Pandemic Processing

Technological Feasibility

November 6th, 2018

Project: Epidemiological Modeling Portal

Team Members:
Anthony Schroeder
Joseph Eppinger
Tanner Massahos

Sponsor:
Dr. Joseph Mihaljevic

Mentors:
Dr. Eck Doerry and Jun Rao

Table of Contents

1 INTRODUCTION 1
2 TECHNOLOGICAL CHALLENGES 2
3 TECHNOLOGICAL CHALLENGES ANALYSIS 3
3.1 Web Framework 3
3.2 Database 10
3.3 Hosting 14
4 TECHNOLOGICAL INTEGRATION 19
5 CONCLUSION 20

1 INTRODUCTION

Infectious disease has been a problem throughout all of recorded human history. Thus, with an increase globalization, infectious transmission is growing. With humankind's current technologies, it should be no problem to accurately model the spread of infectious disease. However, the current process by which models can be shared between Epidemiologists, those who study infectious disease, is incredibly slow.

Epidemiologists from all around the world generate models which represent the spread of infectious disease within a given population. These models can be used to predict and possibly answer important questions within the community. Typically, these models implement complex functions and utilize programming languages such as Python or C to generate a CSV which is then piped to a graphing technology such as RStudio. This process is simple and common, nevertheless, a problem arises when scientists want to share their models within the community for fine tuning or verification. Currently, there is no efficient way for these individuals to share and have their models edited and critiqued by other scientists.

Thus, we propose a secure, fast, and user friendly web application where epidemiologists from all around the globe can share and discuss their models. This discussion will provide ample opportunity for epidemiological models to develop quickly. In preparation for creating such an application, we provide the reader with an outline for how we plan to investigate the technologies we may use.

This Technological Feasibility Report will delve into our decision-making pertaining to specific technologies that we determined will be required when developing a shared interactive portal for exploring epidemiological model predictions:

1. We will explain the major technological challenges and necessities; specifically referencing high-level requirements that we deem to be essential to the creation of the product.
2. The technological analysis will reference specific technologies that satisfy the high-level requirements stated in the previous section.
3. After carefully examining these cutting-edge technologies, we will bring it all together with our vision on how the technologies will integrate with one another followed by a proper conclusion summarizing our findings and plans.

The discussion of epidemiological models is an incredibly important one. The tools which we decide to use will aid us greatly in creating an application which appropriately aids epidemiologists in creating and sharing their models.

2 TECHNOLOGICAL CHALLENGES

The ultimate goal of our web application is to create a greater sense of community which allows for invigorating discussion amongst epidemiologists. As such, the application must have certain integral features for it to function properly. To that end, we must meet the following high level requirements:

- **Enable an Interactive Community** - The app must provide tools through which users can interact with each other and their models.
- **Public/Private Forum Posts** - The portal should allow for teams of epidemiologists to collaborate through forum posts, both privately and publicly.
- **Interactive Model Interface** - Viewing and interacting with visual representations (typically graphs) of models must be possible. Additionally, users should be able to choose whether their model is private or public.
- **Modifiable and Redistributable Models -** Allow users to introduce suggested changes to epidemiological models and repost them within the forum with a small description.
- **Run User Provided Modeling Code -** Models are typically generated by user code, and thus the app should be able to store and run this code.
- **Simple workflow** - The flow of uploading, viewing, modifying and sharing models should be intuitive and relatively easy.
- **Scalability -** The final product should be easily scalable so that this app can be extended to allow models beyond just epidemiological ones.

These requirements must be implemented for an appropriate and well performing web application. Thus, they necessitate a set of technologies necessary for the development of said web application:

- **Web Framework** - For our web application we want to utilize a web framework as it will make things like database integration, developing the overall application simpler, and makes it easier for future developers to manage. Additionally, our framework should have support for modeling data, regardless of whether that support is included or available in external libraries which the application can make use of.
- **Database -** We need a way in which we will store user accounts and forum posts with their corresponding comments, models, and source code provided.
- **Hosting** - We will need a fast, reliable, and portable service to host our website when we are ready for it to go live.

3 TECHNOLOGICAL CHALLENGES ANALYSIS

Naturally, selecting the right tools for our application is essential. There are many tools available to us to create a web application with a sufficient database and reliable hosting. We will analyze those which we believe best suit the needs of our application and compare them to find the most fit. We will use a set of metrics which are the most important components for each particular piece of the application to have.

3.1 WEB FRAMEWORK

Our solution will require a robust web framework. As such, we have outlined what we believe to be the most important components of such a framework:

- **Security:** A suitable framework must be secure so that the data stored in our application is not vulnerable when in use.
- **Ease of use:** Web frameworks can be difficult to use, and thus in order to maintain the future use and support for this project, it is important to note which frameworks are the easiest for future developers to take over.
- **Future support:** A framework will not be useable if it loses support in the near future. Thus, continued development of the framework chosen is essential.
- **Scalability:** This application may grow in the future to model more than just epidemiology, or to model diseases in different ways. Thus, the ability for this application to grow is integral.
- **Modeling Capabilities:** Without proper modeling, the entire application falls apart. As such, a sufficient framework will have a powerful modeling engine the application can make use of.
- **Account and Comment Management:** While the database will store this information, it is important for the framework we choose to properly interpret and manage user data.
- **Speed:** Users will not wait around for very long on typical web pages. As such in order to make the application as usable as possible, it must be fast.

Without these essential features, a framework is not at all suitable for this application.

3.1.1 Django

Django is a web framework which uses python to assist developers with a large number of helpful tools for development. Django is used in nearly every type of website, and is flexible for all types of use. We have evaluated Django based on testimonies of those who use it and analyzed the features which Django promises to include itself:

Pros:

- **Security:** 5/5
  Issues are present in all platforms, as languages don't inherently prevent security issues, they just require some extra attention to have programming logic to check for possible unauthorized data requests. These issues are easy to mitigate.
- **Ease of use:** 5/5
  Python is easy to understand and quick to implement, with a lot of resources relating to Django and Python online.
- **Future support:** 5/5
  Django is widely used and currently has a growing following for web framework development. The current version is suggested to be supported until 2021.
- **Scalability:** 5/5
  Django is known to be utilized on major social media applications such as Instagram. Therefore, pertaining to traffic of individuals it can be seen it can handle millions of users. You can also set up multiple instances of the same application and load balance incoming requests.
- **Modeling Capabilities:** 5/5
  As it is written in python, Django has access to NumPy. NumPy is an incredible library which provides support for data processing, graphing and:
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities
- **Account and Comment Management:** 4/5
  Database tools and management is easily configured within Django. Regarding Database security, simple input checks are required for many database calls to prevent fraudulent database requests.

Cons:

- **Speed** 3.5/5:
  Python which is an interpreted language, which makes it slower than compiled languages, but is still quick.

Django is a wonderful framework for developing a web application. It provides, easy to use Object-Oriented environment, future support, scalability, and provides plenty of online resources for debugging and troubleshooting.

3.1.2 Rails

Rails is an extremely popular and widespread web app framework which uses Ruby. Rails makes for fantastic large-scale applications with incredible support for backend connections (namely with database management systems). We have derived our analysis from our own use of Rails and the features it declares it provides:

Pros:

- **Security:** 5/5
  Rails web applications are only as secure as we make them. Ruby has plenty of the power necessary to create highly secure web applications.
- **Modeling Capabilities:** 5/5
  Gruff is a gem which can be used with Rails to display many types of graphs. Gruff is relatively easy to use and produces simple, but elegant graphs.
- **Account and Comment Management:** 5/5
  Accounts are incredibly easy to setup with ActiveRecord support that Rails provides. Additionally, such support makes it incredibly easy to store comments and other user related data in relational database systems.
- **Scalability:** 4/5
  Rails is quite scalable, but does not lend itself particularly well to small projects. Using rails for smaller projects is a particularly large amount of overhead, especially when compared to the final output.
- **Future Support:** 5/5
  Neither Ruby nor Rails are going anywhere anytime soon. Rails is a popular framework with many trained individuals completely capable of taking over a project like this.

Cons:

- **Speed:** 3.5/5
  Ruby is an interpreted language, which is certainly going to make it slower than compiled languages. Embedded ruby and related controllers increase loading time as well.

- **Ease of Use:** 2/5
  Rails has a steep learning curve. Ruby is a quirky language with several special properties. Rails simply builds on this and many of the things rails does is not always intuitive. Additionally, Rails comes with opinions on how a web applications should be built. When a design steps outside of this, Rails becomes much more difficult to work with.

Rails is a great framework if we choose needed to run a large-scale application. However, smaller applications may suffer from the large amount of overhead required in a Rails application

3.1.3 Ionic

Ionic is a single page framework which can be used to deploy web applications to a multitude of different devices. Ionic operates entirely on a single page, and thus does not make any html requests for a full page after the first. Nearly any website can be built in Ionic, and the most substantial benefit is the lack of loading times. The analysis provided for analysis is primarily garnered from our own experiences and Ionic documentation.

Pros:

- **Speed**: 5/5
  Ionic uses typical web technologies and is the fastest available solution.
- **Security**: 5/5
  Ionic is a front-end only framework. Thus, it is about as secure as we make it. There are plenty of tools to encrypt and safeguard user data which would leave the application.
- **Modeling Capabilities**: 5/5
  The modeling capabilities of Ionic can be derived from our documentation regarding D3.js. Overall, there do not seem to be any issues at all with such modeling capabilities.

Cons:

- **Account and Comment Management**: 3.5/5
  Nothing is immediately built in here, and data management would leverage the power of whatever backend we decide to use. This does mean however that these connections will be made ourselves.
- **Ease of Use**: 3/5
  Ionic use is nontrivial. There is quite a bit which goes into developing a complex app in Ionic. It is by no means unusable, but given Ionic is relatively new and complex, it may be difficult for anyone who develops this in the future.

- **Scalability**: 3.5/5
  Ionic is great for small-scale applications based on our own usage. Larger applications would likely prefer a framework which can support new features more easily as the application evolves.
- **Future Support**: 3.5/5
  Ionic probably isn't going anywhere, but it's an open source project, so there are fewer guarantees than some other frameworks. In addition, Ionic is not heavily used in large scale applications. With fewer dependencies, Ionic is more likely to be dropped than something like Django.

Ionic is not a perfect framework, but its flexibility is worth considering indeed. If the application was to ever be ported to a mobile device, Ionic would simplify the transition greatly.

3.1.4 Flask

Flask is a relatively young framework, only in use since 2010. Flask is considered more "Pythonic" than Django simply because Flask web application code is, in most cases, more explicit. Flask is the choice of most beginners due to the lack of roadblocks to getting a simple app up and running. We have analyzed Flask through the testimony of those with extensive knowledge of it and its documentation.

Pros:

- **Modeling Capabilities:** 5/5
  NumPy is an amazing package for computing in Python, which (among other things) allows for:
  - tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities
- **Speed:** 4/5
  Python is an interpreted language, which is certainly going to make it slower than compiled languages. However, Flask is relatively lightweight, so the extra utility lost when compared to a framework like Django can be translated to a bit of extra speed.
- **Security:** 4/5
  Flask is in many ways no more or less secure than any other web frameworks. However, it does make efforts in the realm of security such as escaping all text so that cross site scripting is nearly impossible. While some other security issues are not solved by Flask directly, other python libraries can be utilized to remedy these issues.

- **Ease of Use:** 4/5
  Flask is an extremely lightweight framework, and thus does not overwhelm us with extraneous features. However, this is somewhat of a double-edged sword, and as such, reaching for more complex features will require more work on our end.

Cons:

- **Account and Comment Management:** 3.5/5
  There are no problems with account and comment management in Flask, but database connection is not immediately built-in. Thus, using Flask may require some heavy lifting to create secure DB connections.
- **Scalability:** 3.5/5
  Flask is relatively scalable, however, with larger systems the benefit of a lightweight framework could diminish. It is important to weigh this out as the project progresses and be ready to pivot to another Python framework if Flask is not a good fit.
- **Future Support:** 3.5/5
  Flask is an open source project which should likely see support for the foreseeable future. In the event support ends, there are other Python frameworks which could easily be migrated to.

Overall, Flask seems like a great framework to use for relatively lightweight applications. There is some extra work we must do for database usage and such, but overall Flask is a great place to start. Additionally, much of the code used in a Flask application can transition to a Django framework if the application grows.

Summary

In summary, we have evaluated a few powerful frameworks and outlined their suitability as far as the key features we require detail. This analysis is recorded in the following table:

Web Application Frameworks	**Django**	Rails	Ionic	Flask
Speed	**3.5**	3.5	5	4
Security	**5**	5	5	4
Ease of use	**5**	2	3	4
Future Support	**5**	5	3.5	3.5
Scalability	**5**	4	3.5	3.5
Account/Comment Management	**4**	5	3.5	3.5
Modeling Capabilities	**5**	5	5	5

As the table shows Django most completely fulfills the requirements we have determined. Django is incredibly powerful in all the areas we are most concerned with except for speed. Even considering this however, such speed can only be surmounted by using native technologies and using a compiled language (which few frameworks still exist for today).

To demonstrate and test the feasibility of the Django Framework, we will create a small demo showcasing the features we find most important:

- The demo will operate using Joseph's model he gave to us, and create a graph which can be modified in the web application.
- The demo will step through the process a user may encounter to upload and view their model.
- The most challenging part of this application, running nonnative code to acquire data for the models, will be tested in this demo.

3.2 DATABASE

Databases are important for keeping all essential information categorized and easy to get to. For our web application, we are looking for something that can manage user accounts login credentials, user messages, and teams. Thus, we have developed a schema by which we will evaluate possible database management systems:

- **Ease of use**: Database management systems are notoriously complex, and as such one which is easy to use will be a boon in future development.
- **Reliability**: The database for our application must keep data in a way which is safe and reliable.
- **Future support:** Losing support for the chosen database could open our application up to future security threats, which is not acceptable.
- **Account Management:** User accounts should be stored within the database and easily managed.
- **Comment Management:** User comments should be stored within the database and easily managed.
- **Security:** The database system used should be secure so as to protect user data.
- **Scalability:** The database system must be able to grow as the size of the application's data, scope and user-base increases.
- **Flexibility**: The application will change over time, and the database must be able to change alongside it.
- **Speed:** Quick access to data is integral to a well-functioning database system.

The database management system chosen for this application will be the cornerstone for storing and retrieving data for the web application which is obviously held to a high importance.

3.2.1 PostgreSQL

PostgreSQL is a Flat File base database allowing for table creation, and standard queries. This is a lightweight database capable of managing large data warehouse operations, allowing for a large amount of flexibility. The following analysis of PostgreSQL is derived primarily from our own usage as well as documentation provided online.

Pros:

- **Ease of use**: 5/5
  Operates much like any SQL database, requiring common SQL commands. There is also plenty of online references that help develop and troubleshoot any issues.

- **Reliability**: 5/5
  PostgreSQL is ACID compliant, meaning that data and operations are 'all or nothing', meaning that you can rely on commits to be whole, and have not been partially submitted. ACID compliance allows users to be confident that the data will not be corrupted if an issue arises.
- **Future support:** 5/5
  The current version, 10.5 will be supported till 2022, along with the large reliance and following within the programming community, support is unlikely to go away anytime soon.
- **Account Management:** 5/5
  Accounts can be managed using a simple bridge table with a PK (primary key) / FK (foreign key) to manage aspects of an account through relationships
- **Comment Management:** 5/5
  As stated above, PK (primary key) / FK (foreign key) relationships are an easy way to manage comment history.
- **Security:** 5/5
  Security might be more involved with the caller code and requires logic to verifying no there are no SQL Injection Attacks, but like most database models, security is mainly done with the Framework, preventing people from making fraudulent database calls.
- **Scalability:** 5/5
  The size of the database can work at differing sizes, allowing anything from a small project to a large data warehouse.

Cons:

- **Flexibility**: 3/5
  As many SQL databases, column types cannot be changed easily and may lead to residual issues when changing the database around. This includes changed values for altered column types. Doesn't differ from Oracle's query like MySQL does, which allows an easy transfer of code.
- **Speed:** 3/5
  Joining large tables is slow, but standard table reference is decently fast compared to MongoDB.

PostgreSQL is a strong database management system which provides many of the different requirements that we are looking for. PostgreSQL's ACID compliance is perhaps its most desirable feature as it helps to ensure reliable data.

3.2.2 MongoDB

MongoDB is a free easy to set up, open-source NoSQL database which utilizes JSON-like documents. MongoDB is used by many web applications to store backend data. Our analysis of MongoDB is garnered from online documentation and online testimony on its behalf.

Pros:

- **Speed**: 5/5
  While MongoDB is extremely fast, it comes at the sacrifice of consistency.
- **Ease of Use**: 5/5
  MongoDB doesn't really have a steep learning curve and since it doesn't follow the ACID principles and isn't relational means it is not very complicated to set up and maintain.
- **Scalability**: 5/5
  MongoDB is extremely scalable, this can be seen from the fact that you can distribute the database over 100+ nodes, store over 1 billion documents, and is capable of 100,000+ read and writes per second with low latency.
- **Account Management**: 5/5
  Can easily store account information.
- **Comment Management**: 5/5
  Can easily store comments within forum posts.

Cons:

- **Security**: 3/5
  MongoDB doesn't provide heavy amounts of security right out of the gate and requires quite a bit of configuration beforehand.
- **Future Support**: 3/5
  We aren't sure about how long MongoDB will survive since it is a newcomer to the world of databases but it is open-source so if there are people continually contributing it should stay relevant.
- **Flexibility**: 3/5
  MongoDB does a good job at storing things like usernames and passwords but is not a relational database.

MongoDB would be valuable when storing user information and possibly comments. It seems to be very fast and secure if configured correctly, also very easy to use and works natively and easily with Django.

3.2.3 MySQL

MySQL is an open-source relational database management system which operates quickly, is platform independent, and makes it easy to query data due to how standardized it is. MySQL is perhaps one of the most ubiquitous database management systems for its use in teaching environments. MySQL has been analyzed based on our own experiences and its own documentation.

Pros:

- **Ease of Use: 5/5**
  Operates much like any SQL database, requiring common SQL commands, and with plenty of online references that help develop and troubleshoot issues.
- **Future Support: 5/5**
  The current version, 8.0 will be supported till 2026 along with the large reliance and following within the programming community, support is unlikely to go away anytime soon.
- **Scalability: 5/5**
  MySQL seems to be extremely scalable and have tremendously high uptime, I wouldn't expect anything less from Oracle.
- **Account Management:** 5/5
  Accounts can be managed using a simple bridge table with a PK (primary key) / FK (foreign key) to manage aspects of an account through relationships
- **Comment Management:** 5/5
  As stated above, PK (primary key) / FK (foreign key) relationships are an easy way to manage comment history.
- **Security: 5/5**
  Again, Security is mainly done within the framework. Security might be more involved with the caller code and requires logic to verifying there are no SQL Injection Attacks, but like most database models, security is mainly done with the Framework, preventing people from making fraudulent database calls.

Cons:

- **Flexibility: 2/5**
  Once table columns have been made, they are extremely hard to change, as changing column types involves modifying or even deleting the entire table.
- **Reliability: 2/5**
  MySQL is not ACID compliant, and thus incomplete or malformed commands may be partially executed. This could harm its ability to provide reliable data.
- **Speed: 3/5**
  MySQL isn't extremely fast but this is just a tradeoff for how consistent it is.

Although there is a lot of support for MySQL, its lack of ACID compliance makes it less appealing. Despite this, its common usage and consistency otherwise does make it still worth considering.

Summary

To conclude, below we have provided a table of how we rate the technologies we analyzed.

Categories	**PostgreSQL**	MongoDB	MySQL
Flexibility	**3**	3	2
Speed	**3**	5	3
Security	**4**	3	4
Ease of use	**5**	5	5
Future Support	**5**	3	5
Scalability	**5**	5	5
Account Management	**5**	5	5
Comment Management	**5**	5	5

As shown in the above chart. We have decided to use PostgreSQL because it has existing libraries which we can use to connect to Django. PostgreSQL is completely free and will remain so given its licensing. Most importantly perhaps, PostgreSQL is reliable and will see support for years to come. Many suggestions are to build the Framework first and implement a database choice later in the project, as it allows for flexibility later in development.

In order to evaluate the usability of PostgreSQL, it will be used to store the information in our demo. This includes and all data related to the modeling the demo will showcase.

3.3 Hosting

A reliable host is incredibly important for a stable web application. Thus, our metrics we will use to analyze various hosts follow:

- **Reliable:** The hosting service for our application must not be unavailable frequently. If the application is not accessible, it is not working properly.
- **Secure:** Secure hosting is important to protect all data for the application, both user and application related.

- **Ease of use:** A complex hosting environment will damage future developers' abilities to work on this application. The hosting platform chosen should avoid this issue.
- **Future Support:** Losing support for a hosting service means a large amount of migration to find a new one. This is extremely undesirable for applications which must maintain nearly constant uptime.
- **Scalable:** The hosting system should be able to grow as the application demands more resources.
- **Portability**: In the event where the application must be moved to a new platform, it should be easy to extract from the old platform.

Selecting a proper hosting service is essential to ensuring the web application is accessible to the users.

3.3.1 AWS EC2

Amazon's Elastic Compute Cloud is a platform provided by Amazon which can grow and shrink as is needed for a website. AWS EC2 is incredibly widespread, and thus is used for all sorts of websites or servers. EC2 has been analyzed based on our own usage and features which Amazon says it provides.

Pros:

- **Reliability**: 5/5
  EC2 is considered highly reliable, and rarely seems to have issues. As seen in many articles, it appears there would be next to nothing to worry about with consideration for reliability. EC2 has several guarantees about uptime and data safety in their service level agreement.
- **Security**: 5/5
  Based on our research, it seems unlikely that we would have any issues caused by Amazon themselves. Connection is secured by encrypted passwords and volumes can also be encrypted.
- **Ease of Use:**4/5
  EC2 is quite easy to use, with an abundance of documentation. That said, it does require some amount of work to set up. Given this, it doesn't get a 5/5, but it is still quite easy to use in our experience.
- **Future Support:** 5/5
  EC2 is not going anywhere. While Amazon does not divulge how many EC2 instances exist, many estimates put the number in the hundreds of thousands. Additionally, amazon is an incredibly large company which additionally is unlikely to go anywhere.
- **Scalability:** 5/5
  The platform is fully scalable based on the needs of the project. More resources can be purchased as necessary.

Cons:

- **Portability:** 3/5
  While moving between instances on an EC2 is quite trivial using EBS volumes, moving an EC2 instance to another hosting platform could be challenging. Amazon does not allow a user to take a snapshot of their instance in any helpful way. Of course, this does not mean the app is stuck on EC2, it just means there will be much more work.

After all the research, we have conducted AWS EC2 appears to be a valuable option which we should not overlook. The cost is low and it is easy to get started.

3.3.2 Heroku

Heroku is a cloud based hosting service which can provide us a space to serve our web application from. Heroku is used primarily to quickly bootstrap small web applications. We have analyzed Heroku based on what it claims to provide, and testimonials from those who have used it.

Pros:

- **Reliability**: 4/5
  It is unlikely that Heroku will have any major issues, but it is worth noting that it does go down occasionally. This of course normal, but certainly not desirable.
- **Security:** 5/5
  While Heroku does put some of the weight of security on us, it does provide several certifications to maintain security and privacy.
- **Ease of Use:** 5/5
  Heroku instances are incredibly easy to set up. They come with some built in architecture for the platform you choose, which makes setup very easy.
- **Future Support:** 5/5
  Heroku is owned by Salesforce, which while not as large as some other companies, is rather stable. We can expect that Heroku will continue to receive support for the foreseeable future.

Cons:

- **Scalability:** 1.5/5
  Heroku only provides support for the web applications it has built in. Making other parts of the system which are disjoint to this work well is an incredible challenge. This includes running client code.
- **Portability:**2.5/5
  Apps running on Heroku are tied into Heroku by default. The repo exists in Heroku, which would make movement of the data and code quite difficult.

At a first glance, Heroku seems quite useful in its ability to instantly set up our instance and platform. However, because of this, extending to tools outside of the chosen platform is difficult, and will likely make using Heroku too difficult.

3.3.3 Digital Ocean

Digital Ocean is also a cloud based hosting service which provides a reliable place for the web application to exist within. Digital Ocean has been the hosting service of choice for a large number of websites for years. The analysis which follows of Digital Ocean is primarily derived from documentation and testimonial from users.

Pros:

- **Reliability:** 5/5
  Digital Ocean has proven from our research to have a high uptime and is fast and cheap.
- **Security:** 4/5
  Would need to keep eyes on security updates and does have default security but we would have to configure our own security for anything extremely specific.
- **Ease of Use:** 4/5
  Digital Ocean can spin up a virtual machine in just minutes and offers a web interface for management and does support SSH.
- **Scalability:** 5/5
  Can copy system image and duplicate to as many virtual machines as needed.
- **Future Support:** 5/5
  Digital Ocean should be around for quite a while as they have a strong foothold within the cloud computing sector.

Cons:

- **Portability:** 3/5
  From our research, we believe it would challenging to copy Digital Ocean images and copy them to a different hosting service.

Digital Ocean appears to be a promising option when it comes to selecting which hosting service we will utilize. It satisfies a large majority of our requirements and scores highly on our ranking system.

Summary

In summary, reliable hosting is essential to creating a proper web application. Thus, below is a table which outlines how the hosting platforms we have analyzed should perform:

Hosting Platforms	**AWS EC2**	Heroku	Digital Ocean
Reliability	**5**	4	5
Security	**5**	5	4
Ease of use	**4**	5	4
Future Support	**5**	5	5
Scalability	**5**	1.5	5
Portability	**3**	2.5	3

As is seen above, we have chosen to use Amazon EC2 as it provides a near perfect balance between ease of use and reliability. Amazon's cloud computing will see support for decades to come, and makes many of our requirements a priority.

We will launch a free EC2 instance which our demo will be tested from. This way, we can easily see how Django and PostgreSQL operate on the server quite easily.

4 TECHNOLOGICAL INTEGRATION

Integrating the major three components of this project should be quite easy overall. Tools already exist for enforcing good communication between these components for us to use to their fullest extent. The most important challenges to overcome however will be ensuring that Django, PostgreSQL and the Amazon EC2 instance all live in harmony and operate stably.

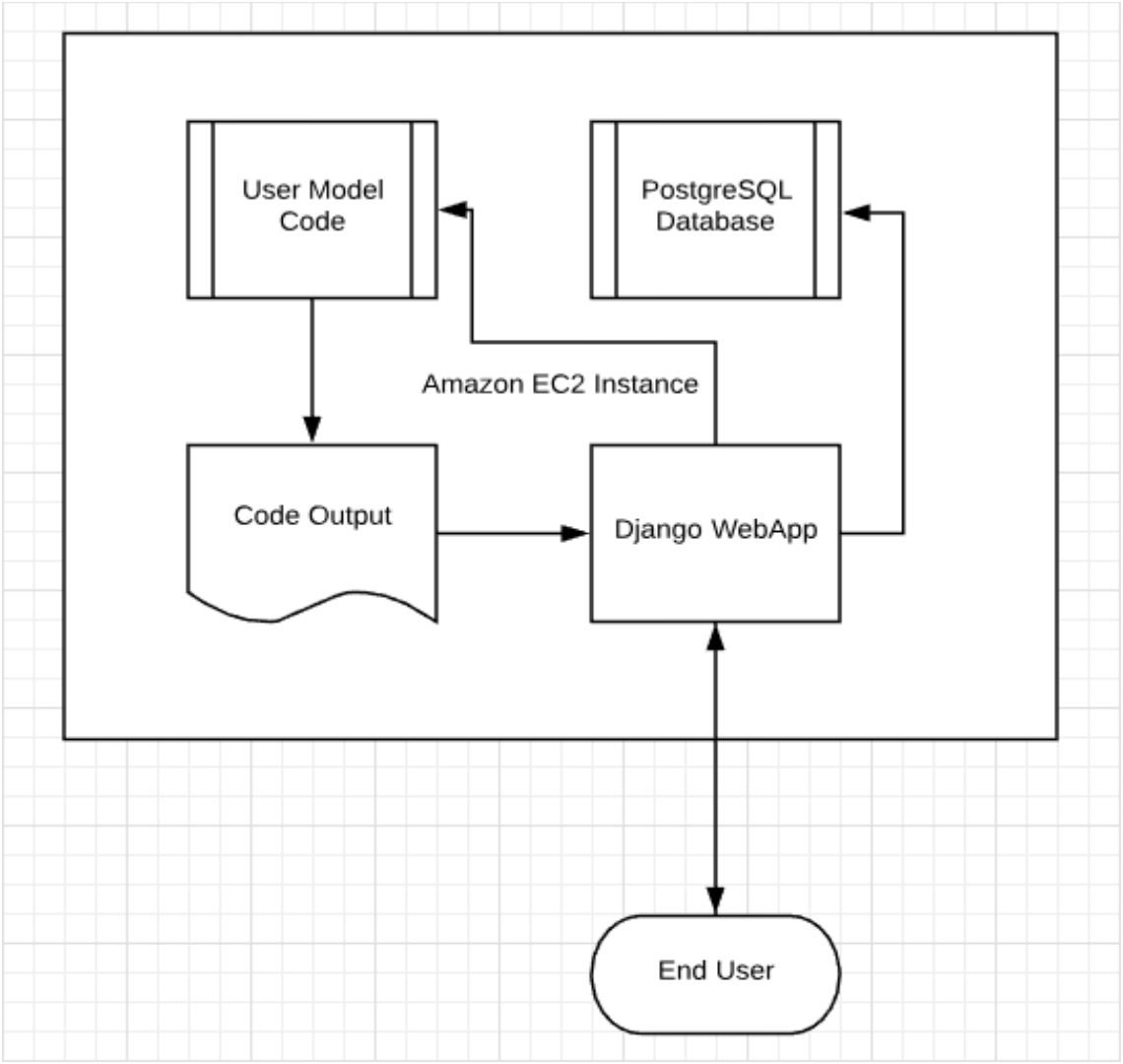

Figure 4.1. A simple model of our system

The model in Figure 4.1 is how our components will interact with each other. In a general sense, the user will send requests to the Django Web Application for all pages on the website. The database backend will provide support for Django by holding all the data Django needs to access. Django can simply run user code and process the output. PostgreSQL and Django live within the EC2 instance and are simply processes and files which exist in the instance.

We will ensure these issues can be solved with our demo, which will incorporate all three of our components. We can thus ensure seamless integration and be sure our decision to use these is correct. These systems already work well together, as we have seen in our research, so integration should be relatively painless.

5 CONCLUSION

Modeling can help epidemiologists immensely when studying diseases. The work these scientists do is integral for preserving the health of our society. Despite this, a system still does not exist by which these scientists can easily communicate. Thus, in order to build such a system, we require a web application with a robust framework, reliable database, and reliable hosting platform.

As shown by our analysis, we have ultimately chosen to use Django as our web application framework, PostgreSQL as our database system, and an Amazon EC2 instance as our hosting service. We find these components to work well together, and be the best solutions to the challenges we have described thus far.

Our plan is to use these components to create a cohesive web application which can be utilized by our users to post, share and interact with epidemiological models. The most pressing question we need to answer is how to efficiently run user code, but that should not be an issue we cannot solve. We know python (as with nearly every other language) can run external code, and thus this should not be an overwhelming task. We are pleased to say that our feasibility prospects have shown great promise, and we are excited to continue to develop this project so that it will serve our client and users well.