CS258

SOFTWARE REQUIREMENTS SPECIFICATION

for

Document Parser

Prepared by

cse210001016
cse210001052
cse210001068
cse210001071

Course Instructor: Puneet Gupta

May 3, 2023

Contents

1 Introduction

1.1 Purpose

The purpose of this document parser is to create an application that extracts text from an image, then search for a specific phrase along with its position in the given image. Document Parsing involves examining the data in a document and extracting useful information. It is essential for companies as it reduces a lot of manual work.A parser uses colors and fonts to highlight different items in a programming language source document.

1.2 Project Scope

A document parser can automate the process of extracting information from large volumes of unstructured text documents, which can save time and reduce manual labor costs. It can reduce the risk of errors and inconsistencies that can occur when information is manually extracted from documents. It can provide a scalable solution for organizations that need to process a large number of documents.
We will be focusing on developing a high-quality, accurate, and efficient software tool that can improve the process of extracting data from documents, and that meets the needs and expectations of its users and stakeholders.

2 Overall Description

2.1 Product Perspective

The context of a document parser is the need to extract information from unstructured text documents, such as contracts, reports, and emails. Traditionally, this task has been performed manually, which is time-consuming, error-prone, and costly. With the increasing amount of unstructured text data being generated, there is a growing need for automated solutions that can extract relevant information from these documents.

The origin of document parsers can be traced back to the field of natural language processing (NLP). NLP is a branch of artificial intelligence (AI) that focuses on the interaction between computers and human language. In the early days of NLP, researchers developed techniques for parsing sentences and extracting grammatical structures from text. These techniques were then extended to more complex text structures, such as paragraphs and documents.

In the late 1990s and early 2000s, there was a growing interest in developing document parsers that could automatically extract information from unstructured text documents. The emergence of the internet and the explosion of online content provided a rich source of unstructured text data that could be mined for insights.

Today, document parsers are used in a wide range of applications, such as contract management, fraud detection, and sentiment analysis. They have become an essential tool for organizations that need to extract insights from large volumes of unstructured text data.

The document parser can exchange data with other components of the larger system in a specific format, such as jpg, png, jpeg, CSV or pdfs.It can receive input data in a specific format, such as email messages or scanned images. It can provide output data in a specific format, such as a database record or a document template.

2.2 Product Functions

The functions of a document parser includes:

1. **Text extraction**: The document parser must be able to extract text from the input document, even when the text is embedded within images, tables, or other

non-textual elements.

2. **Text normalization**: The document parser must normalize the extracted text to remove noise and inconsistencies, such as extra white spaces, punctuation, and capitalization.

3. **Keyword search**: A document parser can allow users to search for specific keywords or phrases within a document, making it easier to find relevant information.

4. **Extraction of structured data**: A document parser can extract structured data from unstructured documents such as PDFs, Word documents, and web pages. This can include information such as names, addresses, phone numbers, dates, and other types of structured data.

5. **Summarization of documents**: A document parser can summarize the content of a document to provide a brief overview of its key points.

6. **Keyword extraction**: A document parser can extract important keywords from a document to aid in indexing and searching.

7. **Language translation**: A document parser can translate the content of a document from one language to another.

8. **Image analysis**: A document parser can analyze images embedded in a document, such as product photos, to extract information or identify people.

9. **Text-to-speech conversion**: A document parser can convert text from a document into speech, allowing users to listen to a document instead of reading it.

2.3 User Classes and Characteristics

Some general user classes and characteristics of a document parser may include:

1. **Business users**: Business users may use the document parser to extract information from unstructured documents for various purposes, such as automating data entry, streamlining business processes, and improving data accuracy.

2. **Data analysts**: Data analysts may use the document parser to extract and analyze data from large volumes of unstructured documents, such as social media posts, customer reviews, and news articles.

3. **Developers**: Developers may use the document parser to build applications that require NLP functionality, such as chatbots, virtual assistants, and search engines.

4. **Researchers**: Researchers may use the document parser to analyze large volumes of text data for various research purposes, such as sentiment analysis, topic modeling, and opinion mining.

The characteristics of users who would use a document parser may include:

1. **Familiarity with the domain**: Users who will be working with the document parser should have a good understanding of the domain in which the document parser is being used, such as legal, financial, or healthcare.
2. **Technical expertise**: Depending on the complexity of the document parser and the application it is being used for, users may need to have technical skills such as programming, data analysis, and machine learning.
3. **Attention to detail**: Users who will be working with the document parser need to be detail-oriented to identify and correct any errors or inaccuracies in the extracted data.

2.4 Operating Environment

1. **Operating System**: The document parser will be compatible with the operating system(s) on which it will be deployed, such as Windows, Linux, or macOS.
2. **Programming Language**: The document parser will be developed in Python, or Javascript.
3. **Memory Requirements**: The document parser may require significant memory resources, particularly if it needs to store and manipulate large amounts of data.
4. **Input and Output Formats**: The document parser will be able to handle input documents in a variety of formats, such as PDF, Microsoft Word, HTML, or plain text. The output format will depend on the specific requirements of the application using the parser.
5. **Third-Party Libraries**: The document parser will be using third-party libraries for tasks such as text processing, machine learning, or natural language processing, Api's.
6. **Security**: The user can choose whether or not to store large documents in our database.

2.5 Design and Implementation Constraints

1. **Data format and structure**: Document parsers must be able to handle a wide range of document formats and structures, such as PDF, Microsoft Word, HTML, and plain text. The parser must be designed to handle these different formats and structures and extract information in a consistent manner.

2. **Processing time and speed**: Document parsers must be able to process large volumes of data quickly and efficiently, especially when dealing with real-time data streams. This may require the use of parallel processing, caching, and other optimization techniques to speed up processing time.

3. **Accuracy and quality**: Document parsers must be able to accurately extract the relevant information from documents, while minimizing errors and inaccuracies. This may require the use of sophisticated NLP techniques, such as machine learning and semantic analysis, to improve accuracy.

4. **Scalability and resource usage**: Document parsers must be designed to handle large volumes of data and scale up or down as needed. This may require the use of cloud computing services or distributed computing architectures to efficiently use resources.

5. **Integration with other systems**: Document parsers must be able to integrate with other systems, such as databases and data warehouses, to store the extracted information in a structured format. This requires careful consideration of data formats, APIs, and data transfer protocols.

6. **Security and privacy**: Document parsers must be designed to ensure the security and privacy of the extracted information, especially when dealing with sensitive data such as financial or medical records. This may require the use of encryption, access controls, and other security measures to protect data.

2.6 User Documentation

The documentation components as detailed Readme of each functionality provided , tutorial of some complex features. The contact of the developers will be shared for online support.

2.7 Assumptions and Dependencies

Here are some potential assumptions and dependencies that should be considered for a document parser project: We would be using some libraries and models whose weights are publicly available.Also connecting the ML models from back end to the front end must be done seamlessly and the connection must be fast.

1. **Availability of Data**: The document parser project assumes that the required data will be available for processing. This could include documents in a specific format or from a particular source. If the data is not available or is not in a suitable format, it could impact the project.

2. **Software and Hardware Dependencies**: The document parser project may be dependent on specific software and hardware tools or platforms. If these tools or platforms are not available or cannot be used, it could impact the project.

3. **External Systems**: The document parser project may need to integrate with external systems, such as databases.

4. **Performance and Scalability**: The document parser project assumes that the system will be able to handle the expected volume of documents and process them efficiently. If the system is not scalable or does not perform as expected, it could impact the project.

5. **User Requirements**: The document parser project assumes that the user requirements have been accurately identified and documented.

3 External Interface Requirements

3.1 Hardware Interface Requirements

1. **Processor**: The processor is one of the key components that determine the speed and efficiency of your document parser. Depending on the complexity and size of the documents you plan to parse, you may need a powerful processor such as Intel Core i5 or i7.

2. **RAM**: Random Access Memory (RAM) is used to temporarily store data during the parsing process. The more RAM you have, the faster your system can process large documents. For a document parser, you may need at least 8 GB of RAM, but 16 GB or more is recommended for larger documents.

3. **Storage**: You will need a sufficient amount of storage space to store the documents you plan to parse, as well as the parsed data. A Solid-State Drive (SSD) is recommended for faster access and data transfer speeds.

4. **Graphics card**: While a graphics card is not necessary for a document parser, it can be useful if you plan to use any visual processing tools or interfaces.

5. **Network connectivity**: Depending on the use case, you may need a high-speed internet connection to download or access documents from remote servers.

6. **Backup system**: It is always recommended to have a backup system in place to ensure data integrity and to prevent data loss due to hardware failures or other issues.

3.2 Software Interfaces

Software interfaces can be used in document parser systems to enable the parser to interact with other software applications, services, or APIs. Software interfaces can help automate the document parsing process and make it more efficient, by allowing documents to be transferred and processed seamlessly between different applications.

Here are some examples of how software interfaces can be used in a document parser system:

1. **Application Programming Interfaces (APIs)**: APIs can be used to enable the document parser to interact with other software applications, such as databases, content management systems, or other document processing tools. For example,

an API can be used to extract data from a specific field in a database and use it to populate a document template.

2. **Web Services**: Web services can be used to enable the document parser to interact with other web-based applications or services, such as online document storage services or document collaboration tools. For example, a web service can be used to extract text from a document stored in the cloud and use it to populate a form.

3. **python libraries that would be used to enable machine learning processes**: NLTK, spaCy, Docx2Txt, Numpy, Pillow, Scipy, Pandas, OpenCV, KerasNLP ,Pytesseract, pdf2image, PyMuPDF, LayoutParser, Poppler, Detectron, tensorflow, matplotlib etc.

4. **Some other libraries**: React.js, Express.js, embed.js etc.

By using software interfaces in document parser systems, businesses and organizations can automate and streamline their document processing workflows, making them more efficient and accurate, while reducing the risk of errors and increasing productivity.

4 System Features

4.1 Stimulus/Response Sequences

Stimulus/Response sequences are interactions between the user and the system. Here are some example Stimulus/Response Sequences for a document parser:

User selects a document for parsing Stimulus: User clicks on "Select Document" button Response: Document selection dialog box opens

User sets parsing options Stimulus: User selects parsing options such as file format and language Response: Parser applies the selected options to the document and begins parsing

User reviews parsed data Stimulus: User clicks on "View Parsed Data" button Response: Parsed data is displayed in a separate window

User edits parsed data Stimulus: User clicks on "Edit Parsed Data" button Response: Parsed data is displayed in an editable format

User saves parsed data Stimulus: User clicks on "Save Parsed Data" button Response: Parsed data is saved in the selected file format and location

User exports parsed data Stimulus: User clicks on "Export Parsed Data" button Response: Parsed data is exported in the selected format, such as CSV or JSON

User requests help Stimulus: User clicks on "Help" button Response: Help documentation or support options are displayed

User closes the document parser Stimulus: User clicks on "Close" button Response: Document parser application closes

4.2 Functional Requirements

Here are some functional requirements for a document parser:

1. **Support for multiple file types**: The document parser should be able to parse multiple file types such as PDFs, DOCX, and TXT.
2. **Data extraction**: The parser should be able to extract text, images, tables, and other relevant data from the documents.
3. **Data normalization**: The parser should be able to normalize the extracted data to make it consistent and easy to work with.
4. **Data validation**: The parser should validate the extracted data against predefined rules and conditions to ensure its accuracy.

5. **Customizable rules**: The parser should allow users to define custom rules for data extraction and validation.
6. **Scalability**: The parser should be able to handle large volumes of documents and process them efficiently.
7. **Error handling**: The parser should be able to handle errors and exceptions gracefully, and provide clear error messages to users.
8. **Integration**: The parser should be able to integrate with other systems and tools as needed, such as document management systems or database applications.
9. **Security**: The parser should ensure the security of the documents and the extracted data, and prevent unauthorized access or tampering.
10. **User interface**: The parser should have a user-friendly interface that allows users to easily upload documents, view the extracted data, and manage the parsing process.

These are just some of the functional requirements that may be important for a document parser. The specific requirements may vary depending on the project goals and stakeholders' needs.

5 Other Nonfunctional Requirements

5.1 Security Requirements

Here are some safety requirements that should be considered for a document parser:

1. **Data Security**: The document parser should have appropriate security measures to ensure that the extracted data is protected from unauthorized access or tampering. This may include encryption, access controls, and monitoring tools.
2. **Error Handling**: The document parser should handle errors and exceptions gracefully to prevent data loss or corruption. Clear error messages should be provided to users to help them resolve issues.
3. **Scalability**: The document parser should be able to handle large volumes of documents and process them efficiently. This will prevent system crashes and reduce the risk of data loss.
4. **Compatibility**: The document parser should be compatible with the systems and tools it will be integrated with. This will prevent compatibility issues and reduce the risk of data loss or corruption.
5. **User Training**: Users should be trained on how to use the document parser properly to prevent errors and reduce the risk of data loss or corruption.
6. **Testing**: The document parser should be thoroughly tested before deployment to ensure that it meets the safety requirements and performs as expected. This will reduce the risk of data loss or corruption.
7. **Regular Maintenance**: The document parser should be regularly maintained to ensure that it remains secure, efficient, and effective. This may include software updates, security patches, and hardware maintenance.
8. **Backup and Recovery**: The document parser should have appropriate backup and recovery procedures in place to prevent data loss or corruption in the event of a system failure or other disaster.

These safety requirements will help ensure that the document parser is secure, reliable, and effective, and will reduce the risk of data loss or corruption. It's important to regularly review and update these safety requirements to ensure that they remain relevant and effective.

5.2 Software Quality Attributes

Software quality attributes for a document parser may include:

1. **Accuracy**: The document parser will accurately identify and extract relevant data from documents, while minimizing errors and false positives.
2. **Performance**: The speed and efficiency of the parser in processing documents, especially when dealing with large volumes of data.
3. **Robustness**: The document parser will be able to handle different document formats, structures, and languages(if possible), as well as variations in data quality and input errors.
4. **Maintainability**: The ease with which the parser can be modified, updated, and improved over time, as new document formats or features become available.We will be able to modify, update and improve over time if new document formats or features become available.
5. **Usability**: It will be able to provide meaningful and actionable output to end-users.
6. **Security**: We will try to protect against unauthorized access, prevent data breaches, and ensure the confidentiality, integrity, and availability of data.
7. **Testability**: The ease with which the parser can be tested, including the ability to create test cases and simulate real-world scenarios.
8. **Interoperability**: The document parser will be able to interact with other software systems and components, using common protocols and standards, and to integrate seamlessly into existing workflows and applications.
9. **Scalability**: he document parser will be able to handle increased document processing loads, by adding more resources or components.

6 Verification and validation

6.1 Acceptance criteria

Acceptance criteria are a set of predefined conditions or requirements that a document parser must meet in order to be considered acceptable for use. The use of acceptance criteria in document parser development can help ensure that the final product meets the needs of its users and performs its intended functions effectively. By using acceptance criteria in document parser development, developers can ensure that the final product meets the specific needs and requirements of its users. This can help improve user satisfaction and increase the overall effectiveness of the document parser.

6.2 Verification and validation procedures

Verification and validation procedures are important for ensuring the accuracy, reliability, and effectiveness of a document parser. Verification refers to the process of ensuring that the parser meets its specified requirements and functions correctly, while validation refers to the process of ensuring that the parser meets the needs of its users and performs its intended functions effectively.Here are some examples of verification and validation procedures that can be used in document parser development: Unit Testing, Integration Testing, System testing, User Acceptance testing, performance testing, security testing etc. By implementing these verification and validation procedures, document parser developers can ensure that their product meets both the technical requirements and user needs, leading to a more effective and reliable tool for document analysis and processing.

6.3 Test cases

Test cases are an important part of the document parser development process as they help ensure that the parser functions correctly and meets its requirements.

1. **Input Validation Test Cases**: These test cases ensure that the parser correctly handles invalid input, such as malformed documents, corrupted files, or documents in non-supported formats.

2. **Document Extraction Test Cases**: These test cases ensure that the parser correctly extracts the required data fields from a given document, and handles variations in formatting, layout, and structure.

3. **Document Classification Test Cases**: These test cases ensure that the parser correctly identifies the type of a given document, such as invoices, receipts, contracts, or resumes.

4. **Performance Test Cases**: These test cases ensure that the parser can handle large volumes of documents, and can process documents quickly and efficiently.

5. **Integration Test Cases**: These test cases ensure that the parser can integrate with other systems

6. **User Acceptance Test Cases**: These test cases ensure that the parser meets the needs of its intended users, and is easy to use, intuitive, and provides accurate results.

7. **Security Test Cases**: These test cases ensure that the parser is secure and protected from potential security threats.

By implementing these and other test cases in the document parser development process, developers can ensure that the parser is reliable, accurate, and meets the needs of its intended users.

7 Project management

7.1 Roles and responsibilities

Roles and responsibilities in a document parser project can vary depending on the size and complexity of the project, as well as the specific needs of the organization. General roles and responsibilities in the project are Project Manager, Business Analyst, Developer, Quality Assurance (QA) Engineer, User Experience (UX) Designer, Technical Writer, System Administrator etc. By defining clear roles and responsibilities for each team member involved in the document parser project, organizations can ensure that the project is completed on time, within budget, and meets its objectives.

7.2 Development plan

general development plan for a document parser project:

1. **Define Requirements**: Identify the requirements for the document parser, including the types of documents to be processed, the specific data fields to be extracted, the workflow to be supported, and any other necessary features.
2. **Research and Select Libraries**: Research and select the appropriate libraries for document parsing based on the specific requirements needs of the project.
3. **Design System Architecture**: Design the system architecture for the document parser. This should include the overall system flow, module interactions, and any interfaces that need to be developed.
4. **Develop the Parser**: Develop the document parser using the selected libraries and the system architecture designed in step 3. Ensure that the parser is tested at each stage of development and that the functionality meets the requirements.
5. **Develop the User Interface**: Develop the user interface for the document parser. The interface should be user-friendly and allow users to easily upload documents, view extracted data, and perform any necessary actions.
6. **Test and Validate**: Test the document parser thoroughly to ensure that it meets the requirements and works correctly. Validate the parser against a range of documents to ensure that it works effectively across different document types.
7. **Deployment**: Deploy the document parser to the target environment. Ensure that it is properly installed and configured, and that users have the necessary permissions and access to use it.

8. **Maintenance and Support**: Provide ongoing maintenance and support for the document parser, including bug fixes, updates, and user support.

By following these steps, we can create a document parser that meets the requirements of our organization and provides an effective and efficient solution for processing documents.

7.3 Schedule

- **Week 1**:
  - Day 1-2: Research and select appropriate libraries for document parsing
  - Day 3-5: Plan the system architecture and data flow for the document parser
- **Week 2**:
  - Day 1-3: Develop and test the document parser using the selected libraries
  - Day 4-5: Develop and implement necessary algorithms for data extraction and processing
- **Week 3**:
  - Day 1-2: Develop a simple user interface for the document parser
  - Day 3-4: Thoroughly test and validate the document parser to ensure it meets the requirements
  - Day 5: Prepare documentation for the project
- **Week 4**:
  - Day 1-2: Deploy the document parser in the target environment
  - Day 3-4: Provide user training and support for the document parser
  - Day 5: Review the project, gather feedback, and make necessary updates and

7.4 Risks and contingencies

- **Technical Risks**:
  - Risk: The selected libraries and tools may not be suitable for the project requirements. Contingency: Conduct thorough research and testing before selecting any library or tool. Have a backup plan in case the selected library or tool doesn't work as expected.
- **Data Security Risks**:
  - Risk: The extracted data may be vulnerable to security breaches or unauthorized access. Contingency: Implement appropriate security measures such as encryption, access controls, and monitoring tools to protect the extracted data.

8 Bibliography

8.1 Models Used

- Bart Model
- Distillbert uncased Finetuned

8.2 References

- Tesseract
- React JS

8.3 API's

- React-PDF
- React-Highlighter
- React Speech

9 Website

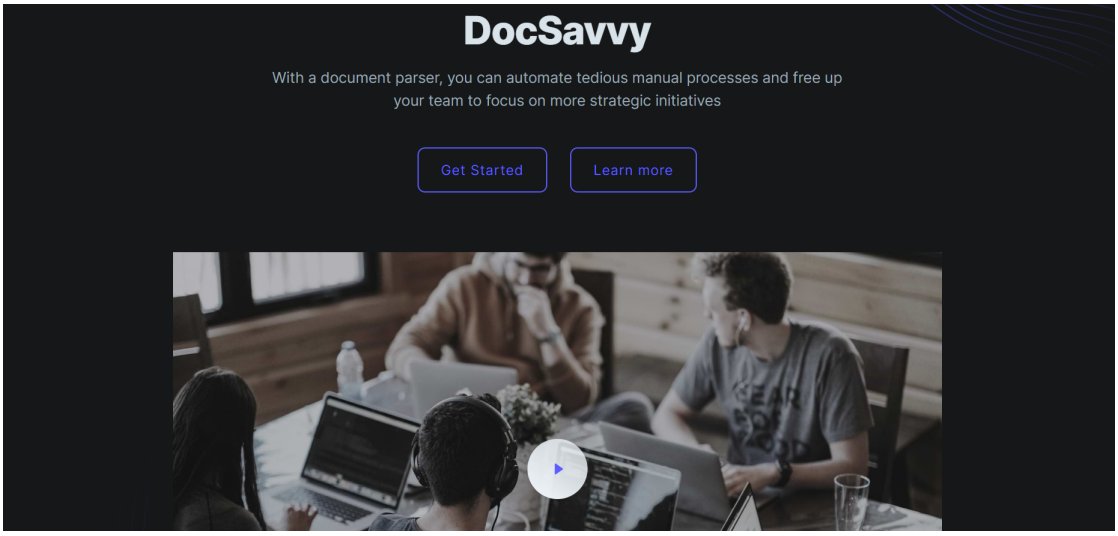

Figure 9.1: Home Page

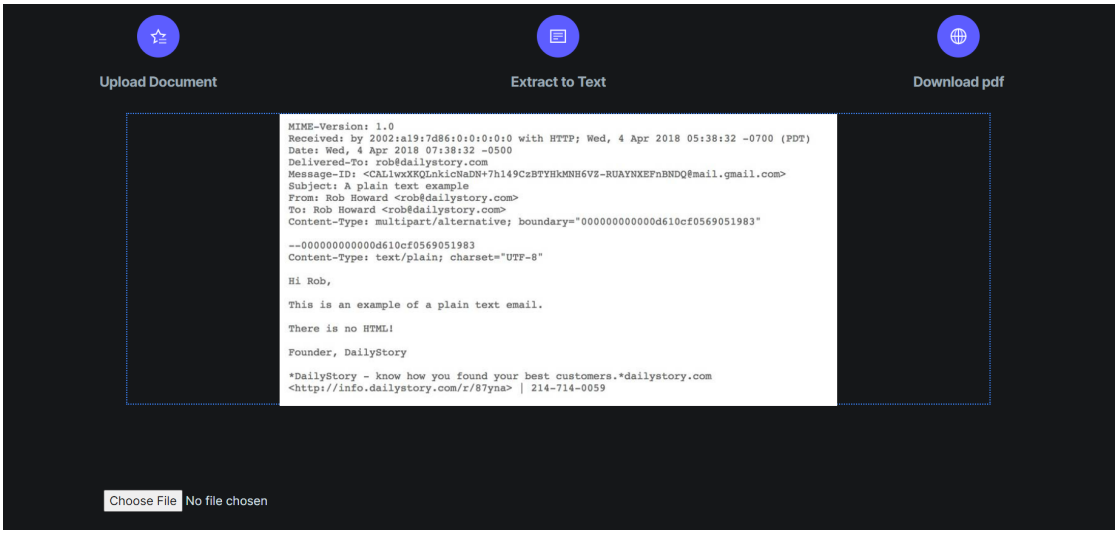

Figure 9.2: Uploaded Image

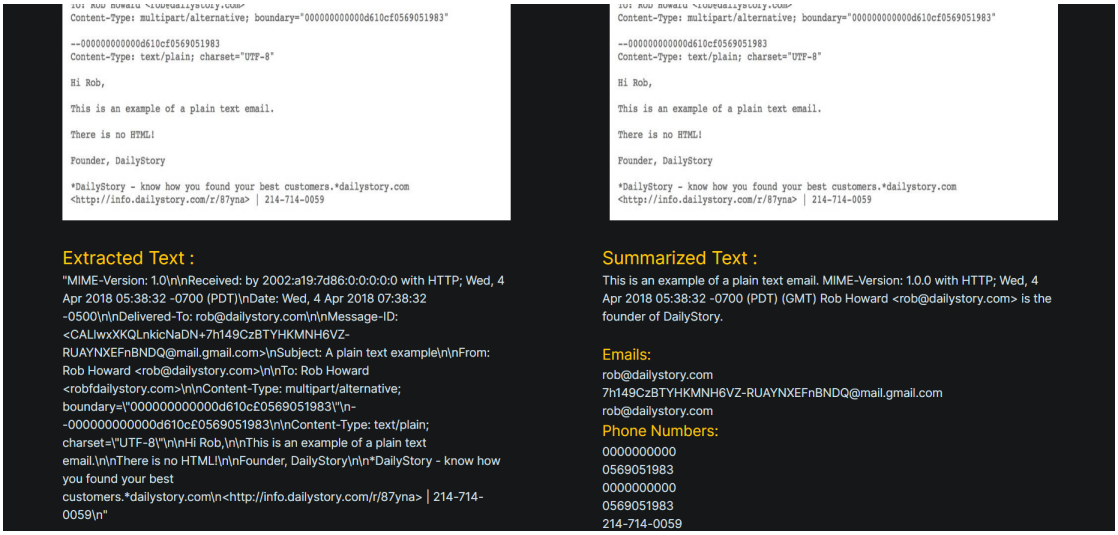

Figure 9.3: Extracted Text

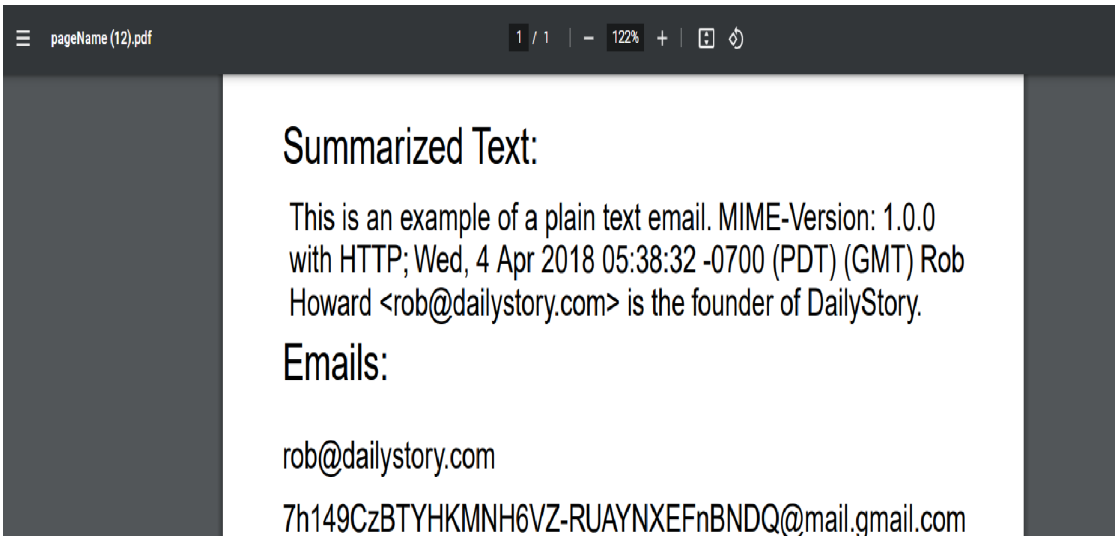

Figure 9.4: Downloaded PDF