Team Outlaws
Technological Feasibility Analysis

Project Sponsor and Mentor

Dr. Eck Doerry

Team Members

Quinn Melssen

Liam Scholl

Max Mosier

Dakota Battle

November 5, 2021

Table of Contents

1 Introduction

As Agile programming practices continue to take the tech industry by storm, the importance of small teams in real world engineering workplaces is increasing tenfold. According to Goremotely.net, over 71% of tech companies either already use, or are in the process of adopting agile methods, which are based off of these teams. Seeing the prevalence of small team workgroups in the professional world begs the question, why are more engineering classes in higher education not team-based as well? A main reason for this is the difficulty for faculty to manage and maintain the teams involved in such an undertaking. Handling hundreds of students is a daunting task to begin with, but managing the relationships and expectations between them as they form into groups can be impossible. There are three primary phases of this process, each involving large amounts of hands-on effort:

- **Gathering projects** - Involves reaching out to dozens of potential clients, exchanging hundreds of emails, and keeping track of the varying stages of each potential project.
- **Forming teams** - Involves taking in priorities, GPAs, and other information about each student by manually inputting all relevant information into an algorithm.
- **Executing class** - Involves using several different mediums to run a program, including email, websites, and verbal/written communication.

Every year, our client, Dr. Eck Doerry must painstakingly gather and communicate with enough clients to provide projects for the year. This process consists of hours of back-and-forth emails between many different potential clients. Once the projects have been gathered and finalized after dozens of drafts, students are expected to fill out preference documents that are ultimately used to assign them to their respective projects. This process too, requires a high amount of hands on effort that could be streamlined by a successful technology. Once the projects have a team, Dr. Doerry will then be responsible for running the capstone course and managing the collection of various assignments. While this process ultimately leads to a successful capstone, Dr. Doerry has been brought face to face with its inefficiency as he has attempted to split leadership of the course with another NAU computer science professor, Dr. Michael Leverington.

During this transitional process, Dr. Doerry has realized that his current solution could be streamlined both for himself and future instructors. This is where TeamBandit will come in. The intent of TeamBandit is to act as a web application that will allow Dr. Doerry and potentially any team manager to organize, collect information, and manage tasks in a centralized location. Features of this web application will include:

- A client acquisition and communication module in order to facilitate the process of finalizing project propositions.
- An assignment module that allows students to select project preferences, as well as sort them into groups using a sorting algorithm.
- A localized hub for work or deliverable submissions for both students and teachers.

At this point in our process we have been working closely with our client to gather requirements and find technologies capable of fulfilling them. This is a key design step as the rest of our project will ultimately be built upon these foundational pieces of technology. In this document we will present each of these major challenges as well as the design decisions that we have designated as solutions for them.

2 Technological Challenges

In order for the TeamBandit system to be successful, we sought out to identify the immediate challenges we foresaw in its creation. Ultimately, we see users as being students, teachers, or clients. This information would need to be stored in a location that we can access and be displayed to the user with a front-end framework. A big problem in our client's current workflow is managing dozens of emails from multiple individuals. We would also like the web application to pull information from these emails and store it in the application to help centralize the information in a specific location. We created a figure displaying a broad overview of our system architecture as shown below.

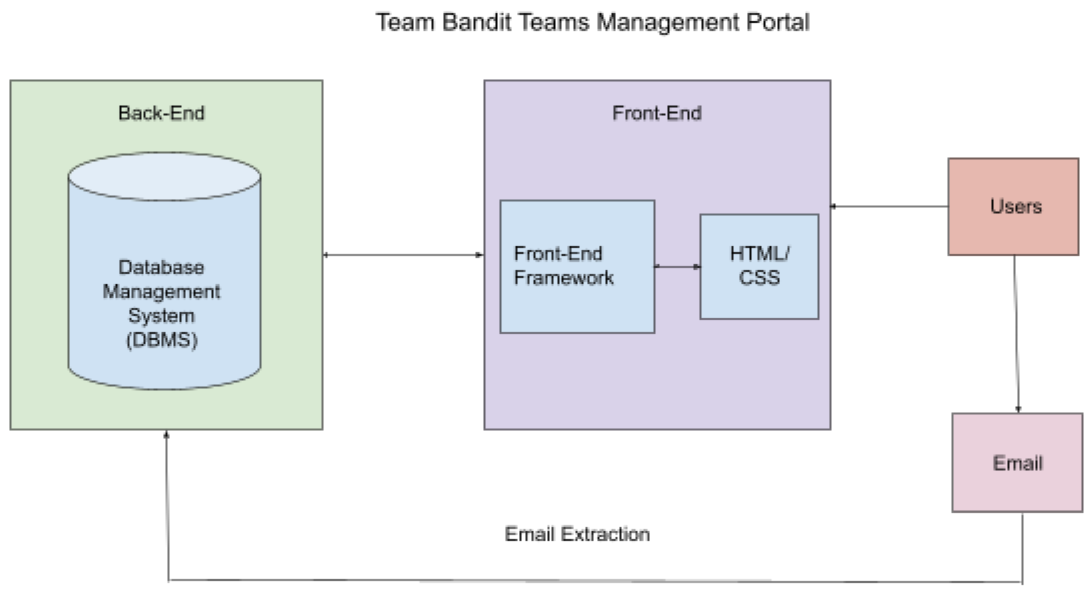


Figure 2.0: This diagram shows a broad overview of the interactions foreseen between the users, front-end, and back-end.

To shape the implementation of this architecture will require three fundamental design decisions:

- **Database Management System** - A database system is required to store the information of courses, clients, professors, mentors, and students, as well as the relationships between each.
- **Front-End Graphical User Interface** - A front-end framework allows you to break up your application into reusable, standalone components that makes it easier to implement quick changes that do not impact the rest of the application. This streamlines the creation of dynamic and responsive web pages.
- **Email Parsing** - An approach to parse and gather information from an email to store in a centralized location on the web application to address the arduous process of keeping track of emails.

Each challenge will now be introduced further and analyzed below in their own specific sections.

2.1 Database Management System

The TeamBandit web application will utilize a database management system (DBMS) in order to store information for its users. A DBMS is simply a software that manages the storage, retrieval, and updating of data. Most databases store data in a table for easy understanding of how multiple pieces of data may relate to one another. A database would be able to handle complicated operations called queries that can enable relationships between tables. This can allow us to, for example, directly relate a student to a course rather than having complicated objects in a CSV file. In order for the DBMS to fulfill the necessary requirements for the web application, it should accomplish the following:

- Have secured accounts for professors, students, and mentors. This is important because this DBMS will be handling sensitive data such as the credentials of university accounts from professors and students, and possibly credit card information.
- Professors need to create any number of courses, add any number of projects to a course, invite students to a course, and have the ability to sort the students into projects according to preferences stated by a student.
- Students will need to be able to sign up for an account and only view courses they have been invited to, as well as put information such as deliverables directly on to their team project page.

- Mentors will need to be invited to the course page so that they can view team pages that they are responsible for.

In order for the DBMS to help fulfill the requirements above, it should have these characteristics:

- **Flexibility** - Prevents the need for developers to revisit the code for the database to continually update functionality. This means that the database should be instantiated in such a way that it will not need to be updated by the team after a simple creation. In other words, there would be a simple and straightforward way to implement the database on, as an example, an Amazon Web Services Server where it is expected that the DBMS takes and gives the proper data from and to the user at the proper moment using easy to learn functions to manipulate and store data. The term Flexibility will also include the overall ease of use for the developers working with the database. For example, there is easy to access official documentation as well as an active online community that answers popular questions related to the DBMS from a developer standpoint.
- **Security** - The DBMS is expected to provide a way to store secure data, as well as storing a potential payment source to access this product, so it is important for the database to have security modules to make sure this information cannot be accessed by those who have malicious intentions. Since this database will be storing information related to individuals in education, it will need to comply with The Family Educational Rights and Privacy Act (FERPA), which protects personally identifiable information from students' education records from unauthorized access.
- **Scalability** - There should not be a limit to the amount of data the DBMS is able to store. Although our initial efforts plan to enhance our client's teaching experience, an effort must also be made to simultaneously support thousands of team based courses for national and global institutions. As an example, a limit to the amount of attributes that a user profile should possess should be large. The number of tables needed to support a large number of universities should be exceptionally large as well. Scalability will also refer to the hardware usage as well as the compatibility with the server that the web application will be hosted on. For example, we should know how efficient the DBMS uses memory and multiple CPU cores to avoid the server from being functionally overloaded with hardware-tasking requests from complex database queries.

The DBMS should meet the characteristics listed above to be a viable option to be used in the web application.

2.2 Front-End Frameworks

A key to success in this project will be a quality GUI and easy interaction with its various components. The front-end will be responsible for displaying information to the user in a way that they can understand. A front-end framework will allow our team to break our application into reusable components that will make it easy to make changes without impacting the rest of our application. Front-end frameworks also contain solutions aimed at addressing common problems in web development that will make it easier to get our web application up and running. In order for the front-end framework to meet the needs of our web application the following list of criteria have been designated. In order to properly compare each of the frameworks we will rate them on a scale of 5 for each criteria. These criteria include:

- **Visual Appeal** - The ideal front-end framework will contain the capabilities of making a graphically appealing web application. In order to receive a high grade in this criteria, the front-end framework will need to include some sort of UI module that contains pre-built components that are easy to add and change if needed.
- **Maintenance** - The ideal front-end framework will also include a low maintenance cost. One of the main advantages offered from a front-end framework is the ability to break up your code into reusable components. This allows for a lower maintenance cost as you can edit or change a component without affecting the rest of the web application.
- **Learning Curve** - As every member of the team, as well as future developers, will be expected to interact with front-end code, a low learning curve is essential. While learning any new technology can be cumbersome, having access to a strong online community and a wealth of learning resources can help ease that burden.

  These criteria will provide a qualitative metric for rating front-end frameworks on an equal standard to see the strengths and weaknesses when compared to one another.

2.3 Email Parsing

One important aspect of the TeamBandit Teams Management Portal is the ability for a professor, or any other authority-given users, to view emails sent to them related to the projects that are displayed in the portal. To ensure that this is a serviceable aspect of the portal, certain characteristics should be met. Three characteristics pertinent to this service are listed below.

- **Database Integration Flexibility -** In order to store information gathered from emails in a centralized location, it is necessary to make use of technologies which support the use of the chosen database. The optimal email parsing technology will be capable of interacting with a wide variety of database types. The functionality should remain consistent regardless of whether the database is MySQL, PostgreSQL, or any other relational model.
  A viable candidate will ideally support all of the databases in consideration.
- **Long-Term Viability -** With the amount of expected emails in a typical semester ranging into the hundreds, it is essential that the chosen technology for this task be capable of performing long-term without a possibility of issues later surfacing due to depreciation or arising incompatibilities as time goes on.
  A parsing solution that would adequately fulfill this criterion would maintain the same level of functionality over time.
- **Consistency -** The user has no way to know when an email will be sent, or how many to expect. It is imperative that this technology provides the assurance of consistency; that is, once set up, the software should not need regular or further configurations. For a task as straightforward as gathering the text from an email and storing it for later use, the software's overall utility would greatly suffer in the case that changes were consistently needed to provide the desired functionality. An inconsistent parser tool would need to be reconfigured regularly or occasionally. A tool that fulfills the expectations will not need to be reconsidered or adjusted in any way.

To consider the best option for this component of the system, the principal criteria to evaluate must reflect the specific needs of the product at hand. To begin picking the criteria with which our assessment would be based on, an analysis was started on the specific tasks that will need to make use of this technology. As the specifications now stand, there is one clear task requirement: capturing targeted email traffic accurately with minimal overhead for users, and making a related email chain accessible in the web application. It was established above that these operations will be structurally interconnected with several of the other chosen technologies and solutions. This enables us to pose a question which will serve as our guide to the comparison of our candidate solutions: *Which of the other components are used or otherwise interacted with during the execution of this task?*

The details of the identified dominant operation are broken down and analyzed in detail to answer this question, directly identifying each interdependent piece of technology by use of <u>underline</u> and the nature of the interaction by use of *italics*:

1) The system *recognizes* that an <u>email has been received</u>.

2) The system *copies the information* from the <u>sender and subject fields as well as the body's full content</u>.
3) The system *stores* this information in the <u>database</u> for later reference.
4) Upon a client request to serve the relevant page, the system *retrieves* all of this stored information via the <u>database</u>.
5) The system *inserts* this accessed information into the <u>webpage</u> in accordance with the predefined web templating structure and displays it.

Organizing the email parser's standard process in this way makes it simple to compile the details of interest with special attention directed toward the specific nature of the interaction. The first of the three guaranteed interactions in this flow is with the email server. The nature of this interaction is that the system automatically recognizes a new email and copies its contained information from various fields. The second is interaction with the database, which has two distinct interactions with the parser solution: storing data and retrieving data. Finally, the system needs to interact with the webpage, which will be structured via a predetermined template for visual consistency across the web application.

3 Technology Analysis

Having established the key technical challenges in the previous sections, this section will highlight possible technology solutions. To make the most informed decision, we decided that we would investigate at least two different options for each challenge. In order to adequately assess which solutions to investigate, we will utilize the internet to search through articles, forums, and each framework's documentation to properly identify which solutions are the best to investigate. According to the criteria decided upon in prior sections, each solution will be rated at the end of the analysis. One solution will then be picked after careful consideration and comparison between the various technologies.

3.1 Database Management System

The choice of an appropriate database management system relies on what application the database is dedicated to support and what purposes it will serve. In general, a DBMS should have the ability to store and retrieve user data, set levels of authority among users, and provide an easy to use development environment that would allow for the desired characteristics to be seamlessly implemented.

In section 2.1, three desired characteristics were introduced: Flexibility, Security, and Scalability. There is currently a multitude of databases to choose from for web applications. Popular databases include MySQL, Microsoft SQL Server, SQLite, PostgreSQL, MongoDB, and the Oracle Database. The databases that will be analyzed in this section for potential use on this project are PostgreSQL, MySQL, and MongoDB. The reasons that these database systems have been chosen are described below.

- Our client mentioned **PostgreSQL** as a database to consider as they had familiarity with it and thought it would operate well within this project.
- Many companies find **MySQL** easy to learn and use for web applications. It is a database used by many commercial businesses including Uber, Netflix, Amazon, and more. All members on this team have had some experience with a MySQL database, and is the only database that any of us currently have familiarity with.
- PostgreSQL and MySQL both share the fact that they use the SQL language and are both relational databases, meaning that the database is structured to recognize relationships among stored items. **MongoDB** is a NoSQL database, meaning “not only SQL”, that enables the storage of data outside the traditional tabular structure found in PostgreSQL and MySQL. MongoDB has a flexible data model that is best suited for storing semistructured or unstructured data, which is simply data that is not organized in a pre-defined manner and would be a departure from what the team is familiar with. This is a different approach, but it could provide us with faster and a more iterative development through the flexible schemas.
- The other databases listed earlier will not be further investigated due to a lack of optimized compatibility with a Linux based server as well as general unfamiliarity throughout the team. While MongoDB is one of these unfamiliar databases, it is one of the most widely used NoSQL databases, so for that reason, it is being further investigated.

Based on these reasons, PostgreSQL, MySQL, and MongoDB are the databases that will be investigated further. The reintroduction and investigation of these technologies and how they utilize the criteria designated in section 2.1 is detailed below.

3.1.1 PostgreSQL

PostgreSQL, or simply Postgres, is an open-source object-relational database management system. Originally, Postgres was a database system named Ingres and was developed at the University of California, Berkeley. It is designed to handle a large range of workloads, anywhere from web services with a heavy concurrent user load to large data warehouses. Postgres is the default database system for macOS Server, but is also available on other platforms such as Linux and Windows.

Our analysis of the pros and cons of Postgres included investigating the official documentation for Postgres, which was explored on the official website, postgresql.org. Along with understanding the basic syntax for Postgres when connecting to a database, research was conducted pertaining to how Postgres is installed on a system and how it handles user authentication. Basic hardware requirements were also investigated on the official Postgres site as well as any highly rated online forums that answered questions regarding the hardware requirements. Following this research, a Python implementation for securely connecting to a Postgres server and submitting various queries was implemented. This implementation was accomplished on a Mac with the following steps:

1. Install PostgreSQL on the machine through homebrew and create a database using the command line.
2. In the Python script, use the psycopg2 library, the most popular PostgreSQL database adapter for Python that is Python 3 compatible.
3. Establish a connection within the script and enter basic SQL queries such as creating tables with different data types and inserting data into these tables.
4. Use the command line once again to verify that the queries executed within the Python script made actual changes to the database.

Note that these general steps are used for all three analyzed databases. The slight differences that are present will be stated in the other respective sections for each database. The details following these tests and analyses of Postgres describing whether or not the DBMS fulfills our desired characteristics are described below.

Flexibility

- **PRO:** Postgres has very detailed documentation on their official site that can be fairly intimidating to a newcomer to database systems. However, if a developer already has some experience with any other SQL database, as well as having knowledge of basic developer traits, Postgres can be simple to adapt to one's knowledge.
- **PRO:** Has a command-line interface to provide a quick and easy way to view table data. The database for TeamBandit will likely be stored on a linux server. The team is familiar with the linux command line and can use the advanced textual interface to our advantage as the database tables are being constructed.
- **PRO:** Since Postgres is an object-relational database, it includes features that are comparable to features in many object-oriented programming languages such as inheritance and overloading. Inheritance enables us to share attributes between objects to provide an easier understanding of the database. As an example, students and professors are both users, so they would share common user functions, but professors would have more functionality on the website. This simply provides an easier understanding of the database implementation for the team.

- **CON:** The team is not too familiar with maneuvering around a PostgreSQL database on a server.

Security

- **PRO:** Postgres manages database access permissions using roles. A role can be a database user, or a group of database users. For instance, roles can own tables in a database and can assign privileges on those tables to other roles to control who has access to which tables.
- **PRO:**Natively supports a great number of external authentication measures, allowing us to utilize these and provide heavy security for the web application. For example, we can take advantage of built-in external authentication measures to protect passwords and ensure user authentication is preserved using the built-in Kerberos protocol, an authentication protocol that uses tickets to allow nodes communicating over an unsecure network to prove their identity to one another securely.

Scalability

- **PRO:** It can utilize multiple hard disks when multiple tablespaces are created, an important feature for ensuring multiple databases can be instantiated at the same time on one machine. This would be important for supporting multiple users creating courses at the same time.
- **PRO:** Uses partitioning, allowing the hard disks to be accessed simultaneously, which makes data processing faster. While not an extremely important feature for this web application, it may be of use if the product becomes a widely used application.
- **PRO:** Postgres is designed to use multiple CPU cores on a machine, can use all available memory for caching, the size of the database is essentially not limited.
- **PRO:** The hardware requirements for Postgres are a 1 GHz processor, and 2 GB of RAM. Note that this is closer to the bare minimum rather than what is preferred, with the typical preferred requirements being a 2.2 GHz processor with a high number of CPU cores, and 4 GB of RAM. These requirements can be easily met by the team.
- **CON:** Consumes memory and CPU resources even when idle. Memory that gets allocated when queries are run is not completely freed up even when the connection goes idle.

3.1.2 MySQL

According to the 2020 Stack Overflow Developer Survey, which surveyed about 65,000 developers, it was indicated that the most popular database today is MySQL, a relational database management system that is also a free and open-source software under the GNU General Public License. MySQL is currently owned by Oracle, an American computer technology corporation, being previously owned and sponsored by the Swedish company MySQL AB. MySQL is often used with other programs to implement applications that need relational database capability. MySQL is used by many database-driven web applications, including Drupal and WordPress. MySQL is also used by many popular websites, including Facebook, Twitter, and YouTube.

Installing MySQL, as well as creating a functional database, is something the entire team has familiarity with on past projects. The approach to create a working implementation of executing queries on a MySQL database followed the same steps taken in 3.1.1 for the PostgreSQL analysis. The Postgres and MySQL implementations only contrast in the database that was installed and the library used in Python. For MySQL, mysql.connector was used to connect to the database. The code can be nearly identical since both Postgres and MySQL use the SQL language. The points detailed below are based on the research conducted using the official website, dev.mysql.com, online forums, and the past experiences that the team has working on a MySQL database directly.

Flexibility

- **PRO:** MySQL can be easily accessed by developers with basic foundational knowledge of the SQL language and not need to have experience with any other database. Because of this, MySQL has a reputation to be easy to adapt to for many database purposes and is used by many companies.
- **PRO:** It is well supported by many major platforms including WIndows and Linux, where constant updates are provided to ensure widespread usage of the popular DBMS.
- **PRO:** The team has familiarity with MySQL, as we have all understood basic table creation within a programming language using MySQL. This would provide the team with extra confidence when using the database.
- **PRO:** An extremely active online community with nearly every possible issue being answered somewhere online. This can be used to the team’s advantage if problems were to arise.
- **CON:** Is not filled with an abundant amount of features as there are in other relational databases such as PostgreSQL. An example being materialized views, which is a database object that contains the results of a query.
- **CON:** Has closed sourced modules since Oracle does not release test cases for bugs in the database. This can defeat the purpose of the database being completely open sourced and could possibly limit what this web application could have access to with regards to the database in the future.

Security

- **PRO:** Similar to Postgres, MySQL has roles. This enables the assignment of privileges to accounts and provides an alternative to granting individual privileges.
- **PRO:** Enables the configuration of MySQL to use Linux PAMs (Pluggable Authentication Modules) to authenticate users for various authentication methods, such as Linux passwords.

Scalability

- **PRO:** Similar to Postgres in a sense that MySQL has the ability to take advantage of the hardware on a machine since it is so well optimized, but does not provide as advanced usage of multiple hard disks as Postgres does.
- **PRO:** The minimum hardware requirements for MySQL are not defined by Oracle, but online resources suggest that bare minimum requirements are fairly low, being only 512 MB of RAM.

3.1.3 MongoDB

Software company 10gen began developing MongoDB in 2007, and in 2009, the company shifted to an open-source development model, with the company offering commercial support and other services. In 2013, 10gen changed its name to MongoDB Inc. and in 2017, MongoDB became a publicly traded company. MongoDB uses files that are comparable to JSON files. licensed under the Server Side Public License. NoSQL databases have dynamic schemas for semistructured or unstructured data. NoSQL essentially means that the database provides mechanisms for storing and retrieving data in ways other than relations among tables.

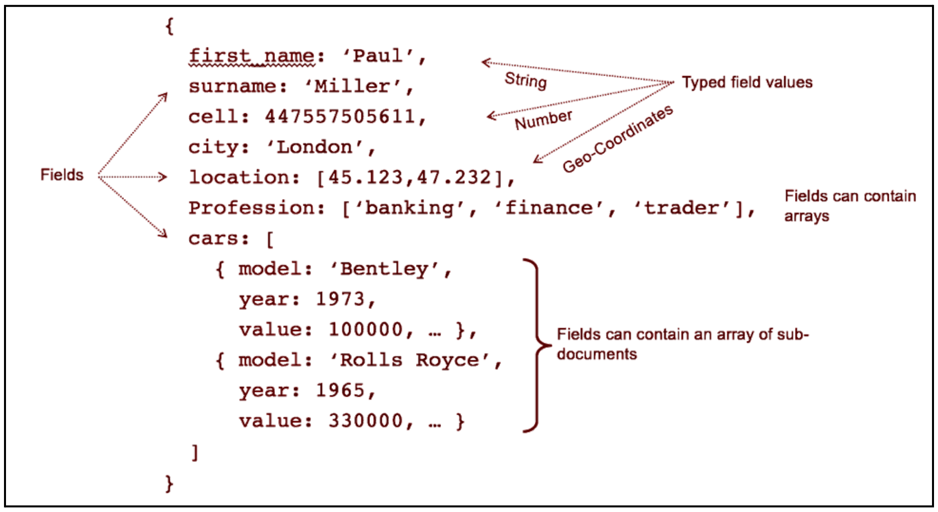
The research detailed in this section was received from the official documentation, docs.mongodb.com, as well as active online forums pertaining to issues that developers typically face when using MongoDB. Using the pymongo library in Python resulted in a working Python implementation to connect to and add data to a MongoDB database. A visual example of the type of data stored in a MongoDB database will be shown below along with the analysis of the desired characteristics using pros and cons.

Flexibility

- **PRO:** Column-oriented, document-oriented, graph-based, or KeyValue stores can be used for data in a MongoDB database. This provides less restrictions for the team and can be used to our advantage by storing data in a way we see fit rather than sticking to a restrictive SQL structure.

- **PRO:** MongoDB allows for the change and addition of fields as the database is being developed, so a structure does not need to be determined before implementation.

An example of the type of data that a MongoDB database would store is shown below and is taken directly from a sample document on a blog post on mongodb.com.



```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields
String
Number
Geo-Coordinates
Typed field values
Fields can contain arrays
Fields can contain an array of sub-documents

Figure 3.1.3: Sample JSON document from mongodb blog post

From this sample document, it is apparent that we can get creative with how we store our data in the database rather than adhering to a simple table structure with pre-defined fields for each piece of data that is inserted.

Security

- **PRO:** using what MongoDB currently offers, we would be able to enable access control among users accessing the database, configure role-based access, and encrypt all incoming and outcoming connections using TLS/SSL.
- **CON:** In the past, data from tens of thousands of MongoDB installations has been stolen. This does lessen the confidence that the team has for this solution.

Scalability

- **PRO:** Referring to the Flexibility advantages of MongoDB, scalability does not seem to be an issue for this scale of a project as different data can be stored in different ways, allowing for the database to be scaled in a way that suits our needs to create this web application.
- **CON:** However, if MongoDB is used as a primary database for a thousand plus machines, it cannot guarantee full data protection or provide scalability due to the document type or unstructured data storage.

- **CON:** The minimum memory requirement for MongoDB is at least 1GB of RAM, and if memory begins swapping to the disk, performance is expected to suffer heavily.

As seen above, MongoDB comes with far more cons than pros, and coupled with our team's relative lack of experience with NoSQL databases, MongoDB is the least likely to be chosen from the analyzed databases.

3.1.4 Analysis of Outcome

	PostgreSQL	MySQL	MongoDB
Flexibility	4	5	3
Security	5	5	4
Scalability	5	4	5
Total	14	14	11

Table 3.1.4: Results of the analysis done on the possible database management systems. Rating is on a 1-5 integer scale, with 5 being the best, and 1 being the worst.

After conducting an analysis of the three different proposed solutions to this technological challenge, rankings for each of them have been determined in the table above. From the analysis of Postgres in section 3.1.1, Postgres receives a 5/5 in both security and scalability, but receives a 4/5 in flexibility since there are a variety of features to learn, which results in a small learning curve for the team.

MySQL easily receives a 5/5 in flexibility and security for having simple and familiar features for the team, as well as being generally secure as long as the developer takes the correct measures, similar to Postgres. Since MySQL cannot handle extremely large databases as well as Postgres can, it gets a 4/5 in scalability.

Since MongoDB does not have a reputable security history, but has since consistently updated its security, it receives a 4/5 for security. Even though a NoSQL database would allow us to store the data in whatever way we see fit, the team has almost no past knowledge of NoSQL databases and no familiarity with MongoDB's flexible schemas, which leads to flexibility to be scored at a 3/5 for MongoDB. A 5/5 is given for scalability since, for the purposes of this project, MongoDB is able to handle an ideal amount of data for data analysis.

Based on the scores given, PostgreSQL and MySQL have tied for first place for the best solution to this challenge. With PostgreSQL having advanced relational database features, a rising online community, great conformance to the SQL language, and a recommendation from our client, PostgreSQL is our preliminary choice as the database to use for this project.

PostgreSQL is fairly simple to implement on any Linux machine and is easily accessible from programming languages such as Python and JavaScript, which will be used in this project. Over the years, Postgres has implemented many external features for developer use and is strictly conformed to the SQL language. We are confident that PostgreSQL is a feasible solution for this technological challenge for this project.

3.2 Front-End Frameworks

For our project, as a web application, especially valuable functions of the front-end will provide access to components that will speed up development of the web application. These components offer a base style as well as functionality that will allow our team to quickly come up with a prototype for our web application. These components consist of UI elements such as Menus, Buttons, Tables, and much more. These components will allow us to quickly create responsive and clean looking user interfaces so we can focus more on the functionality of our website.

Searches for the most popular front-end frameworks revealed Angular, React, Django, Vue, Svelte, Bootstrap, and Ruby on Rails, just to name a few of the more popular frameworks. Instead of analyzing each framework based on the criteria dictated in section 2.2, the initial frameworks would be cut down primarily by its popularity and the age of each framework (an older framework is likely to have more features and less bugs). My analysis of each initial framework is listed below:

- **Angular** - Upon looking into Angular, it was immediately added to the list of potential front-end frameworks. Angular is supported by Google and is hailed as one of the best frameworks to get to know by many forums and articles. Angular is strong at creating efficient and sophisticated single-page applications.
- **React** - Similarly to Angular, React is also hailed as one of the best frameworks on forums and articles. React is supported by Facebook and offers a JavaScript library for building user interfaces or UI components. React being popular, and supported by Facebook caused React to also get added to the list of potential front-end frameworks.
- **Django** - Django has been around since 2005 as a Python-based web framework. Django’s primary goal is to make it easier to create complex, database-driven websites. Django stood out from the previous front-end frameworks as a Python-based web framework. Our team is very familiar with Python and would like the low learning curve associated with Django.
- **Vue** - Vue was created by Evan You after working for Google on Angular. He wanted to take elements from Angular that he liked while introducing new features, such as directives which are a special token that tells your

programming library to do something on your web applications DOM (Document Object Model).

- **Svelte** - Svelte is a relatively new front-end framework as it was created in 2016. Svelte compiles the majority of your web application before it is displayed on your browser. React and Vue for example do the bulk of their work in the browser. Svelte updates the DOM when the state of your application changes.
- **Bootstrap** - Bootstrap was initially released in 2011, and is directed in a mobile-first mindset. Bootstrap contains CSS and JavaScript based design templates for interface components such as typography, buttons and navigation. Although Bootstrap offers the ability to create visually appealing websites, it lacks the foundations to create actual functionality of your web application.
- **Ruby on Rails** - Having its initial release in 2004, Ruby on Rails is the oldest framework in this list. Ruby on Rails provides default structures for databases, web services, and web pages. Ruby on Rails offers seamless database table creations and migrations.

React and Angular easily grabbed spots for further research with their modern approach to web development, their financial support from tech behemoths like Google and Facebook, and its popularity offering learning resources in forums, videos, and documentation. Django ended up being the final front-end framework to look into as it is written in Python and offers a wide range of modules such as user authentication. Vue, Svelte, Bootstrap, and Ruby on Rails all had constraints that ended up deterring us from wanting to pursue them further. With Angular, React, and Django we will have the freedom to create our web application how we want. Angular, React, and Django will be analyzed further based on the desired criteria discussed in section 2.2 which includes visual appeal, maintenance, and the learning curve. We decided to test and research each front-end framework in an identical manner in order to compare each framework with similar standards. The methods we chose to pursue for each criteria are detailed below:

Researching how each framework handles GUI creation is a key component in evaluating the visual appeal criteria. After understanding these basics for each framework, viewing real world examples of websites that utilize them to see these aspects in action will be the final step of analyzing this criteria.

The maintainability of each framework will be analyzed by researching how each framework goes about creating a web application. High modularity is desired, as this will make it easier to break the application into reusable components, speeding up development and repair times dramatically. Another goal is to determine the amount of support that is given for each framework and language.

Each framework will offer a varying amount of learning resources from forums, documentation, and videos. We will identify what each framework offers on their websites as well as through applications such as YouTube. We will also identify what coding language that each framework uses and compare them to the skill level our team already has with each framework.

Finally, we did not want to make a choice for the front-end framework without hands-on testing. We tested each front-end framework to see how interacting with them realistically performs and decided to create a simple “Hello, World!” test page with each. This will be accomplished by:

1. Downloading each framework using their respective download methods.
2. Figuring out how to create a HTML web page that says “Hello, World!” utilizing the frameworks forums and documentation.
3. Implementing the solution found and reflecting on how simple it was to learn and utilize each framework.

Below, we will reintroduce each chosen framework and talk about the results from the research we found from the criteria listed above. Each framework will list its results in a pro and con format to easily identify advantages and disadvantages of the framework in regards to the criteria.

3.2.1 Angular

Angular is a development platform, built on TypeScript, for creating efficient and sophisticated single-page applications. Angular is a popular front-end framework with a strong community and support from Google. Development on Angular started at Google in 2009 by Misko and was released as open source because the team wanted to make it easier for all developers, not just google developers, to build great web applications. Below, Angular’s ability to accomplish the desired characteristics of a front-end framework are discussed.

Visual Appeal

- **PRO**: Angular comes with a package called Angular Material which comes with GUI components for easy creation of a GUI. These components offer functionality such as a calendar datepicker, autocomplete forms, and progress bars. These components will allow us to implement them onto our web application without directly coding them ourselves.
- **PRO:** Angular Material has straightforward Application Programming Interfaces (APIs), with consistent cross platform behaviour. Angular Material follows the formatting of TypeScript, which allows for its functions to be easily integrated. Angular Material’s components also perform consistently on all platforms, meaning a Button will work the same on a Desktop view as it would on a Mobile device viewing the same page.
- **PRO:** Proven efficiency in creating web applications with strong visual appeal include Google, Gmail, Paypal, and Upwork.

- **CON:** Angular Material is customizable within the bounds of the ‘Material Design Specification’ given. This means that we can utilize these components, but the customization of what we would like them to look like is limited.

Maintenance

- **PRO:** Angular Breaks up its web application into components by utilizing a TypeScript class with the @Component() decorator. This is exactly the type of functionality we are wanting with a front-end framework as it will allow us to create separate reusable components to utilize throughout our web application.

Learning Curve

- **PRO:** Angular comes with a well formatted documentation that makes it easy to learn about its various elements. The documentation also comes with hyperlinks to other parts of the documentation if you are confused about a certain subject.
- **CON:** Our team has little to no experience with TypeScript. This is a con when compared to React which uses JavaScript and Django which uses Python as we are familiar with those languages.
- **CON:** Angular offers a wide range of solutions from problems, such as Components, Templates, and Directives for its HTML creation. This can make it hard to learn as it presents multiple solutions when we only really need to use one.

Upon utilizing Angular to create a “Hello, World!” test page, it was revealed that it has no feature to implement the displaying of ‘Hello World’, the features implemented are merely style oriented and still need to implement ‘Hello World’ in a .HTML file. This immediate showcase of Angular’s components is exactly the type of styling wanted from the front-end framework, however setting up Angular’s components was challenging.

```
import { Component } from ‘@angular/core’;

@Component({
    selector: ‘app-root’,
    templateUrl: ‘./app.component.html’,
    styleUrls: [‘./app.component.css’]
})

export class AppComponent {
    title = ‘hello-world’;
}
```

Figure 3.2.1: Code for implementing a ‘Hello World’ feature in Angular

3.2.2 Django

Django was initially chosen as a candidate for a front-end framework as the team members were all familiar with Python and Django utilizes a high level Python. Django was created in 2003 by Adian Holovaty and Simon Willison, when they began using Python to build applications. Django is supported by an independent foundation established as a non-profit, and primarily acts as an open-source project. Below, Django's ability to accomplish the desired characteristics of a front-end framework is discussed.

Visual Appeal

- **CON:** Django comes with no modules for creating a visually appealing website. Compared to Angular which has Angular Material and React which has Material UI, Django would require an additional framework such as Bootstrap for creating its GUI.

Maintenance

- **PRO:** Django utilizes a high level version of Python to create HttpResponses in its functions. This would be great for database interactions with our user tables. This will allow us to keep the modularity we are wanting.
- **CON:** Since Django offers no support for UI, we would need a different framework as another technology to learn. This leads to an uncertainty if the modularity from the Python functions would be easier when another technology is added.

Learning Curve

- **PRO:** Python is a language the team is very familiar with. This will allow us to focus more on the intricacies of Django itself then on learning Python.

Utilizing Django for creating a “Hello, World!” test page, there was little set up to be had on the web application side for the “Hello, World!” to work properly. Django required more work to set up .PY scripts to access the correct server to send its HttpResponse to. The HttpResponse is what actually displays the “Hello, World!” information. The simplicity of utilizing Django as a Python language would be a familiar concept for the team.

```
from django.http import HttpResponse

def homePageView( request ):
    return HttpResponse(‘Hello, World!’)
```

Figure 3.2.2: Code for implementing a ‘Hello World’ feature in Django

3.2.3 React

React is a free and open source front-end JavaScript library for building user interfaces or UI components. React was created in 2013 and is maintained by Facebook and a community of individual developers and companies. Below React’s ability to accomplish the desired characteristics of a front-end framework is discussed.

Visual Appeal

- **PRO:** React comes with a module called Material UI which can be utilized in a similar fashion as Angular Material. This means Material UI offers graphical components that we do not have to program ourselves, such as buttons, tables, and dashboards..
- **PRO:** Proven efficiency in creating web applications with strong visual appeal include Spotify, Amazon, NASA, and Netflix

Maintenance

- **PRO:** React utilizes JavaScript functions to return HTML components. This is exactly the type of modularity we were looking for in a front-end framework, as we can break up our web application into reusable components..

Learning Curve

- **PRO:** React’s simple use of JavaScript functions and our team's familiarity with JavaScript make for little learning when pursuing React.
- **PRO:** React is one of the most popular frameworks which offers a plethora of learning resources from its well formatted documentation to forums and YouTube videos from the community.

Upon utilizing React to create a “Hello, World!” test page, React ended up being the easiest to set up and code for the “Hello, World!” test page. All it required was creating a single JavaScript file that included four lines of code. The simplicity of utilizing React to create “Hello, World!” gave React a high score on the learning curve.

```
ReactDOM.render(
        <h1> Hello, world! </h1>,
        document.getElementById(‘root’)
);
```

Figure 3.2.3: Code for implementing a ‘Hello World’ feature in React

3.2.4 Analysis of Outcome

	Angular	Django	React
Visual Appeal	4	1	4
Maintenance	5	3	5
Learning Curve	3	4	5
Total	12	9	14

Table 3.2.4: Analysis of Front-End Frameworks. A score out of 5 was then given for each characteristic. Rating is on a 1-5 integer scale, with 5 being the best, and 1 being the worst.

Each front-end framework was rated based on the desired characteristics detailed in section 2.2. Angular has the frameworks and backing of a strong community to act as a strong competitor for the front-end framework. Despite having these attributes, the team is unfamiliar with TypeScript despite it being an off branch of JavaScript. Django was appealing in the sense that it utilizes Python. Despite not coming with GUI itself, the frameworks offered by Django offer plenty of features that would be utilized in the web application. Finally React acts as an easy to use framework with tons of room for users to create their desired web application. React comes with a strong community and a popular language in JavaScript. The grades given to each framework are displayed below.

After looking into the various frameworks and giving each one a grade, a conclusion was reached that React would be the best preliminary choice. Each framework offered various pros and cons but in the end React ended up having the best overall scores between the three frameworks. React offers the ability to customize the web application according to the way we specify without limiting us to a specific framework or module.

We plan to use React to quickly prototype the web application that will act as the final product until a more permanent front-end is decided on. Despite planning to use React, there may end up being a technological challenge that requires a different solution. The most difficult challenge that can be foreseen is the implementation between the DBMS and the front-end framework.

3.3 Email Parsing

The TeamBandit Portal’s functionality encompasses the need to capture the text from the emails, store that information, then retrieve it for display to the page. A pressing challenge posed is that essentially all measurable criteria for this technology pertains to the selection of *other* technologies and frameworks being used in the project. Another challenge that arises is that of this component's dependency. If another component of the project later proved unfeasible at a later point in time, the selected technology for the email parsing tasks would need to be reconfigured at best, fully re-evaluated to consider an alternative technology at worst.

The frequency in which emails will be received is not known to the system or the user, so it is necessary to verify the solution's ability to automatically detect and act upon emails being received. The information contained in the emails is necessary to use the product effectively and correctly. By having the risk of failure present, crucial information could be omitted entirely from the web application. This research indicates that a typical email parsing solution parses within the constraints of specified or provided instructions; the results must be uniform. The criterion of performance consistency will be included. Once set up, a solution should not need further configurations or regular maintenance.

Storage and retrieval are the fundamental purposes of any database, so it can be concluded that the actual *capability* to perform these two types of operations is not a worthwhile criteria; any database that is supported by the chosen email parsing technology will possess this capability. The condition that is depended on is the *compatibility* with said database. Thus, one of the criteria for the email parsing solution is the ability to interact with any form of database or frameworks that may be used in the product.

The use of the product is expected to be reoccurring every semester of the college school year. This provides the need for longevity. Our solution for email parsing must be capable of maintaining its standard of performance long-term without a possibility of issues later surfacing due to software depreciation or arising incompatibilities that result from extensive use.

This technology's direct and heavy influence on the system's ability to serve its purpose must be adaptable. Almost every software development process encounters unexpected challenges during the implementation phase, so the solution needs to be able to be reconfigured to account for changes that may be necessary to solve those arising challenges: adaptability.

Each technology was discovered by exploring forums including Reddit and TechSpot to look at discussion on mail parsing solutions by real people with varying uses for the tools. There were two primary solutions among each of these.

3.3.1 Mailparser

The performance of Mailparser topped the list on several of the forum comparisons analyzed. This product has expanded its functionality and features over the span of more than seven years. It is actively supported and maintained by a

10-person team which includes support specialists for assistance with the technology. The product's testimonials come from people who make use of the application for an extensive variety of purposes.

The primary features emphasized by this application are its capability for detailed customization of parsing rules and a list of compatible integrations larger than any viable competitor.

Database Integration

- **PRO:** Integrations include essentially every database management system that the team is remotely familiar with.
- **PRO:** This wide set of options for database integration distinguishes this solution as a top performer in the evaluation of flexibility among database interaction.

Long-term Viability

- **PRO:** Mailparser's long-term viability is promoted heavily by its active support services and continued maintenance of the project.
- **PRO:** The software's active management provides an assurance that this application will be relevant and adequate for years to come.

Research was conducted on how Mailparser performs its duties and examined reviews of those who use the application on a large scale.

Result Consistency

- **PRO:** Pressing mentions regarding inconsistency of results, even after long-term constant use were not found from the analysis done. This is indicative of its ability to perform consistently and repeatedly with no progressive cost.

A notable barrier to this solution is its terms of purchase; it is only available via a monthly monetary subscription priced based on the quantity of permitted usage of the product. For this reason, it may be a non-optimal decision given the anticipated lifespan of the project to choose this option if there are sufficient alternatives that do not pose this requirement, or do so at a lower monetary expense.

3.3.2 Zapier Email Parser

Zapier Email Parser is a solution provided by Zapier's large library of free-to-use tools for automating a massive variety of tasks for both business and personal purposes. This automation tool is specifically designed to support piping the data to a range of thousands of different applications, websites, and services. Zapier was founded in 2012 and continues to offer consistent support and development.

This parsing solution can connect with custom Zapier mailboxes that can be used to make the email parsing procedure direct and simple in setup. Discussion of use in online discussion boards indicated that many people use Zapier as a single component in otherwise larger systems. Seeing as how this is exactly how we intend to use this, the confirmation that this is a feasible and intended use of this tool makes it relevant to consideration.

Database Integration

- **PRO:** Zapier Email Parser offers direct integration with a variety of database types including mySQL, PostgreSQL, and MongoDB.

Long-term viability was analyzed by inspecting the method Zapier uses to handle the collection and exporting of email-contained data.

Long-term Viability

- **PRO:** The tool is organized around setting a trigger that sets off an action. For the purposes of TeamBandit, the trigger is simply receiving an email and the resulting action is exporting select pieces of that information to the desired application or medium.
- **PRO:** This structure ensures that no additional conditional scenario is involved that could potentially cause a detrimental variation in the performance further down the road.

The final criterion of consistency in results after initial setup was judged by Zapier's setup process.

Result Consistency

- **PRO:** The setup process allows you to define a rule set which simply defines the trigger and resulting actions.
- **PRO:** Zapier allows users to save rule sets and toggle their functionality on and off at will. Because this solution keeps a saved record of a configuration and repeats its exact specified operations, we can be confident in the application's ability to stick to those configurations.
- **PRO:** Simply put, the task need only be set up once and will not need to be attended to again.

Zapier's quick and free setup of this tool enabled an easy verification that the information can be accurately and consistently transported. The simple test conducted to gain this verification was simply defining a rule set that collected and stored the body of an email, then sent it to another email inbox, where the body could be analyzed for inconsistencies. After a deep analysis, no inconsistencies were found.

3.3.3 Analysis of Outcome

	Mailparser	Zapier
Database Integration	5	5
Long-term Viability	4	5
Result Consistency	5	5
Cost	$40/mo 500 emails	$20/mo 750 emails
Total	14	15

Table 3.4.4: Analysis of Email Parsing Solutions. Rating is on a 1-5 integer scale, with 5 being the best, and 1 being the worst.

The investigations for selection of this technology lead to the finding that the most well-known tools for email parsing tend to produce similar results overall, likely as a result of the minimal demand for variation being present in the functional needs of individuals and organizations who make use of parsing utilities. They seem to take very similar approaches to getting the job done which makes it even more difficult to identify scale-tipping differences.

This finding led to a final element of criteria not initially considered: cost. With this taken into consideration, Zapier comes out on top in this comparison with respect to lower cost, more offered for the cost, and more offered in the free version which will help to support more accessible testing.

Both MailParser and Zapier Email Parser were exceptional in addressing the posed questions of criteria. Each displayed adequacy in its operations for purposes of the TeamBandit Portal.

The element of criteria concerning consistency in results encompassed the details of the setup process. Zapier Email Parsing simply topped this element with just how simple it is to create a fixed rule set that is followed directly at the signal of a trigger and the preset options which offered not only integration with a database, but also quite a bit of flexibility in *how* those details were presented to the database.

The monetary cost to access these tools was worth factoring in after noting how efficiently the requirements were met, and this was the reason MailParser ranked lower than Zapier Email Parser for long-term viability; increased expense for essentially the same functionality simply contradicts this. This led to the conclusion that the most feasible of these options for our needs is Zapier Email Parser.

Plans for further testing with our specific use case need to prove this solution's viability in working with the other chosen technologies and functioning under varying circumstances. To do this, a series of multiple emails with complex and varying content and styles will be sent to TeamBandit's mail service of choice to ensure that the information is accurately scraped and transported to where it needs to go each and every time.

4.0 Technology Integration

In order for the web application to succeed, all of the various technologies chosen must interact seamlessly with one another. Certain technologies have conflicts with others, however, those chosen in this document do not have any such limitations. Both PostgreSQL and React appear in many production stacks, such as PERN (PostgreSQL, Express, React, Node), proving feasibility for their integration. Having these two appear together in so many tech stacks also means that any problems as a result of their coupling will likely be able to be resolved via research on various forums and talking spaces. Zapier, our selected mail parser, also features PostgreSQL as one of its supported applications, reassuring us that using the two in cohesion will be simple and straightforward.

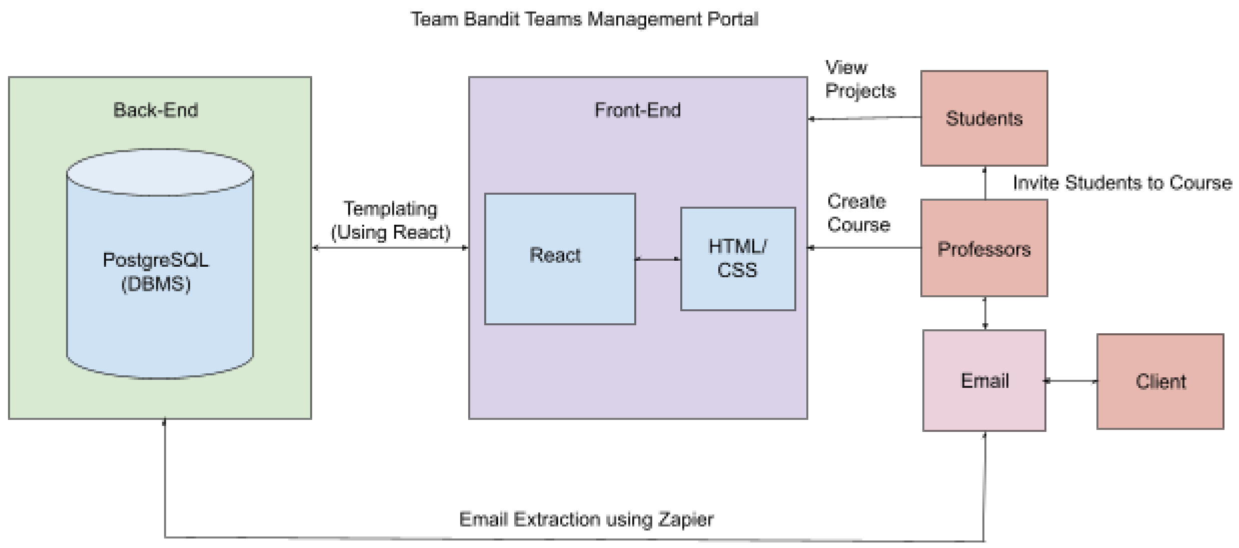


Figure 4.0: Interaction between the users, the front-end and the back-end to create a cohesive working web application.

The interactions between each of the respective components that we have chosen are shown above. Our DBMS, PostgreSQL, will gather information that Zapier passes through, parsed from email interactions between the professor and client users. React, our front-end framework, will display information pulled from our database using templating.

5.0 Conclusion

The TeamBandit web application will be built on an integration of several foundational technologies to produce the final web application. This web application will act as a centralized location for faculty teaching team-based courses, such as our client, to streamline preparation of those courses. This includes creating new courses, gathering information from emails to view in a collective location, adding projects to the course, inviting students to join a course, enabling students to submit project preferences, sorting students into teams, and allowing students to submit assignments for their projects. Key technological challenges associated with creating this web application include: a database management system, a front-end framework, and email parsing. Below is the table containing our chosen technologies paired with a confidence level, which was determined by the technologies given score in their own respective sections.

Challenge	Chosen Technology	Confidence Level
Database Management System	PostgreSQL	14/15
Front-end Framework	React	14/15
Email Parsing	Zapier Email Parser	15/15

Table 5.0: Reiteration of chosen technologies for each challenge

PostgreSQL has been chosen as our preliminary solution for a database as it offers extensively detailed and official documentation, an actively rising community, an abundance of advanced database features, and a simple implementation on an Amazon Web Services server.

The team has chosen to use React for the front-end framework. React offers a base front-end framework that allows the users to create exactly what they need. React also offers dozens of modules like Material UI to help with the creation of a nice clean GUI. Our team is confident that React will offer a strong foundation for creating the front-end.

The chosen email parsing solution is Zapier Email Parser. Its setup is quick, reliable, and consistent and it is flexible in its integration with the above chosen database. For TeamBandit's needs, Zapier Email Parser will get the job done.

With the proposed technologies, we believe that we will be able to create a tech demo by the end of this fall semester and transition it into a working web application for our client. This web application will allow our client to easily manage their capstone course while keeping the information associated with the course in a centralized location.