***SOFTWARE DESIGN SPECIFICATION***

**1.0 Introduction**

**1.1 Goals and objectives**

* Currently, monthly audited financial data is being extracted manually.
* Our objective is to develop a software which will extract specific data values from multiple pdf files, which contain fiscal treasury data from various counties in Michigan. The software is designed to minimize the effort & make data extraction less prone to human errors. The extracted data will be organized in an efficient manner by the program for ease of navigation for further use.

**1.2 Statement of scope**

* Major Inputs
  + - PDF Files containing thousands of data values
* Processing Functionality
  + - Program will read in values from the top of the file
    - Ideally, the program should skip the unnecessary rows & pages
    - The program will get values that are within user’s specification , other values will be ignored
    - User’s specification is predefined within the program as instructed by the client.
* Major Outputs
  + - The program will output one master csv file containing extracted data organized for further analysis.

**1.3 Software context**

* The client has provided a list of desired data fields to be extracted.

**1.4 Major constraints**

* Another constraint (mutually agreed with client) is that the pdfs have to be readable/editable for successful parsing. The program cannot process text to image pdfs.
* A set time limit of the end of the summer semester

**2.0 Data design**

A description of all data structures including internal, global, and temporary data structures.

**2.1 Internal software data structure  
 File Submitter:**

File Submitter contains the following data objects:

* **FILE\_NAME -** Holds the name of the file submitted.
* **FILE\_PATH -** Holds the absolute path string of the file’s location.
* **RESULT\_PATH -** Holds the path to the resulting CSV file.
* **ROOT -** Holds the dimensions of the UI window to be opened.
* **TITLE -** Holds the title to be displayed for the UI window.
* **BUTTON -** Holds the command to be executed based on click.
* More - will be updated as we code the final build.

**File Parser:**

File parser contains the following data objects:

* **HEADER\_LIST -** Holds the specific category of the data to be extracted. These headers are written into the CSV.
* **CURR\_PAGE -** Holds the reference to the current page the parser is iterating over.
* **CURR\_TEXT -** Holds all textual data on the current page.
* **FILTERS -** Filters are used to keep track of various/multiple names the data appears as in the pdf.
  + **COUNTY\_FILTER -** Keeps track of counties that may be formatted as simply county names or possibly the word ‘county’ appended to the county.
  + **STATEMENT\_OF\_ACTIVITIES\_FILTER -** Keeps track of multiple formats which statement of activities may appear within the pdf.
  + **BUS\_ACTIVITY\_FILTER -** Keeps track of multiple formats which business activities may appear within the pdf.
  + **DATE\_FILTER -** Keeps track of all date formats appearing in the pdf.
  + **GOV\_ACTIVITY\_FILTER -** Keeps track of multiple formats which gov activity filter may appear within the pdf.
  + **TOTAL\_FILTER -** Keeps track of multiple formats which total may appear within the pdf.
  + **COMPONENT\_UNIT\_FILTER -** Keeps track of multiple formats which component unit filter may appear within the pdf.
  + More to be filled as we code the final build.
* **DF\_LINE -** Holds tags of all columns of data to be extracted.
* **CURR\_ LINE\_INDEX -** Holds the index of current line iteration on the page.
* **FULL\_PAGE\_TEXT -** Holds all textual data on this current page.
* **SPLITS -** Splits are used to hold data points which may appear as a total different name in the pdf.
  + **GRANTS\_ROW\_SPLIT -** Holds a reference to similar data points which may not be called GRANTS ROW.
  + **PROPERTY\_TAX\_SPLIT -** Holds a reference to similar data points which may not be called PROPERTY\_TAX.
  + **TOTAL\_GEN\_REV\_TRANSFER\_SPLIT -** Holds a reference to similar data points which may not be called TOTAL GEN REV TRANSFER.
  + **CHANGE\_IN\_NET\_POS\_SPLIT -** Holds a reference to similar data points which may not be called CHANGE\_IN\_NET\_POS.
  + More to be filled as we code the final build.

**2.2 Global data structure**

* **FINAL\_DATA -** This is a global component which appends all flagged parsed data from the PDF. It holds parsed data which then will be converted into a dataframe for permanent storage throughout execution.
* **Tkinter Root** - This global root controls all operations of the software. It handles events on click, on transitions, and on changes.

**2.3 Temporary data structure**N/A, As of now we are not using any temporary data structure.

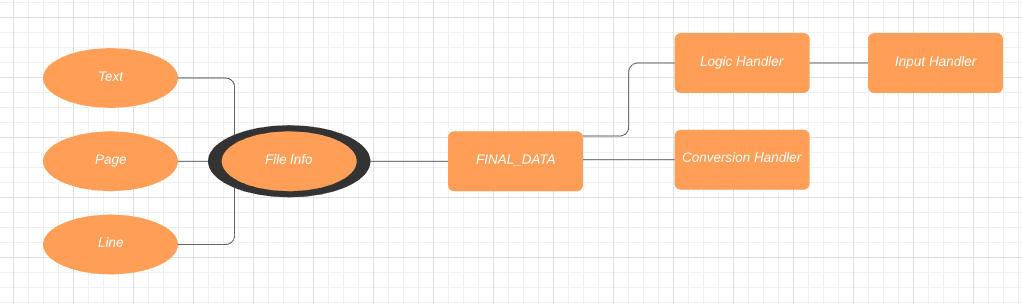
**2.4 Database description**

* **DATAFRAME -** We decided to use a dataframe rather than a database. It reduces steps in order to transfer data to a CSV.
  + **DATAFRAME\_COLUMNS:** Columns hold the specific type of data as displayed on the column header.
  + **DATAFRAME\_ROWS:** Rows hold a number of records of that data throughout the pdf.

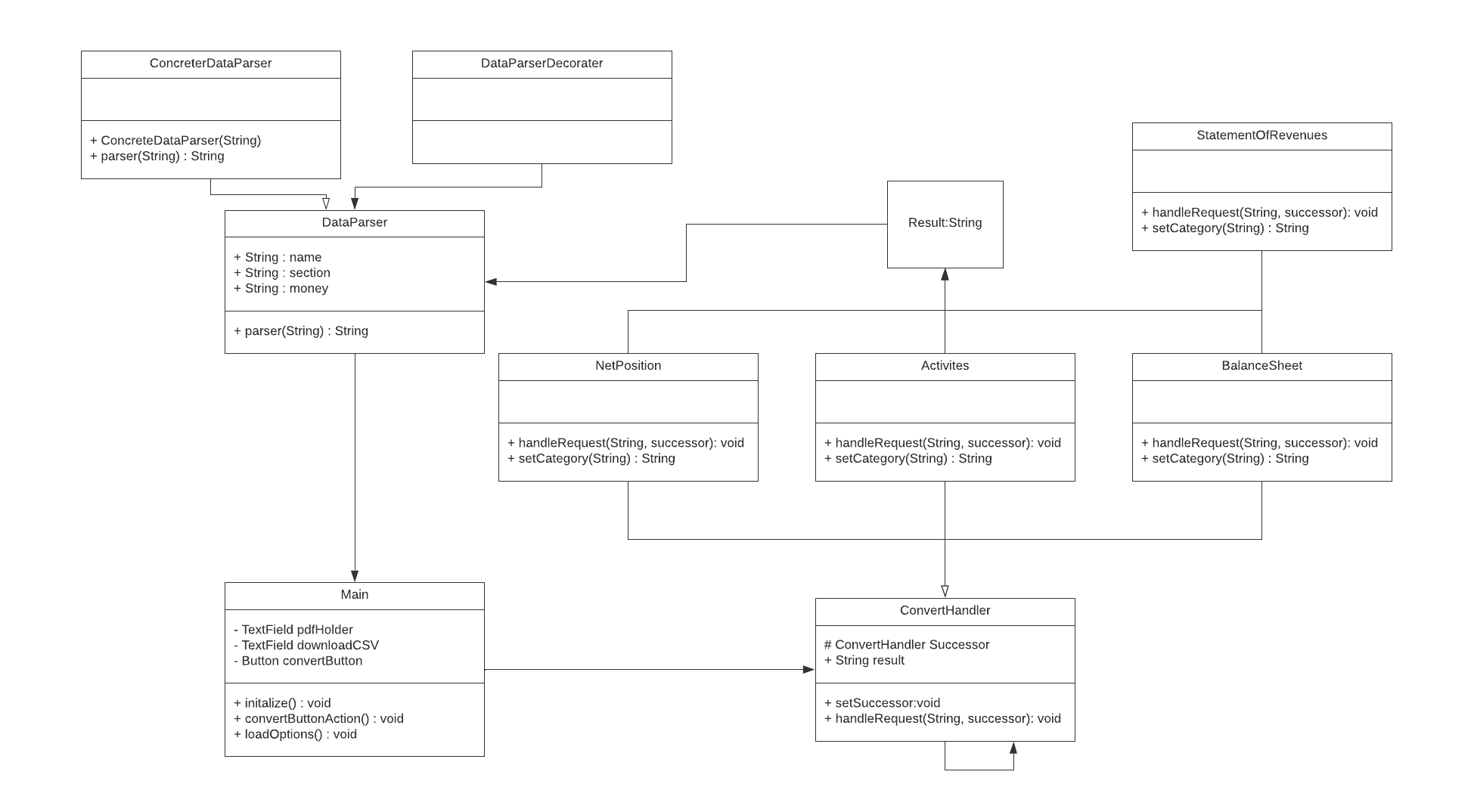
**3.0 Architectural and component-level design**

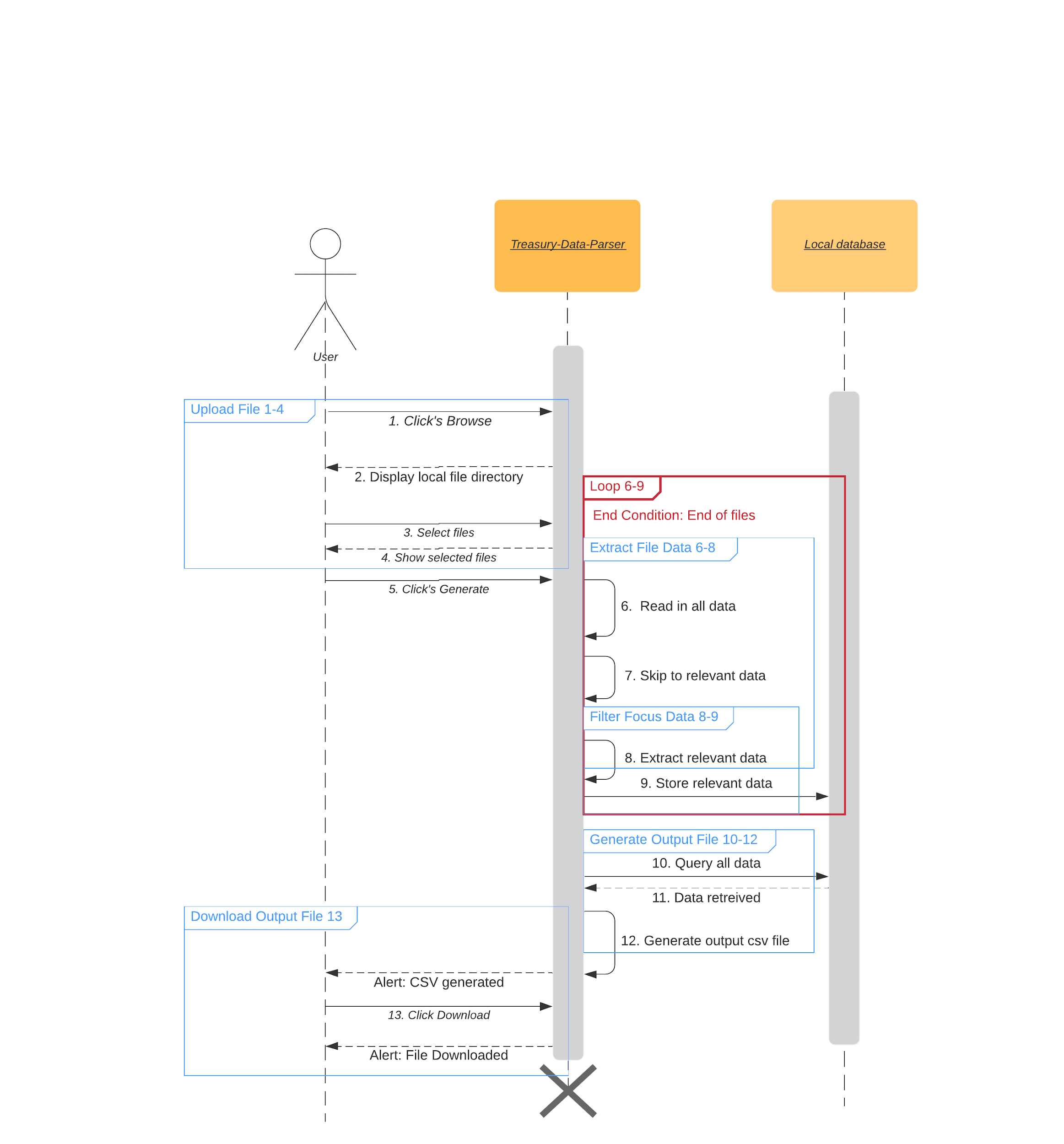
A description of the program architecture is presented.

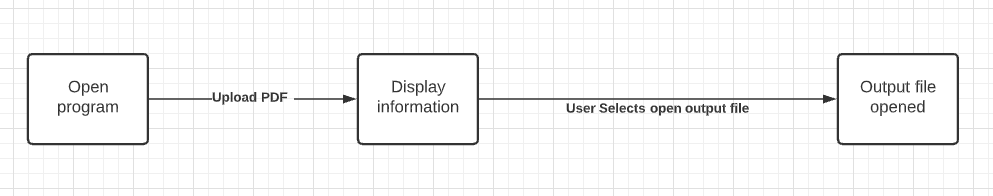
**3.1 Program Structure**A detailed description of the program structure chosen for the application is presented.

**3.1.1 Architecture diagram**

**3.1.2 Alternatives**

****

****

****

**3.2 Description for Component n**A detailed description of each software component contained within the architecture is presented. Section 3.2 is repeated for each of n components.

**3.2.1 Processing narrative (PSPEC) for component n**

**File Info:** A combination of all info gathered from the PDF

**Text:** The specific text at any given time being gathered

**Page:** The page that our program is currently scanning and gathering from

**Line:** The specific line that our program is reading

**Logic Handler:** Handles the logic of the program and makes sure that it keeps moving through the document

**Conversion Handler:** Makes sure that we are properly converting the PDF text into usable information

**Input Handler:** Handles the PDF that is being entered by the user

**FINAL\_DATA:** The heart of it all, this handles everything

**3.2.2 Component and interface description.**

**File Info:** Outputting all the information given by the PDF to be used by the Conversion Handler

**Text:** Outputs the text gathered from the file

**Page:** Outputs to the program which page we are currently on

**Line:** Outputs the current line our program is reading to be broken up

**Logic Handler:** Takes in all the information gathered from the file

**Conversion Handler:** Takes in all the information from the logic handler and outputs it onto a CSV file

**Input Handler:** Takes in the PDF itself

**FINAL\_DATA:** Takes in all the information of the program and outputs it back to the user

**3.2.3 Sub-Component n.m processing detail**

**File Info:** Gathers info from text, page, and line then sends it to FINAL\_DATA

**Text:** Gathers text

**Page:** Gathers page number

**Line:** Gathers current line of PDF

**Logic Handler:** Takes input PDF and sends it off to FINAL DATA

**Conversion Handler:** takes FINAL DATA information and formats it, sends back CSV file to FINAL DATA

**Input Handler:** Takes in a PDF from user and sends it ot Logic Handler

**FINAL\_DATA:** Gathers all things and sends all things.

**3.2.3.1 Interface description**A description of sub-component m inputs and outputs is presented.

**File Info:** No display

**Text:** No display

**Page:** No display

**Line:**No display

**Logic Handler:** No display

**Conversion Handler:** No display

**Input Handler:** No display

**FINAL\_DATA:** Displays the CSV download to the user

**3.2.3.2 Restrictions/limitations**

**File Info:** PDF must be correctly formatted

**Text:** PDF must be correctly formatted

**Page:** PDF must be correctly formatted

**Line:** PDF must be correctly formatted

**Logic Handler:** PDF must be correctly formatted

**Conversion Handler:** PDF must be correctly formatted

**Input Handler:** PDF must be correctly formatted

**FINAL\_DATA:** PDF must be correctly formatted

**3.2.3.3 Performance issues**Information on topics that may affect the run-time performance, security, or computational accuracy of this sub-component are presented.

**File Info:** N/A

**Text:** N/A

**Page:** N/A

**Line:** N/A

**Logic Handler:** N/A

**Conversion Handler:** N/A

**Input Handler:** N/A

**FINAL\_DATA:** N/A

**3.2.3.4 Design constraints**

**File Info:** Needs a properly formatted PDF

**Text:** Needs a properly formatted PDF

**Page:** Needs a properly formatted PDF

**Line:** Needs a properly formatted PDF

**Logic Handler:** Needs a properly formatted PDF

**Conversion Handler:** Needs a properly formatted PDF

**Input Handler:** Needs a properly formatted PDF

**FINAL\_DATA:** Needs a properly formatted PDF

**3.3 Software Interface Description**The software's interface(s) to the outside world are described.

**3.3.1 External machine interfaces**

N/A This will only be held on one machine

**3.3.2 External system interfaces**

N/A This will be a self sufficient program

**3.3.3 Human interface**

2 Screens will be made for users to view and interact with.

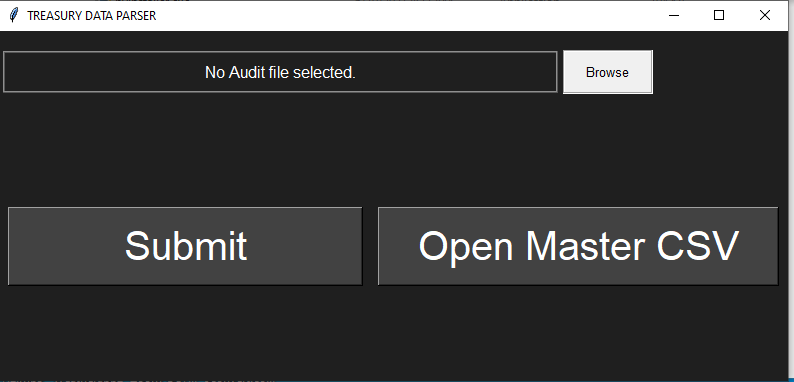
**4.0 User interface design**

A description of the user interface design of the software is presented.

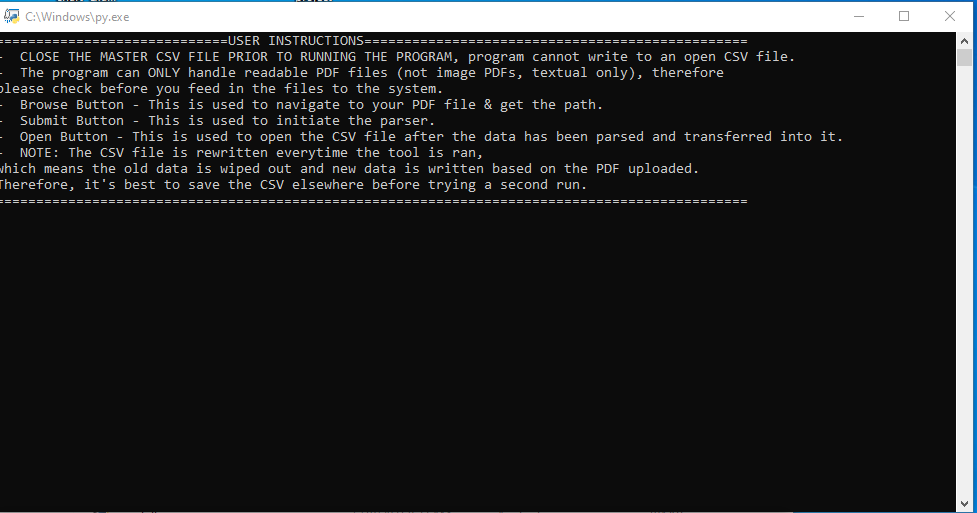
**4.1 Description of the user interface** We should have only 2 screens in our final program

**4.1.1 Screen images**

The image below is the user interface for selecting a pdf file through windows file explorer , submitting the file and opening the master output file .



The image below shows a console window of the program showing a trace of the data extraction as requested by the client.



**4.1.2 Objects and actions**

* 1st Screen
  + Browse button: Will allow the user to search their computer directory for the PDF to enter.
  + Submit button: Will start processing the data of the imputed PDF.
  + Open master output file: will allow the user to open the CSV output file.
* 2nd screen
  + This a console window of the program trace.

**4.2 Interface design rules**

* Python -> TKinter Module was used to create UI screens.
* Easy to Learn
* Readability
* Easy navigate between interfaces

**4.3 Components available**

* TextBox
  + Displays information entered at design time, entered by the user
* Label
  + Graphical area that a user won’t be able to change directly
* Line
  + Displays a line of text
* CommandButton
  + A button used to invoke an action out of the program directly, such as finding a file in your computer when you press the browse button.

**4.4 UIDS description**

The user interface will be developed based on what our code needs, if our code needs a certain file then it will ask the user for it. It will be designed with simplicity and absolute no chance of messing it up with an average user in mind. Only the necessity of information will be displayed as users will not need anything else past the CSV from the PDF that they have entered.

**5.0 Restrictions, limitations, and constraints**

* **Resource Limitation**
  + Not having appropriate libraries or modules is a limitation for us. During the early stages of prototype development, we were looking for modules which could be helpful for parsing a PDF, however there aren’t many in the market. We are limited to one or two which are open & free to use. The third party resources we found offer limited functionalities since they are still under full development.
* **Team Skills**
  + Not having much coding experience with this sort of tool among us as a team is a set back. Plus due to limited resources offered in the market, we are constrained to using one programming language, which most of us are not fully comfortable with.
* **Input File Constraint**
  + We are constrained to use one PDF file provided by the client. Our software is pretty much relying on that one PDF to measure successful parsing. As mentioned by the client, the software must successfully function with the given PDF for approval. Ideally, the software should handle other PDFs. But we are limited to code based on the formatting of the given PDF file.

**6.0 Testing Issues**

Testing is very essential to mark off the final product as successful. Our tool is a very simple tool in terms of a user’s perspective. However it may be the most complicated in terms of back-end parsing, it is a data heavy software. Therefore, we must perform tests to make sure that the data being grabbed is also being transferred accurately in the CSV. The tests we will be performing are listed below.

**6.1 Classes of tests**The types of tests to be conducted are specified, including as much detail as is possible at this stage. Emphasis here is on black-box and white-box testing.

* **Unit Testing**
* We are creating a data parser in order to retrieve treasury data from a compacted PDF into a well formatted CSV. Prior to this, the clients have had no such software to do it. To provide a successful software, we must test each data point being grabbed from the PDF.
  + White Box Testing
    - Our developer will be testing each data point line by line to ensure that parsing is being accomplished accurately. Each component, the parser and the user interface will be tested individually by two individual developers.
* **Final Integration Testing**
  + For the final integration testing, we will first integrate the UI and the back-end parser. Then we will run multiple test runs to ensure that the inputs from the UI are being taken care of by the back end parser. We will run integration tests repeatedly at each component merge to ensure successful integration.

**6.2 Expected software response** The expected results from testing are specified.

* User Interface
  + The inputs from the user are successfully being processed as expected. In other words, user interaction with the UI is fully functional and behaviour is as expected.
    - Ex: Upload button actually helps with feeding the file into the software.
* Parser
  + The parsed page data from the respective page must have an exact match in the CSV file, unless the data is categorized as explicitly specified by the client. The software should only parse flagged data, other data should be ignored/skipped.
* Table Rows
  + The program must properly convert data found in table rows from the readable pdf files into distinct columns. We will test this functionality by comparing various inputted table rows to the outputted excel row.
* Clean Rows
  + The program must remove loose dollar sign ($) symbols extracted from table rows so as to not interfere with numerical figures. If the row contains a dollar sign, the program will loop through and remove all singular, loose dollar signs from the row. We will test this by comparing the initial row list to the resulting row list at the end of the loop. Different rows will be tested to ensure the functionality both operates on dollar sign-containing rows and refrains from modifying rows without such character instances.
* Combine Row Field Names
  + The program must combine each row’s field name such that the full name is stored in a singular list element. The program will loop through the elements of each row list and locate the first element that begins with a numerical character. All list elements up until the aforementioned element will then be combined, forming the full row field name. We will test this by comparing the initial row list to the resulting row list at the end of the loop. Different field names of different lengths will be tested.
* Combine Separated Numbers
  + The program must recombine numerical figures with any numerical characters that become separated from their parent figure as a result of the extraction process. The program will iterate through each row list element and locate any singular digits whose near-adjacent elements are also numbers. We will test this by comparing the initial row list to the resulting row list at the end of the loop. Different rows will be tested to ensure the functionality both operates on separated numbers and refrains from operating on non-separated numbers.
* Store Clean Rows
  + The program must have a standard means of splitting a row string into a list, cleaning and combining the list, and storing the list in an overarching list of cleaned rows. We will test this by comparing the initial row string to the resulting element in the overarching row list.

**6.3 Performance bounds**

*Parser Performance Requirements*

* Users must be using one of the recent windows operating systems with supported tools.
* Users must have the latest version of python installed/or be willing to install with the software.
* Parser must not take any longer than 5 minutes to parse.

**6.4 Identification of critical components***Parser*

The parser is the main component of our software. The parser does the job of reading the actual PDF file fed into the software, then taking important data points from that PDF and storing it elsewhere in a useful manner. The parser component also handles writing to the actual CSV file from the storage (dataframe). Therefore, parser is one of the critical components of our software.

**7.0 Appendices**

Presents information that supplements the design specification.

**7.1 Requirements traceability matrix** - Attached as a separate document.   
**7.2 Packaging and installation issues**

* **NO Issues, however not having the following may cause failure to run the software.**
* The user must install the latest version of python on their PC for a successful run of the software.
* The user needs to install pdfplumber, this is a module we used for our parser.

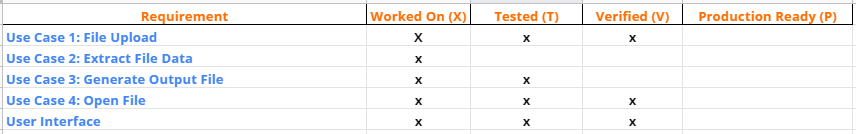
**7.3 Design metrics to be used** - % Maintainability

- % Client Satisfaction

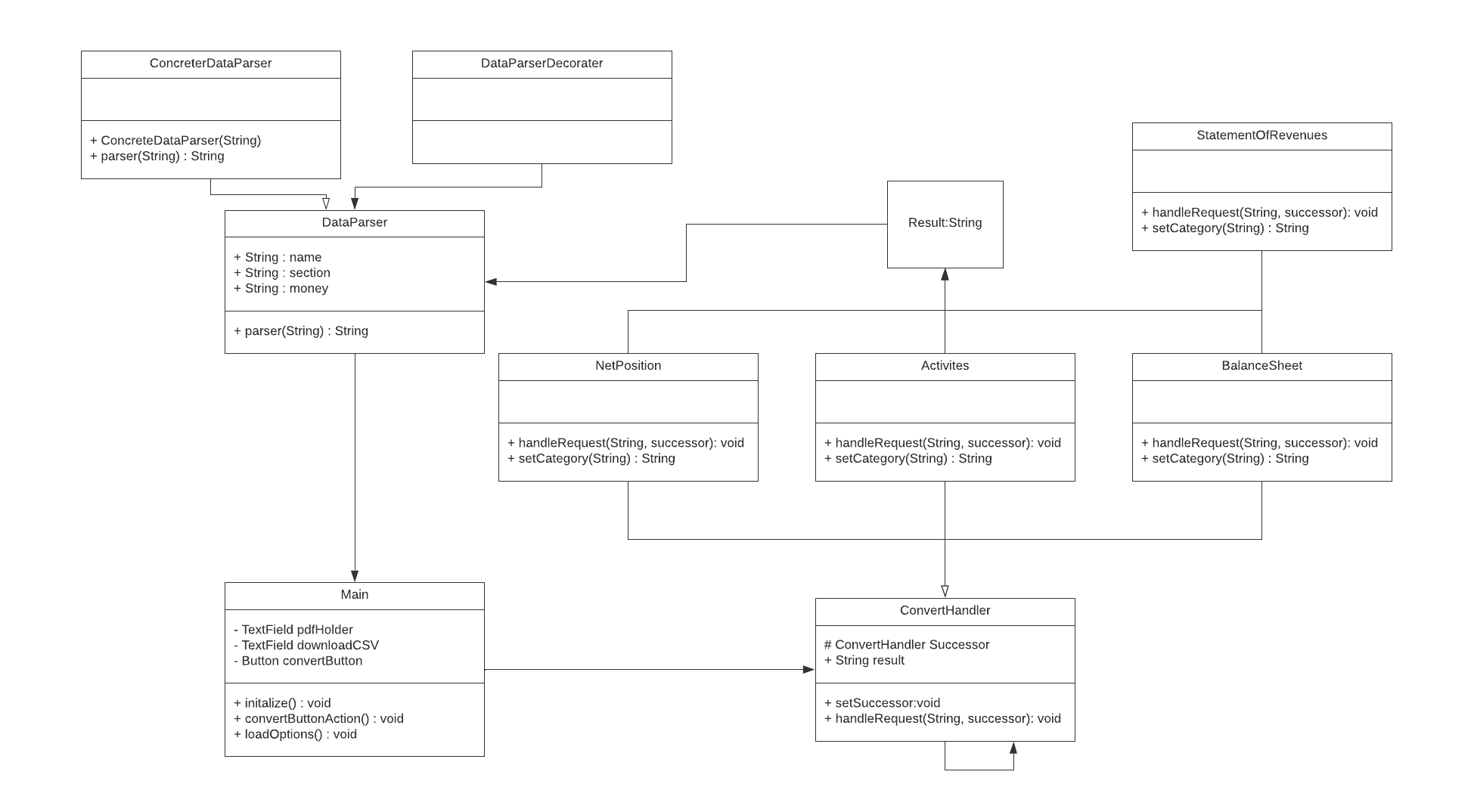
- # of Requirements Satisfied

**7.4 Supplementary information (as required)**

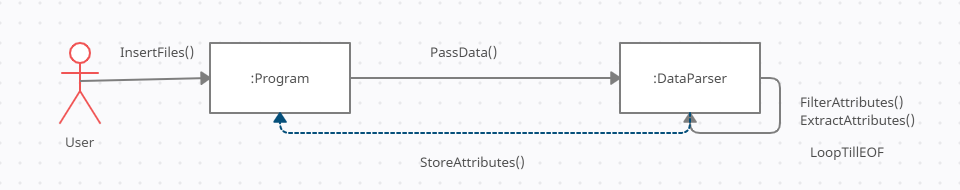
**Traceability Matrix**

****

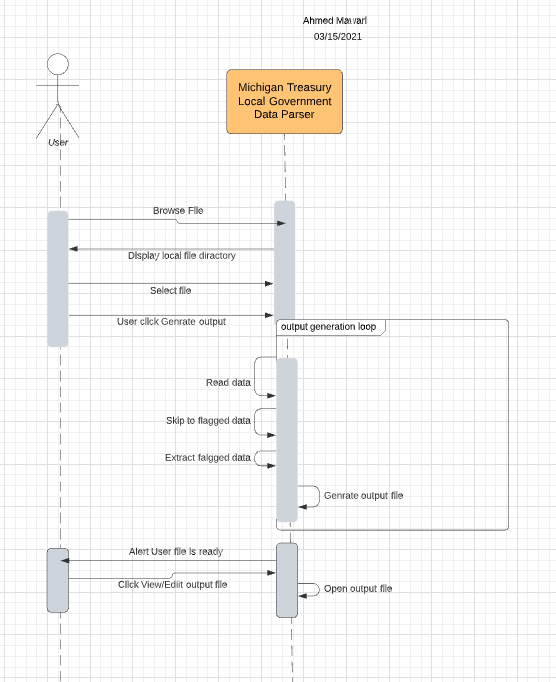
**Data Model**

****

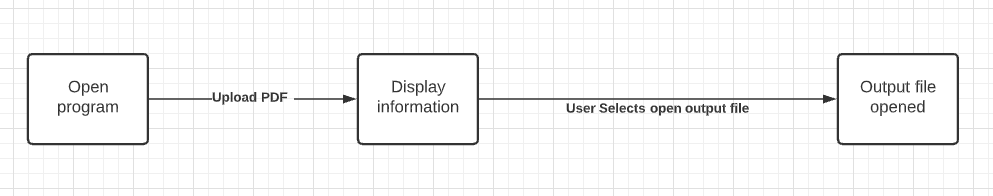
**Communication diagrams**

****

**Sequence Diagram**

****

**Activity Diagram**

****