Auto Registration Documentation version 1.8

by: Nile Walker,

Preface

This documentation is intended for those that want to continue the development of the msu monitor system. For those intending to use it, a high-level explanation can be found in the User Requirements Section and in the User Manual.

In its previous version the system was able to monitor an unlimited number of students and courses to then notify them via email of availabilities. However the internal workings of these operations were very inefficient and presented stability issues when long operations i.e sending emails were blocking the server. The front end design also allowed users to potentially input nonsense data into the course subscription form which caused the system to behave unreliably.

As will be described in this document above the above problems have been solved but both still have room for improvement.

This version has also abandoned the automatic registration option until the software is more stable and user data protections can be implemented since its likely that plain text password it will need to be kept.

Introduction

At times finding availability for gen-ed or popular courses can be difficult while waiting for other students to drop the course this software is intended to provide an additional advantage to those who are willing to use it and automatically register for availabilities in their necessary classes.

The software works by creating a python based browser session and logging into websis. It then makes a list of all of the courses that it should check. And if an availability is there it will email the relevant students. All interactions with websites are done through a websis API built on top of existing modules.

Glossary

- *Websis* - The web-based tool that Morgan State University uses to manage student information and registration
- *CRN* - A six digit identifier for any course available in a given term.
- *Browser Session* - as a server-side storage of information that is desired to persist

throughout the user's interaction with the web site or web application.
- *API* - Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.
- *Subscriber* - Any student that has a course that they want to be notified about or registered for
- *TERM ID* - A value which websis uses to determine which semester a courses is in

User Requirements Definition

The msu monitor system currently provides one core service. Which is available through the site https://msu-register.glitch.me/

- Course subscription
  - Allows students to be added to a list of others that will be emailed whenever an availability is found in one of their courses. The service does not require the user provide a password.

System Architecture

Classes

`Manager`

The system is built and a publisher and subscriber type of approach. All interactions with websis besides registration are handled through a single account which is managed by the `Manager` class. This is done in order to prevent an excessive amount of websites traffic which could potentially impact the systems perfomance. And also to give users the option to not provide their password and only be notified of availabilities rather than registered for them.

`Student`

When a new student is added to the system an instance of the `Student` class is created. This object will hold their relevant account information, subscription status, and the courses that they are subscribed to.

`EmailThread`

Email thread is a thread child class that allows us to send an email without causing the server to wait on it to be sent. It's purpose and design is simple it has a setup method which loads and fills in the template associated with whatever status flag it was given and a send method which frees the thread and sends the email. It's definition can be found in `utils/notifications.py`.

Scripts

`download_courses.py`

It turns out that checking whether or not a course exist on websis is a time intensive task. So in order to lighten the load done with websis we decided to download all available course options at the start of each term. Running the script can take upwards of 20 minutes but the script will generate a file called `static/courses.json`. Which contains all available subjects courses and CRNs for a target term. This approach does introduce new limitations but these will be discussed in the System Evolution section.

Tools

`utils/dev.py`

In order to make changes on the fly to student subscriptions I added a small set of tools that can be accessed through the front end by developers. It's implementation is simple insecure and should be removed/redone in any real production deployment. There are 4 commands which can be accessed by inputting `admin:123456:${cmd_name}:${cmd_str}` it into the MSU username field.

```
Note: cmd_str cannot be empty
```

- clear: Remove all students from the manager and returns how many students were in the manager before hand.
- list: Return to list of the user names for every student in the system currently
- kill: Kills the program forcing it to restart
- remove: Removes the particular student from the manager when passed their user name as the cmd_str

System Requirements Specification

Function	Provide users a list of available courses
Description	In order to successfully subscribed to a course it is important that the subject and course number are provided accurately and it is easier to generate a list of all of the available courses rather than check if a course exists at the time of form submission
Inputs	The current term, Logged In Browser Session
Source	websis API, Manager Object
Outputs	A dictionary object with all available course subjects and course numbers

Destination	Frontend
Action	Log into the websis app, navigate to the search for courses page using HTML parser to find all subject options and then search all subjects for all course options
Requirements	The current term, Logged In Browser Session
Pre-Condition	None
Post-Condition	A drop-down menu in the front and is populated with all available course options separated by subject

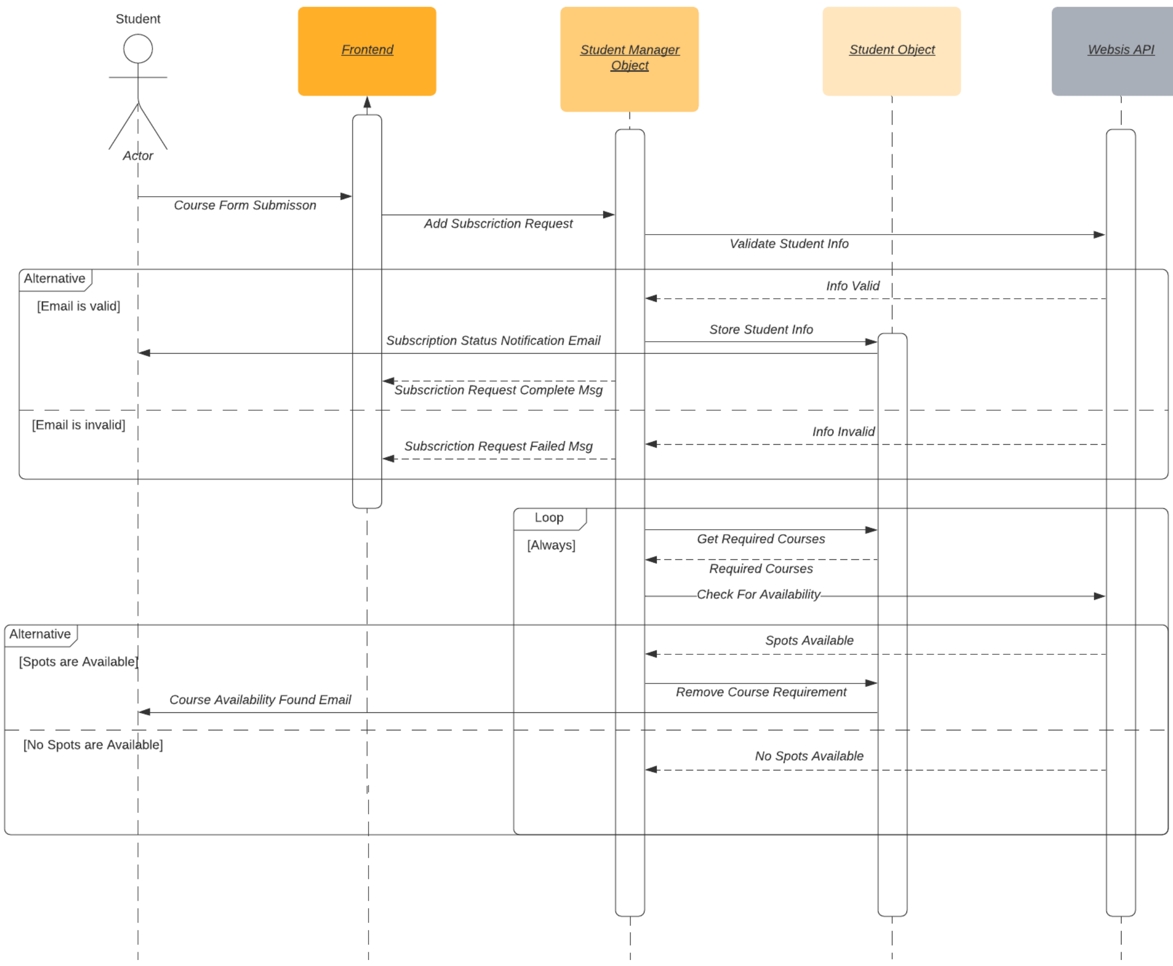
Function	Notify Students to changes in their subscriptions
Description	In order to appropriately interact with both in their course subscriptions are created, fufilled and dropped
Inputs	Student email, Subscription Status
Source	`Student` Object, `Manager`
Outputs	An Email
Destination	Users Email
Action	Open the appropriate template for the current subscription status, Substitute in the relevant information for the student and the circumstances, Send the email from the registration tools dedicated Gmail account
Requirements	Tools Gmail Login, Email Templates
Pre-Condition	User is unnotified
Post-Condition	User is notified

Function	Add Subscriber to waitlist on a course
Description	It's important to maintain a list of who needs what

Inputs	TERM_IN, SUBJECT, COURSE_ID, CRN,Student Username
Source	Frontend
Outputs	None
Destination	None
Action	When the form is submitted if a student with that user name exist in the system we will add the course to its list of needed courses otherwise the `Student` object will be created and of course added after that
Requirements	That the Frontend is operating properly
Pre-Condition	Student is not on the waitlist list for a course
Post-Condition	Student is on the waitlist and was notified via email

Function	Check for updates course availabilities
Description	This part of the system checks for changes in the spaces left and of course
Inputs	A list of all active `Student` objects
Source	`Manager`
Outputs	Emails
Destination	The Relevant Students
Action	If the manager browser session is not currently logged in log in. Create a list of all courses that need to be checked and their corresponding CRNs that are being checked for availability. Using the websis API request the courses page for that course and check for availability if it's found notify the relevant students via email
Requirements	That the manager browser session is logged in
Pre-Condition	Any availabilities have neither been filled and nor have the appropriate students been notified
Post-Condition	Any availabilities have either been filled and or the appropriate students have been notified

System Models



System Evolution

- Limiting course subscriptions to those that were downloaded by `download_courses.py` forces all subscriptions to be within a particular term. In general this will likely not be an issue but could be a problem for those attempting to register for winter/summer courses, since those running the system may choose to download courses for the more popular coming full semester and skip the winter/summer semesters.

  Despite this much of the internal architecture was built to handle courses from different terms and if a course from a different term was inputted into the `Manager` system it would likely appropriately notify and check for updates in the course. The largest issue is with downloading and serving options for multiple terms. As finding the Term ID requires manually going into the site and looking for the value of a specific HTML drop down. I encourage future developers to build an additional scraper that will automatically find that value and all available ones so that the MSU monitor system is just as versatile as traditional websis registration.

- As it's currently implemented the email thread class doesn't allow us to make any response if the email fails to send, the thread simply dies. This issue has been mitigated by validating the structure of the send address with regex before adding them to the system, but there is currently no way to check if the entered MSU username is actually tied to a student.
- This system assumes that the structure of websis as of Oct 2020 does not significantly change since the system has no tools in place to accommodate for changing links or html layout. The use of a seperated API however does allow mitigate this risk. Since changes to the webpages structure only require that these high-level functions be reimplemented rather than the entire system. And though it is still missing several features before it can be considered stable, It is very important that these sections of the software remain separate as we have no official association with University and changes could happen at any time without notice.
- We are also assuming that all development is being done through github. When a pull request to the master branch is closed, its code is synced to the glitch repo which updates the live site. While updating the glitch repo directly is an option it is not recommended as any future changes to the github will overwrite these changes.

Appendices

- Still not really sure how to write one of these

Index

- Or these