UCSD Academic Technology Innovation Student Internship Work Summaries

Podcast Enhancements

Purpose: Expand the functionality of UCSD's podcast system by adding additional features

Description: UCSD's podcast service automatically makes recordings of course lectures which students can later access for review. To expand the functionality of UCSD's podcast system to prepare for the new Kaltura system, I added the following features: adding a confirmation email when an instructor submits a request for podcasting a course, adding a checkbox to allow instructors to choose whether they want their podcasts to be available on canvas, adding an Admin tab to the navigation bar for authenticated users, creating a legend for the scheduled courses based on whether lectures are scheduled, recording scheduled, and recording completed. The confirmation email feature allows instructors to catch and resolve mistakes in their podcasting agreements well before courses.

Technicalities: The podcast backend is coded in C#, and is debugged and tested using the sql-console server which is accessed using the windows app for mac.

Grouper Templates

Purpose: Create Grouper Templates to Replace Stuacct Api

Description: Organization wide, UCSD ITS has opted to shift to Grouper, a database management system to replace databases like Stuacct. Part of this transition involves the creation of Grouper templates to efficiently retrieve information from Grouper which is then used by various important applications such as SAL(Student Account Lookup). The 'concurrentCourses' template retrieves a list of all the active courses from Grouper and their associated data such as end date, course name, department name, instructor names/emails, and facilities. The 'ritsCourses' template retrieves a list of RITS courses which are research courses at UCSD and its associated data such as course name, end date, username, student id, and status. Lastly, the 'describeSal' retrieves the course data for a selected course data such as course id, sections, instructors, facilities, and members.

Technicalities: The grouper templates are built through a Java program that uses native grouper functions combined with a large SQL query to retrieve the relevant data. pgAdmin was used to craft and test the SQL query.

Cloudlabs Allocations

Purpose: Query BBA database instead of Stuacct for allocations in Cloudlabs

Background: CloudLabs is a portfolio of several different virtual lab technologies: Amazon's AppStream (mostly used for CSE and ECE classes), Apporto (mostly used for SE and MAE classes), and Guacamole, which is used to create remote desktop views to on-campus Linux and Macintosh desktops – because the other options only handle Windows. A student's course allocations are used to generate the page in cloudlabs showing the facilities associated with their courses. Initially, the allocations were retrieved from the stuacct database, but UCSD ITS has decided to shelve the database in favour of using grouper, a database management application.

Task Description: To make this shift compatible with the cloudlabs application, I replaced the queries to stuacct with queries to another database BBA which holds student allocations. This entailed editing the cloudlabs aspx file which are in C# and using a simple SQL query to access the relevant data in BBA. SQL server Management Studio was used to view and test SQL queries to BBA. Furthermore, the cloudlabs code was tested on a remote server, sql-console.ucsd.edu, through the Windows App.

CINFO Generate Doorcodes Feature

Purpose: Add a feature to generate doorcodes for courses in cinfo.ucsd.edu site

Background: Instructional Technology Requests (CINFO) is the cinfo.ucsd.edu site used to request computer lab resources by faculty, and for ITS staff to manage those resources. In the LabAssignments Page, there is an overview of the various labs available and assigned.

Task Description: In this task, I added a feature to generate door codes having a drop down to select the term/quarter and a button to generate and download a csv file of the door codes for that term. This was accomplished by using a SQL Query to retrieve the necessary information from the cinfo database and integrating it with the C# .NET site's front end features.

Supporting the new SMP400 Recording Devices

Purpose: To update the podcasting backend to support the new SMP401 devices which UCSD is introducing to upgrade the podcast lecture capture devices in lecture halls.

Description: The new SMP401 devices require different formats and issue different responses. Firstly, the podcast database which contains information regarding the different recorders, lecture hall locations, scheduled lectures, and config definitions was expanded to include new columns such as recorder type and new specifications for the SMP401 which supports higher video resolution in 4K. I also updated the podcasting UI which schedules lectures in rooms to include a checkbox which sets the recorder type in the newly created column in the database. Moreover, the device internally schedules recordings by querying the podcasting system's backend to generate an iCalendar file which is then read by the device. The iCalendar file generation was updated to issue the correct file format depending on the recorder type: the new SMP401 or the older SMP350 devices. Lastly, parts of the podcast-extron-management which monitors the status of recording devices was expanded to accept the different responses to particular endpoints generated by the new SMP401 device.

Technicalities: This task required working with a diverse array of softwares and tools. In SQL Server Management, I used an extensive number of SQL queries to add columns and rows to update the relevant tables in the Podcast database. Moreover, expanding features in the podcasting system required changing the C# backend and testing in Windows Remote Desktop. The iCalendar files were verified by logging into the SMP device's internal UI, manually scheduling recordings, and checking the configuration. Furthermore, upgrading the podcast-extron-management required expanding the Perl code which checked the responses of various endpoints from the SMP device.

Rate Limiting

Purpose: Rate Limits Alerts from Podcast Processing Service

Background: For UCSD's podcast system for classes, the SMP processing application handles the post processing of podcasts once a recording is completed. Two files handle this. CheckExtron.pl - After a podcast completes recording, verify that it is complete on the server, and save information about the recording to *extron_tasks* table for later processing PostprocessExtron.pl - Upon finding an unclaimed extron_tasks, take ownership and download recording from extron for postprocessing. Both these files send email alerts if errors occur in their processes.

Task Description: In my task, I modified these Perl scripts to ensure that when an error occurs, a log file is generated. Moreover, if there is an existing log file and it's less than six hours old, then no email alert is sent. If the log file is older than six hours, then an email alert is sent. This time control was accomplished through using the native Perl time functions.

Stuacct Enhancements

Purpose: Expand the functionality of UCSD's stuacct database api by adding additional features

Description: Stuacct is a database in UCSD which holds course information for students. To expand stuacct api, I implemented a new post api operation for the roster endpoint which creates student allocations when an entry is inserted into the roster log by retrieving the courses from the provided section ids and the users and facilities of each course. If the course has a valid user and canvas feature, then an allocation is inserted. Once an allocation is posted, it also emits a response on the status of the post operation (OK, Partially OK, failure).

Technicalities: The stuacct api is coded in Perl DBI so I used Perl and the Dancer framework to construct the new post operation. The code is run and tested through a docker container. The response from the POST operation was tested and verified in the WSO2 API Manager. Finally, numerous unit tests were created to test invalid fields in the roster course and to check if an appropriate response was produced. Additionally, I implemented splunk logging into the api by configuring tokens for the various environments dev, qa, and prod to output useful data in strategic locations throughout the code.

Creating/Testing Python Base Images

Purpose: Create additional Python Parent Images and Tests for them

Background: ATI maintains the python parent images which are the 3 currently supported versions of Python. The parent image is a RedHat minimal python. To handle newer versions of Python, additional Python parent images were created for version 3.11, 3.12, and 3.13. Additionally, shell script based tests were needed to verify the versions in each docker image.

Task Description: I created docker images for the new python versions modeling them on older docker images for 3.10 and 3.9. However, underlying the base red hat image did natively support installing 3.13 and this required 3.13 to be installed through directly curling the 3.13 version from the Python website and installing the

necessary packages. To ensure correct versions for the Python base images, I created shell scripts to test pip, python, and xml versions for Python version 3.10, 3.11, 3.12, and 3.13. For python 3.13 image, there was a need to download manually some tools to test the xml version. The python images are projects that are built and run on the bamboo application, which automates building, testing, and deploying software. The tests were placed in the bamboo build plan and are designed to stop builds if the tests fail.