La Trobe University

2021 CSE3PRA System Maintenance Document

PREPARED FOR

Dr Laura Rusu, CEO, LENSELL

SUPERVISED BY

Dr Scott Mann
Derek Nguyen

Copyright and Warranty Information

All information contained herein is subject to change without notice.

The vendor assumes no liability, nor accountability for any errors that may be present within this Document.

No warranty, nor representation, either expressed or implied, is made with regard to the quality, accuracy, or fitness of any part of this document. The manufacturer shall not be liable for any direct, indirect, special, incidental, or consequential damages arising from any defect or error in this document or product.

Product names appearing in this document are for identification purposes only. Trademarks and product or brand names appearing in this document are the property of their respective owners.

Copyright on all material and all intellectual property residing in the material in this document, including (but not limited to) graphics, book titles, text, layout, logos, trademarks, and samples is owned by LENSELL GROUP PTY LTD and LA TROBE UNIVERSITY unless otherwise indicated.

No reproduction, modification, translation, transmission, distribution, or use for commercial purposes of this document without written approval from LENSELL GROUP PTY LTD and LA TROBE UNIVERSITY.

This document contains material protected under International Copyright Laws. All rights reserved.

These terms are subject to the conditions prescribed under the Australian Copyright Act 1968.

Financial Records Project
2021 Industry Project
Department of Computer Science & Computer Engineering
La Trobe University, Bundoora, Victoria, Australia

Document Version Control

Version.	Date	Description
1.0	07/06/2021	CSE3PRA Version
2.0	25/10/2021	CSE3PRB Version

Document Properties

Items	Details
Document Title	System Maintenance Document - Lensell
Author	4BDAW
Creation Date	15/05/2021
Last Updated	25/10/2021

Table of Contents

Table of Figures

1.0 Introduction

The Portable Document Format (PDF) was developed by Adobe in 1993 as a solution to the ever-increasing problem of page layout replicability across different visual-based systems. They were able to design a document format, based on the PostScript language that displays text formatting and images in a manner that is independent of application software. For this reason, PDF quickly became popular within industries where visual layout consistency was paramount, such as newsprint and publishing. However, this growth was not limited to these industries.

Up until 2008 PDF was the proprietary property of Adobe, at which point they released it as an open standard and allowed royalty-free usage. With this new openness and its consistent formatting across software systems, PDF quickly became the go-to digital replacement for paper across many industries, including but not limited to the government and the financial sectors. Following this move away from paper and towards document digitalisation, there is now a wealth of data stored in PDF. However, there is a problem: the very thing that made PDF so popular as a paper replacement, its visual consistency, makes it very difficult to extract this data.

Adobe could not have predicted how ubiquitous PDF would become, nor the types of data that would come to be stored within it in the years following their designing it in the early 1990s. Adobe solved the page layout inconsistency problem by designing a dynamic visual representation document, based on spatial coordinates rather than pages-lines and margins, as text documents are normally structured. While this was a brilliant choice for visual consistency, unfortunately, it has led to the current problem we face in this project: extracting business report data from company yearly financial reports.

In general, digital document data parsing comes in two flavours: text-extraction or meta-framework analysis. Digital text extraction is the process of parsing data in the same way one would read any document; it takes each line and extracts the text present from left to right. A meta-framework analysis is a process of using the underlying digital logical framework to identify and locate specific elements and structures within the document. While both these methods are reliable for most digital document types, they struggle against PDF due to its inherent lack of correlation between its underlying logical framework and its visual page layout presentation.

This system aims to solve this extraction problem for our client by reversing the design process that Adobe followed in creating PDF: where Adobe set out to create digital paper to store information, we are setting out to create a digital reader to extract information.

To this end, our system utilises a neural net classifier, along with Python data extraction libraries to replicate the natural visual and physical systems we as humans use when viewing and collecting data from documents ourselves.

1.1 System Overview

The client approached us with the task of developing an application with will enable the user to load company performance documents in the form of PDF into a browser, pass it on to an extraction engine that will extract the relevant data from the company financial statements with the PDF, match the extracted items to a given taxonomy and return the list of matched data to the user.

1.2 Audience Description

The expected users of our system will be:

Investors and interested stakeholders (users): refers to users wanting to use the system for its primary intended function, which is to extract financial data from company performance report documents. These users are not expected to have in-depth knowledge of how the system works, they should be able to use the full functionality of the system with little to no training or guidance.

Researchers & Academics: this is in reference to any individual who wants to explore the capabilities of the extraction engine produced, and configure it to their desire. This set of users are expected to have a deeper level of understanding of the software, based on the System Maintenance Document produced and from the manner the code has been produced.

The system caters to two types of users:

1. Website user
2. API users

Admin: refers to user requiring sysadmin database management access

1.3 Applicability Statement

The software currently passes all required backend user stories presented by the client in the requirement outline.

The back-end server is developed in Django and is both accessible via a user interface developed in Angular, which has basic upload, extract and download functionality, and via an API which provides full access to all system functionality,

1.4 Purpose Statement

The purpose behind creating this software is in response to the ever-expanding digitalisation of company financial and performance reporting documentation. These documents are readily available online but are difficult to extract as they are most commonly published in Portable Document Format (PDF) and with page counts in excess of 100.

This software aims to significantly reduce the time and effort required by investors to search for the financial performance metrics and extract those figures manually, by providing an automated solution for parsing and extracting the relevant data within these large and cumbersome

documents and providing it to the user in an easy to use format, specifically in Comma Separated Values (.CSV) or easy use and compatibility with common spreadsheet software

1.5 Document Usage Description

Describe what each section of the document contains, its intended use, and the relationship between sections. Also, provide any other directions necessary for using the document.

The document contains the following sections:

1. Introduction
    - ➥ An overview of the aim, purpose, application, and the intended users of this project.
2. Software Design Scope
    - ➥ Describes major software functions with their constraints, requirements, and development risk.
3. Reference Documents
    - ➥ Lists all supporting documentation. For instance, the project description document and all user documentation related to the project.
4. User Stories
    - ➥ Describes the goals/use cases of the project, which illustrate what the project plans to achieve in its end functionality. Also includes testing to verify the use and reliability of these functional goals.
5. System Design
    - ➥ Conceptually presents and describes each key System area of the development plan for the project.

        Key areas, such as:
        - ➥ API and extraction engine architecture
        - ➥ Database and System storage architecture
        - ➥ System logging
        - ➥ Use-case modeling
6. Software Release Report
    - ➥ Details final release notes and testing
    - ➥ Outlines the project testing procedure and methodology
7. Additional Designs and changes
    - ➥ Contains additional changes implemented through discovery during both the design and development stages.

1.5.1 Acronyms and Abbreviations

API - Application Programming Interface
AWS - Amazon Web Services
CSS - Cascading Style Sheets
CSV - Comma Separated Values
HTML - Hypertext Markup Language
MP - Multiprocessing
OD - Object Detection
SMD - System Maintenance Document
SQL - Structured Query Language
UI - User Interface
UX - User Experience
YOLO - You Only Look Once (OD)

1.6 Conventions

1.6.1 Documentation

In preparing this document, we have employed key design conventions to improve both general readability and understanding, as well as searchability for referencing specific sections and information herein.

Using a decimal numbering convention, we have presented the system from a top-down structural perspective; separating each section and subsection therein, along with a table of contents for and an overall outline and navigation to areas of interest.

We have employed the color dark-blue for headings throughout, to more easily differentiate between sections.

We have labeled each figure with a figure number and caption with a brief description of its contents; in addition to a 'Table of Figures' following the table of contents for functional navigation and ease of reference.

Lastly, Arial font is used throughout, with paragraphs written in black, font size 11; headings written in dark-blue and a mix of font sizes 14, 18, 20; and figure captions written in dark-blue, font size 11.

1.6.2 Code Base

For a given language, the System code base conforms to the following conventions.

Angular

We will be following the Angular coding style guide

Python

We will be following the PEP 8 -- Style Guide for Python Code

Naming conventions:

Type	Naming style	Requires docstring
variables	snake_case	no
functions	snake_case	yes
Classes	PascalCase	yes

Figure 1.6.2.1: Python naming convention table

2.0 Software Design Scope

2.1 System Functions

Users

- Upload PDF document for data extraction.
- Download extracted data from PDF document.
- Retrieve previous documents and extractions.

System

- Take in PDF documents and store them locally.
- Extract data from locally saved PDF documents to CSV format.
- Store extractions locally.
- Update database.
- Send extracted data externally to the user.

2.2 Requirements

- Utilise Angular 11 or 12 for front-end development
- Utilise Python 3.7 or 3.8 and Django for back-end development

2.3 Design Constraints

- No user involvement in the extraction process beyond uploading documents and downloading results.

2.4 Risks

	Consequences				
Likelihood	**Insignificant**	**Minor**	**Moderate**	**Major**	**Severe**
Expected	Medium	High	High	Extreme	Extreme
Likely	Medium	Medium	High	Extreme	Extreme
Possible	Medium	Medium	High	High	Extreme
Unlikely	Low	Medium	Medium	High	High
Rare	Low	Low	Medium	High	High

Figure 2.4.1: Risk matrix

ID	Risk Description	Likelihood	Impact	Severity	Mitigation action	Contingent action
1	Project purpose and needs not well defined	**Possible**	**Major**	**High**	Complete user stories and use-case analysis	Escalate project review and re-assess approach.

2	Flawed system architecture design	**Possible**	**Major**	**High**	Continual system testing on each respective domain of the system	Escalate system architecture review, locate the problematic domain and re-assess approach.
3	Project Schedule is not clearly defined, understood	**Unlikely**	**Moderate**	**Medium**	Review and assess schedule at each milestone	Revisit the schedule with the project team and relaunch the schedule with a revised timeline
4	Unplanned work that must be accommodated	**Expected**	**Minor**	**High**	Review tasks and workloads at weekly team meetings	Escalate delegations of needed works ad-hoc; review and re-assess works at the following meeting
5	Lack of communication, causing lack of clarity and confusion.	**Likely**	**Major**	**Extreme**	Outline a communication plan which includes: the frequency, goal, and audience of each communication. Use the most appropriate channel of communication for the audience.	Correct misunderstandings immediately. Identify the cause of communication breakdown and rectify to prevent further errors.
6	Scope creep	**Possible**	**Severe**	**Extreme**	Document the project scope, with clear boundaries of requirements	Escalate a re-evaluation of current project requirements and relaunch with reduced scope
7	Unresolved project conflicts	**Unlikely**	**Moderate**	**Medium**	Regular team meetings and watch for conflict signals	Escalate a meeting with all member to resolve conflict immediately
8	Delay in development threatens milestone delivery deadlines	**Possible**	**Major**	**High**	Ensure accurate project plan and development schedule. Identify issues early and present at meetings	Escalate milestone delivery schedule and relaunch with updated scheduling
9	Extraction engine design flawed	**Possible**	**Moderate**	**High**	Ongoing testing and review of the approach	Escalate design review and analysis, pivot design to the best-case candidate

Figure 2.4.2: Project risks table

3.0 Reference Documents

3.1 Existing Software Documentation

Existing software documentation consists of the project requirement documentation provided by Lensell Pty Ltd.

3.2 Vendor Documentation

Angular

Angular is used for all our front-end web user interfaces.

Angular is an open-source software engineering platform for building front-end web user interfaces. Angular combines declarative templates, dependency injection, end-to-end tooling, along with integrating best practices to solve both development challenges and ensure maintainability. While open-source, it was originally introduced by Google and continues to be led by the Angular Team at Google and by a growing community of individuals and corporations

Documentation - Angular Documentation

Python

Python is used for our API Server, as well as our extraction engine.

Python is an interpreted, object-oriented, high-level programming language with dynamic Semantics and its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development. Python has a simple and easy-to-learn syntax, which emphasizes readability and therefore reduces the cost of program maintenance. It also supports modules and packages, which encourages program modularity and code reuse.

Documentation - Python Documentation

- **Django**

  Django is used for our back-end framework, on which we have both our API and extraction engine implemented.

  Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

  Documentation - Django Documentation

- **Django REST framework**

Django REST framework is used as our API framework for system communication between the back-end, front-end, and database frameworks.

Django REST framework is a powerful and flexible toolkit for building Web APIs.

Documentation - DRF Documentation

- **YOLOv3**
  YOLO is used as part of our extraction engine to locate table data within PDF pages and quantify them as page location coordinates.

  YOLO (You Only Look Once) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO is implemented using the Keras or OpenCV deep learning libraries. Object classification systems are used by Artificial Intelligence (AI) programs to perceive specific objects in a class as subjects of interest. The systems sort objects in images into groups where objects with similar characteristics are placed together, while others are neglected unless programmed to do otherwise.

  Documentation - YOLO Documentation

- **Camelot-py**
  Camelot-py is used as part of our extraction engine to extract data from PDF pages in tabular format and parse it to other document formats.

  Documentation - Camelot Documentation

SQLite
SQLite is used for our relational database

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine and is the most used database engine in the world. The SQLite file format is stable, available cross-platform, backwards compatible, and has long-term support planned until 2050.

Moreover, SQLite source code is in the public-domain and is free to everyone to use for any purpose.

Documentation - SQLite Documentation

3.3 Other Documentation

No prior documentation exists for this project.

4.0 User Stories

4.1 User Story Dictionary

Role: Investor and interested stakeholder (user)

User Story ID	User Story	Acceptance Criteria	System	Website	Story Points
US-1	As the user of the System, I would like to upload a document from my local device	• The website/system is accessible • There is a navigable portable for uploading documents/ upload through command line	Yes	Yes	**5**
US-2	As the user of the System, I would like to use the automation aspect of the system to extract PDF document data, which conforms to the data structures within the original document	• The website/system is accessible • There is a navigable portable for uploading documents • There is a navigable portable for downloading document data with a button to download said data/ the extracted data are automatically saved. • The data presented to me has a high degree of meaningfulness	Yes	Yes	**30**
US-3	As the user of the System, I would like to select the range of pages to scan and extract data from the report rather than the whole report	• The website/system is accessible • Upon mentioning the range of page numbers the scanning and extraction will take place for the specified page number range. • The data is presented to me has a high degree of meaningfulness	Yes	No	**5**
US-4	As the user of the System, I would like to save the current extraction/report	• The website/system is accessible • A message is shown on the success of file processing completion • There is a navigable portable for downloading document data with a button to download said data/ the extracted data are automatically saved. • Upon clicking the download data button I receive the data locally on my device.	Yes	Yes	**5**
US-5	As the user of the System, I would like to	• The website/system is accessible.	Yes	No	**5**

	view the previous extractions/reports	• There is “Extracteds” section to view the previous extractions made from a report • There is “Reports” section to view the reports previously extracted			
US-6	As the user of the System, I would like to retrieve previous extractions/reports.	• The website/system is accessible. • Upon viewing the list of “extracteds” and “reports”, the desired files can be downloaded .	Yes	No	**5**
US-7	As the user of the System, I would like to delete the previous extraction/report.	• The website/system is accessible. • Upon viewing the list of “extracteds” and “reports”, the user can select the desired files to be deleted • Upon clicking a delete button, selected files will be removed from the storage/database.	Yes	No	**5**

Figure 4.1.1: User Story Dictionary for investor/stakeholder

User Story Definitions

Document: Any business performance/metric reports written in PDF format.

Document data: Data that has been extracted from the uploaded document and stored in CSV format.

Degree of meaningfulness: The measure of how closely extracted document data matches and retains the original document’s data structures.

4.2 Flow of Interaction

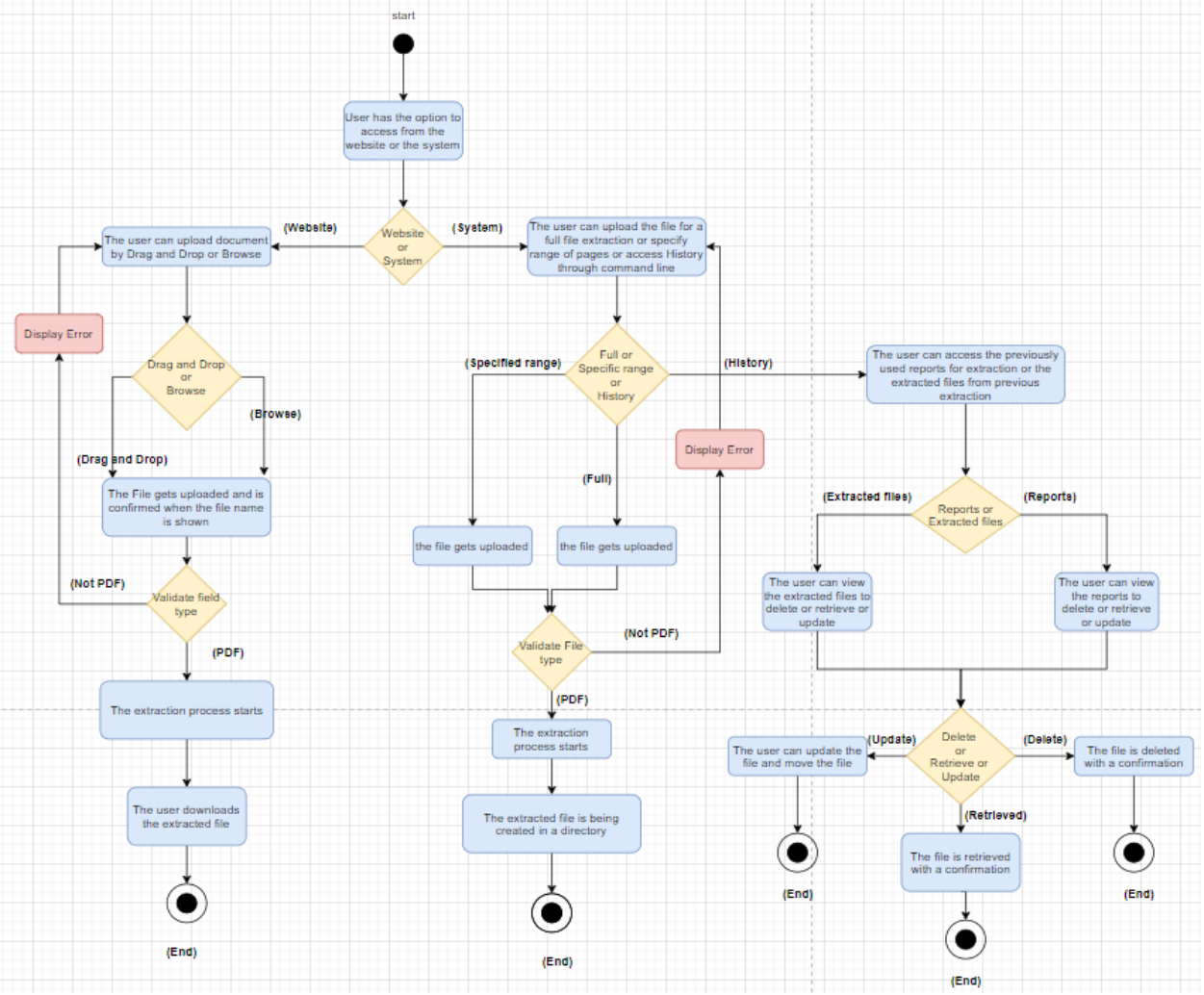


Figure 4.2.1: Flow of interaction diagram

4.3 UI/UX

4.3.1 Wireframes

Initial Design

Following the client's requirement, the system UI development has followed a function over form design paradigm.

To this effect, we have chosen a minimal user interface with minimal interactive interfaces. This is done to help ensure the user does not accidentally click anything that may affect the extraction process, as the extraction process requires the browser to stay active during processing. We have also implemented a feature where the user is presented with the data

found and given the choice to select the tables wanted for extraction and the following download.

Functionally, we chose to utilise toggle-switches for both their visual appeal and for their simple and clear user interaction. We also chose to implement vertical scrolling over horizontal or a sliding show feature to represent extracted table data - this was chosen in favour of desktop users, but further design additions can be incorporated for other platforms where necessary.

Initial Wireframes:

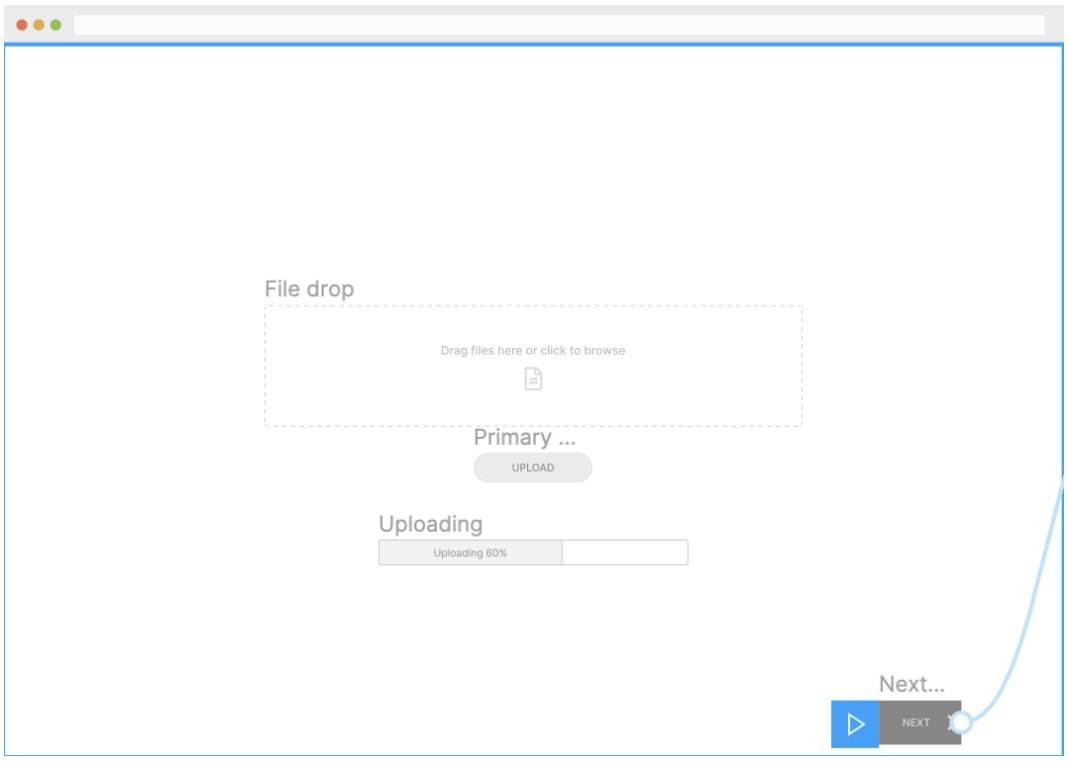


Figure 4.3.1.1: Initial wireframe 1

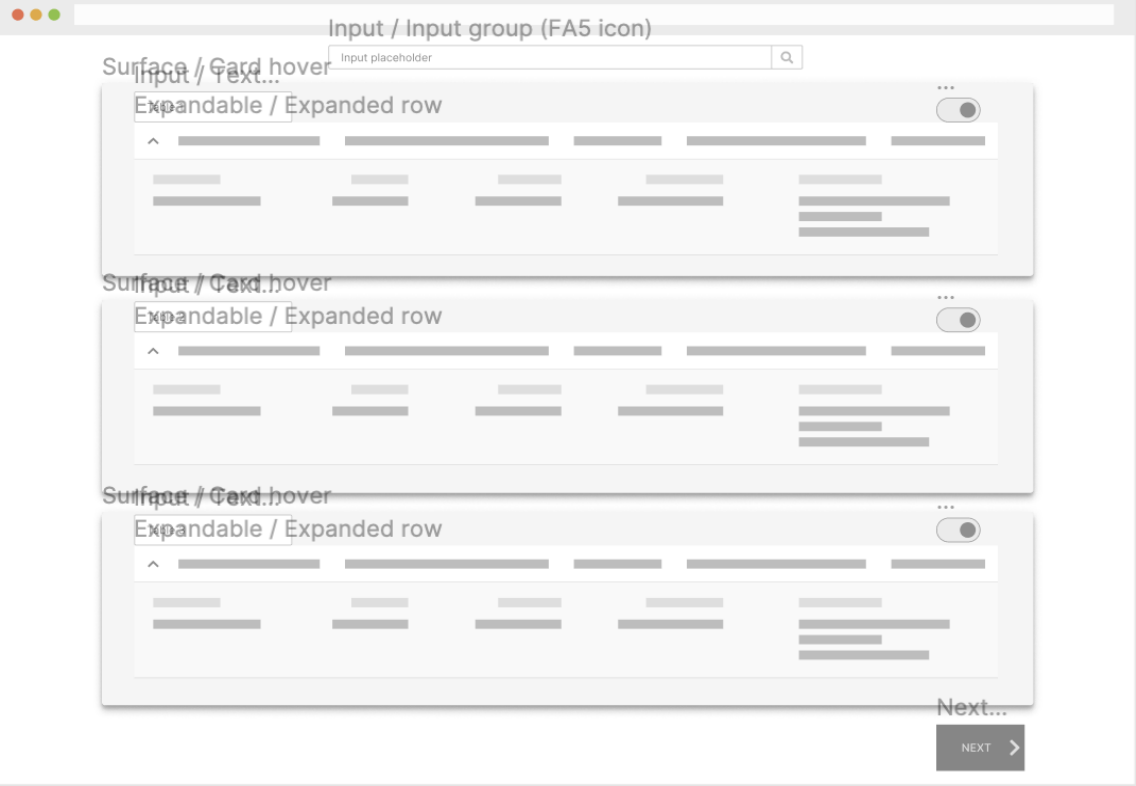


Figure 4.3.1.2: Initial wireframe 2

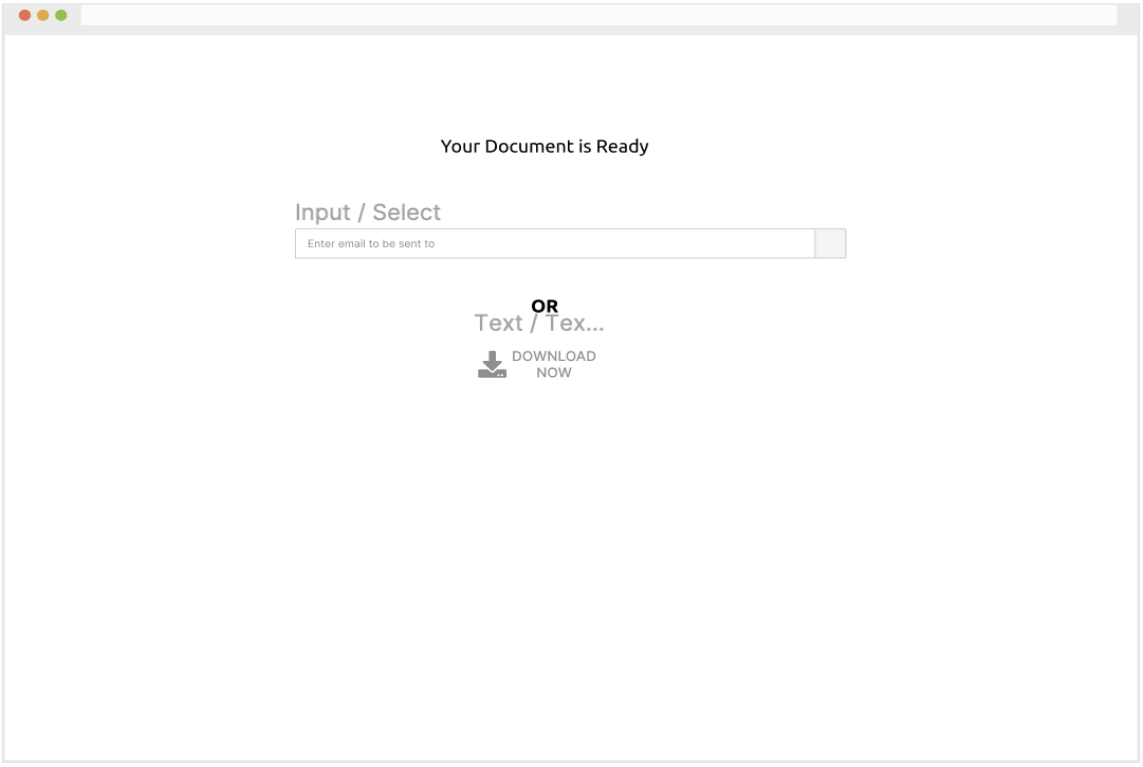


Figure 4.3.1.3: Initial wireframe 3

Low Fidelity Wireframe

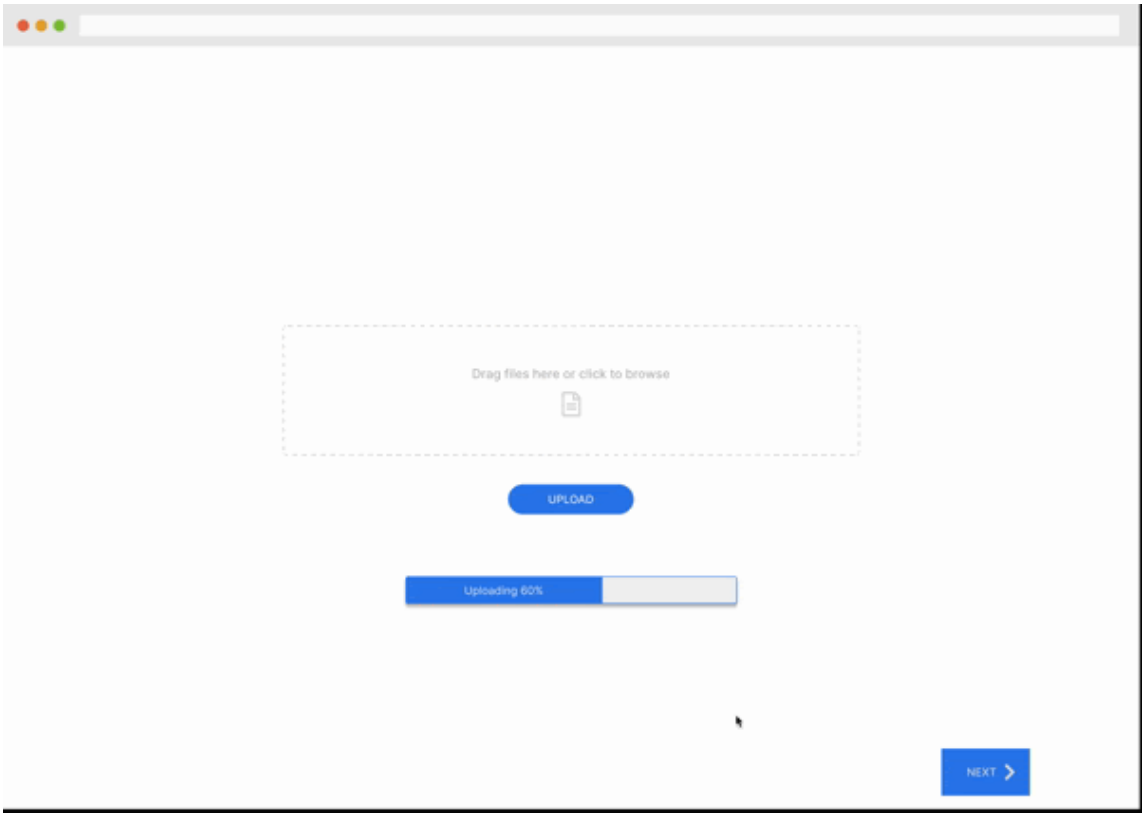


Figure 4.3.1.4: Low fidelity wireframe

Planned High Fidelity Wireframe

Following a positive client response to the initial UI design choices, we further developed the concept and incorporated the client company branding to provide an on-brand visually aesthetic feature. Two such examples incorporating Lensell branding were created for the design proof of concept.

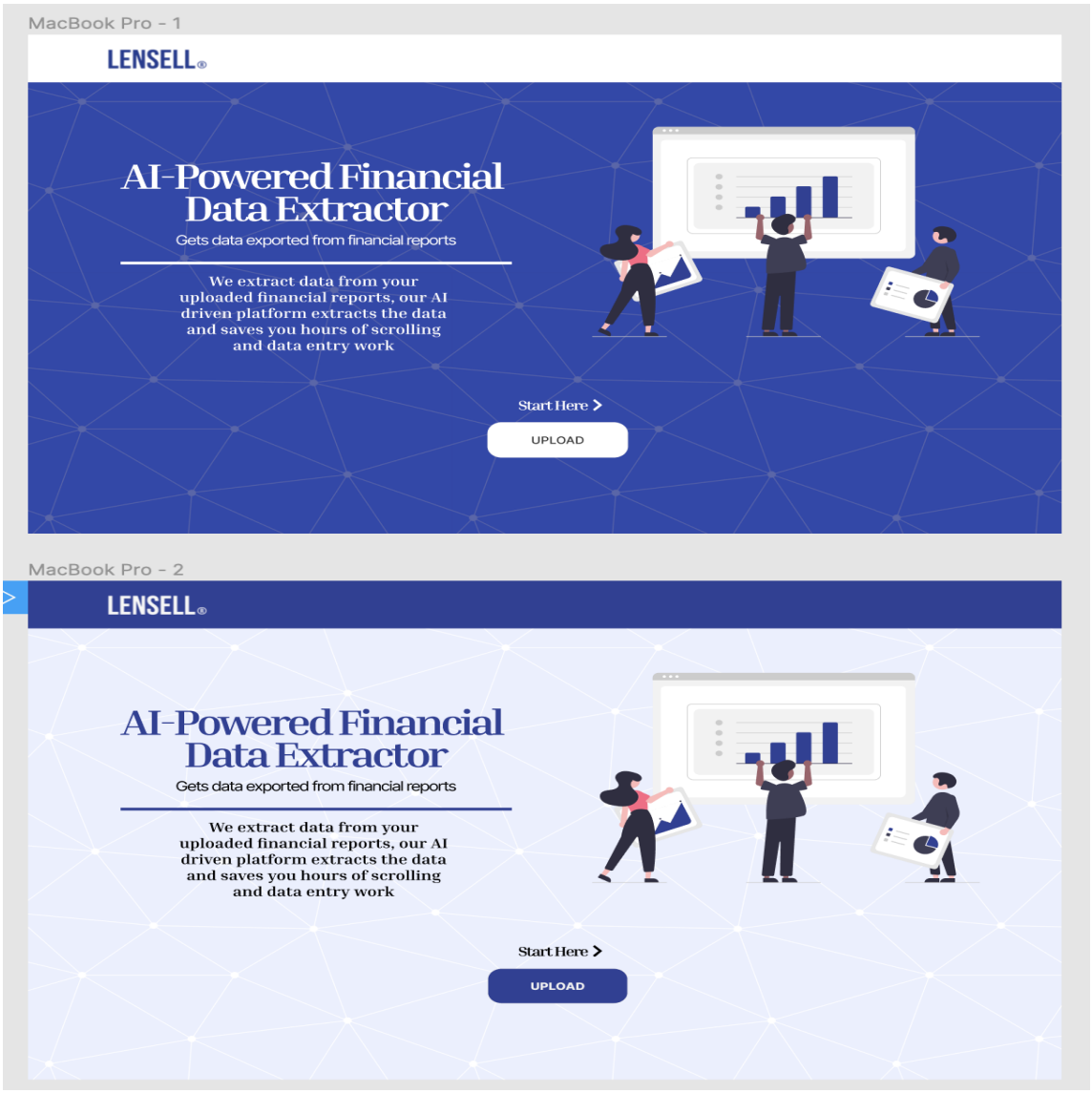


Figure 4.3.1.5: High fidelity wireframe branding

Figure 4.3.1.6: High fidelity wireframe overview

4.3.2 Final Delivery

Following all the work done for the planning of the prototype and wireframing the web application, we ended up having a more structured and simplified approach. Thanks to the pre-planning we had done for the prototype we already had a strong theme and UX which helped us refine the final delivery to a single page web application.

As seen below, we finalised to go for a single web page application as this allowed the user from navigating between multiple pages and also simplified the entire process by providing the user a single download button.

Figure 4.3.2.1: Final Delivery Wireframe

The download button would allow the user to get the CSV version of the extracted tables.

5.0 System Design

5.1 High-Level System Architecture

Following the guidelines of our client, the development of the system is heavily focused on the back-end and extraction engine functions. Therefore, system development has been heavily prioritised in these areas, with most of the early development being done on the extraction engine itself.

The back-end utilises the Python Django framework, to host both the REST API and the extraction engine itself. The API allows system communications between the front-end, back-end, and database, while the extraction engine is the core functional component of the system, pdf data extractions.

The front-end utilises the Angular framework using native components, such as HTML5 and CSS styling along with JavaScript to allow user communication and navigation throughout the site and allow the user to interact with the functional components of the back-end through the API.

Upon the user uploading a document for extraction, all extracted data will then be passed and stored locally and referenced within a SQLite database through the API for later historical and retrieval purposes.

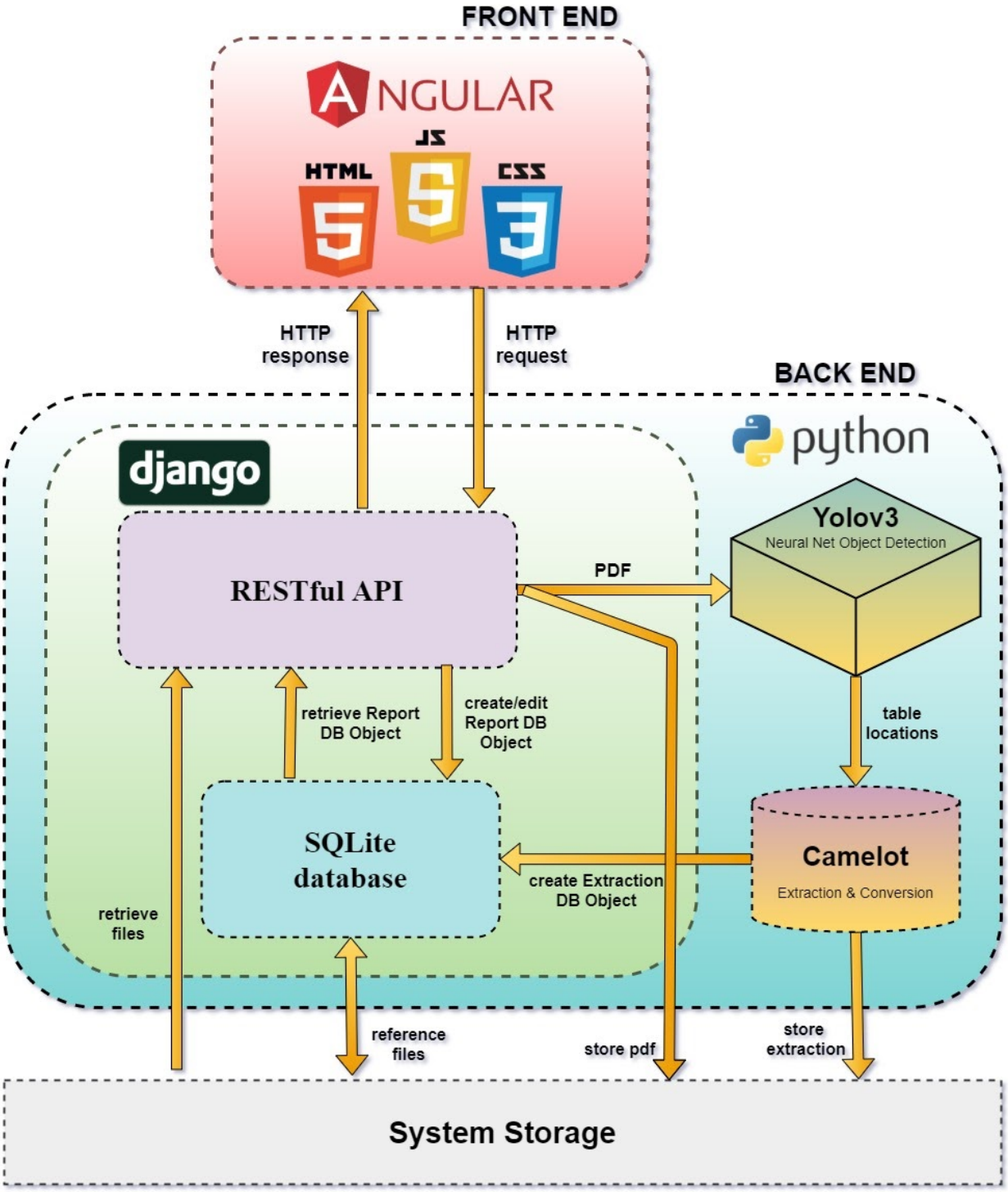


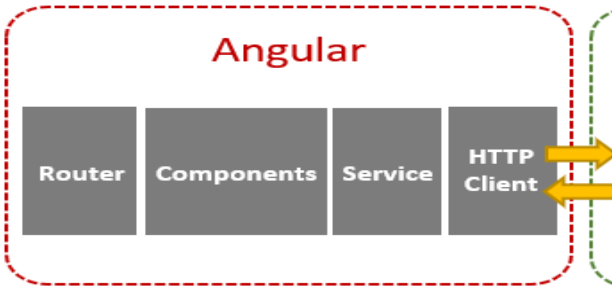
Figure 5.1.1: High-Level System Architecture

Hardware requirements:
There are no specific minimum system requirements. However, while YOLO inferencing can be run in either CPU or GPU acceleration configuration, to utilise GPU acceleration a Nvidia graphics card with CUDA is required.

5.2 Front-End Architecture

For the requirements of the project, we decided to use Angular 11. The concept was simple and to use the MVC model. We created different components which would act as Upload, Download etc. Later we simplified the code base by a single web page hence filtering out much of the components.

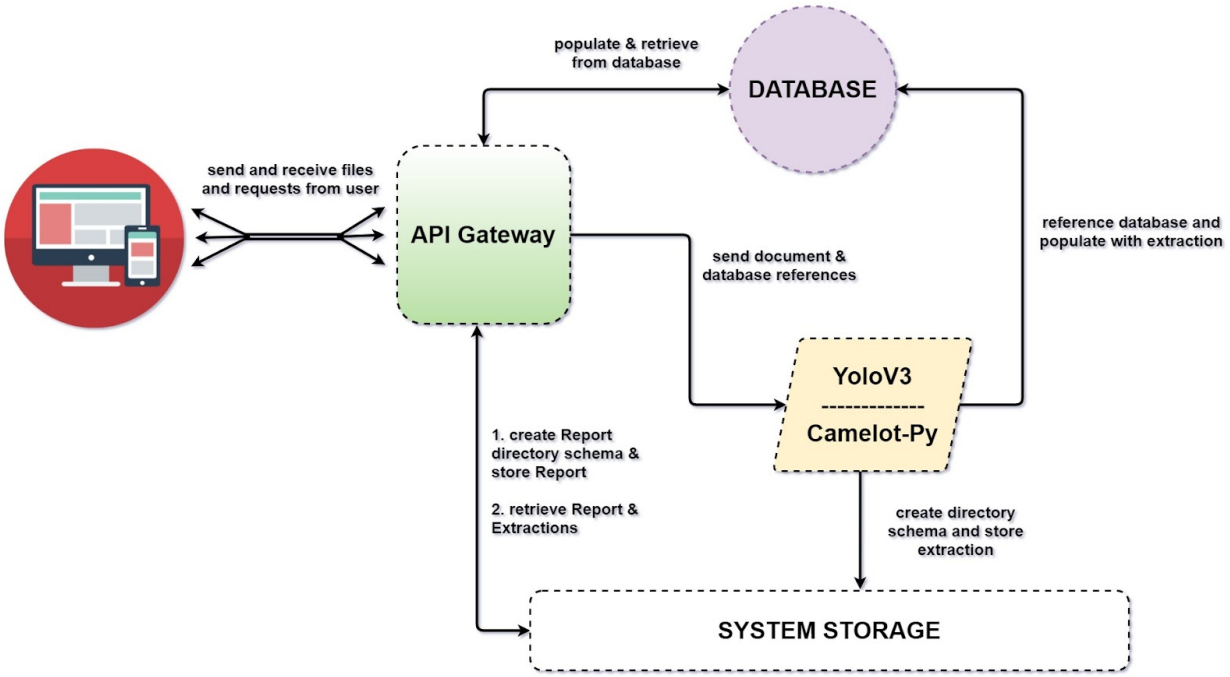
The major part of the architecture was to use the HTTP Client effectively. The Backend was the brains of the project and being able to use the API correctly would allow the extraction engine and the backend functionality to run smoothly.



5.3 Back-End Architecture

The API acts as the gateway to all functionality in the System, it acts as the brain of the backend processing system, handling all requests to and from the database, as well as sending pdfs to the extraction engine, and retrieving extraction from system storage and back to the user.

Using the API as the central process allows the whole process of uploading and extracting to be fully automated.



5.3.1 API

The system utilizes a services-oriented architecture by implementing a REST framework for API level access to the service..

API parameters	
Response formats	JSON; DICTIONARY
Requires authentication?	No
Rate limited?	No

Figure 5.3.1.1: API parameters

Http requests

Operation	HTTP Type	Example Resource Endpoint Path	Required	Optional
Upload and Extract a report	POST	/api/upload/	document=[file]	start_page=[int] end_page=[int]
Retrieve all reports	GET	/api/reports/		
Retrieve report by id	GET	/api/reports/{id}/	id=[int]	
Retrieve report by name	GET	/api/reports/?name={name}	name=[str]	
Update report	PATCH	/api/reports/{id}/	id=[int]	name=[str] total_pages=[int] start_page=[int] end_page=[int]
Download extraction zip file	GET	/documents/filename/filename-CSV.zip		
Delete report by id	DELETE	/api/reports/{id}/	id=[int]	

Figure 5.3.1.2: API HTTP requests

Example upload request

```
## set defaults
FILE_PATH="pdf_doc.pdf"
START_PAGE=13
END_PAGE=13


$ curl -f --location --request POST 'http://localhost:8000/api/upload/' \
--connect-timeout 10 \
--form 'document=@'"$FILE_PATH" \
--form 'start_page='"$START_PAGE" \
--form 'end_page='"$END_PAGE"
```

Figure 5.3.1.3: API upload request

Example upload request response:

```
{
  "report id": 253,
  "file name": "pdf_doc",
  "total pages": 64,
  "start page": 13,
  "end page": 13,
```

```
  "output types": "['csv', 'json']",
  "tables found": 3
}
```

Figure 5.3.1.4: upload response

Example retrieve all reports request

```
curl --location --request GET 'http://localhost:8000/api/reports/'
```

Figure 5.3.1.5: API retrieve all reports request

Example retrieve all reports request response:

```
{
  "id": 255,
  "name": "pdf_doc",
  "f_type": "pdf",
  "document": "http://localhost:8000/documents/pdf_doc/pdf_doc.pdf",
  "zip_csv": "http://localhost:8000/documents/pdf_doc/pdf_doc-CSV.zip",
  "total_pages": 64,
  "start_page": 13,
  "end_page": 13,
  "extracted": [
    {
      "page_num": 13,
      "table_num": 0,
      "f_type": "csv",
      "file": "http://localhost:8000/documents/pdf_doc/csv/pdf_doc-109-13-table-0.csv"
    }, ... ]
},
{
  "id": 255,
  ...
```

Figure 5.3.1.6: API retrieve all reports response

Example retrieve report by id request

```
ID=255

curl --location --request GET 'http://192.168.20.3:8000/api/reports/'"$ID"
```

Figure 5.3.1.7: API retrieve report by id request

Example retrieve report by id request response:


```
{
  "id": 255,
  "name": "pdf_doc",
  "f_type": "pdf",
  "document": "http://localhost:8000/documents/pdf_doc/pdf_doc.pdf",
  "zip_csv": "http://localhost:8000/documents/pdf_doc/pdf_doc-CSV.zip",
  "total_pages": 64,
  "start_page": 13,
  "end_page": 13,
  "extracted": [
    {
      "page_num": 13,
      "table_num": 0,
      "f_type": "csv",
      "file": "http://localhost:8000/documents/pdf_doc/csv/pdf_doc-109-13-table-0.csv"
    }, ... ]
}
```


Figure 5.3.1.8: API retrieve report by id response

Example retrieve report by name request

```
NAME=”pdf_doc”

curl --location --request GET
'http://192.168.20.3:8000/api/reports/?name='"$NAME"
```

Figure 5.3.1.9: API retrieve report by name request

Example retrieve report by name request response:


```
{
  "id": 255,
  "name": "pdf_doc",
  "f_type": "pdf",
  "document": "http://localhost:8000/documents/pdf_doc/pdf_doc.pdf",
  "zip_csv": "http://localhost:8000/documents/pdf_doc/pdf_doc-CSV.zip",
  "total_pages": 64,
  "start_page": 13,
  "end_page": 13,
  "extracted": [
    {
      "page_num": 13,
      "table_num": 0,
      "f_type": "csv",
```

```
      "file": "http://localhost:8000/documents/pdf_doc/csv/pdf_doc-109-13-table-0.csv"
    }, ... ]
}
```

Figure 5.3.1.10: API retrieve report by name response

Example update report by id request

```
curl --location --request PATCH 'http://localhost:8000/api/reports/245/' \
--form 'name="change_my_name"'
```

Figure 5.3.1.11: API update report by id request

Example update report by id request response:

```
{
  "id": 255,
  "name": "change_my_name",
  "f_type": "pdf",
  "document": "http://localhost:8000/documents/pdf_doc/pdf_doc.pdf",
  "zip_csv": "http://localhost:8000/documents/pdf_doc/pdf_doc-CSV.zip",
  "total_pages": 64,
  "start_page": 13,
  "end_page": 13,
  "extracted": [
    {
      "page_num": 13,
      "table_num": 0,
      "f_type": "csv",
      "file":
"http://localhost:8000/documents/pdf_doc/csv/pdf_doc-109-13-table-0.csv"
    }, ... ]
}
```

Figure 5.3.1.12: API update report by id response

Example download extraction zip request

```
wget --no-check-certificate \
  --method GET \
  --timeout=0 \
  --header '' \
  -O pdf_doc-CSV.zip \
  http://localhost:8000/documents/pdf_doc/pdf_doc-CSV.zip
```

Figure 5.3.1.13: API download extraction zip request

Example download extraction zip request response:

```
Connecting to localhost:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1621 (1.6K) [application/x-zip-compressed]
Saving to: 'pdf_doc-CSV.zip'

valid-tables-109.zip
100%[=====================================================================>]
1.58K  --.-KB/s    in 0s

2021-10-23 05:34:02 (82.2 MB/s) - 'pdf_doc-CSV.zip' saved [1621/1621]
```

Figure 5.3.1.14: API download extraction zip response

Example delete report by id request

```
ID=255

# DELETE REPORT
curl --location --request DELETE
'http://192.168.20.3:8000/api/reports/'"$ID"'/'

# TRY AND RETRIEVE REPORT
curl --location --request GET
'http://192.168.20.3:8000/api/reports/'"$ID"'/'
```

Figure 5.3.1.15: API delete report by id request

Example delete report by id request response:

```
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0

checking report has been deleted...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100    23  100    23    0     0   2875      0 --:--:-- --:--:-- --:--:--  2875

{"detail":"Not found."}
```

Figure 5.3.1.16: API delete report by id response

HTTP Response Codes

The API will respond with standard HTTP response codes as outlined in the following tables.

Informational - 1xx
HTTP_100_CONTINUE **HTTP_101_SWITCHING_PROTOCOLS**

Figure 5.3.1.17: API response table 1xx

Successful - 2xx
HTTP_200_OK **HTTP_201_CREATED** **HTTP_202_ACCEPTED** **HTTP_204_NO_CONTENT** **HTTP_205_RESET_CONTENT** **HTTP_206_PARTIAL_CONTENT** **HTTP_207_MULTI_STATUS** **HTTP_208_ALREADY_REPORTED**

Figure 5.3.1.18: API response table 2xx

Redirection - 3xx
HTTP_300_MULTIPLE_CHOICES **HTTP_301_MOVED_PERMANENTLY** **HTTP_302_FOUND** **HTTP_303_SEE_OTHER** **HTTP_304_NOT_MODIFIED** **HTTP_305_USE_PROXY** **HTTP_306_RESERVED** **HTTP_307_TEMPORARY_REDIRECT** **HTTP_308_PERMANENT_REDIRECTH**

Figure 5.3.1.19: API response table 3xx

Client Error - 4xx
HTTP_400_BAD_REQUEST **HTTP_401_UNAUTHORIZED** **HTTP_403_FORBIDDEN** **HTTP_404_NOT_FOUND** **HTTP_405_METHOD_NOT_ALLOWED** **HTTP_406_NOT_ACCEPTABLE** **HTTP_408_REQUEST_TIMEOUT** **HTTP_409_CONFLICT** **HTTP_410_GONE** **HTTP_412_PRECONDITION_FAILED**

HTTP_413_REQUEST_ENTITY_TOO_LARGE
HTTP_414_REQUEST_URI_TOO_LONG
HTTP_415_UNSUPPORTED_MEDIA_TYPE
HTTP_416_REQUESTED_RANGE_NOT_SATISFIABLE
HTTP_417_EXPECTATION_FAILED
HTTP_422_UNPROCESSABLE_ENTITY
HTTP_428_PRECONDITION_REQUIRED
HTTP_429_TOO_MANY_REQUESTS
HTTP_431_REQUEST_HEADER_FIELDS_TOO_LARGE

Figure 5.3.1.20: API response table 4xx

Server Error - 5xx
HTTP_500_INTERNAL_SERVER_ERROR
HTTP_501_NOT_IMPLEMENTED
HTTP_502_BAD_GATEWAY
HTTP_503_SERVICE_UNAVAILABLE
HTTP_504_GATEWAY_TIMEOUT
HTTP_505_HTTP_VERSION_NOT_SUPPORTED
HTTP_507_INSUFFICIENT_STORAGE
HTTP_508_LOOP_DETECTED
HTTP_510_NOT_EXTENDED

Figure 5.3.1.21: API response table 5xx

5.3.2 Extraction Engine

Due to the nature of the PDF itself and the documents this system aims to process (documents with non-standardised formatting) we've decided to employ a trained algorithm (YOLO) using Object image recognition to detect and locate table data sets. Then using coordinates of the found table datasets, parse the table data into a dataframe using the Python library Camelot and convert it into CSV format.

The main problem encountered when trying to reliably parse data from PDF is the format lacks a consistent top-down/left-right structural-semantic layer from which to locate Object types - such underlying logical structure can be found within HTML. Instead, PDF uses a framework of spatial coordinates known as "bounding boxes" to format and orient all Object types within a page.

By utilising the power of image Object detection and only relying on the spatial framework of bounding boxes within the PDF document structure as means to locate a detected Object's location, we think our approach provides the best path to a reliable and adaptable solution.

Outline of the extraction process:

1. The engine takes a page of the PDF document and using PyPDF2 obtains the page dimensions of the PDF page.

2. Then, using pdf2img, convert the PDF page to an image format and with matplotlib obtain the dimensions of the converted image.
3. This image is then passed to the object detection system and if table data is found, returns the location coordinates.
4. The returned location coordinates are then normalised for reference to the original PDF document using the image coordinates and the PDF page coordinates for reference.
5. The PDF page and the normalised coordinates are then passed to the Camelot library for data extraction into the dataframe, which then allows for any data testing/cleaning and finally parsing to CSV.

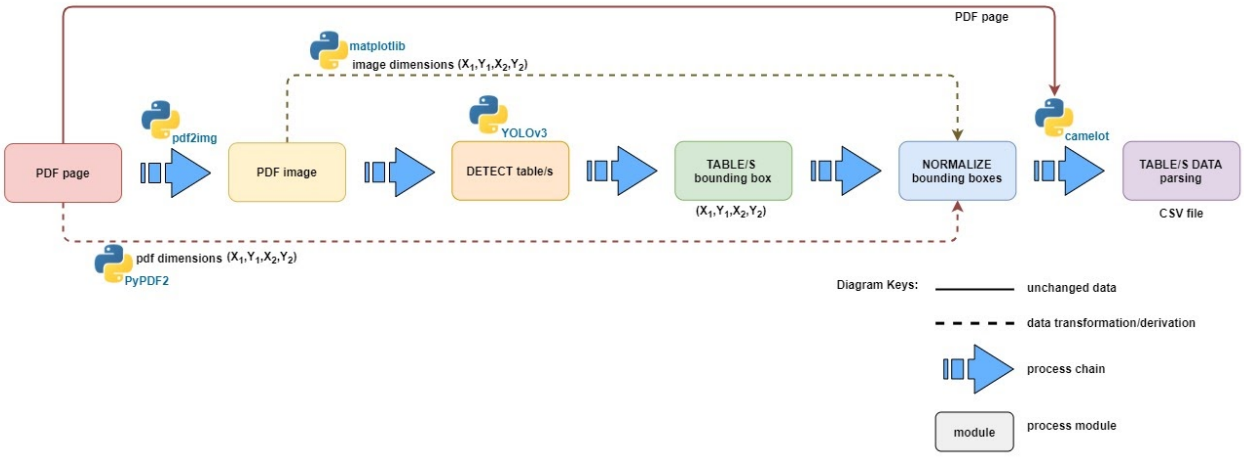


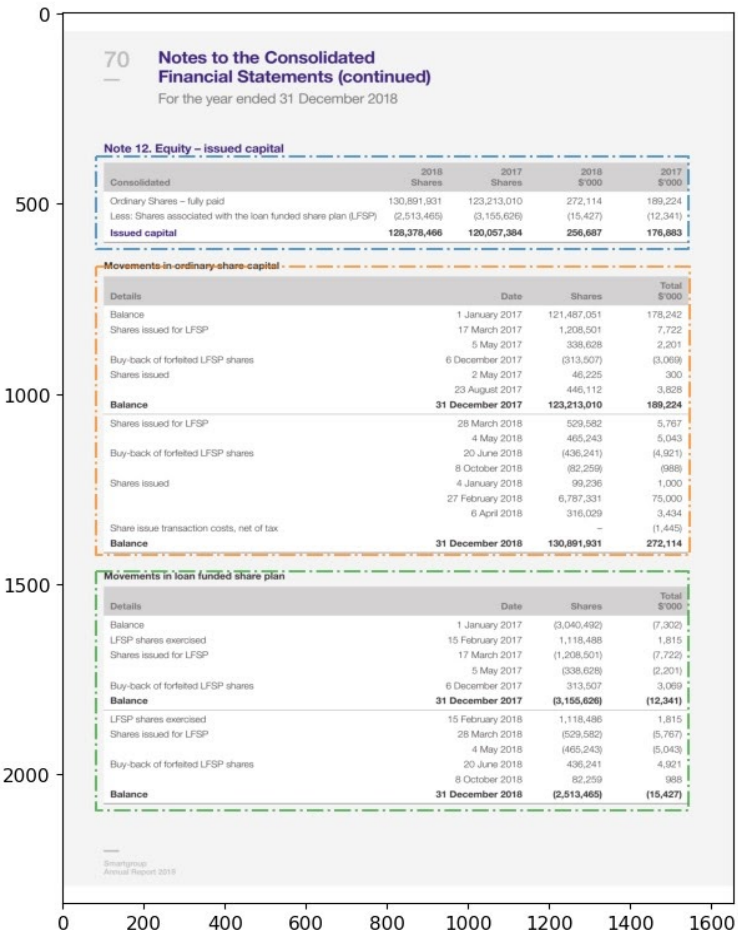
Figure 5.3.2.1: High-Level Extraction Engine Architecture

YOLOv3: Real-Time Object Detection

YOLO - You Only Look Once - is an algorithm proposed by Redmond et. al in a research article published at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) for using neural networks to provide real-time object detection.

The YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. And as the name suggests, the algorithm only requires a single forward propagation through a neural network to detect objects. This means that prediction in the entire image is done in a single algorithm run.

Moreover, as the CNN is used to predict various class probabilities and bounding boxes simultaneously, it is perfect for our use case of detecting and locating tables via coordinates and parsing through Python Camelot's inbuilt *table_areas* feature.



70

Notes to the Consolidated Financial Statements (continued)

For the year ended 31 December 2018

Note 12. Equity – issued capital

Consolidated	2018 Shares	2017 Shares	2018 $'000	2017 $'000
Ordinary Shares – fully paid	130,891,931	123,213,010	272,114	189,224
Less: Shares associated with the loan funded share plan (LFSP)	(2,513,465)	(3,155,626)	(15,427)	(12,341)
Issued capital	**128,378,466**	**120,057,384**	**256,687**	**176,883**

Movements in ordinary share capital

Details	Date	Shares	Total $'000
Balance	1 January 2017	121,487,051	178,242
Shares issued for LFSP	17 March 2017	1,208,501	7,722
	5 May 2017	338,628	2,201
Buy-back of forfeited LFSP shares	6 December 2017	(313,507)	(3,069)
Shares issued	2 May 2017	46,225	300
	23 August 2017	446,112	3,828
Balance	**31 December 2017**	**123,213,010**	**189,224**
Shares issued for LFSP	28 March 2018	529,582	5,767
	4 May 2018	465,243	5,043
Buy-back of forfeited LFSP shares	20 June 2018	(436,241)	(4,921)
	8 October 2018	(82,259)	(988)
Shares issued	4 January 2018	99,236	1,000
	27 February 2018	6,787,331	75,000
	6 April 2018	316,029	3,434
Share issue transaction costs, net of tax		–	(1,445)
Balance	**31 December 2018**	**130,891,931**	**272,114**

Movements in loan funded share plan

Details	Date	Shares	Total $'000
Balance	1 January 2017	(3,040,492)	(7,302)
LFSP shares exercised	15 February 2017	1,118,488	1,815
Shares issued for LFSP	17 March 2017	(1,208,501)	(7,722)
	5 May 2017	(338,628)	(2,201)
Buy-back of forfeited LFSP shares	6 December 2017	313,507	3,069
Balance	**31 December 2017**	**(3,155,626)**	**(12,341)**
LFSP shares exercised	15 February 2018	1,118,486	1,815
Shares issued for LFSP	28 March 2018	(529,582)	(5,767)
	4 May 2018	(465,243)	(5,043)
Buy-back of forfeited LFSP shares	20 June 2018	436,241	4,921
	8 October 2018	82,259	988
Balance	**31 December 2018**	**(2,513,465)**	**(15,427)**

Figure 5.3.2.2: YOLOv3 bounding boxes

YOLOv3 Parameters	
Configuration file (self.cfg)	yolov3-tiny_table.cfg
Weights file (self.weights)	best_v2.weights
Confidence threshold (self.conf_thres)	0.24
Names file (self.names)	table.names
Image size (self.img_size)	416
Overlap threshold (self.iou_thres)	0.30

YOLOv3 Features

To see bounding box detections as shown above during extractions, in api\scripts\YOLOV3\predict_table.py, line 188, set see_example = *True*

```
# do you want to see prediction image?
see_example = True
```

Figure 5.3.2.3: YOLOv3 set see bounding boxes

Camelot: PDF Table Extraction

Camelot is a highly configurable text-based PDF table-data extraction tool for Python.

We've used Camelot both due to its high degree of configurability (specifically it's tables_areas feature, which accepts coordinates that conform to YOLOv3's bounding box format) and its use of dataframes for table data representation.

1. We're calling camelot using the *interesting areas* (an array containing the coordinates of detected tables) found by YOLOv3
2. Camelot is currently being used in 'stream' *flavour*
    a. Stream and lattice are heuristics for predicting table structure
        i. lattice is best suited for traditional bordered tables
        ii. Stream is best suited for also modelling non-traditional borderless tables
3. We're using Camelot version >= v0.10.0, which as a new feature allows the choice of backend pdf rendering library: choices are 'poppler' or 'ghostscript'.
    a. We've chosen to use Poppler as it has less issues across most platforms.

Camelot extraction method call

```
camelot.read_pdf(
    filepath=pdf_file,
    pages=str(pg),
    flavor="stream",
    table_areas=interesting_areas,
    backend="poppler",
)
```

Figure 5.3.2.4: Camelot extraction function call

Camelot Parameters	
flavor	stream
table_areas	Array containing the coordinates of detected tables found by YOLOv3
backend	Poppler

Return type	Panda dataframe

Figure 5.3.2.5: Camelot parameters

Camelot Features

For debugging and System testing, Camelot's inbuilt performance metric method is passed through system logging, which returns a parsing report with %accuracy, %whitespace, table number on page and page number.

```
table parsing_report: {
    'accuracy': 100.0,
    'whitespace': 4.0,
    'order': 1,
    'page': 13
}
```

Figure 5.3.2.6: Camelot performance logging

Exporting Camelot Dataframe to CSV & JSON

To export Camelot parsed tables to CSV and JSON, we're using the Python Pandas Class pandas.DataFrame.to_csv() and Pandas.DataFrame.to_json() methods respectively - Pandas to_csv(); Pandas to_json()

Pandas to_csv() parameters	
path_or_buf	Filepath for filename and save location on system storage
index	False

Figure 5.3.2.7: Pandas csv export parameters

Pandas to_json() parameters	
path_or_buf	filename/save location on system storage
orient	columns

Figure 5.3.2.8: Pandas json export parameters

Export file name expression

```
{filename}[:-4] + "-" + str(pg) + "-table-" + str(i) + "." + key
```

Figure 5.3.2.9: Extraction export naming structure

Where:

filename: is the uploaded pdf document name, minus the extension: eg. pdf_doc.pdf

pg: is the page number of the document where the table was extracted from: eg. 6

i : is the number of the table found on that page (more than one table can be found on any given page): eg 2

key: is the file extension (.csv, .json)

Example csv file: pdf_doc-6-table-2.csv
Example json file: pdf_doc-6-table-2.json

System Performance Improvements

The extraction system has been optimised using multiprocessing for best-case performance.

Testing both multithreading and multiprocessing showed that the latter had far greater effect on performance than the prior.

table detection/extraction performance test:

Test Type	Total time	second/page	Performance change %
BASELINE	12 mins 13.17 secs	~3.98	-
MULTIPROCESSING	2 mins and 24.44 secs	~0.79	403.8%
MULTITHREADING	3 mins and 55.38 secs	~1.28	210.9%

test file: ANZ-Annual-Report-2018.pdf, 184 pages
test machine: AMD fx-8350, 8 cores, 16gb ram

5.3.3 System Storage

For each document uploaded, a distinct storage tree is constructed using the document's name sans file extension.

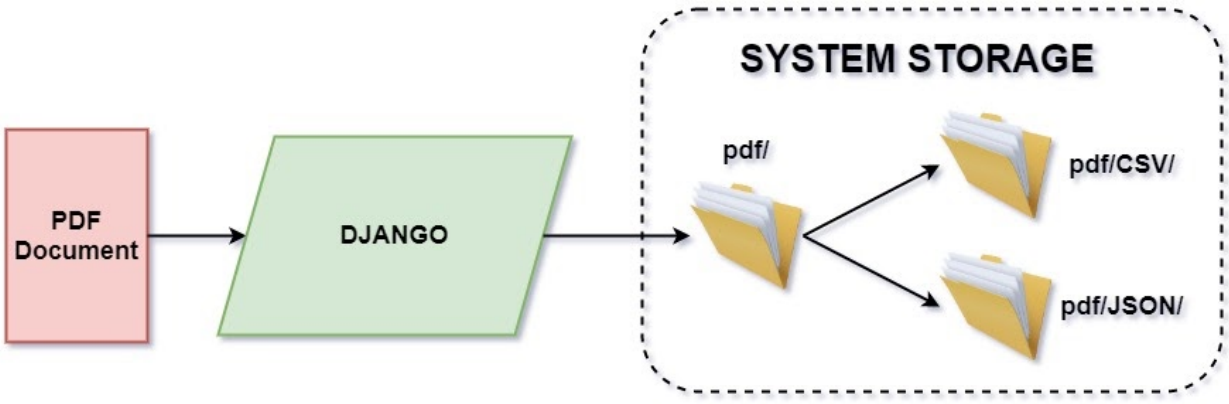


Figure 5.3.3.1: Document storage tree

By default, Django stores files locally, using the MEDIA_ROOT and MEDIA_URL settings.

In addition to this, for PDF document storage a storage Class (MyStorage) was built for handling file and directory issues related to uploading of documents more than once, along with two helper functions for document name and path creation structuring, get_valid_filename() and uplad_path().

Django System Storage Parameters	
BASE_DIR	Path(__file__).resolve().parent.parent
MEDIA_URL	"/documents/"
MEDIA_ROOT	PurePath(BASE_DIR, "documents")

Figure 5.3.3.2: Django storage parameters

```
class MyStorage(FileSystemStorage):
    """
    User defined File System Storage Class
    """

    def get_available_name(self, name, max_length=None):
        """
        If a file name is already in use on storage, creates a converted String
        filename for the duplicate name, attaching '_!added!'
        >>> get_available_name("anz_2004.jpg")
        'anz_2004_!added!.jpg'
        """
        if self.exists(name):
            dir_name, file_name = PurePath(name).parts
            file_root = PurePath(file_name).stem
            file_ext = PurePath(file_name).suffix

            my_chars = "!added!"  # The characters you want to append

            name = PurePath(dir_name, "{}_{}{}".format(file_root, my_chars, file_ext))
        return name
```

Figure 5.3.3.3: Django MyStorage Class

```
def get_valid_filename(s):
    """
    Returns the given string converted to a string that can be used for a clean
    filename. Specifically, leading and trailing spaces are removed; other
```

```
    spaces are converted to underscores; and anything that is not a unicode
    alphanumeric, dash, underscore, or dot, is removed.
    >>> get_valid_filename("john's portrait in 2004.jpg")
    'johns_portrait_in_2004.jpg'
    """
    s = force_text(s).strip().replace(" ", "_")
    return re.sub(r"(?u)[^-\w.]", "", s)
```

Figure 5.3.3.4: Django storage helper function get_valid_filename()

```
def upload_path(instance, filename):
    """
    Returns a file directory path naming schema using the filename as parent
    directory
    >>> upload_path(Report, "anz_2004.pdf")
    'documents/anz_2004/anz_2004.pdf'
    """
    return "/".join(
        [str(get_valid_filename(instance.filename())), get_valid_filename(filename)]
    )
```

Figure 5.3.3.5: Django storage helper function upload_path()

5.3.4 Database

The database model has two Objects: Report, and Extracted. The reason for this model is down to the outlined project requirements provided by the client, along with the specific feedback for minimal user involvement.

To this end, we designed a database which handles the necessary requirement of storing a specific pdf file, along with its respective extracted files created after having been processed by the extraction engine.

ER Diagram:

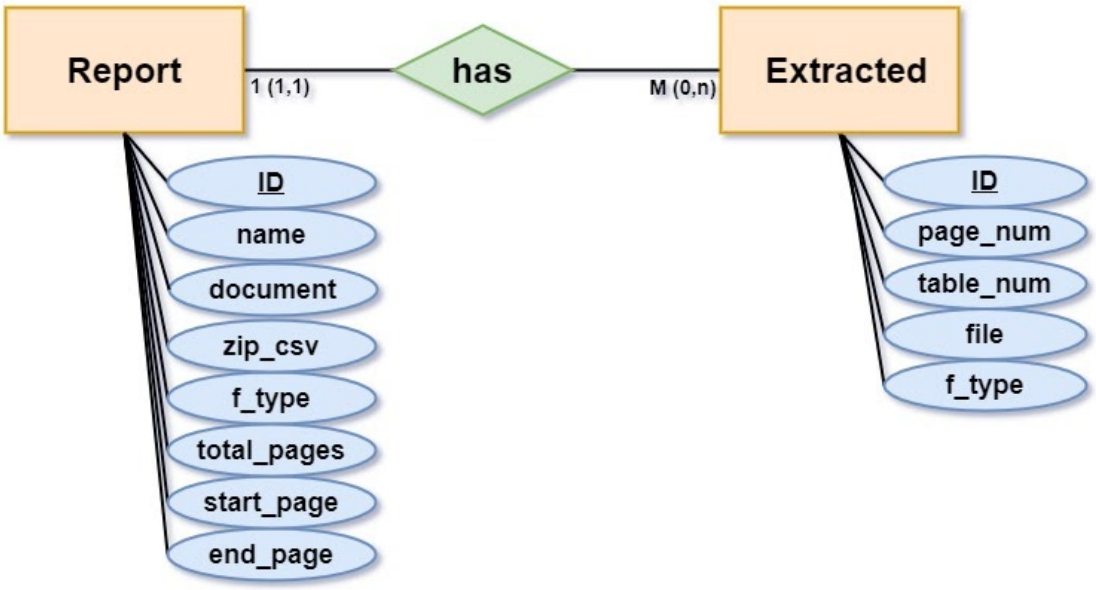


Figure 5.3.4.1: Database ER Diagram

Relations:

Report(**ReportID**, name, document, zip_csv, f_type, total_pages, start_page, end_page)
Extracted(**ExtractedID**, page_num, table_num, file, f_type, *ReportID*)

* Primary keys are **bold** and foreign keys are in *italics*/underlined

Relational Table:

Entity	Name	Data Type	Constraints
Report	**ReportID**	SMALLINT	PRIMARY KEY, NOT NULL
	name	VARCHAR2(100)	
	document	FILE	
	zip_csv	FILE	
	f_type	VARCHAR2(5)	
	total_pages	SMALLINT	
	start_page	SMALLINT	DEFAULT -1
	end_page	SMALLINT	DEFAULT -1
Extracted	**ExtractedID**	SMALLINT	PRIMARY KEY, NOT NULL
	Page_num	SMALLINT	
	Table_num	SMALLINT	
	File	FILE	
	f_type	VARCHAR2(5)	
	ReportID	SMALLINT	Foreign Key - ref Report(ReportID)

Figure 5.3.4.2: Database Relational Table

5.3.5 System Logging

A system logging class was created to handle logging of system information, warning and errors for easier server management. The Logging Class builds a log object that builds timestamped coloured string outputs for system logging that conforms to Django's terminal system outputs for seamless integration.

Log Type	Description
INFO	General system state information (service start/stop, configuration assumptions, system action
DEBUG	System diagnostic information for sysadmins, etc.
WARNING	Anything with potential to cause application oddities, but that automatically recovers or is expected behaviour. Such as missing or duplicated data, etc.
ERROR	Any fatal operation. These errors will force user (administrator, or direct user) intervention.

Figure 5.3.5.1: Logging colour chart

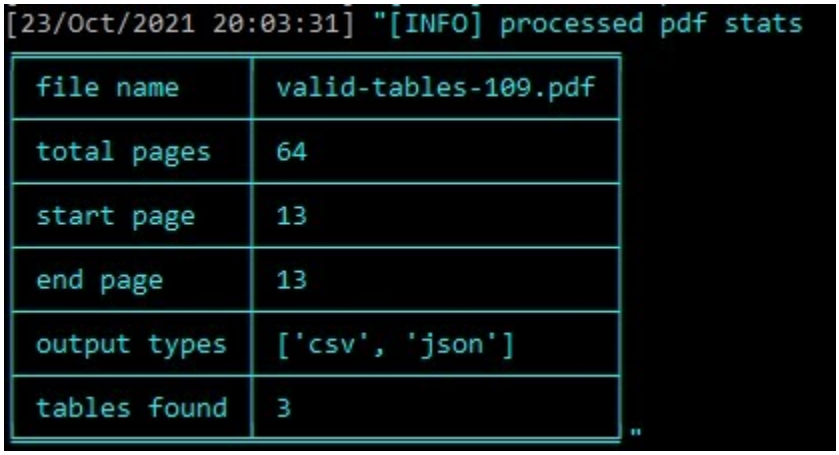


Figure 5.3.5.2: Logging extraction report message

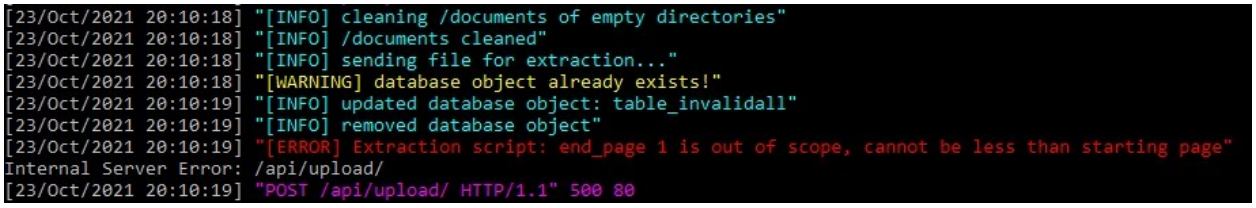


Figure 5.3.5.3: Logging system messages

5.3.6 Data Strategies

Strategies for reducing processing time and validating the quality of data extracted.

Quality Strategies

1. **Table Validation:**

We can test table data validity, by setting lower boundary limits on collected table dataframe depth and breadth - dataframe depth being equivalent to rows and dataframe breadth being equivalent to columns.

Assumptions:

1. Tables do not have less than 2 columns.
2. Tables need more than 1 row.

Dataframe depth and breadth boundaries:

Depth:	>1
Breadth:	>=2

```
def tableValidate(dataframe) -> bool:
    """
    validation function for evaluating extracted tables
    uses column length and row depth as measure of validation

    returns Boolean
    """
    row_val = len(dataframe) >= 1
    col_val = len(dataframe.columns) >= 2

    return col_val and row_val
```

Figure 5.3.6.1: Table validation strategy function

5.4 Use Case Analysis

Use cases are descriptions of how an end-user wants to “use” the system. These “uses” can be thought of as requests made by the user to the system, and use cases describe the system's response to those requests. In this way, we can think of use case analysis as a detailed overlay of the conversation between the system and its user(s).

USE CASE 1: Upload a PDF document
TRIGGER/GOAL: Document is uploaded and available to the system
ACTOR: User
MAIN FLOW:

1. User selects a file from the local device on the webpage upload portal
2. User clicks on the upload button after selecting the file
3. The file is uploaded to the system

ALTERNATE FLOW:
EXCEPTION FLOW:

1. User uploads a non-PDF document
    a. The file is not uploaded
    b. A message is shown to the user stating that the file is not a PDF file.
    c. Empty web page upload portal is displayed

EXTENSIONS:

USE CASE 2: Extract data from PDF to CSV
TRIGGER/GOAL: Extracted data is available for download in CSV format
ACTOR: User
MAIN FLOW:

1. User uploads document for extraction
2. System processes document and extracts contents
3. Contents are converted to CSV format
4. Files are available to the user for download

ALTERNATE FLOW:

1. No data is found to be extracted from the file
    a. The system throws an exception stating no data found
    b. A message is sent to the web page stating no data found in the document
2. Data extracted from PDF file lacks a significant degree of meaningfulness
    a. The system throws an exception stating low data quality
    b. A message is sent to the web page stating low extracted data quality

EXCEPTION FLOW:

1. No file received for the system to process
    a. An error message is thrown stating system process failure

EXTENSIONS:

USE CASE 3: Save processed PDF document data
TRIGGER/GOAL: Processed document data is available for download
ACTOR: User
MAIN FLOW:

1. The file is received by the system from the user
2. The file is processed by the extraction engine
3. Processed data from the extraction engine is available on the system
4. User is shown download webpage with available download
5. User clicks the download button
6. User receives file on their local system

ALTERNATE FLOW:

1. No data is found to be extracted from the file
    a. The system throws an exception stating no data found
    b. A message is sent to the web page stating no data found in the document

2. Data extracted from PDF file lacks a significant degree of meaningfulness
    c. The system throws an exception stating low data quality
    d. A message is sent to the web page stating low extracted data quality

EXCEPTION FLOW:

1. No file received for the system to process
    a. An error message is thrown stating system process failure

EXTENSIONS:

USE CASE 4: Search the history of previous extractions/reports
TRIGGER/GOAL: Previous extractions are available for viewing/searching
ACTOR: User
MAIN FLOW:

1. User accesses the history database through the website database portal
2. Previous extractions are presented to the user on the webpage

ALTERNATE FLOW:

1. No objects in the database
    a. A message is sent to the webpage stating that no objects are in the database

EXCEPTION FLOW:
EXTENSIONS:

USE CASE 5: Retrieve previous extractions/reports.
TRIGGER/GOAL: Previous extractions are available for retrieval
ACTOR: User
MAIN FLOW:

1. User accesses the history database through the website database portal
2. Previous extractions are presented to the user on the webpage
3. Previous extractions are searchable
4. Previous extractions are selectable
5. Previous extractions are downloadable

ALTERNATE FLOW:

1. No objects in the database
    a. A message is sent to the webpage stating that no objects are in the database

EXCEPTION FLOW:
EXTENSIONS:

USE CASE 6: Delete previous extractions/reports.
TRIGGER/GOAL: Previous extractions are deleted from the storage/database
ACTOR: User
MAIN FLOW:

1. User accesses the history database through the website database portal
2. Previous extractions are presented to the user on the webpage

3. Previous extractions are searchable
4. Previous extractions are selectable
5. Previous extractions are deletable

ALTERNATE FLOW:

1. No objects in the database
    a. A message is sent to the web page stating that no objects are in the database

EXCEPTION FLOW:

EXTENSIONS:

5.4.1 Use Case Diagram

The system has accessibility from 2 levels of access, one being a regular **web user**, who can access the system via the domain with limited functionality. The other level of access includes individuals who can access the **API** directly by downloading the source code from the github repository or other means of access will provide a lot more functions to the user.

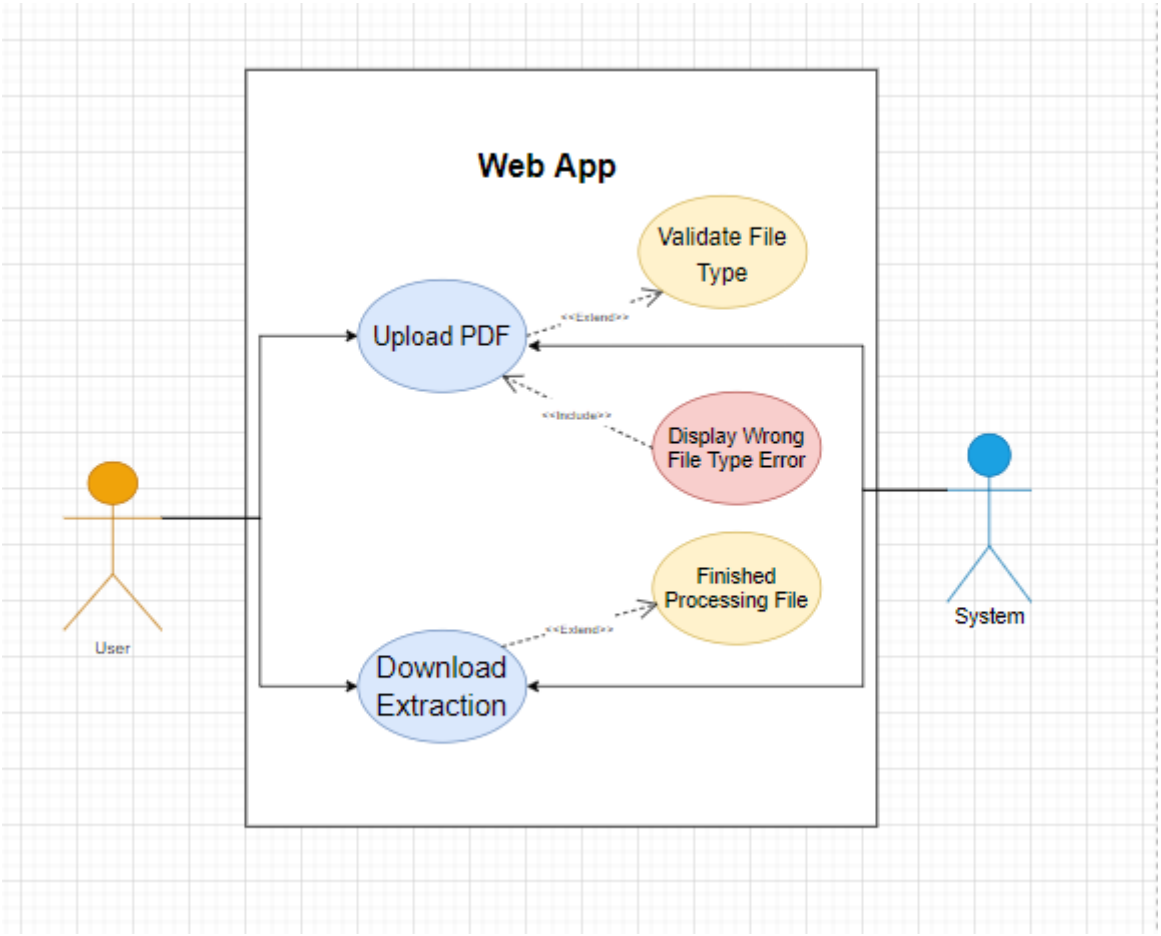


Figure 5.4.1.1: Use Case Diagram for Web User

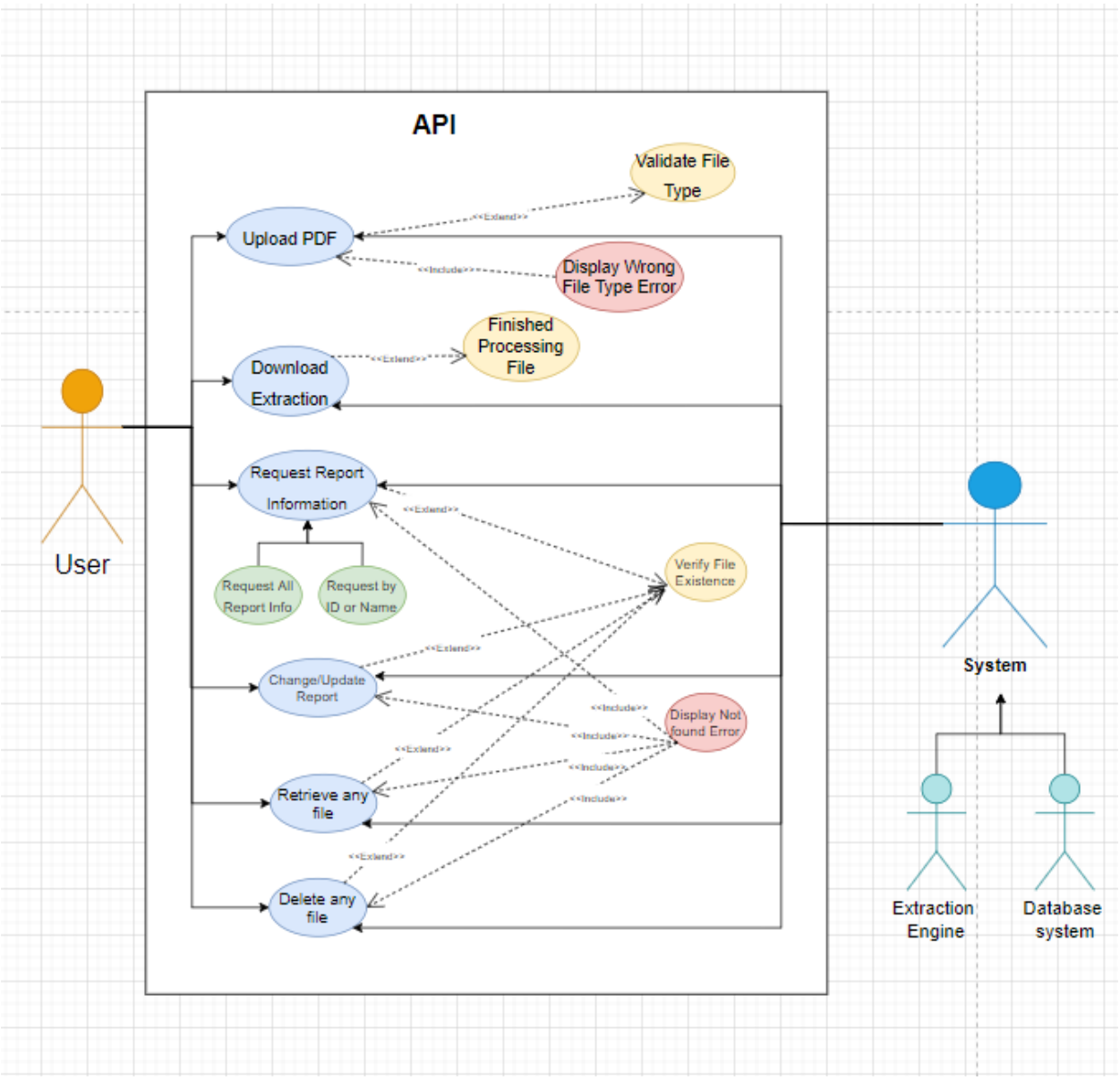


Figure 5.4.1.2: Use Case Diagram for API user

5.5.2 Sequence Diagrams

Sequence diagrams describe the interactions between objects and how the different parts of a system interact with each other to carry out a function, along with the order in which the interactions occur when a particular case is executed.

Our sequence diagrams are segmented across the two types of users that can access the system. Primarily being the web user who can access the system through a domain and our Angular frontend, another being an individual who has access to the source code and can access the system as an Admin or Developer.

Web-User Sequence Diagram

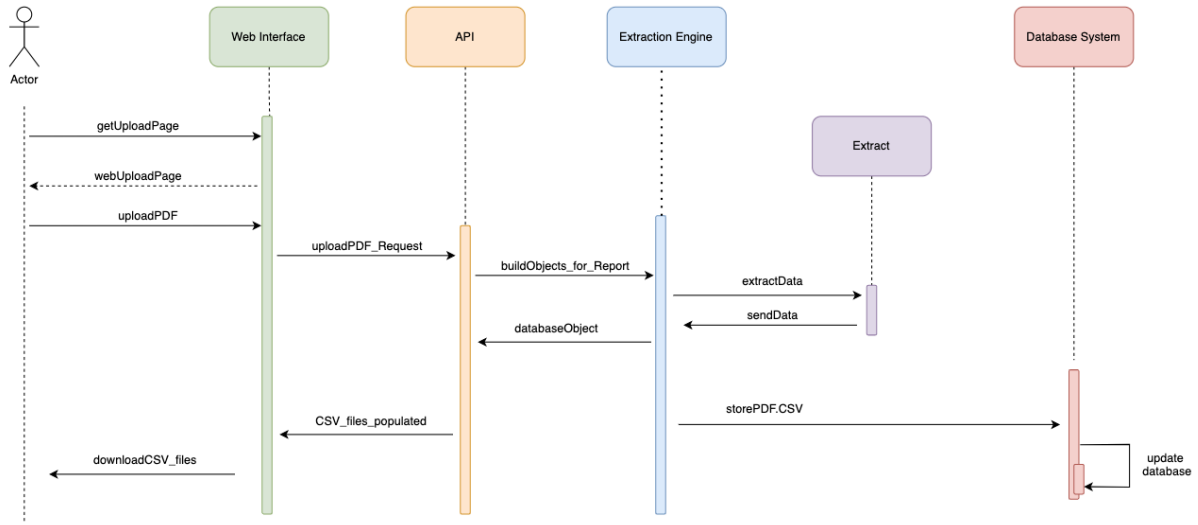


Figure 5.5.2.1: Web-User sequence diagram

Sequence Diagram for Administrative User

An administrative user can perform all the functions of a web-user with the additional reach of obtaining the ability to interact directly with the database.

API-User Sequence Diagram:

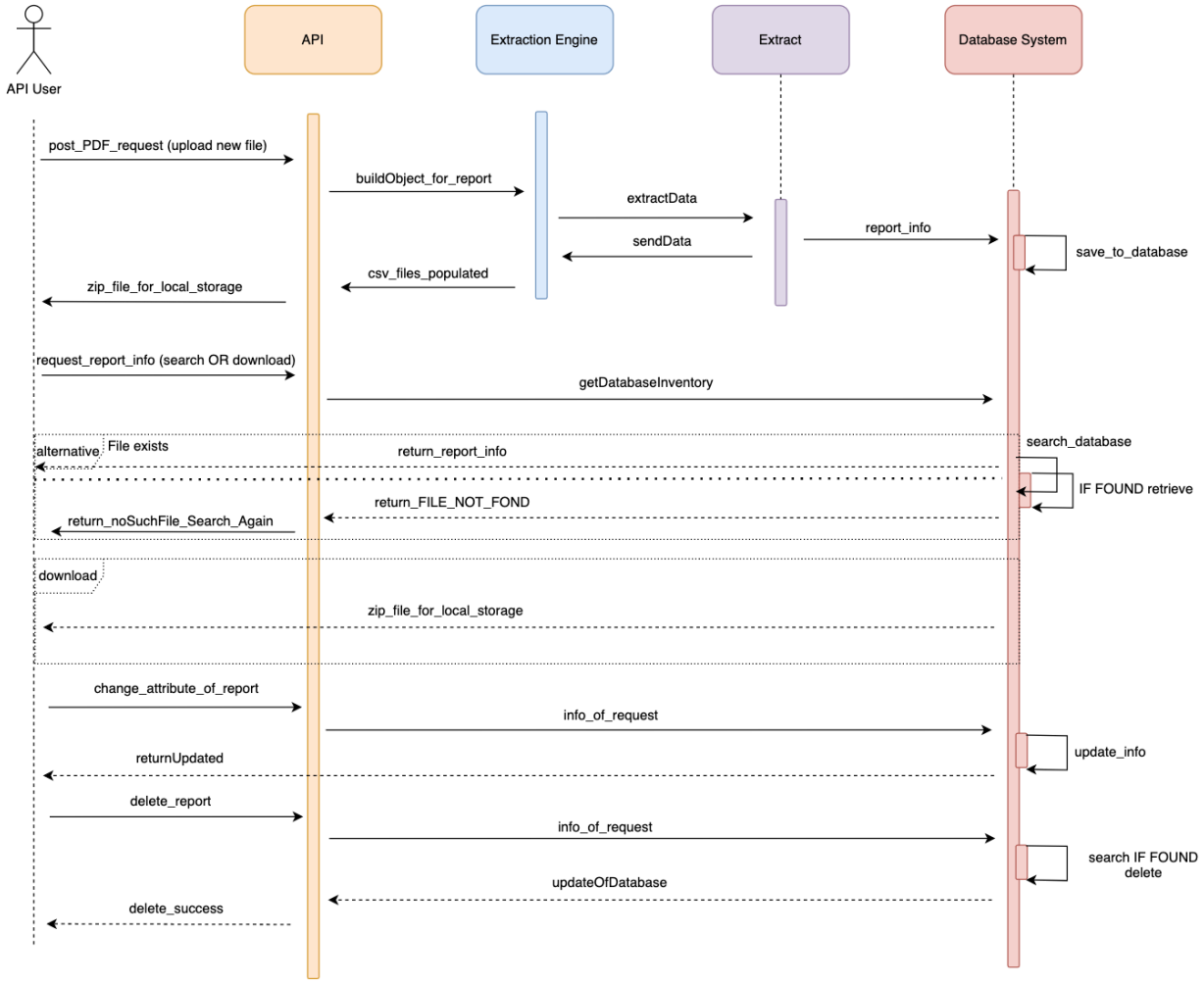


Figure 5.4.2.2: Download PDF/CSV from database sequence diagram

5.5 Object Dictionary

The following is a list of system methods and their respective Objects used throughout the system.

Name	Object	Package	Type	Description
extract(file_path: str, start_page: int, end_page: int) -> dict	Back-end	Extraction Engine	function	The main function for the connecting script between the API, Database, system storage and Yolo/Camelot integration.
detect_tables(file_path, page_number, output_type, report_db, extract_dir) -> list	Back-end	Extraction Engine	function	The main function for Yolo and Camelot integration.
tableValidate(dataframe) -> bool	Back-end	Extraction Engine	function	Basic extracted table validation function testing for improper formatted table dataframe
pdf_stats(filename: str, total_pages: int, start_page: int, end_page: int, output_types: dict, tables_found=None,) -> str:	Back-end	Extraction Engine	function	builds document processing information for logging
get_num_pages(path: str) -> int	Back-end	Extraction Engine	function	returns the total number of pages in pdf document
end_of_range(start_page: int, end_page: int, total_pages: int) -> int	Back-end	Extraction Engine	function	checks and validates user requested extraction end page
start_of_range(start_page: int, end_at: int, start_at=1) -> int	Back-end	Extraction Engine	function	checks and validates user requested extraction start page
collect_result(filename: str, table: dict, tables_list) -> None	Back-end	Extraction Engine	function	appends dictionary of table data to tables list
collect_parsing_report(report: list) -> None	Back-end	Extraction Engine	function	collector function for grabbing multi-processing outputs, appends outputs to report_list
process_extracted_file(filename: str, tables_list: list, full_working_dir: str) -> None:	Back-end	Extraction Engine	function	Builds table representations for response to front-end extraction queries and for printing to logging terminal

Logging	Back-end	API	Class	Logging Class builds a log object that builds timestamped coloured string outputs for system logging
time_stamp(self) -> str	Back-end	Logging	function	builds the timestamp for system logging to match django's inbuilt logging timestamp
output(self, type: str, msg: str) -> None	Back-end	Logging	function	prints coloured log message with timestamp
print_tables(file_type)	Back-end	Utility	function	script to look through current working directory and display any found files matching file type (choice between .csv, .json) and prints data from files (tables) to terminal
MyStorage(FileSystemStorage)	Back-end	Database	Class	User defined File System Storage Class
get_available_name(self, name, max_length=None):	Back-end	MyStorage	function	Checks if a file name is already in use on storage and creates a converted String filename for the duplicate name, attaching '_!added!'
get_valid_filename(s) -> str	Back-end	Database	function	Returns the given string converted to a string that can be used for a clean filename. Specifically, leading and trailing spaces are removed; other spaces are converted to underscores; and anything that is not a unicode alphanumeric, dash, underscore, or dot, is removed.
upload_path(instance, filename) -> str	Back-end	Database	function	Returns a file directory path naming schema using the filename as parent directory
Report(models.Model)	Back-end	Database	Class	Report database Class Model
filename(self) -> str	Back-end	Report	function	Returns document filename without the .extension
__unicode__(self) - >str	Back-end	Report	function	Returns string representation of document file
__str__(self) -> str	Back-end	Report	function	Returns string representation of Report database Object

Extracted(models.Model)	Back-end	Database	Class	Extracted database Class Model
__unicode__(self) - >str	Back-end	Extracted	function	Returns string representation of extracted file
__str__(self) -> str	Back-end	Extracted	function	Returns string representation of Extracted database Object
ExtractedSerializer2(serializers.HyperlinkedModelSerializer)	Back-end	API	Class	Extracted database Class serializer representation for API request handling
ReportSerializer(serializers.ModelSerializer)	Back-end	API	Class	Report database Class serializer representation for API request handling
ReportViewSet(viewsets.ModelViewSet)	Back-end	API	Class	Report Viewset Class for handling all http requests to Report database object
ExtractedViewSet(viewsets.ModelViewSet)	Back-end	API	Class	Extracted Viewset Class for handling all http requests to Report database object
UploadView(APIView)	Back-end	API	Class	Upload http POST request handler class. Connects the API requests to the Extraction Engine, database and system storage.

Figure 5.5.1: Object dictionary table

6.0 Software Release Report

Throughout development and toward the final release of the project, we will be utilising a conventional sequentially-stepped testing methodology to validate each part of the system performs as expected.

The system testing is broken up into 4 key areas for testing:

Unit testing: Testing each isolated unit of the system independently to evaluate readiness to be incorporated with other areas of the system.

Integration testing: Test and evaluate the quality of functionality and communication between incorporated systems.

System testing: Evaluation of the system functionality from the user's perspective, testing all use cases.

Acceptance testing: Evaluation of the system's quality and compliance with business requirements and preparedness for release.

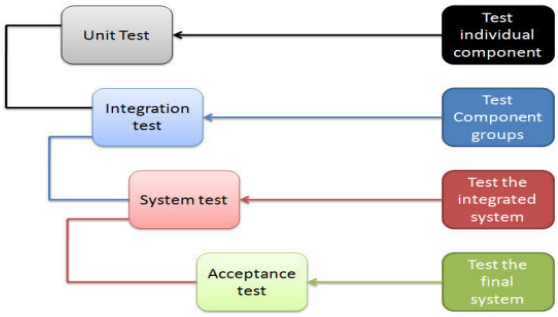


Figure 6.1: System testing methodology diagram

6.1 Unit Testing

Testing that each component functionality works correctly in isolation.

6.1.1 Object Detection System

For measuring the effectiveness of the object detection system, we have devised a simple testing protocol where we take the percentage of missed detections and percentage of false-positive detections and subtract their total against the absolute positive result, where both missed and false-positives are zero.

Such that:

$$Result = 100 - ((MissedTables + FalsePositives)/(TotalExpectedTables) X 100)$$

Detection Model	Input	Confidence threshold	Expected	Missed	False positives	Result
1.0	ANZ-Annual-Report-2018.pdf	30%	125	15	10	79.2%

Figure 6.1.1.1: Object Detection unit test table

Figure 6.1.1.2: Object detection unit test data sample

6.1.2 Data Parsing System

The extraction engine utilises the Python Camelot PDFdata parsing library, which comes with an inbuilt parsing report. For a given data extraction, this report returns %accuracy, %whitespace, the table number on the page, and the page number that the table was located.

For our testing protocol, we are employing both a quantitative and qualitative approach and a simple PASS/ACCEPT/FAIL result metric.

Quantitative: For a given extraction, the higher the %accuracy and the lower the %whitespace, the more likely an extraction was successfully meaningful.

Qualitative: A visual inspection of the extraction to the original document

Test	Source	Page	Type	%accuracy	%whitespace	Visual	Result
1.0	ANZ - Annual Report - 2018.pdf	14	Stream	99.55	10.0	Very good!	PASS
2.0	ANZ - Annual Report - 2018.pdf	16	Stream	99.22	20.0	Very good, but includes an extraneous line at the end	ACCEPT
3.0	ANZ - Annual Report - 2018.pdf	17	Stream	99.29	14.29	Very good!	PASS
3.1	ANZ - Annual Report - 2018.pdf	17	Stream	100.0	31.82	Good, but the table has column spanning cells that are captured as individual cells	ACCEPT
4	ANZ - Annual Report - 2018.pdf	18	Stream	88.95	75.57	BAD. The table was missed and instead, an infographic was captured and parsed.	FAIL
5.0	ANZ - Annual Report - 2018.pdf	19	Stream	99.72	16.67	Very good!	PASS
5.1	ANZ - Annual Report - 2018.pdf	19	Stream	100.0	0.0	BAD. All data captured as single column - difficult table due to lack of clear column/row demarcation	FAIL
5.2	ANZ - Annual Report - 2018.pdf	19	Stream	95.51	76.67	BAD. false-positive: captured an infographic	FAIL

Figure 6.1.2.1: Data parsing unit test table

6.1.3 Back-End System Testing

ID	Functionality	Optional Parameters	Status
1.0	Upload document through API to back-end	start_page end_page	PASS
2.0	Store document uploaded through API on system local storage		PASS
3.0	Build directory tree storage for uploads	document: name	PASS
4.0	Build SQLite database object for uploaded document		PASS
5.0	Retrieve database object from back-end	id :document: name	PASS
6.0	Delete database object and all related table data	id	PASS
7.0	Update database object	any database attribute	PASS
8.0	Pass document between API and extraction engine		PASS
9.0	Pass object location coordinates between object detection system and parsing system		PASS
10.0	Extract data with location coordinates and move to storage tree		PASS
11.0	Build SQLite database object for extracted files (relational to document)		PASS
12.0	Archive all extractions in zip form		PASS
13.0	Retrieve individual extracted files from back-end		PASS
14.0	Retrieve total zipped extracted files from back-end		PASS

Figure 6.1.3.1: Backend testing table

6.2 Integration Testing

Testing that each design component works correctly with its neighbouring components

6.2.1 Extraction Engine Testing

Functional integration testing between the object detection model system and the extraction parsing system. There will be a demonstration with snippets of the object detection model detecting the tables from each valid table and then the extraction engine pulling the table and formatting it to CSV.

```
# do you want to see prediction image?
see_example = True
```

Set the “see_example” to true in order to see the snippets of the object detection model’s table detection in each table.

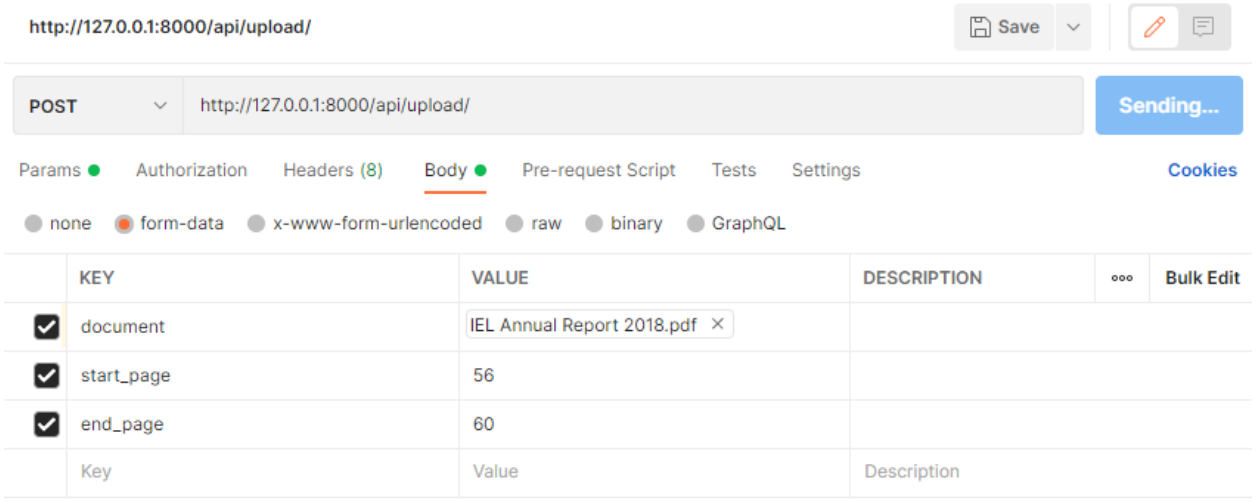


Figure 6.2.1.1: Uploading document in the API

IEL Annual Report 2018.pdf was used with page numbers 56 to 60 inclusively for the testing purpose.

Figure 6.2.1.2: Object Detection System detecting Table

Snippets of the pages that the objection detection model went through. It annotated with borders where it found tables from pages 56 to 60 (left to right).

file name	IEL_Annual_Report_2018.pdf
total pages	116
start page	56
end page	60
output types	['csv', 'json']

```
[24/Oct/2021 03:33:06] "[INFO] multiprocessing using 8 cpu cores"
[24/Oct/2021 03:33:06] "[INFO] starting extractions for pages 56 to 60..."
[24/Oct/2021 03:33:18] "[DEBUG] table parsing_report: {'accuracy': 99.45, 'whitespace': 30.95, 'order': 1, 'page': 58}"
[24/Oct/2021 03:33:20] "[DEBUG] table parsing_report: {'accuracy': 99.87, 'whitespace': 20.0, 'order': 1, 'page': 60}"
[24/Oct/2021 03:33:21] "[DEBUG] table parsing_report: {'accuracy': 99.52, 'whitespace': 42.86, 'order': 1, 'page': 59}"
[24/Oct/2021 03:33:22] "[DEBUG] table parsing_report: {'accuracy': 95.29, 'whitespace': 32.14, 'order': 1, 'page': 56}"
[24/Oct/2021 03:33:23] "[INFO] finished extracting"
[24/Oct/2021 03:33:23] "[DEBUG] camelot parsing report accuracy avg: 98.53%"
[24/Oct/2021 03:33:23] "[INFO] collecting table data for HTTP response..."
[24/Oct/2021 03:33:23] "[INFO] IEL_Annual_Report_2018-CSV.zip created"
[24/Oct/2021 03:33:23] "[INFO] finished collecting table data for HTTP response"
[24/Oct/2021 03:33:23] "[INFO] database updated"
[24/Oct/2021 03:33:23] "[INFO] processed pdf stats
```

file name	IEL_Annual_Report_2018.pdf
total pages	116
start page	56
end page	60
output types	['csv', 'json']
tables found	4

```
[24/Oct/2021 03:33:23] "[INFO] extraction completed in" 17.79 seconds"
[24/Oct/2021 03:33:23] "POST /api/upload/ HTTP/1.1" 201 156
```

Figure 6.2.1.3: Document Processing in the Extraction Engine

The extraction Engine returned positive in extracting the 4 tables that were detected by the Object Detection System. It created a directory which consists all of the extractions in both csv and json like the image below:

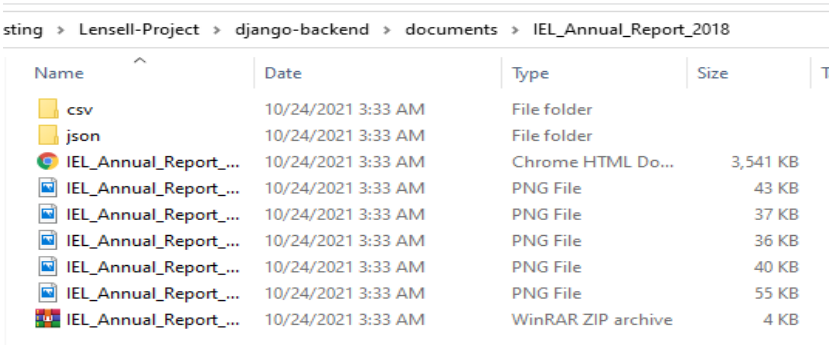


Figure 6.2.1.1: Directories

	A	B	C	D
1	0	1	2	3
2			30-Jun-18	30-Jun-17
3		Notes	$'000	$'000
4	Revenue	3	487,155	394,187
5	Expenses	4.1	-397,859	-325,822
6	Depreciation and amortisation		-13,114	-7,141
7	Finance income		370	326
8	Finance costs	4.2	-2,424	-1,043
9	Share of loss of associate		-258	-
10	Profit for the year before income tax expense		73,870	60,507
11	Income tax expense	5	-22,389	-18,996
12	Profit for the year		51,481	41,511
13	Profit for the year attributable to:			
14	Owners of IDP Education Limited		51,524	41,511
15	Non-controlling interests		-43	-
16			51,481	41,511
17			30-Jun-18	30-Jun-17
18		Notes		
19	Earnings per share for profit attributable to ordinary equity holders			
20	Basic earnings per share (cents per share)	7	20.59	16.58
21	Diluted earnings per share (cents per share)	7	20.14	16.2
22	The above statement should be read in conjunction with the accompanying notes.			

Figure 6.2.1.4: The Output

This is the outcome of the extraction of page 58 of the document. Both the Object Detection model and the Extraction engine are functional and well integrated.

6.3 System Testing

Step by step run-through of all expected and designed functionality of the end system from the user's perspective.

6.3.1 Extraction Engine Quality Testing

Data quality testing for the extraction engine system.

Quality Testing was carried out manually for the extraction engine system. The extracted data and the document table is matched to find out the accuracy of the table. Above 80% accuracy was considered to be a Good quality extraction.

Test	Source	Page	Type	%accuracy	%whitespace	Visual	Result
1.0	ABC Annual Report 2018	4	Stream	100	10	Perfect, but contains 2 extra lines at the end	PASS
2.0	ABC Annual Report 2018	33	Stream	75	15	Data Extracted perfectly, but some data were clustered into 1 cell	FAIL
3.0	ABC Annual Report 2018	34	Stream	100	25	Perfect, but contains 2 extra lines at the end	PASS
4.1	ABC Annual Report 2018	35	Stream	97	15	Perfect, but contains 1 extra lines at the end, and Superscript is put into another cell	PASS
4.2	ABC Annual Report 2018	35	Stream	100	10	Perfect, but contains 2 extra lines at the end/beginning	PASS
5.1	ABC Annual Report 2018	41	Stream	98	15	Perfect, but contains 2 extra lines at the end. 1 line in the table was generated twice but does not affect the table as easily noticeable.	PASS
5.2	ABC Annual Report 2018	41	Stream	99	5	Perfect, but contains 1 extra lines at the end	PASS

Figure 6.3.1.1 System test table

A table was created to document the Quality of the extractions for each document, and afterwards ruling out the bad quality extractions and false positives, the Accuracy for the whole document is calculated by this formula:

$$Result = 100 - ((MissedTables + FalsePositives)/(TotalExpectedTables) X 100)$$

Detection Model	Input	Confidence threshold	Expected	Missed	False positives	Result
1.0	ANZ-Annual-Report-2018.pdf	30%	125	15	10	79.2%
1.0	ABC Annual Report 2018	30%	104	20	-	80.8%
1.0	FXJ Annual Report 2018.pdf	24%	143	32	1	83%
1.0	ValidTables.pdf	24%	109	10	1	90%
1.0	IEL_Annual_Report_2018.pdf	24%	106	5	2	93%

Figure 6.3.1.1: Accuracy table for extraction

Each Document's extraction accuracy is noted in a table with the necessary metrics to describe the accuracy and quality of the output.

6.3.2 Full-System Functional Testing (User Perspective)

ID	Use Case	Steps	Input Parameters	Input	Expected Output	Status
1.0	Upload a PDF document	Drag and drop relevant files OR Upload File and Submit via relevant buttons	PDF file hover over submission area OR select Upload File, click selection of file, press SUBMIT	.PDF	NA to web users, API level access (Developer/Admin) will return 200 or 500.	Completed
2.0	Extract data from PDF to CSV	N/A, Internal	.PDF File	.PDF	.CSV	Completed
3.0	Download/ save processed PDF	Press Download Button/ Automatic for API user	Mouse	N/A	Download of ZIP file/ saved in directory	Completed
4.0	Search the history of previous extractions	Click on extracted history (only available at Admin level/API user level)	Mouse input/commands	N/A	Web page that shows history of previously extracted data or use command line for API users	Completed (API and admin level)
5.0	Retrieve previous extractions/reports.	Click on Zip CSV under each report (admin level) API user can use command line	Mouse Input/commands	N/A	Downloads the zip/saves the zip in local directory	Completed (API and admin level)
6.0	Delete previous extractions/reports.	Check the boxes on the files to be deleted and choose Delete in Action then confirm (admin) API user can use command line	Mouse Input/commands	N/A	Deletes the file	Completed (API and admin level)

Figure 6.3.2.1: System test table

6.4 Acceptance Testing

The acceptance requirements are based on the user stories that were developed in the initial creation of the SMD based on the gathering of requirements from Lensell. The criticality aspect of each user story is based on feedback received throughout development.

Acceptance Requirement	Critical		Test Result		Comments
	Yes	No	Pass	Fail	
Upload document through API to back-end storage	✓		✓		Fully functional
Extract data from PDF to CSV	✓		✓		Multiple approaches used
Save processed PDF document data		✓		✓	Works only on Administrative Access
Search the history of previous extractions/reports		✓		✓	Works only on Administrative access
Retrieve previous extractions/reports.		✓		✓	Works only on Administrative access
Delete previous extractions/reports.		✓		✓	Works only on Administrative access

Figure 6.4.1: Acceptance test table

6.4.1 UX Quality

For testing the User Experience we opted to use the BASIC Framework for UX quality testing, based on the data analytics produced later in the document.

	Goal	Status	Basis
B	Beauty. Is it aesthetically pleasing?	Achieved	Based on Usability Testing
A	Accessibility. Can everyone use it?	Achieved.	Testing the beta front end code on multiple browsers.
S	Simplicity. Does it make life easier?	Achieved.	Performs required functionality of the project.
I	Intuitive. Is it easy to use?	Achieved.	Based on Usability Testing.
C	Consistency. Does it match systems?	Achieved.	Based on CSS styling that has been back tested across multiple screen sizes and browsers.

Figure 6.4.1.1 BASIC framework table

6.4.2 System Usability Questionnaire

The following questions were asked to individuals who participated in the Usability Testing program.

Question	Response Options
Q1. How did you upload the files?	Upload or Drag and Drop?
Q2. How easy was it to upload a document?	Metric of 1 - 5, 5 extremely easy
Q3. How easy was it to download extracted files?	Metric of 1 - 5, 5 extremely easy
Q4. How accurate was the extraction?	Metric of 1 - 5, 5 being highest accuracy
Q5. How satisfied are you with the extraction wait time?	Metric of 1 - 5, 5 being highly satisfied
Q6. Would you use this site for its intended	Yes/No

purpose again?	
Q7. If you could change or add one thing about this website, what would it be?	A qualitative open ended question, capped at 250 words
Q8. Rate your overall experience	Metric of 1 - 5, 5 being very good

Figure 6.4.2.1 Questionnaire

6.4.3 Analytics from Usability Testing

The system usability questionnaire was filled by a range of correspondents including fellow students from the university as well as friends and family of group members involved in the project.

For the quantitative metrics, we've graphed the average user response.

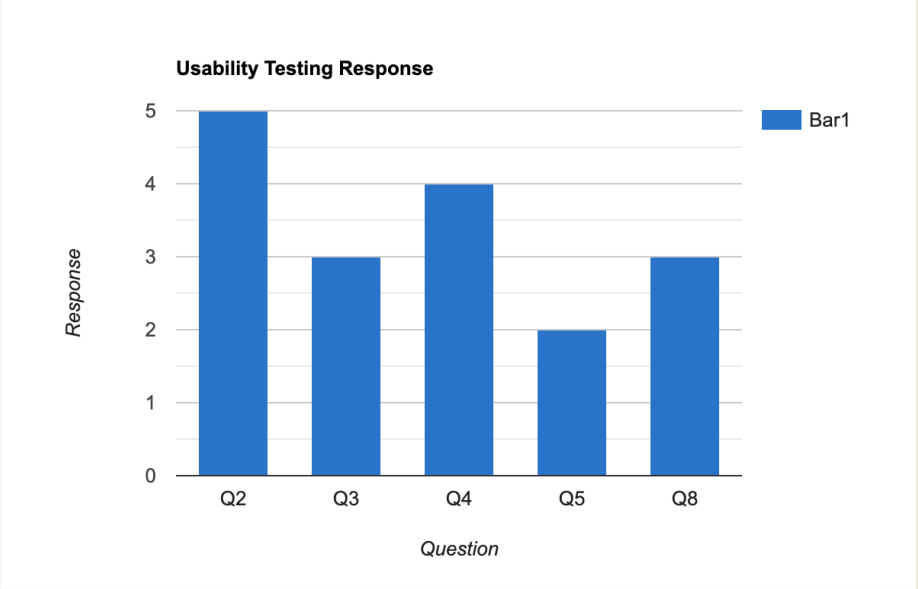


Figure 6.4.3.2 Data Table

For the qualitative metrics, we've graphed the average user response based on responses we've deemed to be positive, again measuring on the carried forward 1 to 5 scale where 1 is not an unsatisfied/negative response to 5 being a highly satisfied/positive response.

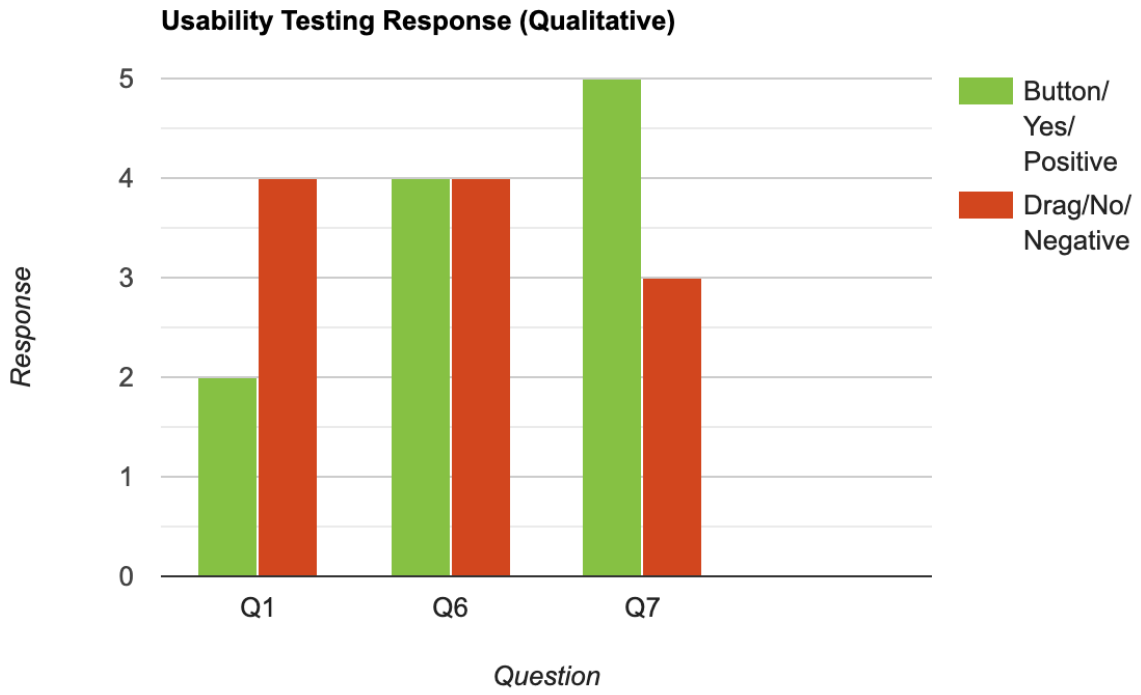


Figure 6.4.3.3 Data Table

7.0 Additional Designs and Changes

Features and/or design additions made to the system as part of the Agile development iterative strategy will be covered here.

7.1 Future Design Possibilities

Design features that may be added to the system at a later date.

1. **Docker deployment:**
   As an additional deployment feature, we are considering the possibility of encapsulating our system inside a ready-to-deploy docker container.

   The benefit of adding such a feature is that docker can package an application and its dependencies in a virtual container that can run on any Linux, Windows, or macOS computer. This enables easy and fast deployment in a variety of locations and systems, including cloud solutions.

2. **Amazon Web Services (AWS) integration:**
   We are considering the integration of Amazon Web Services (AWS) as an additional useful feature.

   The benefit of adding AWS is that it allows a user to preserve their extraction history over time and across many systems and instances of the system.

3. **Further YOLOv3 model training, which would allow greater utilisation of Camelot :**
   At the movement, our inferencing model only contains one Class of Object that it is looking for, Table; this Table Class encompasses all descriptions of what a table could be. However, it's possible to train a YOLOv3 to distinguish between types of tables, specifically bordered tables and non-bordered tables. With such a model, it would then be possible to take greater advantage of Camelot's inbuilt heuristic *flavours* for these table types: *lattice* for bordered tables; *stream* for non-bordered

4. **Ensemble Object detection model voting strategies**

   Ensemble voting is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem.

   In our problem, for any intelligent inference engine we train there are always going to be non-trivial edge cases and false-positives that creep into the system. By combining many trained models and giving them a voting strategy by which they can agree on detections, it's possible to improve the predictive performance of the system overall and reduce the likelihood of a poor detection or a missed one.

The voting strategies available to such an ensemble of models are:

Affirmative.	Any detections made by any model are valid,
Consensus.	Only detections that have a majority agreement between models are valid
Unanimous.	Only detections that all agree on are valid.

Glossary

Administrative Access
Only available to users that have access to the direct django administrative web page with relevant username and login credentials. This may also be referred to as Developer access to the direct API.

API - Application Programming Interface
An API Is a resource manager that encompasses a set of functions and procedures, allowing the transfer of system data and resources between an operating system, application, or other service and third-party external systems.

Architecture Diagram
An architecture diagram is a graphical representation of a set of concepts, that are part of a software system architecture, including respective elements and components.

Code on Demand
Code on demand is any technology that sends executable software code from a server computer to a client computer upon request from the client's software.

ER - Entity Relationship Diagram
A graphical representation of an information system that depicts the relationships among people, objects, places, concepts, or events within the system.

Flow of Interaction Diagram
The interaction flow expresses a change of state of the user interface.

Prototype
Engineering a prototype goes beyond sketches or physical mock-ups by creating a working proof-of-concept of the product.

Sequence Diagram
A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario

Use Case
A usage scenario for a piece of software; often used in the plural to suggest situations where a piece of software may be useful.

User Stories
A user story is an informal, natural language description of the features of a software system, written from the perspective of an end-user of the system

WireFrames

Are images that display the design elements of a website or webpage; typically used for planning and presenting a site's possible final design, structure, and functionality.

Index