SOFTWARE TESTING version 1.0

Apr. 3, 2022

SPONSOR: DR. VIACHESLAV FOFANOV
MENTOR: VOVA SARUTA

JOSEPH DOMABYL V   ANDREW LIDDELL   JUNJIAN YIN   DANIEL DRAKE

Magisters

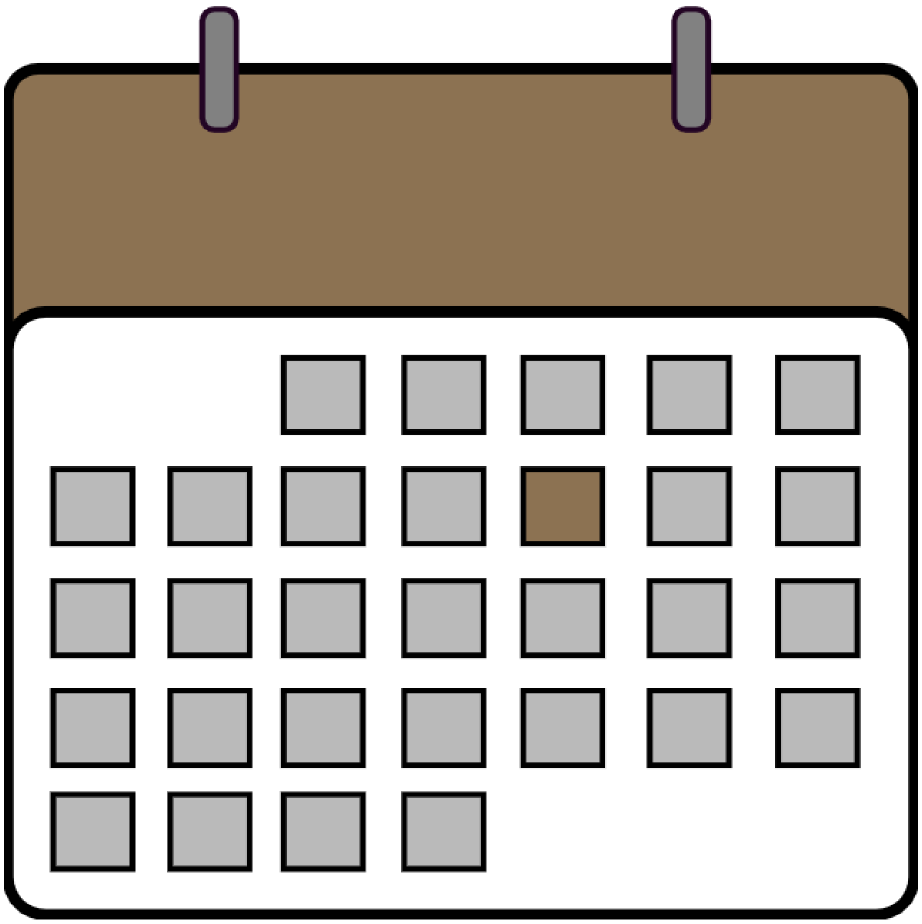

TABLE OF CONTENTS

1 - INTRODUCTION

QuickSched is designed to assist in the scheduling and management of teaching assistants (TA's). The project is sponsored by Doctor Viacheslav Fofanov, the associate director of graduate programs at the School of Informatics, Computing, and Cyber Systems (SICCS) at Northern Arizona University (NAU). Doctor Fofanov is responsible for scheduling TA's to different labs, depending on a multitude of different variables such as TA availability, experience, and employment status. The process has proven difficult and time-consuming in the past, rendering it a necessity for QuickSched; a robust, instance-based web application. Team Magisters has decided to move forward with Django, a Python-based web framework, combined with SQLite3, a lightweight database management system.

In order to provide Doctor Fofanov with the solution envisioned, QuickSched will require thorough testing and quality assurance protocols. The team will be paying special attention to three primary factors; functionality, speed, and the user interface. In order to examine these three aspects, the rest of this document will break down the larger components into smaller, more manageable pieces. Those pieces will be presented in the form of three different testing scenarios; unit testing, integration testing, and usability testing.

The overall testing plan will begin with unit testing. First, the team will examine the unit aspect of the application. In this section, different sections of code will be presented and testing mechanisms will be specifically discussed in terms of the operation of the code. For example, which libraries will be used to test this specific block of code. Next, integration testing will be examined. In the integration section, the team will explain how different components will work in tandem with one another. For example, testing applications will be presented to display how Django works directly with SQLite3, down to the return types and parameters of conjoined methods. Finally, usability testing will be provided to display how the application will appear to the user under higher levels of stress. For example, the team will display how the application will respond when there are too many entries in a specific section of the application and where the overflow will go. Per discussions with Doctor Fofanov, it has been determined that the usability of the application is one of the most important features. For this reason, the team will be paying special attention to the user interface provided with QuickSched. First, we begin with unit testing.

2 - UNIT TESTING

Unit testing is the process of testing singular blocks of code in your system or application. We will heavily focus our testing within the backend views of our application and the utilities coded for both the optimization functions and the lab organizer utilities. We will use the built-in testing functionality of Django to assist us with automatically generating units to make sure our code is stable and covers all different cases. We will also focus our unit testing on the optimization functions, the authentication module, lab input information, and TA information editing.

2.1 - Optimization Functions

Throughout the optimization process, scores will need to be properly formatted. Properly formatted scores will serve as the key aspect of unit testing the different optimization functions.

Regarding the initial score calculation function, the team will be manually presenting the application with a set of scores that TA's *should* receive from the automatic generation. In this case, the previously mentioned Django testing module will be utilized. For example, a set of TA's will be tested against a set of labs, and the assigned scores will be checked against the given score list. In order to pass the test, the scores must adhere to the following format:

- **Scores must be assigned to the proper template schedule.**
    - Each score is assigned to the corresponding template scheduling, paying attention to the year and current semester. Thus, a "score" object must have a key to the template schedule. Scores for specific TA's, for specific labs, for specific templates, must also be unique. No two scores may be duplicated.
- **Scores are returned as integers.**
    - There must be no scenario where a given score is not of data type "integer", thus no TA can receive fractions of points.
- **Scores must be within a given range.**
    - The range is defined by the lab organizer depending on the weights they assign to different criteria. For the purposes of testing, the default values will be used, thus scores must be within the range 0-100.

For a newly generated schedule, the optimization of TA's must be initialized. To initialize a schedule, TA's will be placed in the best possible location depending on their scores. To elaborate, TA's with the highest scores must be assigned to those corresponding labs. In order to test this, a series of comparisons will take place:

- **The highest scoring TA will be awarded the assignment to a lab.**
    - The TA with the highest score will be assigned to the lab that is currently being evaluated. If the highest-scoring TA is already assigned to another lab, the next highest scoring TA will be assigned instead, in a cascading fashion.
- **TA's will attempt to only be assigned to one lab per semester.**
    - The highest scoring TA will be assigned to a lab provided they are not assigned to another lab. TA's will only be assigned to multiple labs per semester if the total number of labs is greater than the number of total TA's.

2.2 - Authentication

In the application, accounts are created via email, which serves as the primary identification key for individual accounts. This means that a single email cannot be used to create multiple accounts, which means we will have to establish error handling if that were to occur. Each email will be tested against the rest of the database to ensure that there are no duplicates. When a lab organizer attempts to upload a new list of emails, the emails must adhere to the following format:

- **An email may not exist twice within the database.**
    - In the event that an email exists within the database and also exists within the list of new emails attempting to be uploaded, the lab organizer will be shown that these accounts already exist and will be handled differently.
- **Emails must follow the syntax "example@example.com".**
    - Each email must begin with a string, be separated by an "@" symbol, and be followed by a domain. If a given email does not match this format, the lab organizer will be informed.

In order for a lab organizer to upload a new list of TA's, they will be required to either manually enter TA information or utilize a file containing comma-separated values, a CSV file. For either circumstance, the following format must be used:

- **Emails are properly formatted, as stated above ("example@example.com").**
- **Each row in the CSV file must begin with a proper email, followed by a list of information the lab organizer wishes to provide.**
    - For example, the lab organizer will have the option of providing first/last names, student ID's, and contract status. Each given element must be tested to match format requirements, i.e. a given first name must be a string of characters. Each row and column in the CSV file will be tested to match proper data types. If the data types do not match, the lab organizer will be informed and required to either remove the entry or reformat it.

User credentials will be checked against each request to access a new page. Unauthenticated users cannot have access to the information in any of the lab organizer views. For example, if a TA attempts to access a lab organizer page, they will be redirected to the sign-in page. This will be accomplished by using an in-place test at the beginning of each method header, utilizing the "login_required" decorator. If the user is not logged in or authenticated, they will be redirected accordingly.

2.3 - Lab Information Input

Similar to the lab organizer inputting TA roster information via a CSV file, they will also have the option to input bulk lab information. When a lab organizer uploads a CSV file containing this lab information, it must follow this format:

- **Semester time and year**
    - The system must be informed which time and year the semester is taking place. Without this data, the following lab information will be denied. This data will be checked against the existing data model for a semester, such that the given time (Spring, Summer, Winter, Fall) must be a string of characters, and the year must be an integer (2022, 2023, etc.).
- **Class name, course ID, subject, and catalog ID**
    - The next row of the CSV file must contain a properly formatted class name, course ID, subject, and catalog ID. Each input will be evaluated against the existing data models at the time of upload, again via Django in tandem with SQLite3.
        - A valid class name may contain spaces, hyphens, or numbers, but no other special characters, i.e. "Computer Science 1".

        - A valid course ID must be given as an integer and it must be unique, i.e. “12345”, where this lab is the only lab with the same course ID.
        - A valid subject must be abbreviated and be no more than 4 characters long, i.e. “CS”, “MAT”, or “CYB” (Computer Science, Mathematics, Cybersecurity).
        - A valid catalog ID must be unique to a single semester but does not need to be unique across different semesters. For example, the Spring of 2022 semester may contain a lab with the catalog ID “126” but not multiple instances of a “126” course.

2.4 - TA Manual Information Changes

Throughout the time a TA may exist within the system, they might need to update their personal information. There are two scenarios that which this is allowed: when their account is first created and when the lab organizer manually gives them access. In the event that an account is freshly created, they will be allowed one opportunity to change their information. When the lab organizer allows TA’s to change their information, they will be allowed to freely change what they please during a given time period. The time period is defined as follows:

- **The lab organizer chooses a date and time.**
    - The lab organizer may choose any future date and time combination, allowing TA’s to access and edit their own information until that exact time.
- **TA’s may not edit their information outside of a given period.**
    - To prevent scheduling conflicts, TA’s may not edit their information as they please. They must be given access by the lab organizer or have freshly accessed their account.

3 - INTEGRATION TESTING

Integration testing is the practice of ensuring individual components interact with each other properly. The Magisters will test the integration of all of the parts primarily by making sure that database objects that are passed between views will be handled in all cases. For example, if a function unexpectedly generates a none type, we will need to have all functions that use said object to be able to handle none types and report the error to either the user or the server-side host. The integration testing will need to focus on database integration, utilizing proper server-side hosting, and ensuring correct data output from optimization functions.

3.1 - Database Integration

Django operates on a request-response basis. Thus, throughout a user's experience with QuickSched, they will be creating a series of requests and expecting a proper response. Part of the main functionality of QuickSched resides in the constant flow of database transactions. These requests may include but are not limited to, adding TA's to the database, creating new semester/lab objects, or editing personal information attached to a single account. In order to ensure that database transactions are properly handled, the team will be utilizing Django's preinstalled testing module, discussed above. The testing module allows for the creation and different tests based on user input and allows for assertions of proper results. For example, when adding a new set of TA's to the system, one would expect the TA to be immediately rendered within the actual database. That TA object would then contain the email, first name, last name, and contract status with the university. From there on, that data must be usable by the rest of the system. For instance, a newly created TA might immediately need to be scheduled. Thus, the team has come up with a set of requirements ensuring proper data manipulation/database transactions:

- **Newly created objects are populated with the designated field requirements.**
  Each object model within QuickSched is defined in the backend of the system. The definition includes a set of data fields that must be adhered to unless explicitly stated otherwise. In the event that an attempt is made to create an object lacking these attributes, the system will raise an error message and inform the user.
- **Each new object contains a unique primary key**.
  Django automatically creates a primary key, whose attribute is defined by the developers, for each newly created object. No two objects may have the same primary key and this is automatically tested at object creation.

- **Objects must pass validation tests.**
  As stated above, Django's testing module will be deployed to ensure proper data formatting for each object. Individual test cases are written for each object. For example, with every Lab object, there may exist a field that defines the days the lab is being held. In this case, the proper formatting of the *days* attribute must be formatted such that each day is represented by a single character and delimited by an empty space character. For example, "M W F" represents "Monday, Wednesday, Friday". If the given day data does not adhere to this format, the test is failed and the object is rejected.

3.2 - Bitnami

Bitnami is the preconfigured LightSail instance that is used for Django deployment on the remote server. Most of the potential headaches of implementing Bitnami will be making sure that the static files and file uploading mechanisms are properly serviced by Apache (Apache is the servicing engine for the preconfigured Bitnami). If the lab organizer can properly upload files, and the CSS and JS files are properly served, that will be most of the potential issues out of the way. We will also need to test the stability of the server by stress testing it with bulks of uploaded data and seeing if there will be any broken pipes or internal server shutdowns. In theory, the stress testing that goes well in the locally hosted application should go well in the remote application.

3.3 - Optimization

Optimization is the algorithm used for both TA score calculation and assignment within a generated schedule for the TA's. The optimization only directly interacts with the database. Thus, in terms of integration tests, we must see if it is both receiving the information from the database correctly and sending it back as expected. We need to test how the optimization actually places the TA's back into the database whenever the template schedule is generated. If the template schedule has majorly unexpected issues with the database inputs, for example, if it receives no TA's at all then the test should be able to account for that. Additionally, there needs to be a test if the optimization is actually sending the correct TA placements back to the database or not.

4 - USABILITY TESTING

Usability testing is focused on the end-user and how well they can interact with the system. The usability will highly depend on two major parts: user interface (UI) and user experiences (UX). UI gives the user a reasonable layout of functional buttons and a clear demonstration of back-end output. UX is focusing on providing users an interface with the perception of the utility, ease of use, and efficiency.

4.1 - User Interface

The user interface (UI) implemented in QuickSched is based on Bootstrap 5, a front-end framework. Under these circumstances, we can reference how other web apps that use Bootstrap do any stress testing. First, the UI issues like overflow and window conflicts are caused by moving elements. Referring to Bootstrap, we will test elements including *collapse*, *off-canvas*, *pop-up modal*, and *list drop-down group*. Elements to test include but are not limited to: moving elements, finding possible overflow elements, and ensuring all the elements will be displayed correctly. Another aspect of usability testing is to test all the buttons on the page, make sure they are fully functional, then test multiple pop-up windows. For example, the user may have clicked on the history pop-up, then clicked on another pop-up window without closing the history. In order for the application to pass this test, the team would need to ensure that the order of buttons clicked within the application has no bearing on their effectiveness. Finally, the team will be testing the transition of different pages. A page will need to have access to another potential use page and there must be no access to any “dead” pages, those that have the ability to access but not leave. These tests will be completed by potential end-users provided to the team by the client, Doctor Fofanov.

4.2 - User Experience

Once we have a stable and functional UI, the second major part is user experience (UX) testing. The two basic things that could affect UX in our app will be the layout of buttons and pages followed by the naming of the buttons. All pages should be in a reasonable layout that is not going to confuse users. The naming of a button should first consider a name that is commonly used to do the same function, then non-term words to show what the button will do. UX could be very subjective and abstract, and for this reason, the best way to test it is similar to testing the user interface. The UX will be tested by the same parties presented above, taking into account feedback and potential changes before the delivery of the final product.

4.3 - Optimization Usage

While the end-user never has to directly interact with the optimization algorithm, the algorithm is used indirectly. The optimization algorithm is used whenever a lab organizer has selected the TA's they intend to employ for a given semester. Based on the TA's they selected using the UI tools, the algorithm takes in those TA's and generates a set of labs assigned for the semester. The algorithm then finds the highest scoring TA for a given lab and places them in that lab, if they are not already assigned to a different lab. Once this process is completed it is output to the screen, where each TA is visually assigned to the optimal lab.

Usability testing for this will involve bringing in several people to act as Lab Organizers, to see if the UI tools, as well as visual aids on the site, make clear the process required to make the optimization algorithm generate a template schedule.

5 - CONCLUSION

The testing plan will consist of techniques of the unit, integration, and usability testing. The four modules we will focus on testing will be the UI, the database, the optimization algorithm, and the authentication system. We will carefully test the inner guts of these modules and their utilities as a first step. If these modules run well on the localhost, we will test how these will integrate within the LightSail instance that hosts our application. After everything works nicely with each other, we will do our best to break our application from the front end to foolproof the system as much as possible. Further increasing the usability and stability of our project is essential to the progression of our development and catering to our client's needs. Strengthening these modules along with further fool-proofing them will allow our application to serve lab organizers worldwide. We at Team Magisters are confident that this application can reach unwarranted potential after this polish, moreover, make a huge impact on university administrations and their important duties.