Question Bank
Extracting questions from PDF

Huzefa Chasmai (15D170013), Akshith Reddy (150050074)
Guide: Prof. Kameswari Chebrolu

May 7, 2019

Abstract

Question Bank is a collection of questions organized according to chapter, difficulty, etc. Using the question bank to automate the task of generating new question papers according to specification of constraints like percentage allotted to chapter, marks and difficulty would save a lot of manual time. To this end, the task of populating the database with questions from already available data, in the form of question papers, is a major hurdle. PDFs are the most common format available for question papers, and hence automated extraction of questions from PDFs are imperative. We extend the current work[1] by improving the PDF extraction algorithm and changing the architecture to make it more robust. Furthermore, correctness guarantees for the automated extraction is difficult and hence some amount of manual intervention is necessary. We tackle this by building our Visual Editing Tool which provides the user with a web based tool for manual verification and easy modification of extracted content.

Contents

1 Introduction

Most of the professors would find it convenient to have a database of questions and answers from which they can generate new question papers. A tool which populates this database would be very useful since a huge source of questions are previous question papers that are already present. A tool that automatically extracts questions and answers from these papers and populate them into a database would be desirable.

The question papers available online are in many different formats, amongst which PDF is the most common. Furthermore loss-less conversion to PDF format is supported by most of the other formats. Therefore, a tool to extract question and answers from the PDF was imperative.

Due to high variability in the format or style of the question papers online, perfect automated extracted would be difficult to achieve. Therefore the tool should involve certain amount of human intervention to guarantee correctness. To this end, the tool should provide the users with an interface where they can easily modify the extracted elements to fit to their needs.

2 Related Work

2.1 Extraction from PDFs

Our project is an extension to the RnD project work done by Sai Sharath and Nagendra Reddy[1]. They had worked on a python2 based CLI project that takes question paper PDF as input and creates annotation boxes around the questions. The user needs to edit these annotated questions using third party tools like Foxit Reader and save the new formatted pdf. The new annotated pdf is given as input to an extractor that extracts the text content and the images in the form of a latex file. The latex captures information like the font style so that the questions can be reproduced as close as possible to the original pdf. We will be extending the work done by them.

2.2 Interface for PDF Interaction

[2] built a web application using the Vue.js (A progressive JavaScript framework). The application is used to create boxes over PDFs and is able to detect the coordinates of the boxes created in the PDF. They generate code which can then be used to extract just the text content out of the bounding boxes. This code has to be used separately using the PDFQuery or tesserac libraries in Python2.

2.3 Extraction from latex

For the purpose of populating the question bank, a similar tool is being developed to extract the questions and answers from latex versions of the question papers is being done as part of a MTech Project here. This project also deals with storing the questions into a database using an extractor working with latex question papers as input.

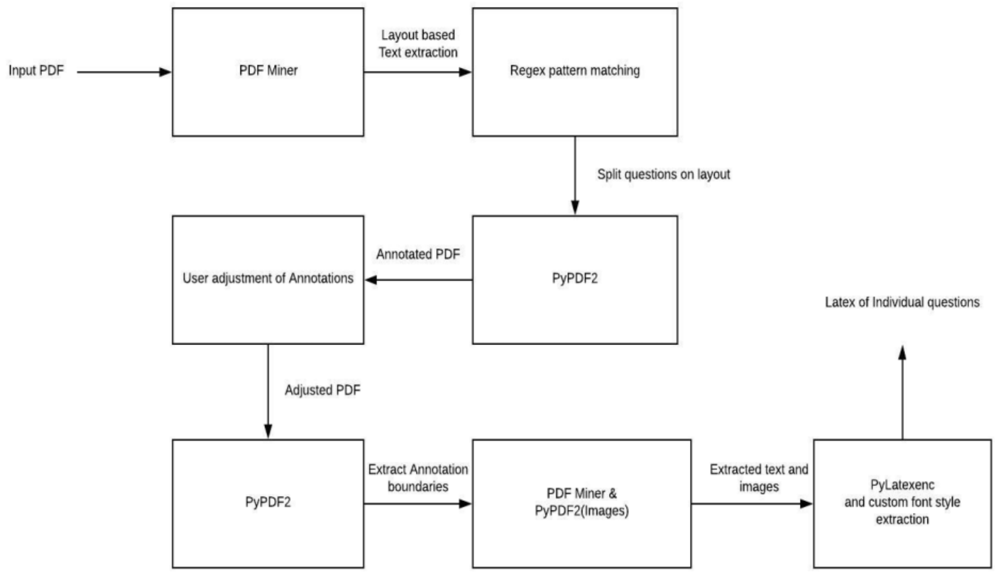
3 Extending the PDF Extraction Program

This section details modifications to the existing code to fix certain issues and our additions to the algorithm[1] of Question Extraction from PDFs.

3.1 Existing code Overview

This section is borrowed from [1]

Create annotation boxes around each question region. The annotated PDF can then be inspected manually in a tool such as foxit reader for any adjustments of the annotations. After these manual adjustments this tool extracts each annotated region as a question along with images and other metadata.


Input PDF
PDF Miner
Layout based Text extraction
Regex pattern matching
Split questions on layout
PyPDF2
Annotated PDF
User adjustment of Annotations
Adjusted PDF
PyPDF2
Extract Annotation boundaries
PDF Miner & PyPDF2(Images)
Extracted text and images
PyLatexenc and custom font style extraction
Latex of Individual questions

The above description lead to the architecture taking the above form. The Input PDF first goes through PDF Miner for the extraction of PDF layout in the form of a tree along with coordinates of each layout. The extracted layout also contains all the metadata of text, images etc... The text part of the entire layout is subjected to a regex pattern matching based on which the layout of individual questions is determined. This extracted layout is used to draw annotation boxes around each question with the help of PyPDF2 library. User can then make adjustments to the annotated PDF and submit it for extracting each question as a latex text. The adjusted PDF then goes through PyPDF2 for the extraction of annotation boxes.

These annotation box boundaries are used to extract the text and images inside these boxes. This is performed using PDF miner and PyPDF2 (for png images). The extracted text is then converted to latex using Pylatexenc. The images and font styles are adopted into the latex programmatically.

3.2 Migration from Python2 to Python3

The existing code was written in Python2. We migrated the code to Python3 with a few minor modifications of changing the versions of pdfminer library and slight modifications to the code for sorting and encoding byte arrays as string. This was done so that no compatibility issues arise in the future.

3.3 Modifications to the existing code

3.3.1 Input the regex used for detection

Working with a fixed set of regex for question detection is often inefficient and doesn't cover all scenarios. The ideal case is to input the regex for question detection based on the text in the PDF, which is incorporated in the modified code.

3.3.2 Complete creation of annotated boxes

The existing code creates annotated boxes around questions after detecting them but the boxes displayed are incomplete in nature. This problem especially occurs when the question is spanned across multiple pages but the code only generates the annotated box for a part of the question present on the initial page. This issue is fixed by generating an extra box for all the lines starting in a page which haven't been bounded by any box. In such scenarios, these boxes are assumed to be the continuation of last annotated box in the previous page.

3.3.3 Fixing problems with PDFMiner

The tool used for extracting text from PDF - PDFMiner is not exactly accurate. For example, there can be scenarios in which the extracted lines may not be in the sorted order. Hence, a pre-processing step in the form of position based sorting has to be taken on the extracted lines for better working of the code. Also, zero width/height lines must be removed to eliminate any problems caused during text extraction. Limit on the bottom position of the lines extracted is set to remove irrelevant lines like page number, unnecessary footers, etc.

3.3.4 Robust Indexing

The existing code has an issue of getting the index of a particular set of lines from the entire array by using the text of the line. This approach is not robust as there can be two or more lines in the PDF with same text. This issue is fixed by storing the indexes of the lines corresponding to the set when generating the set itself.

3.4 Major Architectural Changes

3.4.1 Extraction of Subquestions, Answers and Metadata

The question bank database structure supports multi-level question storage in the form of Questions and Sub Questions. Other than that they have separate fields for metadata like marks. In order to match these we provide the user with options to input separate regexes to detect subquestions, answers and other metadata like marks (we refer to them collectively as elements henceforth). The code for the metadata is robust so that it can be expanded to include additional metadata other than marks in the future. These functionalities are added in a robust and independent manner so that question papers of different formats (for e.g. QQ...AAA... or QAQA...) can be extracted easily. In the first case though, the mapping of questions to the corresponding answers is to be done manually by the user.

3.4.2 Font based detection of Answers

The regex based approach to detect an answer element in the text may not work because there need not be a fixed delimiter to separate answers from the rest of the question. Often in such cases where there's no delimiter for the answers, they are displayed as either bold, italic or an entirely different font from the font used for the question. In such cases, we can use the change in the font/style of the text to partition the answer and question elements in the text. The pseudocode for the approach is as follows

```
def detectAnswerPosition(lines):
    # returns the top two fonts(including styles) in the entire text
    max_fonts = detectMaxFonts(lines)
    # returns the array of leading font used in each line
    font_array = maxFontForLine(lines)
    # returns the index where the font changes from one max_font to another
    index = detectChangeFont(font_array)
    # denotes that the answer starts at this index
    return index
```

3.4.3 MCQ Options Detection

'MCQ Question Papers' i.e., question papers where all the questions were of the multiple choice format proved to be an important distinction from the rest of the papers which were of the descriptive type. So we decided to analyze this case separately, we ask the user to specify if the format of the paper is MCQ, and then we ask the user to give the regexes to detect the options of the questions. This was done in order to make it easier for the user to add/remove options, and basically treat the options object independently from the question instead of it being a part of the question.

3.4.4 Changing the Model for storing the elements

Previously, since only the questions were being detected, the model of storing the elements was in the form of an annotation box which consisted only of the bounding box coordinates. In our current architecture since additional elements other than questions are included, a need to update this model arises. We use a selections object for each element. The structure of the selection object can be thought of as an abstract class as follows :

```
class SelectionObject(AbstractBaseClass):
    id       : #Stores unique identifier of the element
    type     : #Element Type (MainQues, SubQues, Answer)
    color    : #Used to store the color in which the bounding box
    name     : #The name of the element {e.g. Main Ques 1, Sub Ques 2}
    marks    : #Marks corresponding to the element (empty for answers)
    qnum     : #Main Question Number associated with the element
    textData: #The text in the element bounding box
    question_type : #Whether MCQ or descriptive
    options : #In case of MCQs, list of options associated with question
    coordinate : #The bounding box coordinates -> object of Coordinate

class Coordinate(AbstractBaseClass):
    page          : #Page number of the PDF in which element present
    pageOffset    : #Offset from start of PDF to start of Page(#page)
    height        : #height of the element
    width         : #width of the element
    left          : #x position of the left top corner
    right         : #y position of the left top corner
```

3.4.5 Extracting the elements in the form of text

We changed the functionality from extraction of elements as latex in the previous work to extraction in the form of text. This was because eventually the extracted components were going to get saved in the form of fields of database objects and this could be achieved using a text based extraction format which is much simpler. The additional benefits of the latex based extraction can be incorporated in the form of meta-data specifying the font style, color, etc.

4 Visual Editing Tool

This section describes the idea and the program flow of our Visual Editing Tool. This is a visual web application with the purpose of allowing users to manually verify and edit the extracted elements and the corresponding selection boxes.

4.1 Description

We are building a visual tool which integrates the web browser with the necessary functionalities required to perform over a PDF for this project. The tool consists of a Vue.js based FrontEnd and a python Flask based BackEnd. The FrontEnd is a web application to upload the PDFs, create the necessary selection boxes, view and edit text corresponding to the annotations and submit the final changes. The BackEnd listens to the requests from the FrontEnd and serves them with appropriate response. The visual interpretations in FrontEnd are a result of these responses. More on the details are described below

4.2 Front End Program Flow

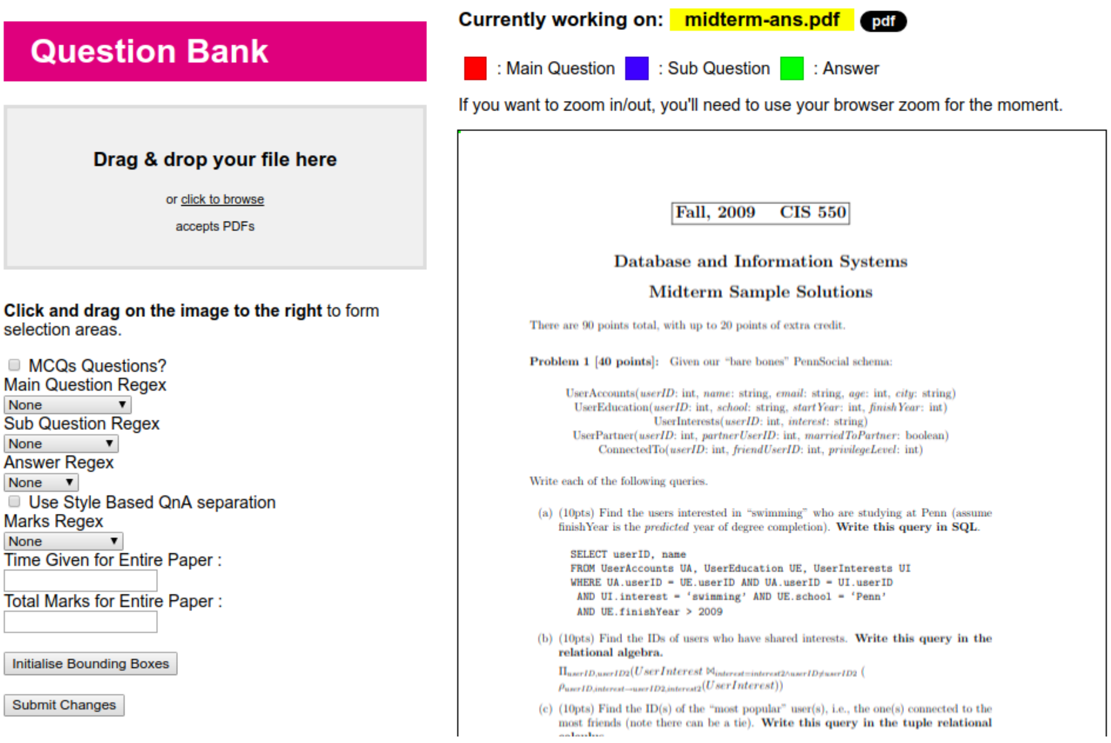
4.2.1 Uploader

This component contains is the section to drag and drop pdfs or browse them through your local files. Once uploaded the PDF will be displayed in the main area as shown in (Figure 1).

4.2.2 Initialize Input

This component corresponds to user inputs for some of the fields mentioned below. These fields (only some compulsory) are necessary for the auto detection of the elements and the button <initialize bounding boxes> to generate and display the bounding boxes on the PDF. The input for automating boxes consists of the following parts

- `Main Question Regex`: Regex pattern to identify the start of a main question
- `Sub Question Regex`: Regex pattern to identify the start of a sub question
- `Answer Regex`: Regex pattern to identify the start of a answer
- `Marks Regex`: Regex pattern to identify the marks


Question Bank

Drag & drop your file here

or click to browse

accepts PDFs

Click and drag on the image to the right to form selection areas.

☐ MCQs Questions?

Main Question Regex: None

Sub Question Regex: None

Answer Regex: None

☐ Use Style Based QnA separation

Marks Regex: None

Time Given for Entire Paper :

Total Marks for Entire Paper :

Currently working on: midterm-ans.pdf pdf

🟥 : Main Question 🟦 : Sub Question 🟩 : Answer

If you want to zoom in/out, you'll need to use your browser zoom for the moment.

Fall, 2009 CIS 550

Database and Information Systems

Midterm Sample Solutions

There are 90 points total, with up to 20 points of extra credit.

Problem 1 [40 points]: Given our "bare bones" PennSocial schema:

UserAccounts(*userID*: int, *name*: string, *email*: string, *age*: int, *city*: string)
UserEducation(*userID*: int, *school*: string, *startYear*: int, *finishYear*: int)
UserInterests(*userID*: int, *interest*: string)
UserPartner(*userID*: int, *partnerUserID*: int, *marriedToPartner*: boolean)
ConnectedTo(*userID*: int, *friendUserID*: int, *privilegeLevel*: int)

Write each of the following queries.

(a) (10pts) Find the users interested in "swimming" who are studying at Penn (assume finishYear is the *predicted* year of degree completion). **Write this query in SQL.**

```
SELECT userID, name
FROM UserAccounts UA, UserEducation UE, UserInterests UI
WHERE UA.userID = UE.userID AND UA.userID = UI.userID
 AND UI.interest = 'swimming' AND UE.school = 'Penn'
 AND UE.finishYear > 2009
```

(b) (10pts) Find the IDs of users who have shared interests. **Write this query in the relational algebra.**

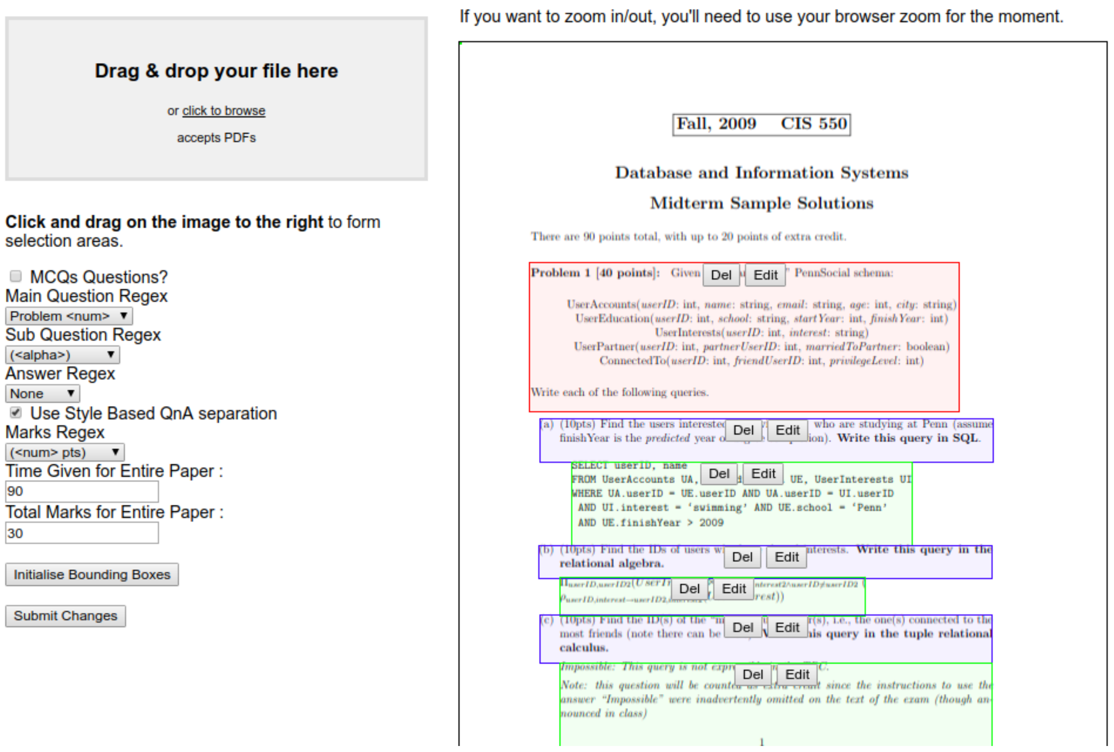
$\Pi_{userID,userID2}(UserInterest \bowtie_{interest=interest2 \wedge userID \neq userID2} ($
$\rho_{userID,interest \rightarrow userID2,interest2}(UserInterest))$

(c) (10pts) Find the ID(s) of the "most popular" user(s), i.e., the one(s) connected to the most friends (note there can be a tie). **Write this query in the tuple relational**

Figure 1: Initial Page

- `Style based checkbox`: Whether or not to use font/style change to detect answer
- `MCQ checkbox`: Whether or not all questions in the PDF are MCQ
- `MCQ Regex`: Regex pattern to identify the start of the options
- `Time Given for Entire paper`: Enter the total exam duration
- `Total Marks for Entire paper`: Enter the total exam marks

Total duration and total marks are used to adjust the scale of the marks extracted from the PDF. Also, when you don't find the relevant option to select in any of the Regex fields, you can click on Other and entire the desired regex input in the description.



If you want to zoom in/out, you'll need to use your browser zoom for the moment.

Click and drag on the image to the right to form selection areas.

☐ MCQs Questions?
Main Question Regex
Problem <num>
Sub Question Regex
(<alpha>)
Answer Regex
None
☑ Use Style Based QnA separation
Marks Regex
(<num> pts)
Time Given for Entire Paper : 90
Total Marks for Entire Paper : 30

Fall, 2009 CIS 550

Database and Information Systems

Midterm Sample Solutions

There are 90 points total, with up to 20 points of extra credit.

Problem 1 [40 points]: Given " PennSocial schema:

UserAccounts(*userID*: int, *name*: string, *email*: string, *age*: int, *city*: string)
UserEducation(*userID*: int, *school*: string, *startYear*: int, *finishYear*: int)
UserInterests(*userID*: int, *interest*: string)
UserPartner(*userID*: int, *partnerUserID*: int, *marriedToPartner*: boolean)
ConnectedTo(*userID*: int, *friendUserID*: int, *privilegeLevel*: int)

Write each of the following queries.

(a) (10pts) Find the users intereste who are studying at Penn (assume finishYear is the *predicted* year o ion). **Write this query in SQL.**

```
SELECT userID, name
FROM UserAccounts UA,      UE, UserInterests UI
WHERE UA.userID = UE.userID AND UA.userID = UI.userID
 AND UI.interest = 'swimming' AND UE.school = 'Penn'
 AND UE.finishYear > 2009
```

(b) (10pts) Find the IDs of users w nterests. **Write this query in the relational algebra.**

$\Pi_{userID,userID2}(UserI$ $nterest2 \wedge userID \neq userID2$
$\rho_{userID,interest \to userID2,}$ $rest))$

(c) (10pts) Find the ID(s) of the "m r(s), i.e., the one(s) connected to the most friends (note there can be his query in the tuple relational calculus.**

Impossible: This query is not expr C.

Note: this question will be count since the instructions to use the answer "Impossible" were inadvertently omitted on the text of the exam (though announced in class)

1

Figure 2: After Initializing Bounding Boxes using our algorithm

4.2.3 Area Select

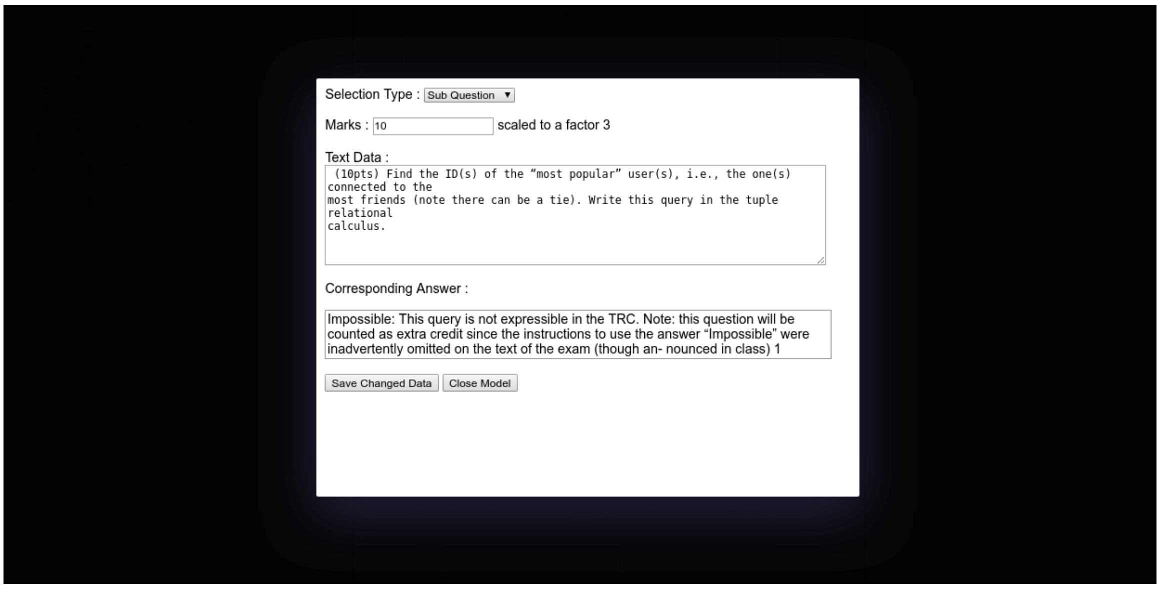
This component corresponds to a selection box displayed on a PDF. The color of the box determines what type of an element the selection corresponds to (red - main question, blue - sub question, green - answer). Additionally the component contains buttons like <Del> to delete the component and <Edit> to open a Modal View. The functionalities of a modal view are described below. Additionally you can create a new selection box by dragging the cursor over the concerned area of the PDF.

4.2.4 Modal View

Upon clicking the edit button in the AreaSelect, a modal view pops (Figure3) up on the screen. This modal view has the following components:

- `Selection Type`: The type of the selection (editable)
- `Marks`: The marks obtained (editable)

- `Scale`: This number is the scale to be multiplied with for marks to be displayed as a function of time in minutes
- `TextData`: The text data corresponding to the element
- `Corresponding Element`: This contains the text data of the corresponding element (for subquestion/question it is the corresponding answer, and for answer it is the corresponding question/subquestion if it exists)
- `Save Changed Data Button`: This button saves the changes done in the local variable at the FrontEnd.
- `Close Modal`: Close the modal and go back to the home screen


Selection Type : Sub Question

Marks : 10 scaled to a factor 3

Text Data :

```
 (10pts) Find the ID(s) of the “most popular” user(s), i.e., the one(s)
connected to the
most friends (note there can be a tie). Write this query in the tuple
relational
calculus.
```

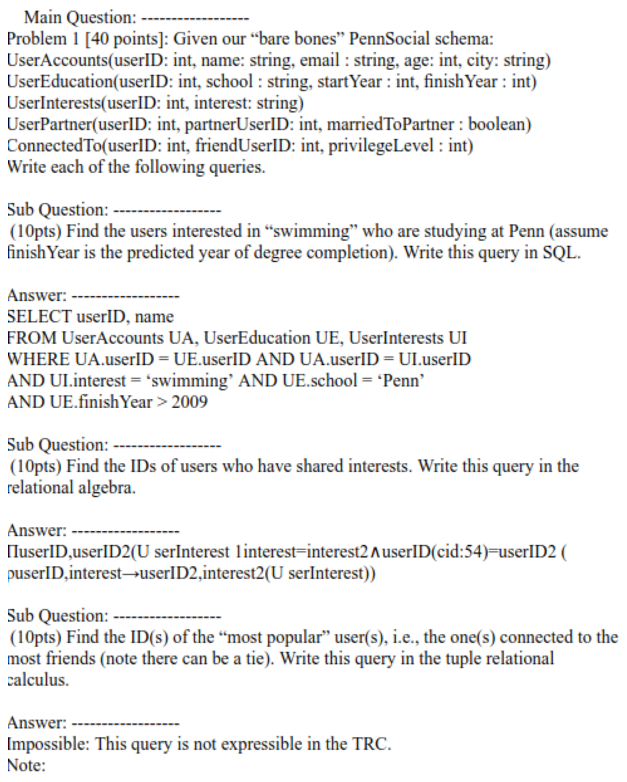
Corresponding Answer :

Impossible: This query is not expressible in the TRC. Note: this question will be counted as extra credit since the instructions to use the answer “Impossible” were inadvertently omitted on the text of the exam (though an- nounced in class) 1

Figure 3: Edit Modal View for editing the selection Boxes

4.2.5 Display Saved Text for Database

Upon clicking the save Data button, the BackEnd extracts the elements from the updated selection boxes and saves it in the database. Upon clicking this button, a link appears to view this saved data. The link to saved data shows the extracted elements in a text file (Figure 4) separated by the delimiters of the form : ”Main Question: ——————”. In addition to this, the necessary images get saved in



```
  Main Question: -----------------
Problem 1 [40 points]: Given our "bare bones" PennSocial schema:
UserAccounts(userID: int, name: string, email : string, age: int, city: string)
UserEducation(userID: int, school : string, startYear : int, finishYear : int)
UserInterests(userID: int, interest: string)
UserPartner(userID: int, partnerUserID: int, marriedToPartner : boolean)
ConnectedTo(userID: int, friendUserID: int, privilegeLevel : int)
Write each of the following queries.

Sub Question: -----------------
(10pts) Find the users interested in "swimming" who are studying at Penn (assume
finishYear is the predicted year of degree completion). Write this query in SQL.

Answer: -----------------
SELECT userID, name
FROM UserAccounts UA, UserEducation UE, UserInterests UI
WHERE UA.userID = UE.userID AND UA.userID = UI.userID
AND UI.interest = 'swimming' AND UE.school = 'Penn'
AND UE.finishYear > 2009

Sub Question: -----------------
(10pts) Find the IDs of users who have shared interests. Write this query in the
relational algebra.

Answer: -----------------
ΠuserID,userID2(U serInterest 1interest=interest2∧userID(cid:54)=userID2 (
ρuserID,interest→userID2,interest2(U serInterest))

Sub Question: -----------------
(10pts) Find the ID(s) of the "most popular" user(s), i.e., the one(s) connected to the
most friends (note there can be a tie). Write this query in the tuple relational
calculus.

Answer: -----------------
Impossible: This query is not expressible in the TRC.
Note:
```

Figure 4: Viewing the saved data

the BackEnd in the images directory in the server. The images are referenced in the text file by mentioning their names along with the corresponding element for reproducibility of the question from the text file.

4.3 BackEnd Architecture

This section details the BackEnd architecture and also the FrontEnd-BackEnd-program/database interactions (Figure 5). The BackEnd is the key for communication between the FrontEnd and the program. We will analyze these interactions by looking at the uses and the flow across the various routes present at the BackEnd.

4.3.1 dbView

- Input
  - `file name:` name of the PDF file for which we require the saved data

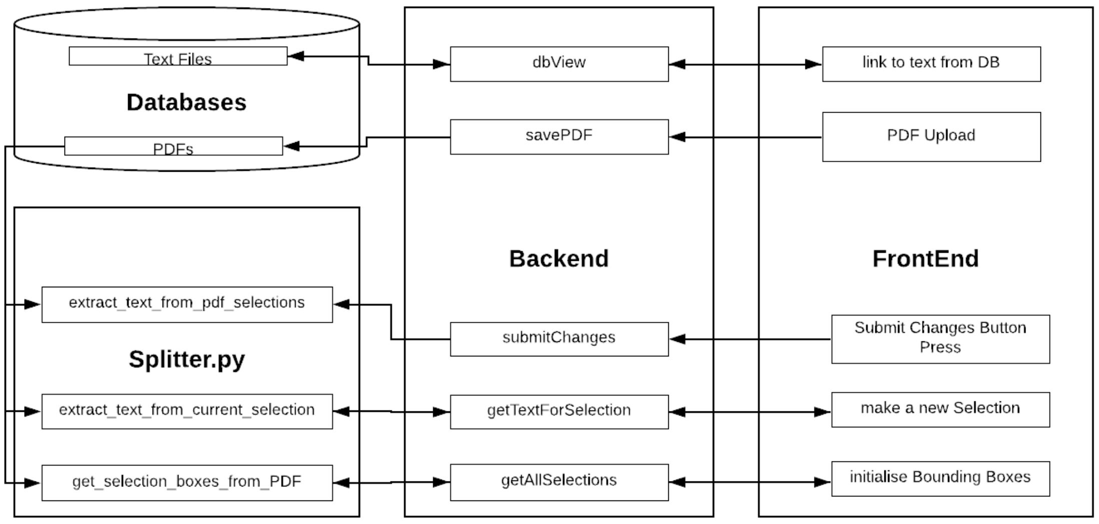


Figure 5: Architecture

- Output - returns the entire contents of the text file in which the data corresponding to the PDF file is saved
- FrontEnd Side - Upon clicking the 'View Saved Data' link, it opens in a new tab with a GET request with this route is sent to the BackEnd. The pdf file name is incorporated as part of the get request and the received response is displayed on this page
- Program Side Algorithm

```
def return_files_tut(input):

        # Get the text file path corresponding to the PDF
        text_filepath = get_saved_filepath(filename)

        #Get text data from the file
        saved_data = readfile(text_filepath)

        #Return the saved data
        return saved_data
```

4.3.2 savePDF

- Input
  - `file`: The PDF file which is to be saved
- Output - saves the received PDF file in the desired location of database
- FrontEnd Side - Upon uploading a PDF file, this route for BackEnd is being called using an ajax POST request with the entire file present in the request
- Program Side Algorithm

```
def savePDF(input):

        # Get the file object from request
        file = extract_file(request)

        #Save the file in the desired location
        file.save(filename, pdf_locatiom)
```

4.3.3 submitChanges

- Input
  - `file_name`: name of the file to access from DB and to save the extracted text and images in the DB
  - `selections`: List of all the selection objects in the FrontEnd
  - `pdf_dimensions`: Dimensions of the PDF
- Output - returns success or failure. We store the text corresponding to all the selection boxes in the BackEnd in a file names file_name.txt
- FrontEnd Side - after modifying all the boxes to suit their requirement, the user clicks on the 'Save Changes' button that leads to an AJAX call to this route
- Program Side Algorithm

```
def extract_text_from_pdf_selections(input):

        #Preprocessing to get the layout and the images
        pdfInput = PdfFileReader(open(pdf_path, "rb"))
        image_res = get_image_res(pdfInput)
```

```
layout = get_PDF_layout(pdf_path)

page_ht = pdf_dimensions['height']

#Sort the selections list in ascending order of y position
sorted_selections = get_sorted_selections(selections)

text_list = []

#Iterate throgh each selection in the selctions list
for curr_selection in sorted_selections:

   #Get coordinates in the BackEnd dimension scale
   curr_bbox = get_bounding_box_coords(
      curr_selection['coordinates'], page_ht)

   #Get a list of all images in this box
   LT_list_images = get_LT_image_list_from_box(
      layout[curr_selection['coordinates']['page'] - 1],
      curr_bbox[0], curr_bbox[2], curr_bbox[1], curr_bbox[3])

     #Extract the text from the current selection
   curr_text = get_text_corr(curr_selection)

   #Save the images and link them in the text file
   curr_text_add += save_image(LT_list_images,
                               curr_selection, image_res)

   #Add current selection text in the list
   text_list.append(curr_text)

#Write the text in the DB text file
file = open(db_path, "w+")
file.write("\n".join(text_list))

return text_list
```

4.3.4 getTextForSelection

- Input
  - `file_name`: name of the file to access from DB
  - `page_num`: page number of PDF
  - `pdf_dimensions`: PDF dimensions
  - `coordinates`: Coordinates of the selection

- Output - returns the text extracted from the selection in consideration
- FrontEnd Side - while making a new selection on the PDF by dragging a box, we need the text corresponding to the box to be displayed to the user in the modal upon clicking Edit. This route is called using the AJAX query.
- Program Side Algorithm:

```
def extract_text_from_current_selection(input):

        #Height of the pdf
        page_ht = pdf_dimensions['height']

        # Get layout(pdfminer) object of the PDF
        layout = get_PDF_layout(pdf_path)

        #Get the coordinates in the BackEnd pdf dimension scale
        curr_bbox = get_bounding_box_coords(coordinates, page_ht)

        #Get all the lines within the box
        LT_list = get_LT_line_list_from_box(layout[page_num - 1],
               curr_bbox[0], curr_bbox[2], curr_bbox[1], curr_bbox[3])

        #Sort the lines so that text displayed is in order
        LT_list = get_sorted_lines(LT_list)

        #Extract the text out from each of the lines
        text_data = get_text_from_LTList(LT_list)

        return text_data
```

4.3.5 getAllSelections

- Input
  - `file_name`: name of the file to access from DB
  - `regexes`: regex Pattern corresponding to question, subquestion, ans and metadata
  - `use_style`: boolean for whether to use style or not
  - `pdf Dimensions`: the dimensions of the pdf
  - `question_type`: boolean for whether paper is MCQ
  - `mcq_reg`: Regex for MCQ options

- Output - returns a list of selection objects corresponding to each element
- FrontEnd Side - upon pressing the initialize bounding box, this route of the BackEnd is being called using an AJAX POST request. The request is passed using the input fields as parameters of the query.
- Program Side Algorithm - we call the 'get_selection_boxes_from_PDF' provided by our program. This functions basic structure is :

```python
def get_selection_boxes_from_PDF(input):

        # Get layout(pdfminer object) of the PDF
        layout = get_PDF_layout(pdf_path)

        #Compile all regexes
        regs_all = [re.compile(patterns)]

        #Read line by line and
        #get the lines corresponding to elements
        lines = get_lines_by_pages(layout)
        ele_lines, ele_line_indices = get_element_lines(lines,
                                           regs_all)
        #Style based detection
        if use_style:
                ele_lines, ele_line_indices = modify_element_
                lines_style(lines, ele_lines, ele_line_indices)

        #Add the metadata regexes
        regs_all_w_meta = regs_all + [re.compile(pattern)
                        for pattern in meta_regex_patterns]

        #Get the coordinates of the Element Selection Boxes
        ele_boxes = get_ele_Bboxes(lines, ele_lines,
                        ele_line_indices, meta_regex_patterns)

        #Get the selection box list
        selection_boxes = get_selection_boxes(layout,
                        ele_boxes, regs_all_w_marks)

        #In case of MCQ questions extract
        #the options in the selection boxes
        if question_type == 'MCQ':
                selection_boxes = extract_options(selection_boxes,
                                        mcq_reg)

        return selection_boxes
```

Here we would like to mention a small detail/caveat in the get_selection_boxes function. The pdf has certain dimensions and the coordinates are displayed in those dimensions, but the way we are displaying the PDF in the FrontEnd the size of the displayed PDF is fixed for uniformity. Thus the coordinates in the FrontEnd are different that those in the BackEnd. The aspect ratio of the PDF is maintained which gives us a way of interchanging the coordinates.

5 Implementation Details

The code for our project can be found here [6]. This section describes the major tools and software used in the various components of the project.

5.1 PDF Extraction

This is a python3 based implementation. The packages used are

- `PDFMiner`: Extract text and metadata from the PDF
- `PyPDF2`: Extract images from the PDF

5.2 BackEnd

The BackEnd server is implemented using Python Flask framework

5.3 FrontEnd

The FrontEnd web application is designed using Vue.js framework. The application is hosted on a node server(functionality provided by Vue itself). Additional npm packages used are

- `vue-js-modal`: Create a modal view
- `jquery`: Send AJAX requests to BackEnd server

6 Conclusion and Future Work

We studied and analyzed the previous work [1] and extended it by modifying the program at certain points where they didn't produce desirable outputs. We analyzed multiple PDF based question papers and tried to incorporate generalized features to the program with the aim of providing desirable outputs. We analyzed various intelligence based methods and went ahead with the regex based approach because of its simplicity and effectiveness.

We solve the major problem of providing the manual verification, by our Visual Editing Software implementation.

We recognize that working with PDFs can be quite difficult since most of the open source libraries are not well documented and do not work under all circumstances. Moreover, each library provides only certain functionalities. Building a tool to extract data from PDFs often requires the usage of multiple libraries.

6.1 Future Work

The tool we have built is to populate the existing Question Bank Database. So integration of the tool with the Question Bank interface is essential. The interface is built as a Django project which can redirect PDF Question Papers to our tool which can be hosted independently.

6.1.1 PDF Extraction

- Image extraction is incomplete, partly due to non existence of a single library to extract all kinds of images. Proper integration of various python packages can lead to a good image extraction model
- Extracting tables from the PDF. Python pdfplumber package can be used for extracting tables
- Extracting math equations from the PDF. For math equations, [3] can be used for math equation detection where as [4] can be used for math equation extraction to a latex code
- Incorporate format of paper into the existing code(ex:- QAQAQA, QQQAAA) for better detection of various elements
- Block Number based approach to detect selection boxes spread around multiple pages for a single element. Basically this approach assigns a block number to each selection box, and so the boxes spread across pages will have different

block number and the box with block number as 1 is the starting box for that element.This approach is more robust than the current method used.

- More robustness in style based answer detection code by trying out various other approaches
- Machine Learning based approaches like the ones mentioned in [5] to detect questions and answers in the paper. These approaches should be able to provide sufficient accuracy guarantees

6.1.2 FrontEnd UI

- Design an alternate approach for taking regex inputs so that users can enter them with ease. The current approach has limited select options and the input for 'Other' case is only taken as an exact regex
- Provide option to give multiple regexes as input for any element
- Current UI implementation is not fast and efficient, which is a necessity for a good software application. The UI has to be made light-weight for more user friendly experience
- Improve the display features of the current UI
- Sanity checks while passing on the inputs should be carefully implemented at necessary places

7 Acknowledgement

We wish to express our sincere gratitude to Prof. Kameswari Chebrolu for her constant guidance and support. We would like to thank Sai Sharath and Nagendra Reddy, who initiated this work and Vamsi Krishna and Tanvi Jadhav for helping us with FrontEnd and BackEnd implementations. Also, Yashasvi Sriram for his suggestions with UI related issues.

References

[1] Nagendra, Sai Sharath, " QUESTION BANK REPORT", RnD Project

[2] `https://github.com/jsoma/kull`

[3] `http://www.iapr-tc11.org/archive/icdar2011/fileup/PDF/4520b419.pdf`

[4] `https://snapcraft.io/mathpix-snipping-tool`

[5] Tamura, Akihira, Takamura, Hiroya, Okumura and Manabu. "Classification of Multiple-Sentence Questions". In Natural Language Processing – IJCNLP 2005 on, pg 426-437. IJCNLP 2005

[6] `https://github.com/huzzzz/QB-PDF`