Text Extraction from Documents into Elasticsearch

Final Report
sddec22-19

Client: Buildertrend
Advisor: Goce Trajcevski

Team Members - Roles
Bruce Bitwayiki – Backend
Jared Hayashi – Backend
Rushal Sohal – Frontend
Tiffany Mayberry – Frontend

sddec22-19@iastate.edu
sddec22-19.sd.ece.iastate.edu

Revised: December 6, 2022 / V1

Executive Summary

Development Standards & Practices Used

IEEE-29119: We will write Unit Tests for our web application to ensure functionality in each separate piece.

IEEE-12207: We will follow the standard software lifecycle process.

IEEE-7.8: Code of ethics: We will follow ethical development practices.

Summary of Requirements

The application should...

1. be a desktop or web application (constraint)
2. utilize Elasticsearch to store the metadata (constraint)
3. be able to extract and index metadata from several different file types
4. able to support jpg, png, pptx, pdf, docx, xlsx, xls, and txt files
5. match search keywords with text fragments from the files
6. provide the ability for users to upload supported file types

Applicable Courses from Iowa State University Curriculum

- ComS 227, 228, & 327 - Basic programming skills
- ComS 309 - Project development skills
- ComS 319 - User Interface design
- SE 329 - Project management skills
- ComS 363 - Database management skills
- ComS 409 - Requirements discovery

New Skills/Knowledge acquired that was not taught in courses

- React framework
- Working with a client
- Elasticsearch setup

Table of Contents

List of figures/tables/symbols/definitions

1. Team

This section describes the basic overview of our team and skills required for the project.

1.1 Team Members

- Bruce Bitwayiki
- Jared Hayashi
- Rushal Sohal
- Tiffany Mayberry

1.2 Required Skills For The Project

- Backend and database knowledge
- Frontend development for UI
- Communication and documentation
- Familiarity with Agile & waterfall methodology

1.3 Skill Sets Covered By The Team

- Backend and database knowledge - Jared & Bruce
- Frontend development for UI - Tiffany & Rushal
- Communication and documentation - All
- Familiarity with Agile and waterfall methodology - Tiffany, Jared & Bruce

1.4 Project Management Styles Adopted By The Team

Agile + waterfall methodologies and bi-weekly sprints. More details are in the Project Plan.

1.5 Initial Project Management Roles

- Bruce Bitwayiki - Backend documenter, Backend Architecture Design
- Jared Hayashi - Client communication, Backend Architecture Design
- Rushal Sohal - Documentation and reports, Frontend and UI
- Tiffany Mayberry - Faculty Advisor communication, Frontend and UI

2. Introduction

We now describe the problem and discuss the requirements and constraints.

2.1 Problem Statement

Construction managers and workers using Buildertrend's software are unable to quickly find information inside documents they have uploaded quickly. Information may be spread across multiple files over a period of time, making it difficult and time-consuming to manually search through the documents. The time spent looking for the file could be

better spent doing other, more important, tasks. A new way to search and store the file's content needs to be added to Buildertrend's application. This will enable the construction managers and workers to spend the time they would have spent looking for the file on more productive tasks.

2.2 Requirements & Constraints

We now enumerated the requirements, constraints, and stretch requirements. Stretch requirements are outside of the basic functionality of the project, but if time allows, some may be included in the final deliverables.

2.2.1 Requirements

1. Functional Requirements:
    - 1.1. The application shall extract and index metadata from several different file types
    - 1.2. The application shall support jpg, png, pptx, pdf, docx, xlsx, xls, and txt files
    - 1.3. The application shall match search keywords with text fragments from the files
    - 1.4. The application shall allow users to upload supported file types
    - 1.5. The application shall provide search filters such as file type, file name, author, and date
2. User Interface and Aesthetics:
    - 2.1. The application shall have a simple UI that allows users to search files that are indexed
    - 2.2. The application shall display the results of a search query (ordered by best fit)
    - 2.3. The application shall indicate if no search results were found
    - 2.4. The application shall display the filename and path of the files that match the search criteria
    - 2.5. While the application is searching for results, the application shall indicate the results are loading
3. Resource Requirements:
    - 3.1. The application shall be easily deployable to a server(s) (potentially provided through Iowa State in the initial stages of development)
4. Environmental Requirements:
    - 4.1. All users will have the same view and permissions
    - 4.2. No authentication and authorization will be needed for this project

2.2.2 Constraints

1. The query keyword shall be up to 140 characters long
2. The application shall be a desktop or web application
3. The application shall be readable in windows larger than 500 by 600 size

4. The application shall utilize Elasticsearch to store the metadata
5. The application shall return a result within 10 seconds
6. If the application does not return a result within 10 seconds, the application shall limit the number of results returned or indicate an exception

2.3 Engineering Standards

- **IEEE-29119**: We will write Unit Tests for our application to ensure functionality in each separate piece. This would help us ensure that the individual components function correctly.
- **IEEE-12207:** We will follow the standard software lifecycle process. This shall help us in the development of our application.
- **IEEE-7.8:** Code of ethics: We will follow ethical development practices as it defines the core values of our team.

2.4 Intended Users and Uses

The primary users are Buildertrend's customers and support staff. Below are some use cases:

2.4.1 Use Case One

Users will interact with the system via a web application. The web application will display a search bar and filters that the user can use to query for files. The results would be ordered by best fit, and the user can select the desired file (based on filename and path).

2.4.2 Use Case Two

Support staff shall have additional access to reports and queries made by regular users.

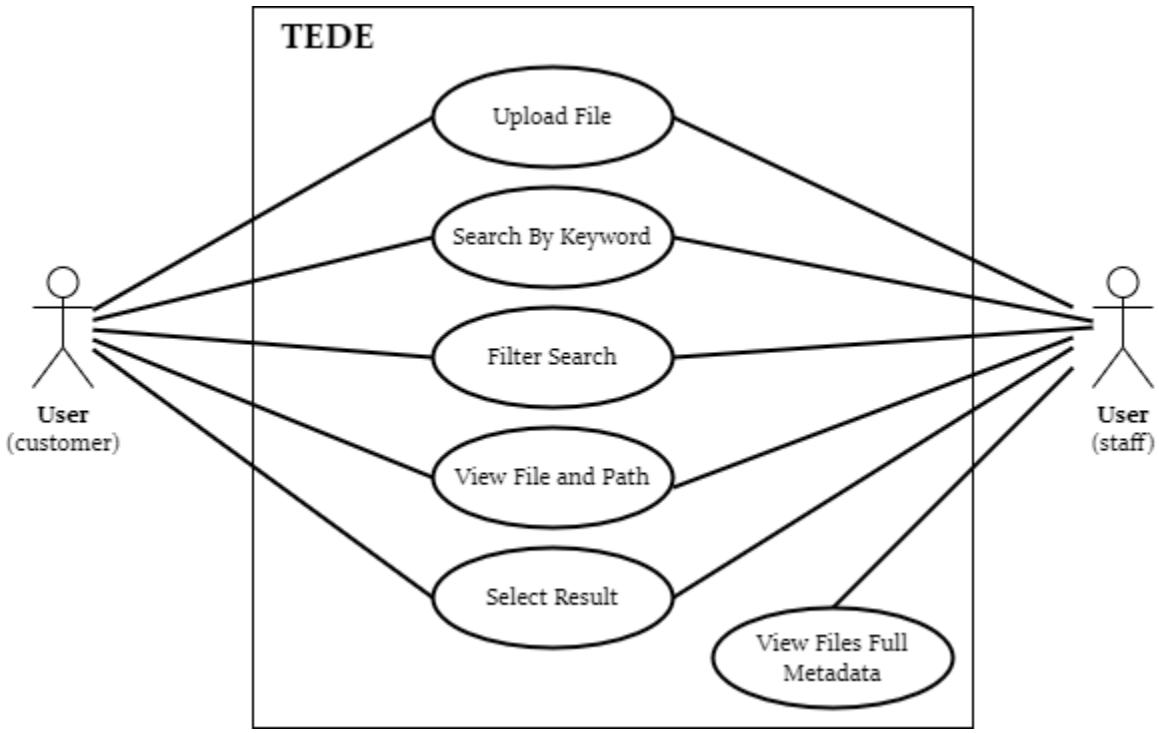

Figure 1: Use Case Diagram

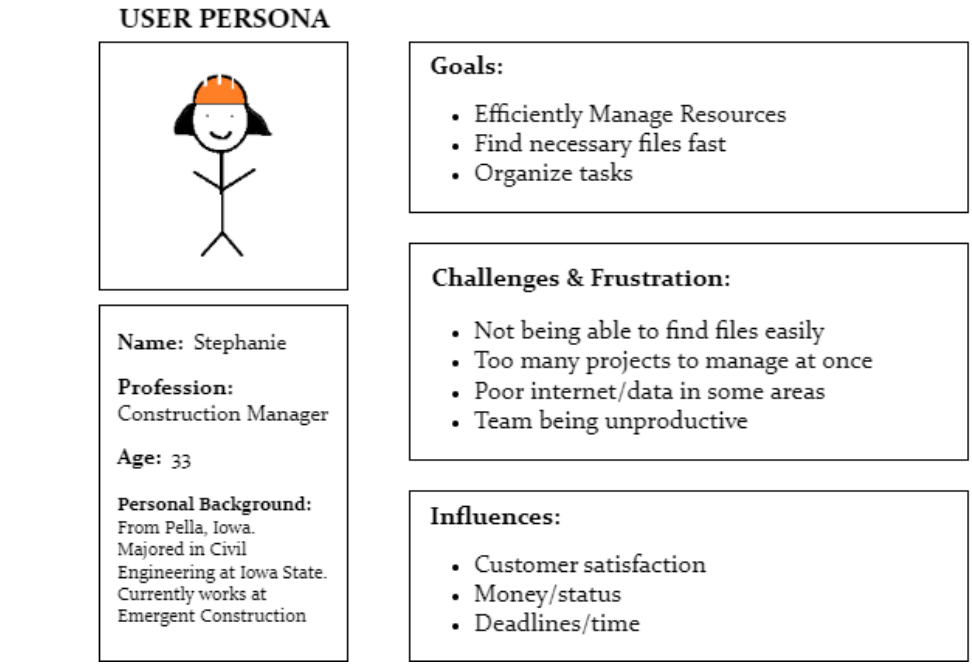

Figure 2: User Persona Diagram of Buildertrends customer base

3. Project Plan

We now discuss our project management plan and describe the breakdown of project tasks, milestones, and schedule.

3.1 Project Management/Tracking Procedures

We plan on using a combination of Waterfall & Agile management styles. The linear approach of the Waterfall process will allow us to take account of the initial requirements and help us in developing the application life cycle. The flexibility of the Agile Application Development process will allow our team to easily adapt to changes regarding features (documents support), timing constraints, new technologies, or, in an (unlikely but) extreme case, the overall scope of the project.

Our group plans on using a combination of GitLab and a shared Google Drive for this project. GitLab will be used to keep track of tasks and a repository for code. Our primary means of communication between team members will be through Discord. Microsoft Teams will be used for primary communication with BuilderTrend. Email will be used for communication with our faculty advisor.

3.2 Task Decomposition

The breakdown of the project's tasks is described in the following table.

Task	Task Description
1. Project Planning	Flush out requirements received from the client and figure out our team dynamic.
2. Develop Web Application User Interface (UI)	Create a UI for users to search using keywords and filters and see the results of related files. UI will also need to allow the uploading of files.
3. Develop REST API	Design and implement application functions that communicate between the frontend and Elasticsearch and between the text extractor and Elasticsearch.
4. Setup Elasticsearch Node	Setup Elasticsearch node to store uploaded data.
5. Develop File Text Extractor	Design and implement a way to extract text and metadata from various files.

6. Application Testing	Test individual components of the web application, text extractor, and REST API. See Section 5 for more details.

Table 1: Task Decomposition

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

We now discuss our project milestones and how we plan on measuring the success of the project.

3.3.1 Milestones

The project milestones for each task can be seen in a Gantt chart shown in Figure 3. The corresponding descriptions of the activities are provided in Table 2.

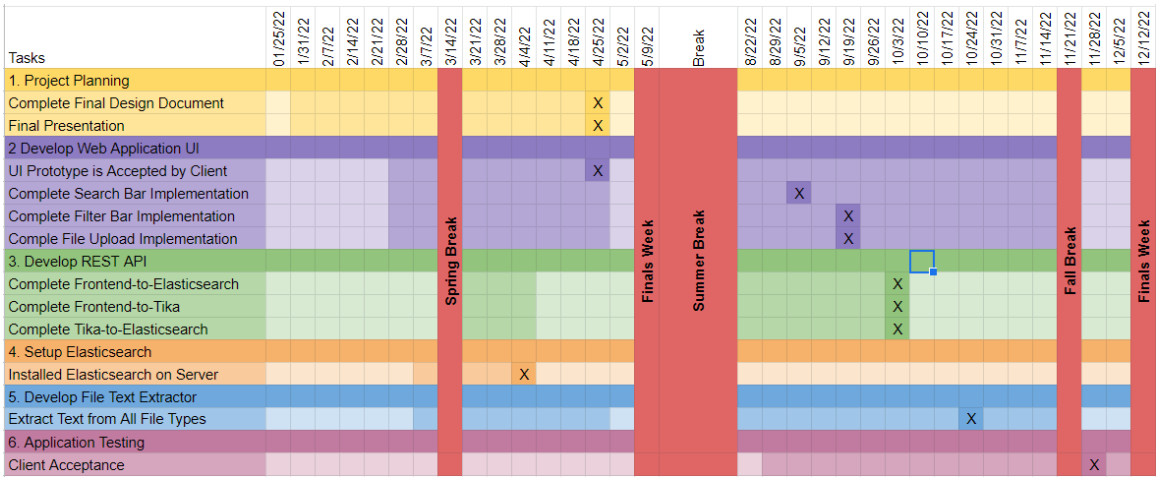

Figure 3: Gantt chart for milestones

Task	Milestone Description
1. Project Planning	The main milestones for this phase are completing this document and presenting our project. This will allow us to have a solid base when beginning implementation phases after the summer break.
2. Develop Web Application User Interface (UI)	The first milestone for this phase is to complete a UI prototype and present it for approval from our client, Buildertrend. This enables us to verify the requirements and what they are expecting from the final deliverable. The other milestones of this phase are implementation end goals for the core components of the UI: Search Bar, Filter Bar, and File

	Uploader. The milestones are concurrent due to how closely related the functionality of the components are.
3. Develop REST API	The milestone of this phase is the complete functionality of the interfaces between the core components: Web Application to Elasticsearch, Web Application to Text Extractor, and Text Extractor to Elasticsearch. More information can be found in Section 5.2 (Interface Testing).
4. Setup Elasticsearch Node	Elasicsearch is a core system of our application. The main milestone for this phase is a successful installation of an Elasticsearch node on our development server.
5. Develop File Text Extractor	The main milestone for this phase is the ability of our application to extract text and metadata from the required files (see requirement 1.2) with at least 90% accuracy.
6. Application Testing	The main milestone of this phase is to receive approval from the client. More information about our testing plan can be found in Section 5.

Table 2: Milestone Decomposition

3.4 Project Timeline/Schedule

Our project schedule is broken down as illustrated in the Gantt chart shown in Figure 4.

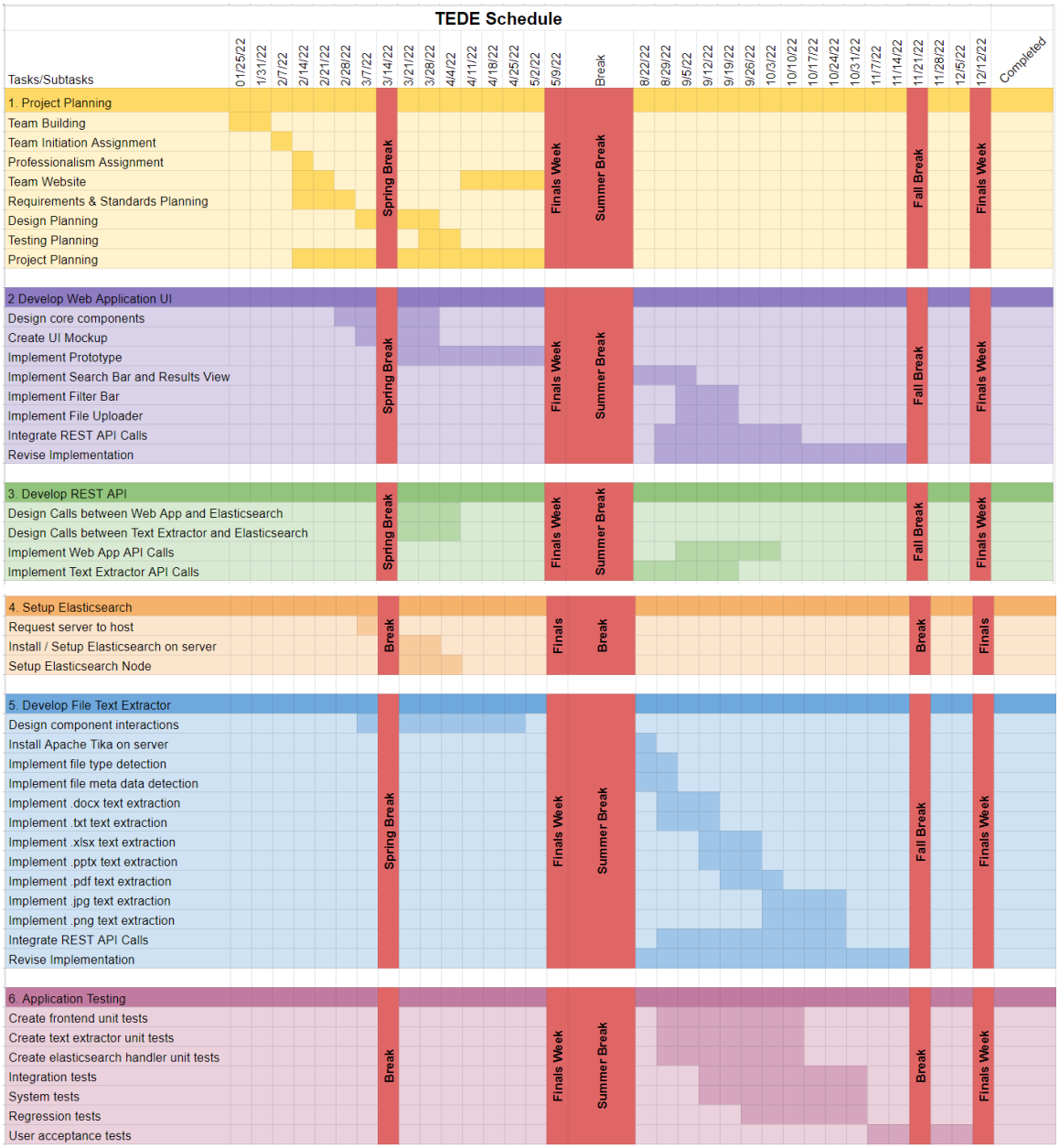

Figure 4: Gantt chart for schedule

3.6 Personnel Effort Requirements

A time estimate to complete each task is included in Table 3 and a brief description of our reasoning.

Task Name	Est. Hours	Explanation
1. Project Planning	75	Including most of the assignments for the first semester, research for learning how to utilize the APIs, and asking our client questions. The estimated time should be between 75 and 100 hours.
2. Develop Web Application User Interface (UI)	50	Our frontend requirements are not that rigorous, so it should take less time relative to our backend applications
3. Develop REST API	100	The REST API will interface with all our applications and can result in many issues transmitting messages between our applications, so the most time will be spent here
4. Setup Elasticsearch Node	75	Elasticsearch is one of the core parts of the backend, so a large amount of time will be allocated to developing and refining it for maximum performance
5. Develop File Text Extractor	75	Apache Tika is the other core part of the backend. We need to ensure we get the correct info from the files as efficiently as possible.
6. Application Testing	25	We will need to spend some time testing some edge cases and the entire system's performance. Although there should not be too many test cases depending on how many subsystems, we implement from the stretch goals

Table 3: Personal Efforts Requirements

3.7 Other Resource Requirements

There should not be any need for additional resources for our project apart from the server. Since it is a software project, we won't need any parts or materials.

4 Design

We now describe the context of our project at different scales. We also describe our design process/reasoning and present our proposed design.

4.1 Design Context

In this section, we elaborate on the context of our project at a broad scale and a user scale. We also discuss the current solution of Buildertrend's search feature and the technical complexity of our project.

4.1.1 Broader Context

Buildertrend provides construction project management software and solutions for construction companies, homeowners, and builders. Our search feature will enable them to save time and increase productivity by reducing the time spent searching through various documents.

Area	Description	Project Relevance
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or maybe indirectly affected (e.g., a solution is implemented in their communities)	Our project tries to make information inside uploaded files more accessible to the different users. It also allows users to find all the files relevant to their search and cross-reference them. The goal is to save the user's time.
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	The main contextual groups are professionals in the construction industry. This project will enable them to have much greater search capabilities in terms of not only technical specifications but also contextual bindings, user requests, etc. Our project does not conflict with any ethnic or racial group and is transparent

		to different groups. However, the product is for now limited to the construction industry.
Environmental	Direct and indirect environmental impacts such as deforestation or unsustainable practices related to materials manufacture or procurement.	By having the capability and access to different search filters, construction workers can gain environmental awareness (e.g., municipality constraints on forestry practices near construction sites). Should future software versions scale beyond a threshold, resources such as energy consumption may be a concern.
Economic	Possible financial viability of your product within the team, company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	The reduced search time will allow Buildertrend customers to save on labor/billed time costs and work on more projects. Giving them an edge over the competition and attracting more customers for Buildertrend increasing sales and profits. In addition, other constraints with financial impacts may be searchable.

Table 4: Responsibility Considerations

4.1.2 User Needs

Our application has three different user groups; their needs are defined below.

Builders & Contractors

Builders and workers need a quick and efficient method to search for information on site, such as measurements, designs, orders, etc., to complete the project according to requirements.

Homeowners

Homeowners need a way to search for documents regarding the construction of their home at any time. The documents could include important information such as previous

contracts, costs, or design parts/specifications as necessary in order to make a decision regarding the project.

Buildertrend Staff

Buildertrend staff need a way to support the users on any potential issues they face with the search feature. They will have access to the full entries saved in Elasticsearch to understand why certain results were returned for a query.

4.1.3 Prior Work/Solutions

Buildertrend has a current implementation of Elasticsearch that searches files only by file name. There is also an implementation that allows staff members to search for files based on their SQL representation.

4.1.4 Technical Complexity

The design consists of several subsystems of various complexity, each requiring background knowledge and/or additional research. These subsystems include:

- **Searching algorithm:** Implement a search algorithm to find a return the best-fit result within 10-seconds.
    - *Requirements:* Algorithm analysis and implementation, CI/CD
    - How can we maximize query result precision and recall?
- **Simple frontend UI/UX:** User-friendly web app with the ability to navigate, search, and upload files
    - *Requirements:* Web development, React, CI/CD
- **Backend architecture:** Use decided tech stack to implement a system to extract, and index provided documents to be served to the frontend UI when queried.
    - *Requirements:* Database management, Tika, Elasticsearch, Investigate the impact of "mediator" that translates the text from different file types
- **Test cases:** Design proper testing suite to check for project requirements as well as optimal functionality
    - *Requirements:* Experience with previous testing libraries(e.g., JUnit)
- **Project management:** We must combine all components of the project which requires the frontend, backend, and other components to be properly implemented and tested to enable them to communicate seamlessly and efficiently.

4.3 Proposed Design

We now discuss the project design using several diagrams and describe how it meets the requirements.

4.3.1 Design Visual and Description

The following diagram, Figure 5, shows a high-level overview of the interactions of our application with relevant external systems. The following are brief explanations of the numbered interactions:

1. Receives a list of files relevant to the query made and displays them to the user
2. Triggers a query using a keyword filter
3. Add other filters to a query
4. Upload a file to have the text content extracted
5. Store a copy of the upload file on server
6. Returns a list of files from Elasticsearch that match the query filters
7. Query the data stored in Elasticsearch that matches the given filters
8. Stores the metadata and content of the uploaded file into Elasticsearch

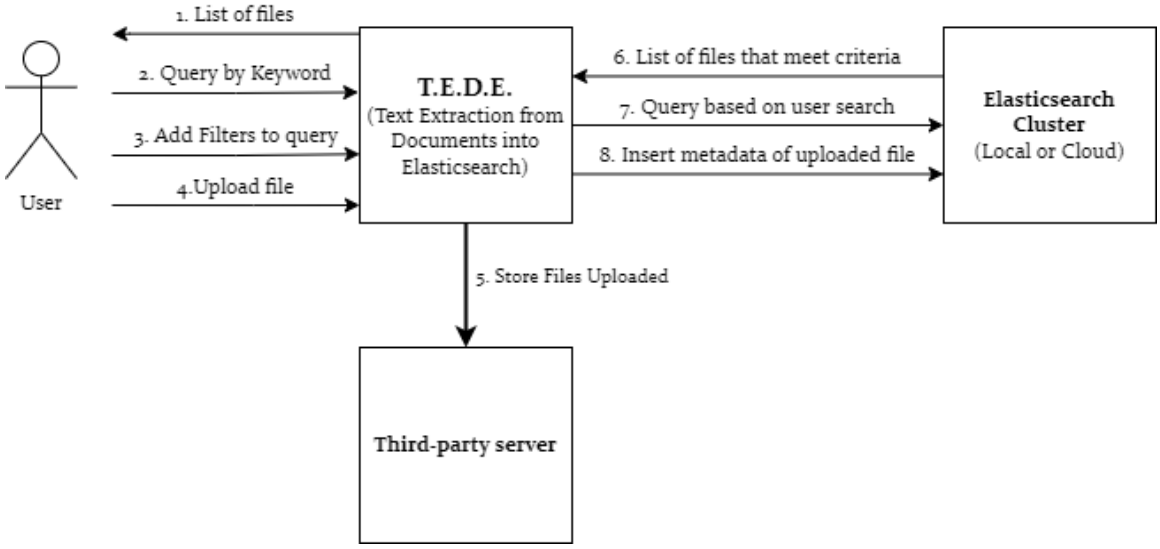

Figure 5: TEDE Context Diagram

The following diagram, Figure 6, shows a breakdown of the components of TEDE. The components are responsible for the following responsibilities:

- **Web Application SubSystem:**
  - *Search & Filter Component:* Handles the applying keyword and other filters from the user.
  - *Request Handler:* Recieves the applied search parameters and formats a request to the Elastic Query Handler.

    - *Result View Component:* Formats and displays the results received from a query request to the Elastic Query Handler. Each Result is formatted into a *Result Card*, showing the file name, file path, and matched filters.
    - *Upload Component:* Initializes an upload of a file to store the file on the server.
- ***Elastic Query Handler:*** Formats an elasticsearch query from the Web Applications search query and retrieves data from the Elasticsearch cluster. Sends retrieved data back to the Web Application.
- ***File Upload Handler:*** Stores the received file on the server and triggers a text extraction/ index of the file's contents.
- ***Text Extractor****:* Uses Apache Tika to extract metadata and text content from the given file and indexes and stores the data in Elasticsearch cluster
- ***Elasticsearch Cluster:*** Local instance of the search engine running in a docker container on the ETG Server.

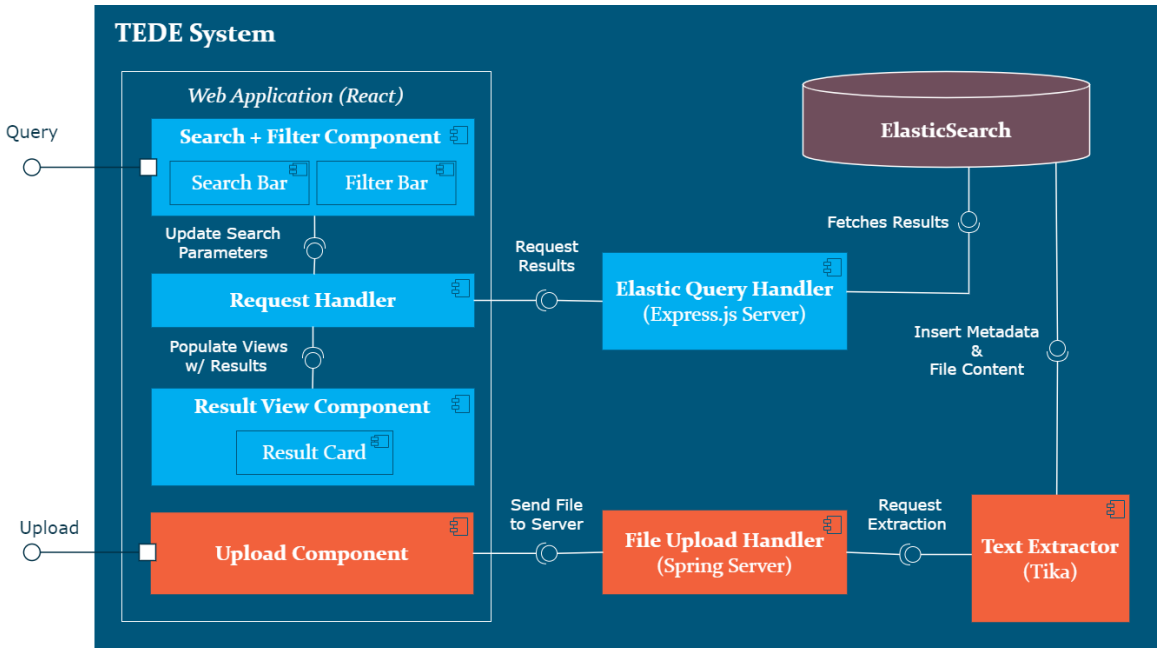

Figure 6: TEDE Component Diagram

4.3.2 Functionality

Users of our application will be able to search using keywords and search filters to find relevant documents. It will also be able to extract text from uploaded files and store the important information from the file. The application would support various file types like txt, docx, pdf, pptx, etc.

Our current design addresses these function needs for the specified file type but may need to be adjusted based on what is attainable. For the UI/UX requirements, we might need further client approval.

4.3.3 Areas of Concern and Development

We have the following concerns about our current design and our plan of handling them.

Uploading docs

Concern: No clear specification of what this feature entails.

Plan: The plan is to be able to upload document types that we can search. As we scale our search document types, we also diversify our upload document types.

UI/UX

Concern: No detailed specifications for this one either. We will have to reiterate our design until we get approval.

Plan: Propose a UI design, receive feedback, and reiterate until final approval.

Accuracy of results

Concern: No qualitative measures were specified by the client.

Plan: For now, we will just follow the traditional measure - if the result fits the user's need or what the user was actually looking for.

Query speed

Concern: The client has not specified anything yet. Nonetheless, we want to return our results within 10s. Anything longer than this, we will have to rework and optimize our algorithm.

Plan: Develop a search algorithm and optimize it until an acceptable speed is attained.

Scalability of doc types

Concern: The list of document types is quite open-ended. Less of concern and more of a development.

Plan: We would be starting with txt and docx files. As we find success, we will progress to other types like pdf, ppt, jpeg, etc.

Overall, most of our concerns can be resolved with better client specifications.

4.5 Design Analysis

Prior to initial implementation, we were given the opportunity to explore and experiment with various tools and technologies to complete the project.These experiments allowed us familiarize ourselves with the tools, and have a system designed to best fit the needs of Buildertrend. Following the selection of the tech stack, we had several iterations of system design (diagrams, sketches, etc) and received feedback from the client. Based on the previous experiments conducted on various components of the system and the positive feedback received from the client, we felt confident and moved forward with the implementation of the design.

4.6 Design Plan

Our design process will follow our current schedule o utlined in Section 3.4. We will be developing most of the components concurrently and getting them approved during our weekly meetings with the client. We allocated some of the time in the implementation sections to making changes and refining the components as we develop them based on the feedback we receive. We will adjust and update our schedule accordingly based on the progress that we make throughout the semester as well. We plan to finish the overall implementation before fall break. This will give us a couple of weeks to make adjustments.

5 Testing

Some of the unique testing challenges in our project include the complexity of testing communications between the frontend and backend. As well as testing the accuracy of our text extractor module. Below is our testing plan to handle these challenges.

5.1 Unit Testing

As shown in our design diagrams in section 3, there are several distinct modules in our system. Each of these modules has identifiable units of work that are planned to be a part of the final deliverable.

Below is a list of the identifiable elements:

1. **Web Application**
    a. All components and functionality displayed to the user such as:
        i. Search bar queries and filters
        ii. Proper presentation of results
        iii. Uploading of new documents
2. **Text Extractor**
    a. Extraction of metadata from newly uploaded files. It should be able to handle all of the file types specified in Requirement 1.2.

    b. Be able to identify file types and determine what pieces of metadata to send to Elasticsearch. (e.g., text body for pdf, resolution for jpg, etc.)
3. **Elasticsearch Query Handler and Text Extractor**
    a. Proper indexing of the metadata from Tika Text Extractor
    b. Building queries to search the Elasticsearch cluster based on inputs received from the web application

Below is a list of tools that will help us test the above elements:

1. **Web Application**
    a. We plan on using Jest and React Test Library packages for unit testing. Since they are React's recommended testing tools, they will integrate well into the project. Jest will allow us to see the code coverage of our tests and will allow us to mock data inputs. React Testing Library provides virtual DOMs for testing React components without a browser.
2. **Text Extractor**
    a. We plan on using JUnit for unit testing. Since the primary package we are using to support this component is a Java package, JUnit gives us the ability to easily test our code. JUnit is also well documented and will integrate into our development environment easily.
3. **Elasticsearch Query Handler and Text Extractor**
    a. Since the primary purpose of this component is to handle communication between components. See 5.2 Interface Testing for more information.

5.2 Interface Testing

There are several identifiable levels of interface testing:

1. **Web Application to Elastic Query Handler**
    a. Is responsible for fetching and retrieving query data
2. **Web Application to Text Extractor**
    a. Is responsible for sending a selected file to be extracted by the Tika Text Extractor component
3. **Text Extractor to File Upload Handler**
    a. Is responsible for inserting a file's metadata and text content into Elasticsearch

Elasticsearch provides a REST API for inserting data. We plan on using Postman to validate data insertions into Elasticsearch are successful. To test our components' interaction with Elasticsearch, interfaces 1 and 3, we plan on using Jest and JUnit, respectively.

5.3 Integration Testing

Integration testing will be critical for our project as we will have several components communicating with each other through various endpoints. Incremental integration testing will be the approach for this project, allowing us to test individual modules prior to integrations. It will be important to test and validate the functionality between the Elasticsearch endpoint and the tika application as well as the integration between the frontend and Elasticsearch; this will be the core of the application. Additionally, if the group reaches the stretch goal, integration of the application into a cloud-based environment would require additional integration testing as there would be additional components to take into consideration during development (VPC, Cloud DB, etc.). However, our project will primarily be focused on the local implementation prior to integrating with the cloud. As this project serves as a proof-of-concept, we anticipate that Buildertrend will also be integrating this product into their own application, implying additional testing during this stage.

5.4 System Testing

Considering that our overall system is composed of three major components with minimal communication among them, our system testing should be fulfilled by our unit and integration tests. The individual components should be tested via the unit tests to ensure that they are functional before testing the communication among them. Following the unit tests, there are three interprocess communications that are specified in the interface testing section that need to be tested. These communications will be tested using the methods specified in the integration testing section which will complete our system testing.

5.5 Regression Testing

It is important that new additions made to the system should not break it and instead enhance the user experience. Our most essential features are web application, Elasticsearch handler, and tika text extractor, and we need to make sure they are robust enough. We would do enough unit and integration testing on our critical components and then add a new feature and see if we still get the same results. This way, we can have our core system to be robust and can easily figure out what new feature is breaking the old functionality.

5.6 Acceptance Testing

Requirements agreed upon by the client and the team should be fully visible and usable in our minimal viable product - the search web application. We plan on having several iterations of acceptance testing as we expect to receive feedback from the client while making progress toward the final deliverable. The final iteration will include

documentation in order to aid users and developers, as well as the working application based on the feedback received, showcasing the requirements and features.

5.7 Security Testing

Security testing is now more important than ever, however, it will not be applicable to our project as it's just a proof of concept. We expect Buildertrend to work on the security when they integrate it into their system.

5.8 Results

We followed a test driven development approach when implementing our project. The large majority of our tests were unit tests covering the frontend as well as the Apache Tika extraction.

For the frontend, the UI tests primary focused on how the subcomponents passed data to each other. These tests allowed us to verify changes did not break other components directly or indirectly. Each subcomponent in the web application was tested with a range of possible inputs. The tests covering the interaction between the web application's Request Handler & Elastic Query Handler were useful in capturing the requests we initially would run in Postman, and helped us keep track of the expected query formats.

For the backend, the tests primarily focused on ensuring Apache Tika could parse all of the required file types. It also helped streamline the development process since new test cases could be implemented easily to test new features. We used parameterized tests to cover over 25 files of varying types. This helped us ensure that the end product is able to handle many different files of different sizes, contents, etc.

Finally, we primarily used Postman to handle the communication aspects of our project. This helped us test the frontend sending files to Tika and the frontend displaying various files that may be sent by Elasticsearch.

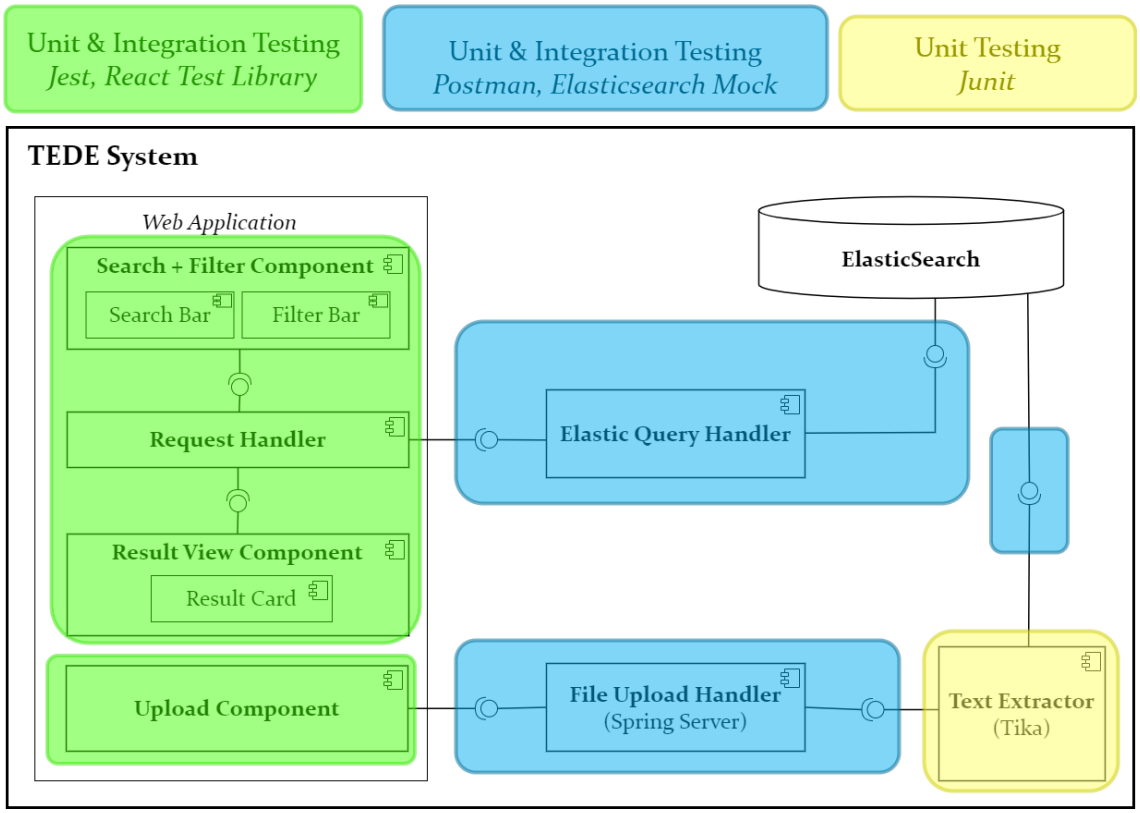

Figure 7: Amended Component Diagram with the testing strategy

6 Implementation

Our implementation followed closely to our initial design plan. It consisted of 5 main components: Web Application, Elastic Query Handler, File Upload Handler, Text Extractor and an Elasticsearch instance. For this project, implementation is inseparable from the design activities. See the design section for more details. We now describe the final implementation. Appendix 1 contains details regarding setup options and selection.

6.1 Web Application

The web application consists of 3 main sub-components: Search Bar, Filter Bar, and File Upload Window. The following figures capture the functionality of these components. Figure 8 captures starting page, which include the Search Bar component as well as access buttons to the Filter Bar and Upload Window.

Figure 9 shows an expanded view of the Filter Bar. It contains an area showing what filters are affecting the query, as well as buttons to add / removed file types to the query. It also includes several text based filters as seen below that can be added and removed from the query.

Figure 10 shows the Result View containing Result Cards for each of the results for the query. Each result card shows the file name, an icon for the file type, file path, and filters that were a match.

If a error occurred during the query, the user will see a screen captured in Figure 11.

Figure 12 shows the use of the File Uploader window. It allows the user to choose multiple files to upload to the server to be processed into Elasticsearch. The uploader will notify to the user if it was successful or if a problem occurred, as seen in Figure 13.

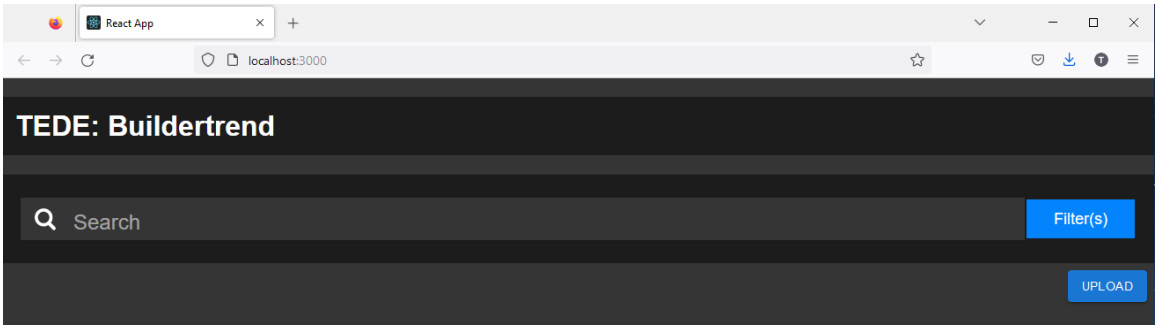

Figure 8: Starting page of the web application

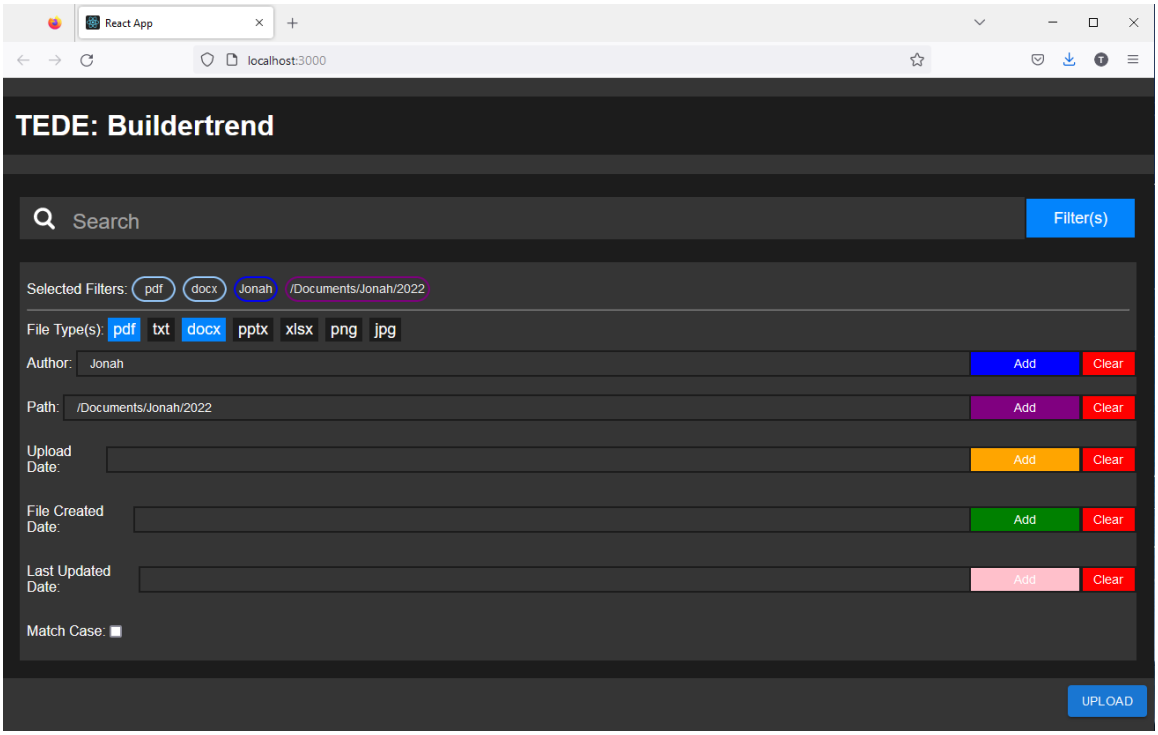

Figure 9: Expanded filter bar with file type, author and path filters selected

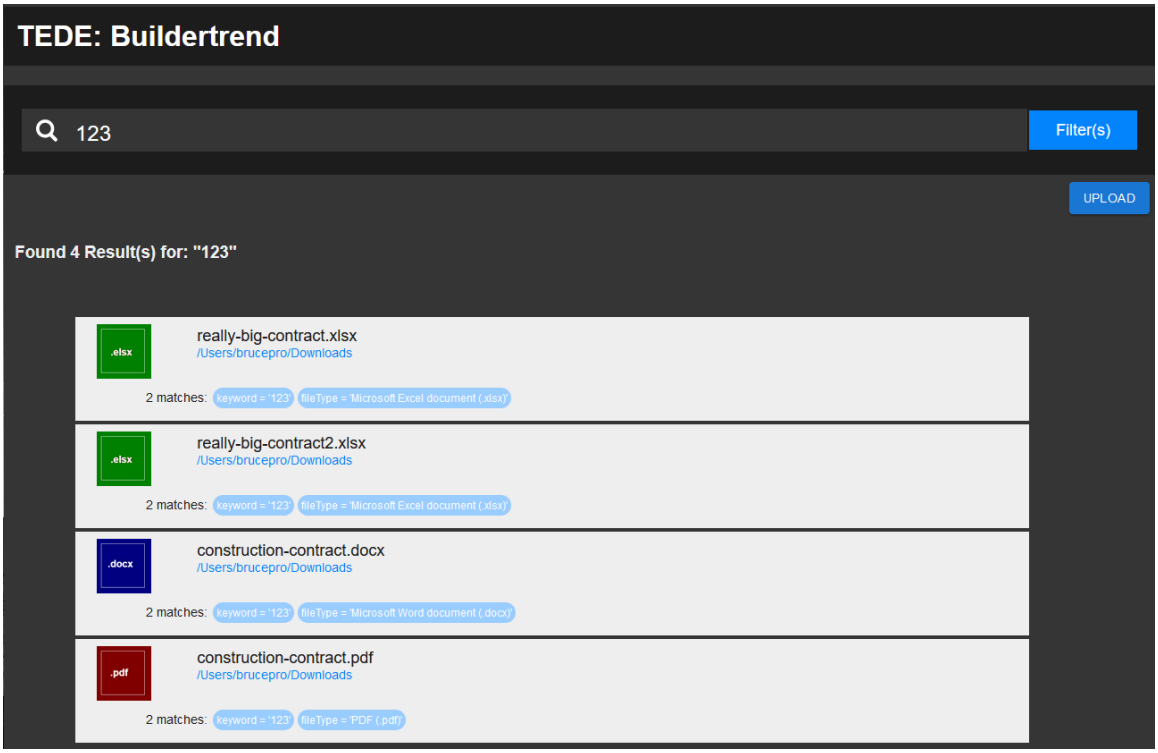

Figure 10: Results View showing 4 Result Cards of different file types

Figure 11: Connection error occurred during the query

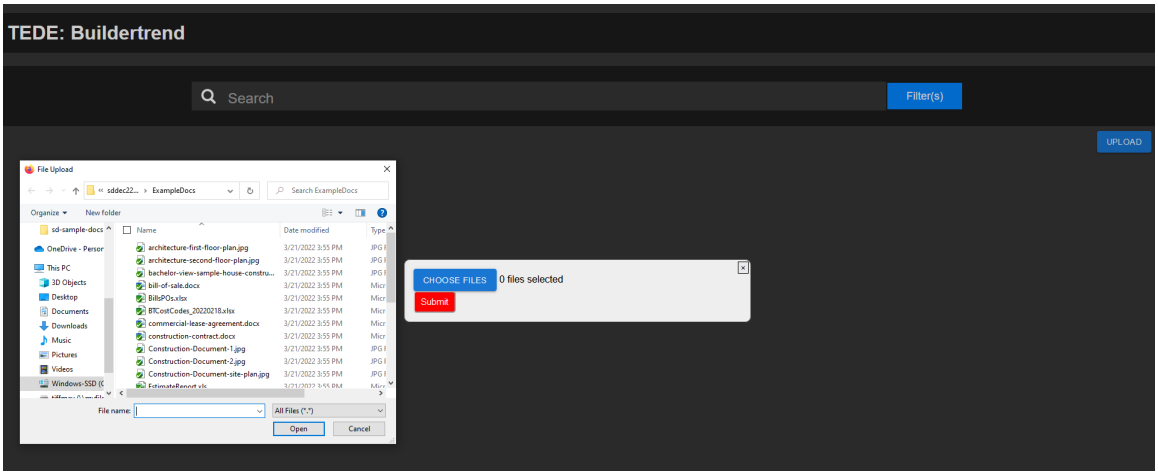

Figure 12: File Upload window makes use of the os native file selector to choose files

Figure 13: Shows the File Uploader's possible response messages.

6.2 Elastic Query Handler

This component's responsible of formatting the POST request received from the Web Application into an Elasticsearch query and format Elasticsearch's response in the format the Web Application is expecting. Figure 14 shows the format of a POST request and response from and to the web application. Figure 15 shows the format of a POST request and response from and to the Elastic Query Handler to the Elasticsearch Instance. The actual component is implemented using the Express.js framework and Elasticsearch's Javascript API.

Post Request

```
{
    "contents": "*contract*",
    "types": [
        "pdf",
        "txt"
    ],
    "authors": [
        "brucepro"
    ],
    "uploadDates": [],
    "fileCreatedDate": [],
    "lastUpdatedDate": [],
    "containingFolder": []
}
```

Post Response

```
{
    "matches": [
        {
            "id": "1",
            "fileName": "really-big-contract2.txt",
            "filePath": "/Users/brucepro/Downloads",
            "author": "brucepro",
            "fileType": "Text (.txt)",
            "contents": [
                "This is another contract for XXX company."
            ]
        },
        {
            "id": "2",
            "fileName": "another_sort_of_file.pdf",
            "filePath": "/Documents",
            "author": "brucepro",
            "fileType": "PDF (.pdf)",
            "contents": [
                "THE FILE_CONTENT. contract."
            ]
        }
    ]
}
```

Figure 14: POST format from and to the web application to the Elastic Query Handler

Elasticsearch Query

```
{
    index: 'documents',
    query: {
        bool: {
            "must": [
                { "query_string": {
                    "default_field": "contents",
                    "query": "*contract*"
                }}

            ],
            "filter": [
                { "bool": {"should": [
                    {"match": { "type": "PDF (.pdf)" }},
                    {"match": { "type": "Text (.txt)" }}
                ]}},
                { "bool": {"should": [
                    { "match_phrase": { "owner": "brucepro" } }
                ]}}
            ]
        }
    }
}
```

Elasticsearch Response

```
{
    query: params.body.query,
    hits: { hits:[
        {
            _id: "1",
            "_source" : {
                "type" : "Text (.txt)",
                "title" : "really-big-contract2.txt",
                "path" : "/Users/brucepro/Downloads",
                "owner" : "brucepro",
                "contents" : "This is another contract for XXX company."
            }
        },
        {
            _id: "2",
            "_source" : {
                "type" : "PDF (.pdf)",
                "title" : "another_sort_of_file.pdf",
                "path" : "/Documents",
                "owner" : "brucepro",
                "contents" : "THE FILE_CONTENT. contract."
            }
        }
    ]
    }
}
```

Figure 15: POST format from and to the Elastic Query Handler to Elasticsearch Instance

6.3 File Upload Handler

This component handles the files that are uploaded to the server by the user–a simple spring server is used for this purpose. The user can select one or multiple files and upload together. Once submitted by the user, the files are extracted via the Tika extractor and the extracted data is put into the elasticsearch instance. The files are also written to the "/uploads" directory on the server. One major feature that we wanted to implement with this handler was to upload the local file path and other attributes as well. However, this wasn't possible due to the browsers not being able to access the user's disk due to security concerns.

6.4 Text Extractor

This component extracts all of the relevant information from files uploaded to the server. It extracts data like title, content, path, etc. from all files and extracts information specific to certain file types such as page count on word docs. It also uses Tesseract to attempt to read text from images and convert it to text in the JSON file it creates. It then opens a connection to our Elasticsearch instance using the Elasticsearch API and sends the JSON file containing the file's metadata to it to be indexed.

6.5 Elasticsearch Instance

We were provided a server from ETG to be used for this project; this server was used to host our instance of Elasticsearch. We went with a default configuration as it was to be used primarily as a proof-of-concept; nonetheless, it can be easily customised for specific security or availabilty needs. Once installed, we were able to access the instance via port 5601 and login using the credentials provided during setup. In order to run queries and vsualize the data, we made use of the Kibana console in the Development Tool section of the Elasticsearch landing page. The initial step was to create an index to hold our data. Once the index was successfully created, we made a mapping according to the fields to be extracted from the documents (Author, Title, etc.).

8 Closing Material

After describing our project in detail, we now highlight the main results of our project. We also reiterate the work done during the implementation phase of our project.

8.1 Discussion

We have designed our solution to meet our client's requirements. We have included support for the request file types for extracting text. As requested, we have also included a user interface that allows the user to query by keywords and filters.

8.2 Conclusion

We started by researching potential packages to help us extract text from files and develop a simple user interface. Then based on that, we decided on our technology stack. After this, we continued to learn more about how to use the technologies we used.

To reiterate, our goal is to design a prototype application that uploads the text content and metadata of supported file types into Elasticsearch and develop a simple user interface to query and filter uploaded content. Our application will serve as a proof of concept as part of Buildertrend "Global Search" initiative.

Though the course of the semester our group has met biweekly with out contracts at Buildertrend to receive feedback and approval of the functionality of our project. This iterative approach was chosen during the planning phase of the project. It came more clear that they were interested in the limits, if any, discovered with Elasticsearch and Apache Tika package. So in addition to the code deliverables we will be providing Buildertrend a report of these findings.

Appendix I: Operation Manual

Instructions on how to run our application given access to the code repository.

Frontend Setup

1. Using git clone the code repository.
2. In the cloned repository, go to the Frontend directory. This is separated into two sections: elastic-handler & tede.
    a. Both packages use npm for managing dependencies, to install see npm's install instruction page.

"elastic-handler" Setup

This component is required to be running in order for tede (the User Interface component) to make requests to the elasticsearch instance on the server.

1. In the elastic-handler directory, run *npm install* to install the required dependencies.
2. To start the handler, run either *npm start* or *npm run dev*. The later will automatically restart when your code changes. These commands will run the server on http://localhost:8626/

"tede" Setup

Note: this component is dependent on being connected to elastic-handler component (see above setup instructions) and being connected to Iowa States network (or vpn) in order to connect to Elasticsearch.

1. In the tede directory, run *npm install* to install the required dependencies.
2. To start the handler, run *npm start*. This will start the web application on *http://localhost:3000/*.

Frontend Tests

"elastic-handler" Tests

In the elastic-handler directory, run *npm server-tests*.

Note: This command will start automatically start the Elastic Query Handler server and run a set of tests that mimic the tests we would run manually in Postman. When the tests, finish the server instance it started will also be shut down. Example output can be seen below in Figure 16.

```
PS C:\projects\sddec22-19\Frontend\elastic-handler> npm run server-tests

> elastic-handler@1.0.0 server-tests C:\projects\sddec22-19\Frontend\elastic-handler
> start-server-and-test start-server http://localhost:8626/api/v1/debug/ test

1: starting server using command "npm run start-server"
and when url "[ 'http://localhost:8626/api/v1/debug/' ]" is responding with HTTP status code 200
running tests using command "npm run test"


> elastic-handler@1.0.0 start-server C:\projects\sddec22-19\Frontend\elastic-handler
> node index.js true

The server is listening on port 8626

> elastic-handler@1.0.0 test C:\projects\sddec22-19\Frontend\elastic-handler
> jest test/query.test.js

 PASS  test/query.test.js
  √ Query 1: keyword + file filter (only pdfs) (1553 ms)
  √ Query 2: keyword only
  √ Query 3: keyword + file filter (.txt and .pdf) + author filter
  √ Query 4: keyword + path filter

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.982 s, estimated 6 s
Ran all test suites matching /test\\query.test.js/i.
PS C:\projects\sddec22-19\Frontend\elastic-handler>
```

Figure 16: Example output the user may see when running the elastic-handler tests

"tede" Tests:

In the tede directory, run *npm test*.

Note: While running the user will be presented with options of how to choose what tests to run, see Figure 17 below for example.

```
No tests found related to files changed since last commit.
Press `a` to run all tests, or run Jest with `--watchAll`.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

Figure 17: Options shown when tede tests are ran

Backend Setup

"Elasticsearch" Setup

Elasticsearch offers 3 self-managed options: Running it on any local machine, running it in a Docker container, or running it on Kubernetes with Elastic Cloud on Kubernetes. We opted for the Docker container option as it provides fast deployment with ease of creating and deleting instances. We used Elasticsearch 8.2.3 and Kibana 8.2.3 docker images for our ELK Stack with security enabled and configured by default. Once completed, the instance can be accessed via port 5601.

"Spring/Tika Server" Setup

The Spring server containing the Apache Tika code is bundled as a JAR file. When executed, it creates a spring server using port 8080 which listens for POST requests at the directory "/uploads". For example, if the server is run locally, requests should be sent to "http://localhost:8080/uploads". This is the entry point for the backend, so files sent to this server will be parsed and indexed by Elasticsearch which will then later be displayed on the frontend.

Backend Tests

"Elasticsearch" Tests

Elasticsearch functionality was tested by making calls to the REST API using Postman and the UI to verify connectivity.

"Spring/Tika Server" Tests:

Automated unit tests for this system are included in the source code. They use files provided by the ExampleDocs directory.

Appendix II: Alternative/Initial Designs & Evolution

Frontend

Initial Mockups

As a part of the project planning phase, we implemented a simple prototype of the user interface. Screenshots of our client-approved mockup can be found in Figure 18.1, Figure 18.2, and Figure 18.3 For this project, implementation is inseparable from the design activities and the mockups were used solely for the purpose of showing the client our design plan.

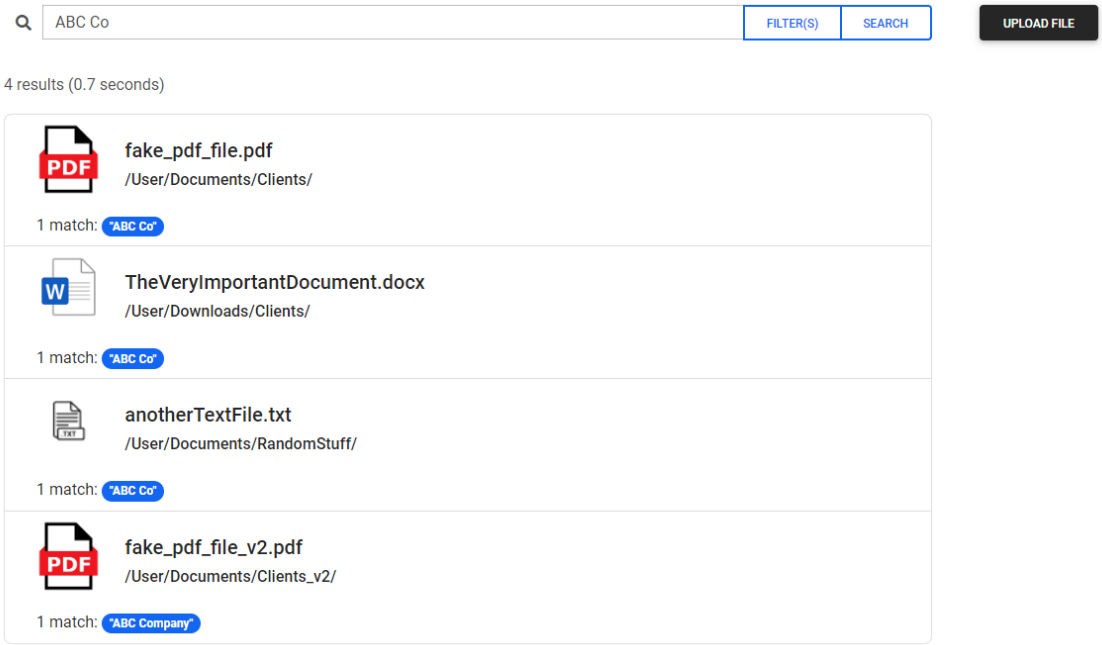

Figure 18.1: Mock-Up of the Results View of the User Interface.

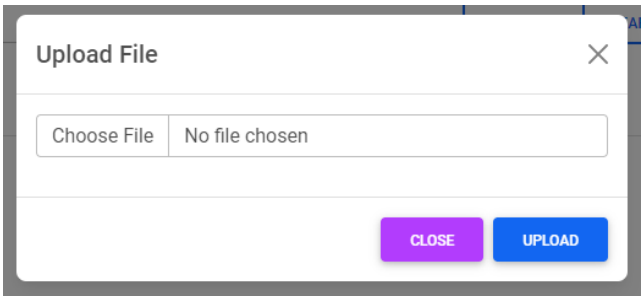

Figure 18.2: Mock-Up of the Upload File View of the User Interface.

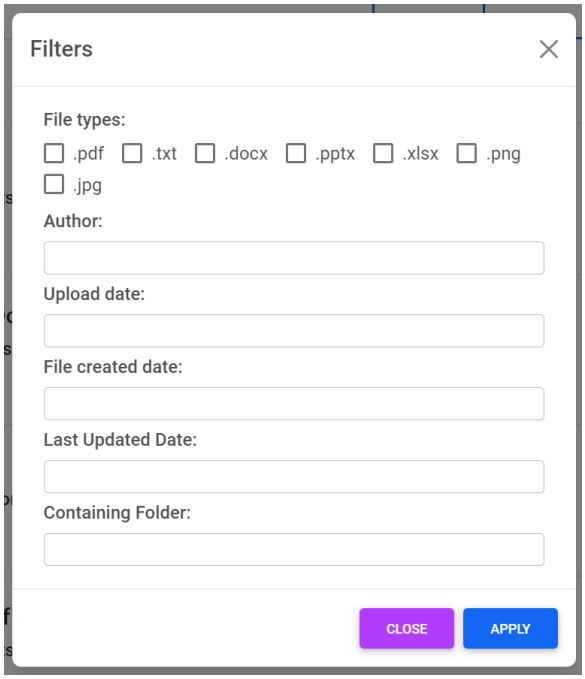

Figure 18.3: Mock-Up of the Filters View of the User Interface.

Backend

One of the major changes to our backend design that we tackled during the implementation was how we would design our metadata extraction as an event driven system. We initially thought that Apache Tika could be run as its own server system, however, this turned out to not be the case which made us allocate a portion of development time to researching and implementing another server system for Tika to extract the metadata. We ultimately landed on creating a Spring server which would allow the frontend to make post requests when the user uploaded a file and when the server received them, it would call the Apache Tika code to extract the metadata, convert it to a json file, and send it to Elasticsearch.

Component Diagram from 491 Design Document

The diagram described in the initial design document had a couple flaws.

1. It grouped components that have separate responsibilities together. For example, The “Search & FIlter Bar” component should separate those out into another level. The two components in the “Elasticsearch” component are another example.

2. It does not capture the Results View component that is responsible for showing the responses to queries made.
3. It is confusing what work we are doing in the “Elasticsearch” and “Apache Tika” components. The naming convention and grouping of components made it unclear where the work we were planning on / responsible for separated from the Elasticsearch tool and Apache Tika library.

The following diagram, Figure 19, shows a breakdown of the components of TEDE as described in the initial design document. The components are responsible for the following responsibilities:

- **Web Application System:**
  - *Search & Filter Bar:* Handles the applying keyword and other filters from the user
  - *Upload Cache:* Initializes an upload of a file to store the file on the server and triggers a text extraction/ index of the file's contents
- **Apache Tika System:**
  - *Tika Text Extractor:* Extracts the metadata and text content from the given file and calls the Tika Handler to index/store the data
- **Elasticsearch System:**
  - *Web-App Handler*: Creates a query call from user input and retrieves data from the Elasticsearch cluster
  - *Tika Handler:* Inserts data into the Elasticsearch cluster

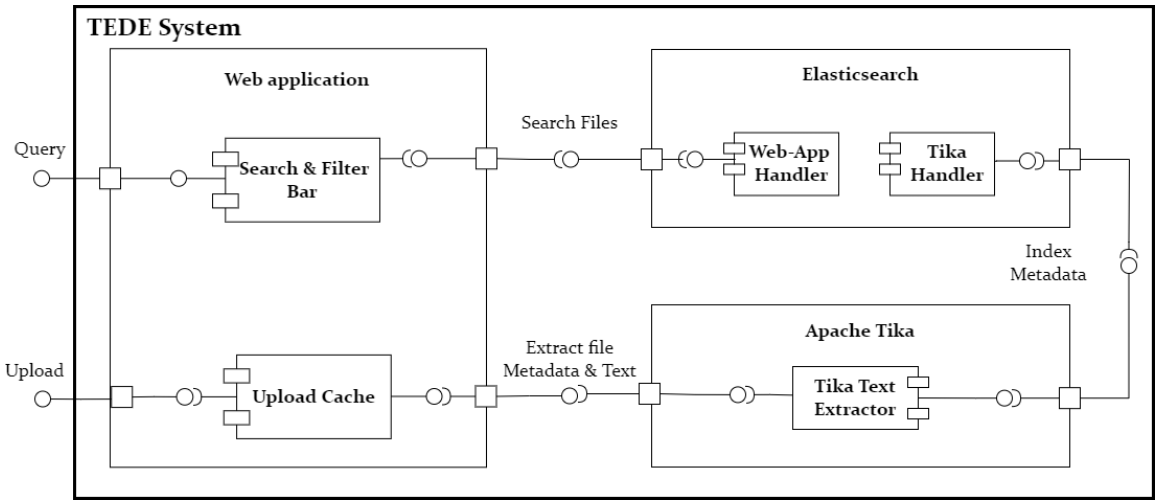

Figure 19: Original TEDE Component Diagram

Appendix III: Code

Since an NDA was signed, the team can only share code that has been approved by Buildertrend. No code has been approved.