ECE1779 - Introduction to Cloud Computing

Assignment 3 - **Serverless Computing**

Money Planner

Group members

Name	**UTorID**	**Student No.**	**Email id**
Adeel Syed	syedadee	1004689688	adeel.syed@mail.utoronto.ca
Yogesh Iyer	sivasan7	1005603438	yogesh.iyer@mail.utoronto.ca
Akshata Malghan	malghana	1004966860	akshata.malghan@mail.utoronto.ca

Table of Contents

Motivation

This project addresses two main issues related to keeping track of expenses:

1. Generally, all the receipts that are collected in any household, company or retail shops are sorted manually and their details are entered into the database using some type of system that requires quite a lot of effort. This process consumes a great deal of time and might be expensive in terms of labor hours. Thus, there is a need to convert this lengthy process into an automated one.

2. Also, tracking your expenditures can help you identify serious spending issues. You can also see if your spending matched your priorities. It is easier to make changes when you realize that you are not reaching your financial goals because you eat out every night. Tracking your spending allows you to see where your money is really going. It is essential if you want to understand your financial habits and make changes to them. It can help you realize when you need to stop spending, so you do not end up with a financial hangover.

Objective

In order to address the above mentioned problems, we have created a web application for planning your money expenditure, that can perform the following tasks:

1. Allow user to upload an image of a receipt taken by a camera or scanned

2. It then extracts the text information from it

3. This data is then cleaned to extract and store only the information that is of use for calculating expenses.

4. The user can now see how much he/she has spent in each category separately.

5. The cost limit reached can be seen under each category for the user.

6. The user can also search for details of specific category of receipts.

Each user can choose to sign up for face login to provide an extra layer of security, and prevent unauthorized access to their financial information.

Application Outline

As required in the project guidelines, we utilised AWS Lambda, DynamoDB and S3 services as platforms to build our application.

Along with these AWS services, we also made use of **AWS Textract.** Amazon Textract is a service that automatically extracts text and data from scanned documents. Amazon Textract goes beyond simple optical character recognition (OCR) to also identify the contents of fields in forms and information stored in tables.

Also, we have used **AWS Rekognition** for face detection during logging in. Amazon Rekognition is a cloud-based Software as a service (SaaS) computer vision platform that was launched in 2016. Amazon Rekognition makes it easy to add image and video analysis to your applications. You just provide an image or video to the Rekognition API, and the service can identify objects, people, text, scenes, and activities. It can detect any inappropriate content as well. Amazon Rekognition also provides highly accurate facial analysis and facial recognition. You can detect, analyze, and compare faces for a wide variety of use cases, including user verification, cataloging, people counting, and public safety.

A user can login by entering details and password or first register themselves using user details or face detection. The user details are stored in DynamoDB and images of faces are stored in a S3 bucket.

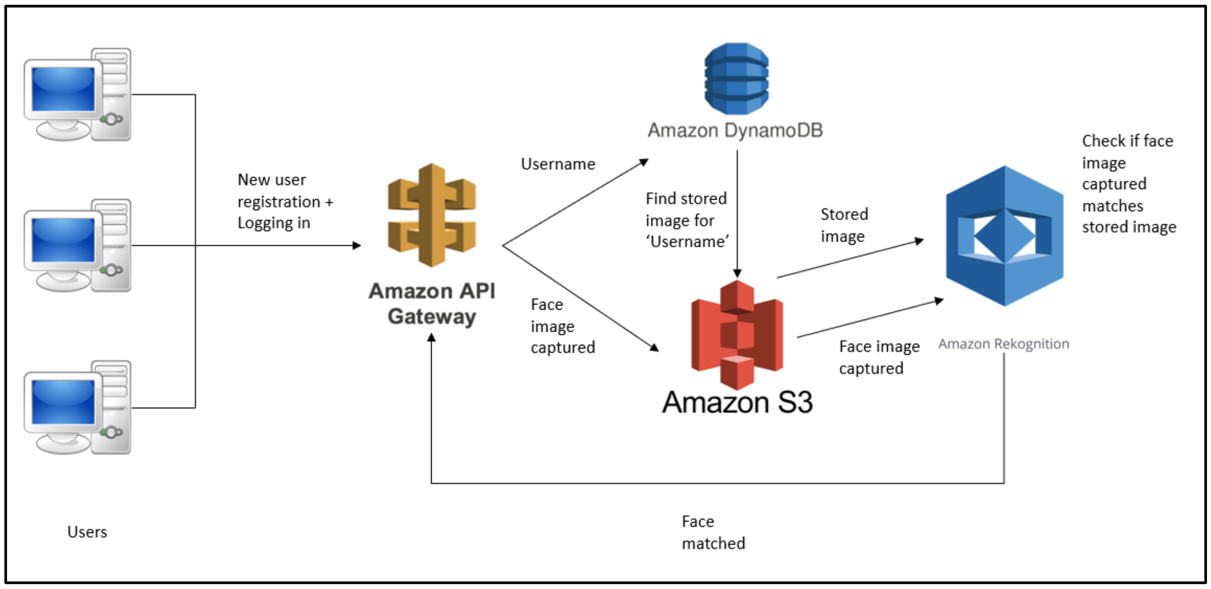


Figure 1: New Registration / Login Using Face Detection

When a user tries logging in using face detection, the face image stored under the entered username is found in S3. This stored image is then compared with the currently captured face image while logging in, using AWS Rekognition. If a match is found, the user is logged in.

Once the user is logged in, the user can start uploading receipt images. These are stored in the same S3 bucket under the user's folder (which are only visible to that user). Uploading an image into the S3 bucket triggers a Lambda function that converts these images to text using AWS Textract and stores these text files in another S3 bucket. Storing of a converted text file then triggers another Lambda function that extracts which store the receipt is from and the total amount spent. The name of the store is used to categorize it as either 'clothing', 'restaurant', 'retail' or 'misc'. The category and total amount spent is now stored in the second S3 bucket mentioned. The main application now uses this information to display the total expenditure in each category for the user.

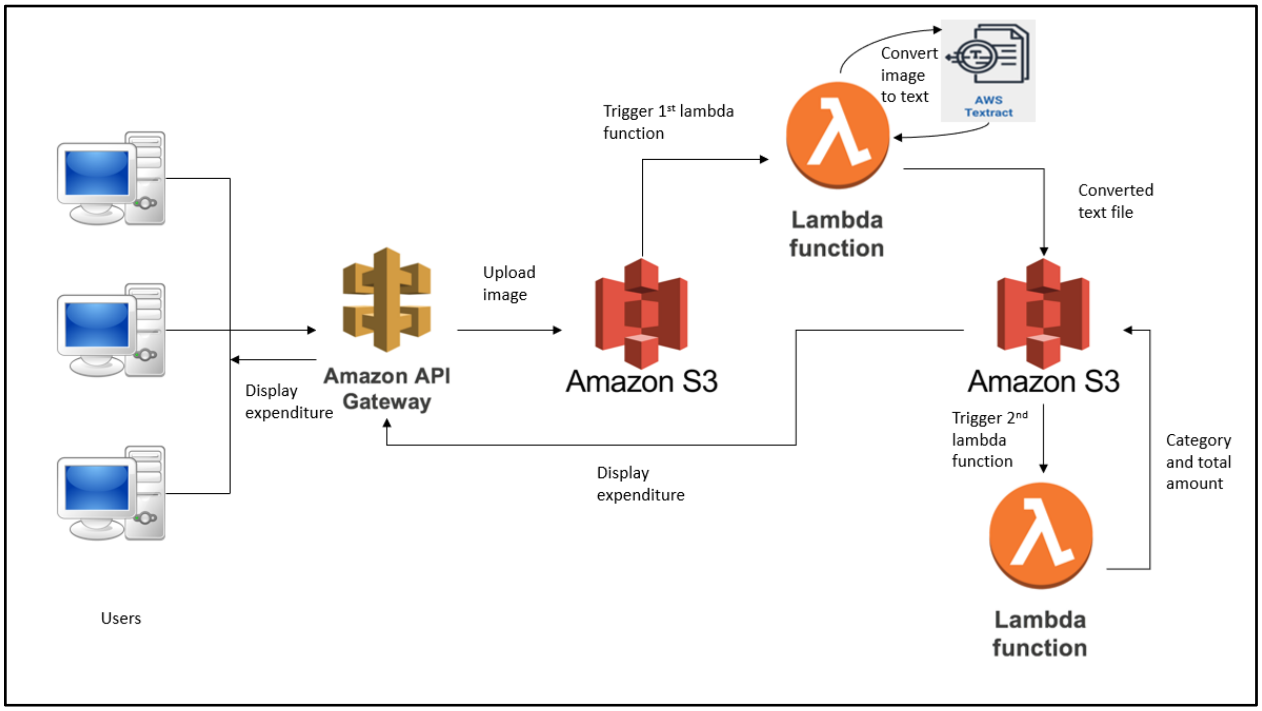


Figure 2: Uploading A Receipt Functionality

Walkthrough

We have access to the application using the following link:

https://x0it8vrvwa.execute-api.us-east-1.amazonaws.com/dev/

1. This link lands the user to the main home page. From here a user can navigate to the registration page or login page. A new user can register by entering their details or choose to sign up for face login for enhanced security. The following is an illustration of the manual method of registration:

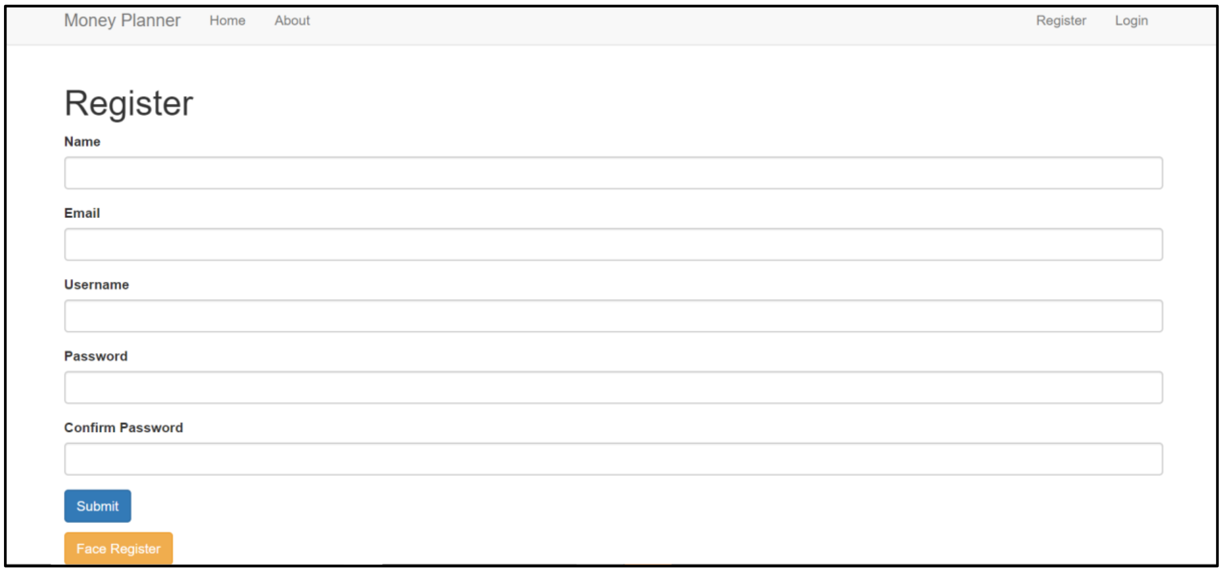


Figure 3: Registration Using Details

2. For face registration, the user has to enter a relevant username that has not been used before and click on the webcam window with their mouse to store the image. Do note if the webcam window is not clicked a new image is not saved by the application. Once the user has entered a username and clicked on the webcam window they can click on the Submit button. The process of face registration will be completed. The following is an example of the face registration page:

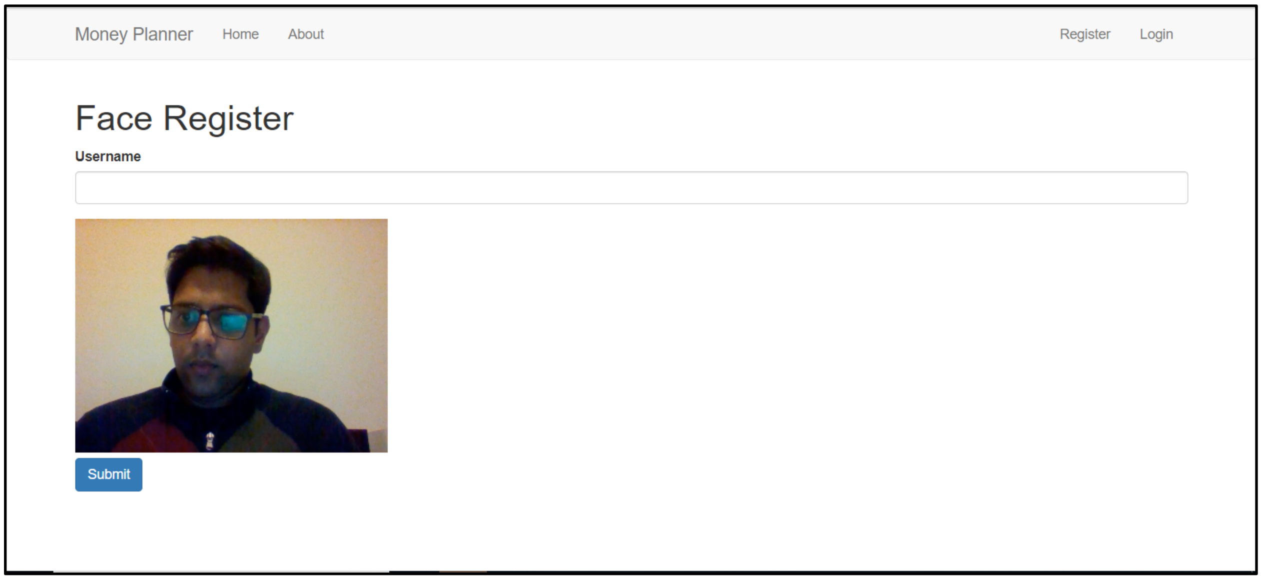


Figure 4: Registration Using Face Detection

3. Next, the user can now login with through manual input of username and password, as shown in Figure 5 below or can navigate to the face login method by clicking on the 'Face Unlock' button.

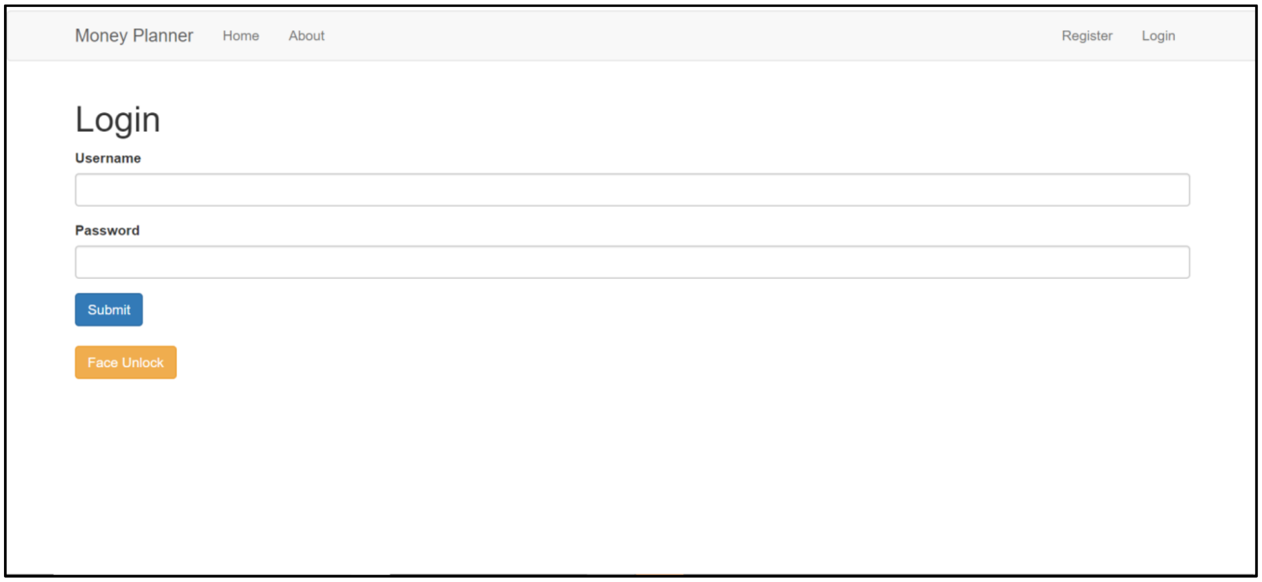


Figure 5: Login Using Details

4. Similar to face registration, the user has to enter a relevant username and click on the webcam window with their mouse to store the image. Do note if the webcam window is not clicked a new image is not saved by the application. Once the user has entered a username and clicked on the webcam window they can click on the Submit button. The

backend process will check whether the face on the newly saved image matches with the face image stored for that particular user. If the faces match, the user is logged in the web application navigates to the users Dashboard. The following is an example of the face login page.

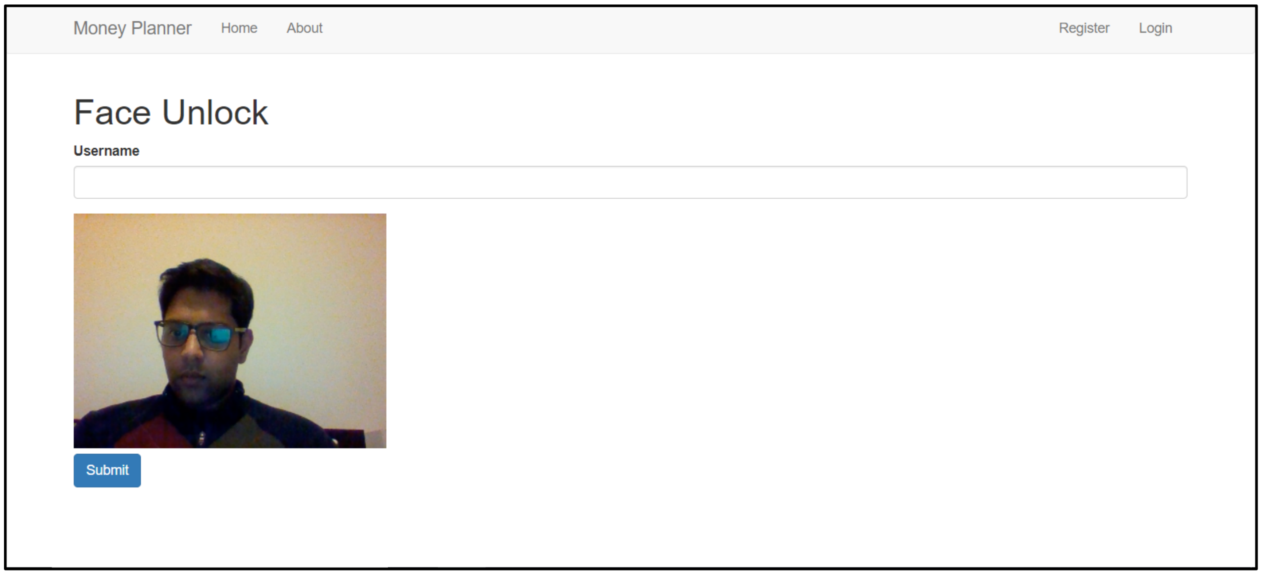
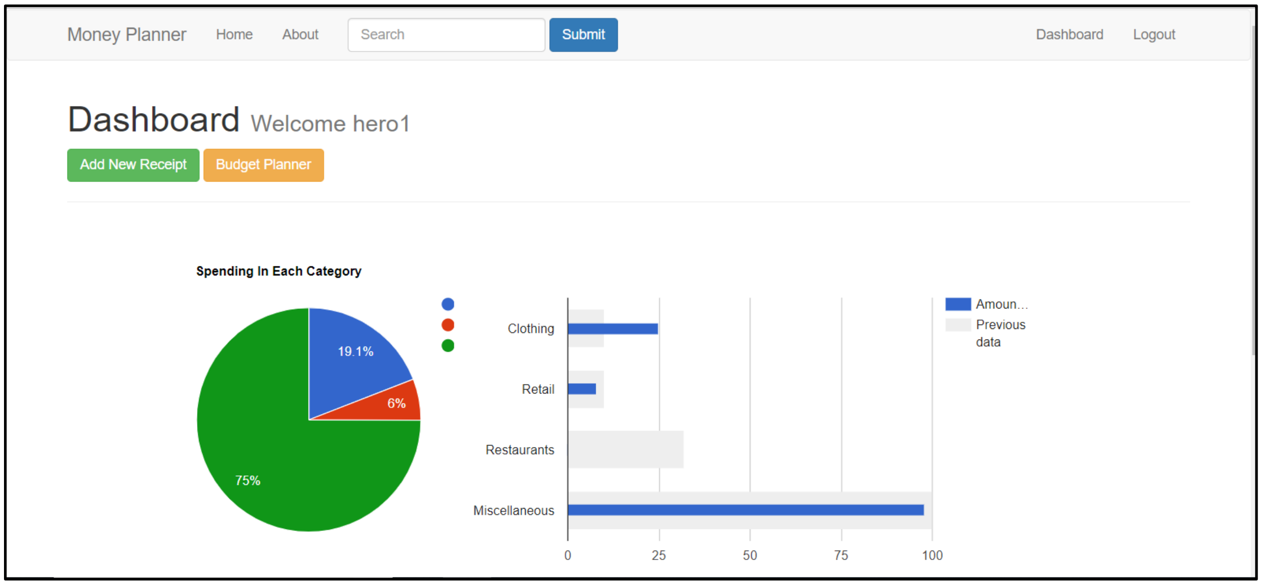


Figure 6: Login Using Face Detection

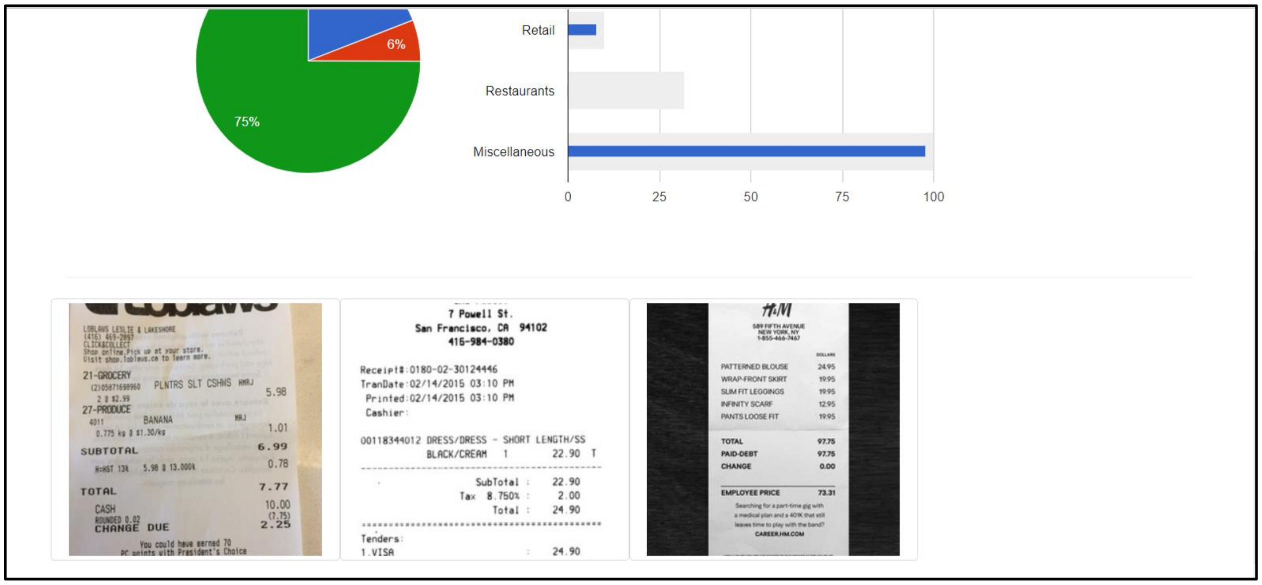
5. After logging in, the user can see a Dashboard in which charts are displayed that show the user's expenditure in each category as well as the amount spent against the allocated budget for the category. The pie chart shows the percentage of total spending in each category as well as the dollar amount. The bar chart shows the spending in each category against the budgetary allocation which is shown in light blue.

   The Dashboard also shows the thumbnails of the user's previously uploaded images, clicking on each thumbnail displays them in full size.

   From the Dashboard the user can click on the 'Add New Receipt' button to upload another receipt. The user can also update their budget allocations in each category by clicking on the 'Budget Planner' button. Updating budget values automatically adjusts the bar chart. The images in Figure 7 below provide an illustration of the user's Dashboard.



Scroll down:



```
LOBLAWS LESLIE & LAKESHORE
(416) 469-2897
CLICK&COLLECT
Shop online.Pick up at your store.
Visit shop.loblaws.ca to learn more.
21-GROCERY
 (2)05871698960   PLNTRS SLT CSHWS  HMRJ
                                       5.98
   2 @ $2.99
27-PRODUCE
 4011          BANANA             MRJ
                                       1.01
   0.775 kg @ $1.30/kg
SUBTOTAL                               6.99
   H=HST 13%   5.98 @ 13.000%          0.78
TOTAL                                  7.77
   CASH                               10.00
   ROUNDED 0.02                       (7.75)
   CHANGE  DUE                         2.25
        You could have earned 70
     PC points with President's Choice
```

```
                 7 Powell St.
          San Francisco, CA  94102
                 415-984-0380

 Receipt#:0180-02-30124446
 TranDate:02/14/2015 03:10 PM
  Printed:02/14/2015 03:10 PM
  Cashier:

 00118344012 DRESS/DRESS - SHORT LENGTH/SS
             BLACK/CREAM    1        22.90  T
 -----------------------------------------
                     SubTotal :      22.90
                Tax  8.750% :         2.00
                       Total :       24.90
 *****************************************
 Tenders:
 1.VISA                      :       24.90
```

```
                 H&M
          589 FIFTH AVENUE
            NEW YORK, NY
           1-855-466-7467
                               DOLLARS
 PATTERNED BLOUSE               24.95
 WRAP-FRONT SKIRT               19.95
 SLIM FIT LEGGINGS              19.95
 INFINITY SCARF                 12.95
 PANTS LOOSE FIT                19.95

 TOTAL                          97.75
 PAID-DEBT                      97.75
 CHANGE                          0.00

 EMPLOYEE PRICE                 73.31
   Searching for a part-time gig with
   a medical plan and a 401K that still
   leaves time to play with the band?
            CAREER.HM.COM
```

Figure 7: Dashboard Page

6. By clicking on the 'Add New Receipt' button, the user can upload a new receipt.

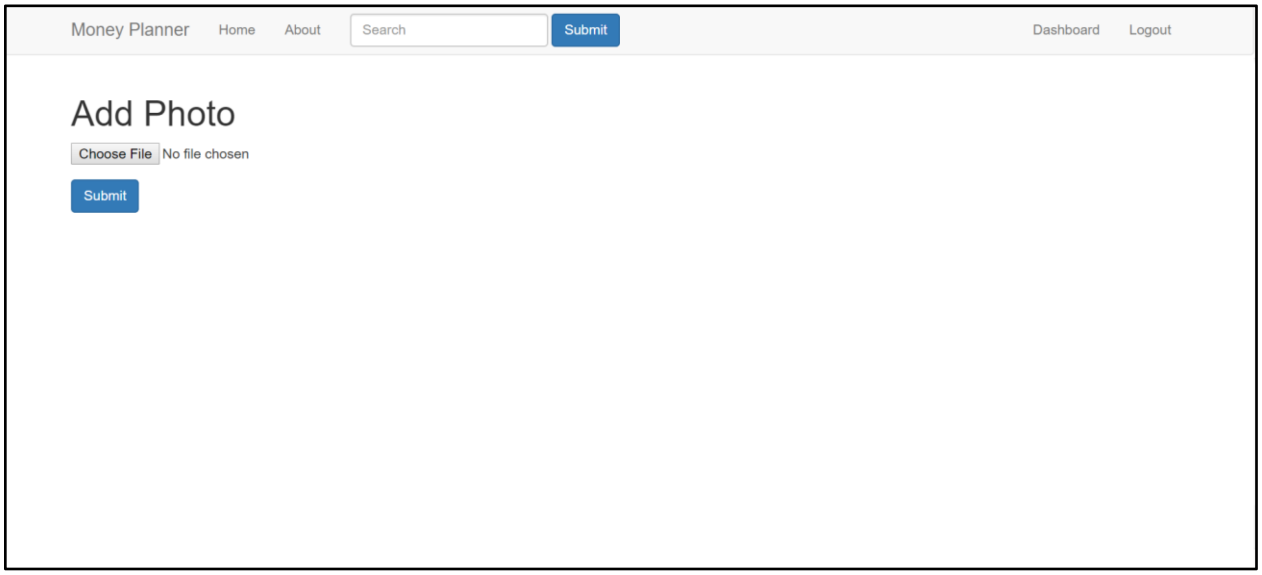


Figure 8: Uploading New Receipts

7. After the user uploads a new receipt, the backend Lambda function detects its category and the total amount from the receipt and automatically adjusts the charts on the Dashboard. Notice how when we uploaded a receipt for “Restaurants” category both charts updated from their states shown in Figure 7.

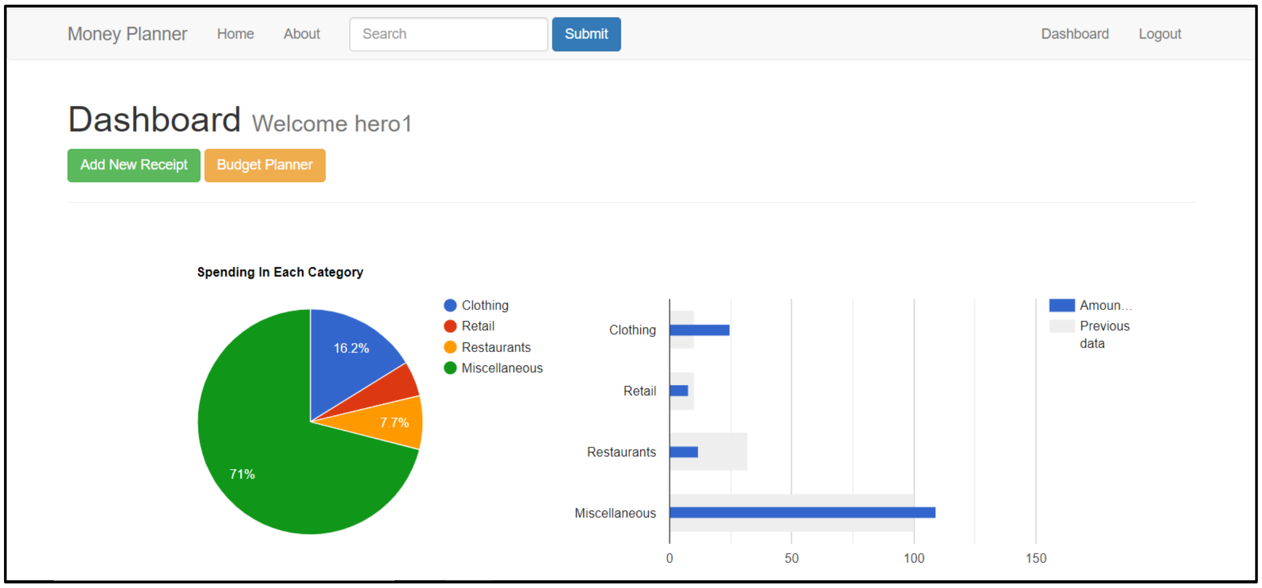
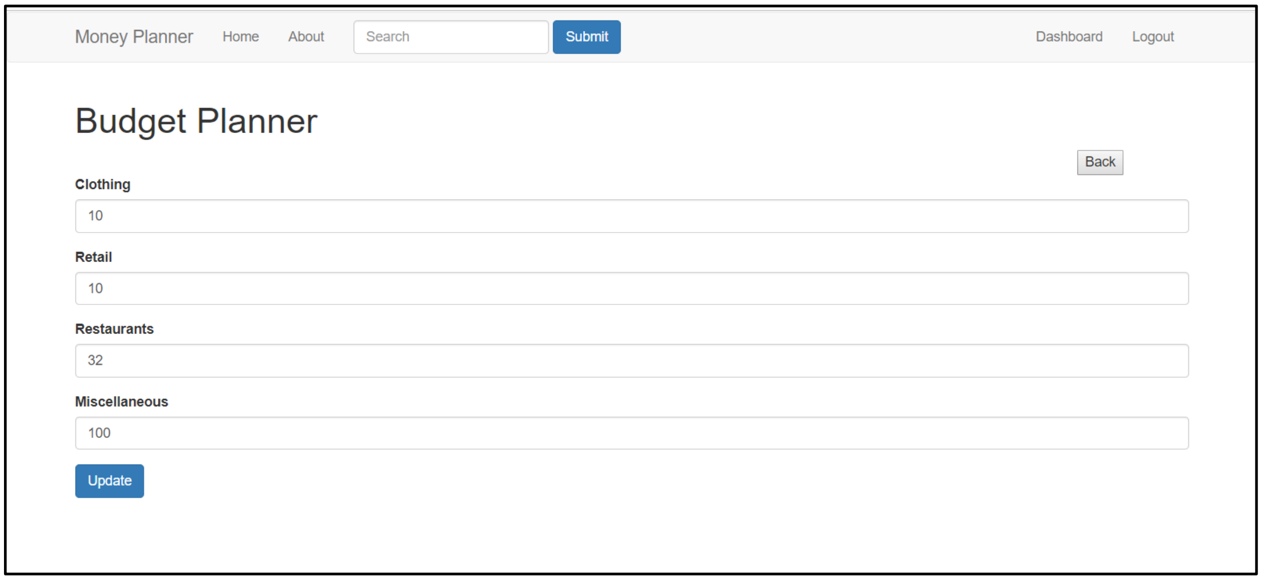


Figure 9: Update Charts

8. The user can click on the ‘Budget Planner’ button to navigate to the Budget form which shows the current allocations for each category, the user can change the values displayed

on the form and click 'Update' to update the values on the database as well as adjust the bar chart on the Dashboard.

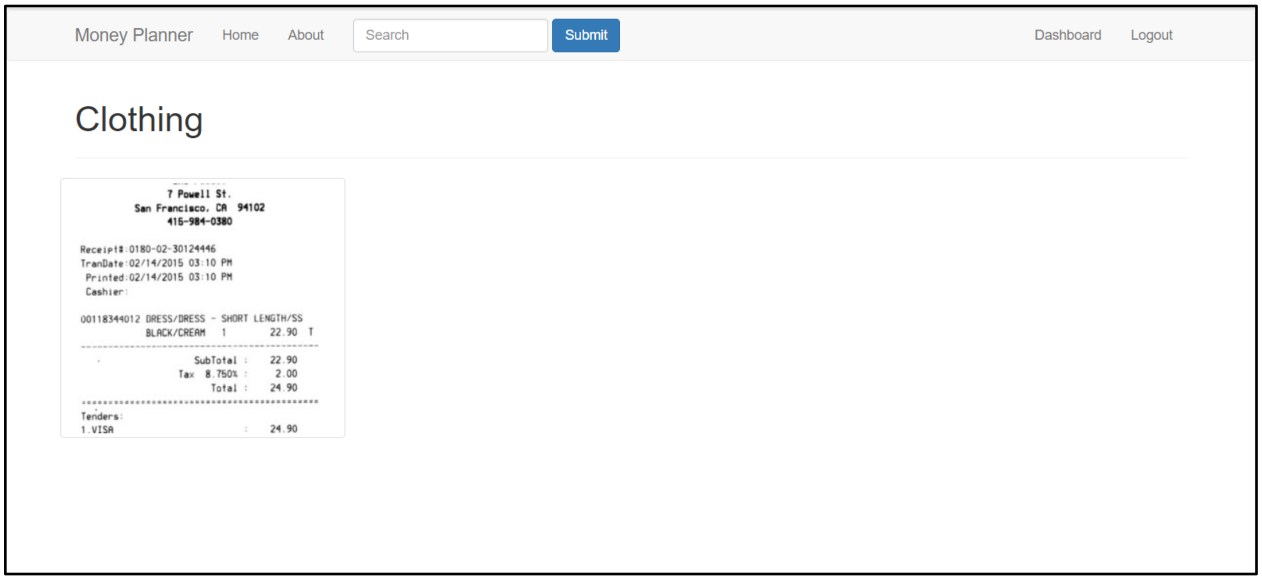


Budget Planner

Clothing	10
Retail	10
Restaurants	32
Miscellaneous	100

Figure 10: Update Budgets

9. The user can also search for particular category's receipts by using the search option on the navigation bar. The accepted search values are 'clothing', 'restaurant', 'retail' or 'misc'. The page will then display the thumbnails of all the receipts uploaded under that category. The user can click on each thumbnail to view the receipt in full size.



Clothing

```
7 Powell St.
San Francisco, CA  94102
415-984-0380

Receipt#:0180-02-30124446
TranDate:02/14/2015 03:10 PM
 Printed:02/14/2015 03:10 PM
 Cashier:

00118344012 DRESS/DRESS - SHORT LENGTH/SS
            BLACK/CREAM   1        22.90  T
---------------------------------------------
                      SubTotal :   22.90
                   Tax  8.750% :    2.00
                         Total :   24.90
=============================================
Tenders:
1.VISA                         :   24.90
```

Figure 10: Search Option

Design Topology

Front-end application

Our webpages are created using CSS and HTML. The software in the background is developed using Flask. AWS interface is though boto3 package. Once the application is completed locally we used Zappa to deploy the application to AWS Lambda.

We have a modular design with separate python files defining different web routes utilized in the application. Their descriptions are as below.

__init__.py : Instantiates the application object of Flask, imports required libraries.

images.py: This program handles all the functions required for image processing for the user. The functions within the program are as follows**:**

Function	Description
is_logged_in()	This wrapper function is used to prevent a user from accessing restricted pages such as Dashboard without logging in.
updateItem_Image()	The Image table is updated
updateItem_Image_Cat()	It updates Image table and places the image name in relevant category
getItem_Image()	Get image names for particular user
getItem_Image_Cat()	Gets image names for particular user for a specific category
save_file()	Saves the original uploaded and its thumbnail to S3
allowed_img()	After user uploads an image, this function checks if the uploaded image meets the allowable criteria for file extensions and image size
def add_photo()	The function to handle image upload by user
def view_photo()	To display the image in full size
read_receipt()	Reads the receipt data from the second S3 bucket and updates information in the

	database, the function returns the category of the receipt
putItem_Cost()	Updates the Cost table
search()	Provides search functionality
dashboard()	Creates the user dashboard

main.py: The main program is used to load the landing page of the application and also provides the link for the about page.

users.py: This program provides all the functions required to handle user requests for registration, login and logout.

Function	**Description**
putItem_User()	The User table is updated
putItem_FUL()	The FaceUnlock_Users table is updated
login()	Handles user login requests
face_unlock()	Handles user face unlock requests
face_match()	To check if there is a face match
image()	Gets image from webcam
image_register()	Get image from webcam when registering new user
face_register()	Handles user registration for face unlock
register()	Register user for unlock using username and password
logout()	Handles user logout request

Back-end functions

In the back-end we are using two Lambda functions:

1. **Converting image receipt to text:**

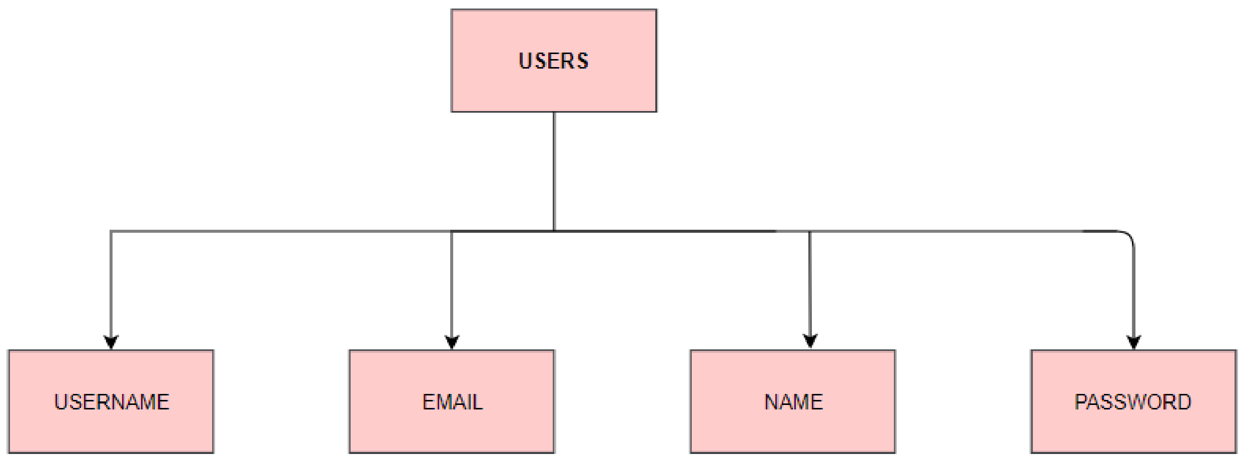
   This Lambda function is triggered by the first S3 bucket that stores the newly added receipts. Once a receipt is uploaded, the S3 bucket triggers the Lambda function. The function receives that image from the bucket and converts it to text using Textract. This converted text is now put in a .txt file in the second S3 bucket under the user's folder.

2. **Finding out category of receipt and total amount:**

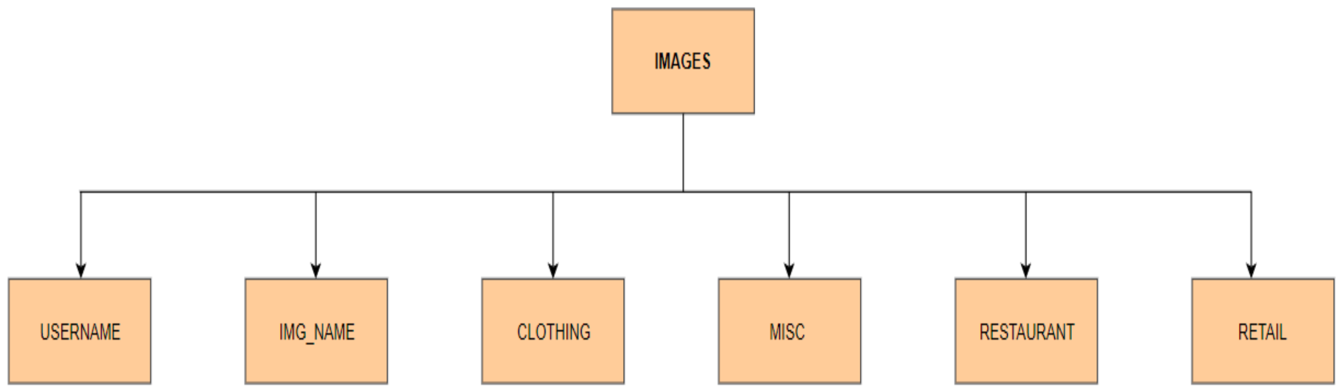
   This Lambda function is triggered by the second S3 bucket that stores the newly converted .txt file. Once a .txt file gets stored, the second S3 bucket triggers this Lambda function. The function opens the .txt file and finds out keywords from it. The stores that fall under clothing for example 'H&M', 'FOREVER21', etc are searched for and if any of these stores are found, it labels that receipt as 'clothing'. Similarly it checks for 'restaurant' and 'retail'. If the receipt doesn't fall under any of these categories, it is labeled as 'miscellaneous'. Next, the 'Total' or 'Amount' is extracted from the .txt file. The category and total amount details of that receipt are then output in another .txt file in the second S3 bucket. This information is used by the front-end application to update the user Dashboard charts.

Database Structure

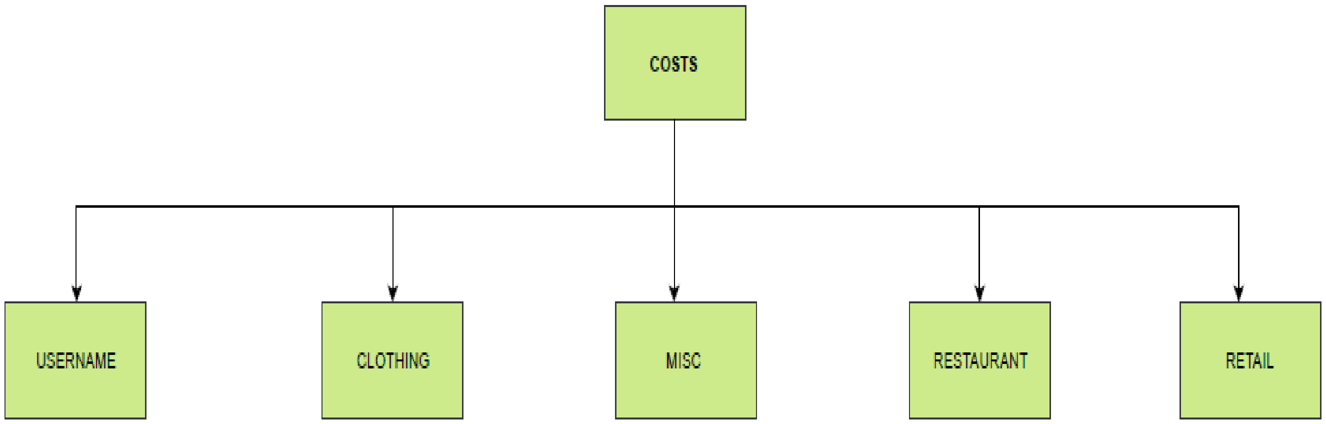
Users Table



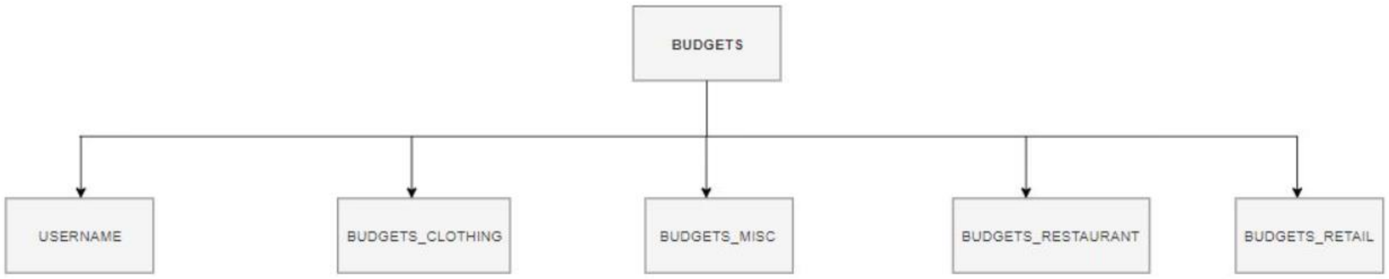
Images Table



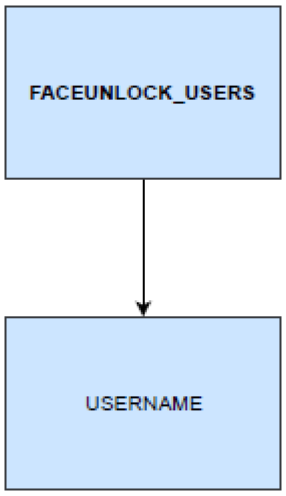
Costs Table



Budgets Table



FaceUnlock_Users Table



AWS Access Credentials

We did not use the AWS educate account for Assignment 3. Please find below the credentials to log in to the AWS account.

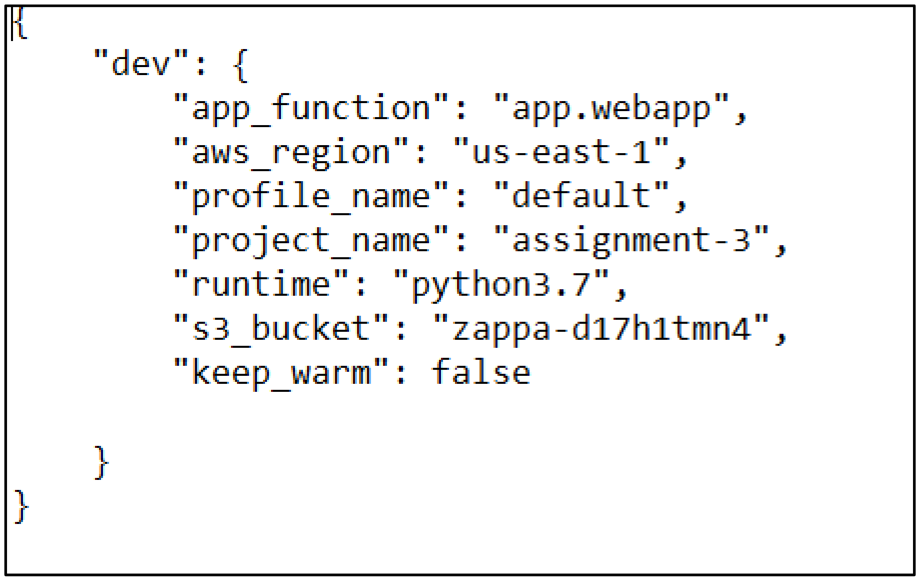
Access Link: https://846048079560.signin.aws.amazon.com/console

Username: ece1779

Password: ece1779

Configuration Settings

The following are the Zappa configurations used:



```
{
    "dev": {
        "app_function": "app.webapp",
        "aws_region": "us-east-1",
        "profile_name": "default",
        "project_name": "assignment-3",
        "runtime": "python3.7",
        "s3_bucket": "zappa-d17h1tmn4",
        "keep_warm": false

    }
}
```

Application File Structure

The following is the file structure for our web applications:

```
run.py
app
├── __init__.py
├── images.py
├── users.py
├── main.py
├── templates
│       ├── about.html
│       ├── add_photo.html
│       ├── budget.html
│       ├── clothing.html
│       ├── dashboard.html
│       ├── face_register.html
│       ├── face_unlock.html
│       ├── home.html
│       ├── layout.html
│       ├── login.html
│       ├── misc.html
│       ├── register.html
│       ├── restaurant.html
│       ├── retail.html
│       ├── view.html
│       └── includes
│              ├── _formhelpers.html
│              ├── _messages.html
│              └── _navbar.html
```

Cost Model for AWS

At the beginning, let's take a look at just 1 user and 1 request. For 1 user uploading 1 image (i.e. 1 request), the AWS services used are:

AWS SERVICE	REQUEST / PUT	GET
API Gateway	1	
DynamoDB	3	3
Rekognition	1	
Lambda	3	
S3	5	4
Textract	1	

NOTE: Everything mentioned below is on per month basis

The service charge of each AWS service is as follows:

S3

PUT	GET	Storage (upto 50TB)
$0.005/1000 requests	$0.004/1000 requests	$0.023/GB
First 2000 requests free	First 20000 requests free	First 5 GB free

Rekognition

Metadata Face Storage	Face Recognition (2 Images/request)
$0.00001/face	$0.001 per image
1000 faces metadata free	5000 images free

DynamoDB

READ	WRITE	Storage (upto 50TB)
$0.25/million reads	$1.25/million writes	$0.25/GB
25 unit-hr of Read free	25 unit-hr of Write free	First 25 GB free

Lambda

1M free requests per month and 400,000 GB-seconds. We used the Lambda pricing tool for our calculations.

Memory(MB)	Price per 100ms
128	$0.000000208
512	$0.000000833

API

Number of Requests (per month)	Price (per million)
First 300 million	$1.00
300+ million	$0.90

Textract

1 image is 1 page here

Monthly	Price (per million)	Effective Price per 1,000 pages
First 1 Million pages	$0.0015	$1.50

Over 1 Million pages	$0.0006	$0.60

Pricing of AWS services for one month

Assuming we have an average of 20 users for one month and assuming that they all login once each day and upload 1 receipt each day.

AWS Service	**20 users (per month costs)**
S3	0
Rekognition	0
DynamoDB	0
Lambda	0
API Gateway	0
Textract	0
Total	**$0**

Pricing of AWS services after six months

Using the cost information and the table above, the following cost calculations are made:

Assumptions:

1. The total is calculated for one month with the different number of users
2. 1 user logs in 1 time each day (month of 30 days is taken)
3. 1 user uploads 1 receipt image per day. Which means 1 request by 1 user each day.
4. For Lambda functions, the average duration elapsed in msec is taken.
5. For Rekognition, during login face analysis, we are using 2 images to compare with. Thus, for each login, the Rekognition takes in 2 images and analyzes them.
6. For API Gateway, we have assumed that for 1 user, there are 10 loads in 1 day. Thus in 1 month, for 1 user, there are 300 loads.

AWS Service	**10 users (per month costs)**	**1,000 users (per month costs)**	**1,000,000 users (per month costs)**

S3	0	PUT = 0.74$
		GET = 0.4$
		Storage (0.03 TB) = 0.575$
		
		Total = 1.715$
			PUT = 750$
			GET = 480$
			Storage (30 TB) = 690$
			
			Total = 1920$

Rekognition	0	metadata storage = 0$ Login face analysis = 50$ Total = 50$	metadata storage = 1$ Login face analysis = 2000$ Total = 2001$
DynamoDB	0	0	Read and write + under 25 GB storage = 135$ Total = 135$

Lambda	0	0	Zappa = 299.13$
			Lambda 1 = 174.13$
			Lambda 2 = 17.88$
			
			Total = 491.14$

API Gateway	0	0	Total = 300$
Textract	0	Receipts converted = 30000 (1000 will be free) Total = $43.5	Receipts converted = 30000000 (1000 will be free) Till 1 million = $1500 Above 1 million = $17,399.4 Total = $18899.4
Total	**0**	**$95.215**	**$23,746.54**

Future Work

Due to time constraints, there were some features that could not be implemented but are suitable for this web application:

1. Expanding the categories of expenses that a user can add receipts to.
2. Under each category, currently, only a certain number of stores are recognized to be in that category. These number of stores can be increased in order to make the application more general and useful.
3. Also, instead of the user providing expenditure limits, they can be deduced from the user’s previous expenditures, their income or spending habits.
4. Email notifications to the user if expenses in any category crosses the budget limit.