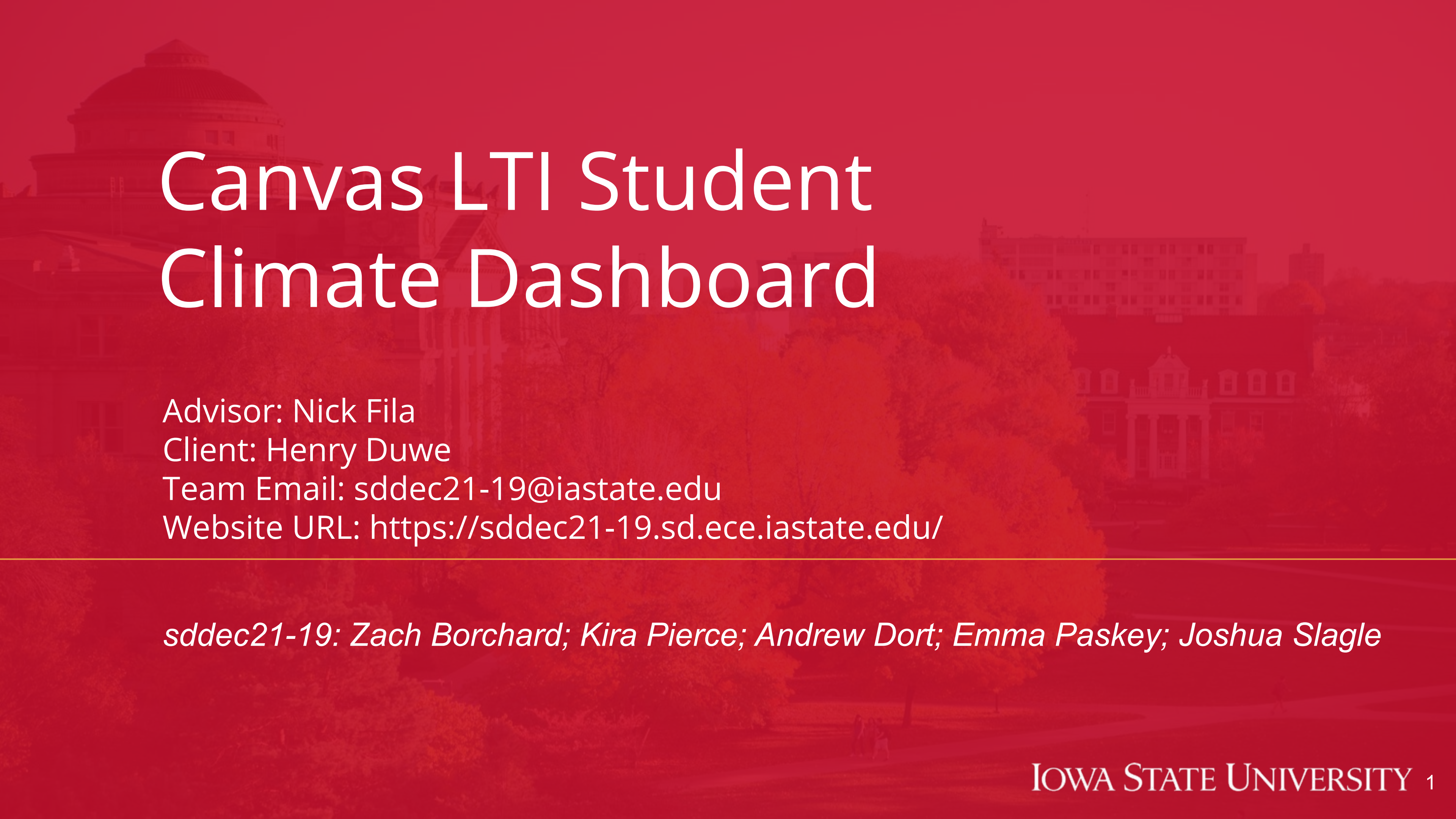


Canvas LTI Student Climate Dashboard

Advisor: Nick Fila
Client: Henry Duwe
Team Email: sddec21-19@iastate.edu
Website URL: https://sddec21-19.sd.ece.iastate.edu/

sddec21-19: Zach Borchard; Kira Pierce; Andrew Dort; Emma Paskey; Joshua Slagle

Presentation Agenda

- Project overview
- Project Implementation
- Recorded Demo
- Post-Project Thoughts
- Questions

Project Overview

Problem Statement/Solution

Problem statement:

- Instructors currently create journey maps to chart student resonance with the goal of understanding and identifying shared experiences among students within a course.
- Gathering data and building a graph is an intensive, subjective, and time consuming process.

Solution:

- Provide instructors with a software tool to automate the most time-consuming aspects of the journey-mapping process (data gathering & visualization).
- Users are provided with an interactive charting tool, and have the ability to aggregate student feedback and statistics from Canvas.

Requirements / Standards

Functional Requirements:

- The system should be able to create Journey Map from data.
- The system should automatically categorize students into groups.
- Professor should be able to view class Journey Map.
- Professors should be able to view journey maps with only a specific set of variables taken into account.

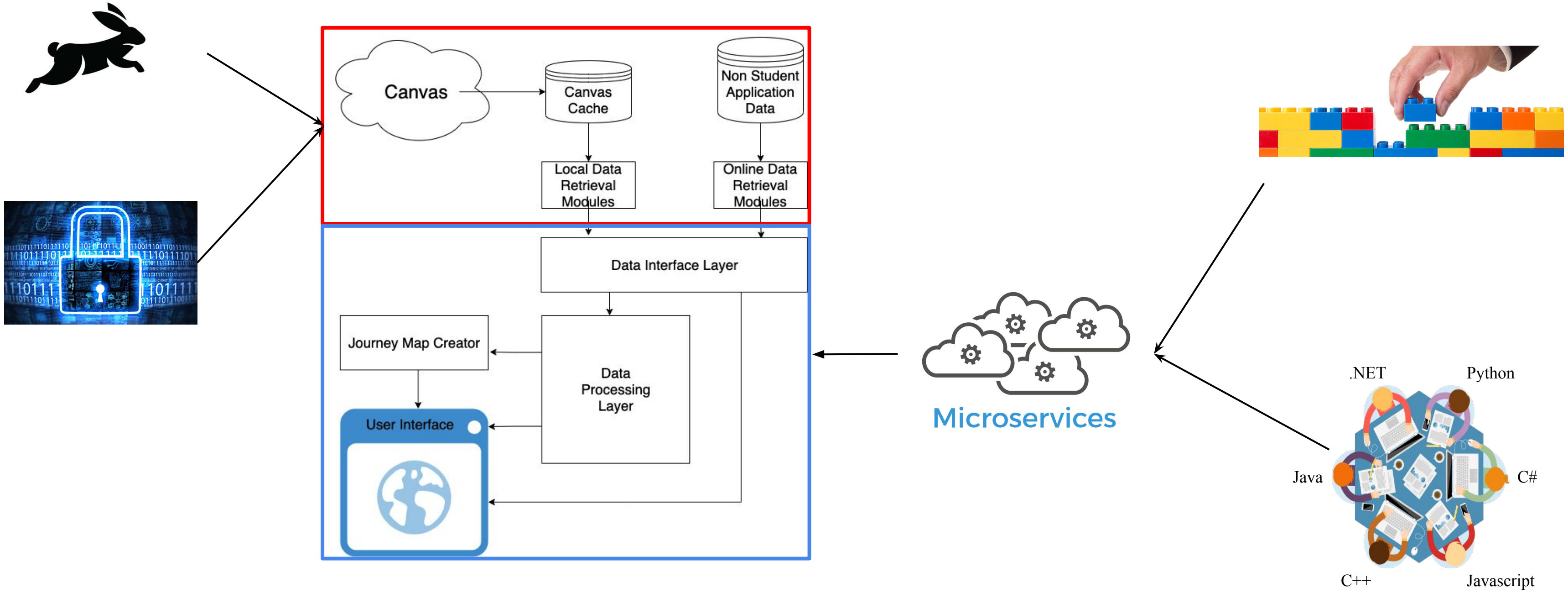
Non-Functional Requirements:

- Data integration should be modular for future extensions.
- Student data should not be accessible by other students.
- The system should be easily extensible.

Engineering Standards

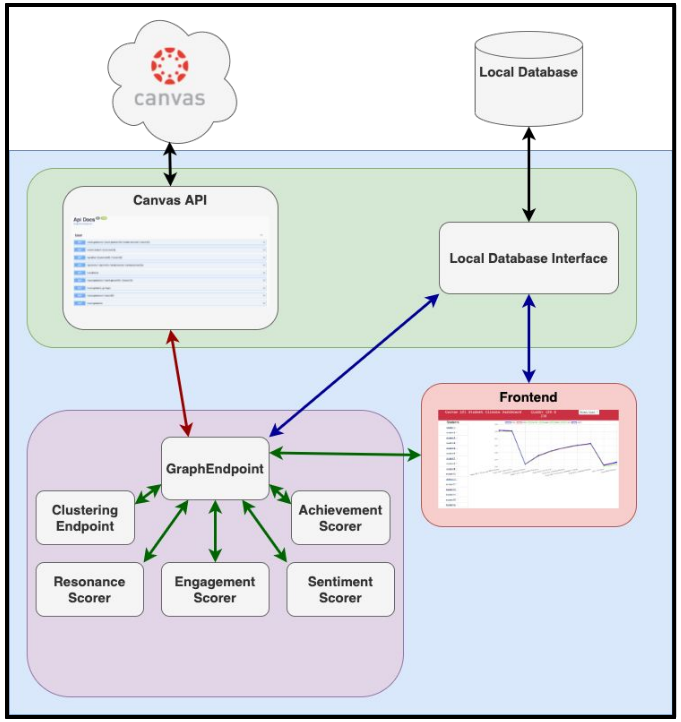
- Agile
- Acceptance / Integration testing
- Docker Containerization
- SOLID Principles
- Kubernetes

Technical Constraints/Considerations



Project Implementation

Overall Application



Application Components

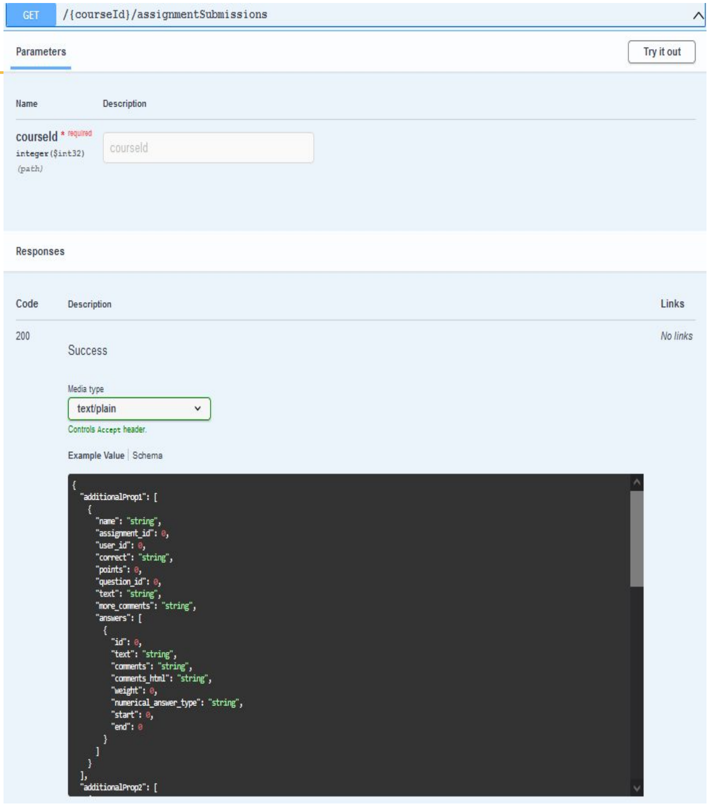
- Canvas API Wrapper (Green - Left)
- Data Analysis Pipeline (Purple)
- Data Storage (Green - Right)
- Frontend UI (Red)

Implementation

- Microservice-Based Architecture
- Containerized Modules
- Deployed in Kubernetes Cluster on ETG VM

Canvas API / Wrapper

- Used to interact with the Canvas API
- Written in C# and uses .NET Core framework
- Gets custom aggregate objects for the Data Analysis pipeline
- Caching to greatly decrease times of requests for application
- Completely documented on swagger
- Does not store data into database for security purposes
- Dockerized into its own container that is hosted on ISU's resources
- Can be spun up without needing the other services to make API calls


GET /{courseId}/assignmentSubmissions

Parameters

Try it out

Name	Description
courseId * required integer($int32) (path)	courseId

Responses

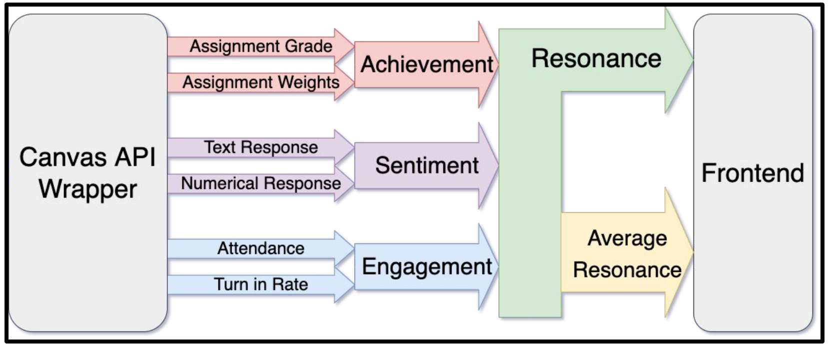
Code	Description	Links
200	Success Media type: text/plain Controls Accept header.	No links

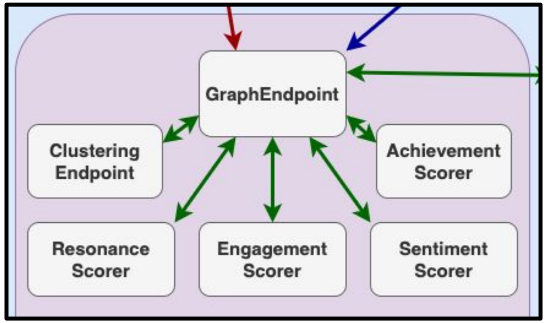
Example Value | Schema

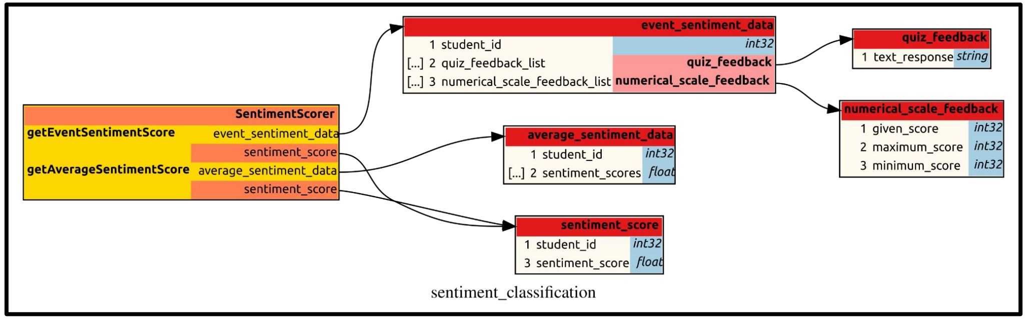
```
{
  "additionalProp1": [
    {
      "name": "string",
      "assignment_id": 0,
      "user_id": 0,
      "correct": "string",
      "points": 0,
      "question_id": 0,
      "text": "string",
      "more_comments": "string",
      "answers": [
        {
          "id": 0,
          "text": "string",
          "comments": "string",
          "comments_html": "string",
          "weight": 0,
          "numerical_answer_type": "string",
          "start": 0,
          "end": 0
        }
      ]
    }
  ],
  "additionalProp2": [
```

Data Analysis Pipeline (DAP)

- Converts student data to resonance data
- Six microservices written in Python
- Each microservice is a GRPC servicer
- Communication is done through Protobufs
- Main entrypoint is through GraphEndpoint







Backend Database

- MySQL database running on the VM
- Storage for non-sensitive information such as groups and filters
- Python Flask application as the REST API for GET/POST requests from the frontend
- Persistent deployment on a Docker container

Frontend

- Node JS - Express framework mixed with asynchronous requests
    - Utilizes connection to database and session storage
- Responsive layout, Bootstrap 5, single-page application style
- Okta authentication for session security
- Journey map implemented through Chart.js library
- Toggleable view of each student and group’s resonance data
- Interface for modifying resonance weight by assignment type
- Clickable expansion of data points for more detailed student information

Project Demo

Video

(see video in website)

Post Project Thoughts

Major Design Changes

- User no longer needs to have docker installed on their local machine to run application.
- No longer need to predict student resonance using trend lines - data analysis was deemed enough.
- The system cannot update real time because of the cache-ing used to speed up subsequent updates. Instead a 'force refresh' button has been added to the UI which forces the cached data to update.

Lessons Learned

- How to interact with external APIs
- How to interact between microservices in a medium-sized project
- Containerization makes application deployment and dependency checking easy
- Usefulness of modeling during planning stages
- Spending time on a good design pays huge dividends
- Setting arbitrary internal milestones helps keep project on track
- Software project constantly change throughout their lifecycle

Future Work

- Hand-off to Client
- Add a larger testing infrastructure to the application for CI/CD
- Implement student resonance prediction tooling
- Expand the statistical analysis presented to the professor