Canvas LTI Student Climate Dashboard Mark II

Final Report

Team Number: sdmay23-42
Client: Henry Duwe
Adviser: Nicholas Fila
Team Members:

Joshua Harvey – Team Manager & Resonance Scoring
Colin Hasbrook – Canvas API Wrapper & Sentiment Analysis
Elias Simpson – Kubernetes, Docker, Server, Infrastructure
Howard Chi – UI
Hailee Leonard - UI
Jonathan Giblin – Okta Authentication, Infrastructure, CI/CD

Team Email: sdmay23-42@iastate.edu
Team Website: https://sdmay23-42.sd.ece.iastate.edu/

Revised: 4/29/2023

Contents

List of Figures

1. Introduction

1.1 Problem Statement

The problem we are addressing is the lack of instructor and student resources that analyze student and course data to predict student resonance with the course. Instructors need a way to accurately understand students' experiences in their courses, so they can empathize with students, improve student interactions, and course design. Students need a tool that allows them to reflect on their class performance and use it as a basis for instructor communication. Currently, ISU's course management software Canvas, does not have a solution for in-depth automated data analysis that measures students' resonance with the course.

Over a semester, instructors and students are missing feedback that could dynamically influence student resonance with the course. The problem we are addressing is preventing instructors from accurately understanding their students' experiences with the course and how each could change their respective actions to improve student resonance and course achievement. Our solution uses automatically generated Journey Maps to visualize student and course data in an interactive application that enables instructors and students to discover new insights that can be used to change course design and student interactions with the course material.

1.2 Terms and Definitions

- Journey Map – The journey map is a visual representation of the resonance of a student or a group over time. The X-axis represents time; the Y-axis is the resonance for that point in the course. By visually representing the data, the instructor and student can gain new insights that can drive class design and personal behavioral change.
- Resonance – A loose term for quantifying student experiences with the course throughout the semester. Resonance is a value generated from Canvas data and data processing to gauge student experiences in the course. The model of resonance is based on achievement, sentiment, and engagement components, each provides information that can help an instructor understand how students are resonating with the course. The model can be customized to investigate resonance on a more granular level, or to look for subtle correlations between inputs.
- Persona – Students are sorted into groups based on various metrics; these groups are then graphed accordingly. Persona refers to a specific grouping of students. Default grouping is based according to students with similar resonance scores. Personas help the instructor understand how different groups of students resonate with the course.

1.4 Intended Users and Uses

The target audiences for the application are professors and students. Professors and students could both benefit from a tool that provides users the ability to reflect on course delivery and student experiences in the course. The tool could create meaningful interpretations of course data and a means for students and instructors to engage in productive conversations that lead to better class design and student outcomes.

2. Evolution of Project Design

Our project's evolution had some changes as we got further into coding the project and had a better understanding of it and what was possible using different elements of it. Through time constraints and

technical constraints by both members of the team and the software's that was used, some elements of the project design had to change throughout the course of the project.

2.1 Canvas API

One change to our design was what data was able to be retrieved from Canvas using the Canvas API. As it has become more important to have a full collection of data to use for our resonance scores, pulling sentiment statements was an element that had to change for the implementation of our project. Due to FERPA constraints as students, we are using a mock course to try mimic real data students would enter, but there were some difficulties we ran into that required a change in plan. One of the difficulties is there are no actual students in the course. As it is a mock course and all team members needed full access, everyone in the course is a co-teacher. Within Canvas a co-teacher cannot submit an assignment. Scores for assignments can be entered but not the student's response. Due to the confidentiality of students' scores in Canvas, even if there were students, there is no way to pull the text that a student enters in Canvas. For a workaround we used an Excel file to extract the sentiment statements. This works, theoretically, the same as if we were able to pull the responses directly from Canvas. This way the application can still get a sentiment score for each student even with the limitation that came with having a mock Canvas course.

Another limitation when using the Canvas API was the speed it takes to retrieve all the data from canvas. Since every assignment for each student must be gone through it can take around six minutes just to get all the needed data from Canvas. This is much slower than what the client was looking for out of our system and this led to us having to make a design change to account for this slow data collection process.

2.2 Resonance & Journey Map

Three core features we hoped to implement pertaining to resonance and journey map creation were a dynamic definition of resonance, fade dynamic, and trend enhancer. None of these features were implemented due to time, complexity, and team member capability issues. The dynamic definition of resonance would have allowed the user to define alternate components that would be used in the resonance calculation, more than the current selection. The fade dynamic would have applied weights to resonance scores based on temporal relevancy, allowing the user to determine how much or how little past and present scores impact the resonance score. The trend enhancer would have identified trends in the resonance scores of individual students and personas to help the user interpret the journey map and changes in resonance. One additional feature we hoped to implement was persona definition and categorization that works beyond the basic definitions used by the past team. These features would have made a significant difference and improved the functionality and user experience.

2.3 Front-end

For the front-end, there were enough changes asked of us, so we decided to start a new front-end application that uses more modern techniques. We implemented a Node.js and React.js project that works seamlessly with the existing backend architecture. Using primarily JavaScript allows us and future teams to create a more interactive and intuitive user experience quickly and easily. React.js has many existing open-source libraries and dependencies that have already created UI components that we can then customize to fit our needs. Since our client heavily asked for a more immersive interface, React.js seemed like the best solution.

To create a more immersive experience for the user, the client asked for the journey map to be more centralized and be the focus of the screen. We worked with the client and went through several iterations of how to maximize the journey map size, while still having all other functions be accessible and intuitive. We are taking advantage of many collapsible features to give the user a better experience than the previous iteration.

As the project progressed and limitations for back-end functionality became more apparent, the front-end was adjusted to best highlight the functionality that would be available. We did not have to change too many of the UI features. We briefly discussed if it would be reasonable to still implement UI components for back-end functionality that is expected in the future, but have it disabled. We explored this option because it would show users what will be available soon, and create less work for the next team, but decided to only show working features on the UI since the update to React.js makes it easier to create new UI components, we are already simplifying work that will be needed later.

When we began connecting the back-end to the front-end there were some difficulties that would require major refactoring of the code and layout of the project. So, for this case we decided to keep the projects separate but give the next team the foundations for both so they can focus on connecting the two. The front-end design is ready to take in data and we have implemented it with some hard-coded data so it could be properly tested.

3. Requirements

3.1 Functional requirements (specification)

1. Create journey map using stored data
2. Take user input to display student, persona, and class journeys
3. Take user input to customize journey map
4. Journey map should use defined weighting based on achievement, sentiment, engagement
5. Filter definitions can be saved and selected
6. Persona definitions can be saved and selected
7. The application should retrieve data directly from Canvas API
8. Categorize personas based on user input (proximity, student defined persona, best performing category, sentiment, engagement)

3.2 Resource Requirements

1. The application should use Kubernetes for cluster management
2. The application should be able to interact with Canvas API
3. The application should run on a server provided by ETG
4. The application should use Docker containerization
5. The application should be setup with CI/CD

3.3 Physical Requirements

1. The ETG provided Server should be reliable place to host the application

3.4 Aesthetic Requirements

1. Appealing and attractive layout, colors, and charting
2. Visualization choices represent meaningful data and/or user preferences
3. The web page design should be intuitive for pro and novice users

3.5 User Experiential Requirements

1. The application should be integrated with Canvas API
2. The application should be easy to use
3. The application should be able to compatible with specific needs
4. Page should load cached data within five seconds
5. Page should load non-cached data within thirty seconds

3.6 Economic/market Requirements

1. Our project will not be designed for sale or distribution outside of ISU
2. There are currently no economic requirements for our project. All costs are subject to ISU financing, currently no technologies will require payments or subscriptions.
3. Must meet FERPA and additional ISU requirements prior to official use at ISU.

3.7 UI Requirements

1. For use on a full screen browser
2. Fullscreen option for journey map
3. Accommodates multiple screen sizes
4. Interactive, allowing toggling and resizing
5. Accommodate different web browsers
6. *Professor*
    a. Save filter definition
    b. Load filter definition
    c. Change weights of assignments and resonance components
    d. Preview information when journey map data point clicked
    e. Toggle individual students
    f. Toggle individual groups
    g. Chart color coded journey map based on persona groups and/or students
    h. Clicking on data points provides granular data information

4. Engineering Standards

Below is our list of Engineering standards that we used during the development process and in the developed application. We identified software development standards learned from previous classes and are common solutions in industry. Following these standards will help steer our efforts and ensure we are developing a safe and secure application.

Agile – We employed agile values and principles. This helped us respond to user needs, prioritize tasks, and improve the quality of work.

FERPA – Followed all FERPA rules and guidelines when handling student data and information.

Kubernetes – Kubernetes is an open-source system for automating deployment, which affects the scope and management of “containerized” applications. This is a great system for any project to reduce unnecessary spending and to apply changes to our project. We continued to use Kubernetes for our project.

Docker – Docker is a platform that is used for developing, shipping, and running applications. This provides the application with different services for the project, being able to implement continuous integration of the application.

CI/CD - We used GitLab for CI/CD software development. Updates to the environment will reinforce accountability and maintainability.

SOLID Principles – Solid software design principles will allow for greater flexibility for later additions and modifications.

Acceptance / Integration Testing – We will continually ensure that our software is meeting customer requirements and integrating with the system.

5. Engineering Constraints

The client offered us the choice to build from the past team's project or start over with a new one. We chose to continue with the past team's project and their chosen technologies. Some of their technologies include Python, JavaScript, HTML, and SQL for programming languages, Docker and Kubernetes for containerized deployment, Canvas for receiving grading information, and Okta for user authentication. An important constraint was FERPA, and handling student data properly. We cannot store any student or instructor data, but rather we must pull the data from Canvas when we need it and handle it securely. Ensuring that the application can run and get this data is important to this project's implementation. For our iteration of the project, we did not get access to the official ISU Canvas nor their Okta authentication systems. The Canvas API has a rate limit for data that can be retrieved from Canvas. Our application experiences slow load times for fresh data pulls versus the application's ability to cache data once it has been pulled to speed up future data requests.

6. Security Concerns and Countermeasures

The previous team took great care in designing the application to be secure and meet FERPA requirements. Data is not kept on machines unless necessary. Only data with slow load times from the Canvas API is cached. Currently only cached data for some workloads is kept on the client's machine or it is cached for a short period of time behind a firewall and Kubernetes. Cached data is not stored after the session has ended. Data that is stored on the server is not personally identifiable. No data can be accessed prior to authentication. There may be additional requirements beyond the minimum specified by FERPA that ISU would require before the application could be approved for ISU Canvas, Okta, and deployment certification.

We must ensure that the Canvas API Tokens are handled appropriately. The Canvas API Token is required to start the application and enable access to the Canvas API. We must ensure that the token is never hard coded in the code, available via a bash history, and is only stored in an encrypted location.

We rely on Kubernetes Secrets and container environment variables to maintain a secure environment. We only temporarily store the Canvas access token inside a file when required. A deployment script overwrites a placeholder for the Canvas API token inside the Kubernetes secret file. Kubernetes secrets are automatically encrypted when accessing the secret, there is no practical way a bad actor could know the name of the stored secret and decrypt the Canvas API token, we do not believe this to be a likely attack vector.

7. Implementation Details

7.1 Implementation Overview

The implementation goals of Mark II are different from those of Mark I. Mark I was the first group to engage with this problem and started from scratch. The previous team did a great job, but the client had additional goals for the project. The client gave us three objectives for the Mark II iteration, a more efficient user interface, an improved algorithm for creating journey maps, and a student facing visualization. More information and implementation details about the Mark I implementation can found here, https://sddec21-19.sd.ece.iastate.edu/docs.html. The Mark I implementation took advantage of great technologies and software engineering techniques; however, we ran into significant issues running their application. Starting last Fall, it took a significant amount of resources to resolve issues and get the application running this semester. Original planning did not account for these unforeseen issues, nevertheless we would like to share what we have achieved for the Mark II implementation.

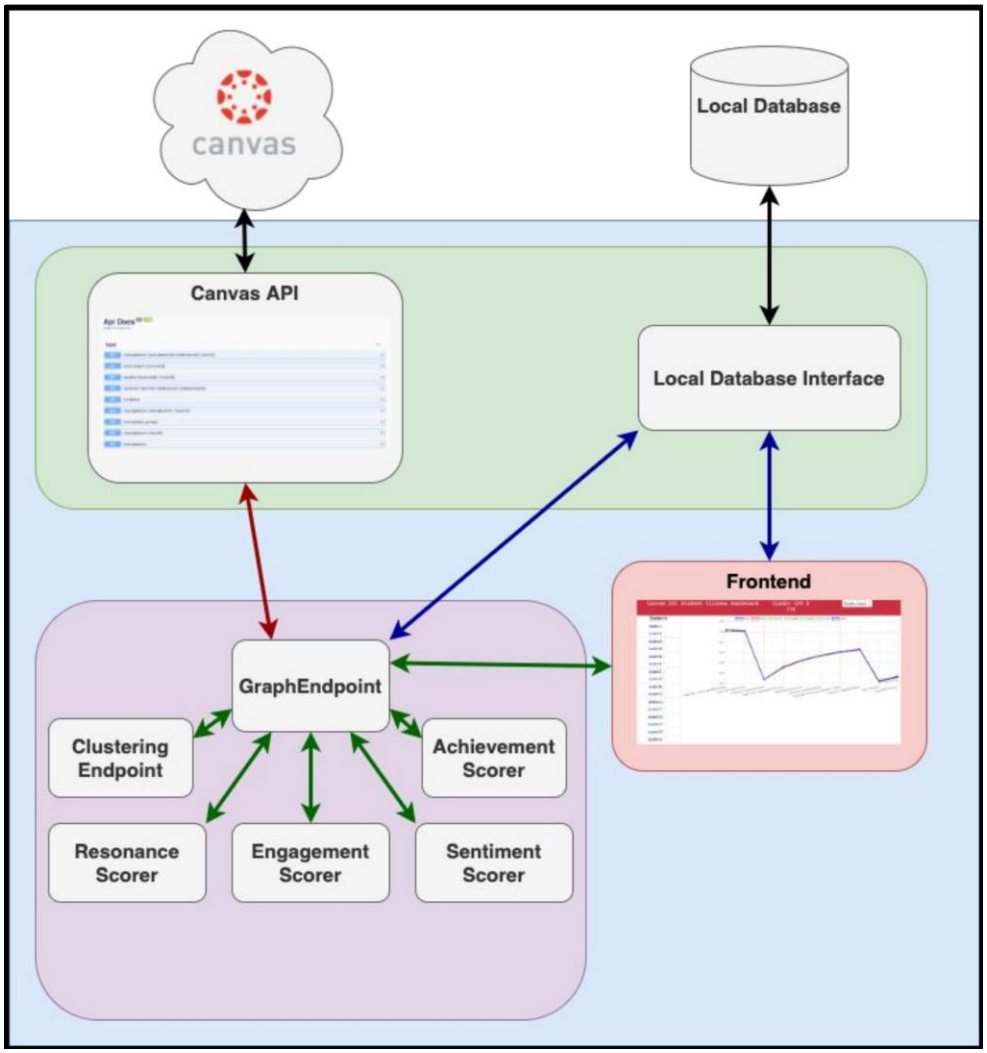

Figure 1: Overview of system implementation, link

7.2 Canvas API

To get the data from canvas the Canvas API was used. This is a collection of calls that can pull various aspects from Canvas that can be stored and then analyzed.

To implement this into the application the first step of this is to have a Canvas token. This is a key that everyone can produce to access the different data in their own personal canvas. There are security and privacy restraints within this that limit the data that can be retrieved. This data is different for a student from a class compared to an instructor.

To get the course information the course id must be retrieved so that a course object can be created. Doing this allows the application to get different information about the course like different submissions and the grades from each of these submissions. These objects are created for each student so that the application can retrieve the relevant information needed to do a sentiment analysis for each individual student in a course.

To analyze the sentiment text from a student there had to be a way to get the sentiment responses from these students and score them. The python's Vader Lexicon sentiment analyzer to get these scores. This will produce four scores based off the text based off negativity, neutrality, positivity, and compound. The negative, positive, and neutral scores will add up to one and the compound score is a weighted score that utilizes all these scores, so this is what was used for the sentiment analysis. A compound score will be between -1 and 1 where a lower score indicates a negative sentiment and a higher score being a positive one. Another metric that is used is if the score is above 0.05 this is considered a positive score and less than -0.05 would be a negative one. This gave the application a way to score the responses of students and gives insights to the instruction on their response without having to read each response by every student.

7.3 Resonance & Journey Map

An improvement to the journey map creation and resonance calculation was the modification of section score values for achievement, sentiment, and engagement. The change we made was expanding the standardization of the resonance value range to the class's grade scale. This change affected ten or so files, modifying how those components translate raw data to data that can be plotted on a journey map. By increasing the number of categories from three to five and using the grade scale to determine value to ranking relationship. These changes were targeted to bring greater definition to the resonance scores by increasing the categories for which raw data scores could be standardized for resonance plotting on the journey map. The code changes were added to the repository but were not activated due to not having time to verify accuracy of changes. We hope a future team can build upon the changes made.

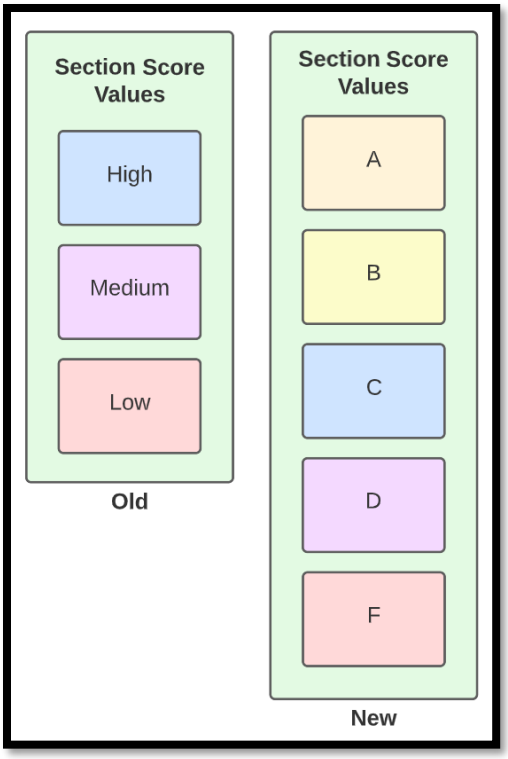

Figure 2: Expansion of Section Score Values from three to five, now match grade scale.

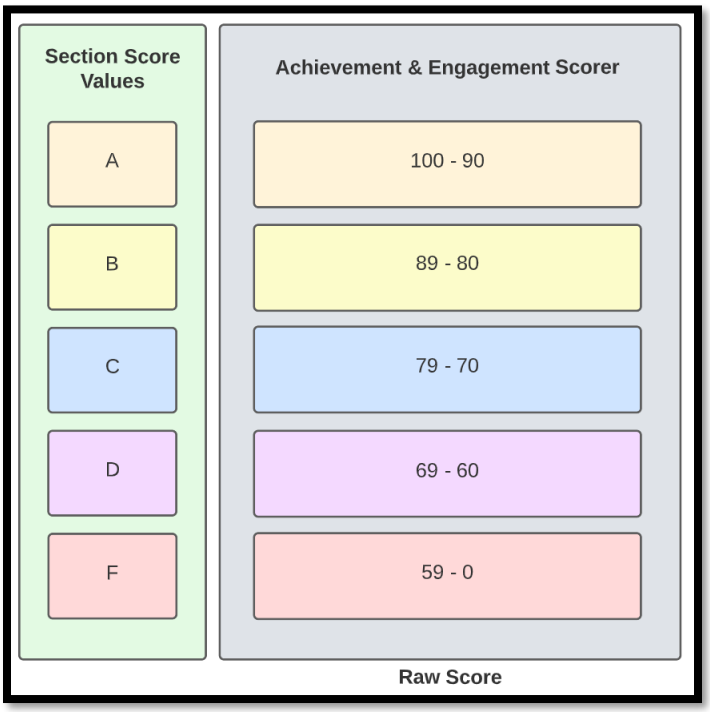

Figure 3: Achievement and Engagement Scorer visual representation using new Section Score Values.

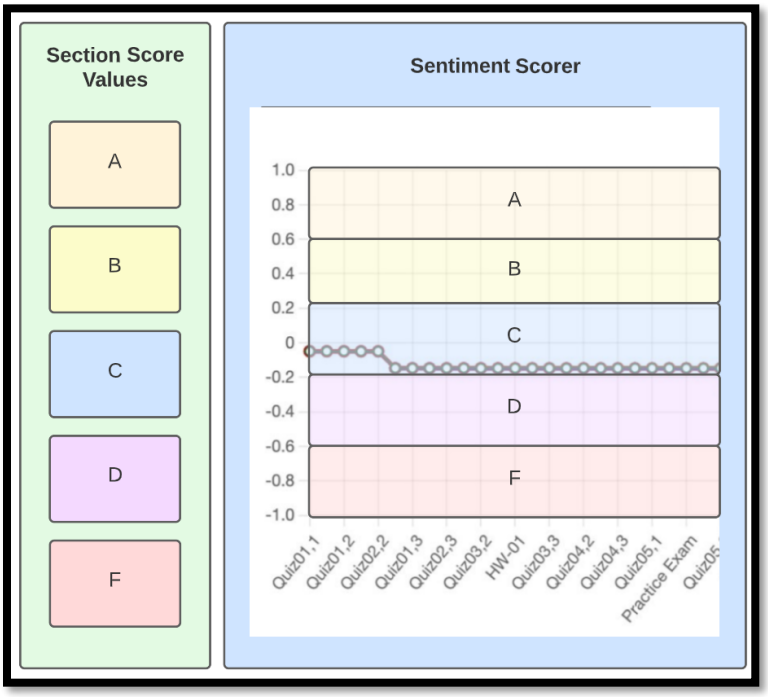

Figure 4: Sentiment Scorer visual representation using new Section Score Values.

7.4 Front-end

To start implementing the front-end we created a Node.js and React.js project. We chose to use JavaScript because it is the most common front-end programming language. Node.js is a framework that handles many of the networking and server connections. React.js is a front-end library that makes building UI components faster and more seamless. We also utilized many pre-built UI components in the Material UI library. Material UI has many pre-built React.js components that can be customized to fit individual application needs. React.js also integrates well with Express, which we use to make calls to the back-end.

We first made a prototype of the web application to get feedback from the client. We went through several iterations to get a final prototype that seemed reasonable to complete within the project's timeframe. From there, we built the basic UI components from the prototype. Once that was completed, we started connecting any components to the back-end to be able to display real data. We did run into issues in this step with the current project layout. We decided to keep the two projects separate because the refactoring would be beyond the scope and limits of this project. Instead, we used some hard-coded data to be able to show what the front-end will look like once the front and back end are connected.

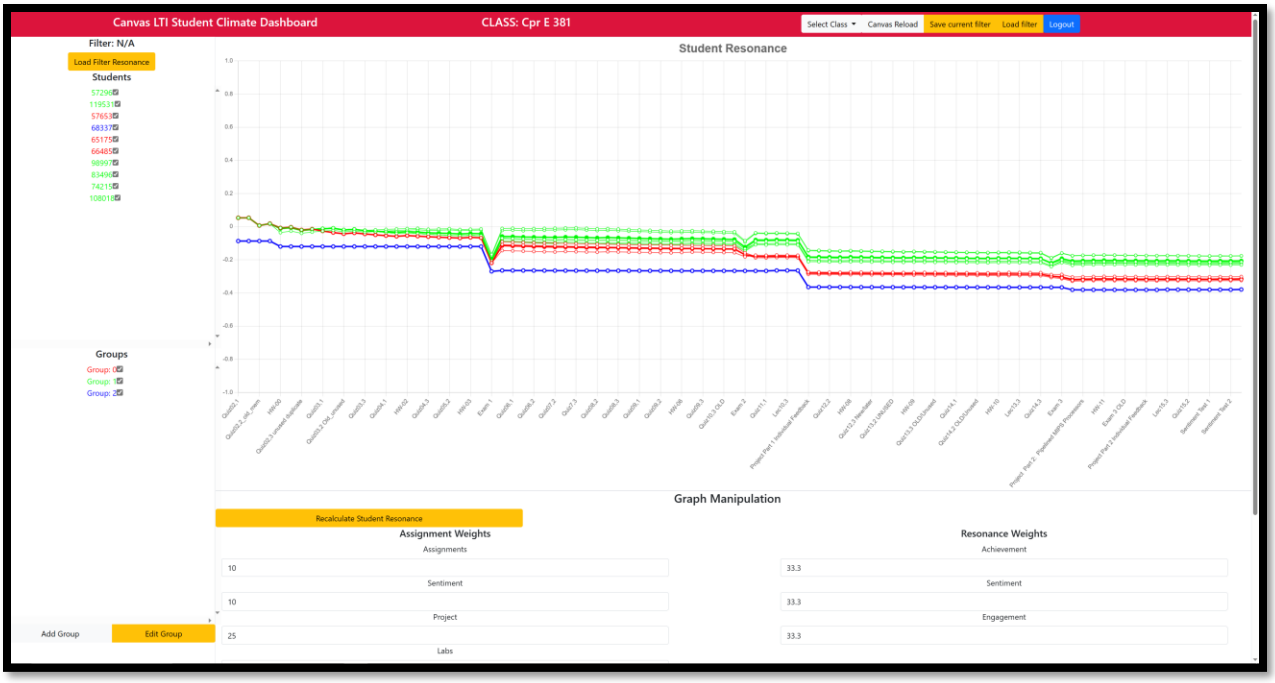

Figure 5: Running Mark I application with Journey Map.

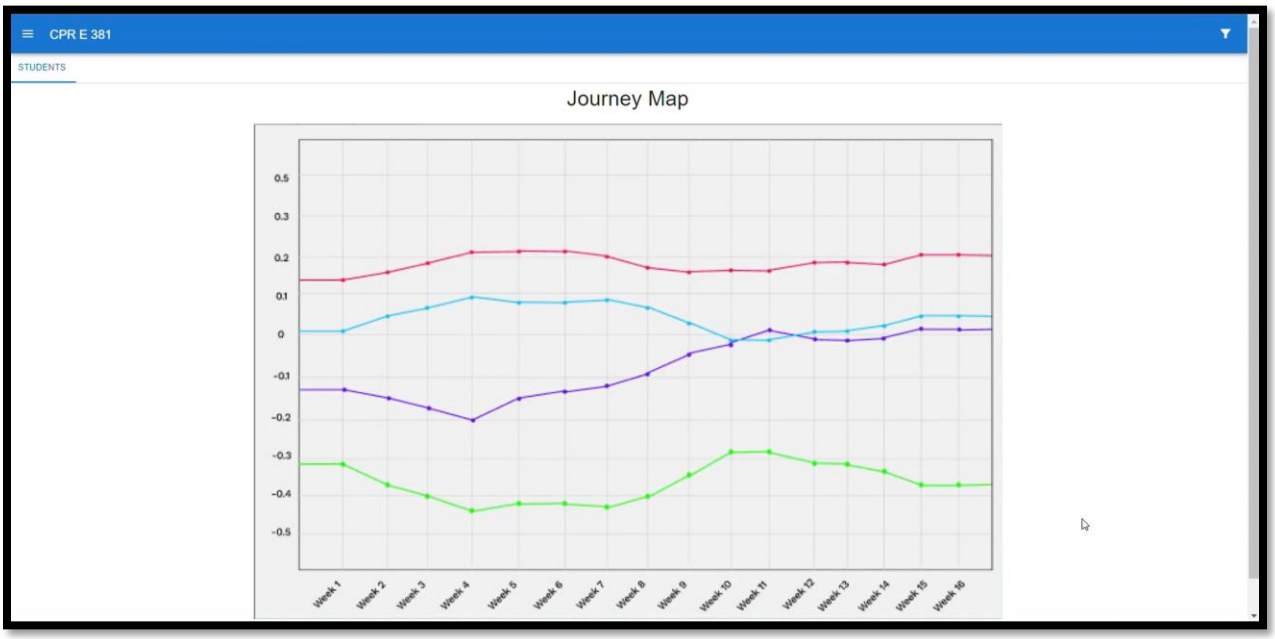

Figure 6: New Prototype Application showcasing new front-end UI, some features not shown.

7.5 Server & Microservices

Several challenges were overcome to run the application on a fresh server. Due to the complexity of the stack, familiarity with Kubernetes, Docker, Python, YAML, micro-service architecture, APIs (C# and Python), frontend (JS, html), and server management was needed to address and solve a series of issues with getting the application to build, deploy, and load.

1. Installation
    a. Deprecated packages: The installation of several dependencies was failing due to changes in library names and compatibility issues. The problematic imports were identified and addressed in application-level Python files.
    b. Server requirements (Protoc, grpc-web): Some requirements weren't detailed in the installation guide. Through troubleshooting, it was determined that the deploy errors were occurring because the server needed additional technologies for the microservices to be able to communicate (protobufs).
    c. Deprecated documentation related to Kubernetes container-runtime: In the past, Kubernetes used Docker for container-runtime. The project documentation referred to Docker's sources for allowing insecure registries (our own microservice APIs) to communicate with the Kubernetes cluster. Kubernetes now uses containerd for container-runtime. The daemon for the correct library was updated to solve the access issues.
2. Missing documents and command statements
    a. Dockerfile: Some images were missing from the registry after build and execution of Docker scripts. A Dockerfile for the service was created to copy dependencies and build/push the image to the registry. Kubernetes was then able to use the image for pod deployment.
    b. Push statement in script: Some key lines were missing from the image scripts. In one case, an image was created but not added to a registry. The cause of the problem was determined to be a lack of command to push the Docker image.
    c. Dev infrastructure mixed with prod: In one case, the Docker script was referencing a "test" version of the YAML file for the architecture's envoy proxy, resulting in the production nodePort being unavailable. The team inherited a lot of unused, outdated, and "commented" code, which made troubleshooting time consuming and, in this case, caused direct issues in the application.
3. Debugging (tokens, IDs, SQL access)
    a. Missing token parameters: Once Kubernetes pods were deployed and functioning properly, several runtime errors occurred. A missing line in the deployment YAML was discovered to be the cause of an undefined Canvas token in runtime.
    b. Runtime errors: Null errors from hard-coded ID values had to be handled on a case-by-case basis. Ideally, the app would run without hardcoded values for courses and students.
    c. SQL Access: We were initially unable to access the database from the front end due to user configurations. A new user was created with special domain permissions.

```
vm-user@sdmay23-42:~$ sudo kubectl get deployment --all-namespaces
NAMESPACE              NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
achievement-scorer     achievement-scorer     1/1     1            1           25d
canvas-api             canvas-api             1/1     1            1           25d
clustering-endpoint    clustering-endpoint    1/1     1            1           25d
engagement-scorer      engagement-scorer      1/1     1            1           25d
frontend               frontend               1/1     1            1           25d
graph-endpoint-envoy   graph-endpoint-envoy   3/3     3            3           25d
graph-endpoint         graph-endpoint         3/3     3            3           25d
kube-system            coredns                2/2     2            2           25d
resonance-scorer       resonance-scorer       1/1     1            1           25d
sentiment-scorer       sentiment-scorer       1/1     1            1           25d
sql-connection         sql-connection         1/1     1            1           25d
```

Figure 7: Kubernetes command to view deployments with results.

For each challenge getting the application to load correctly, we learned about practices for maintaining, accessing, and debugging a Kubernetes cluster with 11 deployed pods. For example, the Kubernetes cluster uses namespaces to separate the pod deployments. It was initially impossible to use the command line to identify pods and troubleshoot deployment issues because pods deployed inside of a namespace won't show up unless the command explicitly refers to the namespace. Lessons learned from the process have been detailed in Appendix I.

8. Testing Process and Test Results

We optimized the existing tests and frameworks to work with the changes made for Mark II. Testing was completed on the Canvas API, API Speed, SQL, Data Analysis Pipeline, and Front-end. Tests were conducted automatically and manually.

Canvas API

The Canvas API Speed is very similar to the Mark I implementation. Non-cached and numerous API calls take 15+ seconds due to sever throttling of the Canvas API. Cached API calls load very quickly, usually less than one second. Since the Canvas API is an external application that was implemented into our project it is already tested externally and no changes can be made. Testing was done to make sure that the correct data was being pulled and processed correctly.

SQL Testing

The goal of SQL testing was to verify tables are storing data correctly. This ensured that the tables were created correctly and storing the data correctly, there were no issues between the frontend and the backend. The tests matched what was being stored with what was being displayed on the front-end with the correct values being shown on the journey maps.

Data Analysis Pipeline

The data analysis pipeline we tested each microservice for expected outputs using the protobuf message. The tests tried various values that checked if the final value matches the expected and that the application did not crash. We also used simple processes for checking the application runs as expected for proper users and different web browsers. This was done using unit testing. The application can run

on popular web browsers and invalid user inputs such as trying to login from a user with no permissions are blocked.

Front End

The front end was tested to ensure that different users have the proper permissions. This will verify speeds for creating the journey map along with the proper filters and data. A VPN was used for the application, and this helps to make sure the users cannot access other data. Trying to log in as a different user is not possible, which means that it is working properly. We also used different browsers to make sure that the journey map still displayed properly on different ones. This was done by doing use case testing and after looking at the application on different browsers and so no results that resulted in us having to make any changes to the application.

9. Related Literature and Products

Journey maps are traditionally used in a business context to aggregate customer experiences and identify aspects of customer service that can be improved. Their application is highly conceptual and theoretical, a prompt which helps managers plan and map out a customer experience. Some journey maps based on specific user responses and data have been applied to a university setting. However, existing commercial products focus on a more general student experience—such as onboarding—and less with a specific course. For example, Rains Media offers a product to help analyze the recruitment of prospective students visualized in figure 8.

Figure 8: Screen shot of the journey map tool on Rains Media website.

The only known attempts at applying data-based journey maps to a course setting are part of a grant from the National Science Foundation. The grant enabled professors and teaching assistants the time and resources to plan and implement journey maps for engineering courses. The drawbacks of these early attempts at journey maps were that they involved manual accumulation, recording, and mapping of data. The study does indicate that journey maps improve professor empathy with students and leads to greater understanding of student issues and performance.

We thus have the benefit and disadvantage of being pioneers in this space. In general, an academic course offers a high potential reward for the application of a journey map, due to the abundance of data provided throughout a semester.

For our project, we will follow the work of a previous Senior Design group that worked on the first iteration of this LTI Canvas Resonance tool. For their advantage it has a working interface for the instructor. The journey map lays out the assignments and lets the instructor change some features. This iteration did have some shortcomings, which is our goal to improve in our iteration. The first of these was the issue with the resonance algorithm. As it stands it does not do a wonderful job giving the correct resonance information and needs to be improved in that regard. Second would be the issue that it currently cannot connect directly within the Canvas API. The work around that they used was to use their own Okta to run the application, but this would not serve the benefit of students as is the goal of the design. The final shortcoming is it does not have a student interface, so nothing is provided to the students.

10. Conclusion

The goals of the Mark II iteration have not been met; however, each team member was able to contribute to improvements that strengthened the state of the project, and any future work on the application. Unforeseen complications with setting up the application, complexity of the past team's implementation, and lack of team cohesion limited the success of project. The team was able to set up a mostly working Mark I application, make infrastructure improvements, and prototype changes to Canvas API calls, sentiment analysis, resonance scoring, and UI features. We hope the improvements and work completed on Mark II help future work on the project and all team members gained the experiences that will help them in their careers.

Appendix I - Operation Manual

Installation of Application

Environment Set Up

In the following section of the document, we go through the process of setting up our application from scratch. Our application is Kubernetes based and thus we make two key assumptions in that section:

- You have access to a vanilla Kubernetes cluster using the Kubectl command line tool
- The Kubernetes cluster and you alike have access to a container registry on which to store container images

Because Kubernetes administration is not a commonly taught skill set at the time of writing, we will briefly go over how to install and set up a necessary cluster and container registry in the context of an Ubuntu 18.04/20.04 Virtual Machine.

In order to set up the cluster and container registry, the following steps will need to be followed:

1. Install Outside Tools
    a. Install Docker
    b. Install Docker-compose
    c. Install Kubernetes (Kubelet, Kubectl, Kubeadm)
2. Set Up Cluster Infrastructure
    a. Initialize the Kubernetes Cluster
    b. Setup Container Networking Interface
    c. Setup Container Registry

Install Outside Tools

Our infrastructure can be set up quite easily without too much understanding of what is going on under the hood. To accomplish this, we will rely on some external packages to do some of the heavy lifting for running the container registry (where we will house our application images -think binaries) and Kubernetes cluster instantiation/administration. Thus, we will need to install the following packages on the VM which the application will be deployed:

- Docker
- Docker-Compose
- Kubernetes (Kubelet, Kubectl, Kubeadm)
- Protoc

Installing Docker on Ubuntu 18.04/20.04

The first thing we want to do is install docker onto the machine which will be running our cluster. This will allow Kubernetes to use docker as the container runtime and allow you to build the application from the VM if so desired. To install docker on ubuntu, follow the instructions at this link: https://docs.docker.com/engine/install/ubuntu/ . For Linux distributions, it is recommended to add the user to the docker users' group to manage docker as a non-root user account (link here: https://docs.docker.com/engine/install/linux-postinstall/).

Installing Docker-Compose on Ubuntu 18.04/20.04

The docker-compose toolset allows you to manage containerized applications on a small scale. In our specific usage however, we are simply going to use the command line tool to spin up a container registry that we can push and pull our container images to. To install docker compose, navigate to the link here (https://docs.docker.com/compose/install/), click on our operating system (for us, Linux) in the 'Install Compose' section, and follow the instructions there.

Installing Kubernetes on Ubuntu 18.04/20.04

Kubernetes is the container management system that we will be using to manage the state of our application. We have provided scripts and YAML files that will set up the Kubernetes cluster for you if not already done. Before using the scripts however, we need to install the command line tools that the script relies on. To install these tools, follow the instructions at the link provided here: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/.

Installing Protoc on Ubuntu 18.04/20.04

Using snap, you can install Protoc library with `sudo snap install protobuf –classic`. In addition, *GRPC web* needs to be installed. The binary should go directly into usr/local/bin. It should be given permissions with: `chmod +x`.

Set Up Cluster Infrastructure

At this point you should have kubelet, kubeadm, kubectl, and Protoc installed on your VM, and we can now use the scripts in the git repository to set up the cluster and container registry.

Initialize the Kubernetes Cluster

To initialize the cluster, clone the git repository to the VM on which we are going to run our application. Inside the cloned repository, navigate to the Infrastructure/Cluster_Specification/Kubeadm directory. Run the create-cluster.sh file and wait for completion. This will create a Kubernetes cluster for you with the help of kubeadm and then set the proper permissions and file locations for you to be able to control the cluster. Before running the command, execute `swapoff –a` to prevent configuration issues. Once the cluster is initiated, the taint needs to be removed from the control plane. Execute: `kubectl taint nodes --all node-role.kubernetes.io/control-plane-`

Setup Container Networking Interface

Next we need to set up the Container Networking interface so that our microservices have a way to communicate with each other. There are tons of different CNIs you could choose to use for different needs, however our application does not require anything fancy, so we chose to use the most basic (and usually the default) CNI - flannel. To install flannel, navigate to CNI directory parallel to the Kubeadm directory we were just in (Infrastructure/Cluster_Specification/CNI). Once there, run the command:

`kubectl apply -f flannel.yaml`

This will create the Kubernetes objects necessary and install flannel onto the cluster.

Setup Container Registry

Finally we need to set up the container registry that will house the images that we create for our microservices. The container registry will simultaneously allow the Kubernetes cluster to (pull and then run) our application as we specify.

To create the container registry on the VM, navigate to the Infrastructure/Cluster_Specification/Container_Registry directory and follow the steps below:

1. Create a directory named data
2. Run: `docker-compose up`

This will open a container registry on port 5000 for pushing and pulling to. To use this with the just-installed docker instance, edit the daemon for the registry package. Docker is now deprecated for Kubernetes, instead, update the daemon for containered to allow unsecure registries. You can verify that the "container-runtime" is containered and not Docker by running: `kubectl get nodes –o wide`.

At this point, you have an Ubuntu 18.04 VM provisioned to you that houses a Kubernetes cluster with an installed CNI and image registry. Now we can begin setting up the actual application.

Application Set Up

In order to set up the application, three steps are required:

1. Build the Application's Microservices and Push to Image Registry
2. Deploy the Application's Microservices and Kubernetes Objects
3. Setup MySQL Database

Build Application

To make building the application easy, there is a 'build-application.sh' file located in the root of the git repository once cloned. To use this script, simply run:

```
./build-application <REPOSITORY NAME>
```

And wait for the text "BUILD COMPLETE" to be displayed. On the first run, this may take a while. Afterwards, this will take substantially less time. It is important to note that you should not include the angle brackets around the repository name when running this script. For example:

```
./build-application sddec21-19.ece.iastate.edu:5000
```

Is correct while:

```
./build-application <sddec21-19.ece.iastate.edu:5000>
```

Is incorrect.

If you've already built most of the application but just want to build a portion, navigate to the directory

containing the code for the portion of the application you wish to build and run the script that begins with the text “image_registry_dockerize”, passing in the image registry address as an argument just as in the “build-application.sh” script. For example, if we wish to just update the Graph Endpoint Service, we:

1. Navigate to the directory “Data_Analysis/GraphEndpoint/”
2. Run: ./image_registry_dockerize_graph_endpoint.sh <REPOSITORY NAME> with the image repository name you are using.
3. The pods for the services are set to “imagePullPolicy: Always.” However, a name change to the image version is expected to trigger an update. You can manually update the pod with the build image with: kubectl rollout restart deployment <pod name> --namespace=<namespace name>

Deploy Application

At this point in the setup, we have a Kubernetes cluster, a container network interface, and an image registry filled with container images that we just built in the last section. All that is left now is to:

1. Set proper Access Token from Canvas
2. Deploy Microservices to Kubernetes

Setting Proper Access Token from Canvas

In order to set the proper access token from canvas, we first need to retrieve the token from canvas. To do this, open and log into canvas, then follow the steps below:

1. Generate your own Canvas Access Token
    a. Click on the Account button on the left side of the screen.
    b. Choose the Settings selection that appears on the pop-up menu
    c. Under Approved Integrations click the button that says + New Access Token
        i. In the Purpose Textbox, say “Canvas LTI Student Climate Dashboard”
        ii. In the Expires Box, choose when you want to need to refresh the token
    d. Generate this token and copy the value to the right of Token
2. Set the Access Token in Application
    a. On the VM, go to the Infrastructure/Application_Infrastructure directory
    b. Open the bearer_token.txt file, delete all contents, and place only your token in the file. Save.

Deploying Microservices to Kubernetes

Now that the token has been saved to the file, we are ready to deploy the application by following the steps below:

1. Run the deploy-application.sh file, passing in the image repository address that you are using, like below:

    ./deploy-application sddec21-19.ece.iastate.edu:5000

2. Delete your token from the bearer_token.txt file.

The writing your token to the bearer_token.txt file and then deleting it accomplishes a few things with respect to security:

- Keeps the plaintext representation of the token on the computer to a minimum.
- Hides the token from the bash_history file and anyone who can view your command history.

It is important to note that anyone who has the Kubernetes token can access this token, so it is important not to distribute this token without care.

Set Up MySQL Database

1. Install MySQL Database
    a. apt install mysql-server
    b. apt install mysql-client
2. Ensure that the server has the correct IPTables/permissions for your specific machine
3. Install MySQL Workbench (optional but easier than command line client)
4. In either workbench or client, log in as root user (or set up a new user)
5. Additionally, add database information in "Data_Analysis/SQLConnection/config.ini"
6. Setup Database Schema:

```
create table if not exists users (
        userID int unique not null,
        defaultCourse int,
        bearerToken int not null,
        primary key (userID)
);

create table if not exists courses (
        courseID int not null,
        userID int not null,
        primary key (courseID),
        foreign key (userID) references users(userID) on delete cascade
);

create table if not exists categories (
        uniquePairID int not null auto_increment,
        groupNum int not null,
        groupName varchar(256) not null,
        filterID int not null,
        courseID int not null,
        userID int not null,
        primary key (uniquePairID),
        foreign key (filterID) references filters(filterID) on delete cascade,
        foreign key (courseID) references courses(courseID) on delete cascade,
        foreign key (userID) references users(userID) on delete cascade
);

create table if not exists students (
        uniquePairID int not null auto_increment,
        studentID int not null,
        courseID int not null,
        initialResonance double,
```

```
        primary key (uniquePairID),
        foreign key (courseID) references courses(courseID) on delete cascade
);

create table if not exists studentGroups (
        pairID int not null auto_increment,
        studentID int not null,
        groupID int not null,
        primary key (pairID),
        foreign key (studentID) references students(uniquePairID) on delete cascade,
        foreign key (groupID) references categories(uniquePairID) on delete cascade
);

create table if not exists filters (
        filterID int not null auto_increment,
        filterName varchar(256) not null,
        courseID int not null,
        userID int not null,
        assignmentWeights varchar(256) not null,
        achievementWeight int not null,
        sentimentWeight int not null,
        engagementWeight int not null,
        primary key (filterID),
        foreign key (courseID) references courses(courseID) on delete cascade,
        foreign key (userID) references users(userID) on delete cascade
);
```

User Manual

Updating Canvas Access Token

In order to update the canvas access token with a new token, simply:

1. Open the bearer_token.txt file.
2. Delete all contents.
3. Place your new token inside of the file with no extra characters.
4. Save.
5. Deploy the new K8s Secret by running:

```
cat bearer_secret.yaml | sed -e "s/<BEARER_TOKEN>/${BEARER_TOKEN}/" |
kubectl apply -f -
```

6. All as one line.
7. Delete your token from the “bearer_token.txt” file.

Frontend

1. How to access webpage
    a. On a modern web browser, enter the hostname of your cluster into the URL bar and append *:30011* to the end of the URL.
    b. Navigating to this page takes the user to the Okta-managed login page.

i. NOTE: The Okta-managed login page must be configured so that login is redirected to their Okta Organization of choice. The user either needs to create and manage their own Okta organization or set it up with the university's Okta credentials. See #2 for more details.

c. Enter user credentials and click "Log In". If login is successful, Okta will return a login cookie and the frontend application will automatically redirect the user to *:30011/graph*. This page is inaccessible without the Okta authentication cookie.

d. The user is now logged in and viewing the main page of the application.

2. How to setup an Okta Login
    a. This process requires making edits to the program's files. Settings for Okta authentication are in the file "/FrontendApplication/server.js".
    b. The user must be an administrator for the Okta organization they would like to connect to the application. Users can create and manage their own Okta organization.
    c. To connect the Okta to the application, first make a new Open ID Connect App Integration in the Okta if you do not have one already. This can be done by going to the Okta Dashboard, clicking on "SSO Apps", then selecting "Create App Integration".
        i. Once created, you can view the configuration and relevant settings for this integration by selecting it from the list of applications.
        ii. Make sure that the Okta integration uses Client secret for authentication.
    d. The first change to make in server.js is swapping the issuer in the Okta Configuration section for your own issuer. This is essentially the URL to your okta and is unique for each Okta. This is necessary for making sure the application is looking towards your Okta for authentication.
    e. The next set of changes is to client_id and client_secret in the Okta Config section. Client id is provided in the client credentials of the Okta integration. Navigate to that section and copy the id over into server.js. As for client secret, it is provided just below the client credentials section. If one does not already exist, you will need to make a new one. Once a secret exists, just copy it over into the corresponding spot in the server.js file.
    f. The last set of changes in the Okta Config is to the appBaseUrl and the redirect_uri. The appBaseUrl should be set to whatever the base URL for your application is. This is the URL you enter when you want to access the application. As for the redirect_uri, this will need to be configured in both the server.js file and in the Okta integration. This will end up being whatever the base URL is with "/authorization-code/callback" added to the end. I.E., if the base URL is http://sdmay23-42.ece.iastate.edu:30011, the redirect URI is going to be http://sdmay23-42.ece.iastate.edu:30011/authorization-code/callback.
        i. If you are having trouble find the redirect uri in the Okta integration, it is located on the general tab in the login section. You need to setup the sign-in redirect URI for the authorization to work.
    g. Once all the above changes are made, any valid log-in to the Okta should allow you to access the application.

Kubernetes

Useful commands

kubectl get pods/svc/nodes/deployments --all-namespaces : gives an overview of running services

kubectl describe <pod name> --namespace=<namespace name>: gives deeper view of pod, useful for troubleshooting deployment errors

kubeadm reset: will kill any running Kubernetes nodes.

sudo kubectl rollout restart deployment <pod/deployment name> --namespace=<namespace name>: Will update a pod with the newest available image in the registry.

Error logs

Logs for errors from applications running on Kubernetes may not be accessible through the browser console. Messages can be viewed through Kubernetes' `log` command:

sudo kubectl logs <pod name> --namespace=<namespace name>

Troubleshooting deployments

Several layers of "infrastructure as code" facilitate the deployment through simple scripts. If an issue arises in deployment, it can be helpful to understand the deployment flow.

Building images

The script for building images will cd into the directory for each service. Here, it executes a more specific script called "./image_registry_<service name>.sh". The scripts are responsible for:

- Copying dependencies into a common folder
- Building the docker image
- Pushing the docker image to the registry where Kubernetes will access it

Docker build configurations are handled by a file called `Dockerfile` within the same directory as the script.

Pod deployment

The script for deploying the app will, similarly, call a suite of files for respective services. Each deployment will refer to images hosted in the Docker repository. Information for each pod is defined through "yaml" documents. The primary dependency is the image defined under "containers." It is also important to note the "nodePort" defined for each pod, as it defines the port for the pod/service.

Appendix II: Alternate/ other initial versions of the design

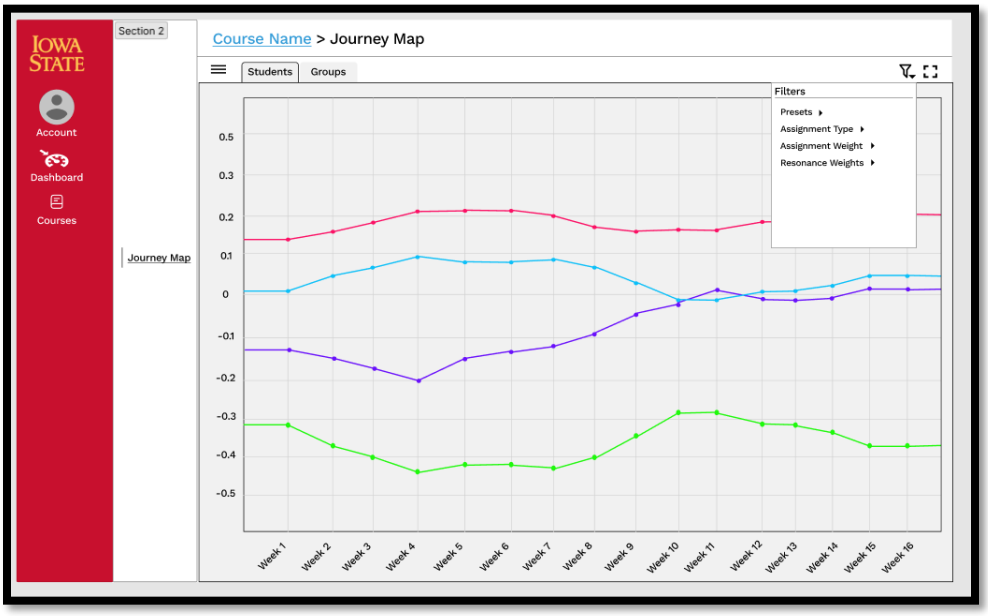

Figure 9: Mark II Canvas integration, class list options, and journey map.

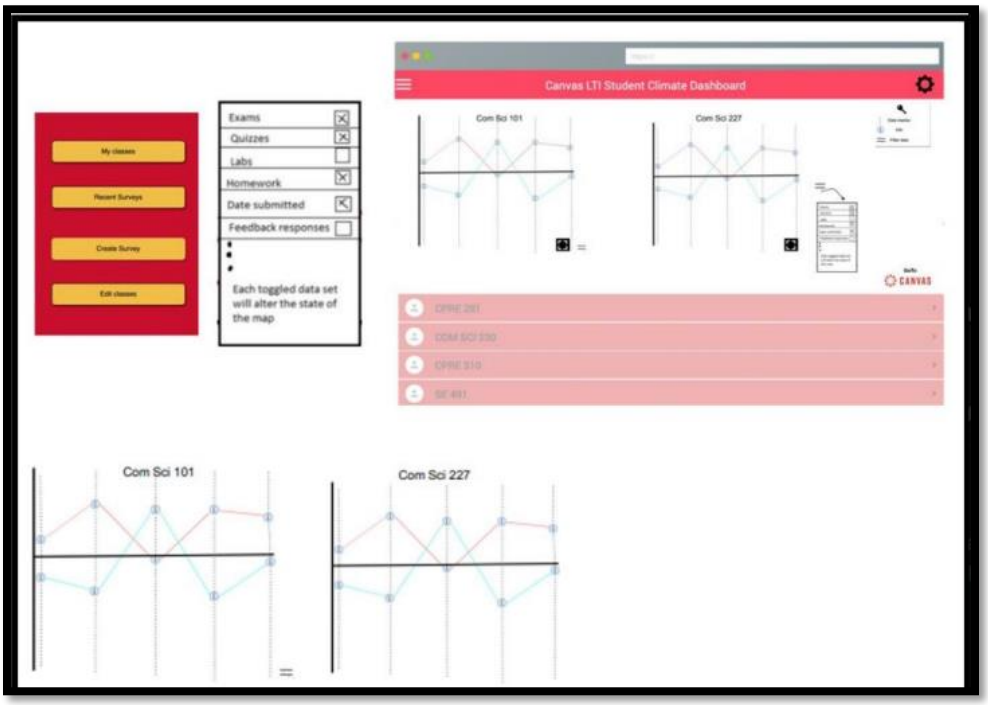

Figure 10: Mark I Graph layouts and Canvas integration.

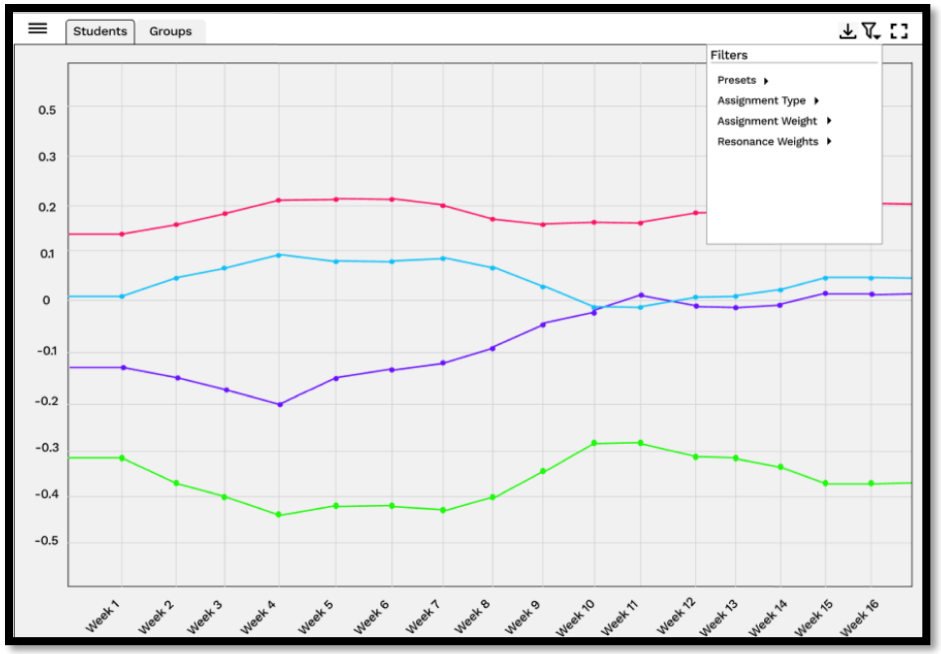

Figure 11: Figma mockup of a full screen journey map with tabs, week markers on the X-axis, and selectable functions.

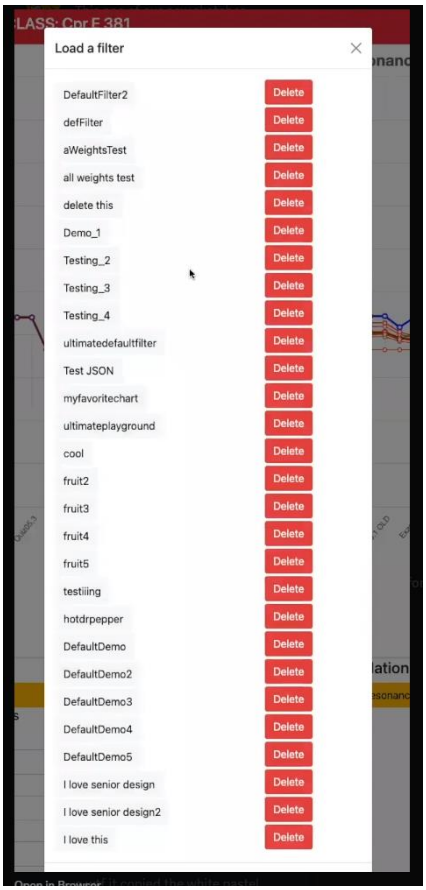

Figure 12: Available filter list for specifying resonance's component weights.

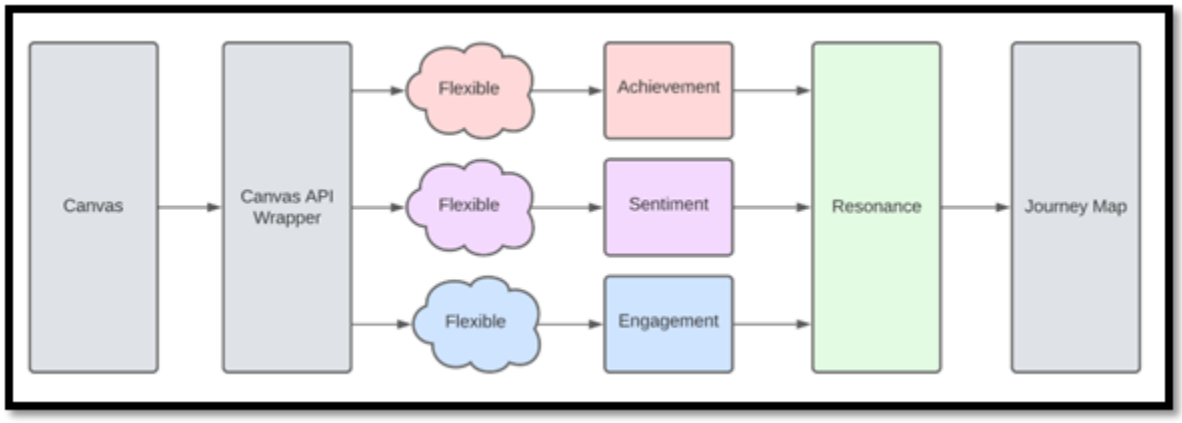

Figure 13: Proposed Mark II implementation that supports flexible definitions of Resonance using available data from Canvas.

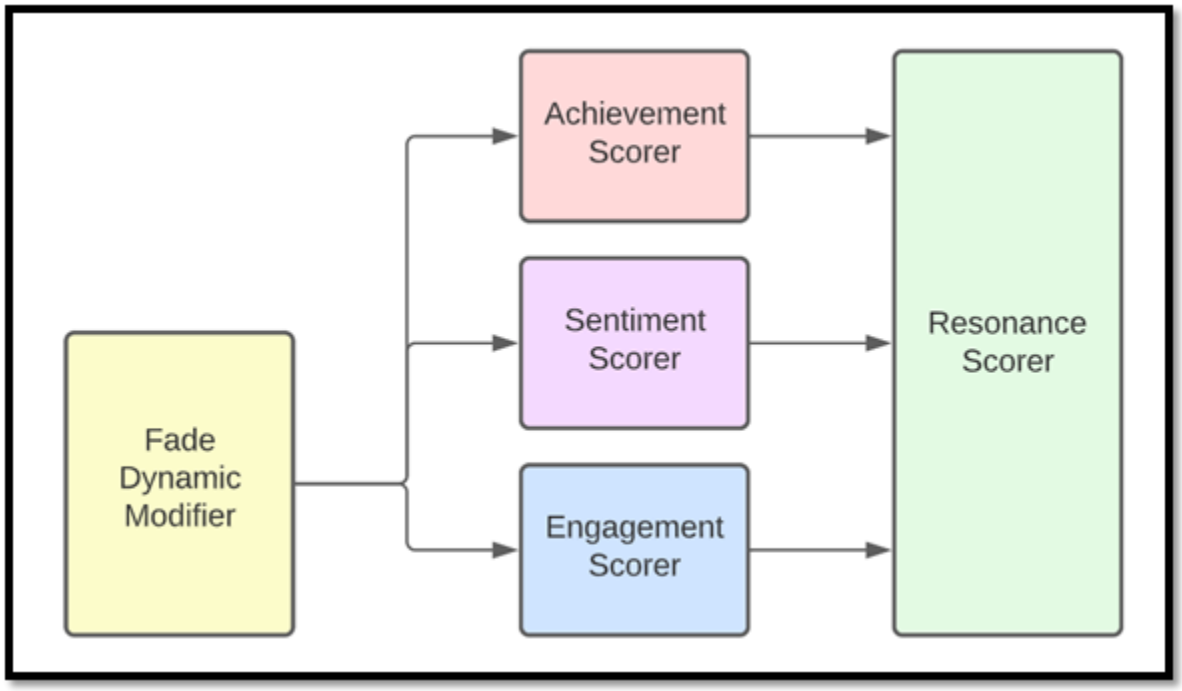

Figure 14: Proposed Fade Dynamic modifier being applied to scores used by resonance.

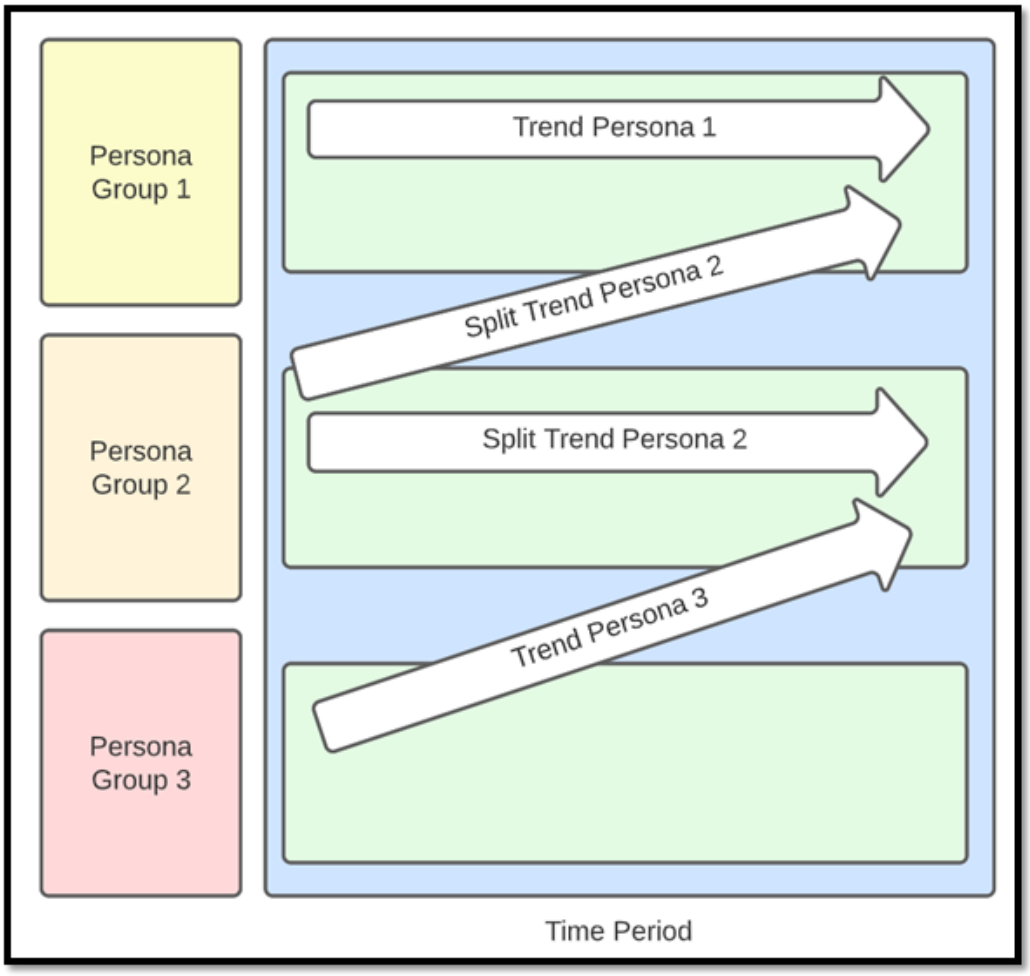

Figure 15: Proposed implementation of feature that allows for the visualization of trends of students between persona groups.

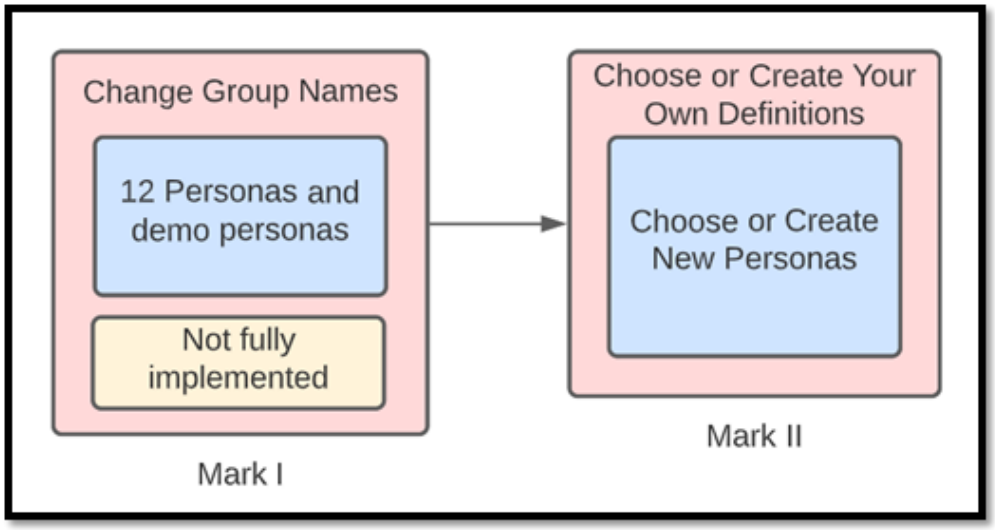

Figure 16: Mark I vs proposed Mark II Persona implementation.