# Introduction

We are doing a Google Cloud POC for Image import and text extraction.

1. Scan documents and extract pertinent information and document types
2. Find out what documents are missing and if contents are complete and any discrepancies.
3. Extracted information and type compared to trained / learning ML models.
4. Side by side evaluation between scanned and extracted information.
5. Train the ML with a sample set and use the ML against subsequent cases.

# Overall Architecture

1. UI is angularjs, bootstrap and HTML5
2. We are using Jersey to expose REST services.
3. Database will be postgresql.
4. We will use google cloud APIs – Vision API, ML API and Natural Language API
5. We will use google cloud engine to deploy the code and run it.

# Overall flow of work

## Upload workflow

1. User clicks on a link, for example
   1. [www.importbillshsbc.com/uploadDoc](http://www.importbillshsbc.com/uploadDoc)
2. The screen has an interface where the user can upload one file or multiple files at one time. Allowed are image formats (jpg, gif)
3. The uploaded files will be processed by the FileUploadService class.
4. For each file, this class will save the file into a local directory.
5. Then the file is uploaded into a google storage bucket.
6. Then we call the Google Vision API passing the location of the storage bucket.
7. Google Vision API will scan the image and send back a JSON response.
8. We will store this response along with image name and metadata (filesize, filetype, document category, timestamp) into the database.
9. The JSON response will be sent back to the UI.

## Modelling workflow

1. The vision API response is evaluated and the main entity models are extracted.
2. For example, for document type: Invoice, the entity models would be:
   1. Shipper Id
   2. Seller Id
   3. Country of origin
   4. Carton Count
   5. Gross Weight
   6. Net weight
   7. Packer Info
   8. Grower Info
   9. Manufacturer Info
   10. Commodity Description
3. Each entity model would have 4 features – start\_x, end\_x, start\_y, end\_y



STARTX – Left top pixel

ENDX – Right top pixel

STARTY – Left bottom pixel

ENDY – Right bottom pixel

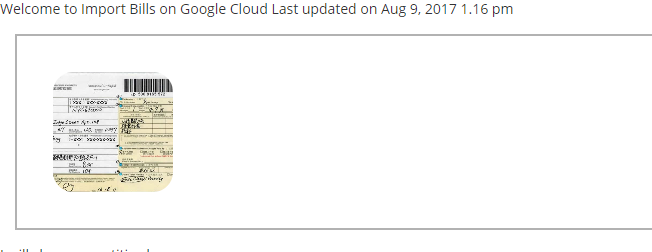
1. We store a number for every feature. This is the pixel position.
2. How we detect entity model as present is by
   1. Looking for some text within a range of pixels
   2. Doing some string pattern matches to ensure that the string makes sense for that feature (weight is a number, packer info is a string etc).
   3. We can also whitelist values for a feature to check for specific set of values.

The training dataset should contain atleast 100 combinations from existing document types. Then we apply this model to every new document that is imported and the created model is tested against existing models. End result is a list of missing entity models for each document type.

## Train Model workflow

1. The entity model information gets returned to the user for every document that is uploaded.
2. This is displayed in a format where the expected model is displayed on the left and the actual model is displayed on the right and the list of features are presented to the user to check.
3. User can choose to add new features and this information is persisted into the model.

# Screenshot 1 – Upload Screens



Use Cases to be developed