Project Requirements Document Version 2: CJ Affiliate

Project Title/Name: Intelligent Offer Categorizer
Team Name: Odyssey

Team Member Names	**Emails:**
● Xiao Sun	xiaosun@ucsb.edu
● Danny Millstein	millsteindanny@gmail.com
● Winfred Huang	whuang@ucsb.edu
● Haochen Shi	shi@ucsb.edu
● Delin Sun	delinsun@ucsb.edu

Team Lead: Xiao Sun
Team Scribe: Danny Millstein

Background:

CJ affiliate, as an affiliate marketing firm, facilitates advertisers in finding appropriate publishers to publish their offers. In the context of affiliate marketing, an advertiser is a company with some sort of product they are selling, and a publisher is a person with access to a medium (website, instagram page, youtube channel, etc.) on which they can display ads. CJ affiliate presents publishers with thousands of possible ad offers daily, and as a result has accumulated large databases full of data about these offers.

Problem:

The problem that we have been faced with is to develop a programmatic way to categorize all of this data, and to develop a way to display this data in a useful way to CJ affiliate employees and clients. To accomplish this we are creating a multi purpose web application that will primarily categorize the offer data accumulated by CJ affiliate, use this newly categorized data to create a repository of past offer performance, and lastly present this offer performance to CJ affiliates clients in an interactive format that will help them create optimal offers for their specific situation.

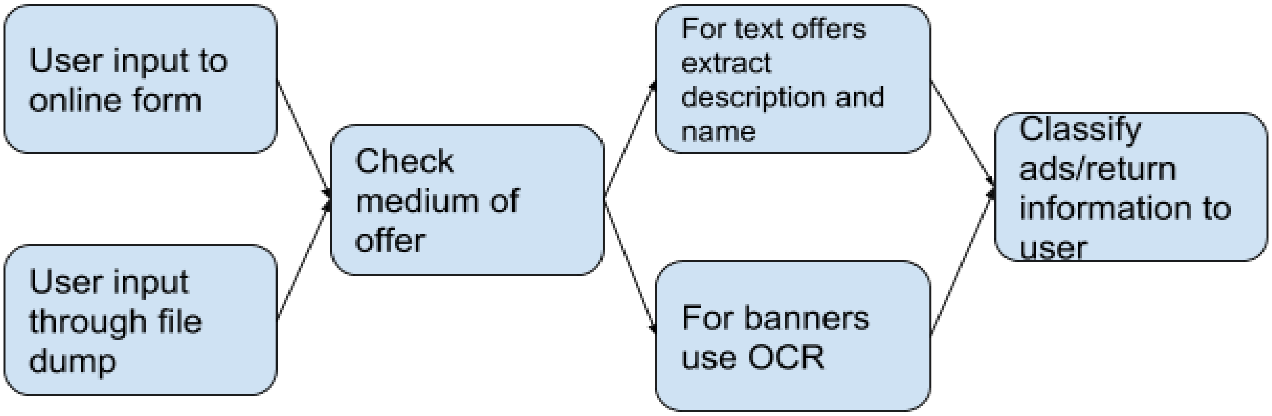
Project Specifics:

By categorizing this data CJ affiliate employees will gain the ability to compare performance data between different categories of offers (for example: BOGO, % off, $ off, $ off when you spend 100$,....). By employing data analytical tools to the newly categorized data CJ affiliate will have the ability to determine the best performing offers, with respect to certain mediums of publications and other relevant factors. For example, it is possible that advertisements on instagram produce more revenue during the holiday season, or % off advertisements on blogs are more successful than $ off advertisements, or possibly 5% off offers pay off bring in the same amount of traffic as 10% off offers. With this information CJ affiliate would be able to predict what kind of offers will do best with certain publishers or during certain times of year. Also this will facilitate CJ affiliate advertisers and publishers in choosing specific offers that will be most beneficial to them depending on the product, category of offer, medium of publisher, and other relevant offer/publisher attributes. Lastly this will give CJ affiliate solid numerical evidence to demonstrate to their clients, both publishers and advertisers, the value of their work.

Team Goals:

The Odyssey team has three main goals to accomplish in this project. First we want to create an automated tool that can categorize an offer, from the CJ affiliate database, based on the offer data. We are doing this with a three step process. We first check the medium of the offer, if the offer is text based we extract the offer description, and if it is image based we use optical character recognition to extract relevant text. We then input the extracted text into an offer categorizer algorithm, that we created, to determine the category of an offer. Lastly we append this category to the original offer data, and return this to the user. Once we are able to automatically categorize offer data we will use this newly categorized data to create a repository of past offer performance. We plan on doing this by storing the categorized data in a searchable data structure, so that we can parse out performance data for all offers based on a single general attribute or multiple more specific attributes. Since CJ affiliate has so much accumulated data we will be able to get a good idea of the difference in performance of offers that are only differing slightly in description. For example we could see if an offer for 10% actually performs better, in terms of revenue brought into the advertiser, than an offer for 5% off. Lastly we will present this offer performance to CJ affiliates clients in an interactive format that will help them create optimal offers for their specific situation. Our application will have the ability to compare previous offers from specific industries, with certain types of products, during certain times of the year, and with precise offers, so a client can recreate a situation similar to their own and see what offers in the past performed best.

High Level Diagram


User input to online form
User input through file dump
Check medium of offer
For text offers extract description and name
For banners use OCR
Classify ads/return information to user

User interaction and design:

User stories/use cases

Scenario 1: Processing text data from files

As a CJ data analyst, I can upload files with offer text data into the system so that the system can automatically categorize offers, which can save a lot of time.

Given that we have a file with offer text data, we get a file containing offer data plus additional category fields for each offer.

If the program successfully processed the file, there will be a pop-up window reminding the user that the data have been processed. The result will be stored into the database/system as a JSON file data.

Then the system will ask the user two options, “Do you want to stay at this page?” or “Go back to the main menu”.

Test: Upload several data files into the system to see if the result matches user’s requirement.

Scenario 2: Error from processing text data from files

As a CJ data analyst, I can upload files with offer text data into the system.

Given I have uploaded a file that is not containing the necessary information or is of the wrong format.

Then application will notify me that my file is of the wrong format.

Test: If any non JSON/BSON file is entered into the application, a pop up window will appear notifying the user.

Test: If any non JSON/BSON file is uploaded and the system does not pop up a window, the code should be modified.

Scenario 3: Creating single new offer from a single JSON file

As an individual advertiser user, I can create a new offer into CJ's system.

Given that the advertiser enters data into the system and they are creating a new offer in our system, the system will tag the offer based on input.

When they are trying to enter description and name, the system automatically tags the offer with the correct category.

Test: User clicks "create offer" button, the system gets into the "creating offer" page. The user can input offer information here.

Scenario 4: Creating several new offers from a single JSON file

As an advertiser or CJ employee, I can input a single file into the application and receive that file with a category of offer applied to it in the form of an additional json variable.

Given the file inputted was in the desired BSON/JSON format and contains enough information to generate a category.

If the file contains several offers in Json format, the system read these offers.

Test: Input a file with a known categorization, and compare the systems output to what was expected.

Scenario 5: Creating several new offers from multiple JSON Files

As an advertiser or CJ employee, I can input multiple files into the application and receive similar files with a category of offer applied to each in the form of an additional json variable.

Given the files inputted were in the desired BSON/JSON format and contained enough information to generate categories.

Test: Input multiple files with known categorizations, and compare the systems output to what was expected.

Scenario 6: Categorize the text file after uploading

As a CJ employee, I can use get the uploaded Json file to be automatically categorized.

Given the files are successfully uploaded by scenario 3,4,5, the system read files based on Java Json libraries. It will take each offer as an object, extract information from the object by using JSON parser from Json library.

Test: If the uploaded file only contains one offer, the output of the categorizer should be keyword of that offer.

Test: If the uploaded file contains several offers, the output of the categorizer will the Json file that contains several categories. Offers that have same category will be put together.

Test: If the file does not contain any offers, the output should be null.

Scenario 7: Uploading images

As a consumer, I can upload an image to the system. This helps the user from having to type what was in the image.

Given that the algorithm has the image, then the algorithm (using OCR) will extract the text from the image. The image can be anything as long as there is text on it. For example, there is an image with text “20% off with minimum $75 purchase”. Then the result should be “20% off with minimum $75 purchase”.

Test: If there is text and there are offer keywords (like “% off” or “minimum purchase”), then the result should match the text found on the image.

Test: If there is text but no offer keywords found, then the result should be shown and notify user that there is no offer on image.

Test: If there is no text, then the result should tell user that there is no text on image.

Test: For every test above, if the text extracted appears to be incorrect, then the user will correct it manually.

Scenario 8: Categorizing Uploaded Images

As a consumer, I would like to have uploaded images to have a tag to link to an existing offer in text file or to see the performance of that specific image.

Given that the image is already uploaded, then I can “tag” the image with a specific category so I can possibly track it or link it to an offer found in the text file.

Test: Append a “tag” to the image. If successful, then the tag should be visible when image is present.

Test: If “tag” can’t be appended, then the image will not have the “tag” and user is notified of that problem.

Scenario 9: Converting the Banner Ads into Binary or HTML

As a CJ employee, I want the banner to be binary since it will be much easier to read in Java.lang and Json.jar. If we want to use the images in the future, we can just type the binary\HTML, then the images will show up.

Given that the Banner Ads are uploaded into the system, the system should convert an image binary.

The code should embed images as they are into HTML. The HTML will also be stored in the output JSON file.

Scenario 10: Importing Images

As a CJ analyst, I can import an image which is uploaded into the system.

Given images are uploaded into the system, the user wants to import them to see the images. The images will be stored in JSON file in HTML format. After entering the HTML into the web page. The expected output is to have the image shown up on the web page.

If I can not get the image, which means that the HTML is wrong or something else is wrong, this Ad will be labeled “cannot open” so that CJ employee can modify the Ad.

Scenario 11: Mixed medium input

As a CJ employee I can upload a file containing text offers and banner offers to the web application, so that I do not need to specify the type of offer, and I am still able to get my offers categorized correctly.

Test: Populate a json file with offer objects with known categories of medium 1. Input the file into the webapp. The app should output a new JSON of the input that includes categories.

Test: Populate a json file with offer objects with known categories of medium 3. Input the file into the webapp. The app should output a new JSON of the input that includes categories.

Test: Populate a json file with offer objects with known categories of medium 3 and 1. Input the file into the webapp. The app should output a new JSON of the input that includes categories.

Scenario 12: Automatically Categorize Offers after Uploading and Processing Data

As an advertiser I can enter the name and description of an offer, and in real time see the application label my offer with one to multiple categories.

Given the information inputted is sufficient enough to generate a category for this offer.

Test: Input a description and name that we know the corresponding appropriate category for, and in real time compare the output of the application with the expected output.

Scenario 13: Search the Existing Ads

As an user, I can search the existed ads by Ad ID.

Given Ads are uploaded into the system, users can extract the information from the website. There is a button on the web named “search”. The blank box will pop up after the user click that button. User can enter Ad ID, then the information of that Ad will show up.

Furthermore, there will be several categories on the top of the website. User can choose one or more than one categories, then ads which satisfy these categories will remain on the website.

If I can not view the filtered offers, then I will receive a message and remain in the same list of unfiltered offers.

Scenario 14: Edit Existing Ads

As an user, I can edit the specified offer to make some changes.

Given that a file is already uploaded and categorized, user first want to search the file from scenario 12. The Ad's information should contain name, description, Ad ID, categories, etc. The user can edit the offer description and save it. The system will store the information and re-categorize the offer like scenario 3,4,5.

Test: If I can not edit, then the offer file will remain unchanged.

Scenario 15: Add new Categories

As a CJ analyst, there will be a situation that the existing categories cannot satisfy the updating Json requirement. For example, Sephora had 25% off; however, they changed the offer this year to be 25% off if purchase is more than $75. So the system needs a new category.

Given that the output categories do not satisfy user's requirement. I can directly type the description to update the existing offer with a correct category of the offer. If I can not update, then the existing offer will not be altered.

Scenario 16: Create Online Offers

In this scenario, as the advertiser or the analyst of CJ, I am able to access to CJ's website. Then I can access the "Create Offer" page. The form with some blanks will show up. I can type the "name", "description", "medium", and "Ad ID". After I click the "Create" button, the website will gather the information and create the offer automatically for me. The categorizer will also categorize the offer and show the performance of similar past offers, as it works in scenario 9.

If the system cannot categorize the description of this new offer, the return value will be null and the user can use method in scenario 14 to create new categories.

Scenario 17: **Upload files on the Web**

As an analyst, I want the web to be more convenient that I can directly drop files to the web.

Given the analyst open the Capstone UI system based on visual stack, there will be a drop box asking the user to drop the JSON file into the box. If I drop the file into the box, the system will automatically upload the JSON file, read the file, categorize it, and write another JSON file containing the categorized information for me.

Scenario 18: Create New JSON File Based on Online Offers

As a CJ analyst, I want the put one or more results of creating online offers (scenario 16) into a new json file.

Given in scenario 16, we entered some data, and it has been categorized, so now we're going to put that data into a json file. Thus, in the future, we can use it anytime.

Test: Use the Unit test for reading JSON to check if the new JSON file output matches the expected result.

Scenario 19: Split Categorized Json File

In this scenario, I want split the json file (result of scenario 18) to many individual json files. Each new JSON file contains offers with same category.

Given in scenario 18, I get a json file that include all categories, to facilitate a more user friendly experience, split it to small json file only have one category.

Test: Use the Java Unit test to test the output. Check if each JSON file contains same category.

Scenario 20: Show overall performance of category

In this scenario, as a consumer of this product, I am able to pick out a category so I can see the overall performance of that specific category like a BOGO offer as an example.

Given that the consumer has data stored in the database full of sorted and categorized offers, the consumer should be able to view some, if not all, of the offers.

Test: When the consumer chooses a specific offer, whether it's BOGO or some percent off, the expected output should be a summarized performance of that offer. That summary output can have elements like monthly average earnings, which month had the most profit, or which month had the least profit just to name a few.

Test: If the output can not produced like there is not enough data to produce yearly earnings, then the program will produce a pop-up error dialog with specific notes to alert the consumer about how the earnings were calculated.

Test: Otherwise for other errors like the computation took too long, then the pop-up dialog will show why and return back to the “home” page.

Scenario 21: Categorize data using the web application

As a CJ analyst, I can log onto the offer categorizer web page using my login credentials. I can then navigate to the file offer categorizer page. On this page I can drag and drop multiple files into the file dropzone, and press the categorize offer button to categorize all of the files I inputted. Lastly I can press the download button, and receive a set of JSON files of categorized data.

Given the inputted files are correctly formatted JSON files.

Test: We input to the website correctly formatted JSON files, of which we know their proper category, and compare the downloadable output to the expected output.

Scenario 22: Input a new offer using the web application

As a CJ client, I can log onto the offer categorizer web page using my login credentials. I can then navigate to the offer form page. On this page I can fill out the form with appropriate answers, and as a result recieve an interactable graph that displays performance information of similar past offers.

Given the inputted answers contain offer relevant information.

Scenario 23: Adjust similar offer performance graph to include more parameters

As a CJ client looking at my new offers projected performance graph (graph of similar past offer performance) I can add new or more restrictive parameters to compare and contrast how slightly changing my offer could benefit me.

Given the inputted answers contain offer relevant information, and that a valid graph of similar past offer performance has been generated already.

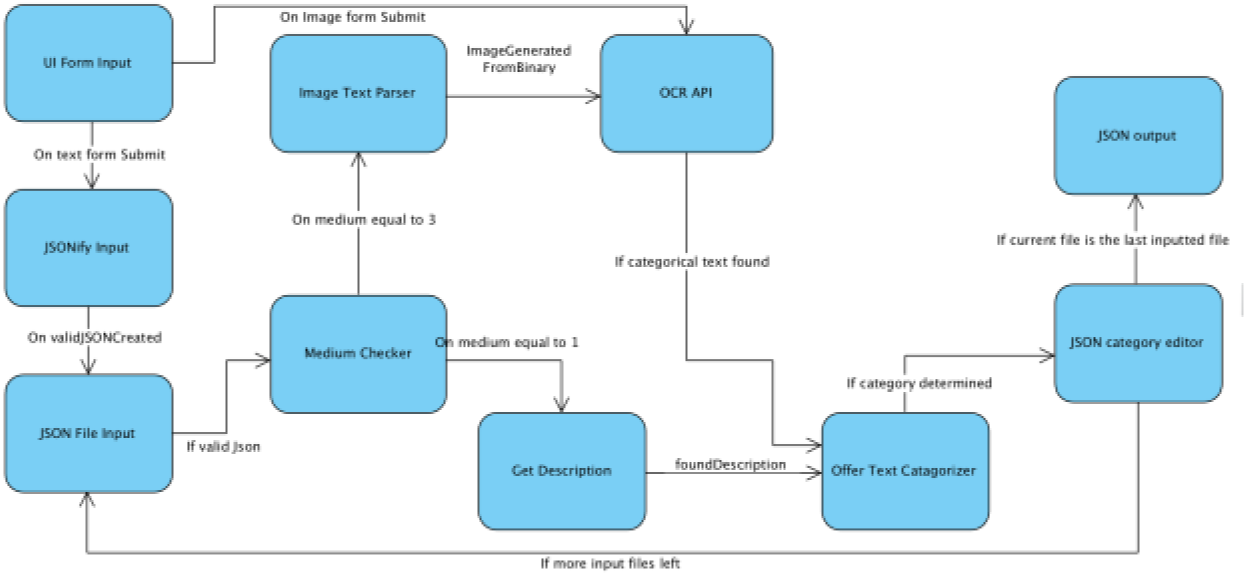
Prototyping code, tests, metrics:

Master Branch: **https://github.com/cjdev/capstone2018/tree/master**

UI Branch:**https://github.com/cjdev/capstone2018/tree/UI**

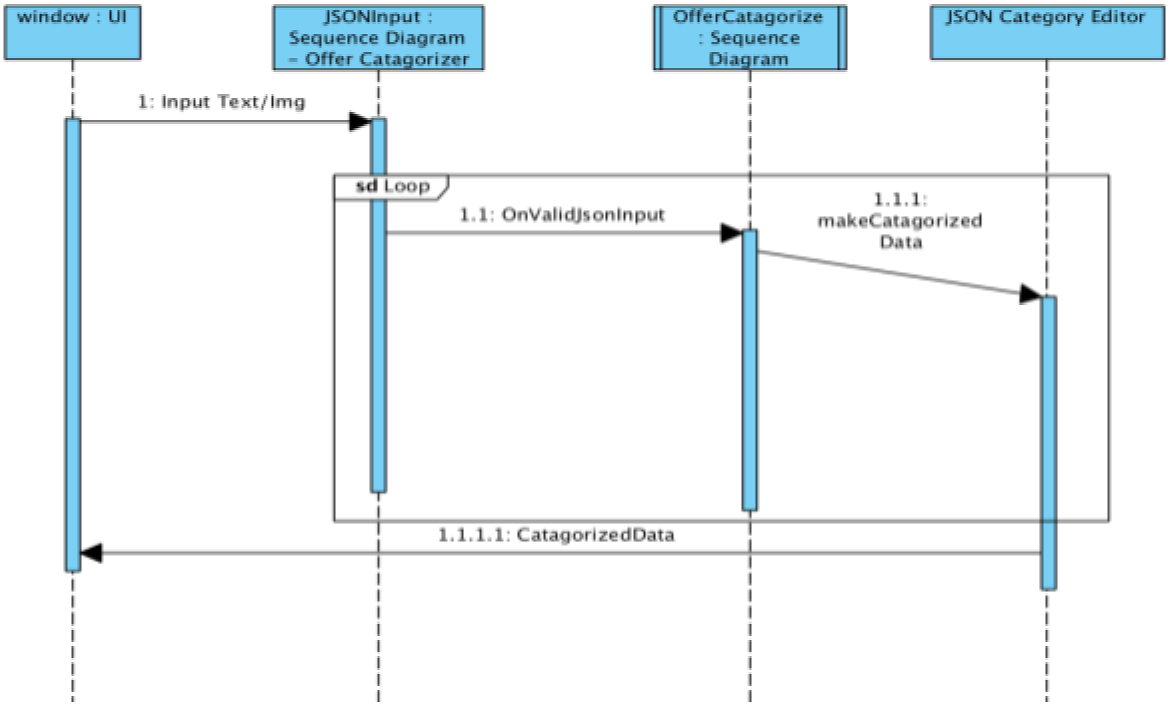
Backend Branch:**https://github.com/cjdev/capstone2018/tree/mavenize**

State Diagram

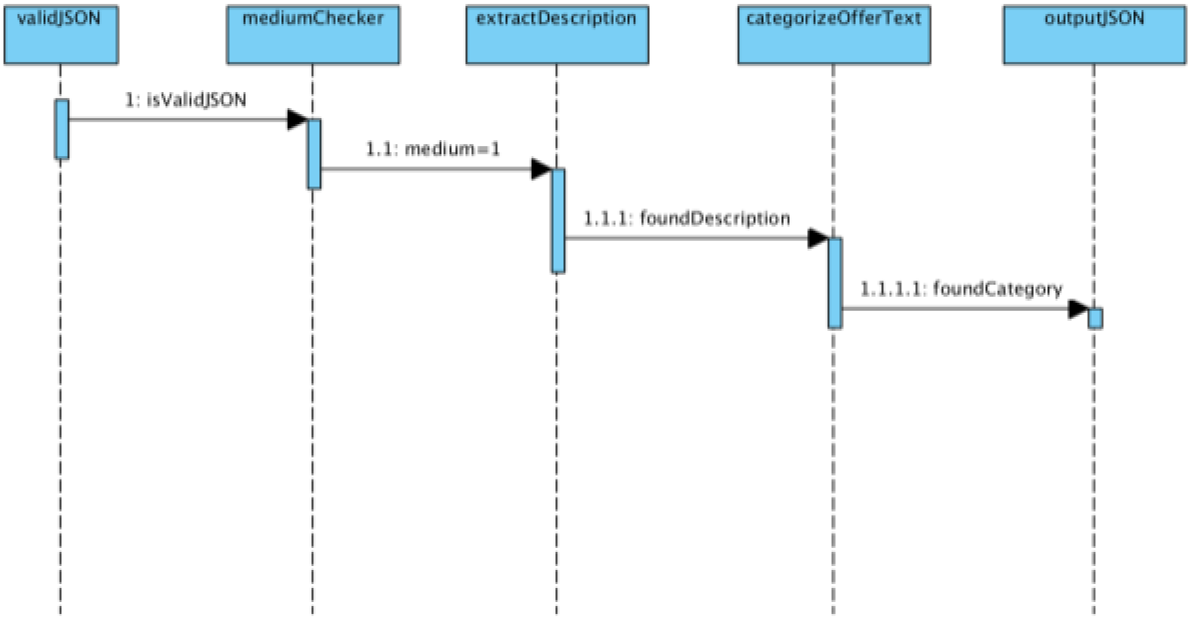


Sequencing, event response, classes/objects:

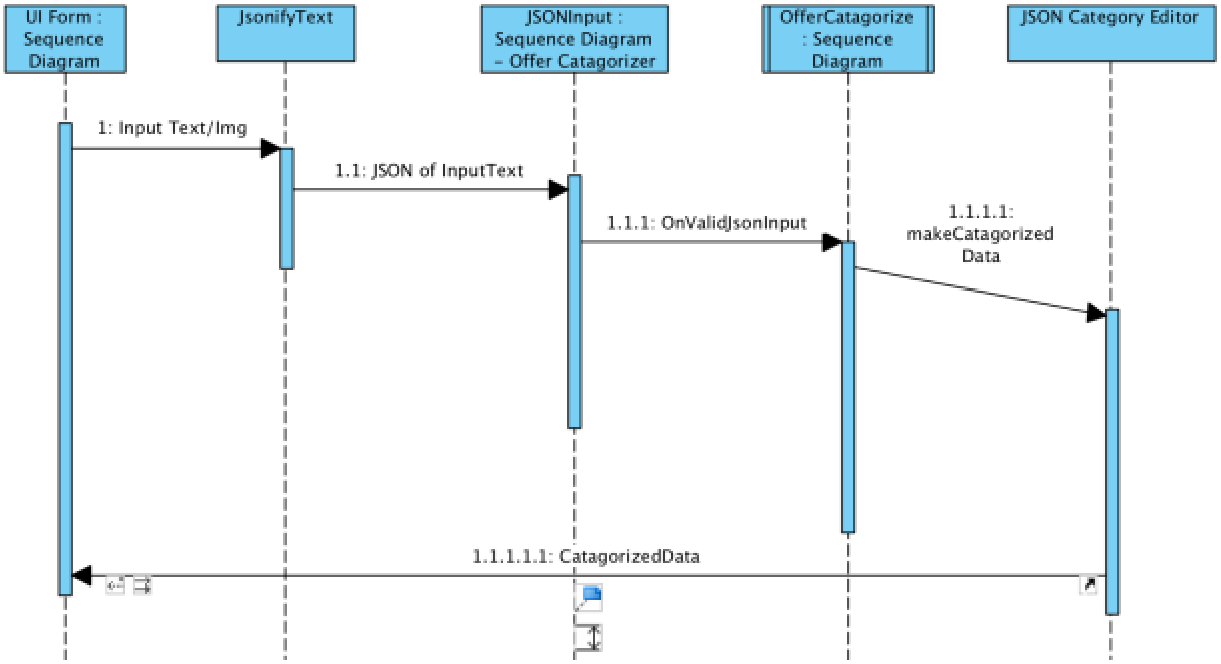
JSON File Input Sequence Diagram


window : UI
JSONInput : Sequence Diagram - Offer Catagorizer
OfferCatagorize : Sequence Diagram
JSON Category Editor
1: Input Text/Img
sd Loop
1.1: OnValidJsonInput
1.1.1: makeCatagorized Data
1.1.1.1: CatagorizedData

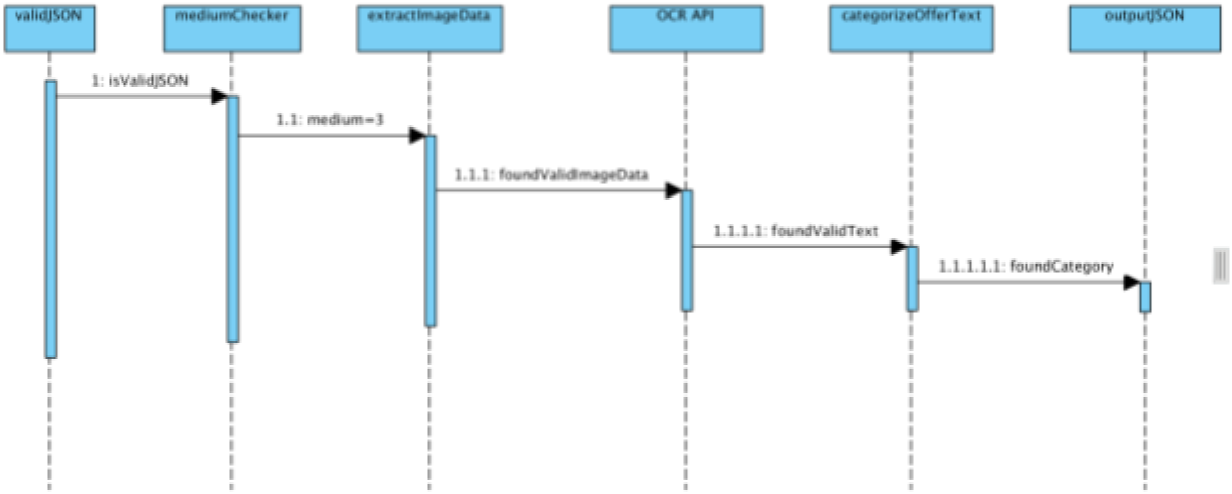
Algorithm sequence on text based offer



Form Input Sequence Diagram

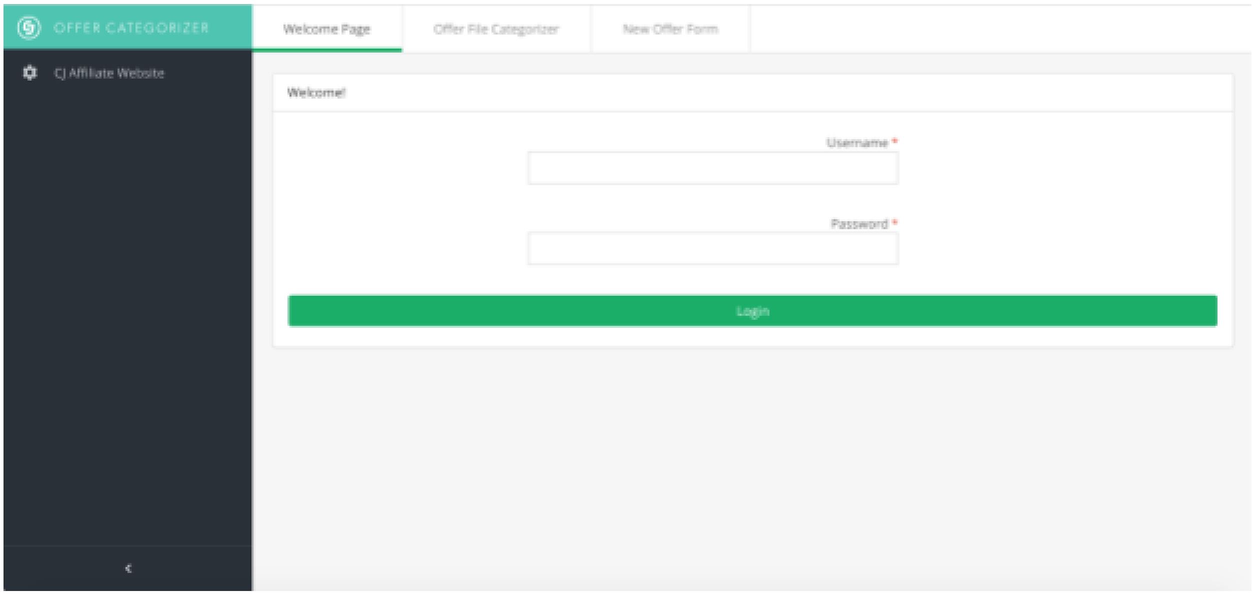

UI Form : Sequence Diagram
JsonifyText
JSONInput : Sequence Diagram - Offer Catagorizer
OfferCatagorize : Sequence Diagram
JSON Category Editor
1: Input Text/Img
1.1: JSON of InputText
1.1.1: OnValidJsonInput
1.1.1.1: makeCatagorized Data
1.1.1.1.1: CatagorizedData

Algorithm sequence on image based offer

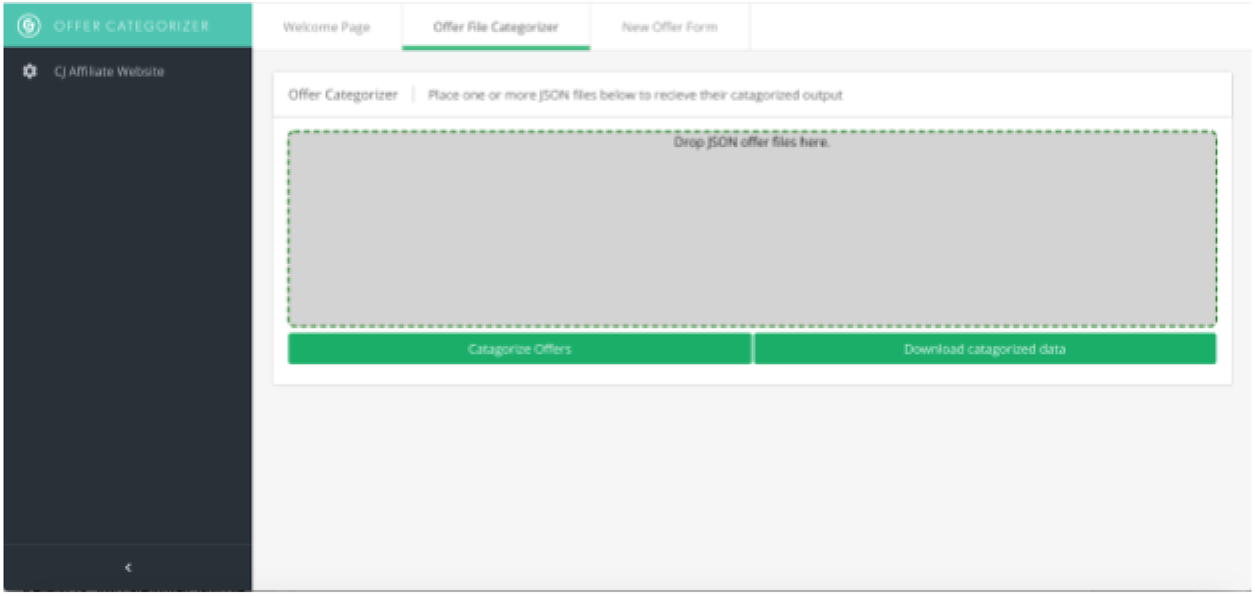


UI and Design:

Welcome Page



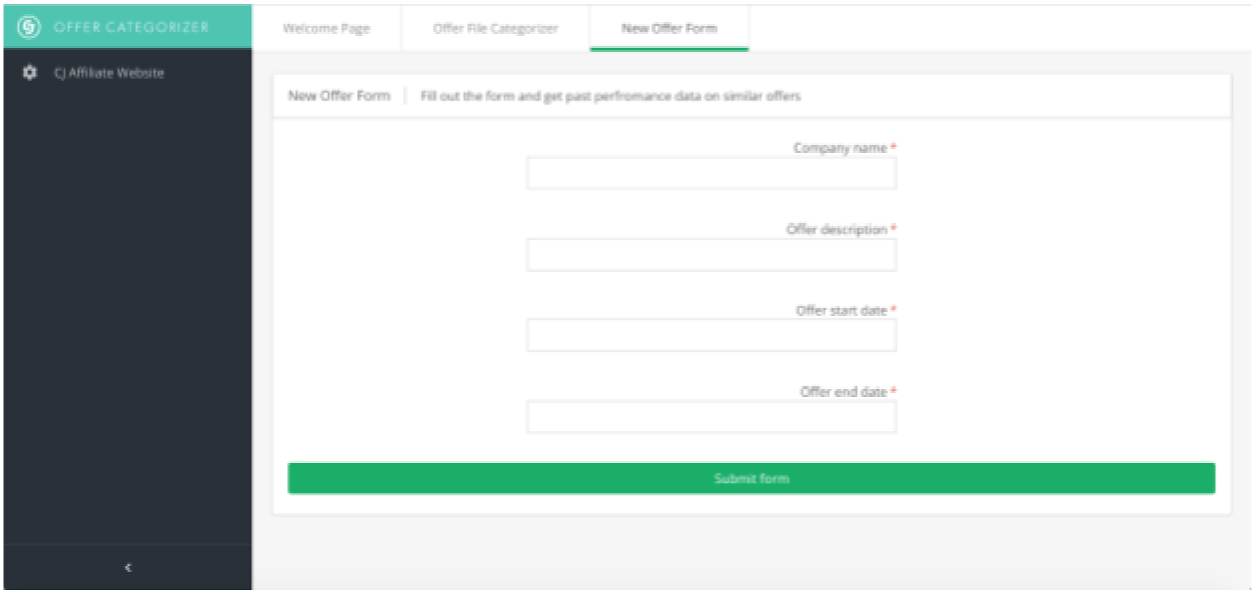
Offer File Categorizer Page



Offer Categorizer | Place one or more JSON files below to recieve their catagorized output

Drop JSON offer files here.

New Offer Form Page



New Offer Form | Fill out the form and get past perfromance data on similar offers

Company name *

Offer description *

Offer start date *

Offer end date *

Technologies employed:

- ReactJS
  - CJ Visual Stack (ReactJS component library)
  - React Dropzone
  - React Download Link
  - superagent

- JAVA
  - JSON Simple Java Library
  - Junit Java Library
  - Javax JSON Java Library
  - Java Tesseract API from net.sourceforge.tess4j (for OCR)