np.zeros(image.shape[:2], dtype='uint8')
        cv2.drawContours(mask, [c], -1, 255, -1)
        mask = cv2.erode(mask, None, iterations=2)
        mean = cv2.mean(image, mask=mask)[:3]
        # initialize the minimum distance found thus far
        minDist = (np.inf, None)
        # loop over the known L*a*b* color values
        for (i, row) in enumerate(self.lab):
            # compute the distance between the current L*a*b*
            # color value and the mean of the image
            d = dist.euclidean(row[0], mean)
            # if the distance is smaller than the current distance,
            # then update the bookkeeping variable
            if d < minDist[0]:
                minDist = (d, i)
        # return the name of the color with the smallest distance
        return self.colorNames[minDist[1]]
```

Classifier Function – To create model for number recognition.

```
dataset = fetch_openml('mnist_784')

# Extract the features and labels
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')

# Extract the hog features
list_hog_fd = []
for feature in features:
    fd = hog(feature.reshape((28, 28)), orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1))
    list_hog_fd.append(fd)
hog_features = np.array(list_hog_fd, 'float64')

# Normalize the features
pp = preprocessing.StandardScaler().fit(hog_features)
hog_features = pp.transform(hog_features)

print("Count of digits in dataset", Counter(labels))

# Create an linear SVM object
clf = LinearSVC()

# Perform the training
clf.fit(hog_features, labels)

# Save the classifier
joblib.dump((clf, pp), "digits_cls.pkl", compress=3)
```

2. Test cases:

ID	Test Case Objectives	Pre-requisites	Steps	Input Data	Expected Output	Status
1	Detect Shapes	Image should be given	1. Resize 2. Ratio 3. Blur 4. Grayscale 5. Thresholding 6. Dilation 7. Find Contours 8. Detect shapes with edges	1. image.png	Classified Shapes	Pass
2	Detect Colors	Image should be given	1. Resize 2. Ratio 3. Blur 4. Grayscale 5. Converting the RGB to LAB color space(L*,A*,B*) 6. Detect colors with LAB values	1. image.png	Classified Colors	Pass
3	Detect Numbers	Image should be given	1. Resize 2. Ratio 3. Blur 4. Grayscale 5. Deskewing 6. Histogram of Oriented Gradients descriptor 7. Training and testing the model 8. Detect numbers	1. image2.png	Classified Numbers	Pass
4	Generate Website	Image and filter for website needs to be given	1. Click an image 2. Upload the image 3. Select the website filter 4. Submit image 5. View the design and download source code	1. image.png 2. type=website	HTML, CSS, Jacascript	Pass
5	Generate Mobile App	Image and filter for mobile app needs to be given	1. Click an image 2. Upload the image 3. Select the mobile app filter 4. Submit image 5. View the design and download source code	1. image.png 2. type=mobile app	React Native Ecmascript	Pass

3. Instruction to compile and run the code:

Getting the files

```
git clone https://github.com/utastudents/ShapeRecipe.git
```

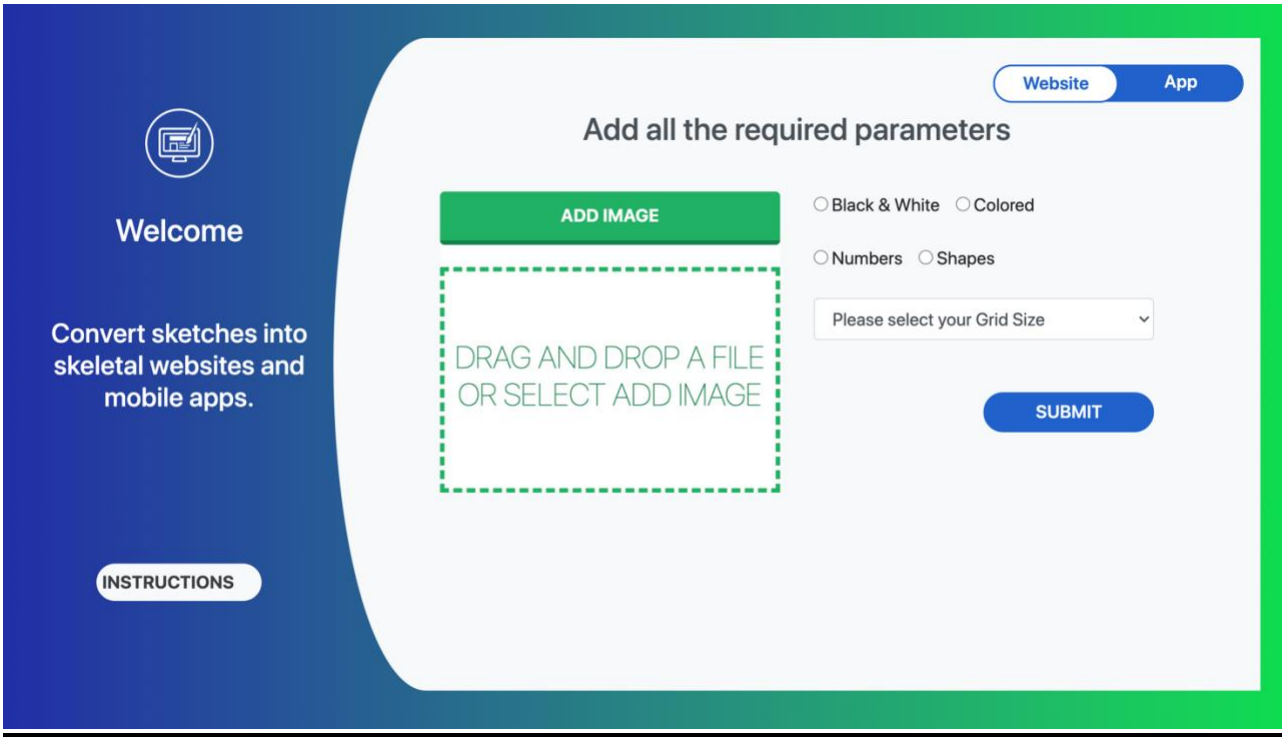
Choose an image or Sketch one.

1. Upload an image, or sketch with shapes.
2. Submit the image for processing.
3. Run it over a localhost.

Prerequisites

- Python 3 (not compatible with python 2)
- pip
- Php

4. UI Layout:



D. Customers & Users

1. Who are the customers and their needs:

Our potential customers are teenagers with age ranging from 13-19 and people who don't have any prior coding experience or people who are into web design.

2. Customer feedback:

For the feedbacks we have asked our cousins and friends from India, who are in high school or in the first year of their Bachelor's degree, and importantly they all are someone who shares their interest in web design but doesn't have any prior experience with the coding.

After showing and giving the demo to our cousins, they thought that using numbers inside the shape would be tricky and hard to remember. So we decided to keep numbers separate from the shapes.

E. Reference

- https://fronty.com/
- https://github.com/ashnkumar/sketch-code
- https://pandas.pydata.org/
- https://opencv.org/
- https://blog.expo.io/developing-a-react-native-app-with-expo-cf6566732311
- https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/
- https://www.researchgate.net/figure/An-example-of-histogram-oriented-gradient-features-6-applied-to-digit-recognition_fig2_320410861
- https://blog.expo.io/developing-a-react-native-app-with-expo-cf6566732311
- http://hanzratech.in/python/handwritten-digit-recognition-using-opencv-sklearn-and-python/