Technology Feasibility

Jack's Peer Pages

October 15th, 2025

Sponsor: Dr. Keith Nowicki
Team Mentor: Md Nazmul Hossain
Faculty Mentor: Dr. Ana Chavez

Team: Haley Berger (Team lead), Haley Kloss (member), Jeremiah Lopez (member), Tyler Austin (member)

Table of Contents

Introduction

Science is dependent on the ability to share knowledge, but publication of research has failed to keep up with the demands of modern scholarship. The peer-review system, established more than seventy years ago, is faltering under the weight of today's publishing environment. Millions of new articles are produced annually, and with the growing use of tools like large language models for writing and editing, matters of credibility and trust have become increasingly pressing. What was once an assurance of scientific quality has now become an overused process that renders it difficult for researchers, particularly new researchers, to sift through an overwhelming amount of work.

Peer reviews are not currently widely and easily accessible. This makes it difficult for young researchers and students to learn from them to understand how to make their own. It also makes it difficult for researchers in general to find the most up to date information about a specific article. For example, a paper review might include a correction to an element in a research paper, the discovery of which could improve scientific understanding. Peer reviews are often conducted, but remain difficult to access and sometimes even locked under a pay wall through journals. It's as though they are separate from the world of academics, despite its necessity. At Northern Arizona University, Dr. Keith Nowicki and his research colleagues recognized this challenge as well as emerging opportunities through which graduate students can practice and refine reviewing skills. This includes getting students to actively engage in peer review to properly train a new age of researchers. Despite their proposal which has students do peer reviews each semester, there is no proper infrastructure for aggregating and dispensing those reviews. This ultimately makes the effort solitary and less valuable than it could be and does not solve the dissemination and access problem previously mentioned.

Our project, Jacks Peer Pages, solves the infrastructure and dissemination issue. It's a web-based tool that connects peer reviews with the publications they're reviewing, so they're searchable and accessible. It will also offer a way of notifying users whether the original article publisher has used artificial intelligence prior in their writing for the sake of transparency and potential information accuracy. By creating a shared pool of peer reviews, the platform allows for student training, enhances literature review practice and publication honesty, and builds a resource beneficial to the wider academic community that is not locked under a pay wall. The key features of Jacks Peer Pages that will facilitate this are: a website that is hosted with scalability in mind, a system for making accounts that are verified, a hosted backend database of articles and peer reviews, a search feature so that these articles and reviews can be accessed, a feature that allows users to make a peer reviews, a system by which an article can be flagged on suspicion of AI, and an easily navigable UI.

The intended outcome is a more open and credible system of peer review that benefits students by giving them valuable experience while also improving access to the dissenting opinions in peer reviews that challenge ideas and build scientific knowledge. By doing this, Jacks Peer Pages will rebuild trust in research and prepare future generations of scholars to further the values of integrity and rigor in science.

Now that we have established why our project is important and what the intended product will be, it is important to discuss the technological challenges that might be faced in the development of the project and how they can be solved. By outlining both these issues and several alternative solutions during the initial stages of the development process, we will set our team up for success during implementation. In this Technological Feasibility Analysis document, we will begin by outlining the four primary technical challenges that we expect to face. Then, we will carefully research each of these issues, summarize the potential solutions, and weigh them against one another. Each issue will end with our final decision for what technology to use to handle it, a detailed explanation of this choice, and a proof of concept. This begins on the following page with a section listing our central Technological Challenges.

Technological Challenges

We have identified a small subset of potentially major technical challenges that could pose problems or setbacks in our project. Those challenges include:

1. (Backend database) We need a scalable and secure database for any number of users to make an account and for any additional website data, including but not limited to user posts. For our website to be effective at providing peer reviews and articles to users quickly, the database must be able to handle complex queries while still remaining efficient. There are several alternatives that can be used for this, such as NoSQL, Postgres, and MySQL, which will be evaluated in this section. It is a priority for our database to be scalable, secure, and efficient in order for the website to support the data requirements of its users.

2. (Backend database hosting) We need hosting for our database that will support concurrency for fetching and reading data from the database into the website, which will also support data input. Furthermore, the hosting platform will need to communicate quickly with the database itself so that users are not stuck waiting for their data requests to be fulfilled. This will allow the multitude of users on the front-end to access any information needed promptly through the hosted database.

3. (User authentication) We need a secure, verifiable user authentication system on our website. This is because our project is predicated on creating a compendium that is strict about AI usage and does not contain bots, which makes it necessary to verify user accounts. This can be done through several methods, such as CAPTCHA services, email authentication, and Multi-Factor Authentication (MFA). Each of these approaches provides additional security to our user account system to ensure that all users are real people who are interacting with the site as intended.

4. (Fuzzy search) We need a fast lookup system for searching documents and authors. It must also offer recommendations, similar searches, and account for misspellings. This project will include a huge database of documents and peer reviews, so this search up function will need to be fast and responsive. The database will also constantly be growing with more documents being put in, so the fuzzy search should be scalable as well. All of this would need to be maintainable as well whenever some significant change happens it does not create errors. There are many options to choose from, such as; django ORM, SQLalchemy, fuzzywuzzy, rapidfuzz, etc. In order to find the right one, we will have to see which ones meet our desired characteristics.

5. (UI Development) Users need an easy way to interact with the website through a clean, intuitive UI where they can review posts and post peer reviews themselves. Due to the fact that our website will be focused on user interaction, as the users are the ones who will be adding and searching for content, it is crucial that the website be user friendly. Users need to be able to easily figure out how to use the website, so making it intuitive

and easily navigable are high priorities. Not only that, but the user experience is significantly improved by a simple and aesthetically pleasing. These factors make UI development a key technical issue for our project.

In order to minimize any potential issues, we have identified potential solutions to all the above-mentioned technical challenges. In the next section, we will discuss the pros and cons of all potential solutions and analyze which will best fit our use cases. Once a conclusion has been reached on the best technological choice, we will explain the reasoning for the decision and provide testing results that verify the feasibility of the solution for our product.

Technology Analysis

1. (Backend database) We need a scalable and secure database for any number of users to make an account and for any additional website data, including but not limited to user posts.

Introduction:

Due to the sheer volume of scientific articles that are published every year and our client's wish to expand the Continuous Community Review Compendium up to at least 100,000 users, the backend database that is used for this product will need to be scalable and secure. The overall goal may not be to upload all of the articles themselves; instead, they will most likely have a link stored that goes to the page where they are published. However, the database will need to store all of the user account info, flags that have been put on articles, and the peer reviews left by users. This means that there will be a steadily growing amount of data that will need to be stored as more users join and create peer reviews for articles that can be accessed through the database. Furthermore, as the data steadily grows, our database system will need to be efficient with its querying and data fetching capabilities as well as easily being able to communicate with the front-end side of the system where users will be present.

Desired Characteristics:

1. Scalable
2. Secure
3. Ability to handle complex data queries
4. Efficient

The desired characteristics for the solution to this issue would be to have a database that is scalable, secure, and can quickly handle complex data queries for users. For scalability, it must be able to continually put new data into designated tables within the database without drastically slowing down and while maintaining data relationships. These data relationships will be in the form of peer reviews being linked to their specific article as well as the user who created them. Furthermore, flags to specify certain attributes of articles will be linked to the user who made them. In terms of security, personal data within the database will need to be stored as encrypted information that will then be decrypted by middleware algorithms and will also need to be able to communicate with the front-end using secure data transmission.

In order to be able to handle data queries quickly, the database will need to be able to take in possibly complex search parameters that will then be used to gather specific information from available data and then return it to the user. This could include a combination of specific article types, authors, years of publication, number of peer reviews, etc. All of these different parameters will need to be handled by the querying system as well as the database itself to process these requests and return the required data quickly.

Alternatives:

There are several types of databases that we are considering for the project, and they are as follows:

1. NoSQL (also known as Not Only SQL) is a database language that differs from the typical relational format and is best suited for handling diverse data structures that need to be stored in a “more natural and flexible way” [12]. This structure of database creation is extremely versatile and can support many different methods of storing information, such as in documents, keys, tables, and graphs.
2. PostgreSQL is an open source database language much like MySQL, but it provides increased data storage flexibility through extending the standard SQL language system [13]. It is an object-relational database where data is stored as properties within objects that are then related to each other.
3. MySQL is the classic open source relational database management system and is great for storing, managing, and organizing large amounts of data in a structured manner through tables and rows [15]. Everything in MySQL is stored in tables that then have relations to each other and can be queried to provide different information based on the query format.

Analysis:

During the analysis of these three different database types, we were focused on finding which one met our client’s needs, as well as being secure, scalable, able to handle complex queries, fast, and easy to use.

NoSQL provides a very flexible database system that can work with a large variety of data and does not have the limitations of regular relational-based systems. However, the open nature of the system makes it less secure when compared to PostgreSQL and MySQL and it is only BASE compliant by default [12]. Furthermore, due to the format of the data we will be using in the database, the open nature of NoSQL is not truly necessary for the project. On top of that, our client has decided that he wants to use an SQL-based database, so we have decided to avoid using NoSQL for this project.

PostgreSQL was developed as a new approach to the MySQL database language and has been in operation for almost 40 years. As mentioned above, instead of the normal table approach to storing data, PostgreSQL is object-based, and the data is stored as properties. This allows for more flexible data storage than MySQL regularly allows due to the objects being able to have a parent-child relationship and be very similar to how programming languages format their objects. It is even possible to “write code from different programming languages without recompiling your database” [14]. On top of this, PostgreSQL is ACID-compliant at its core. This means that the database will make sure it stays in a valid state even when errors occur by not allowing any modifications to occur in existing data until the error is resolved [15]. Also, besides being ACID compliant, PostgreSQL is able to support encrypted information transfer as well as storage. However, one downside of PostgreSQL is that its added ability for more flexible data storage and querying makes it have a steeper learning curve as compared to other database languages and has a limited tool set for beginner users.

MySQL is a table-based, open-source, relational database management system. It uses tables to store the required data for the database that can then have relations through foreign keys to other tables for data referencing and querying. MySQL is especially well-suited to reading from large amounts of data and handling fast, complex user queries, as we will be doing with this project. Furthermore, MySQL supports encrypted information storage and transfer which will allow us to keep user information secure and private. Also, with an added module, MySQL can be ACID-compliant just like PostgreSQL [15]. In addition, MySQL is great for prototyping systems as well as providing extensive database creation support and documentation, providing an easier entry point for developers.

The table below goes over our rankings for each database model/language based on our criteria (where 5 = very good, 1 = poor fit).

Alternative	Scalability	Query Ability	Secureness	Efficient
NoSQL	5	4	3	4
PostgreSQL	5	5	5	4
MySQL	5	5	4	5

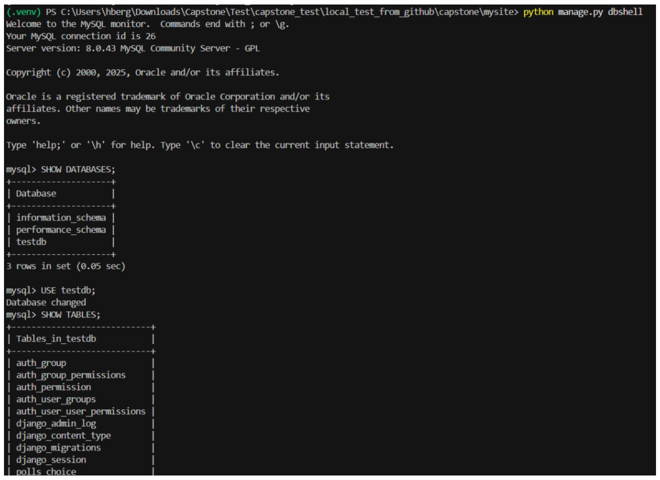
Chosen Approach:

Based on the above analysis, we have decided to go with MySQL for our database model of choice for this project. This is because our client wanted a SQL-based database for the compendium as well as MySQL is better suited for large amounts of data querying/reading, where PostgreSQL is best for writing data [15].

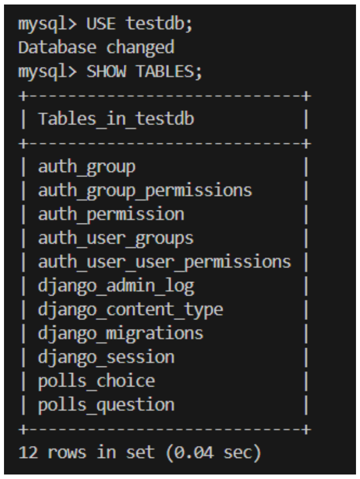
Out of the above technologies, it is obvious that PostgreSQL and MySQL are the two frontrunners for the database choice. We decided on MySQL due to its familiarity with us and the fact that it is easier to get a functional database up and running with it than with PostgreSQL. And, while it is true that PostgreSQL has more security in terms of data integrity, MySQL can easily catch up to this by using good data insertion techniques and possibly using some modules to enforce ACID compliance.

Proving Feasibility:

Implementing the database through MySQL is fully feasible for the product that we wish to build as can be seen below. The two images below show the terminal feed testing an initial MySQL database working with Django (the Python-based framework we plan to use with MySQL) and displaying a testing table within an existing database. This proves that it is possible to use MySQL to implement the database we wish to use for this product.



```
(.venv) PS C:\Users\hberg\Downloads\Capstone\Test\capstone_test\local_test_from_github\capstone\mysite> python manage.py dbshell
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 8.0.43 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| testdb             |
+--------------------+
3 rows in set (0.05 sec)

mysql> USE testdb;
Database changed
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_testdb           |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user_groups           |
| auth_user_user_permissions |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
| polls_choice               |
```



```
mysql> USE testdb;
Database changed
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_testdb           |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user_groups           |
| auth_user_user_permissions |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
| polls_choice               |
| polls_question             |
+----------------------------+
12 rows in set (0.04 sec)
```

Overall, the way our website connects to the database can be viewed in this repository. It can be found under the mysite/mysite folder, specifically in the settings.py file, under the Databases section.

2. (Backend database hosting) We need hosting for our database that will support concurrency for fetching and reading data from the database into the website, which will also support data input.

Introduction:

Our database hosting service must be able to support a substantial amount of concurrent users that will all be accessing information within the database, as well as sometimes writing to it itself when they create a new account, write a review, or are adding new scientific article entries. This is vital to the operation of the website, as there needs to be fast and efficient communication between the hosted database and the hosted website. In addition, the hosting service needs to be reliable in order to ensure that it does not go down while users are requesting data from the database.

On top of reliability, due to the constantly growing nature of the database itself, where articles and peer reviews will continually be added, the hosting service will need to provide the ability to be scaled over time to provide more storage while also maintaining its responsiveness. Furthermore, the database will need to back itself up in order to ensure that data is not lost if the site goes down or some sort of deletion incident occurs. This means that an up-to-date backup will need to be stored on the hosting platform itself or be transferred to an external location to where it is backed up to ensure data integrity and security.

Desired Characteristics:

1. Scalability
2. Affordable
3. Efficient
4. Reliable
5. Secure/Data integrity

To provide a solution to this issue, the platform that our database is hosted on will need to be scalable, affordable, efficient, reliable, and provide data integrity/security through back-ups. In terms of scalability, the host service will need to provide the ability to increase the amount of storage being used over time as more users and articles are added to the database. The hosting service must also be affordable with its pricing, especially during the early development and deployment phases when there are few users and a small budget. Then, the hosting platform will need to be efficient in its data retrieval and communication with the website hosting provider. Furthermore, the platform will need to be reliable and have little to no downtime so that users can experience a cohesive platform that provides them with all the information they need in a quick and reliable manner. If this does not happen, then users will just move on to a different service that gets them what they need faster or is more reliable in terms of uptime. Finally, the hosting platform will need to back up data in order to make sure that nothing is lost in the event of any new data corrupting the website or the hosting going down, and data being lost.

Alternatives:

For our backend hosting, there are a couple of hosting services that we considered:

1. AWS is a large-scale database hosting platform that provides ample configuration ability to its users with many different pricing options. This ability to manipulate and configure your database as needed allows for developers to manage their database however they need as it scales and grows.
2. DigitalOcean is great for speed and efficiency in terms of setting up a hosted database and comes at a lower price point. Furthermore, it is great for initial prototyping and products that are not being deployed on an enterprise or global level.
3. Microsoft Azure provides a very secure database that can be scaled up to handle users' needs as well as providing different pricing options. In addition, Azure has data centers around the world that provide reliable uptime and efficient information travel even when something goes wrong somewhere.

Analysis:

During our analysis of these three database hosting platforms, we were looking for the platform that best matched our needs as well as the clients for our desired characteristics of being scalable, affordable, efficient, reliable, and secure.

Starting off with AWS, they provide a large variety of hosting options as well as vast configuration ability for the database being hosted. On top of this, AWS is great for complex services while allowing for products to be scaled up towards global use as the need arises. However, this configuration ability causes the initial startup and setup of a hosted database to be quite complex and quite cumbersome due to having to figure out how exactly you want your database to operate as well as the resources it will require from AWS. This leads to AWS's pricing options, which while they are very configurable and can be fine-tuned to your needs, can increase in cost very quickly and cause you to spend more money than might be necessary due to them being usage based [16]. So, while AWS is great for large scale products, which we hope our project might eventually become, it requires a more complex and in-depth setup for startup projects trying to get off of the ground.

Moving on to DigitalOcean, they provide a variety of hosting options like AWS. On top of this, they provide the option for self-managed droplets that are completely configured by the developers or having a managed database service that automatically will backup the existing database as well as providing an easy configuration setup to get your product off the ground. In addition, the pricing points for DigitalOcean are much more streamlined and easier to understand when compared with AWS's usage-based pricing. Besides the more streamlined pricing, DigitalOcean is generally cheaper when compared to AWS due to its adding on additional charges for necessary features instead of the flat price that DigitalOcean provides. For instance, the most basic model of DigitalOcean's droplets start out at just $4 a month [16].

Finally, for Microsoft Azure, they provide the great security that comes with the Azure platform, as well as many security configuration options for the database. However, this addition leads to a higher price when compared to AWS & DigitalOcean. On top of this higher price, their pricing options are quite complicated and include payments for many different services (storage, backup storage, CPU series, etc.) [17]. Furthermore, the setup for an Azure-hosted database can be a bit complicated, as the documentation that Microsoft provides can be quite dense and hard to follow at times. But, it must be said that there are great scalability options that are laid out quite nicely in their documentation. Additionally, their 38 datacenters around the world ensure that the service maintains its reliability for the users as well as providing fast connections globally for if the product grows substantially [18].

The table below goes over our rankings for each database hosting platform based on our criteria (where 5 = very good, 1 = poor fit).

Alternative	Scalability	Affordability	Efficiency	Reliability	Secureness/ Data Integrity
AWS	5	4	4	4	4
DigitalOcean	4	5	5	4	4
Microsoft Azure	5	3	4	5	5

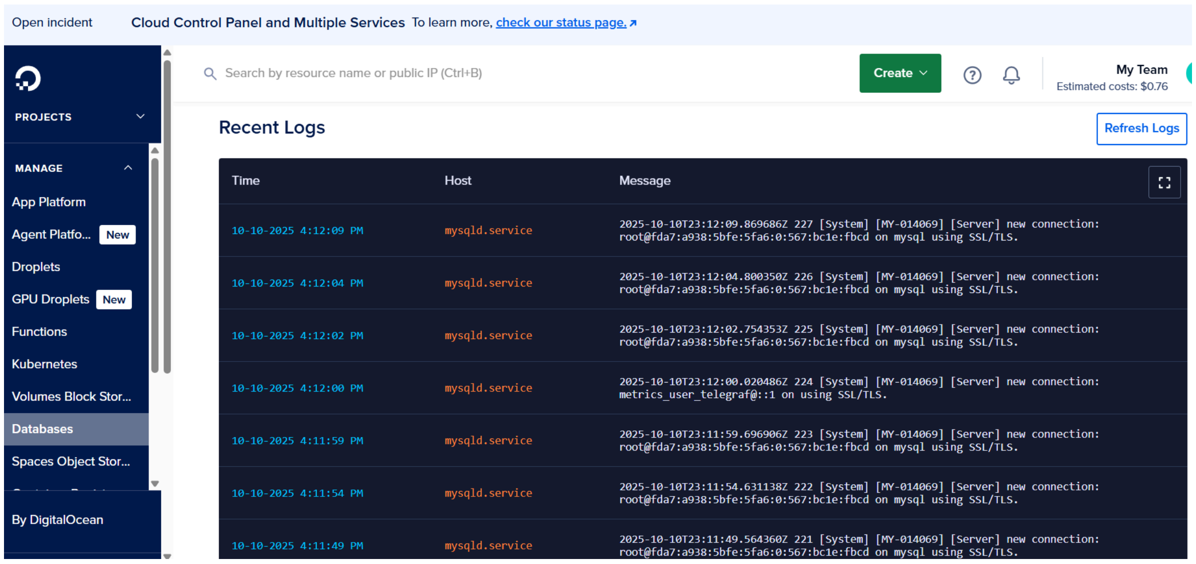
Chosen Approach:

Based on the above analysis, we have decided to use DigitalOcean for our database hosting platform. This is because it provides the best experience for new developers, as well as providing easy pricing options. Furthermore, DigitalOcean is great for providing the ability to prototype and demo products while they are in their initial stages of development and testing. Products can be deployed within minutes when you need to “move from idea to production quickly” as will be necessary during user testing with quick modification of database aspects/relationships possibly being required [16].

On top of its affordability and ease of use, DigitalOcean is reliable for the number of users that we wish to achieve with the initial product as well as providing automatic data backups with the managed database that we plan to use. And, while AWS and Microsoft Azure both provide their own advantages through scalability and security, respectively, DigitalOcean provides the best all-around capabilities for this project and its startup requirements.

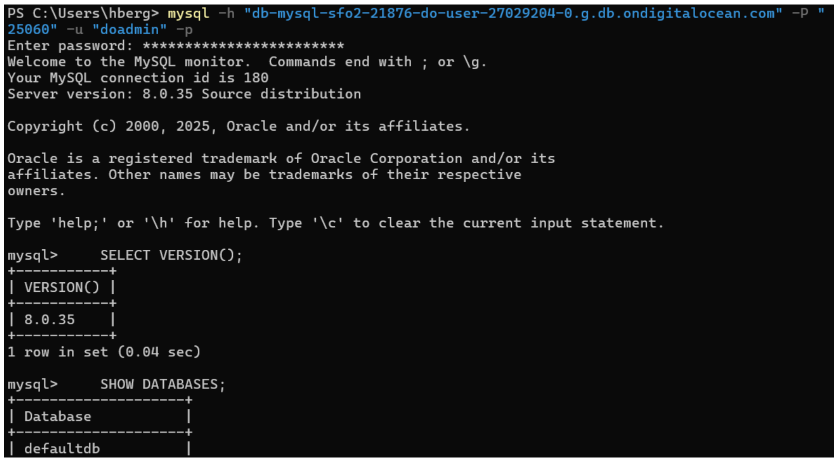
Proving Feasibility:

Implementing the database hosting through DigitalOcean is fully feasible within the scope of our project as shown in the two images below. They show the DigitalOcean Database service window logs with a terminal screenshot of basic MySQL commands with their responses, demonstrating that the hosted database is functional and able to be operated. This proves that DigitalOcean will work to host our product’s initial MySQL database.



Recent Logs

```
Time                    Host              Message
10-10-2025 4:12:09 PM   mysqld.service    2025-10-10T23:12:09.869686Z 227 [System] [MY-014069] [Server] new connection:
                                          root@fda7:a938:5bfe:5fa6:0:567:bc1e:fbcd on mysql using SSL/TLS.
10-10-2025 4:12:04 PM   mysqld.service    2025-10-10T23:12:04.800350Z 226 [System] [MY-014069] [Server] new connection:
                                          root@fda7:a938:5bfe:5fa6:0:567:bc1e:fbcd on mysql using SSL/TLS.
10-10-2025 4:12:02 PM   mysqld.service    2025-10-10T23:12:02.754353Z 225 [System] [MY-014069] [Server] new connection:
                                          root@fda7:a938:5bfe:5fa6:0:567:bc1e:fbcd on mysql using SSL/TLS.
10-10-2025 4:12:00 PM   mysqld.service    2025-10-10T23:12:00.020486Z 224 [System] [MY-014069] [Server] new connection:
                                          metrics_user_telegraf@::1 on using SSL/TLS.
10-10-2025 4:11:59 PM   mysqld.service    2025-10-10T23:11:59.696906Z 223 [System] [MY-014069] [Server] new connection:
                                          root@fda7:a938:5bfe:5fa6:0:567:bc1e:fbcd on mysql using SSL/TLS.
10-10-2025 4:11:54 PM   mysqld.service    2025-10-10T23:11:54.631138Z 222 [System] [MY-014069] [Server] new connection:
                                          root@fda7:a938:5bfe:5fa6:0:567:bc1e:fbcd on mysql using SSL/TLS.
10-10-2025 4:11:49 PM   mysqld.service    2025-10-10T23:11:49.564360Z 221 [System] [MY-014069] [Server] new connection:
                                          root@fda7:a938:5bfe:5fa6:0:567:bc1e:fbcd on mysql using SSL/TLS.
```



```
PS C:\Users\hberg> mysql -h "db-mysql-sfo2-21876-do-user-27029204-0.g.db.ondigitalocean.com" -P "
25060" -u "doadmin" -p
Enter password: ************************
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 180
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>     SELECT VERSION();
+-----------+
| VERSION() |
+-----------+
| 8.0.35    |
+-----------+
1 row in set (0.04 sec)

mysql>     SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| defaultdb          |
```

3. (User authentication) We need a secure, verifiable user authentication system on our website.

Introduction:

The overuse of artificial intelligence (AI) tools in research and writing has introduced new challenges to the integrity of the scientific publishing system. The credibility of legitimate scholarly work can be compromised by inaccurate, AI-generated papers, reviews, and user accounts. To ensure that our compendium remains a reliable, human-authored resource, one of our primary goals is to control AI-generated content. This means that automated and AI accounts must not be allowed on the site, where they could submit fabricated content and compromise the accuracy of the compendium.

The solution to this issue is to implement a secure and verifiable system for user authentication. Specifically, there must be methods to detect bots and prevent them from creating accounts. This process must balance strong security and accessibility for genuine users. If there are excessive or overly complex authentication steps, intended users might be frustrated or discouraged and not make an account.

There are several technologies that can be used to address these challenges, such as CAPTCHA services and authentication mechanisms. CAPTCHA services can be used to differentiate between automated interactions and humans. Authentication systems can confirm the legitimacy of a user once they have created an account. These can be implemented through a number of methods, such as multi-factor authentication and email authorization. The following sections will explore several potential technologies that could be used for these purposes.

Desired Characteristics:

1. Security
2. Verifiability
3. Reliability
4. Usability
5. Maintainability
6. Scalability

The desired characteristics for the solution to this issue are an account creation process that is secure, verifiable, and not overly complex. In terms of security, the system must screen out automated interactions that indicate AI or bot activity to ensure that no automated accounts are created on the platform. For verification, accounts must be confirmable once they have been created to prevent fake registrations, as well as enable stronger security features, such as account recovery. Implementing features that promote a high level of security is a key part of solving this issue.

It is equally important that the account creation process is simple and user-friendly, as adding too many steps or layers of verification might cause a user to become frustrated or discouraged from completing the process. This is not a technology in and of itself, but is a guiding principle for selecting the appropriate tools to incorporate in conjunction with one another. For instance, a 10-step reCAPTCHA process might be highly secure, but it would not be suitable for our project due to its excessive complexity. For our solution, it is also important that any technology that we implement is easily maintainable and scalable. Overall, an authentication system that combines multiple technologies to achieve these characteristics is essential to the success of the product. Options for these technologies will be evaluated below based on these criteria.

Alternatives:

Our team considered several technologies that could promote the security and verifiability of our compendium. For CAPTCHA screening, we explored reCAPTCHA and Django Simple CAPTCHA. For user authentication, we considered multi-factor authentication using PyOTP and Google Authenticator, as well as email authentication with django-allauth.

1. reCAPTCHA is a free-to-low-cost Google service that differentiates between human and bot interaction by “analyzing user behavior and other factors to determine if an action is being performed by a human” [3]. In some cases, this includes presenting a challenge for the user to complete. There are multiple versions currently available, including reCAPTCHA v2, which displays the “I am not a robot” checkbox, and reCAPTCHA v3, which operates invisibly in the background to assess user legitimacy.
2. Django Simple CAPTCHA is a Python library that provides “extremely simple, yet highly customizable” CAPTCHA functionality for Django applications [5]. It offers multiple types of challenges, such as simple math problems and random character inputs, which developers tailor to the needs of a project. It is simple and flexible, which makes it an accessible option for smaller-scale projects.
3. Multi-factor authentication (MFA) requires users to verify their identity using at least two forms of authentication. Using PyOTP alongside Google Authenticator, we could allow users to connect their accounts to an authenticator app that generates a one-time code for login verification, such as Google Authenticator. This helps to ensure the site remains secure, since “while important, usernames and passwords are vulnerable to brute force attacks and can be stolen by third parties.” [8]. PyOTP is the library used to connect an account to an authenticator app. This adds an extra layer of security that ensures only legitimate users can access the platform.
4. Email authentication, for which we would use django-allauth, is a process in which a verification email is sent to confirm that the user attempting to log in or register has access to the email associated with the account. This can help prevent fake registrations and "proactively detect and prevent fraudulent activities," as well as support the account recovery process [1]. Using django-allauth, the built-in Django package for authentication, email verification, and password resets, we can implement this functionality securely and with minimal overhead.

Analysis:

In our analysis, we considered the desired characteristics of security, verifiability, and simplicity for users as overarching considerations. We also examined more specific factors such as scalability, maintainability, and accuracy in our assessment. Based on these criteria, we analyzed our alternative technologies.

Google's reCAPTCHA is a CAPTCHA service that differentiates between human and automated interaction, and "by implementing reCAPTCHA, businesses can gain proactive protection against malicious bots, ensuring a more secure and trustworthy online environment for their users" [3]. Standard reCAPTCHA is free for up to 10,000 assessments per month, and offers a paid tier at $8 for up to 100,000 assessments per month, remaining cost-effective even at scale. Although this is a Google service, reCAPTCHA works on other browsers without issue. There are two main versions that are commonly used: reAPTCHA v2, featuring the "I am a robot" checkbox, and reCAPTCHA v3, which runs invisibly in the background. reCPATCHA v3 offers a seamless user experience and has customizable thresholds for suspicious behaviors. However, it requires careful tuning and regular maintenance to stay accurate. It also collects more user data, raising potential privacy concerns. reCAPTCHA v2 is less seamless, but it offers a higher level of security and requires less maintenance, and it provides audio options for accessibility. Overall, reCAPTCHA, specifically v2, is reliable, effective, low-maintenance, affordable, and scalable, though it slightly interrupts the user's flow.

Django Simple CAPTCHA (DSC) is a free, open-source Python library that provides flexible and customizable CAPTCHA functionality for Django applications. It offers multiple "custom challenges (e.g. random chars, simple math, dictionary word, ...)", which can be adapted to suit the needs of the project [7]. DSC is easy to implement and maintain, and it includes audio options for accessibility. However, it is currently restricted to image-based CAPTCHAs, and as a result is less secure and easier for scammers to bypass compared to reCAPTCHA. Overall, DSC is well-suited to projects that are looking for a service that is free, customizable, and simple to both implement and maintain, but provides weaker protection against advanced automation.

Multi-factor authentication (MFA) with PyOTP and Google Authenticator adds an extra layer of verification during login by requiring users to enter a time-sensitive code from an authenticator app. This approach is free, very easily scalable, and provides additional security since "rather than just asking for a username and password, MFA requires one or more additional verification factors, which decreases the likelihood of a successful cyber attack" [8]. However, it introduces maintenance and security concerns associated with MFA, including secure storage of PyOTP user secrets, HTTPS protection during code exchange, and account recovery challenges. MFA also creates friction for users, as it is a relatively long and involved process. Overall, MFA is an effective, low-cost, scalable way to boost account security, but it requires maintenance and reduces convenience for users.

Email authentication with django-allauth uses verification emails to confirm that the individual registering or logging in has access to the associated email account. Django-allauth is a free, open-source library that is well-established, secure, low-maintenance, and easy to implement into a Django project, as it is known for being a "secure, well-integrated, reusable authentication solution for the Django framework" [7]. This service is also highly scalable, with a reputation for being able to handle high-traffic websites. Email verification does add a small step to the registration process, but it provides significant benefits in terms of filtering out fake accounts and facilitating account recovery. Overall, email authentication with Django-allauth is a scalable, maintainable, and secure way to provide user accounts with higher security, but it does create a minor hurdle for new users.

Each of these technologies presents pros and cons in terms of security, verification, and usability. The next section will synthesize these findings and outline our final decisions based on this analysis.

Based on this analysis, each technology has been ranked in the table below, which ranks each alternative according to the desired criteria (where 5 = very good, 1 = poor fit).
The evaluation categories are defined as:

Scalability - How well the technology handles growth in traffic and users, with consideration for cost at scale.
Maintainability - The ease of long-term upkeep, updates, and configuration.
Reliability - The consistency of the tool's performance for its intended purpose.
Security - The strength of the protection it provides is against automation or malicious activity.
Usability - How seamless or simple the process feels for legitimate users.

Alternatives	Scalability	Maintainability	Reliability	Security	Verifiability	Usability
reCAPTCHA	5	5	4	4	5	3
Django Simple CAPTCHA	3	3	3	3	3	2
Multi-Factor Authentication (Google Authenticator & PyOTP)	5	3	4	5	5	1
Django-allauth	5	5	5	4	5	4

Note: These are based on our collective research, small tests, user feedback, and our application-specific needs considerations.

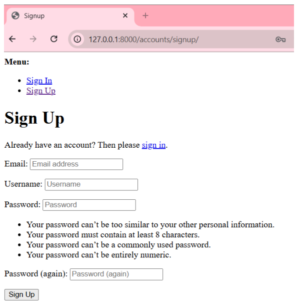
Chosen Approach:

Based on the analysis and table above, the best option for CAPTCHA functionality is reCAPTCHA, while the best option for user authentication is Django-allauth. For implementing a CAPTCHA service, reCAPTCHA outperforms Django Simple Captcha in every category. It is more scalable while remaining cost-effective, requires less maintenance, is more accurate, provides stronger protection against more advanced automated attacks, and offers a smoother user experience, as checking a box is faster and easier for users than solving a math problem. It is also extremely reliable for our use case, as it "leverages a sophisticated and adaptable risk analysis engine to shield against automated software, thwarting abusive activities within an organization’s website and mobile apps” [3]. Overall, reCAPTCHA is far better suited to the needs of our project than Django Simple CAPTCHA across all criteria.

For user authentication, the decision is not quite as clear-cut. While both Django-allauth and MFA are equally scalable, Django-allauth is ultimately the better option due to its balance of characteristics. Although MFA provides slightly better security, Django-allauth offers greater reliability, significantly better maintainability, and superior usability. Weighed against each other, these factors make Django-allauth the best choice for implementing user authentication within our project.

Proving feasibility:

Implementing user authentication with Django-allauth is fully achievable within the scope of our project. Initial testing showed that the integration process is well-documented, simple, and immediately effective. The screenshots below illustrate a successful sign-up flow on a Django test server configured to require email authentication through Django-allauth.



Menu:

- Sign In
- Sign Up

Sign Up

Already have an account? Then please sign in.

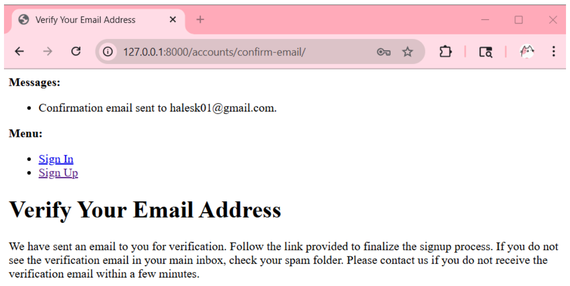
Email: Email address

Username: Username

Password: Password

- Your password can’t be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can’t be a commonly used password.
- Your password can’t be entirely numeric.

Password (again): Password (again)


Verify Your Email Address
127.0.0.1:8000/accounts/confirm-email/

Messages:

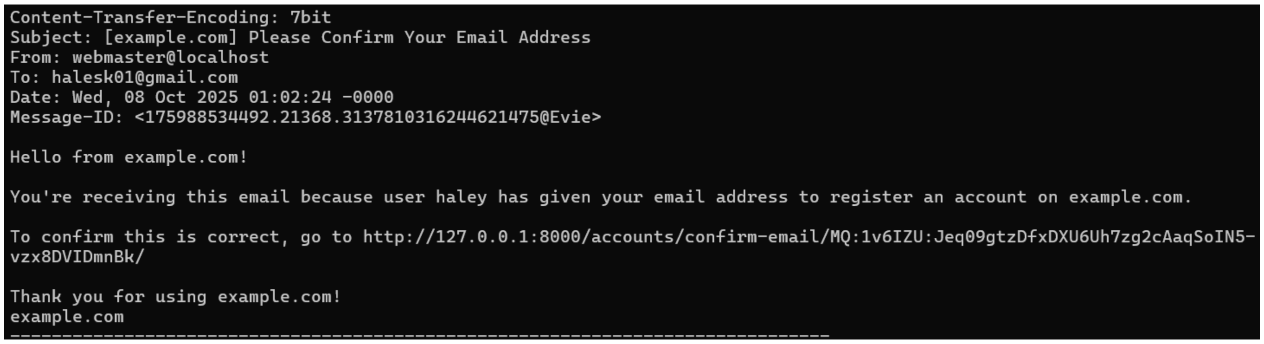
- Confirmation email sent to halesk01@gmail.com.

Menu:

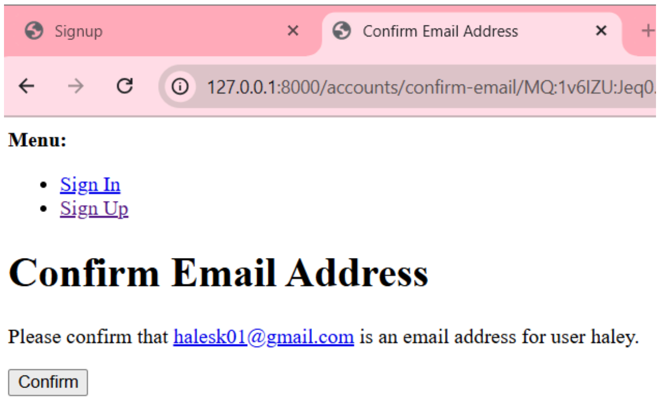
- Sign In
- Sign Up

Verify Your Email Address

We have sent an email to you for verification. Follow the link provided to finalize the signup process. If you do not see the verification email in your main inbox, check your spam folder. Please contact us if you do not receive the verification email within a few minutes.



```
Content-Transfer-Encoding: 7bit
Subject: [example.com] Please Confirm Your Email Address
From: webmaster@localhost
To: halesk01@gmail.com
Date: Wed, 08 Oct 2025 01:02:24 -0000
Message-ID: <175988534492.21368.3137810316244621475@Evie>

Hello from example.com!

You're receiving this email because user haley has given your email address to register an account on example.com.

To confirm this is correct, go to http://127.0.0.1:8000/accounts/confirm-email/MQ:1v6IZU:Jeq09gtzDfxDXU6Uh7zg2cAaqSoIN5-
vzx8DVIDmnBk/

Thank you for using example.com!
example.com
-------------------------------------------------------------------------------
```

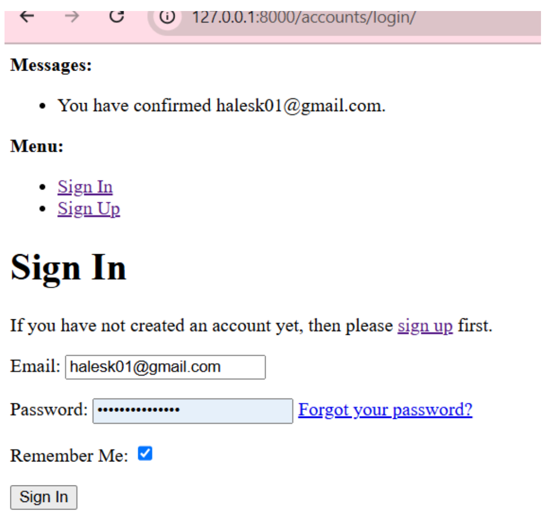


Menu:

- Sign In
- Sign Up

Confirm Email Address

Please confirm that halesk01@gmail.com is an email address for user haley.



Messages:

- You have confirmed halesk01@gmail.com.

Menu:

- Sign In
- Sign Up

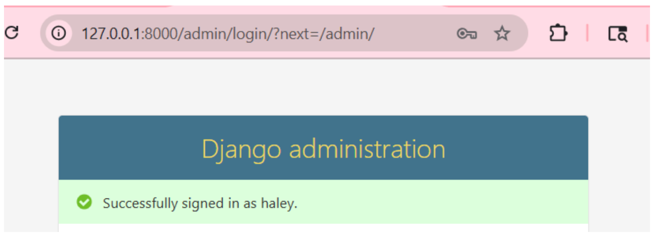
Sign In

If you have not created an account yet, then please sign up first.

Email: halesk01@gmail.com

Password: ••••••••••••••• Forgot your password?

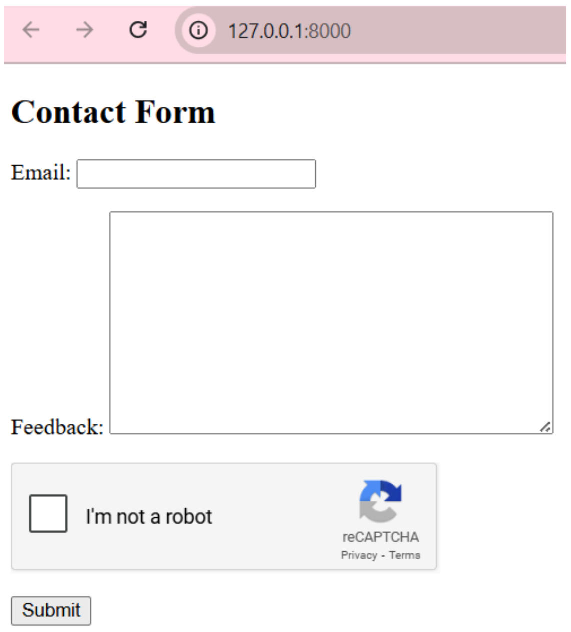
Remember Me: ✓


127.0.0.1:8000/admin/login/?next=/admin/

Django administration

Successfully signed in as haley.

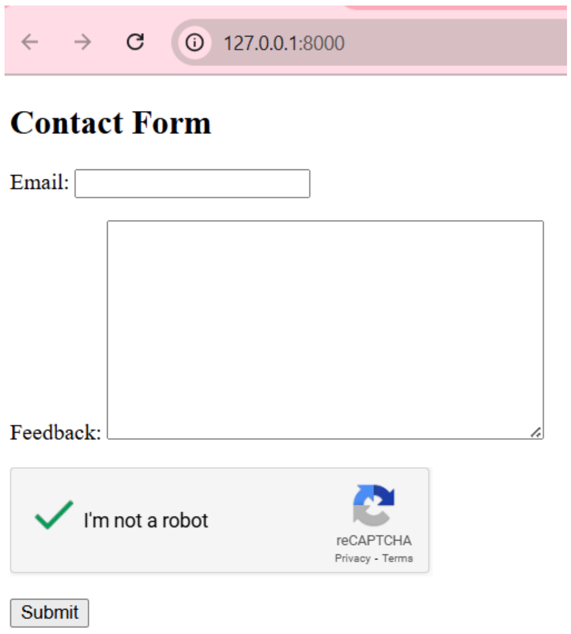
Implementation of Google reCAPTCHA v2 in Django is well-documented and manageable for this project [2]. The screenshots below show reCAPTCHA v2 integrated into the same test Django server, confirming that it is compatible.


127.0.0.1:8000

Contact Form

Email:

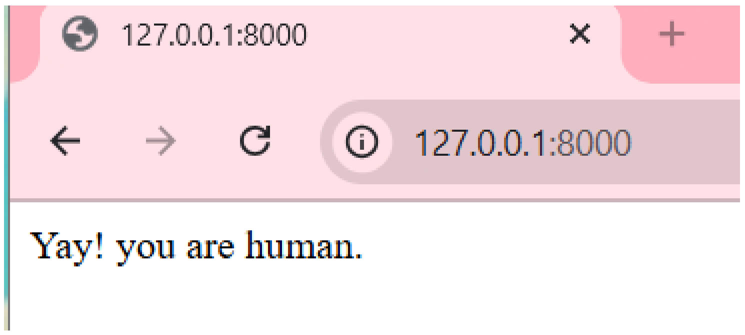
Feedback:


127.0.0.1:8000

Contact Form

Email:

Feedback:


127.0.0.1:8000

Yay! you are human.

The code used to make the above screenshots can be found in this repository. It can be found under the mysite/allauth_manager.

4. (Fuzzy search) We need a fast lookup system for searching documents and authors. It must also offer recommendations, similar searches, and account for misspellings.

Introduction:

Since we are going to be working with a large database of peer review documents, it will be hard for the users to manually search up any specific documents they would like to see. In order to address this challenge, we will need to implement a fast lookup system that is able to accurately search across multiple documents. This is also known as an object-relational mapper. ORMS can serve as an essential tool in this process by providing an abstraction layer that simplifies database manipulation and querying. It allows developers to interact with database entities as objects while using the functional power of SQL.

In addition to having basic search capabilities additional features that will be needed are recommendations, similar searches, and account for misspellings. All of these will enhance the user's experience with the search function. It will make their search for their specific document much easier and smoother. This will require implementing mechanisms to analyze document content and metadata, and enabling the system to suggest related documents. We also need to keep in mind user errors, such as typos and misspelling and incorporate fuzzy matching algorithms that can interpret user input and return relevant results despite the user's inaccuracies. There are multiple lookup systems with these features that we can implement into our project and solve the many issues said; one of them is django's built-in ORM and features.

Desired Characteristics:

1. Scalability
2. Maintainability
3. Abstraction
4. Performance
5. Ease of Use

Desired characteristics for this search up function is to have it fast and responsive. Users expect instant feedback when searching a database, so the search algorithm should be optimized to handle massive queries efficiently. Accuracy is also equally as important, the function must return correct and relevant results to the user's input in order to maintain the trust and validity the user has to this system.

The search function should also be easy to learn and utilize, with clear comprehensive documentation. This is to ensure that any new members or collaborators that join the project can quickly understand what they are dealing with and how to tackle any errors. This is essential as the team needs to be up to date about what they are working on and be able to maintain a large database. This search up function should facilitate easy adaptation to accommodate the scalability of this project.

Maintainability is a huge and important factor to consider for ORMs. This is to ensure that any code changes or database interaction does not lead to unfortunate errors. All of this will help maintain the scalability of the search up functionality, which is another part to keep into great consideration. Despite the scale, searching should be effortless, seamless, and fast. It should be able to keep up with the ever growing amount of documents and data efficiently.

Alternatives:

1. SQLAlchemy is a powerful python SQL toolkit and Object-Relational Mapping library. Provides well-designed APIs for connection to relational databases, schema management, and flexible querying capabilities. It is highly customizable and flexible, supporting both ORM and core SQL expression language. SQLAlchemy also has extensive documentation due to its large community. Many other database backends are supported including MySQL, SQLite, Oracle, and PostgreSQL. Despite its flexibility, it has a steep learning curve and is more verbose for simple use cases which makes it for people who are not experienced with this type of stuff. SQLAlchemy also has great stability and is often used in many large-scale production environments. This ORM is open-source and free to use, so we will have no worries about costs on this part of the project.
2. FuzzyWuzzy is another form of fuzzy string matching based primarily on Levenshtein distance calculations. It is useful to compute similarity scores between strings, find similar matches/approximate searching when using the search bar. FuzzyWuzzy is easy to use with a simple APiI and provides multiple ratio calculation methods. It can also be helpful in scenarios where exact matching is impossible due to typos, formatting, or abbreviations. Although it has many upsides, many of the cons include not being optimized for large datasets, it has to assume the input data is fairly clean so there is no built-in profiling or data cleansing.
3. Django haystack comes part with django applications and provides an abstraction layer over various search backends. Haystack provides a backend-agnostic nature, allowing developers to switch search engines without rewriting their codebase. It also simplifies the implementation of full-text search and offers flexible query-building which is very beneficial for projects requiring these complex search functions. However, it may lose some performance optimizations specific to individual search engines and can be less than ideal for high volume data scenarios.
4. Django's built-in ORM is another option for an ORM. This ORMG greatly simplifies database interactions by allowing developers to work with python objects instead of writing complex SQL queries. Through this, it imports developer productivity, makes code more readable, and reduces and common errors. It supports multiple database backends without requiring changes to the code. There is a learning curve for its features and debugging has proven to be challenging. Despite this it appears to be a strong choice for maintainable, consistent data access.
5. Another option for fuzzy string matching is rapidfuzz. It is a fast and efficient python library which offers many advantages including superior speed. Speed is achieved through optimized C++ backend, making it roughly 40% faster in processing comparisons. It is also great in mult-threaded environments with minimal performance loss. It is also memory efficient, which is good for handling large datasets. On the downside, its accuracy can depend on the language or error types processed. It also might require more flexible or domain-specific approaches not fully covered by its algorithms.

Analysis:

In our analysis we considered the desired characteristics of our ORM, which is scalability, maintainability, powerful abstraction, performance, and ease of use.

SQLAlchemy was a good choice in terms of being a powerful ORM and SQL toolkit. It supports synchronous and asynchronous workflows, deep customization and strong database compatibility. It is very scalable amongst large datasets and has power abstractions of SQL expression language. However, it requires a lot of time to master, meaning that if any person joined the team working with this ORM without prior knowledge, it will be very hard for them to learn and utilize the system.

Django's Built-in ORM was seen best fit for typical web applications, with decent power abstraction and performance. It is also simple, you do not need to be an SQL expert, but instead just know basic python skills. [21] The maintainability was incredible as it allows for automatic data migrations, includes an admin interface, and a tight framework integration aid maintenance. It also beats SQLAlchemy in terms of ease of use as many people have worked with this intuitive API before.

In terms of fuzzy matching functions, fuzzywuzzy did not meet the requirements. It struggles with very large datasets due to pairwise comparison costs, and has very little power abstraction due to focus narrowly on string similarity with few features beyond scoring. Fuzzywuzzy is also slower than other alternatives like Rapidfuzz, because it is not using a pure python implementation. The libraries that rapidfuzz produces employ techniques like SIMD instructions to accelerate string comparisons, making datachunking able to break down large datasets into smaller manageable chunks [22]. One thing that it has going for it is the ease of use, it's well-documented and straightforward. Rapidfuzz meets the requirements of performance, scalability, and maintainability, with its high-performance string match library and offering a fast and memory-efficient alternative. There are no extended matching workflows, but the API is straightforward for users. Rapidfuzz is a strong choice for performance-critical fuzzy match tasks and able to handle large databases.

Django haystack is a high-level python web framework that matches many of the criteria needed for this ORM search function. It can scale well with caching, database replicas, and supports large deployments. It also supports faceting, highlight, spatial search, and many other features [23]. It is maintainable with strong conventions and integrated admin functions, while also being easy to use for beginners and rapid development. Power abstraction and performance are pretty decent and are sufficient for a strong ecosystem integration.

Based on the previous criteria listed, here how each of the technologies stand on a 1-5 scale (where 5 = very good, 1 = poor fit).

Alternatives	Scalability	Maintainability	Abstraction	Performance	Ease of Use
SQLAlchemy	5	4	5	3	2
FuzzyWuzzy	2	3	2	2	4
Django Haystack	4	5	3	3	5

Django ORM	3	5	3	3	5
RapidFuzz	4	4	2	5	4

Chosen approach:

Our chosen approach is to use the built-in django ORM for the use of general querying. This ORM has been proven to be very flexible with growing datasets and provide basic filtering and querying within these database fields. It is powerful yet straightforward for general querying. It is backed up with the strong integration with the django framework. It has greater flexibility and supports growing datasets which will be essential for our project.

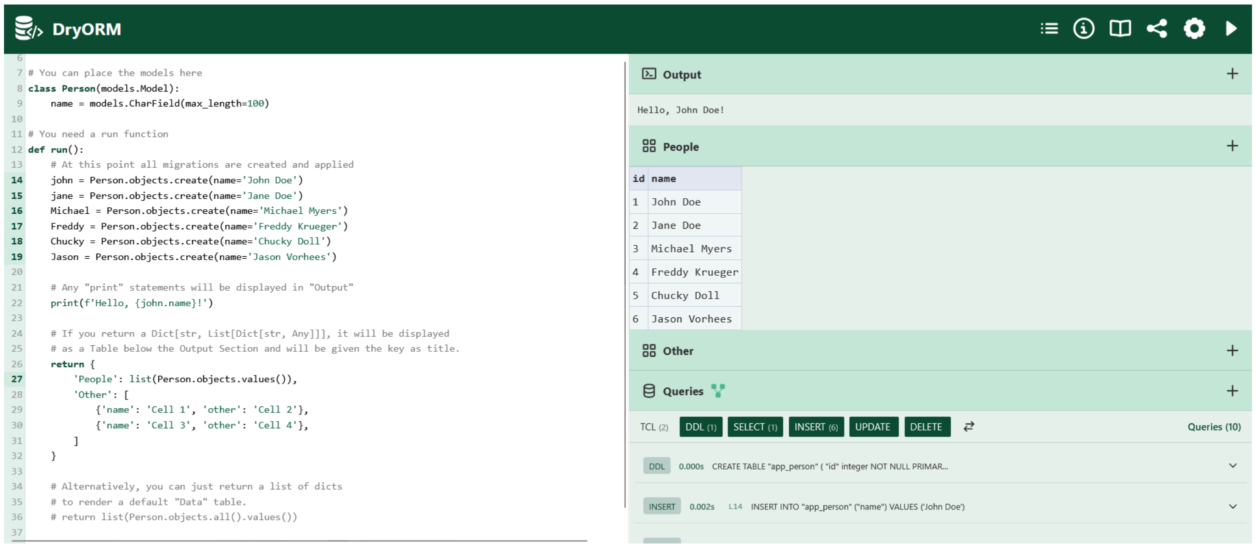
We will also use the rapidfuzz for fuzzing matching of search terms. This allows for a more general search whenever the user searches for a specific document or author, making this system more detailed and have more features. Rapidfuzz's speed and memory efficiency are crucial for maintaining responsiveness as the dataset scales, making the search system's capabilities better.

Lastly, we will also have django haystack, which is django's built-in autocomplete search feature allowing for time efficiency and smooth responsiveness for anybody searching through the database. This autocomplete functionality enhances user experience and reduces search time, making database navigation a lot easier to go through and more efficient.

Django is python based, which is the coding language primarily used for this project, allowing for all team members to easily navigate a database with previous knowledge on python. It is also very maintainable if we ever want to switch different large databases, such as MySQL, or PostgreSQL. We can also optimize to minimize any performance issues and allow for a responsive UI and save time for the user.

Proving feasibility:

Django ORM has been proven to work well with big datasets and can scale well with them. This ORM is fully compatible with the size of our project and easy to maintain for future use. The screenshots show the possible potential it has with our project.



```python

# You can place the models here
class Person(models.Model):
    name = models.CharField(max_length=100)

# You need a run function
def run():
    # At this point all migrations are created and applied
    john = Person.objects.create(name='John Doe')
    jane = Person.objects.create(name='Jane Doe')
    Michael = Person.objects.create(name='Michael Myers')
    Freddy = Person.objects.create(name='Freddy Krueger')
    Chucky = Person.objects.create(name='Chucky Doll')
    Jason = Person.objects.create(name='Jason Vorhees')

    # Any "print" statements will be displayed in "Output"
    print(f'Hello, {john.name}!')

    # If you return a Dict[str, List[Dict[str, Any]]], it will be displayed
    # as a Table below the Output Section and will be given the key as title.
    return {
        'People': list(Person.objects.values()),
        'Other': [
            {'name': 'Cell 1', 'other': 'Cell 2'},
            {'name': 'Cell 3', 'other': 'Cell 4'},
        ]
    }

    # Alternatively, you can just return a list of dicts
    # to render a default "Data" table.
    # return list(Person.objects.all().values())

```

```
Hello, John Doe!
```

id	name
1	John Doe
2	Jane Doe
3	Michael Myers
4	Freddy Krueger
5	Chucky Doll
6	Jason Vorhees

```
DDL     0.000s       CREATE TABLE "app_person" ( "id" integer NOT NULL PRIMAR...
INSERT  0.002s  L14  INSERT INTO "app_person" ("name") VALUES ('John Doe')
```

5. (Website hosting) We need hosting for our website that will support high concurrency for both posting and reading from the website.

Introduction:

Our site must be able to support a high volume of total real users as well as a high concurrency of users performing activities such as posting, editing, and reading reviews. Our system will offer peer review of scholarly papers in the public domain and be referenced to the publications that engage them. With such an academic focus and high usage potential by scholars and researchers, the system must remain responsive and predictable at high loads.

This requirement provides a number of technological challenges. The site must support large volumes of read operations (as users browse and query reviews) and write operations (as users add or modify reviews). We also need to be able to scale elastically, either vertically by increasing the capacity of a single machine or horizontally by distributing traffic over multiple machines. Also, the infrastructure needs to be reliable and cost-effective.

For our project, we chose to use Django as our web framework. There are multiple reasons for using this web framework, but it primarily came down to half of our team being familiar with it and it also being Python-based, making it easy to learn. Additionally, we trust its ability to scale, considering that sites like Spotify, Instagram, and even YouTube use it, as well as through our own research conducted. It additionally has a large online community, making it easy to get information and potentially help. For ease of setup and initial testing, we will serve our Django application on PythonAnywhere and on a DigitalOcean App for production-oriented test runs. We also researched Hostwinds as a second hosting option, but determined it would not work for our production needs. The following sections describe the features we require, the choices we researched, how we tested them, and why DigitalOcean (specifically, a Digital Ocean App) became our primary deployment platform.

Desired Characteristics:

1. Scalability
2. Performance
3. Flexibility
4. Reliability
5. Cost
6. Support

An ideal hosting solution for our project would provide a combination of performance, scalability, reliability, cost-effectiveness, and adaptability. These features are crucial because the reputation of the system hinges on reliable availability and responsiveness, particularly for educational users looking at time-critical research data.

Scalability is key. The hosting environment must accommodate expected growth in user traffic and stored data without a complete redesign of the infrastructure. Vertical scaling (increasing CPU, memory, or storage) and horizontal scaling (across multiple servers or containers) should be feasible with minimal disruption.

Performance matters too. The site should exhibit minimal latency for writes and reads, even under growing concurrent usage. If response degrades under load, the user interface would be slow, discouraging usage by students and researchers.

We also require control and flexibility over the host environment. Django applications will often rely on specific server configurations, e.g., Gunicorn or ASGI setups, and may then add background processing or caching. The ability to install required packages, configure modifications, and manually manage services is crucial for long-term sustainability.

Reliability and high availability are equally important. As a research instrument, downtime gives rise to distrust as well as disruption of research workflows. The hosting platform must thus offer uptime stability and recovery abilities in case of failure.

Finally, we considered cost and ease of maintenance. Since this is budgeted work and under academic constraints, hosting must be inexpensive and manageable without hiring full-time system administrators. Nevertheless, the platform must also enjoy a thriving community as well as good documentation so that our community can tap into resources and resolve issues in a timely manner.

Alternatives:

We considered three main hosting options that may be suitable for these requirements: DigitalOcean, PythonAnywhere, and Hostwinds.

1. DigitalOcean is a cloud hosting service that offers fuss-free App hosting. It has a nice mix of scalability and maintainability, making it easy for our client to use for a long-term project with little technical background. We discovered it via developer forums, Django deployment guides, and word of mouth. DigitalOcean has existed since 2011 and is widely used by startups, small businesses, and individual developers to host web applications, including Django applications.
2. PythonAnywhere is a Platform-as-a-Service (PaaS) specifically for Python applications. It conceals most of the server administration, providing a managed environment that simplifies deployment. We selected PythonAnywhere for initial testing because it supports Django and provides a simple environment in which to experiment without extreme system configuration.
3. Hostwinds is a general host provider of shared, VPS, and cloud hosting. It has competitive pricing and high uptime guarantees. We considered it after finding it on various hosting comparison charts and forums, considering the cheapest DigitalOcean alternatives. But as stated later, users' reported performance and reliability problems made it an unacceptable option for our use.

Analysis:

During our evaluation, we checked how each hosting provider performs in the following criteria: scalability, performance, control and flexibility, reliability, cost and maintenance, and maturity.

DigitalOcean provides good solutions for vertical and horizontal scalability. The Apps are easy to resize, and developers can have a group of servers behind load balancers or use DigitalOcean's managed database and storage solutions. PythonAnywhere offers limited scaling; customers must purchase more costly plans to receive additional resources, and its setup limits advanced scaling. However, it is great for testing a dynamic Django website for free and is very easy to set up. Hostwinds also features scalable VPS and cloud hosting plans, but based on customer reports, heavy load performance is not always consistent, especially at lower plans. Some reports also show difficulty with their customer service [11].

For performance, DigitalOcean consistently gives strong results, with most developers getting rapid response times and little lag for web applications. PythonAnywhere performs reasonably well for light or moderate use, but since it is a shared platform, large concurrent traffic can lead to lags. Hostwinds' performance results are not as consistent. Third-party testing [9] shows slower average response times compared to the competition, particularly when under heavy load conditions.

In terms of control and flexibility, DigitalOcean provides complete root access and permits total web server and operating system configuration. PythonAnywhere's managed environment is designed to be restrictive; certain administrative tasks cannot be done, and installation of random software is not permitted. Hostwinds gives a bit more control than

PythonAnywhere, but tends to be without the documentation and user community support found with DigitalOcean.

As for high availability and reliability, DigitalOcean's uptime and regional redundancy are strong, supported by snapshot and recovery [10]. PythonAnywhere enjoys good uptime for regular use, but is less resilient to spikes in load due to shared infrastructure. Hostwinds boasts 99.9999% uptime, but external monitoring recorded actual uptime at approximately 99.89%, which equates to a few minutes of downtime each week [9 & 11].

For maintenance and price, DigitalOcean maintains solid pricing and full administrative access for a constant, upfront monthly fee [10]. PythonAnywhere cuts down on maintenance by carrying out system administration, so it's ideal for small-scale testing or proof-of-concept. Its base version is also free. Hostwinds appears cheap initially, but may end up with unabsorbed expenses in the form of add-ons and scaling requirements.

Finally, in terms of maturity and documentation, DigitalOcean has a big community of developers and extensive tutorials, particularly for Django deployments. PythonAnywhere also supports good documentation and an active Python-focused community. Hostwinds, however, has weak community support and more fragmented documentation, making it harder to troubleshoot and optimize. [11]

Why Not Hostwinds?

While Hostwinds positions itself as an affordable alternative to the larger cloud providers, there are a number of factors that make it unacceptable for Jack's Peer Pages. Third-party testing indicates that despite its advertised uptime of 99.9999%, the mean monitored uptime is closer to 99.89%, which amounts to around 19 minutes of downtime within two weeks [9]. For a website used by researchers, even brief outages have the potential to interfere with workflow and undermine trust.

Performance is also a problem. Benchmarks indicate that Hostwinds servers have slower response and unreliable throughput compared to alternatives like DigitalOcean. This unreliability would have catastrophic effects on user experience under high usage conditions. As well, customer experiences indicate intermittent slow technical support response time and unclear documentation. Combined with the reduced developer population, this would introduce time and risk into managing our project.

Considering these aspects, Hostwinds simply is not providing the predictability, reliability, and transparency of operation needed for a mission-critical educational application. Therefore, we excluded it from our consideration.

Here is a brief table of each alternative's ranking against the desired criteria (where 5 = very good, 1 = poor fit):

Alternatives	Scalability	Performance	Flexibility	Reliability	Cost	Support
DigitalOcean	5	5	5	5	4	5
PythonAnywhere	3	3	3	4	5	4
HostWinds	3	2	3	2	3	3

Note: These are based on our collective research, small tests, user feedback, and our application-specific needs consideration.

Chosen Approach:

Based on the above, DigitalOcean Apps(s) is the most suitable choice for production, with PythonAnywhere being suitable as our staging/testing/early use environment.

In general, DigitalOcean is unequivocally better at flexibility, performance, and scaling reliably. PythonAnywhere is strong for development/early stage/testing or low traffic, with low maintenance overhead. Hostwinds lags in reliability under load, performance, and introduces risk of unpredictable uptime and less predictable behavior.

Hence, the chosen strategy is:

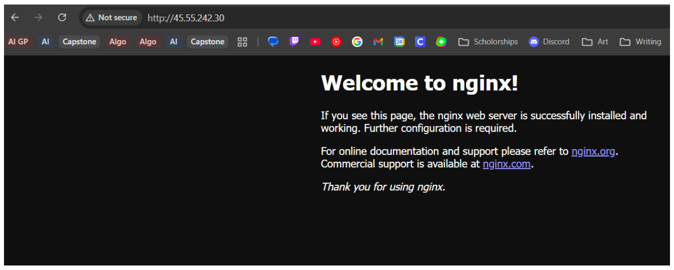
- Use DigitalOcean Apps with proper design (e.g., caching, standalone database, background workers) for production.
- Use PythonAnywhere for testing, staging, early deployment, and initial demos.

Together, they are the best choice for out development and ultimately what we and the client need. This again includes, but is not limited to, scalability, reliability, and performance. Because the final product is also anticipated to be used for the foreseeable future, we wanted to make sure that our client is well set-up to have all these from the start.

Proving Feasibility:

We have already validated that Django is good to go on both PythonAnywhere and DigitalOcean. Both sites successfully hosted the site, meaning that they could support enabling users to register, log in, post reviews, and retrieve data without any issues. These hosting tests guarantee that both platforms are compatible with our tech stack and capable of executing our core website functionalities.

Below are screenshots of evidence that the chosen platforms are Django compatible and hostable. This proves the feasibility and that the system operates without problem in test and production environments. As we develop the website, most strain and scalability testing will occur.

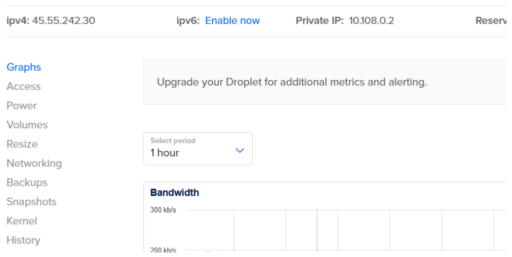

Not secure http://45.55.242.30

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

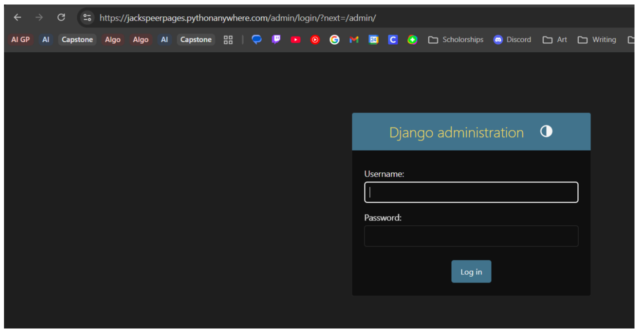

ipv4: 45.55.242.30 ipv6: Enable now Private IP: 10.108.0.2 Reserv

Upgrade your Droplet for additional metrics and alerting.

Select period: 1 hour

Bandwidth

And below here are screenshots from PythonAnywhere, displaying that it also works, this time more explicitly on the Django admin page, in order to show off two different ways the site is viewed (the one above shows what the current homepage looks like, and the one below displays what the admin login page looks like:

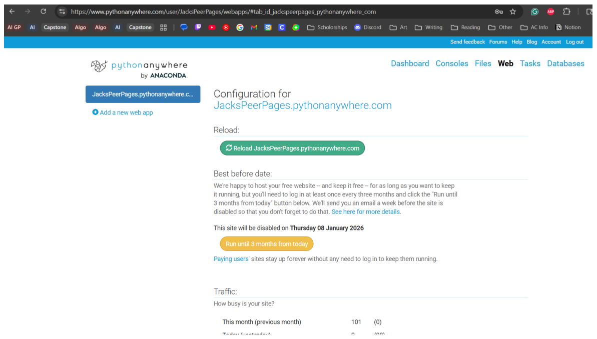

https://jackspeerpages.pythonanywhere.com/admin/login/?next=/admin/

Django administration

Username:

Password:

Log in


https://www.pythonanywhere.com/user/JacksPeerPages/webapps/#tab_id_jackspeerpages_pythonanywhere_com

Configuration for JacksPeerPages.pythonanywhere.com

Reload:

Reload JacksPeerPages.pythonanywhere.com

Best before date:

We're happy to host your free website -- and keep it free -- for as long as you want to keep it running, but you'll need to log in at least once every three months and click the "Run until 3 months from today" button below. We'll send you an email a week before the site is disabled so that you don't forget to do that. See here for more details.

This site will be disabled on **Thursday 08 January 2026**

Run until 3 months from today

Paying users' sites stay up forever without any need to log in to keep them running.

Traffic:

How busy is your site?

This month (previous month)	101	(0)

The code used to make the above screenshots can be found in [this repository.](#) It can be found under the mysite/mysite folder and the mysite/static/admin folder.

6. (UI Development) Users need an easy way to interact with the website through a clean, intuitive UI where they can review posts and post peer reviews themselves.

Introduction:

It is essential that users are able to read, review, and search for articles with a clean, intuitive UI. Our site is primarily driven by its users, as they provide the majority of the content through uploading articles and peer reviews. Not only that, but the purpose of the website is to enable users to both search for and post this content. As a result, removing as much friction as possible in the process of user interaction is key for ensuring that users have a satisfactory experience and are inclined to continue to look at and upload content to the site. Making it as easy as possible for users to interact with the site is imperative for building out the database and fulfilling the website's purpose as a way for researchers to access information.

For our project, we are using the Django framework, which is compatible with both PythonAnywhere and Digital Ocean. As a result, our team will be using the Django stack for UI development. Django allows for HTML, CSS, and Javascript to be easily integrated to create a seamless, efficient, and intuitive user experience. This system is highly flexible, and will allow our team to implement complex handling and create elegant, user-friendly, clean design. During testing for the website, we will do user experience testing to ensure that the website meets these criteria. The following section details how our team will use the Django web framework to create our easily navigable user interface and experience.

Desired Characteristics:

1. Intuitive
2. Easily navigable
3. Simple
4. Aesthetically pleasing
5. Useability

The desired characteristics of our website's UI and UX is for it to be intuitive, easily navigable, simple, aesthetically pleasing, and easy to use. Making the site intuitive and easily navigable helps to ensure that users are able to find what they are looking for and use the website without any extensive training or trial and error. A site that is intuitive is easy for users to understand how to use because they follow conventions and clear that people are used to seeing and already know the meaning of. As a result, the user can use their prior experience and figure out how the website works automatically, without having to struggle to figure out where things are or what certain buttons do. A site that is easily navigable will ensure that a user is never more than one to two pages away from their desired location on the website. This avoids forcing users to go through long chains of pages to access specific content. Instead, we will include clear navigation bars and descriptive page titles to help users find features and information easily.

Not only is it important for a user to be able to navigate and understand a website, but it should also be simple and aesthetically pleasing, so that the user can enjoy the experience of using the site. A website that is too cluttered or has too many graphics and bright colors can be distracting and unpleasant for users, especially in the context of an academic compendium . As a result, it is essential that our site is simple, without confusing, extraneous features or over the top design. This is augmented by the website being aesthetically pleasing. A UI that is well designed, even a simple UI is made much more palatable to users if it is styled well. Having a UI and UX that are simple and aesthetically pleasing improves the users impression of the site. Usability is a factor that is tied closely to the navigation, layout, and aesthetics of the site. Ensuring that these aspects all contribute to greater useability, which is imperative to user satisfaction. A site that is intuitive, easily navigable, simple, aesthetically pleasing, and easy to use not only enhances the way a user feels about the site, but is also critical for users to effectively use the product.

Alternatives:

Due to our prior decision to use the Django framework, it makes the most sense for us to use either combination of Django Templating system (HTML, CSS, Javascript), Bootstrap, and React.

1. The Django Template System (HTML/CSS/JavaScript) natively integrates with Django's back-end logic. This makes it ideal for displaying dynamic pages and managing forms, authentication, and presenting data.
2. React.js Front-End with Django REST Framework offers an interactive and easily scalable interface. It is known to be a favorite among developers, but requires significantly more setup and code separation.
3. Bootstrap and Tailwind CSS Frameworks focus on design standardization and responsiveness. Both are integrateable with various development strategies, including alongside Django Templates, to simplify layout and style.

Analysis:

The three primary alternatives each have strengths and weaknesses to take into account when deciding on the most optimal approach for our project.

The Django Template system is highly compatible with our chosen Django framework, as it natively integrates with the logic of its back-end [11]. Not only that, but it is also a robust system that allows for highly customizable, interactive, fast, and dynamic pages. This allows a website to be intuitive, navigable, and aesthetically pleasing. It is also well suited to managing forms, works well with the authentication tools that our team has decided to use, and can present data so that it is efficient and aesthetically pleasing. This system is well integrated with Django, extremely customizable, and dynamic.

React.js Front-End with the Django REST Framework is also extremely interactive, and highly scalable. React is a very high performance system, with exceptional scalability and customizability, which would bolster navigability and intuitiveness. However, it introduces significant architectural complexity to the system, and is not as easily compatible with Django. Using React.js would allow for optimized performance and flexibility, but would highly complicate the backend of our site.

The Bootstrap with Tailwind CSS Frameworks are highly focused on design standardization and responsiveness. This is highly compatible not only with Django but also with other UI/UX systems. The ease with which our team could integrate Bootstrap into the Django Template System or React.js Front-End is one of its greatest strengths. It is often used for simple, aesthetically pleasing design in the UI/UX that can be written with efficient, polished code.

These three alternatives are all viable solutions to creating an intuitive, navigable, simple, and aesthetically pleasing. Django and React.js provide remarkable performance, scalability, and flexibility for development, while Bootstrap is a great option to complement either of these systems with standardized, elegant stylization. The table below takes these points into account to present each alternative's ranking against the desired criteria (where 5 = very good, 1 = poor fit):

Alternatives	Intuitive	Navigable	Simple	Aesthetically Pleasing	Usability
Django Template System	5	5	5	5	4
React.js Front-End with Django Rest Framework	3	5	3	4	2
Bootstrap	5	2	4	5	4

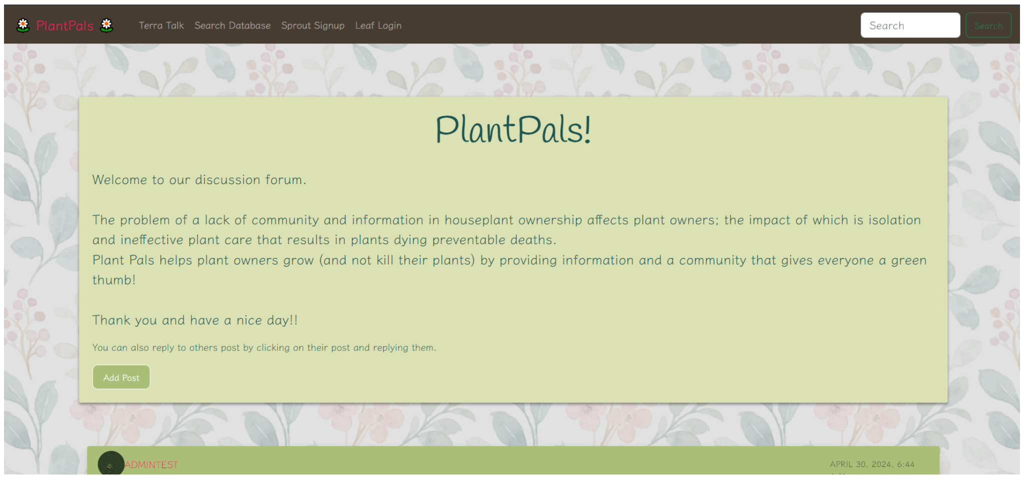
Chosen Approach:

For this project, we have decided to go with the Django Template System due to its ability to integrate with Django's back-end logic as we will be using Django for the front-end of the project. In addition, we will use Bootstrap in parallel with the Django Template System due to their ability to integrate with each other and bootstrap simplifying the template process as well [20]. Furthermore, this will help us avoid the setup that the React.js front-end with Django REST Framework would require, allowing us to get straight to developing the user interface itself.

The reasons above highlight why we have chosen to develop our user interface with the Django Template System in tandem with the Bootstrap Framework. This will allow us to quickly develop and redesign our user interface as needed throughout our user testing process and the feedback we receive from it. Also, this decision provides us with development tools that multiple team members have used in the past, reducing the setup and learning required before project development.

Proving Feasibility:

We have already tested and even have background knowledge of using Django templates and bootstrap. As an example of both in action, working together to create a hosted website that was developed for a former project, here is a screenshot of the website, showing both in action to create a functional UI:



PlantPals!

Welcome to our discussion forum.

The problem of a lack of community and information in houseplant ownership affects plant owners; the impact of which is isolation and ineffective plant care that results in plants dying preventable deaths.
Plant Pals helps plant owners grow (and not kill their plants) by providing information and a community that gives everyone a green thumb!

Thank you and have a nice day!!

You can also reply to others post by clicking on their post and replying them.

In the future, seeing as the UI can be made using this method, our plans will revolve around ensuring that the UI is user friendly. To do this, we will involve user testing and get feedback regarding the intuitive (or non-intuitive) nature of the website and its layout.

Technology Integration

Having resolved each of the technical issues on an individual basis, we would next like to outline how all of these individual solutions fit together into a single overarching whole system. The overall architecture weaves each part of the project into a unified, secure, and scalable platform that is capable of achieving all the objectives of the product.

Central to our system is the Jack's Peer Pages web application. It is accessed by users using a web browser, which communicates with our website hosted by DigitalOcean for final deployment and PythonAnywhere for development. These services host the front end of the website, where users can see pages, write reviews, and search for details.

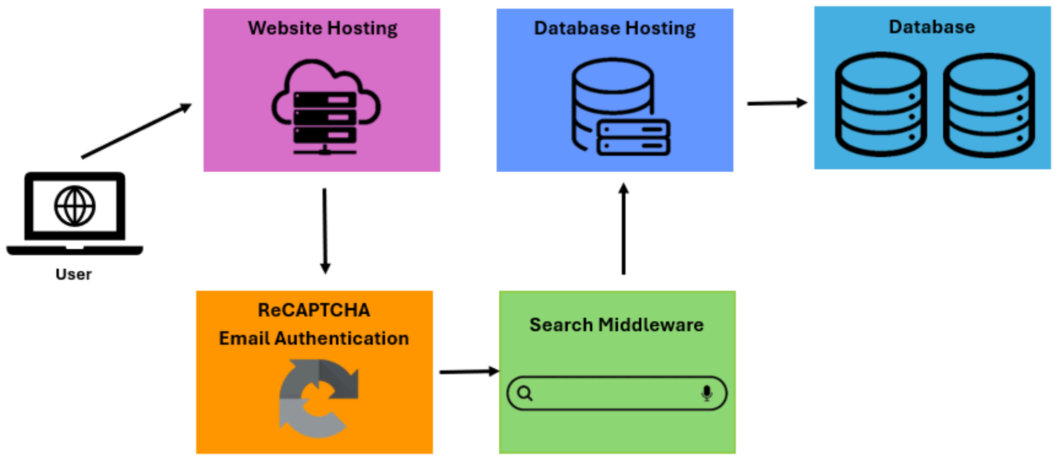
During login or registration, the user passes through Google reCAPTCHA and Django-allauth, both of which handle bot prevention alongside email authentication. They succeed in keeping access to the site only accessible to authenticated human users.

After authentication, the user's action (e.g., searching for a paper or composing a review) is handled by the search middleware. This system component uses Django's ORM, Haystack, and RapidFuzz to quickly search through many documents and give accurate results even in case of spelling mistakes or related terms.

The middleware communicates with the MySQL database, which will also be hosted on DigitalOcean. The database stores all the site's information, including user accounts, peer reviews, and cited articles. It supports concurrent writes and reads to allow many users to access and update information at the same time. Regular backups and encryption protect the information for it to be reliable and secure.

The user interface, built with Django’s templating system and Bootstrap, will serve as the primary point of interaction between users and the platform. It will deliver a responsive and intuitive design that allows users to easily navigate, search, and submit peer reviews. Because the UI connects seamlessly with the Django backend, it dynamically renders data and ensures that every action (such as logging in, to posting a review) will feel smooth. This is part of what we are referring to as the middleware.

The following diagram depicts how these bits of software interact and the data flow amongst them. Arrows are drawn to represent communication among the website, middleware, and database, with results and user input traveling back and forth through these tiers.



As a whole, these pieces form a harmonious system, where each part is needed and works together. DigitalOcean handles hosting and scaling, MySQL handles secure data storage, Django-allauth and reCAPTCHA secure the logins, and the search middleware provides users with an easy way to find what they need. Together, they give a web application that is efficient, secure, and ready to scale as the project expands.

Conclusion

Over the course of the coming months, our team will be faced with many difficulties in producing this project. By defining sound solutions to key technical problems at an early stage, we will be in a position to approach these problems better and more easily. Early on in the paper, we identified that the most important technological factors for this project will include database hosting, selecting an appropriate database framework to use, creating a secure and provable user authentication mechanism, creating a speed and reliability maximized lookup system, and hosting the website. For each of these issues, we canvassed the technological solutions available to us, evaluated their pros and cons, and selected the optimal ones for our project's specifications. We also conducted preliminary tests of our solutions to verify practicability within the project's constraints. This combination of testing and analysis has allowed us to create a concrete, evidence-based plan of how we are going to complete our project to the specifications and expectations of our client.

The most significant outcomes of this process was our decision to use MySQL as the database, reCAPTCHA and Django-allauth for user verification, and DigitalOcean Apps as the final hosting service, with PythonAnywhere for development for the final product, and Django templates and Bootstrap for the UI. We will also use PythonAnywhere for testing (including initial user testing to determine the intuitiveness of our system) and deployment during the early phases of development. These choices reflect our desire to have a system that is secure, scalable, and sustainable.

Apart from technical information, we would also like to recite the wider purpose of Jack's Peer Pages. The project hopes to open up peer review, make it more accessible and learning-oriented, plugging a gap in the academic publishing process and empowering students and researchers. By offering an open platform where peer reviews can be easily shared and linked to published output, we hope to enhance the credibility and openness of scientific communication.

In the future, our subsequent activities will include completely deploying the testing environment, completely integrating the MySQL database into our Django web application rather than into a test application, and completely implementing and integrating the selected authentication and search mechanisms. During development, our efforts will be devoted to refining functionality, optimizing performance, and preparing for user testing. With these principles laid down, our group is confident that Jack's Peer Pages shall be a priceless and lasting tool that fosters trust, collaboration, and education among the research community.

References

[1] Emrahdiler. (2024, May 20). *What is account verification? - best practices*. Fraud.com. https://www.fraud.com/post/account-verification

[2] GeeksforGeeks. (2025, July 23). *How to add google recaptcha to Django Forms ?*. GeeksforGeeks. https://www.geeksforgeeks.org/python/how-to-add-google-recaptcha-to-django-forms/

[3] Google. (n.d.). *reCAPTCHA bot protection and online fraud prevention*. Google. https://cloud.google.com/security/products/recaptcha?hl=en

[4] Koch, T., & Nguyen, S. T. (2025, September 16). *Email authentication*. Federal Trade Commission. https://www.ftc.gov/business-guidance/small-businesses/cybersecurity/email-authentication#:~:text=Email%20authentication%20technology%20makes%20it,then%20notify%20you%20about%20them.

[5] Mbi. (n.d.). MBI/Django-simple-CAPTCHA: Django Simple Captcha is an extremely simple, yet highly customizable Django Application to add CAPTCHA images to any django form. GitHub. https://github.com/mbi/django-simple-captcha

[6] *Microsoft*. What is: Multifactor Authentication. (n.d.). https://support.microsoft.com/en-us/topic/what-is-multifactor-authentication-e5e39437-121c-be60-d123-eda06bddf661

[7] *Welcome to django-allauth!*. allauth. (n.d.). https://docs.allauth.org/en/latest/

[8] *What is multi-factor authentication (MFA)?*. OneLogin. (n.d.). https://www.onelogin.com/learn/what-is-mfa

[9] Cybernews. “ScalaHosting Review: Can This Managed VPS Replace Shared Hosting?” *Cybernews*, 14 Aug. 2024, https://cybernews.com/best-web-hosting/scalahosting-review/.

[10] DigitalOcean. “Pricing.” *DigitalOcean*, https://www.digitalocean.com/pricing.

[11] Hostwinds. “Shared Hosting.” *Hostwinds*, https://www.hostwinds.com/hosting/shared.

[12] MongoDB. “NoSQL Explained.” *MongoDB Resources*, https://www.mongodb.com/resources/basics/databases/nosql-explained.

[13] PostgreSQL. “About.” *PostgreSQL.org*, https://www.postgresql.org/about/.

[14] PythonAnywhere. “Plans and Pricing.” *PythonAnywhere*, https://www.pythonanywhere.com/pricing/.

[15] Amazon Web Services. “The Difference Between MySQL vs. PostgreSQL.” *AWS*, https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/.

[16] Northflank. “DigitalOcean vs AWS: A Guide for Developers, Startups, and AI Companies.” *Northflank Blog*, 26 Aug. 2025, https://northflank.com/blog/digitalocean-vs-aws. Northflank

[17] Microsoft Azure. “Pricing — Azure SQL Database Single Database.” *Microsoft Azure*, https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/single/

[18] Microsoft, “What is the Azure SQL Database service? - Azure SQL Database,” *learn.microsoft.com*, Sep. 28, 2022. https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql

[19] Django Software Foundation. (n.d.). *Templates — Django 5.2 documentation*. Retrieved October 23, 2025, from https://docs.djangoproject.com/en/5.2/topics/templates/

[20] W3Schools. (n.d.). *Django – Add Bootstrap 5*. Retrieved October 23, 2025, from https://www.w3schools.com/django/django_add_bootstrap5.php

[21] SQLServerCentral. (2024, January 9). *Django's ORM vs. Raw SQL: When to Use Which – SQLServerCentral*. https://www.sqlservercentral.com/articles/djangos-orm-vs-raw-sql-when-to-use-which

[22] Anand, R. (2025, February 14). *Fuzzy Matching Python: Optimizing large datasets*. TECH CHAMPION. https://tech-champion.com/programming/python-programming/optimizing-fuzzy-matching-for-large-datasets-in-python/

[23] *Django Haystack* | *Compile n run*. (n.d.). Compile N Run. https://www.compilenrun.com/docs/framework/django/django-advanced-topics/django-haystack/