TECH FEASIBILITY

Nov. 4, 2021

SPONSOR: DR. VIACHESLAV FOFANOV
MENTOR: VOVA SARUTA

JOSEPH DOMABYL V   ANDREW LIDDELL   JUNJIAN YIN   DANIEL DRAKE

Magisters

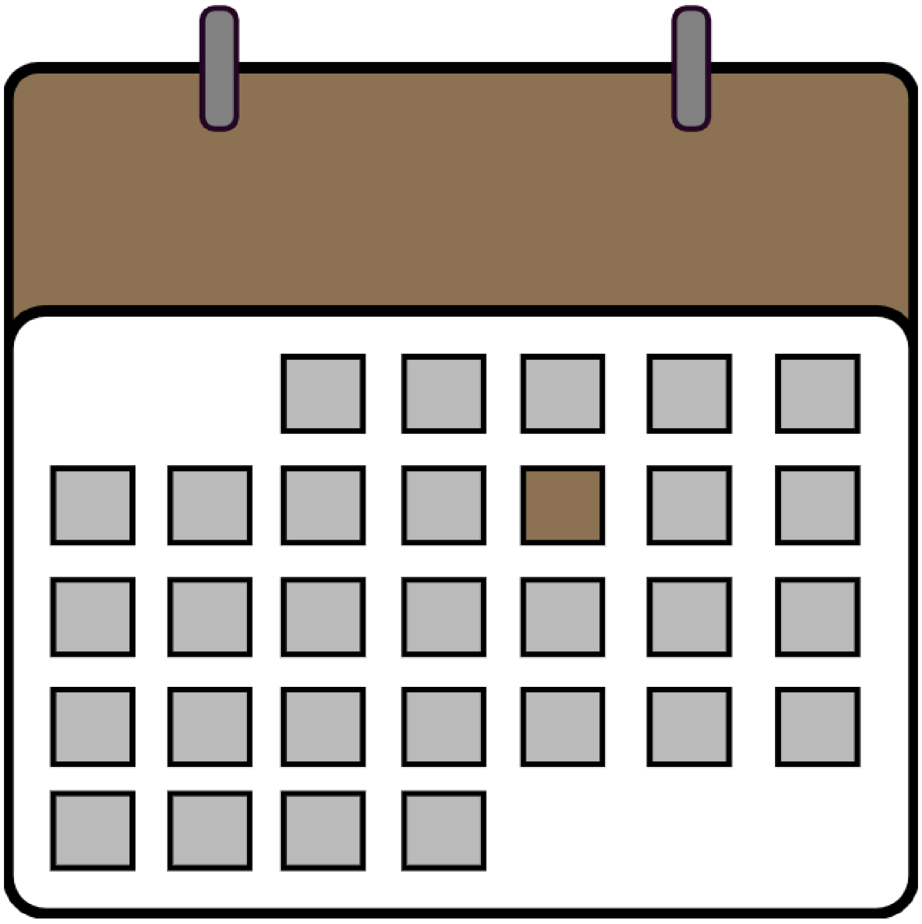

TABLE OF CONTENTS

1 - INTRODUCTION

Scheduling large groups of people can be a challenging task. Consider a scenario where you've spent weeks, if not months, crafting the perfect schedule for potentially hundreds of different people. You've paid attention to their strengths, their weaknesses, their weekly availability, etc. Then, you get a call from one of your TA's telling you that they cannot make the 2 pm timeslot on Friday that they said they could. You now need to go in, manually move them around, and find them a new spot. However, by doing this, you may now need to move fifteen other people just to make the schedule work. This is a persistent issue that plagues many university departments semester after semester, particularly regarding the scheduling of teaching assistants (TAs) and graduate teaching assistants (GTAs) for lab positions. For the rest of this document, we will be referring to both TAs and GTAs simply as TAs unless specified otherwise.

The current solution used by our client, Dr. Viacheslav Fofanov, is to use an Excel spreadsheet and manually assign these TAs to where they need to be. Dr. Fofanov is the associate director for the School of Informatics, Computing, and Cyber Systems (SICCS) at Northern Arizona University (NAU). Specifically, Dr. Fofanov coordinates GTAs to teach labs and TAs to assist their graduate counterparts. Dr. Fofanov currently spends an estimated 20-30 hours creating the schedule but emphasized that this number is largely dependent on a multitude of other variables. This might include elements such as how long it takes for students to respond with their personal schedules, how long it takes to create the table depending on the number of TAs and the number of labs, or how long it takes to cross-reference Excel cells in order to ensure an accurate schedule. Perhaps one of the most important aspects of this problem is that it is persistent and has the potential to be troublesome *throughout* each semester, not just at the beginning. That is, a TA might unexpectedly leave their position and their spot would need to be filled.

The team's aim is to provide a comfortable working environment in the form of a web application that automates the scheduling and management of TAs. The team's primary focuses include:

- GUI: An intuitive and functionally stable environment that renders all the information necessary for lab organizers to view the current standing of TAs in the system in an appealing format such as a calendar view.
- Automation: Provide an organized display list of potential TA assignments for an entire lab offering.

- Optimization: Optimize the placement of TAs by assessing their qualifications, time availability, and student feedback to ensure they are assigned to their most appropriate spots.

Throughout the rest of this document, the team will be providing information regarding the technological feasibility of this solution. This document will discuss technological challenges, an analysis of each proposed technology, possible alternatives, and how well the technology can be integrated.

2 - TECHNOLOGICAL CHALLENGES

For the success of the application, the Magisters will need to overcome a set of technical challenges. The team has developed a list of seven different problems that they will encounter in the foreseeable future. In this section, the document will begin with a brief description of each potential problem and will later delve deeper into the "Technological Analysis" section.

2.1 - Web Framework

The team will need a web framework that will provide fast response times, security, and a foundation for every other design decision made. The chosen database framework will need to integrate well with the web framework, as it must run efficiently on the hosting server. This will need to be one of the first decisions that the team makes.

2.2 - Graphical Interface

One of the most important problems conveyed to the team by Dr. Fofanov is the graphical representation of the system. The team will need to provide a reactive and visually appealing interface with great emphasis on the design of the dashboard and calendar. The backend of the interface must be modifiable so that further design decisions can be swiftly integrated.

2.3 - Data Storage

The team will need a fast and lightweight database system capable of servicing a single department. The current plan is to use one database per department, so the team will need no more than a couple thousand entries in the database. The database will need to store enough information regarding each TA, including name, student ID, and availability. The database will also need to integrate with the chosen web framework.

2.4 - Admission

The team needs to provide a platform for account creation that integrates well with the database. The team's options include either scraping data from a Google Form or have the Lab Organizer send generated links to the TAs they wish to schedule. Following the latter option will require the TAs to create their own accounts.

2.5 - Web Hosting Service

The web application will need a stable server to run on. It must be a lightweight and secure server capable of efficiently responding to the users when requested.

2.6 - Security & Permissions

The application will need to provide secure user authentication in the form of a "login" to our application. Providing a format for secure passwords and subsequently user-based permissions is of great importance. For example, users of the "TA'' role should not be able to modify their own or anyone else's schedule. This implies TA's should not be able to log in at the "Lab Organizer" level.

2.7 - Penalty System

To aid in the process of optimization, the team will need to develop a constraint system to be applied to TAs within the database. For example, we would not want a TA that is unavailable on Fridays to be scheduled on that day. In this case, we will supply a negative "penalty" to them, thus eliminating the possibility of a Friday scheduling. Furthermore, if a TA has not TA'd for a specific lab before, they may be given less priority than a TA that has, thus a different penalty value.

2.8 - Summary

The above has defined seven different technological challenges that the team needs to face. After examining these challenges at a deeper level, the team has compiled a list of frameworks and products that might act as possible solutions.

3 - TECHNOLOGY ANALYSIS

In order for us to truly understand the problem and be able to develop a clean and high-quality solution, we are deeply investigating each individual challenge. Throughout this section, we are outlining each technological challenge, providing potential candidate solutions, and explaining our reasoning for our decisions. It should be noted that we have combined technological challenges that overlap with each other. For example, "security and permissions" will be discussed within both sections 3.1 and 3.4, web frameworks and web servicing respectively.

To better understand our decision-making process, we have created a metric system for each technological challenge. Each criterion will be defined in each subsection relevant to the solution we are looking for and will be placed on a scale of **one to five**, except in the case of web frameworks, which we will elaborate upon further.. Each subsection will include a summary in which we will examine the ranking of each alternative and use these metrics to define our chosen solution.

3.1 - Web Framework

One of the key requirements for this project identified by our client is a refined dashboard and UI of the web application. This is what our customers/clients will be facing consistently throughout all of the deployment. We want to choose a framework that will nicely fit all categories needed to make an easy and efficient web interface to work with. It must look attractive, run fast and be secure enough so that all users' sensitive information stays confidential. So, to choose a valid web framework, the team had to examine:

- **Performance**: When looking at performance, we will consider the speed, efficiency, and responsiveness of the front-end framework. We will want the user to interact with the website with low waiting times while also maintaining a professional and refined appearance. However, this isn't a criterion of great relevance considering the application's small scale. Therefore, we will weigh the score of this criterion out of **3**.
- **Applicable Security**: We will want to protect sensitive information with potentially large numbers of technologically savvy users. We will be examining what security solutions each framework will have out of the box. This criterion will be of great importance to us, so we will be weighing this score out of **7**.
- **Development Flow**: When looking at the development flow, the team will want documentation to be readily available and the implementation process to be as simple as possible. This will allow for timely development in the context of a time-limited capstone class,

but also greater future development possibility. The development speed and applicability of the web framework is essential to ensure that the product is ready to deploy as soon as possible. It will be weighted out of **7** as well.

With the above criterion in mind, the team narrowed the options to Angular, Vuejs, and Django. Next, the document will provide an in-depth analysis of each framework, beginning with Django.

3.1.1 - Django

Django was one of the first options considered by our web tech, Andrew. After dabbling within the framework on his own time, he remarked that this framework will nicely fit our needs in many aspects. Many of the components for this application will come out of the box when we use Django and will be able to utilize many of the built-in features without it being considered bloated. Django is a free and open-source web framework developed by the Django Foundation. It was released in 2005 and has steadily grown in popularity since then. To this day it remains one of the most popular frameworks due to its great scalability and "batteries included" philosophy.[1]

To analyze Django, we will first examine its **performance**. Django's performance is fairly lackluster.[2] As it is commonly known, Python is certainly not the fastest or the most optimized language. Django's design is extremely monolithic and tightly coupled, which means that "fat", or "bloat", cannot be trimmed off. This is a considerably large framework and may be a little large for this project. However, the ability to design good code architecture and properly optimize the program may offset these drawbacks. We should also note that we would be utilizing the majority of Django's features, which may counteract the issue of bloat in the first place. Overall, **1/3** will be our designated score for the framework's performance.

The next criterion to examine would be **applicable security**. In terms of security, Django is absolutely packed with features out of the box. These include protection against SQL Injections, CSRF, Clickjacking SSL, HTTPS, and Host header validation as well as many others. In terms of security, Django is certainly the most prepared and will accomplish many of the security headaches web developers have to hassle with without much configuration. Overall a score of **7/7** will be given to the security criterion of Django.

[1] https://www.benchmarkit.solutions/lets-understand-the-pros-and-cons-of-using-django/
[2] https://www.netguru.com/blog/django-pros-and-cons

Lastly for this framework, we will look at its **development flow.** Django is extremely well documented with a bustling community of online users.[3] Further, with Django's aforementioned "batteries included" philosophy, a huge load of the work is completed when you set up an environment. With a built-in user/superuser system along with an easily manipulated component module, two major aspects of our project will be done. Also, having libraries nicely integrated with whatever backend database we use will make it greatly adaptable. The GUI development contains heritability and HTML injections, which make the process of designing the interface lightning-fast. A score of **7/7** will be given to Django for its development flow.

3.1.2 - Angular

We have directed our research to look at purely front-end frameworks that are extremely popular and are widely used in the industry. Considering the other choices, we decided to take a look at Angular. Angular is a massive open-source web framework developed by Google. It was initially released in 2016 and has grown to be the second most popular web framework in 2018. The framework features TypeScript-based development and is designed for major enterprises to have an extremely dynamic front-end web application. We considered this framework due to its great ability to make interactive and nice-looking web applications.

We will first look at how well Angular scores in terms of **performance**. As stated above, Angular is a massive framework designed for enterprise applications. This could cause a huge amount of lag for smaller projects. However, if the SBA is relatively simple, this shouldn't be a problem. Angular features asynchronous programming due to it utilizing RxJS as its major development language.[4] This can be utilized as long as the developer knows how to properly use it. Lastly, it features hierarchical dependency injection, which allows for the dependencies to run in parallel, greatly increasing performance. Angular will receive a **2/3** for its score in terms of performance.

Next, we will look at Angular's **applicable security** attributes. Angular's documentation does have a few methods for implementing security features such as cross-site scripting and sanitization as well as cross-site request forgery. However, the security is not as conveniently included out of the box. Security measures must be implemented by the developers themselves. The documentation for security is also significantly less than that of the other frameworks we have glanced over. The security of Angular is certainly decent, but not the best. This means we will score Angular's security attributes a **5/7.**

[3] https://hackr.io/blog/what-is-django-advantages-and-disadvantages-of-using-django
[4] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

Lastly, we will look at the **development flow** for Angular and how that compares with the others. The learning curve of Angular is extremely steep, which means learning how to effectively utilize this framework correctly may take months. Many aspects of the app will have to be built from the ground up, which will take even more time. None of the team is familiar with TypeScript, so one or two of us will have to learn a new language. The last con is notably difficult as the CLI documentation of Angular is extremely lacking. However, due to the mechanics of TypeScript, the code is clean, readable, and easily reproducible. Other sources have also noted that it has great unit testing ability and maintainability. Weighing in all of the above, we think a score of **5/7** would be the most appropriate for this framework's development process.

3.1.3 - Vuejs

When considering the options for our project, we noticed that this project is relatively small in comparison to other projects needed by other corporations. While we did research front-end frameworks that can scale down to our level, we wanted to check out a framework that is designed for a project of our size. That's where we came across Vuejs. It is an extremely lightweight framework that focuses on componentization and manipulating HTML pages using directives. It was developed and released in 2014 by Evan You and is fairly popular with many companies despite it being more bare-bones than other frameworks on the market.

How Vuejs will score on **performance** should be fairly obvious. Its lightweight design with its built-in directives are extremely fast and significantly more efficient than the other frameworks.[5] Considering the drawback of many things needing to be built from the ground up, its performance will be its greatest strength in comparison to the others. Because of this, it is widely considered to be one of the best front-end frameworks if it's used in the right hands. We will give this a performance score of **3/3**.

We will next examine how well it will be able to implement **applicable security**. The security of any framework this barebone will be most certainly dependent on the developer. Most enterprises that use Vuejs will implement another library for security alone.[6] There is some documentation for the best security practices for beginning developers. However, most of the features will be left up to us. Unfortunately, since this team is not very experienced in managing security in web applications, we will have to score this criterion for Vuejs very low. That being said, if we as a team were more experienced with security then it may have been different. We will give Vuejs a security ranking of **2/7**.

[5] https://ronakataglowid.medium.com/pros-and-cons-of-the-vue-js-framework-8015dcbc05ef
[6] https://ddi-dev.com/blog/programming/the-good-and-the-bad-of-vue-js-framework-programming/

Last but not least, we will examine its **development flow**. Most of our team is at least familiar with JavaScript. However, the approach to development is blocked by a language barrier. The developer for this framework is not a native English speaker, which may leave a lot of ambiguity in the usage of framework methods. Not to mention we will need to build almost everything from the ground up which is too costly for the team's time. However, the code is clean, readable, reusable, and great with unit testing. These properties are expected from a barebones framework. From the information stated above, we will designate a score of **4/7** to Vuejs in terms of development.

3.1.4 - Chosen Approach

Now, we can broaden the focus into the big picture. As we have previously discussed the pros and cons of each framework, we can turn our attention to Table 1:

Table 1. Web Frameworks

	Django	Angular	Vuejs
PERFORMANCE	**1**	**2**	**3**
SECURITY	**7**	**5**	**2**
DEVELOPMENT	**7**	**5**	**4**
TOTAL SCORE OUT OF 17	**15**	**12**	**9**

All three of the considered frameworks scored relatively high. The best all-around highest-scoring framework happened to be Django, as it absolutely shines in terms of its development process as well as performing decently in the other fields. Due to this scoring, we have chosen **Django** as the best option to suit our needs. The out-of-the-box philosophy will be a great fit for speeding up the development flow in terms of applicable web security and database integration which will allow us to greatly focus on the UI aspect of the application.

To prove that the decision will work, the team will set up our out-of-the-box Django environment and make sure our chosen database can nicely integrate into it. The team will also render out some HTML elements and pages to make sure that the navigation of the application will be sound. Then, the team will test the functionality of the sysadmin module to ensure it will suit their needs.

3.2 - Data Storage

To create our application, we will need to utilize some form of data storage. This will include a database framework and, in most cases, a location to store that data. We will be focusing on two "types" of data that we need to keep track of. First, we will need to store identification information for each TA. In the early stages, we believe that this will include their name, their student ID, their weekly availability, and their personal TA experience. In further stages, this may include extraneous elements such as a date of birth, a photograph, or a short biography. The second "type" will be the semester information. This will include elements such as year, labs being offered, and the assignments of TAs. That being said, the most important aspect of this decision will be the framework itself.
To choose a valid database, we know we need to look into four different criteria; speed, accessibility, integration, and cost. All of these criteria will be ranked out of five:

- **Performance**: How quickly can we access data from the database? How long will it take for even the largest queries to complete? How well will the database scale with the addition of new data?
- **Accessibility**: How simple is it to perform queries? Will it take extreme amounts of developer time to learn the library? Where will the data need to be stored on the server?
- **Integration**: How well will the database framework integrate with our chosen web framework?
- **Cost**: Will we need to purchase a separate server in order to use the database? Financially, how much will the database cost to run?

With that criterion in mind, we have decided to investigate three different database frameworks; Redis, SQLite, and MongoDB. Next, we will provide an in-depth analysis of each database, discussing the viability of each with respect to the predefined criterion.

3.2.1 - Redis

Redis is an open-source database framework that runs within the memory of its host machine. Running in-memory means that Redis is *extremely* fast, and round trips to the disk aren't necessary, This is its primary selling point. Redis is also a key-value database rather than a relational one. This means that each value is given a key and is subsequently only accessible by that key. Our group initially began looking into Redis as an option by recommendation of our mentor, Volodymyr (Vova) Saruta.

Redis was initially released on May 10th, 2009 by Salvatore Sanfilippo, who has since left the project[7]. It is written in C and is used by a multitude of different projects such as Twitter, GitHub, StackOverflow, and more[8].

Examining each of our desired criteria, we will begin with **performance**. The performance capability of Redis is its shining quality. Due to the dataset being stored in-memory, Redis is one of the fastest database structures commercially available. However, this also means that with scale, Redis can become very expensive. Depending on how large the database becomes, Redis could end up taking a large portion of the available RAM. According to our client, the entire dataset to accommodate his specific department will remain roughly fixed, around 1800 data entries. For these reasons, Redis will be receiving a score of **5/5** within the performance metric within the scope of our application.

Next, we have **accessibility**. Redis comes with a breadth of available commands, all of which are documented on their website[9]. Further, their website also provides a lightweight tutorial to learn the absolute basics[10]. Thus, we believe that Redis will not have a difficult learning curve within the scope of our project. However, persistence with Redis requires special configuration. This means that the team will have to develop a system using predefined Redis tools[11] in order to decide where and how to store the data outside of the memory. For these reasons, Redis will receive a **4/5** within the accessibility metric.

In terms of **integration** within our chosen web framework, Django, we have a specific tool that is perfect for the job; an open-source package called *django-redis*[12]. Furthermore, Redis also supports the Python programming language, the same language that Django is written in. In terms of running Redis on our chosen web servicing platform, Redis can be installed on any server, providing no issues for integration. For these reasons, Redis will be receiving a score of **5/5** within the integration metric.

Finally, we have **cost.** The cost of Redis is minimal. However it should be noted that there is a distinction between its open-source project and its enterprise[13]. Utilizing Redis the open-source project, there is no cost whatsoever. We would run Redis on our chosen web server, for free. There are

[7] https://en.wikipedia.org/wiki/Redis
[8] https://redis.io/topics/whos-using-redis
[9] https://redis.io/commands
[10] https://try.redis.io/
[11] https://redis.io/topics/persistence
[12] https://github.com/jazzband/django-redis
[13] https://redis.com/blog/becoming-one-redis/

options to run Redis on explicit Redis servers, offered by the company. This will not be necessary for our application, thus Redis receives a score of **5/5** for the cost metric.

3.2.2 - SQLite

SQLite is an SQL database engine. It is known for being small, fast, and self-contained. Due to SQLite being an SQL database, this means that it is relational, not key-value like Redis. This provides for more robust data storage, but it does make queries more complex. SQLite was brought to the attention of the team members that have taken CS345, Database Systems, at NAU.

SQLite was initially released on August 17th, 2000 and is written in C. It was designed by D. Richard Hipp while working for General Dynamics[14]. SQLite is the most widely used database engine in the world and can be found almost anywhere you look. For example, it is a part of every mobile phone, every Windows 10 machine, every web browser, and more[15].

Now we can begin ranking our desired criteria. Beginning again with **performance**, SQLite is also known for its speed and its small size. For example, SQLite by itself is approximately 35% faster than the filesystem and uses 20% less disk space[16]. However, SQLite reads and writes data directly to a disk, making it slower than Redis. In terms of scalability, SQLite is not explicitly designed to scale. However, SQLite is applicable within the scope of our application. For example, SQLite themselves claims that their own website receives about 400 thousand to 500 thousand HTTP requests per day, 15-20% touching the database, and there are no problems [17]. For this reason, SQLite will receive a score of **4/5** in the performance metric, particularly due to it being slower than Redis.

Regarding **accessibility**, it is important to remember that SQLite is a relational database. This implies the usage of primary and foreign keys and the usage of complex queries such as JOIN, UNION, or CROSS. This will increase the learning curve for SQLite, requiring more time to be spent just to initialize the database. However, because SQLite is self-contained, we will not be requiring an outside server to utilize it. This means that all data will be stored on a local disk relative to the web service we decide on. For these reasons, SQLite will be receiving a score of **3/5** in the accessibility metric.

[14] https://en.wikipedia.org/wiki/SQLite
[15] https://www.sqlite.org/mostdeployed.html
[16] https://www.sqlite.org/fasterthanfs.html
[17] https://www.sqlite.org/whentouse.html

In terms of **integration** with Django, SQLite is actually the default configuration, which also implies that SQLite has support for Python. As we have previously stated, SQLite is entirely self-contained. Thus, we would not need to worry about integration issues onto our chosen web service, SQLite can be simply installed. For these reasons, SQLite will be receiving a score of **5/5** within the integration metric.

The **cost** of SQLite does not exist. SQLite is completely free and can be deployed anywhere, private or commercial. For this reason, SQLite will be receiving a score of **5/5** for the cost metric.

3.2.3 - MongoDB

Turning our attention away from a relational SQL database, we can now examine our final contender: MongoDB. MongoDB is considered a NoSQL-document database. A document database is one that is nonrelational, it is designed to store and query data as documents, like JSON[18]. This allows the data to be more human-readable and provides more flexibility and modularity. For example, documents can be considered hierarchical, allowing them to move with our application's needs.

MongoDB was initially released on February 11th, 2009 and is written in C++, JavaScript, and Python. It was designed by a company known as 10gen, now known as MongoDB Inc[19]. Multiple companies deploy MongoDB in their tech stack such as Sega, Google, and Verizon[20].

To begin our rankings for MongoDB, we will look at **performance**. Recalling that MongoDB is a document based database, it is designed for massive queries and storage. This, in turn, means that as MongoDB scales upward, it also becomes slower. However, within the scope of our application, the sheer size of MongoDB is unnecessary. Thus, MongoDB will receive a rating of **3/5** for the performance metric of our application.

Looking into the **accessibility** of MongoDB, the queries are performed in a very similar manner to that of a relational database. However, MongoDB does not support the same operations semantically. For example, in place of JOINs, MongoDB might use an operation such as "lookup"[21]. This will increase the learning curve for the team, as nobody has experience with document databases.

[18] https://aws.amazon.com/nosql/document/
[19] https://en.wikipedia.org/wiki/MongoDB
[20] https://www.mongodb.com/
[21] https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/

MongoDB also stores all data on a disk within a predefined MongoDB cloud server, meaning it will not run locally outside of lower "tiers". This will also influence the cost factor of using MongoDB. For these reasons, MongoDB will receive a score of **2/5** regarding the accessibility metric.

To **integrate** MongoDB with Django, our chosen web framework, we have three tools available: *PyMongo,* *MongoEngine*, and *Djongo*. All three of these options provide an effective connection to Django, though *PyMongo* is the official and preferred way to connect[22]. MongoDB would not be running locally on our web service, it would be running on it's own dedicated server. For these reasons, MongoDB will be receiving a score of **4/5** within the integration metric.

Finally, examining **cost**: MongoDB does provide limited service for free. There are two free options, MongoDB Atlas and MongoDB Community Server. Atlas is a cloud database, though there is a limited amount of storage available for the free tier. The Community Server offers a similar solution, though it runs locally. However, it also only provides limited storage. That being said, there exists an option to run a shared server, though it is recommended only to use this as a sandbox environment and is not entirely secure. A dedicated server will cost $57 per month and a serverless setup will cost $0.30 for the first 5 million reads per day[23]. For these reasons, MongoDB will receive a score of **1/5** for the cost metric.

3.2.4 - Chosen Approach

Now, we can broaden our focus into the big picture. As we have previously discussed the pros and cons of each database, we can turn our attention to Table 2:

Table 2. Data Storage

	Redis	SQLite	MongoDB
PERFORMANCE	5	4	3
ACCESSIBILITY	4	3	2
INTEGRATION	5	5	4
COST	5	5	1
TOTAL SCORE	19	17	10

22 https://www.mongodb.com/compatibility/mongodb-and-django
23 https://www.mongodb.com/pricing

It is immediately apparent that Redis excels in every considered metric. Though it is tied with SQLite in terms of integration and cost, we believe that the speed and accessibility of Redis allows it to outperform SQLite, especially for our application. MongoDB is far too large for a project of this scale and thus will no longer be considered. Due to this information, we have chosen **Redis** as the best option to suit our needs, with SQLite as a close second. We believe that our application needs to be fast and modular. Redis provides us with both of those requirements quite simply: Redis runs in memory and has Python support, allowing for exceptional speed and simple implementation.

Finally, to prove that our decision will work, we will establish a Django environment to be used for testing purposes. Using this environment, we will initialize a Redis database, populate it with sample data, and perform unit tests. This will work hand-in-hand with our Django prototype, allowing the team to test multiple facets at once. The team will provide a demonstration of the technology at a later date.

3.3 - Admission

In the team's scheduling application, TAs will be able to view their time schedule, update their profiles, and update their time availability. The system will be designed to only give one account to one TA at a time. Otherwise, this may cause the following problems: providing a large number of irrelevant accounts to the database, increasing the potential risk of malicious account creation, and holding duplicate accounts that are related to one TA. Obviously, those problems increase the database size and decrease its security, which also leads to extra work for the lab coordinator. To make the app more effective and secure to use, the team needs to come up with a solution that allows TAs to not only create their own accounts but also ensure that only one account is created for each TA. The proposed solution has four indicators:

- **User-friendly:** How easy will it be for users to create an account under the chosen solution? Will the user have difficulty learning how to create an account?
- **Accessibility:** Will TAs be able to create duplicate accounts under the solution? Will the solution provide non-useable account information? Will the solution connect to the system in an efficient fashion?
- **Privacy & Security:** Will this solution cause user privacy leakage during the process? Will it prevent malicious behavior?
- **Technical Difficulties:** How easy is it to implement the solution?

Under these circumstances, there are two possibilities: utilizing pre-existing Google Forms or using the application to send personalized links and forms to the TA's. This document will now provide a detailed analysis of both options.

3.3.1 - Google Form

Google Forms is a survey administration software managed by one of the largest corporations in the world. It's free to use and acts as a web-based application.[24] The app allows users to create and edit surveys online while collaborating with other users in real-time. The collected information can be automatically injected into a spreadsheet. The team is considering this solution because an organized list of accounts would be convenient. The accounts will be located in a file changeable by multiple people, which is somewhat similar functionality to a cloud-based collab document, such as Google Docs. The Google Form will be modified by the recipient of the private link, which would prevent any extraneous account creation.

Since Google Forms are widely used for multiple different purposes, a Google Form will be a highly **user-friendly** solution. There is a minimal learning curve, thus the team have given Google Forms a score of **5/5** in terms of being user-friendly.

In terms of **accessibility**, Google Forms doesn't have any particular use-cases within the scope of the project. There are, however, two potential options: conversion into an Excel spreadsheet or scraping the HTML itself. Both solutions to this problem require extraneous programs to do this automatically. For this reason, Google Forms will be receiving a score of **2/5** for accessibility.

The **privacy & security** of Google Forms has two elements: the security of Google themselves and the privacy of the link being sent out. The only issue with security using Google Forms could be a data breach upon Google. However, we would not be asking users for sensitive data via the form, thus the risk is negligible. In terms of privacy risks, the link to the form could arrive in an incorrect inbox, for example, if a TA sends their link to a friend. However, since only one account can be created per link, this would not threaten their privacy. For these reasons, Google Forms will receive a score of **4/5** for privacy & security.

The **technical difficulties** of setting up a Google Form is virtually nonexistent. The technology already exists as such the only challenge would be data extraction. For this reason, Google Forms will receive a score of **4/5** for technical difficulties.

[24] https://en.wikipedia.org/wiki/Google_Forms

3.3.2 - Custom Form Creation

The next option involves manually creating a private link sending it to recipients in order to sign up. This method is used in most apps' account creation verification. The team is considering this method as they are very familiar with email confirmation through links.[25]

Examining the **user-friendliness** of this approach, this is entirely up to our implementation. The developers will be responsible for designing and sending out personal links and every decision can be customized specifically for the application. Furthermore, the team will also be able to construct the sign-up page with further instructions, allowing for a deeper understanding of creating an account. For these reasons, the team will be giving themselves a **5/5** for user-friendliness.

In terms of **accessibility**, TAs will create their accounts directly within the application. From there the team would be able to plug the data immediately into the backend system. For this reason, the team will be giving themselves a score of **5/5** for accessibility.

Next, we have **privacy & security**. In this scenario, TAs will follow the link to sign up and fill out the information as needed. Then, the account will be created and stored in the backend. No duplicate account will be created since every link can create only one account and will expire after a set time period. The link is private to a specific TA, barring the possibility of the link being passed around by the TA themselves. This means that the link's security is dependent on the TA, though the account is pre-created and the link simply provides access as well as a temporary password.. Thus, the team will be giving ourselves a score of **5/5** for privacy and security.

The primary issue with this approach will be the **technical difficulty**. For this to function properly, the team must develop account-creation pages with unique links for each TA. Then, they would need to send those links out to each individual TA separately. Finally, the developers would need to set a timer on those web pages after expiration to avoid cluttering the storage on the server. All of the stated issues will generate more work and require a more technical understanding of the chosen web framework. For these reasons, we will be receiving a score of **2/5** for technical difficulty.

[25] https://firebase.google.com/docs/dynamic-links/use-cases/user-to-user

3.3.3 - Chosen Approach

After our research and assigning scores to each candidate, we can look at Table 3:

Table 3. Admission

	Google From	Custom Form Creation
USER-FRIENDLY	**5**	**5**
DATA USABILITY	**2**	**5**
PRIVACY & SECURITY	**4**	**5**
TECHNICAL DIFFICULTY	**4**	**3**
TOTAL SCORE	**15**	**18**

Examining these results, it is clear that developing our own internal system for account creation beats using a Google Form in nearly every respect. The higher technical difficulty is a good tradeoff for the benefits of the custom-created form controlled by the developers. The client has asked The Magisters to provide an environment in which most of his work will be automated. If the team chose to use Google Forms, automatic service isn't entirely guaranteed. Due to this information, the team can confidently say that creating our own system and **custom form creation** will be the best option.

Finally, to prove that the decision will work, the team will create a system within the Django environment to test the ability to dynamically create links to be sent out to individual TAs. The developers will also prototype the idea of sending links directly from the website, eliminating the manual work for the client to await responses within his own inbox.

3.4 - Web Servicing

Next up, we will examine the problem of web servicing. The application will need a safe and secure servicing platform to deploy. An efficient web server within the scope of the application will require a number of different elements. For example, the server will need to be responsive, durable, and be protected against failure. The developers will need to be able to access the server, and users must successfully connect to the website. The server must also be secure, doing everything possible to avoid unauthorized access. Finally, the cost of the server must be calculated in order to maintain it over extended periods of time. For these reasons, the document have defined a set of four criteria:

- **Performance**: How quickly can the server respond to requests? How will the server respond to changes? How does the chosen service provide failure protection?
- **Accessibility**: How easy will it be for the end-user to access the web service? How easily will the team be able to make changes?
- **Security**: How secure is this web service? Does the service provide its own form of authorization?
- **Cost**: How much does the web service cost to maintain? Does the platform provide any extra services, and if so, how will that cost be calculated?

Having defined these metrics, the document can begin an in-depth analysis of three different candidates; AWS (Amazon Web Services), hosting a VPS (virtual private server) via NAU, and using a third-party provider, Hostwinds.

3.4.1 - AWS

AWS is a subsidiary company of Amazon, which was created to provide an on-demand cloud computing service. AWS offers over 200 different products[26], each with specific properties designed to aid any use case. AWS was first created in 2006 and has since been used by many companies for their provided service, one such company being Pfizer. The group decided to look into AWS as it is widely popular and was mentioned by Dr. Fofanov as a possibility.

In terms of **performance**, AWS is largely dependent on which services are chosen to deploy. There are a plethora of different front-end and web application services[27] that would allow the team to quickly get the product up and running. Furthermore, the servers run 24/7, are very responsive, and provide a network of different servers automatically allocated in the event of a single server failure. For these reasons, AWS will receive a score of **5/5** for performance.

AWS is known for its **accessibility** given that it is designed for consumers. It also allows domain name registration as part of the service. The team would be required to create an AWS account shared between the four of them in order to access the files/backend. Accessing the website is similar to accessing the source files to any other. Given its ease of access for both the team and the end-user with minor configuration, AWS is ranked **4/5** in terms of accessibility.

[26] https://en.wikipedia.org/wiki/Amazon_Web_Services

[27] https://aws.amazon.com/products/frontend-web-mobile/?nc2=h_ql_sol_use_ms

For **security**, AWS is packaged with secure servers by default. These servers would protect against common web-based attacks as well as provide authentication services such that nobody outside of the team would be able to access them. For these reasons, AWS will receive a score of **5/5** for security.

In terms of **cost**, AWS provides a "pay as you go" architecture. This means that the server can be configured to only charge the account based on up-times, services requested, and packages deployed. AWS also offers free options, which include free trials, 12 months free, and permanently free options[28]. Utilizing the non-free options, the cost would scale with usage. Thus, AWS will receive a score of **2/5** with respect to cost.

3.4.2 - NAU VPS

NAU, being a large university, maintains and provides its own series of networks. For example, NAU runs its own website as well as their own private networks, such as the CEFNS Linux server. Asking NAU to provide a VPS was a decision made by the group in an effort to keep our application localized, thus modifiable for our client.

The **performance** of an NAU VPS will largely be dependent on the traffic of a large university. For example, the performance may drop during busier times of the year, but may rise during slower periods. The performance may also be defined by the level of clearance given to the team by the university. Given that it is difficult to gauge the performance of NAU's network, the group will assume that an institution such as NAU maintains their servers. Thus, NAU will receive a score of **2/5** with respect to performance.

In terms of **accessibility**, it may be difficult to maintain access to NAU's network for extended periods of time. Given the awkward accessing routine of the team's capstone website, the team can assume a similar scenario with a VPS. In terms of the end user accessing the website, we would be given an extension of NAU's network, meaning they would not have our own domain, thus reducing the portability of the final product. For this reason, NAU will receive a score of **2/5** in terms of accessibility.

Regarding **security**, the team can assume that NAU as a large institution, provides proper security measures in accordance with the modern standard. It should also be noted that in order for anyone to

[28] https://aws.amazon.com/free/

access the network off-campus, they must be connected to a VPN. For this reason, NAU will receive a score of **4/5** within the security metric.

Finally, looking at **cost**, we will assume that an NAU VPS would not cost anything for the team nor for our client. For this reason, NAU will receive a score of **5/5** regarding cost.

3.4.3 - Hostwinds

Finally, the team will be considering a third party option for the web servicing needs. Hostwinds is a VPS hosting service founded in 2010 by Peter Holden. It is a private company that offers a variety of different VPS options for a multitude of different operating systems. Hostwinds also provides a secondary option in the form of dedicated web hosting, which comes with different options. The team decided to investigate Hostwinds as most of us have experience with them in the past.

Regarding **performance**, Hostwinds provides different service-levels based on subscription tier for VPSs. For example, at their lowest tier, we would receive one CPU and one terabyte of bandwidth. At the highest level, they will give sixteen CPUs and nine terabytes of bandwidth. In addition, all Hostwinds plans come with 99.999% uptime, custom ISOs, and solid state drives[29]. For these reasons, Hostwinds will receive a score of **4/5** in terms of performance.

In terms of **accessibility**, Hostwinds provides the ability to SSH directly into the server. This means that, with the proper encryption, every member of the team would be able to access the server from anywhere. Thus, the team would be able to push changes at any time. Further, the servers do come with a guaranteed 99.999% uptime, meaning they would remain accessible to end users for a majority of the time. For this reason, Hostwinds will receive a score of **5/5** within the accessibility metric.

For **security**, Hostwinds provides two options in terms of VPS platforms; managed and unmanaged. Managed servers come with security options and service via Hostwinds while unmanaged servers do not. Using a dedicated web host, the website would also receive free SSL certificates. However, security configuration would still be required for either option. For these reasons, Hostwinds will receive a score of **3/5** for security.

For **cost**, Hostwinds ranges anywhere from $6 a month to $300 a month. Within the scope of the application, we expect to only require roughly $8 a month for the advanced web hosting service, which

[29] https://www.hostwinds.com/vps/linux

provides unlimited bandwidth as well as unlimited disk space. As Hostwinds is not free, it will receive a score of **3/5** for the cost metric.

3.4.4 - Chosen Approach

Now, the document can assess the scores of the above criterion. As we have previously discussed the pros and cons of each system, we can turn our attention to Table 4:

Table 4. Web Servicing

	AWS	NAU	Hostwinds
PERFORMANCE	**5**	**2**	**4**
ACCESSIBILITY	**4**	**2**	**5**
SECURITY	**5**	**4**	**3**
COST	**2**	**5**	**3**
TOTAL SCORE	**16**	**13**	**15**

Based on this information, AWS has received the highest total score, winning with respect to performance and security. The primary downside to using AWS is the cost, though a project of this scale will not grow to exponential size, thus reducing the overall cost of AWS. Hostwinds is a very close second, with the determined accessibility ranking higher. However, the team believes that the overall benefits of AWS outweigh the personal use-cases of Hostwinds. The team will no longer be considering an NAU VPS as an option due to our needs of portability, and they are not permanently connected to the institution. Due to this information, **AWS** will be the best service to host the application.

Finally, to prove that this decision will work, the team will establish an AWS environment to deploy a Django prototype, also utilizing the Redis prototype. Further information will be elaborated upon within a technical demo.

3.5 - Penalty System

In terms of technical challenges, the penalty system will be the most logic-intensive. The team will be required to develop an intelligent algorithm that will sort TAs based on multiple different constraints. For example, we will need to sort TAs based on their strengths, and weaknesses, and weekly availability. To accomplish this, the team has decided to investigate open source solutions to avoid reinventing the wheel. Within the scope of the project, the group will be using a penalty system that assigns integer values, each with a different weight, to different constraints offered by the TA. For example, if the TA cannot work between the hours of 2 pm and 4 pm on a Thursday afternoon, they might be assigned a value of "-1", ensuring that they will not be scheduled during that time. The team has determined two different metrics to determine the best solution:

- **Implementation**: How costly will the algorithm be to implement? Will it take the team too much time?
- **Speed:** Will the algorithm provide timely output? Will it be feasible to run the algorithm multiple times with different constraints?

Using those two metrics, the team has realized two methods to approach this problem. Firstly, the team has found a discussion of an optimization algorithm from a website hosted by Cornell University[30] that attempts to solve a problem similar to the projects' own optimization issue. To start off, the document will delve into the details of each. Second, the document will weigh the pros and cons of developing a custom algorithm internally. It should be noted that the team will keep their options open, should any other algorithms be brought to their attention in the future.

3.5.1 - Nursing Optimization Algorithm

The nursing optimization algorithm was developed by Murphy Choy and Michelle Cheong in 2012[31]. Choy and Cheong used a mixed integer programming model that monitors both the hospital's requirements and the nurse's preference. The algorithm optimizes both of these values in an effort to maximize the total utility and minimize the total cost of all nurses. Further, the algorithm defines a set of "key constraints" that entirely prevent a nurse from working during specific hours of the day. For example, in a *shift constraint*, a nurse may not work more than a single shift per day. The researchers say that the algorithm has been deployed to a general care ward that effectively automates the scheduling system.

[30] https://arxiv.org/abs/1210.3652
[31] https://arxiv.org/ftp/arxiv/papers/1210/1210.3652.pdf

In terms of **implementation**, the nursing optimization algorithm would be quite simple. The team would outfit an already existing algorithm for their own purposes. However, the team would be required to translate the algorithm into code. Thus, the nursing optimization algorithm will receive a score of **4/5** in terms of implementation.

Next, **speed**. The nursing optimization algorithm appears to have too many attributes with respect to what we need for our project. For example, the number of shifts worked in consecutive days is not useful to the project, assuming the labs already have predefined dates. Additionally, the unassignment of all TAs within the algorithm would be extremely slow for large amounts of TA's. In terms of throughput, the algorithm could be trimmed down and become an efficient solution to the problem. The nursing optimization algorithm will receive a score of **3/5** with respect to speed.

3.5.2 - Internal Optimization Algorithm

The team has considered the possibility of developing a custom algorithm to satisfy the requirements. This way, the team would not need to consider the irrelevant aspects of the nursing optimization algorithm and could focus on the immediate needs of the client.

Our algorithm would essentially be a tree building algorithm or a clustering algorithm, the basic idea of which is to take in a list of variables and "weights" for those variables and run them through a heuristic such as the following:

Weight that could be assigned by Lab Organizer	
C1 = -999	A- Scheduling conflict(0/1)
C2 = 5	B - Experience (0,1,2,3)
C3 = 2	C - prev teaching (0,1)
C4 = 10	D - Gta eval score (0,1,2)

Total score= C1A + C2B + C3C+ C4D

The algorithm attempts to maximize its score as much as possible. The client would be able to change the weights associated with each of these example criteria or add their own. This could be accomplished by assigning each new criteria to a two-dimensional array with the name of the criteria and the cost associated with it being considered. One such approach could be a greedy algorithm that iterates through all of the TAs and finds the top three classes for them based on their total cost. However, the team also must consider that some classes are more difficult to staff than others due to difficult time ranges. As such, the developers must give priority to the classes that are hardest to staff. For example, a TA might have a higher total score for one class than another in a naive approach, but the other class only could be staffed by one TA, and thus that class should be picked instead. This will be accomplished by looking at the time availability of the TAs as the algorithm finds classes with minimum listed time availability. When TAs are assigned, we give those classes higher scores within the algorithm.

With respect to **implementation**, the developers would start from scratch. That means the team would be required to develop a setlist of constraints based on the specifications of the client. With that, the algorithm would be more appropriate for the specific use-case of the client. Given these algorithm suggestions come directly from our client, they will not require much change and should be implemented with ease. Additionally, this will mean less refactoring and modification to implement the solution into code. For these reasons, the team's own implementation will be receiving a score of **5/5**.

Speed for our algorithm will likely be very fast as it is essentially accomplished within a single loop and thus should take negligible amounts of time. The solution is essentially a clustering algorithm and thus has a low maximal time complexity. Given that as opposed to the nursing problem the team does not have a need to unassign all of the scheduled TAs, and could likely be accomplished by a single loop. The Internal Optimization algorithm will receive a score of **5/5** in terms of speed.

3.4.4 - Chosen Approach

Now that we have discussed each candidate, we can examine the table:

Table 5. Penalty System

	Nursing Optimization	Internal Optimization
IMPLEMENTATION	**4**	**5**
SPEED	**3**	**5**
TOTAL SCORE	**7**	**10**

The team has determined that writing their own internal algorithm will provide an advantage over utilizing the nursing algorithm. Constructing a personalized algorithm will produce a better final product for the desired goal in terms of both the team's ability to implement it as well as the speed of the solution. With that in mind, the developers have decided to **write our own algorithm** based on the suggestions of our client Dr. Fofanov.

Finally, to prove the feasibility, the team will be prototyping the algorithm with respect to their chosen database, Redis. Utilizing that information, they will be providing a technological demonstration at a later date, on top of the Django prototype.

4 - TECHNOLOGY INTEGRATION

After the team's numerous decisions on the best technology to implement a sound solution, the document will now show the big picture as a diagram. Since everything integrates nicely into one another, a stable, fast, and reliable interface will be underway very soon. The team will be integrating Django, Redis databases, AWS, and the internal optimization algorithm in a way that is shown in Figure 1:

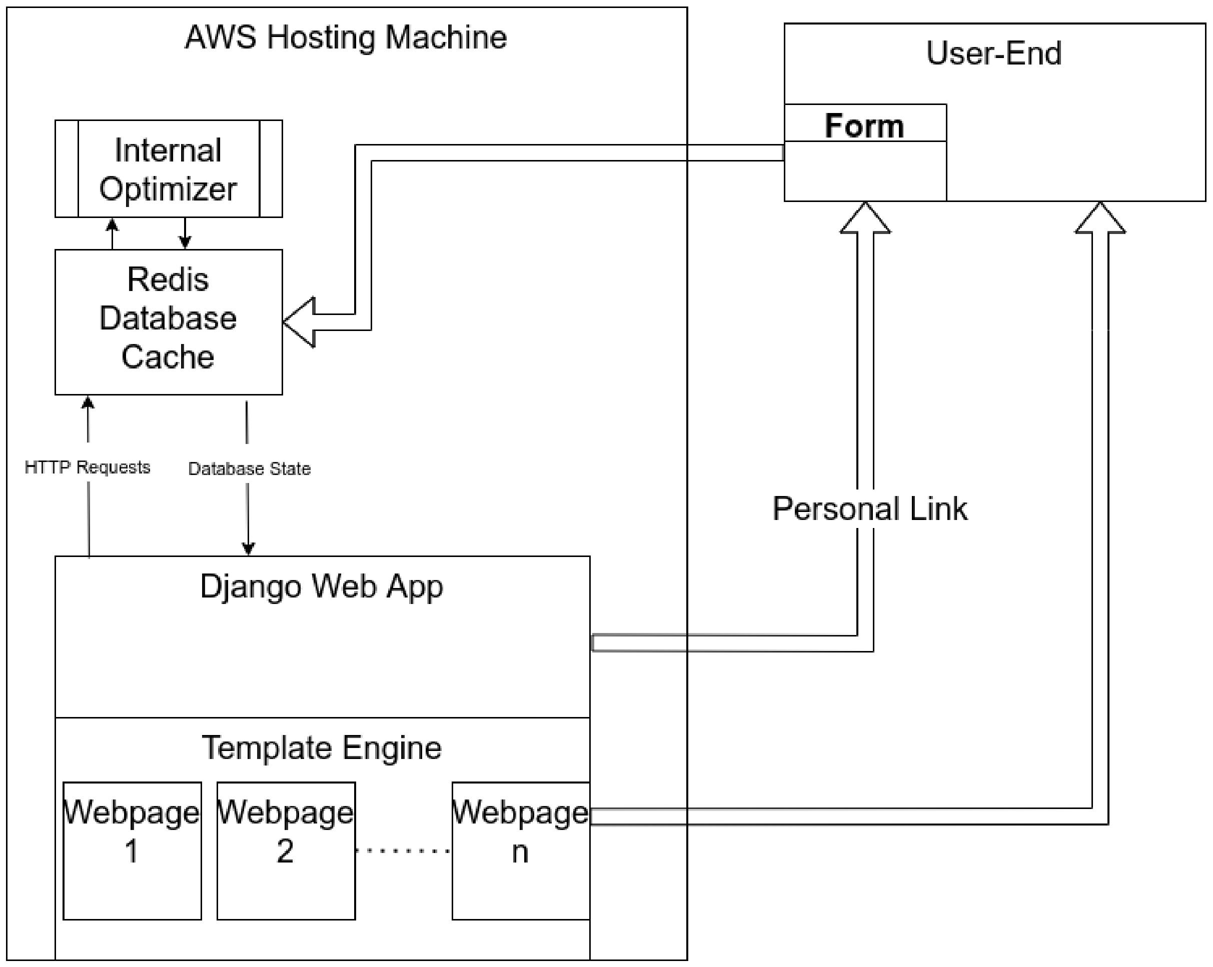

Figure 1. Tech Integration

The hosting machine will be provided by AWS. The Magisters will have to choose a service that will suffice the RAM requirements for the database. The database will be stored in memory rather than disk, which means the changes to it will occur there. The database will be interfaced by Django and will render the different pages necessary for the user to complete their tasks. Some of which will send requests to the server to edit parts of the database. Requests from the user/superuser will be processed and will trigger changes within the database. Meanwhile, the state of the database will always be rendered by Django so that all users can examine which changes were made. To initialize the Redis database, the team will send out personal links to the various users so they can fill out a form likely provided by Django. The web app will then take the filled information and directly inject it to the Redis database. Any changes made to the database will be handled by the internal optimizer so that the data can be readily used to make scheduling assignments. Then it would display using the current schedule generated by the optimization algorithm as well as the option to update existing TAs or to send a form to add new TAs.

With this model in mind, they have thought out the direction they want to head towards and how they will want to implement the solution. The specifics of the technology have been greatly thought through, and this diagram is a culmination of their work. Given all of the technologies the document has discussed thus far, the team intends to integrate them all into a single application that solves the problem of sorting TAs by time availability and schedule; which leads the document to its conclusion.

5 - CONCLUSION

Scheduling large groups of people can be a challenging task, mainly because of its time consumption. Dr. Fofanov currently uses an Excel spreadsheet alone to assign TAs. This process can take days and is a regularly occurring issue not only for our client, but for all group organizers scheduling GREAT masses of people. Team Magisters seeks to remedy this issue and turn a multiple day problem into a minute-long one.

The Magister's solution involves hosting a TA organizing web application on an Amazon Web Service hosting machine. This system would use a Redis database in pair with an internal optimization algorithm that automatically sorts TAs into an optimal position based on their time availability and relevant skills. This information is received from a form sent by the Lab Organizer. The TA will then complete their account creation via email verification. The team chose these technologies based on the various qualities that we have listed earlier in the document. Due to those criteria, we know that the best technologies have been chosen to suit the needs of the product. The team is very confident in their ability to integrate and implement our chosen technologies together into one complete application. After completing the first milestone, the developers will progress to the prototyping and testing phase of development. Once that is finished, they will then move on to a final implementation and release the completed web application.