SASTRA DEEMED TO BE UNIVERSITY

SCHOOL OF COMPUTING

B.TECH COMPUTER SCIENCE & ENGINEERING

Software Engineering Project

Question Paper Analyser

Aravind Srinivasan (121003029)

Ashwin Ram S (121003035)

Contents

List of Figures

List of Tables

0.1 Problem Statement

Bloom's taxonomy is a set of three hierarchical models used to classify educational learning objectives into levels of complexity and specificity. The three lists cover the learning objectives in cognitive, affective and sensory domains. The cognitive domain list has been the primary focus of most traditional education and is frequently used to structure curriculum learning objectives, assessments and activities.

The Bloom's Taxonomy helps us distinguish between Higher Order Thinking Skills (HOTS) and Lower order Thinking Skills(LOTS).This helps in forming strategies / exercises / tools for delivering the content. Similarly while assessment is done one must be clear on what level of knowledge we want to assess and accordingly questions may be set.

High School and University question papers are standardized with Bloom's Taxonomy. But often there are a lots of human errors and also it is very time consuming process to validate every single question.On a University or Institutional scale this becomes a very hectic task.

Our goal is to automate the validation process and smoothen the experience for teachers and professors. Finalising a question paper is much easier now as our software automates the whole process.

We use Machine learning models to classify the questions according to Bloom's Taxonomy standards.

0.2 Aim and Objectives

0.2.1 Aim

The aim of this project is to automate the current process of validating Question Papers. Blooms Taxonomy is a standard used by several institutions to regulate the quality of question papers.

To acheive this we intend to build machine learning models to classify questions into the 6 categories accurately

0.2.2 Objectives

The objectives of this Software Eng. Projects are as follows:

- Evaluate the standards of questions by teachers
- Figure out the total percentage of each type questions according to Bloom's Taxonomy
- Figure out what categories a question belongs to

0.3 Existing Solutions

Currently there are no Full Fledged software for handling this task, although people currently use Microsoft Excel or any other spreadsheet software to handle creation and labelling of questions.

Although there are many Research Papers published on the same topic. Some of them are:

- Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.
- Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.
- Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.
- Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

0.4 Limitations of Exisiting Solutions

There are no existing softwares for the Problem Statement. But there are several ways in which it is better than the conventional or normal usage.

Most of the Universities and Institutions, use a word processing software or Spreadsheet software for maintaining the list of questions. Our Software is better than the conventional ways because

- Can analyse which category the question falls in on the fly.
- Calculate an overall score for the Question paper's standard of difficulty
- Privacy and Security. We don't store any user data, therefore the question papers analysed are as safe as possible.
- Ease of Use
- Accurate classification with the help of machine learning models

0.4.1 Limitations of already used algorithms

The Machine Learning model we built uses XGBoost algorithm.

It is an implementation of gradient boosting machines created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library.

Tianqi Chen provides a brief and interesting back story on the creation of XGBoost in the post Story and Lessons Behind the Evolution of XGBoost.

XGBoost is a software library that you can download and install on your machine, then access from a variety of interfaces. Specifically, XGBoost supports the following main interfaces:

- Command Line Interface (CLI).
- C++ (the language in which the library is written).
- Python interface as well as a model in scikit-learn.
- R interface as well as a model in the caret package.
- Java and JVM languages like Scala and platforms like Hadoop.

Another famous algorithm in implemetation is SVM (Support Vector Machine)

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

0.4.2 Comparison between XGBoost and SVM algorithm

SVM	1.0	0.46	0.53	0.57	0.70	0.89
XGBoost	0.87	0.61	0.81	0.6	0.71	0.65

Table 1: Comparison between SVM and XGB

The above Table 1 describes the Accuracy of prediction for each category of questions
From the above table we can infer that XGBoost algorithm is cleary more accurate for many of the categories

0.5 Methodology

0.5.1 Modules

This Project consists of 4 modules.
They are :

- OCR
- Machine Learning Model
- Question Paper
- Blooms Analyser

Machine Learning Model:
Machine Learning Module is responsible for the prediction logic of the application. This model has been built with XGBoost algorithm and implemented the application layer using Python3

OCR:
OCR (Optical Character Recognition) is responsible for the conversion of images into text with the help of tesseract library

Question Paper Module:
This module is responsible for handling all the parsing and manipulation of the Question Paper object.

Also it is responsible for parsing the given image to text with the help of OCR module. This whole module has been implemented with the help of FLASK framework with Python3

Bloom's Analyser Application:
This module acts as the GUI interface between the application and its users. A user can upload images of question papers to get them analysed.

The whole front-end application has been implemented with the help of React.Js library with Javascript.

0.5.2 Functionalities

Machine Learning Module

Labeling questions based on Bloom's Taxonomy using machine learning techniques Machine learning algorithms can be broadly split into either supervised or unsupervised training implementations. Generally, supervised training is adopted when, during training, labels have been pre-determined and questions are labeled by an expert.

The most commonly used method in such cases is the term frequency- inverse document frequency (TF-IDF). The algorithm assigns weightages to individual words in a question statement to define a custom vector space to each question.

Remember
Consider a signal described by y[n] = 2n +4. What would be the amplitude of the signal at sample index n=3?
Apply
Consider the following input and output signals: find the transfer function and state the poles and zeros of this transfer function.
Transfer
Describe how the bandpass filter can be utilized for radar applications.

Figure 1: Example of Questions for Bloom's Taxonomy

Machine learning techniques such k-nearest neighbors, Naïve Bayes and support vector machine (SVM) have been implemented for labeling questions. When doing a performance comparison among these three techniques, an F1 measure of 0.71 was achieved using SVM [20]. To increase the accuracy level, additional features were incorporated in future versions of the work.

Three different feature selection processes, namely: Odd Ratio, Chi-square statistic and Mutual Information were used with the three machine learning techniques. The F1 measure result reached 0.9 [21]. Furthermore, an integrated approach of feature extraction has been proposed by using headword, semantic, keyword and syntactic extractions, which are fed into SVM [22]. However, this work has not yet been completed by using a testing dataset to quantify the reliability of prediction.

Word (alphabetical order)	Weightage
begin	0.392
for	0.140
given	0.140
one	0.222
side	0.356
signal	0.116
the	0.007
unilateral	0.392
when	0.279
which	0.230
ztransform	0.216

Figure 2: Word weights in Alphabetical order

A major downside in existing works is that both the training as well as testing questions are part of the same course curriculum; the questions are generated by the same author/instructor. Even when a high F1 measure is achieved,

it does not enable the algorithm to label questions written by another subject matter expert. Our work increases the flexibility of labeling methods by testing our models with a new set of questions compiled from textbook and online resources.

In addition, our work introduces extreme learning machine (ELM), which has been shown to outperform SVM during similar labeling tasks [23]. Moreover, we introduce LDA as an alternative technique to TF-IDF for transforming question statements into numerical word weightages.

Learning outcome	Frequency (number of questions)
Remember	62
Apply	131
Transfer	23

Figure 3: Learning Outcome vs Frequency

By comparing combinations of these new techniques with more traditional techniques, we aim to gauge which combination attains the highest labeling reliability with the subject matter expert when automatically labeling untrained questions. For our purposes, using the combination with the highest F1 measure (fewest false negatives and false positives) becomes paramount. In our use case, a mislabeling by the algorithm will lead to the wrong set of practice questions to be given to students and diminish the impact of deliberate practice on reaching the intended learning outcomes.

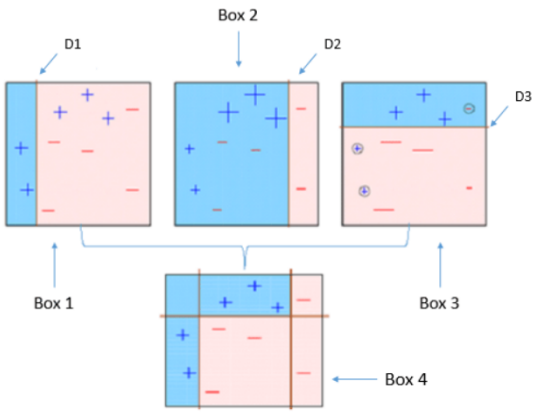


Figure 4: XGBoost Model Training

Question Dataset

The dataset consists of a total of 150 questions used for training and testing the machine learning algorithms based on the content of an undergraduate electrical and electronic engineering course. For this study, we formed a training set of 120 questions by randomly selecting 40 Remember, Apply, and Transfer items from the larger question pool of more than 200 questions used in that course. The pool came from a repository of four years' worth of assignment, homework, quiz and exam questions presented to students. These questions prompt students for a range of answer types (i.e., open-ended, multiple-choice, short-structured, essay). We then created a testing set of 30 new questions compiled from external sources such as textbooks and online question banks. This set was also balanced with equal representation of Remember, Apply, and Transfer questions.

Question Paper Module

Backend Application is made using FLASK Framework This Applications Functionalities are as follows:

- Serve different HTTP requests
- Generate Text from Image using Tesseract library for OCR (Optical Character recognition)
- Interface with Machine Learning Model to predict / classify questions generated by OCR

Blooms Analyser Application Module

This application is made using React JS library This Applications Functionalities are as follows:

- Interface between the bussiness logic and User
- Acts as UI for the application
- Upload images as input to the backend server
- Generate reports based on response from Backend

0.6 Design Engineering

0.6.1 UML Diagrams

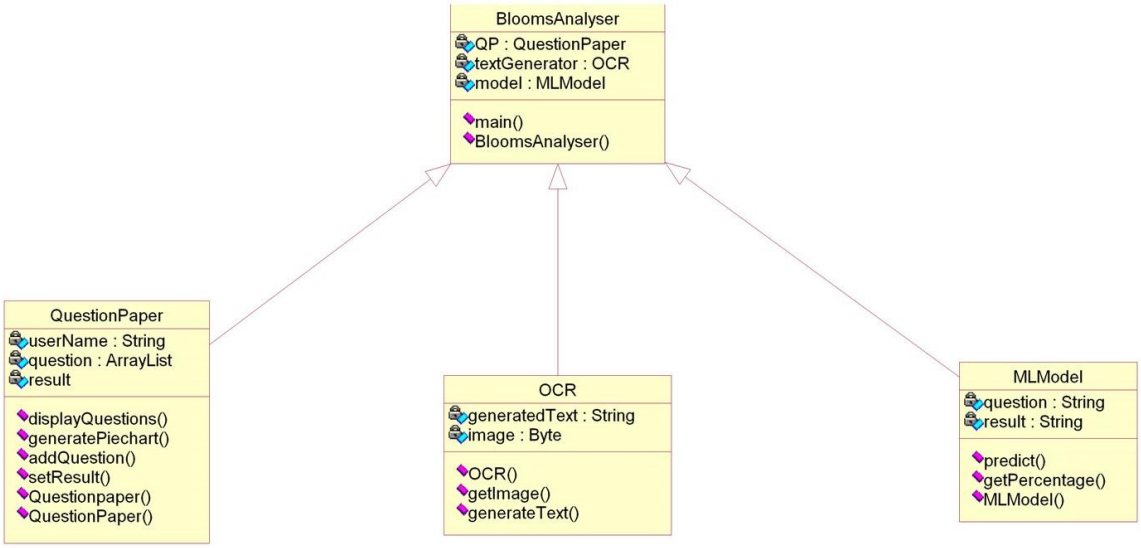


Figure 5: UML class diagram

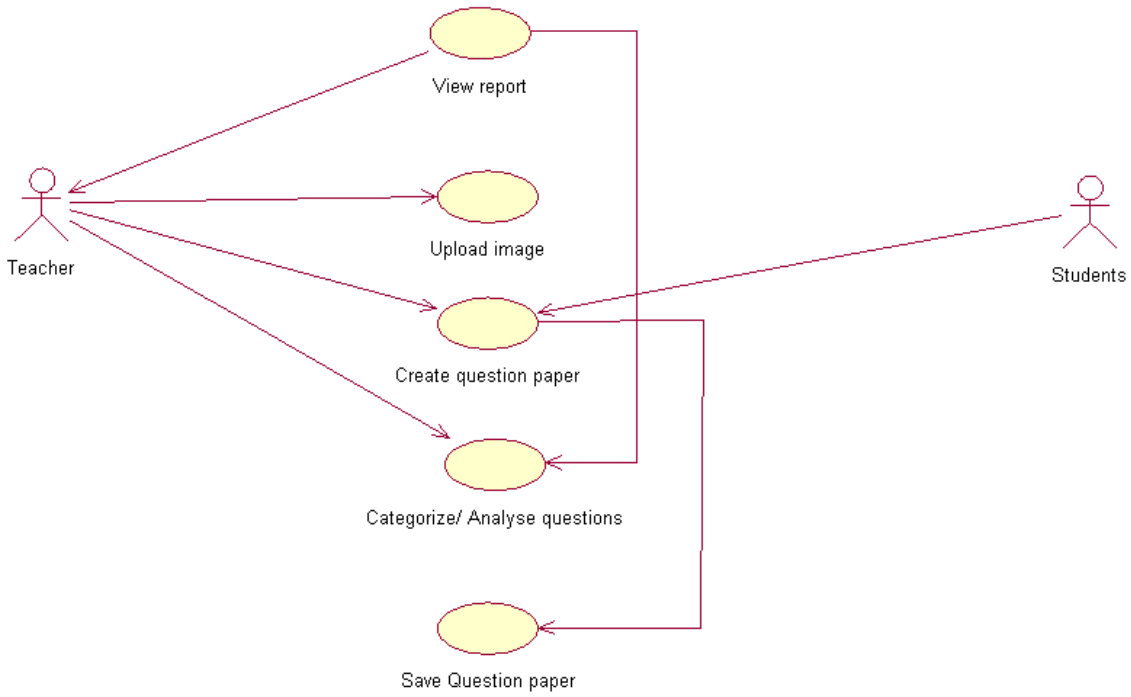


Figure 6: Use case diagram

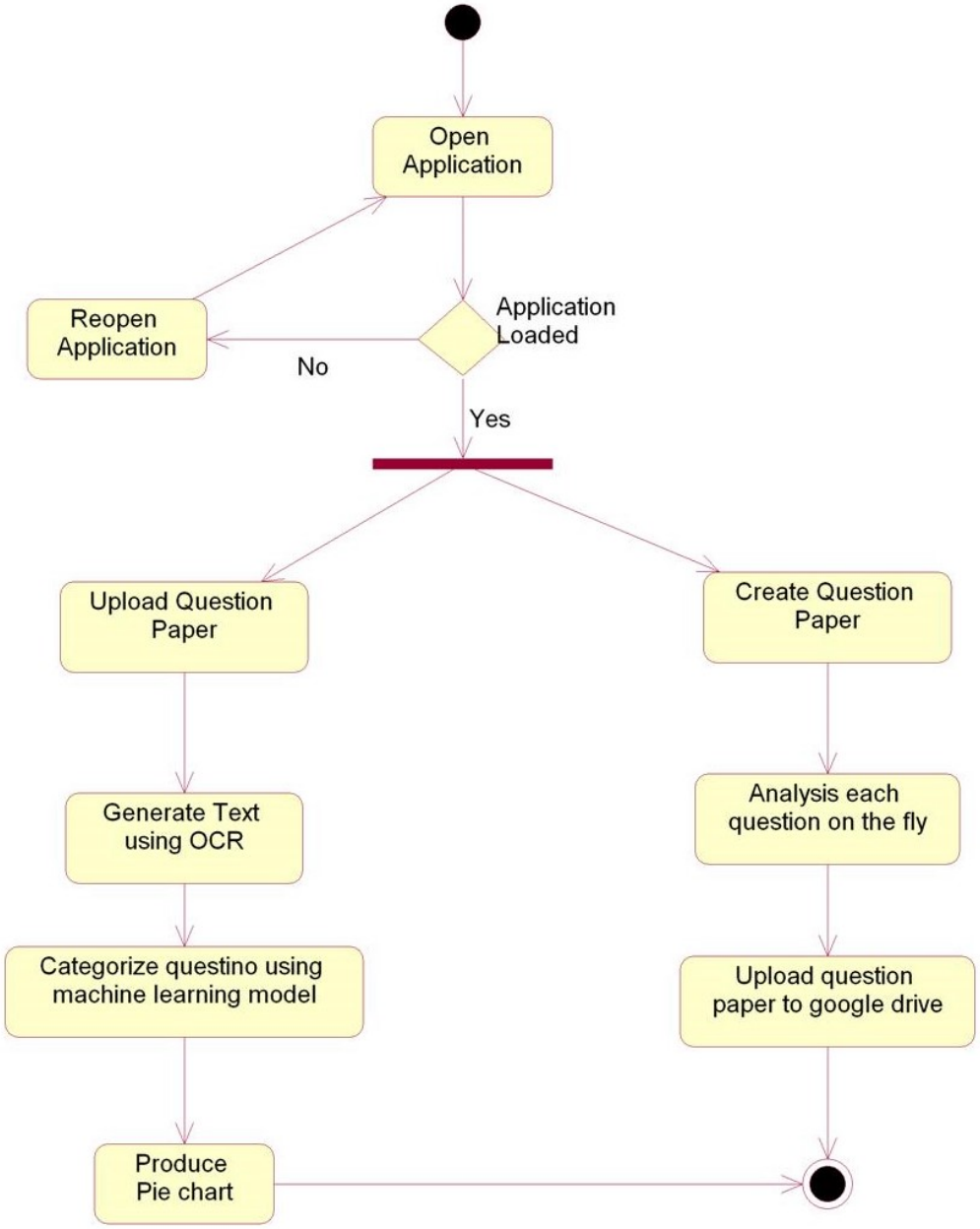


Figure 7: Activity diagram

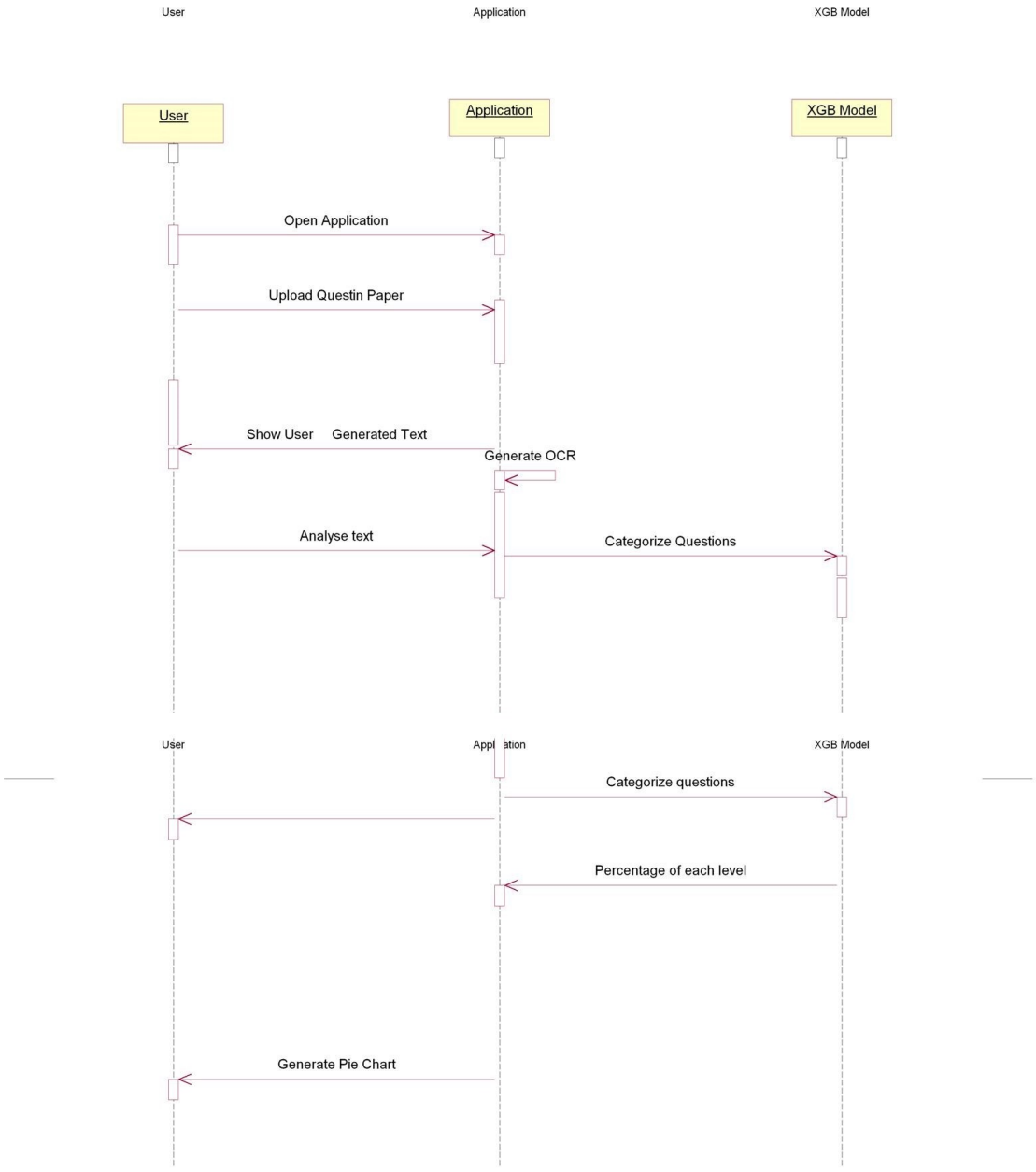


Figure 8: Sequence diagram

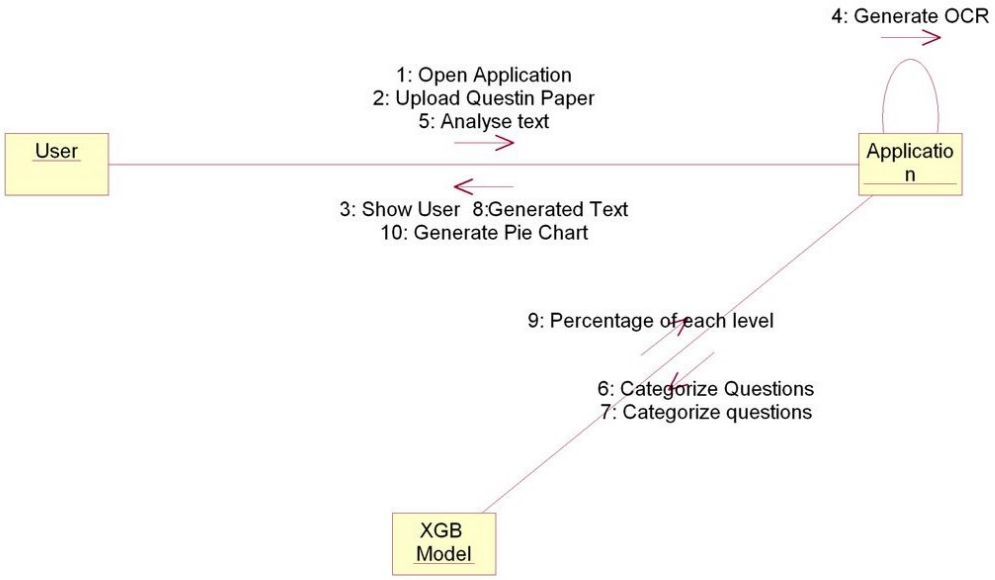


Figure 9: Collaboration diagram

0.6.2 Data Flow Diagrams

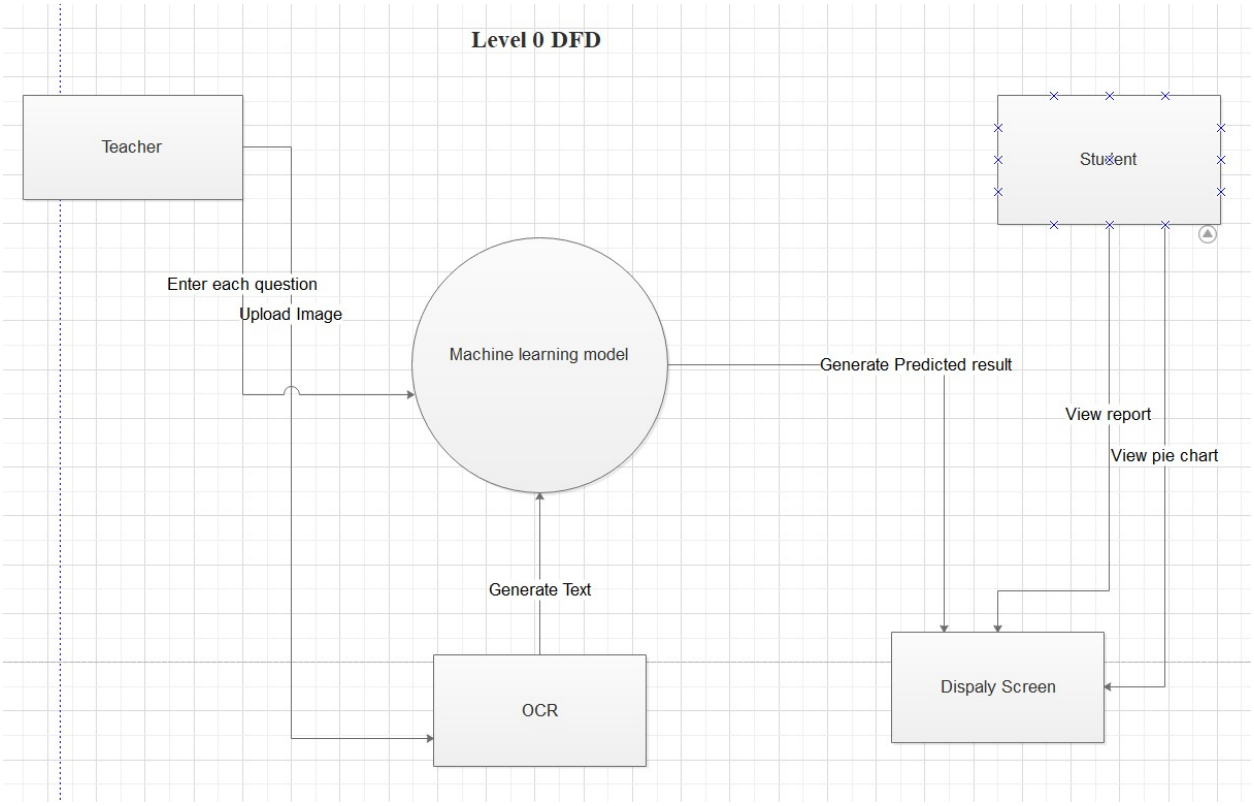


Figure 10: Level 0 Data Flow Diagram

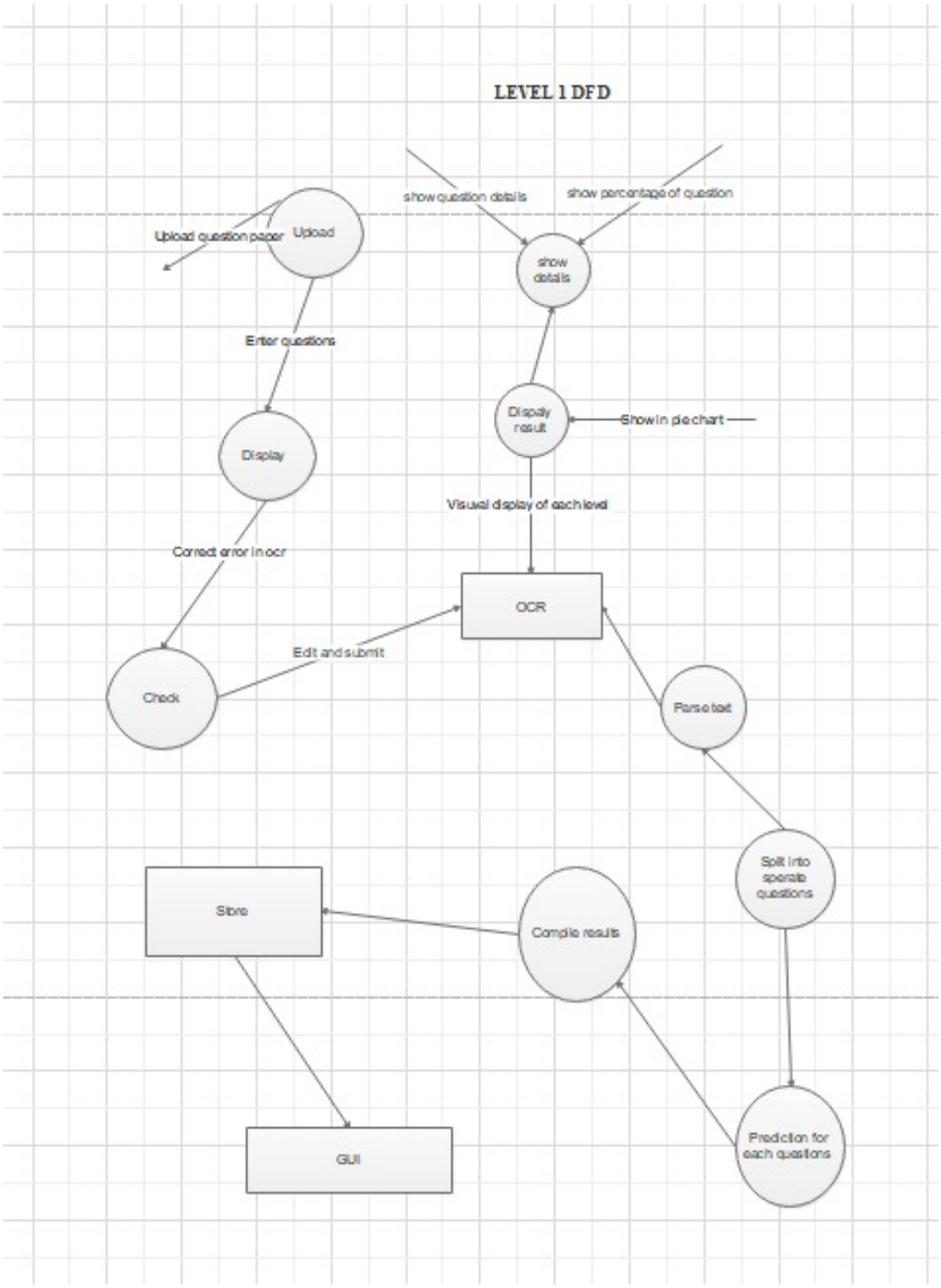


Figure 11: Level 1 Data Flow Diagram

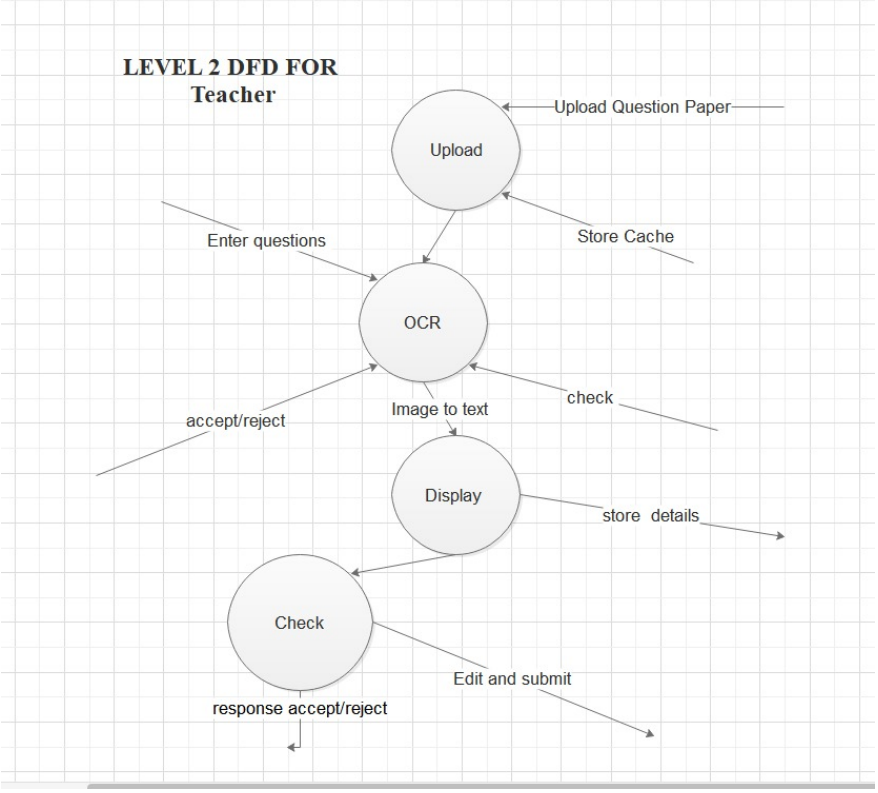


Figure 12: Level 2 Data Flow Diagram for Teachers

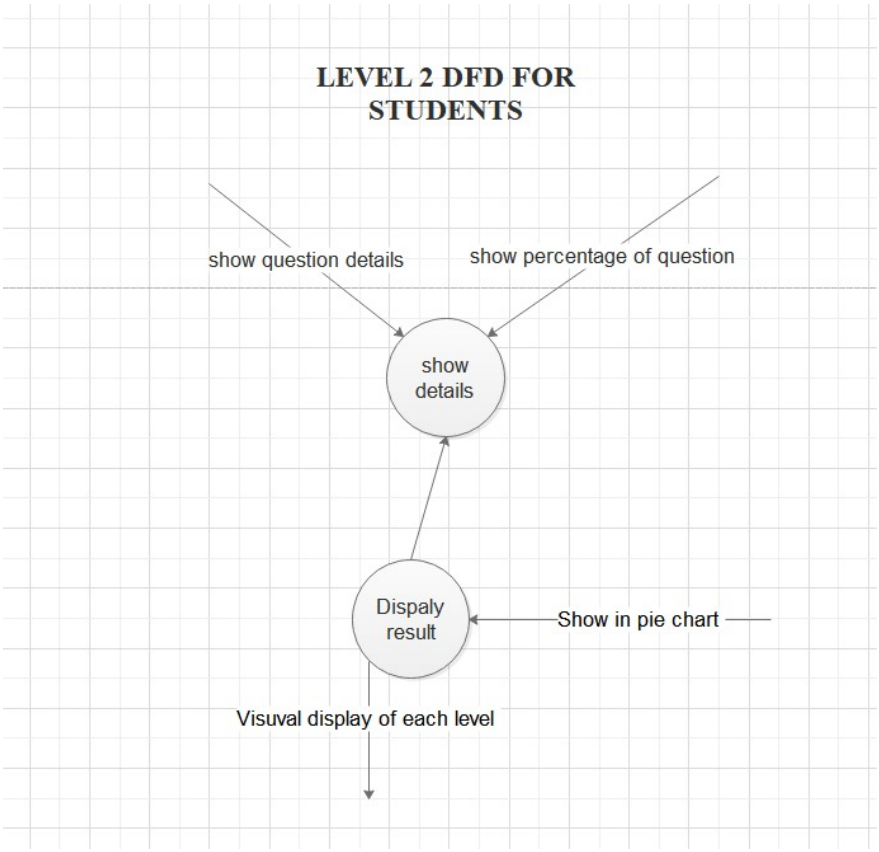


Figure 13: Level 2 Data Flow Diagram for Students

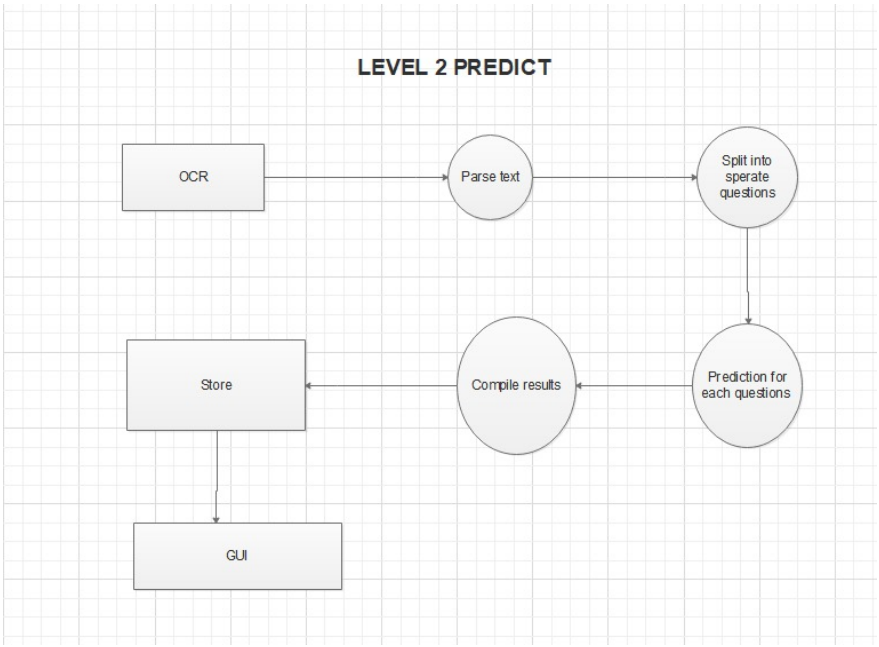


Figure 14: Level 2 Data Flow Diagram for ML Model

0.7 Development

0.7.1 Implementation

Hardware Requirements

- Processor - Pentium / Core i3
- Speed - 1.1 Ghz
- Hard Disk - 200GB
- Standard I/O (Keyboard and Mouse)

Software Requirements

- Operating System - Windows / Linux / MacOs
- Compilers - Python3, Python2 , NodeJs
- Libraries - SciKit Learn, FLASK, React.js
- IDE - Visual Studio Code
- Web server - Apache TomCat

0.7.2 Forward & Reverse Engineering

Forward Engineering

```
//Source file: C:\\Program Files\\Rational\\RUPBuilder\\OCR.java


public class OCR extends BloomsAnalyser
{
   private String generatedText;
   private Byte image;

   /**
   @roseuid 5EA0708D039D
    */
   public OCR()
   {

   }

   /**
   @roseuid 5EA070910096
    */
   public void getImage()
   {

   }

   /**
   @roseuid 5EA070950254
    */
   public void generateText()
   {

   }
}
```

```
//Source file: C:\\Program Files\\Rational\\RUPBuilder\\QuestionPaper.java

import java.util.ArrayList;

public class QuestionPaper extends BloomsAnalyser
{
   private String userName;
   private ArrayList question;
   private int result;

   /**
   @roseuid 5EA075540201
    */
   public QuestionPaper()
   {

   }

   /**
   @roseuid 5EA0717F039B
    */
   public void displayQuestions()
   {

   }

   /**
   @roseuid 5EA071880344
    */
   public void generatePiechart()
   {
```

```
32
33     }
34
35     /**
36     @roseuid 5EA07191036F
37      */
38     public void addQuestion()
39     {
40
41     }
42
43     /**
44     @roseuid 5EA0719701DD
45      */
46     public void setResult()
47     {
48
49     }
50
51     /**
52     @roseuid 5EA0719D03A9
53      */
54     public void Questionpaper()
55     {
56
57     }
58 }
```

```
1  //Source file: C:\\Program Files\\Rational\\RUPBuilder\\MLModel.java
2
3
4  public class MLModel extends BloomsAnalyser
5  {
6     private String question;
7     private String result;
8
9     /**
10    @roseuid 5EA0715702C2
11     */
12    public MLModel()
13    {
14
15    }
16
17    /**
18    @return java.lang.String
19    @roseuid 5EA0714C0059
20     */
21    public String predict()
22    {
23     return null;
24    }
25
26    /**
27    @return java.lang.Double
28    @roseuid 5EA07153012B
29     */
30    public Double getPercentage()
31    {
32     return null;
33    }
34 }
```

```
1  //Source file: C:\\Program Files\\Rational\\RUPBuilder\\BloomsAnalyser.java
2
3
4  public class BloomsAnalyser
5  {
```

```
    private QuestionPaper QP;
    private OCR textGenerator;
    private MLModel model;

    /**
    @roseuid 5EA07059017C
     */
    public BloomsAnalyser()
    {

    }

    /**
    @roseuid 5EA0704E0389
     */
    public void main()
    {

    }
}
```

0.7.3 Reverse Engineering

Class Generated from Code

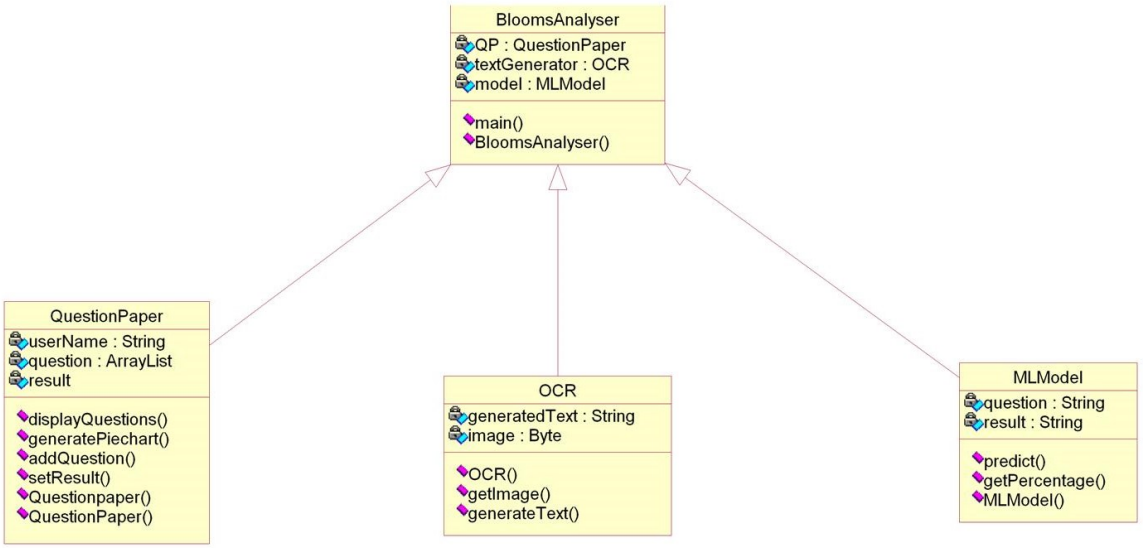
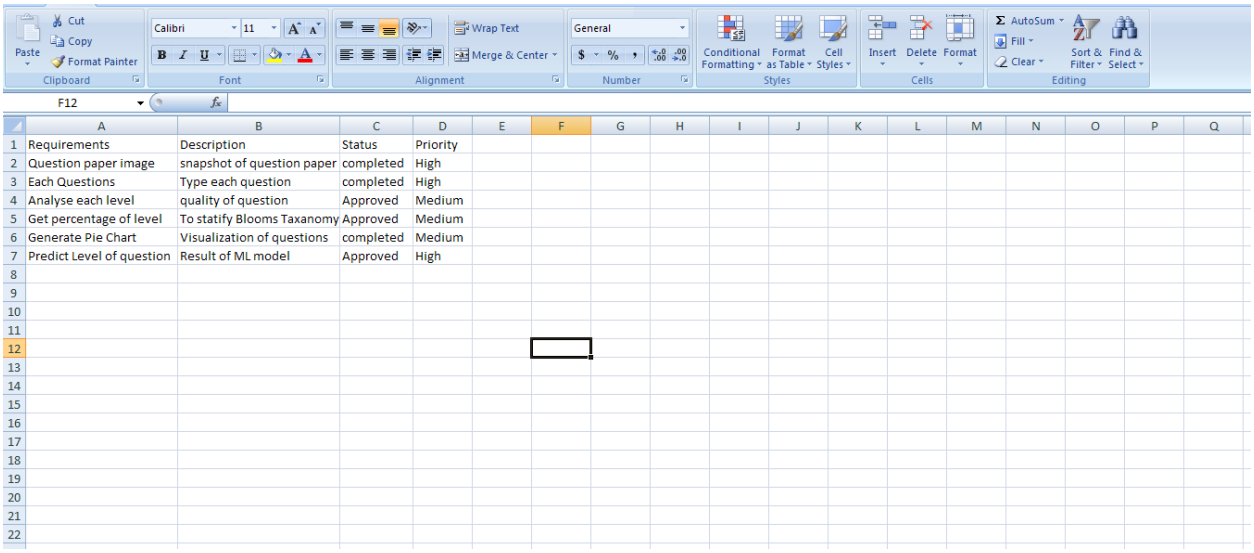


Figure 15: UML class diagram

0.8 Testing

0.8.1 Manual Test Script



	A	B	C	D
1	Requirements	Description	Status	Priority
2	Question paper image	snapshot of question paper	completed	High
3	Each Questions	Type each question	completed	High
4	Analyse each level	quality of question	Approved	Medium
5	Get percentage of level	To statify Blooms Taxanomy	Approved	Medium
6	Generate Pie Chart	Visualization of questions	completed	Medium
7	Predict Level of question	Result of ML model	Approved	High

Figure 16: Manual Test Script in Spreadsheet

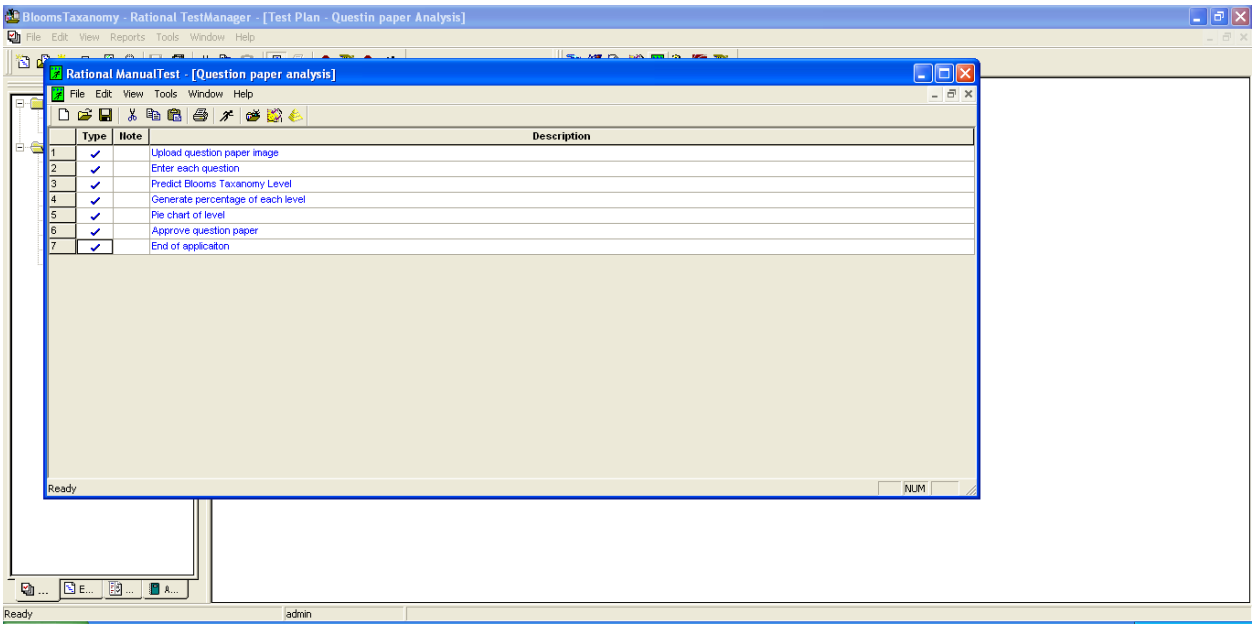


Figure 17: Test Script

0.8.2 Test Results

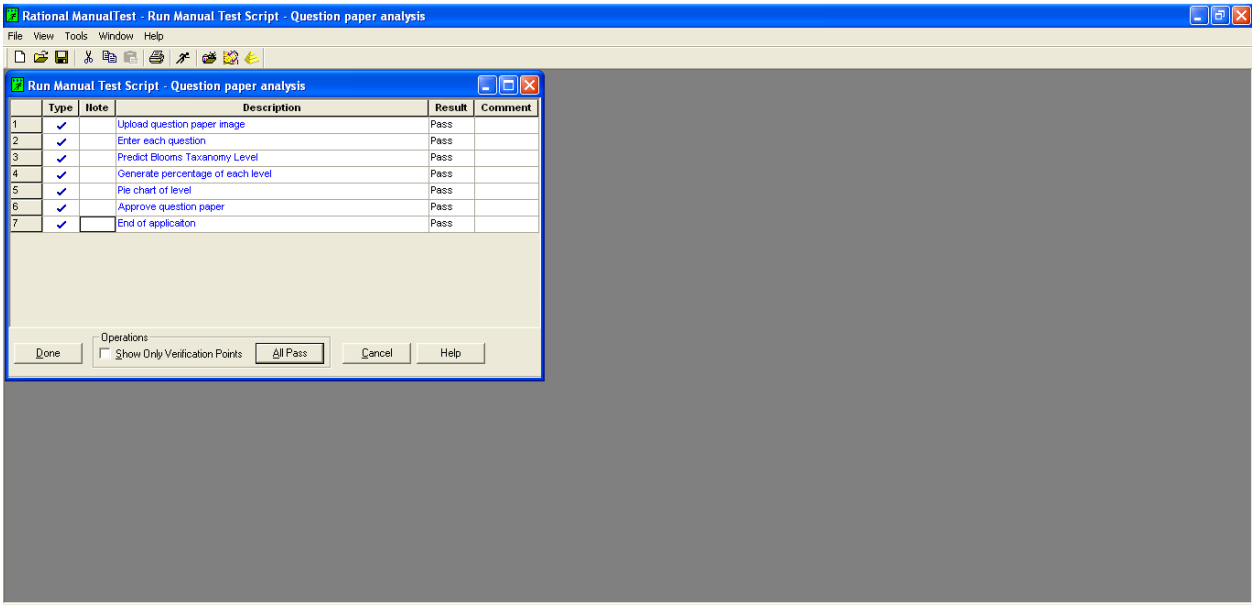


Figure 18: Test Result -1

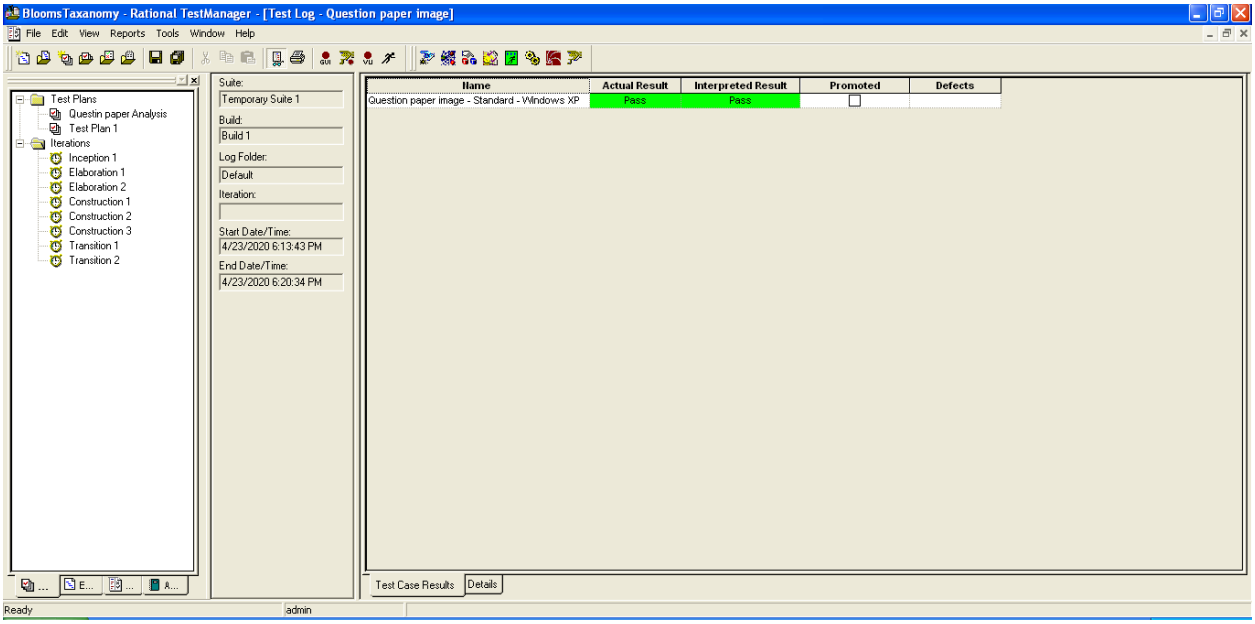


Figure 19: Test Result 2

0.9 Emperical Estimation

0.9.1 LOC Estimation

Add Questions	54
Generate Pie Chart	31
Display Questions	105
Set Results	46
Set User Name	78
OCR	147
Predict	25
Get Percentage	77
Clean Data	112
Normalize Data	29
Randomize Data	17
Plot Chart	63
Split Data randomly (For Test and Train)	31
XGBoost Algorithm	354
Train Model	129
Measure Performance	27
Serialize Model	40
Predict Results	10
Parse Result	5
Config files	104
Total	**1,484**

Table 2: Table of modules and their number of lines of code

0.9.2 Functional Points

External Inputs	Complexity
Data	High
Configuration Files	Low
User Input	Medium
Question Papers	High

Table 3: External Inputs and their Complexities

External Outputs	Complexity
Home Page	Low
Question Upload Page	High
Result Visualize Page	High

Table 4: External Outputs and their Complexities

Internal Logical Files	Complexity
OCR	Medium
Parse Questions	Medium
Classify / Predict Question Category	High
Parse Result and Generate Charts	Low

Table 5: Internal Logic Files and their Complexities

External Interfaces	Complexity
Tesseract OCR API	Medium

Table 6: External Interfaces and their Complexities

External Queries	Complexity
None	—

Table 7: External Queries and their Complexities

0.9.3 Function Point Calculation

After identifying all possible parameters one can calculate the function points as follows:

Parameters	Low	Medium	High
External Inputs	$1*3=3$	$1*4=4$	$2*6=12$
External Outputs	$1*4=4$	$0*5=0$	$2*7=14$
Internal Logical Files	$1*3=3$	$2*4=8$	$1*6=6$
External Interfaces	$0*7=0$	$1*10=10$	$0*15=0$
External Queries	$0*5=0$	$0*7=0$	$0*10=0$

Table 8: Function Points

Here first number in the multiplication is the count parameter in that category and the second number is a fixed number given by the Function Points Method itself.

If we add up the result of each, we get the value **64**. This is called the *un-adjusted FPs value*.

Either we can take this un-adjusted function point value as size or we adjust it using a multiplier.

This multiplier may come from the past project i.e. historical data or from the type of project for which you are estimating. We will try with 3 multipliers as shown below:

Multiplier	Adjusted Function Point
1	64
1.4	90
0.7	44

Table 9: Multipliers for Function Points

Now one can choose any multiplier and then multiply it with the un-adjusted FPs. This will give us the Adjusted function point count.

There is table which can convert the function point's calculation to Source Lines of Code (SLOC) measure. For each function to implement one need 25 to 60 Lines of HTML code and **median value is 35**.

Adjusted Function Points	Median # LOC	Range of # LOC
64	2,240	1,800 - 4,500
90	3150	2,500 - 6,120
44	1540	1,400 - 3,200

Table 10: Median LOC for Adjusted Function Points

We have counted the whitespace and the designer generated code but we do count the comments because a lot of effort was invested in writing good comments.

Therefore in actual code we have got 1484 lines of code which is very near to estimated size of 1540. One can see that this estimation is very close to the actual lines of code written. So, the multiplier for adjusting the function points is **0.7**.

0.9.4 Effort

For effort estimation we used the ISBSG (International Software Bench-marking Standards Group) method. ISBSG takes function points as inputs and number of staff to complete the project and return the effort estimate.

Since we have estimated the size of the project and there are two persons doing the job we can estimate the required effort as:

ISBSG Effort = $0.157 * functionPoints^{0.591} * MaximumTeamSize^{0.810}$ ISGB Effort = $0.157 * 44^{0.591} * 2^{0.810}$
ISGB Effort = 2.5764 staff-month
Hence one can see there is only 6% error in estimation.
One can calculate using basic schedule equation.
Schedule Equation In months= $3.0 * Staff - Months^{1/3}$
Schedule Equation In months= $3.0 * 2.576^{1/3}$
Schedule Equation In months= **4.11 Months**

This 3.0 can be 2.0 or 4.0 based on your organization historical data, but if you don't have the historical data you can go with the number 3.0.

The real effort or the number of actual months in which the project is completed is 4 months which is roughly equal to the schedule equation's output.

0.10 GUI Interface

0.10.1 Front-end Screenshots

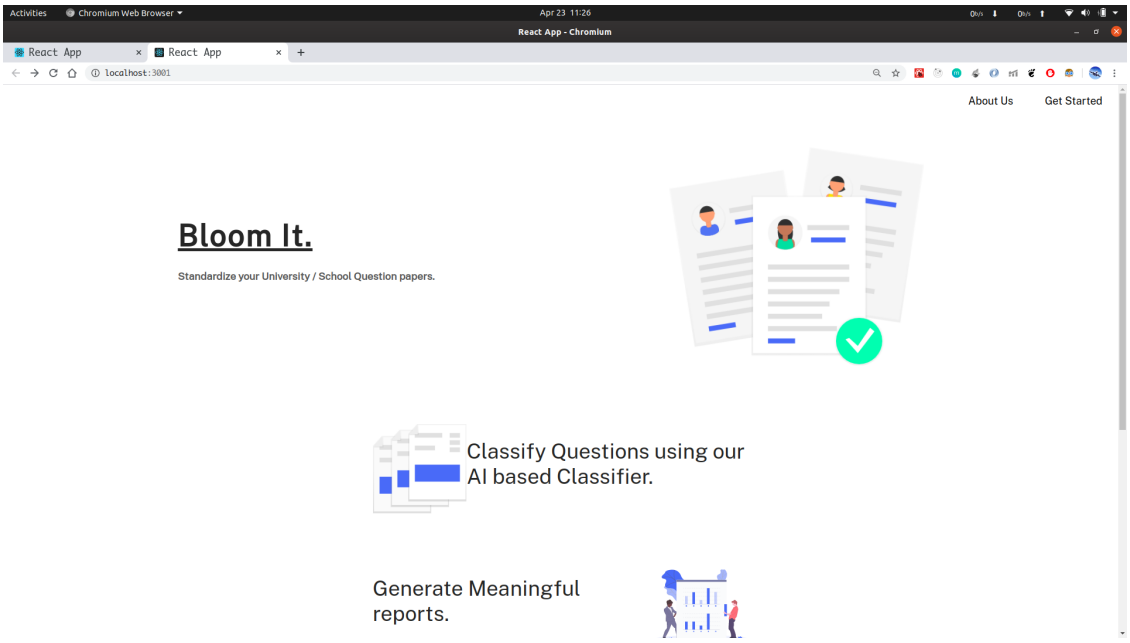


Figure 20: Home Page 1

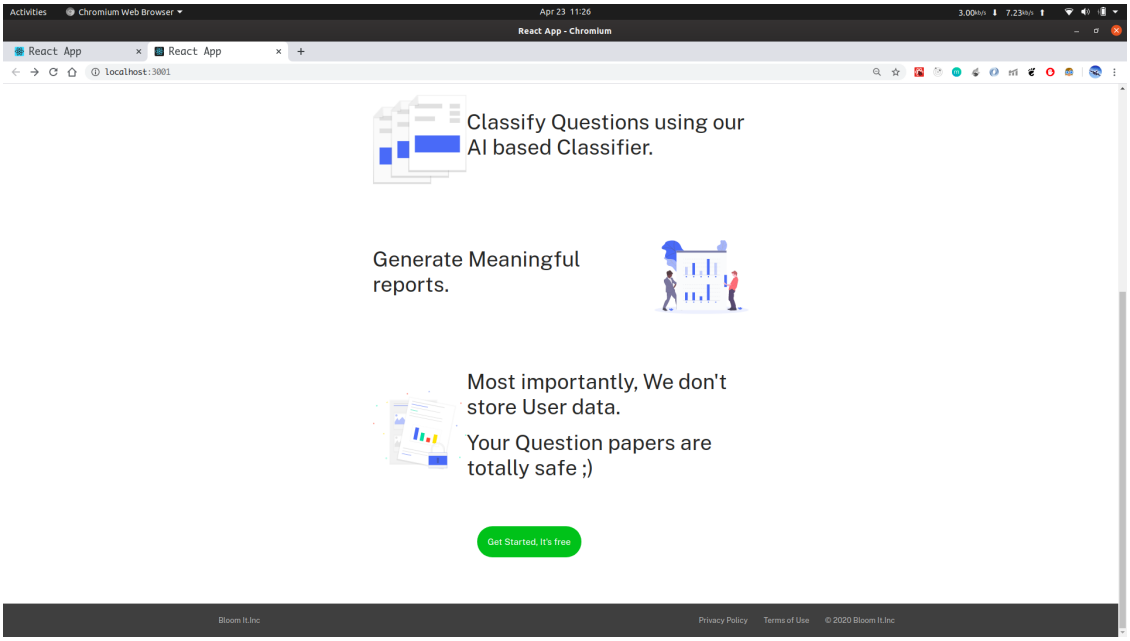


Figure 21: Home Page 2

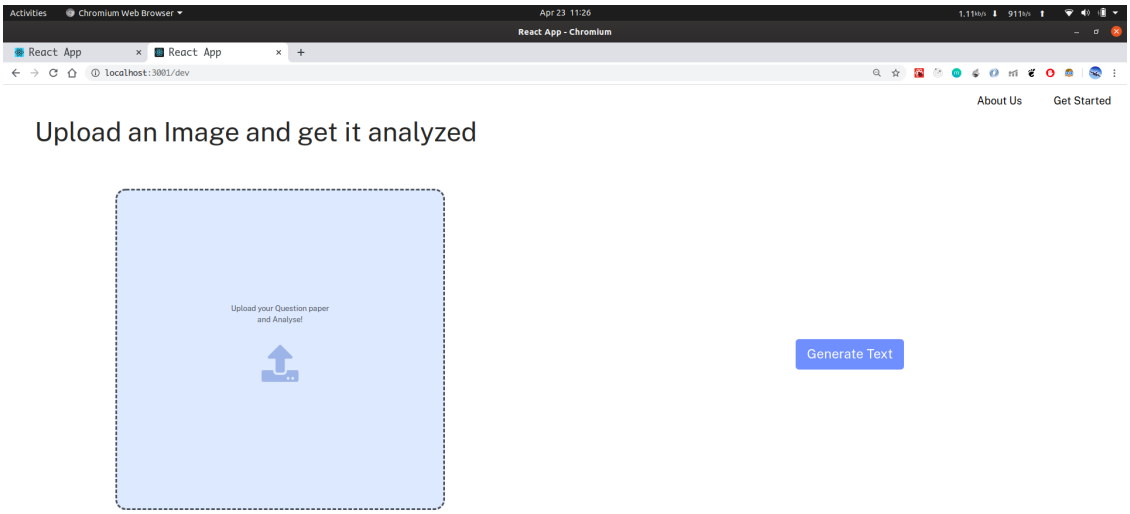


Figure 22: Upload Questions

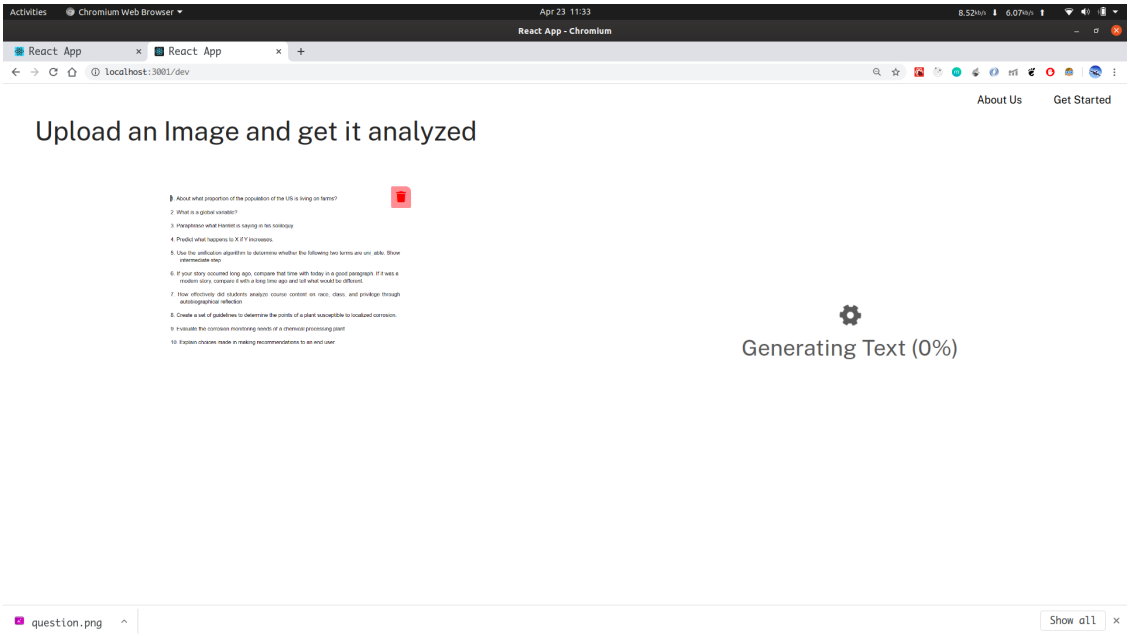


Figure 23: Generating Text

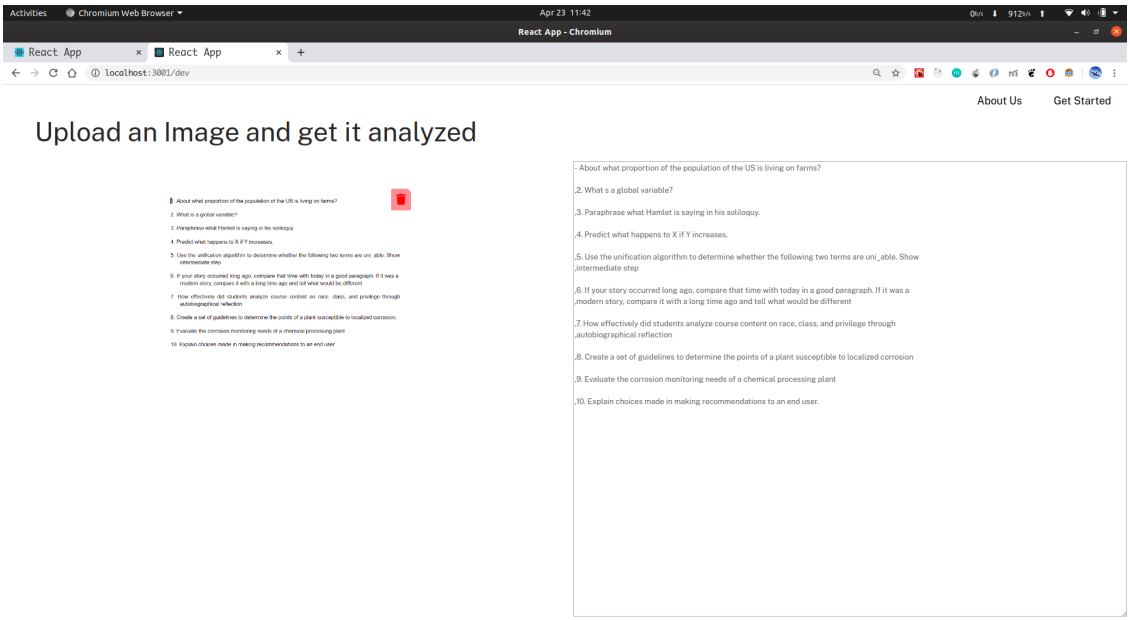


Figure 24: OCR Text

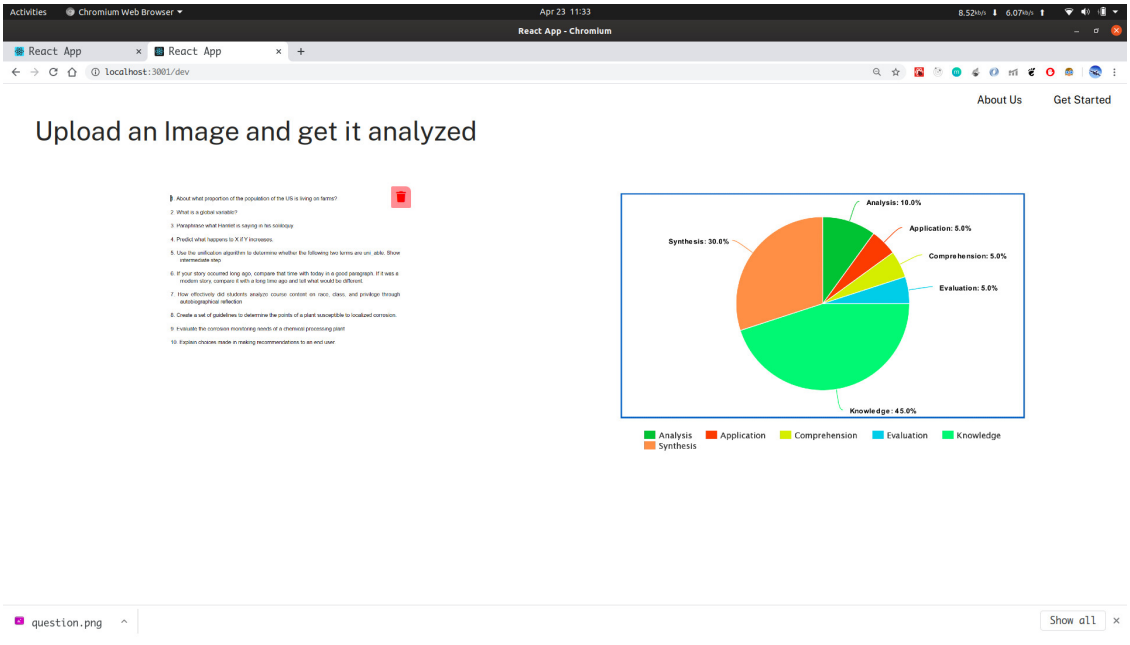


Figure 25: Result

0.10.2 Backend Screenshots

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 31 21:11:06 2019

@author: Ashwin Ram1
"""

import os
import tensorflow as tf
import re
import joblib

from flask import Flask, request, url_for, Response
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import simplejson as json
import numpy as np
import argparse
import imutils
import cv2
import time
import uuid
import base64
import pytesseract
from PIL import Image, ImageEnhance, ImageFilter
import cv2



app = Flask(__name__)


@app.route('/line', methods=['GET', 'POST'])

def upload_line():
    if request.method == 'POST':
        # file = request.files['image']
        name = request.form.get('name')
        name = str(name)
        print(name)
        t = name.split(" ")
        t=[t]
        t = str(t)
        t = [t]
        pred = pipe1.predict_prob(t)


```

Figure 26: FLASK Application - 1

```
54
55          data = {
56
57                  'result': result
58
59
60
61              }
62
63          js = json.dumps(data)
64          res = Response(js, status=200, mimetype='application/json')
65          return res
66
67
68  @app.route('/', methods=['GET', 'POST'])
69
70  def upload_file():
71      if request.method == 'POST':
72          file = request.files['image']
73          filename = file.filename
74          file_path = os.path.join("./", filename)
75
76          file.save(file_path)
77          img = cv2.imread(file_path)
78          img = cv2.resize(img, None, fx=2, fy=2, interpolation=cv2.INTER_LINEAR)
79          img = cv2.medianBlur(img, 3)
80          cv2.threshold(img,127,255,cv2.THRESH_BINARY)
81
82          text = pytesseract.image_to_string(img)
83
84          print(text)
85
86          cor = {
87              0: 'analysis',
88              1: 'application',
89              2: 'comprehension',
90              3: 'evaluation',
91              4: 'knowledge',
92              5: 'synthesis',
93          }
94
95          r = {
96               'analysis': 0,
97               'application': 0,
98           'comprehension': 0,
99               'evaluation': 0,
100              'knowledge': 0,
```

Figure 27: FLASK Application - 2

```
                s=i
                data = re.sub("\d+. ", "", s)
                t = data.split(" ")

                t=[t]
                t = str(t)
                t = [t]
                pred = pipe1.predict_proba(t)

                resu = np.argmax(pred)
                print(resu)

                resu=int(resu)
                res = cor[resu]
                r[res]=r[res]+1

        for key in r:
            r[key]=((r[key])/len(text.split("\n")))*100


        print(r)




        js = json.dumps(r)
        res = Response(js, status=200, mimetype='application/json')
        return res




if __name__ == "__main__":
    pipe1 = joblib.load("filename.pkl")


    pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

    app.run(host="localhost",port=8000,debug=True)
```

Figure 28: FLASK Application - 3

In [ ]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under
the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

In [ ]:
```
import logging
import pandas as pd
import numpy as np
from numpy import random
import gensim
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import re
from bs4 import BeautifulSoup
%matplotlib inline
```

In [ ]:
```
import gensim
wv = gensim.models.KeyedVectors.load_word2vec_format("/kaggle/input/googlenewsvectorsnegative30
0/GoogleNews-vectors-negative300.bin.gz", binary=True)
wv.init_sims(replace=True)
```

In [ ]:
```
def word_averaging(wv, words):
    all_words, mean = set(), []

    for word in words:
        if isinstance(word, np.ndarray):
            mean.append(word)
        elif word in wv.vocab:
            mean.append(wv.syn0norm[wv.vocab[word].index])
            all_words.add(wv.vocab[word].index)

    if not mean:
        logging.warning("cannot compute similarity with no input %s", words)
        # FIXME: remove these examples in pre-processing
```

Figure 29: Machine Learning Model - 1

In [ ]:
```
import gensim
wv = gensim.models.KeyedVectors.load_word2vec_format("/kaggle/input/googlenewsvectorsnegative300/GoogleNews-vectors-negative300.bin.gz", binary=True)
wv.init_sims(replace=True)
```

In [ ]:
```
def word_averaging(wv, words):
    all_words, mean = set(), []

    for word in words:
        if isinstance(word, np.ndarray):
            mean.append(word)
        elif word in wv.vocab:
            mean.append(wv.syn0norm[wv.vocab[word].index])
            all_words.add(wv.vocab[word].index)

    if not mean:
        logging.warning("cannot compute similarity with no input %s", words)
        # FIXME: remove these examples in pre-processing
        return np.zeros(wv.vector_size,)

    mean = gensim.matutils.unitvec(np.array(mean).mean(axis=0)).astype(np.float32)
    return mean

def  word_averaging_list(wv, text_list):
    return np.vstack([word_averaging(wv, post) for post in text_list ])
```

In [ ]:
```
def w2v_tokenize_text(text):
    print(text)
    tokens = []
    for sent in nltk.sent_tokenize(text):
        print(sent)
        for word in nltk.word_tokenize(sent):
            print(word)

            if len(word) < 2:
                continue
            tokens.append(word)
    return tokens
```

In [ ]:
```
df = pd.read_csv("/kaggle/input/finalset/blooms.csv",error_bad_lines=False)
```

In [ ]:
```
df=df.drop(df.columns[[0]], axis=1)
```

In [ ]:
```
d[0]
```

In [ ]:
```
l=literal_eval(ans[0])
l=[l]
```

In [ ]:
```
str(l)
```

In [ ]:
```
def extractDigits(lst):
    res = []
```

Figure 30: Machine Learining Model - 2

```
d=[i for i in X_test]
from sklearn.preprocessing import LabelBinarizer, LabelEncoder

from ast import literal_eval
# ans = [i for i in d[0]]
# print (ans)
# ans[1]
# # [i for i in ans]
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
porter=PorterStemmer()
lemmatizer = WordNetLemmatizer()
text=[literal_eval(i) for i in d]


tt=[[str(j) for j in i[0]] for i in text]
t=[]
for u in range(0,len(tt)):
    t.append([tt[u]])
# tt2=[j for j in tt[0]]

# print(X_train)
# print(tt)

wordList= []
i1=0
# w[1000]=[]
w={}
for i in tt:
#     print(i)
#     i = ngrams(i,3)


#     wordList.append([])
    w[i1]=[]
    for j in i:

        h=wordnet_lemmatizer.lemmatize(j)

        w[i1].append(porter.stem(h))
#           w[i1].append(j)
    i1=i1+1

# wordList
# ans=[]
ans=[str(en) for en in w.values()]
# tt2
l=[]
for i in range(0,len(ans)):
    l.append(str([literal_eval(ans[i])]))


X_test=l
```

Figure 31: Machine Learining Model - 3

0.11 Contributions

Aravind Srinivasan (121003029)

- Front End Development
- Integration with Backend
- OCR text generation

Ashwin Ram S (121003035)

- Data population and Cleaning
- Training ML Model
- Backend FLASK Application