SOFTWARE DESIGN

Feb. 15, 2022

SPONSOR: DR. VIACHESLAV FOFANOV
MENTOR: VOVA SARUTA

JOSEPH DOMABYL V ANDREW LIDDELL JUNJIAN YIN DANIEL DRAKE

Magisters

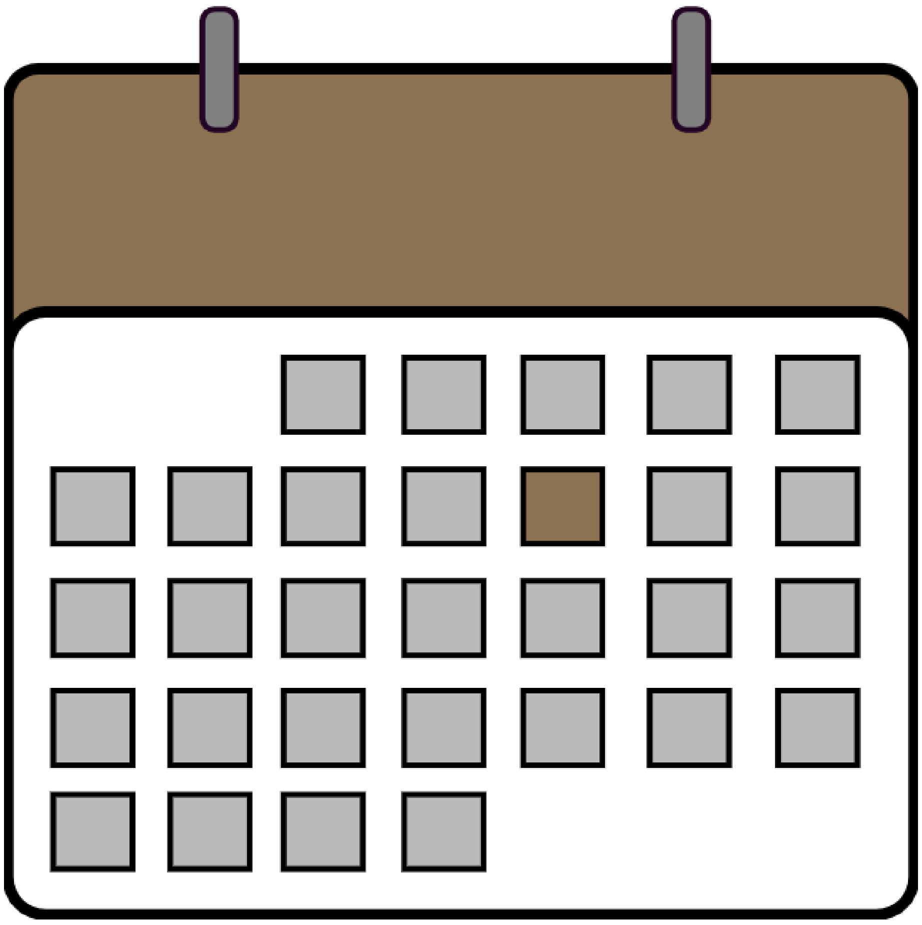


TABLE OF CONTENTS

1 - INTRODUCTION

Working with and coordinating large groups of people can be a time-consuming task. Consider a scenario where you've been set to schedule potentially hundreds of different people, all with different strengths, weaknesses, and availability. You've been working on this schedule for a couple of weeks only to find that one of your team members can't make the 2 pm timeslot on Friday they said they could. You now have to find them a new spot in the schedule they can attend and you will have to factor in everyone else's schedule before changing it. For example, moving one person might require that you move three more people down the line. This has the potential to become destructive for your workflow and is what university departments all around the nation deal with regularly. This is a persistent issue that plagues these departments semester after semester, particularly the scheduling of teaching assistants (TA's) and graduate teaching assistants (GTA's) for lab positions. For the rest of this document, we will be referring to both TA's and GTA's simply as TA's unless specified otherwise.

The current workflow used by our client, Dr. Viacheslav Fofanov, is to use an Excel spreadsheet and manually assign these TA's 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 GTA's to teach labs and TA's 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 volume, or how long it takes to cross-reference time slots 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.

To assist our client in manual optimization, the team has opted to create a fully featured web application, which the team is calling QuickSched. To that end, different sets of requirements for the implementation were drafted and designed. Starting hierarchically from the highest level of requirements, domain requirements, the following description will trickle down into both functional and nonfunctional requirements on the lower end.

The first domain level requirement that we found was the need for security; user authentication and permissions would be necessary to fulfill our client's vision of utilizing a safe and secure application. As a consequence, there will be two types of users within the application; Lab Organizers and TAs. These roles will be handled differently with Lab Organizers acting as administrators and TA's only having viewing permissions. The next major domain requirement is that of an intuitive GUI for both user types to use. This GUI will function as a visual tool for Lab Organizers and allow them to make changes to TA placement using button clicks. Finally, the last domain level requirement will be the organization and scheduling of TA's for lab placement. This requirement is critical in achieving the primary objective of providing automated assistance in the TA scheduling process.

Based upon the domain level requirements, the team has determined sets of both functional and nonfunctional requirements. To address the requirement of security we determined that we will need different permissions for both Lab Organizers and TA's, as previously mentioned. Additionally, we found that it will be necessary to include information security for the TA's to prevent the issue of interpersonal conflict due to desired scheduling placement. For example, TA's will not be able to view the names of other TA's when viewing the overall schedule. To assist in the process of role assignment, Lab Organizers will be able to send account creation links via email to desired TA's. This will allow for Lab Organizers to control account creation that will allow TA's access to the application. To address the domain requirement of a GUI for both user types, we opted to create a separate interface based on user permissions. To elaborate, Lab Organizers will have a distinct interface from TA's and vice versa. The purpose of this GUI will then be to display the data on TA's to the user. Finally, to address the domain requirement of scheduling TA's for lab placement, we will be implementing a data management system as well as a TA scheduling optimization algorithm. The data management system will provide data that will be displayed by the previously mentioned GUI, as well as store TA and semester information in a persistent state. This is done so that data can both be displayed to the user through GUI as well as allow it to be run through the aforementioned optimization algorithm. The optimization algorithm will "rank" TA's based on qualifications and time availability. Additionally, the Lab Organizers will be able to add additional custom criteria by which to sort TA's. This is to make the algorithm customizable for specific sortation requirements designated by the Lab Organizer. The algorithm will also have the ability to distinguish between GTA's and TA's, for which their qualifications may differ. Finally, the algorithm will be designed to run in an efficient fashion to not deviate from the purpose of the overall application.

Additionally, there are sets of nonfunctional, or performance, requirements. These include security, having a professional look-and-feel, and modularity. Security will be necessary to protect against outside application threats such as clickjacking, cross site request forgery, and injection attacks. Creating a professional look-and-feel will be done in order to have a clean GUI with self intuitive elements to assist in user input. The last of the performance requirements is modularity, this will be done to ease future modification and maintenance of the application.

2 - IMPLEMENTATION OVERVIEW

As stated above, the project will be built as a web application holding several modules. We will host the application via an AWS server, which will hold a database to store the TA information, a web framework environment to ensure the ease of operation and manipulation of the database, and an optimization algorithm to automatically sort the TA's when any changes are made by the front end. Lightsail will be our chosen package for AWS, as our project will not need a largely customized machine to host the application.

The Python-based web framework Django will be running our main GUI for the functionalities, and its administration page will be the view for the lab organizer to edit the information of the TA's. The TA's will have a page of their own. On the backend of the application, there will be both an SQLite database and a Redis cache working in tandem. The information sent in by forms hosted by Django will be stored in the SQLite database then cached into Redis to increase response time. The optimization algorithm will request information from the SQLite database to organize it and set up different possible schedules for the lab organizer. The database info will then be rendered to the Lab Organizers view such that they may confirm the schedule. Django will also be able to send emails to the recipients wanting to become TA's with a personal link so that they may submit a form of their information. This will be provided by a Django emailing core utility.

The TA users will be able to authenticate themselves the minute they enter the domain in their browser, which will be a login screen provided by Django. Once they have authenticated, Django will recognize them based on their account type and take them to the overview of their job schedule. They will be able to see when they will work, along with how many people they are working with. If a TA needs to change their personal information, they will need to contact the Lab Organizer directly.

The Lab Organizer will be presented with the same login authentication once they visit the domain. After authentication, they will be taken to their personal dashboard page. There, they will be able to see the schedule themselves and see who is assigned. From the admin page, they will be able to change the information about each of the TA's and save the changes to the SQLite database. This will be handled by the URL routing and the template engine within the Django core utilities. As a note, each type of user will not be able to access each other's respective pages.

3 - ARCHITECTURAL OVERVIEW

The following section will explain how each module works with one another, and the key features of each module from a high-level perspective. Figure 1 below provides an overview of the entire architecture.

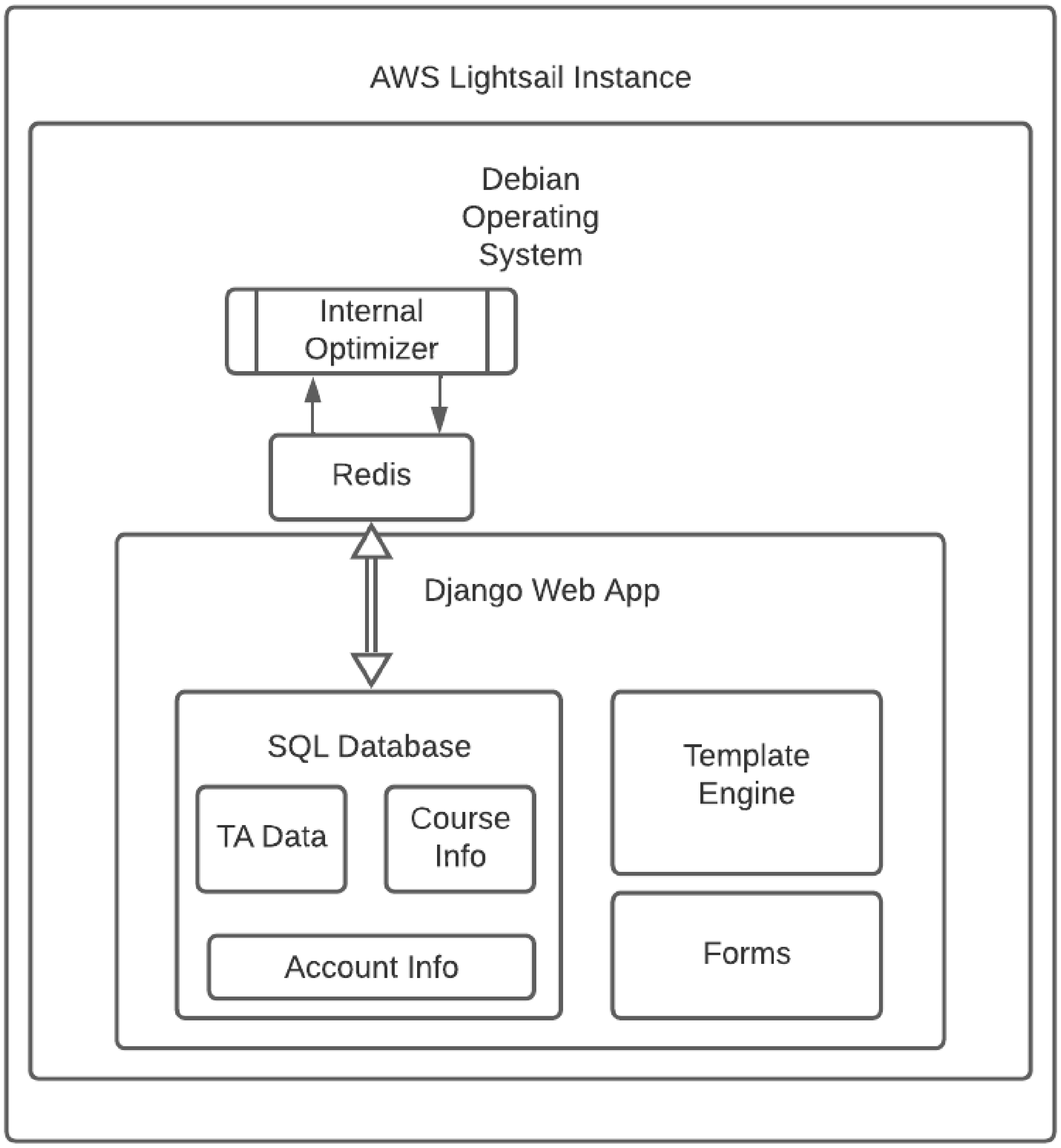


Figure 1. High-Level Architecture Diagram

The user-end primarily interacts with Django. Forms will be the primary means of interaction between the user and the backend. The forms that are completed will be parsed and passed into the SQLite database for storage. The Django framework is where the main functionality lies. The primary duty is to handle the user input, generate visual output, then display it to the user. Django has an internal template engine that generates HTML dynamically so that pages are efficiently generated based on demand and need. Once requests are received from the user side, the Python code within Django will pass the requests to wherever they are needed, be it requesting data from the database, sending data to the database, or displaying data to the user. Like a hub between user and data, this structure enhances the security of the server's data and also improves user accessibility. The AWS Lightsail Instance will be responsible for processing requests, getting data from the SQLite database, and generating output based on the optimization algorithm. In addition, the AWS server provides a cloud service to the application, users can access the app through the internet. The server receives the form from the user or requests from Django by HTTP. On top of the SQLite database will be the Redis cache. SQLite will be the primary system that persistently stores data. Now that the overall architecture has been discussed, we can examine each module at a deeper level.

4 - MODULE AND INTERFACE DESCRIPTIONS

4.1 - Django

Django will be the driver for the UI of QuickSched. First, it will provide the login page and authentication of the users that wish to login. It will render the entries of the database to be viewable for both the Lab Organizer and the TA's. It will send out emails and links for TA's to register their information, where the optimization algorithm will take over once the entries are added. It will also provide an admin page, which will serve as a tool for the Lab Organizer. All of the views stated above will be handled by the template engine. The environment itself will be hosted by an AWS server instance and will encase both the SQLite and Redis cache in the backend. Here, we will explain the functionalities of the different parts of Django. Figure 2 below showcases the overall architecture of the Django environment.

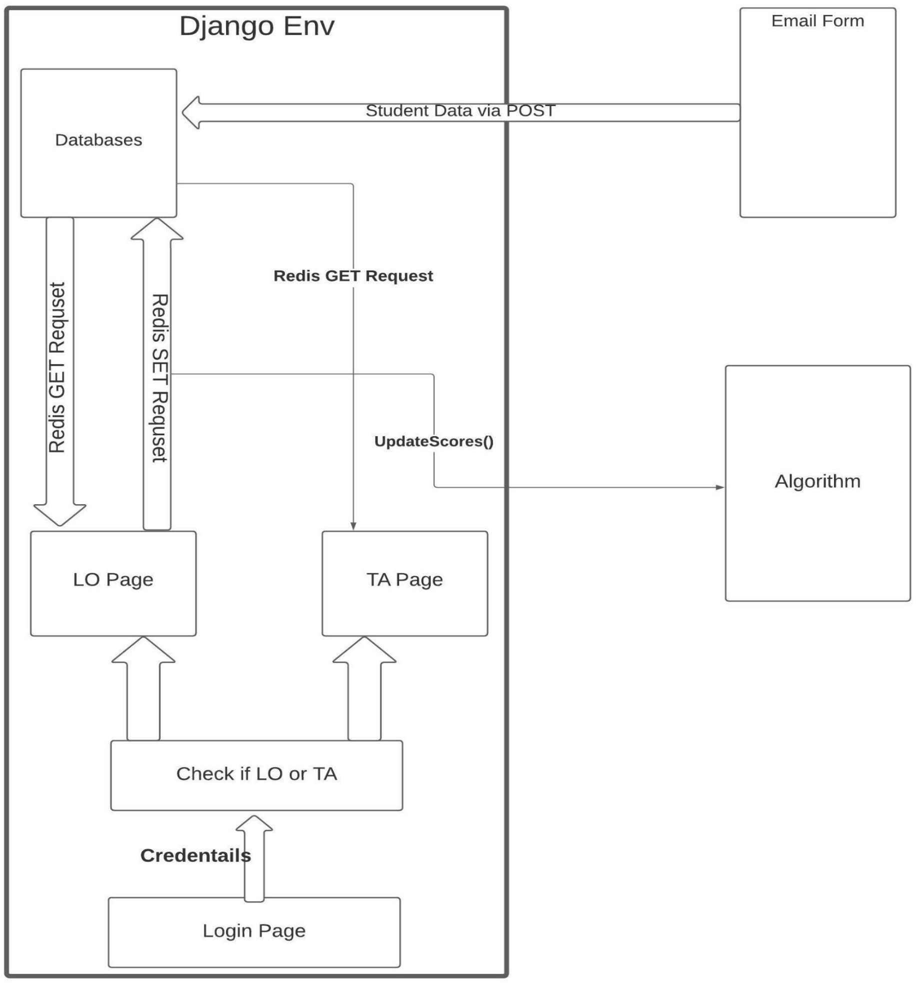


Figure 2. Django Environment

4.1.2 - Authentication

Django has its own utilities for authentication out of the box. When verified users enter the site domain, they will be first prompted by the authentication screen. Depending on their role as a user, they will be redirected to a different page. This will be handled by the URL routing system within Django. It will either take them to the Lab Organizer view or the TA view depending on their user permissions. The means to gather this information will be described down below.

4.1.2 - Email and Registration

Along with authentication tools, Django has its own modules for sending emails and making forms to edit our database. As said earlier, the application will be using an SQLite database and a Redis cache for the records of all users. The Lab Organizer will collect all of the emails and each recipient will receive a unique link to register their account. The forms to do this will be handled by Django's template engine component, which will be referenced down below. The potential TA's will enter their information in the fields, and insert the information within the database via a POST request functionality, also provided by Django. From there the TA's information will be stored in the database, and will undergo the process of scheduling the TA's via the algorithm. The Lab Organizer will then be able to view the information and edit it as they see fit. The recipients will be stored in a CSV file through their emails, acting as the primary key. When the Lab organizer uploads a new CSV for a new semester, the application will detect whether there are returning TA's or not. From there, it will send them a reminder to update their scheduling information, so they aren't prompted to remake their account.

4.1.3 - Admin Page

The admin page built into Django provides all of the functionality necessary to edit changes in the database via a GUI. The GUI will include a way to edit the fields of the TA's, optimization criteria, and semester information. Any changes to these fields will automatically be stored in the SQLite database, and subsequently into the Redis cache when necessary. The admin page will be accessible only by Lab Organizers.

4.1.4 - Lab Organizer View

The Lab Organizer view will include the following elements. The right side of the page will include the available switches between the students and their score will be returned at the bottom of the switch. If there is an empty spot for the class, certain students will be able to switch with the empty spot, depending on their score. All of the information on the webpage will be accessed through a "get" request to the database system. Figure 3 below represents the overall Lab Organizer view.

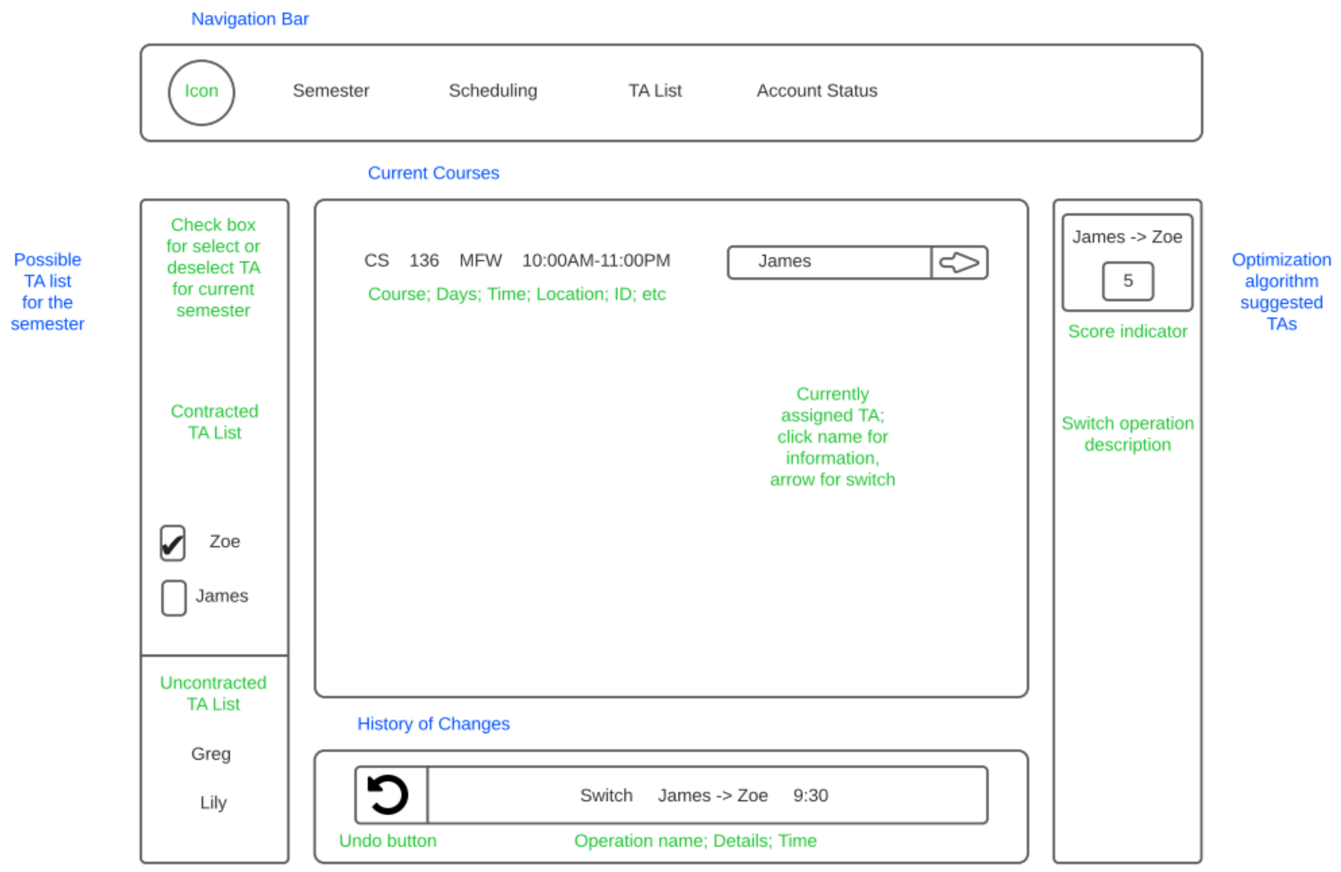

Navigation Bar
Icon
Semester
Scheduling
TA List
Account Status
Current Courses
Possible TA list for the semester
Check box for select or deselect TA for current semester
Contracted TA List
Zoe
James
Uncontracted TA List
Greg
Lily
CS 136 MFW 10:00AM-11:00PM
Course; Days; Time; Location; ID; etc
James
Currently assigned TA; click name for information, arrow for switch
James -> Zoe
5
Score indicator
Switch operation description
Optimization algorithm suggested TAs
History of Changes
Switch James -> Zoe 9:30
Undo button
Operation name; Details; Time

Figure 3. Lab Organizer View

Before the availability of switches, there will be an option to initialize the schedule. From there, all management of the schedule will be handled by switches. When a change is confirmed by clicking the switch, it will send out a “set” request to the database to confirm it. This will also call the algorithm to update the scores for the two that were switched, but they can still be switched to their former positions after it was confirmed. There will also be a functionality where the lab organizer will be able to force a switch between 2 TA’s regardless of their scoring. The bottom of the page will include a history component that will allow the lab organizer to undo any changes that they previously made. The left side of the bar will filter the students to see their schedules and set times in a list view in the middle. This will be handled by the HTML itself with some Django utilities. It should be noted that in most scenarios, TA’s will not be able to edit their own information without express consent by the Lab Organizer. However, there will exist a toggleable switch accessible by only the Lab Organizer that when activated, will reroute TA’s to different forms that will allow them to change their information. For example, a couple of weeks before a semester, a Lab Organizer may decide to open the “Change

Availability" forms for all TA's and set a timer of 7 days. After those 7 days, the system will revert back to what is essentially a "read-only" view for the TA. Figure 2 below represents the overall Lab Organizer view.

4.1.5 - TA View

The TA view will share the same general layout as the Lab Organizer view, but not the functionality. When a TA logs in to the system, they will be presented their own view of the schedule, but they will not be able to edit it. Further, they will also be able to view their account status, i.e. if information is missing from their overall profile. In order for TA's to provide their own schedule, they will be required to fill out a form that contains start and end times on a day-by-day basis. Figure 3 below displays the TA view.

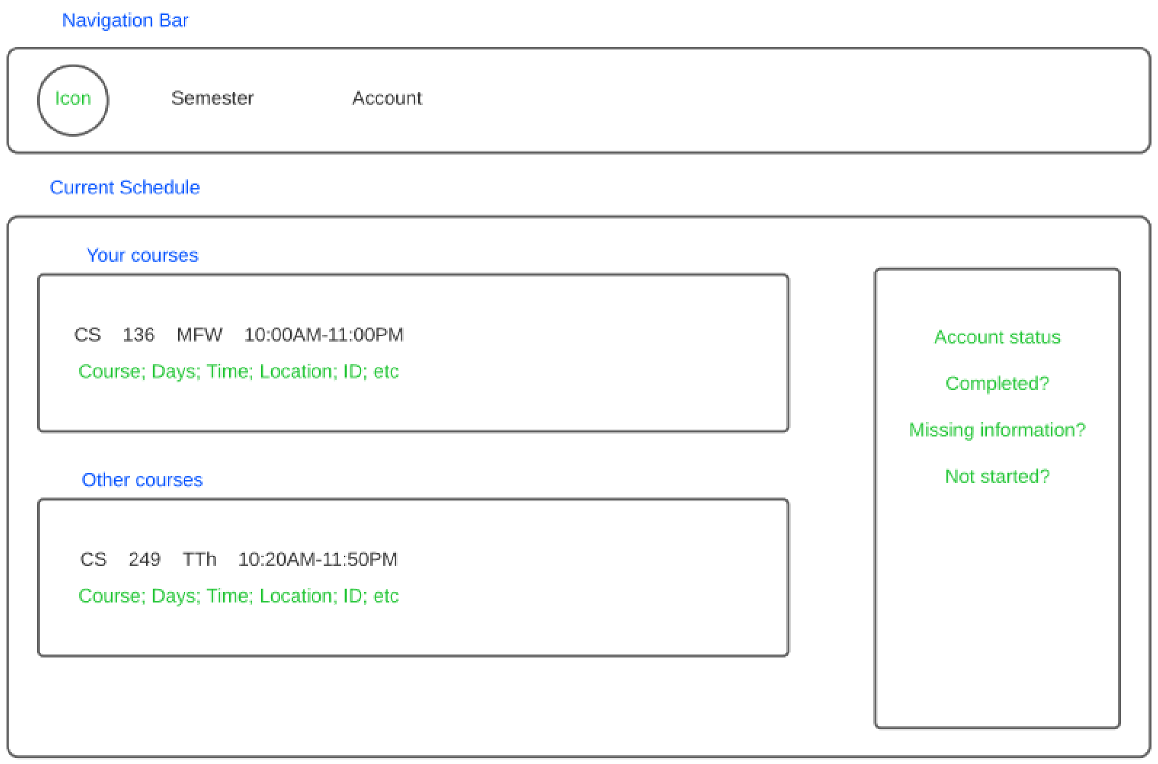


Figure 4. TA View

4.1.4 - Template Engine and URL Routing

The template engine will serve as the base for both the Lab Organizer view and the TA view. Once the user has authenticated, Django will check if their credentials are that of a Lab Organizer or a TA. From there, the web app will fork to either view. Further requests will be handled independently within the context of the action that initiated them.

4.2 - Database

The product will be delivered with a primary database system as well as a database caching system. SQLite will act as the primary database and Redis will act as the cache. Both Redis and SQLite will provide data persistence, creating another layer of data stability.

4.2.1 - SQLite

The primary database, SQLite, will be responsible for maintaining larger records that will subsequently be accessible by Redis. For example, SQLite will store semester information, account information, TA information, and optimization information. Within the larger architecture, SQLite will reside under Redis in the stack.

4.2.2 - Redis

The database cache, Redis, will enhance the overall speed of the entire application by storing requested information in memory. Redis will cache data stored in SQLite, providing faster response times for frequently accessed information. In the larger architecture, Redis will reside right on top of SQLite in the stack.

Figure 5 below illustrates the relational nature of the internal components of the overarching database system.

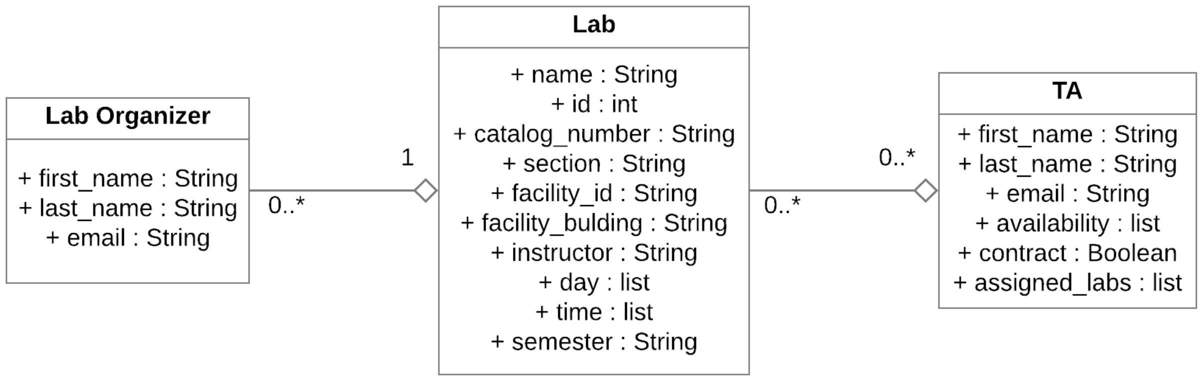


Figure 5. Database Architecture

There are three primary data types. The lab organizer (LO), the lab itself, and the TA. The LO object contains user information and can be assigned many labs. The lab object contains all relevant lab information and can contain as many TA's as necessary. The TA object contains user information and can derive which labs they are associated with by relation. Lab organizers and TA's can exist independently of each other, however a lab must be associated with an LO. For example, if a TA does not have a contract and does not get assigned to a lab, their information will still be available for future semesters.

4.3 - Optimization Algorithm

The optimization algorithm will have the primary function of assisting in the manual optimization of the TA scheduling process. First, the algorithm will receive TA data from the Redis cache and SQLite models. Once the algorithm has received the data on both the TA's and the labs for the semester, the algorithm will start. It will go through each criteria and the weights that it has been provided. If it hasn't been provided any criteria, it will still use the default criteria of time availability, relevant skills, and if the TA is a GTA or not. With all of this data, it will initialize a schedule by comparing each TA to every lab available, saving the overall score of that TA for every lab. Once these scores have been calculated, the TA's will then be placed based on the best scoring TA for that lab. While TA scores are being determined, they will also be stored within the database for future use. Figure 6 Below demonstrates the Optimization Algorithms process.

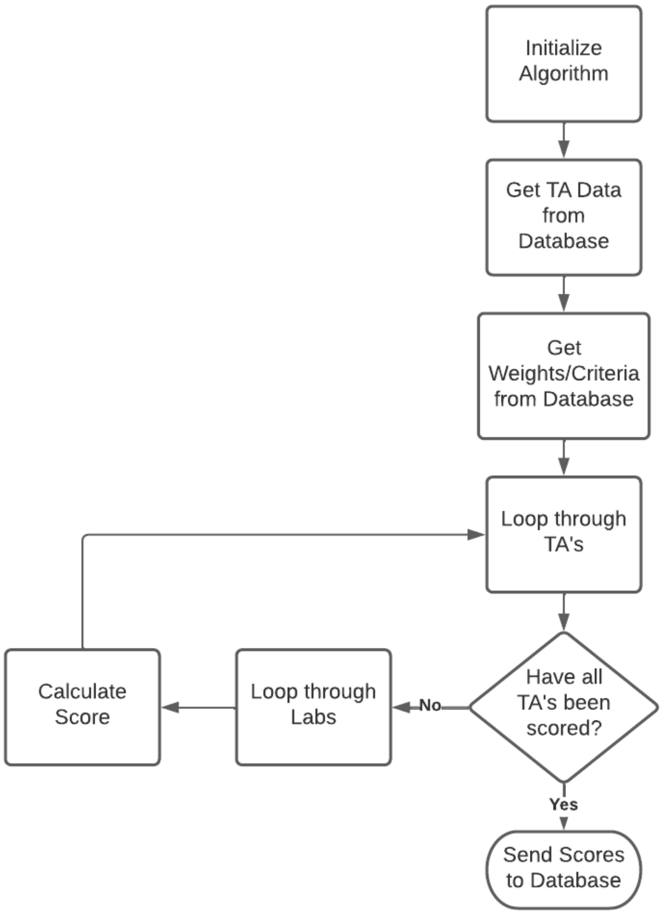


Figure 6: Optimization Algorithm

The public interface of the algorithm includes two parameters it receives as arguments. These include the criteria and their weights which are provided by the Lab Organizer. Both of these parameters are entered by the Lab Organizer within the GUI and is then sent to the database. These are essential to the value comparison of the algorithm since they provide the mathematical basis on which the TA's are sorted. If no values are provided, the algorithm will simply use the previously mentioned default criteria of time availability, relevant skills, and if the TA is a GTA or not. Once the algorithm has run and the scores for the TAs have been calculated, those scores can be requested to be displayed using Django's UI tools.

4.4 - AWS Lightsail

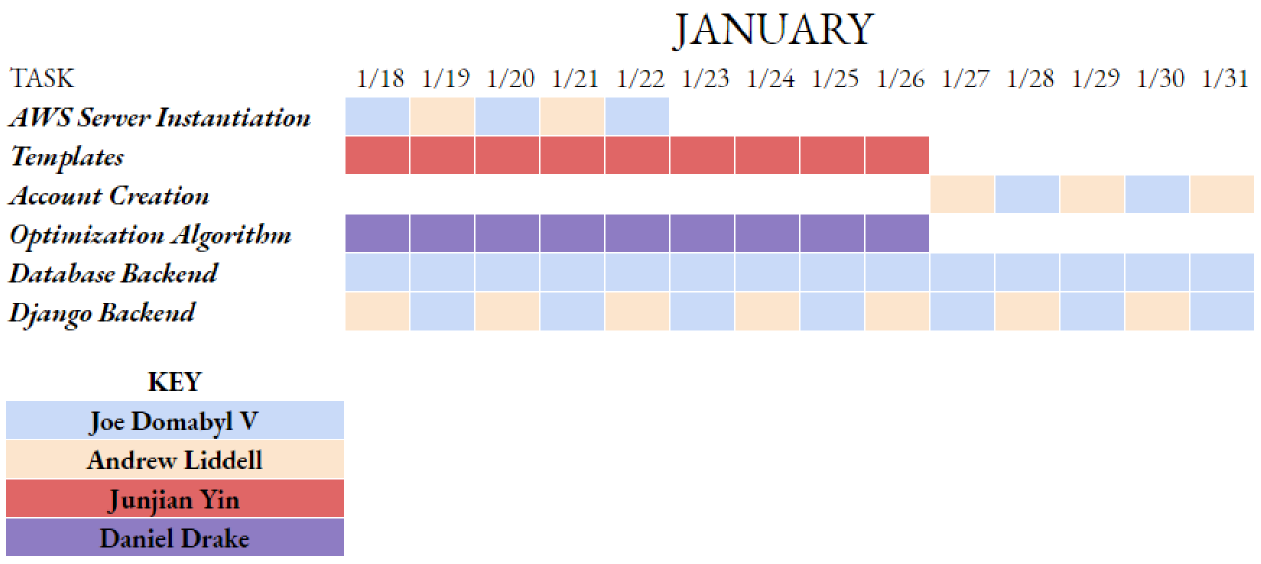
AWS Lightsail is a virtual private server (VPS) service provided by Amazon. The Lightsail instance will store the entire application, to be accessible at any time by any authenticated user. Django, SQLite, and Redis will all exist within the server. The entire project will be packaged in a GitHub repo such that any potential Lab Organizers can download the code and establish their own instance. The server will be hosted using a Debian operating system. In order to serve the Django project, the instance will utilize both Nginx and Gunicorn.

5 - IMPLEMENTATION PLAN

The team has opted to develop work cycles on a monthly basis to establish an MVP. Throughout the rest of this section, the months January and February will be discussed. Each month will have two different Gantt charts. One version of each month will be displayed here which provides a broad overview of the coming work. A more detailed representation of the work being done will be discussed in the narrative below each Gantt chart.

Within each Gantt chart, the colors of each row represent who on the team is doing what. Alternating colors represent multiple people working on the same, or very similar, tasks.

5.1 - January



JANUARY

TASK	1/18	1/19	1/20	1/21	1/22	1/23	1/24	1/25	1/26	1/27	1/28	1/29	1/30	1/31
AWS Server Instantiation	🟦	🟧	🟦	🟧	🟦									
Templates	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥					
Account Creation										🟧	🟦	🟧	🟦	🟧
Optimization Algorithm	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪					
Database Backend	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Django Backend	🟧	🟦	🟧	🟦	🟧	🟦	🟧	🟦	🟧	🟦	🟧	🟦	🟧	🟦

KEY	
🟦	Joe Domabyl V
🟧	Andrew Liddell
🟥	Junjian Yin
🟪	Daniel Drake

Figure 8. January Gantt Chart

At the time of writing this document, the team has considered the first official day of development for this semester to be January 18th, 2022. In the January Gantt chart, each column represents one day. To begin, the team will have multiple simultaneous tasks being worked on. First, Andrew and Joe will begin establishing an AWS development environment for the team to begin implementation. This task includes creating a team AWS account, creating a Debian Lightsail instance, and installing any necessary components on the server. At the same time, Junjian will be developing multiple templates to be used within Django, also discussed above. These templates include the login page, the general TA view, and the general LO view. Daniel will also be working on developing a solid foundation for the optimization algorithm to be used when the product launches, in the form of pseudo code. Finally, towards the end of the month, both Daniel and Joe will be creating the functionality for account creation within the application. This will include authentication, security, and sending account creation emails to new users.

Throughout the entire month, Andrew and Joe will be responsible for Django backend work. To elaborate, this includes routing views, ensuring template functionality, creating correct directory structure, and other necessary "under-the-hood" work within Django. Joe will also be responsible for maintaining the database structure discussed above. This will include creating proper data models as well as ensuring they can be accessed by other files.

The team's current standpoint is represented by the red "nowline" and is expected to make great progress in the coming weeks. With the foundation developed above, the team will then move into the development plan for the month of February.

5.2 - February

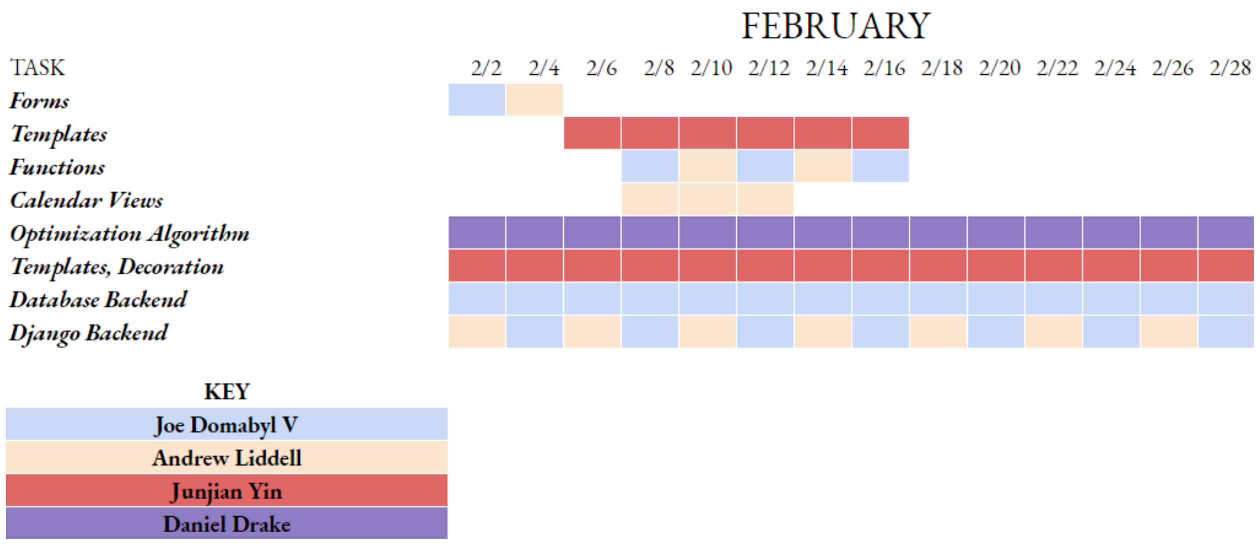


Figure 9. February Gantt Chart

The month of February is expected to establish the final version of the MVP. At the beginning of the month, Andrew and Joe will establish any necessary forms within the Django architecture. These forms include new users and new semesters. As with the month of January, Junjian will be developing further templates for the rest of the application. These templates include LO semester views, TA lists, TA account views, and TA semester views. Beginning around the same time, Andrew and Joe will establish primary functions. These functions include the ability for the LO to view a history of previous changes made to the schedule, a button to select/deselect TA's in the current semester, and a function to display the score given to each TA that responds in real time. During the function development, Andrew will be developing a calendar view for use by both LO's and TA's, to display information relevant to their purpose. For example, the LO will be able to view each TA's schedule, the TA will only be able to view their own. Throughout the entire month of February, Daniel will be finalizing the optimization algorithm to be implemented in the completed MVP. Also throughout the month, Junjian will be delivered different elements of the application (such as forms) to decorate in correspondence with the rest of the application. Finally, as with the month of January, Andrew and Joe will be maintaining the backend of Django and Joe will be maintaining the database backend.

With this implementation plan for the months of January and February, the team is confident that an exceptional MVP will be delivered on time. As a final note, the later half of the month of February will remain open in an effort for the team to remain agile. The open time will then be allocated to any necessary changes/features.

6 - CONCLUSION

In conclusion, this project is designed to aid Doctor Fofanov in the scheduling and management of TA's. The problem lies in the time consuming nature of manually scheduling large teams. Our client currently manages TA's using an Excel spreadsheet, which takes roughly 30-40 hours per semester; however, the total time required is dependent on a number of different variables and has the potential to increase indefinitely. Our proposed solution will be creating a web application that allows users to create, edit, and assist in the manual automation of sorting TA schedules for a given semester. Users will be presented with a visual representation of the data they have created, be able to access the system at any time, and be able to utilize different configuration features to better suit their needs. By discussing the software design outlined above, the team is confident that the application, QuickSched, will be able to save countless hours for countless institutions *worldwide*. Further, the system may also be adapted to conform to organizations outside of the realm of scheduling TA's, as is the nature of a modular Django product. With all of this information, the team is incredibly excited to produce a fantastic product ready to be used for years to come.