<DROPTABLE>Final Report

Team DROPTABLE, Chenlei Fu, Han (Gary) Liu, Yihong Jian, Chenhui Zhang

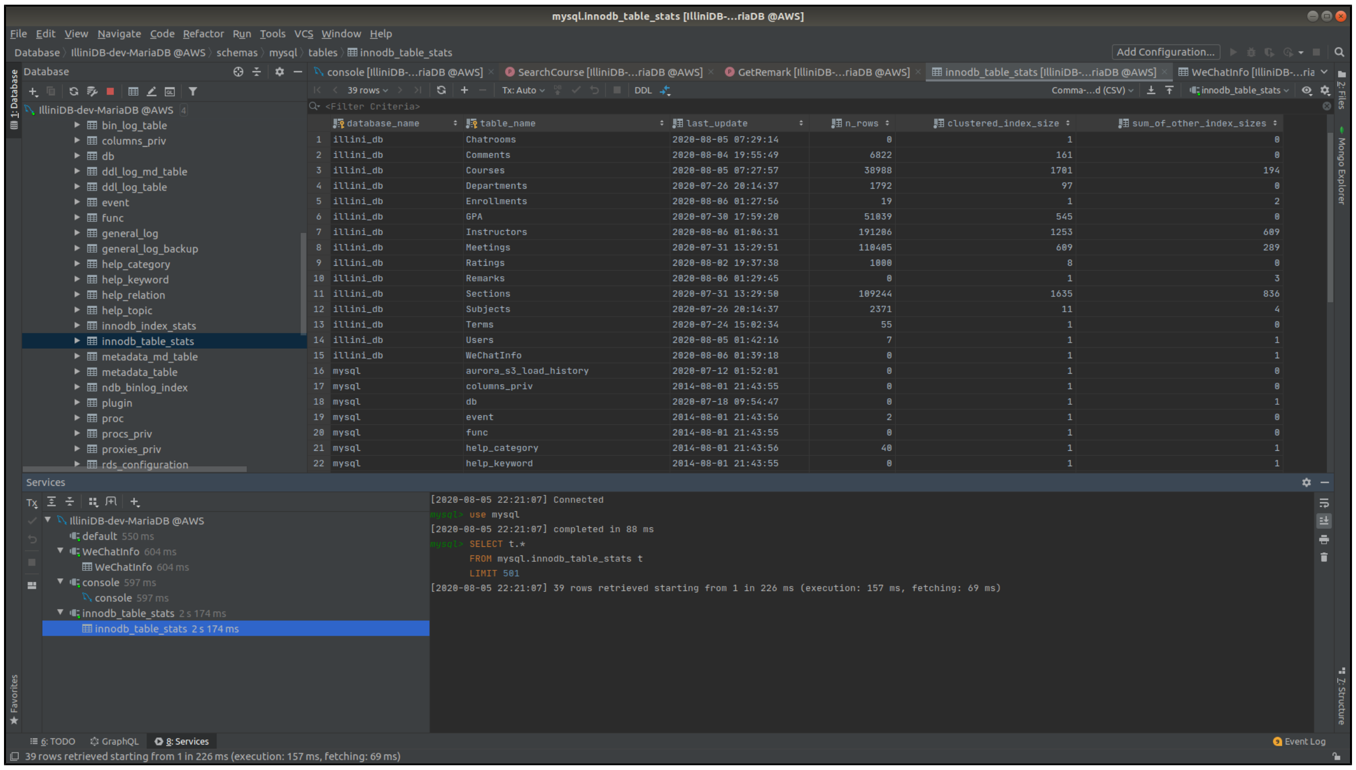
In this project, we created a versatile mobile app to smooth U of I students' academic experience by integrating functionalities including course scheduling, daily calendar management, course data visualization, course difficulty metrics, and instructor evaluation using data gathered by fetching official APIs and crawling third-party platforms. We also implemented a chatbot to extend our service into WeChat, the social media platform that is commenly used by some of our daily users every day.

Introduction

In this project, we created a versatile mobile app to smooth U of I students' academic experience by integrating functionalities including course scheduling, daily calendar management, course data visualization, course difficulty metrics, and instructor evaluation using data gathered by fetching official APIs and crawling third-party platforms. We also implemented a chatbot to extend our service into WeChat, the social media platform that is commenly used by some of our daily users every day.

Data

Our database (and Elasticsearch cluster) have three hundred thousand records of U of I class records and third-party information about GPA and professor ratings and comments. All of our course records are crawled from Course Explorer's official API. Our course difficulty metric and GPA trending visualization also heavily rely on these data. For GPA trending visualization, we used Professor Wade's highly organized GPA dataset to produce histograms that reflect the distribution of a certain course's GPA. Another important portion of our course difficulty metric comes from the students' review on a certain professor. However, since this is a new project and we don't have the time and resources to accumulate user data, we implemented a web crawler that fetches reviews about U of I professors. More details about how we fetch and manipulate data are elaborated later in "Technical Challenges".


mysql.innodb_table_stats [IlliniDB-...riaDB @AWS]
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Database 〉 IlliniDB-dev-MariaDB @AWS 〉 schemas 〉 mysql 〉 tables 〉 innodb_table_stats
console [IlliniDB-...riaDB @AWS]
SearchCourse [IlliniDB-...riaDB @AWS]
GetRemark [IlliniDB-...riaDB @AWS]
innodb_table_stats [IlliniDB-...riaDB @AWS]
WeChatInfo [IlliniDB-...ria
39 rows
Tx: Auto
DDL
Comma-...d (CSV)
Add Configuration...
Services
IlliniDB-dev-MariaDB @AWS
WeChatInfo 604 ms
console 597 ms
innodb_table_stats 2 s 174 ms

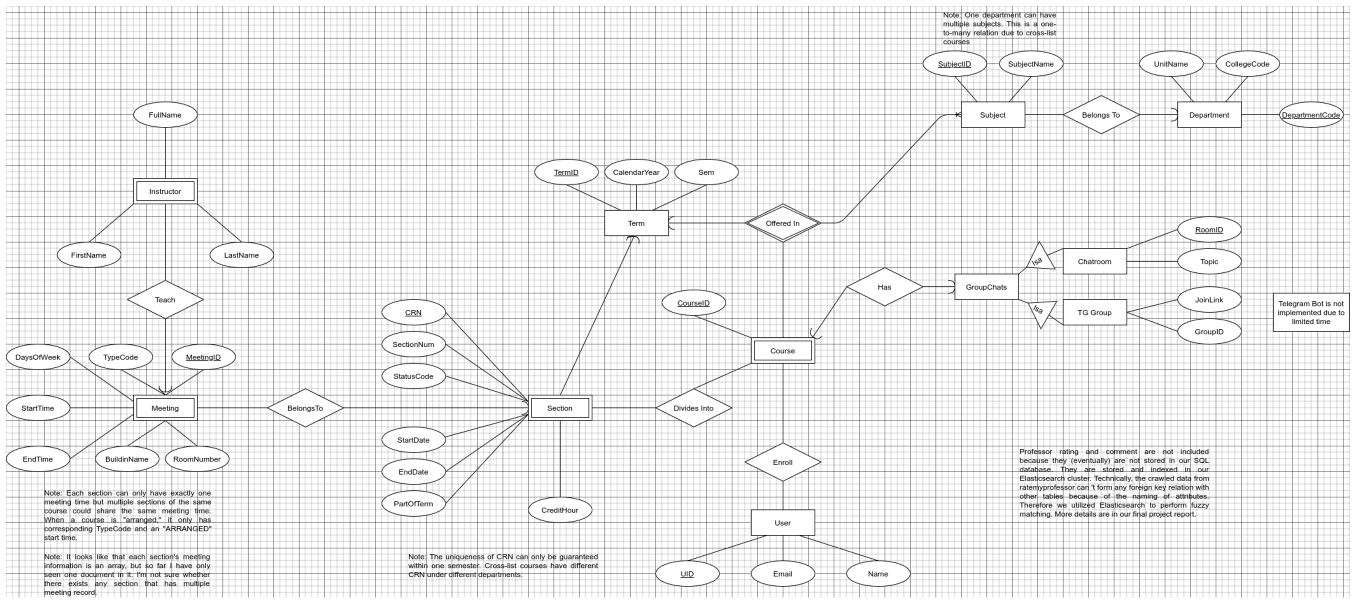
	database_name	table_name	last_update	n_rows	clustered_index_size	sum_of_other_index_sizes
1	illini_db	Chatrooms	2020-08-05 07:29:14	0	1	0
2	illini_db	Comments	2020-08-04 19:55:49	6822	161	0
3	illini_db	Courses	2020-08-05 07:27:57	38988	1701	194
4	illini_db	Departments	2020-07-26 20:14:37	1792	97	0
5	illini_db	Enrollments	2020-08-06 01:27:56	19	1	2
6	illini_db	GPA	2020-07-30 17:59:20	51039	545	0
7	illini_db	Instructors	2020-08-06 01:06:31	191206	1253	609
8	illini_db	Meetings	2020-07-31 13:29:51	110405	609	289
9	illini_db	Ratings	2020-08-02 19:37:38	1000	8	0
10	illini_db	Remarks	2020-08-06 01:29:45	0	1	3
11	illini_db	Sections	2020-07-31 13:29:50	109244	1635	836
12	illini_db	Subjects	2020-07-26 20:14:37	2371	11	4
13	illini_db	Terms	2020-07-24 15:02:34	55	1	0
14	illini_db	Users	2020-08-05 01:42:16	7	1	1
15	illini_db	WeChatInfo	2020-08-06 01:39:18	0	1	1
16	mysql	aurora_s3_load_history	2020-07-12 01:52:01	0	1	0
17	mysql	columns_priv	2014-08-01 21:43:55	0	1	0
18	mysql	db	2020-07-18 09:54:47	0	1	1
19	mysql	event	2014-08-01 21:43:56	2	1	0
20	mysql	func	2014-08-01 21:43:55	0	1	0
21	mysql	help_category	2014-08-01 21:43:56	40	1	1
22	mysql	help_keyword	2014-08-01 21:43:55	0	1	1

```
[2020-08-05 22:21:07] Connected
mysql> use mysql
[2020-08-05 22:21:07] completed in 88 ms
mysql> SELECT t.*
       FROM mysql.innodb_table_stats t
       LIMIT 501
[2020-08-05 22:21:07] 39 rows retrieved starting from 1 in 226 ms (execution: 157 ms, fetching: 69 ms)
```

39 rows retrieved starting from 1 in 226 ms (execution: 157 ms, fetching: 69 ms)

ER Diagram and Schema

ER Diagram



Note: One department can have multiple subjects. This is a one-to-many relation due to cross-list courses

Professor rating and comment are not included because they (eventually) are not stored in our SQL database. They are stored and indexed in our Elasticsearch cluster. Technically, the crawled data from ratemyprofessor can 't form any foreign key relation with other tables because of the naming of attributes. Therefore we utilized Elasticsearch to perform fuzzy matching. More details are in our final project report.

Note: Each section can only have exactly one meeting time but multiple sections of the same course could share the same meeting time. When a course is "arranged," it only has corresponding TypeCode and an "ARRANGED" start time.

Note: It looks like that each section's meeting information is an array, but so far I have only seen one document in it. I'm not sure whether there exists any section that has multiple meeting record.

Note: The uniqueness of CRN can only be guaranteed within one semester. Cross-list courses have different CRN under different departments.

Schema DDL

Schema DDL

```
CREATE TABLE Terms(
    TermID INT NOT NULL PRIMARY KEY,
    TermName VARCHAR(10) NOT NULL,
    TermDetailUrl VARCHAR(255) NOT NULL CHECK ( TermDetailUrl LIKE "%https%" ),
    CalendarYear INT NOT NULL,
    PublicIndicator BOOL NOT NULL,
    ArchiveIndicator BOOL NOT NULL,
    AttendingTerm BOOL,
    DefaultTerm BOOL,
    EnrollingTerm BOOL

);

CREATE TABLE Subjects(
    SubjectID VARCHAR(10) NOT NULL,
    SubjectName VARCHAR(100) NOT NULL,
    DepartmentCode VARCHAR(5) NOT NULL,
    TermID INT NOT NULL,
    PRIMARY KEY(SubjectID, TermID),
    FOREIGN KEY (TermID) REFERENCES Terms(TermID) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Departments(
    TermID INT,
    SubjectID VARCHAR(5),
    DepartmentName VARCHAR(255),
    CollegeCode VARCHAR(5),
    DepartmentCode INT,
    ContactName VARCHAR(255),
    ContactTitle VARCHAR(255),
    AddressLine1 VARCHAR(255),
    AddressLine2 VARCHAR(255),
    PhoneNumber VARCHAR(40),
    Url VARCHAR(255),
    DepartmentDescription VARCHAR(255),
    PRIMARY KEY (TermID, DepartmentCode, SubjectID),
    FOREIGN KEY (TermID) REFERENCES Terms(TermID)
);

CREATE TABLE Courses (
    SubjectID VARCHAR(10) NOT NULL,
    TermID INT NOT NULL,
    CourseID INT NOT NULL,
    CourseName VARCHAR(255) NOT NULL,
    CreditHours VARCHAR(20) NOT NULL,
    CourseDescription VARCHAR(255),
    CourseSectionInformation VARCHAR(255),
    SectionDegreeAttributes VARCHAR(255),
    SectionRegistrationNotes VARCHAR(255),
    ClassScheduleInformation VARCHAR(255),
    GenEdCategories VARCHAR(10),
    PRIMARY KEY (SubjectID, TermID, CourseID),
    FOREIGN KEY (TermID) REFERENCES Terms(TermID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE INDEX Courses_CourseID_index
        ON Courses (CourseID);

CREATE TABLE Sections(
    CRN INT NOT NULL,
    TermID INT NOT NULL,
    CourseID INT NOT NULL,
    SubjectID VARCHAR(10) NOT NULL,
    SectionNumber VARCHAR(10),
    Credits INT,
    StatusCode VARCHAR(10) NOT NULL,
    PartOfTerm VARCHAR(5) NOT NULL,
    EnrollmentStatus VARCHAR(255) NOT NULL,
    SectionText VARCHAR(255),
    SectionNotes VARCHAR(255),
    SectionCappArea VARCHAR(255),
    StartDate DATE,
    EndDate DATE,
    PRIMARY KEY (CRN, TermID),
    FOREIGN KEY (TermID) REFERENCES Terms(TermID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Meetings(
    CRN INT NOT NULL,
    TermID INT NOT NULL,
    MeetingID INT NOT NULL,
    TypeCode VARCHAR(5) NOT NULL,
    TypeName VARCHAR(255) NOT NULL,
    StartTime TIME,
    EndTime TIME,
    DaysOfWeek VARCHAR(5),
    BuildingName VARCHAR(255),
    RoomNumber INT,
    FOREIGN KEY (CRN, TermID) REFERENCES Sections(CRN, TermID) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Users(
    UUID VARCHAR(255) PRIMARY KEY,
    Email VARCHAR(255) UNIQUE NOT NULL,
    Name VARCHAR(255) NOT NULL,
    SaltedPassword VARCHAR(255) NOT NULL
);


CREATE TABLE Enrollments(
    UUID VARCHAR(255) NOT NULL,
    TermID INT NOT NULL,
    CRN INT NOT NULL,
    PRIMARY KEY (UUID, TermID, CRN),
    FOREIGN KEY (UUID) REFERENCES Users(UUID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (TermID) REFERENCES Terms(TermID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (CRN) REFERENCES Sections(CRN) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Instructors(
    CRN INT NOT NULL,
    TermID INT NOT NULL,
    MeetingID INT NOT NULL,
    FullName VARCHAR(255),
    LastName VARCHAR(255),
    FirstName VARCHAR(255),
    PRIMARY KEY (CRN, TermID, MeetingID, FullName),
    FOREIGN KEY (CRN) REFERENCES Sections(CRN) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (TermID) REFERENCES Terms(TermID) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE Remarks(
    RID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    UID VARCHAR(255) NOT NULL,
    CRN INT NOT NULL,
    TermID INT NOT NULL,
    Remark VARCHAR(255) NOT NULL,
    FOREIGN KEY (CRN) REFERENCES Enrollments(CRN),
    FOREIGN KEY (TermID) REFERENCES Enrollments(TermID),
    FOREIGN KEY (UID) REFERENCES Enrollments(UUID) ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE GPA(
    TermID INT NOT NULL,
    TermName VARCHAR(20) NOT NULL,
    CRN INT NOT NULL,
    SubjectID VARCHAR(10) NOT NULL,
    CourseID INT NOT NULL,
    CourseName VARCHAR(255) NOT NULL,
    TypeCode VARCHAR(5) NOT NULL,
    Ap INT,
    A INT,
    Am INT,
    Bp INT,
    B INT,
    Bm INT,
    Cp INT,
    C INT,
    Cm INT,
    Dp INT,
    D INT,
    Dm INT,
    F INT,
    W INT,
    PrimaryInstructor VARCHAR(255),
    PRIMARY KEY (TermID, CRN),
    FOREIGN KEY (TermID) REFERENCES Terms(TermID),
    FOREIGN KEY (CRN) REFERENCES Sections(CRN)
);

CREATE TABLE Ratings(
    TeacherID INT NOT NULL PRIMARY KEY,
    NumRatings INT NOT NULL,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    AvgRating REAL NOT NULL
);

CREATE TABLE Comments(
    CID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ID VARCHAR(63) NOT NULL,
    LegacyID INT NOT NULL ,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    Class VARCHAR(255) NOT NULL,
    Tags VARCHAR(255) NOT NULL,
    IsAttendanceMandatory BOOL,
    Clarity INT,
    DifficultyRating INT,
    HelpfulRating INT,
    WouldTakeAgain BOOL,
    IsForCredit BOOL,
    IsOnline BOOL,
    Grade VARCHAR(7),
    Comment TEXT,
    Date DATETIME
);
```

- `SubjectID` is the abbreviation of the subject. For example, `CS` and `ECE`.
- `CourseID` is the identifier of the course under a certain subject. For example, the `CourseID` for `CS411` is `411`.
- `TermID` is an unique identification generated by course explorer for each term.
- `Credits` is a string encoded by ourselves. We need to do this because some courses have different options for credit hours.
- `CourseSectionInformation` usually describes prerequisites.
- `SectionDegreeAttributes` usually describes the GenEd attribute that the course could fulfill.
- `GenEdCategories` is an encoded string of GenEd codes. For example, `QR:1QR1` means Quantitative Reasoning I.
- We use `CRN` and `TermID` as primary key because `CRN` is only unique within a certain term. For example, 2020 Fall's `CS 225` section `AL1 (35917)` has the same CRN as 2020 Spring's `LAW 610`.
- Since `CourseID` is a foreign key, it has to be in an index of the referenced relation where `CourseID` is the starting column.

Usage and Functionality

Usefulness

Our project is useful in the sense that it integrates much information into one portable and convenient app for students to deal with course planning, class scheduling, and workload estimating. Admittedly, some systems, such as MyIllini or Banner, have similar functionality in scheduling, but they are not able to integrate information from multiple, especially third-party, sources and they can't provide students with a way to connect with others. From the user experience's perspective, the existing course related applets are confusing and somewhat inefficient. We've encountered several instances where we had to login to MyIllini or other systems during class breaks, trying to figure out which classroom we shall proceed. It's often difficult to read through webpages that haven't been optimized for cell phones. Therefore, we believe it's beneficial to develop a system to streamline that process, emphasizing user experience and mobile compatibility.

As for course evaluation, Professor Wade's GPA visualization and Ratemyprofessor have existed for a very long time. Professor Wade's project provides students with a great visualization for GPA distribution, but that's not an accurate indication of the course's difficulty due to differences in class levels and types. In addition, it doesn't reflect the change of GPA distribution among different semesters. Ratemyprofessor can give students some qualitative reviews but they are not always reliable and are prone to abuse. Therefore, we proposed an evaluation algorithm that considers those bias and estimates schedule workload.

Functions

- Authenticate through FireAuth (Frontend + Third-party API)
- A calendar with user's schedule, class time count down, and user's remarks about each class (Frontend + Backend API + Basic Database Query)
- Add/Delete account by and change user name (Frontend + Backend API + Basic Database Query)
- Search for classes (Frontend + Backend API + Advanced SQL query with fuzzy search)
- Add/Delete classes to user schedule (Frontend + Backend API + + Basic Database Query)
- Add/Modify/Search/Delete notes on classes (Frontend + Backend API + + Basic Database Query)
- GPA visualization through a histogram (Frontend + Backend API + Advanced SQL Query)
- Professor ratings and comments (Frontend + Backend API + Elasticsearch Query)
- [Advanced Function] Checkout estimated workload and difficulty breakdown (Frontend + Backend API + Advanced SQL query)
- Link account, check schedule, automatic course study group chat finding (Chatbot frontend + Backend API + Basic Database Queries)

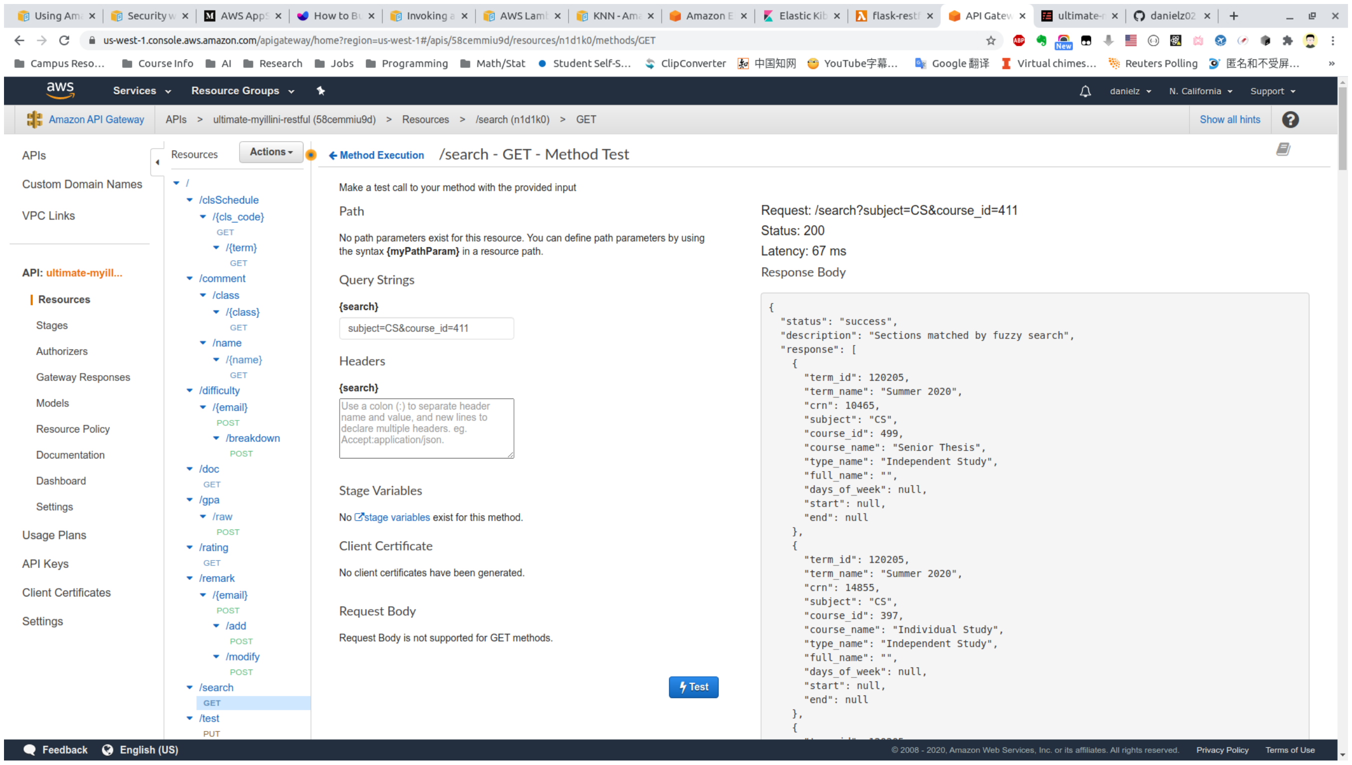
As an example, the add class function would start with user clicking add button in the app. Then, the app sends an POST request to our API Gateway, which inserts the record into our database.

Dataflow

The majority of the data, such as course sections and GPA histories, was crawled and added to our database before the deployment. The user only contributes to enrollment information, remarks, and chatroom information. Our dataflow work as follow:

1. Client sends requests (GET/POST/PUT) to our API Gateway.
2. API Gateway receives and validates the requests and distributes them to our elastic computing backend. (Invalid requests are rejected without invoking actual API services.) Instances of our API services are invoked, "woken up", or created based on the intensity of requests and then query our database.
3. Database performs the query and returns info to the API backend.
4. API services parse the query result, send back info to clients, and "fall asleep." (We are billed based on the number of invocations and the number of milliseconds our API is running.)
5. Client parses response and displays output.

Below are some details about our API service.


Amazon API Gateway > APIs > ultimate-myillini-restful (58cemmiu9d) > Resources > /search (n1d1k0) > GET

/search - GET - Method Test

Make a test call to your method with the provided input

Path

No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path.

Query Strings

{search}

subject=CS&course_id=411

Headers

{search}

Stage Variables

No stage variables exist for this method.

Client Certificate

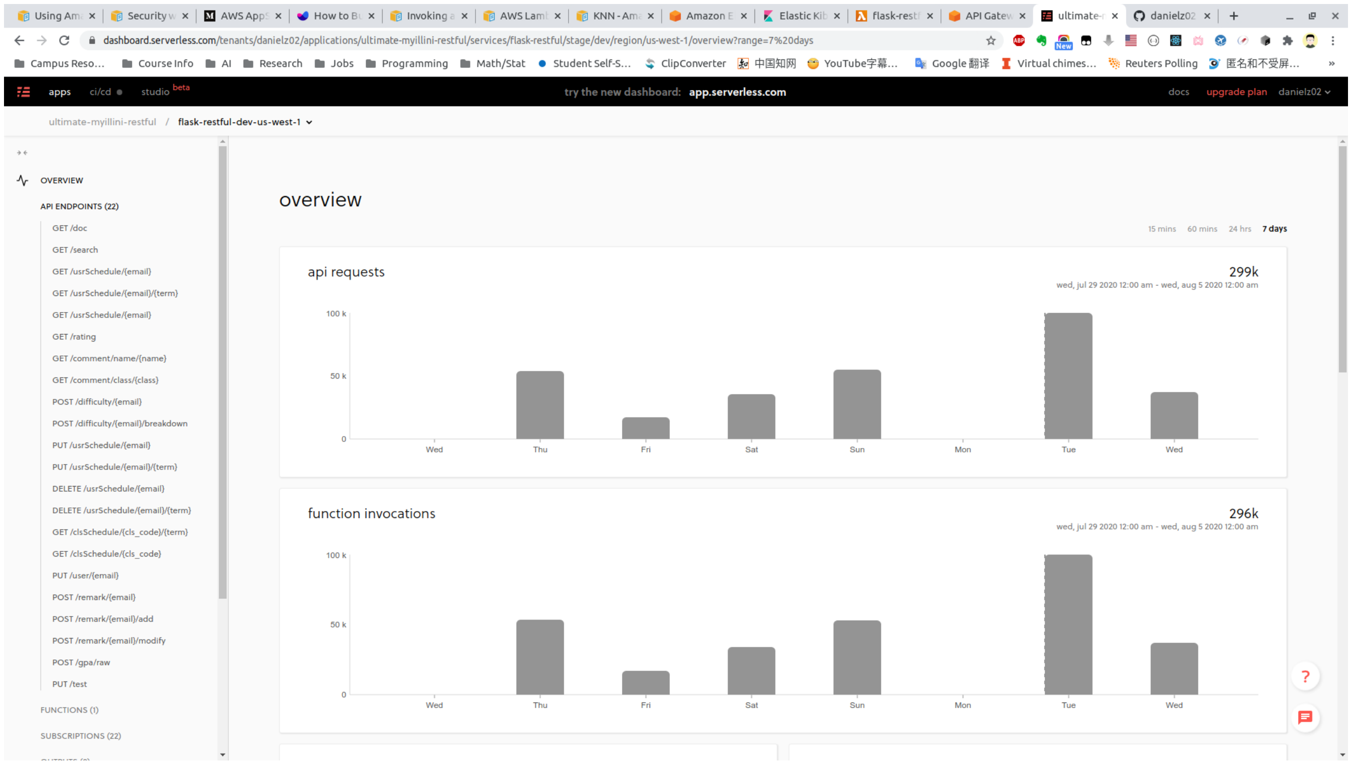
No client certificates have been generated.

Request Body

Request Body is not supported for GET methods.

Request: /search?subject=CS&course_id=411
Status: 200
Latency: 67 ms
Response Body

```
{
  "status": "success",
  "description": "Sections matched by fuzzy search",
  "response": [
    {
      "term_id": 120205,
      "term_name": "Summer 2020",
      "crn": 10465,
      "subject": "CS",
      "course_id": 499,
      "course_name": "Senior Thesis",
      "type_name": "Independent Study",
      "full_name": "",
      "days_of_week": null,
      "start": null,
      "end": null
    },
    {
      "term_id": 120205,
      "term_name": "Summer 2020",
      "crn": 14855,
      "subject": "CS",
      "course_id": 397,
      "course_name": "Individual Study",
      "type_name": "Independent Study",
      "full_name": "",
      "days_of_week": null,
      "start": null,
      "end": null
    },
    {
```


ultimate-myillini-restful / flask-restful-dev-us-west-1

overview

api requests 299k
wed, jul 29 2020 12:00 am - wed, aug 5 2020 12:00 am

function invocations 296k
wed, jul 29 2020 12:00 am - wed, aug 5 2020 12:00 am

API ENDPOINTS (22)

GET /doc
GET /search
GET /usrSchedule/{email}
GET /usrSchedule/{email}/{term}
GET /usrSchedule/{email}
GET /rating
GET /comment/name/{name}
GET /comment/class/{class}
POST /difficulty/{email}
POST /difficulty/{email}/breakdown
PUT /usrSchedule/{email}
PUT /usrSchedule/{email}/{term}
DELETE /usrSchedule/{email}
DELETE /usrSchedule/{email}/{term}
GET /clsSchedule/{cls_code}/{term}
GET /clsSchedule/{cls_code}
PUT /user/{email}
POST /remark/{email}
POST /remark/{email}/add
POST /remark/{email}/modify
POST /gpa/raw
PUT /test

FUNCTIONS (1)

SUBSCRIPTIONS (22)

Advanced Function

Evaluation metrics for course workload based on past GPA information, student review, and credit hours, etc. obtained through the course schedule crawler and web scraping. Show an estimation of overall workload throughout the semester. These data would go through a proposed algorithm:

$$\frac{\sum_i hrs_i}{cap}\sum_i hrs_i\frac{\mu}{5}(1+\lambda\lfloor\frac{clsNum_i}{100}\rfloor)(4.0-GPA_i)$$

In the formula, i represents the ithith class that user attends, λ represents the weight for class level, represents the sentiment weight derived from RateMyProfessor student ratings, and capcap represents the maximum hour of the semester. Then we performed a benchmark by feeding schedule of different workloads through the algorithm and determined the cutoff for numeric output.

Data Feeding and Serving Pipeline

- Ratemyprofessor crawler requires a lot of trial and error on different web crawling techniques. A lot of websites have anti-crawler techniques and we need to find a work-around before being thrown into jail. To avoid legal issues or accidentally attacking the webpage, we limited the scraping frequency of our web crawler and only scraped the information about UIUC professors. We also need to match the crawled data with our existing data. For example, our database doesn't have the full name of professors (their first names are abbreviated). This could be a problem when two professors have the same abbreviation (Prof. Bo Li of CS department and STAT's department chair Prof. Bo Li). More details about our solution are in the "Technical Challenge" section.
- Course crawler fetch all the data from the University's course explorer API and then parse the XML files into the format that can be inserted into our database. We have crawled some of the available data from the course explorer and implemented a mechanism to incrementally update the changes in the data in a timely and efficient manner. This advanced function involves implementing type definitions based on the XML schema to parse the data and the pipeline for converting XML schema into our customized relational schema. The incremental update function involves calling outside service directly from the database and propagating relevant changes back to the database through stored procedure and triggers.

Technical Stack

- Frontend: React Native and Chart.js
- API Service: Flask, PyMySQL, and the Request library.
- Course Crawler: Typescript and Node.js, Node-fetch, Fast-XML-Parser, MySQLJs, and Elasticsearch's Node.js driver
- Database: Amazon Aurora
- Search Engine: Elasticsearch
- DevOps: Github Action, the Serverless Framework, AWS VPC, AWS API Gateway, AWS Lambda Function, AWS RDS, AWS Elasticsearch Service, and AWS Internet Gateway

SQL Snippet

Course Fuzzy Search

Course Fuzzy Search

```
DELIMITER ;;

CREATE PROCEDURE SearchCourse(IN CRN INT, CourseName VARCHAR(255), SubjectID VARCHAR(5), ID INT, IsCurrentTerm
BOOL, NumRecords INT)
BEGIN
    DECLARE CurrentTerm INT;
    SELECT TermID INTO CurrentTerm FROM Terms WHERE AttendingTerm = TRUE;

    SET NumRecords = IFNULL(NumRecords, 50);
    SET IsCurrentTerm = IFNULL(IsCurrentTerm, TRUE);

    IF CRN IS NULL AND CourseName IS NULL AND SubjectID IS NULL AND ID IS NULL THEN
        SIGNAL SQLSTATE VALUE '23333'
        SET MESSAGE_TEXT = 'Don\'t even try to blow up the database! You must specify at least one search
field!';
    END IF ;

    SELECT c.TermID, t.TermName, s.CRN, c.SubjectID, c.CourseID, c.CourseName,
           m.TypeName, i.FullName, m.DaysOfWeek, m.StartTime, m.EndTime, s.StartDate
    FROM Terms t NATURAL JOIN Sections s NATURAL JOIN Meetings m NATURAL JOIN Courses c NATURAL JOIN
Instructors i
    WHERE s.CRN = IFNULL(CRN, s.CRN) AND c.CourseName LIKE CONCAT('%', IFNULL(CourseName, c.CourseName), '%')
AND
          c.SubjectID = IFNULL(SubjectID, c.SubjectID) AND c.CourseID = IFNULL(ID, c.CourseID) AND
          (NOT IsCurrentTerm OR c.TermID = CurrentTerm)
    ORDER BY TermID DESC
    LIMIT NumRecords;

END ;;

DELIMITER ;
```

Remarks

Course Fuzzy Search

```
DELIMITER ;;

CREATE PROCEDURE AddRemark(IN Email VARCHAR(255), CRN INT, TermID INT, Remark VARCHAR(255))
BEGIN
    DECLARE UID VARCHAR(255);

    IF Email IS NULL OR CRN IS NULL OR TermID IS NULL OR Remark IS NULL THEN
        SIGNAL SQLSTATE VALUE '23333'
        SET MESSAGE_TEXT = 'Missing field!';
    END IF ;

    SELECT UUID INTO UID FROM Users u WHERE u.Email = Email;

    INSERT INTO Remarks(UID, CRN, TermID, Remark) VALUE (UID, CRN, TermID, Remark);
    SELECT RID, UID, CRN, TermID, Remark FROM Remarks WHERE RID = LAST_INSERT_ID();
END ;;

DELIMITER ;


DELIMITER ;;

CREATE PROCEDURE ModifyRemark(IN TargetRID INT, userEmail VARCHAR(255), TargetCRN INT, TargetTermID INT,
NewRemark VARCHAR(255))
BEGIN
    DECLARE userID VARCHAR(255);

    IF userEmail IS NULL OR TargetCRN IS NULL OR TargetTermID IS NULL OR NewRemark IS NULL THEN
        SIGNAL SQLSTATE VALUE '23333'
        SET MESSAGE_TEXT = 'Missing field!';
    END IF ;

    SELECT UUID INTO userID FROM Users u WHERE u.Email = userEmail;

    UPDATE Remarks SET Remark = NewRemark WHERE RID = TargetRID AND UID = userID AND CRN = TargetCRN AND TermID
= TargetTermID;
    SELECT RID, UID, CRN, TermID, Remark FROM Remarks WHERE RID = TargetRID;
END ;;

DELIMITER ;


DELIMITER ;;

CREATE PROCEDURE GetRemark(IN UserEmail VARCHAR(255), TargetCRN INT, TargetTermID INT, SearchString VARCHAR
(255))
BEGIN
    DECLARE UserID VARCHAR(255) DEFAULT NULL;

    IF UserEmail IS NULL AND TargetCRN IS NULL AND TargetTermID IS NULL AND SearchString IS NULL THEN
        SIGNAL SQLSTATE VALUE '23333'
        SET MESSAGE_TEXT = 'You must specify at least one search field!';
    END IF ;

    SELECT UUID INTO UserID FROM Users u WHERE u.Email = UserEmail;

    SELECT RID, Remark, TermID, CRN, CourseName, SubjectID, CourseID
    FROM Remarks NATURAL JOIN Courses NATURAL JOIN Sections
    WHERE CRN = IFNULL(TargetCRN, CRN) AND TermID = IFNULL(TargetTermID, TermID) AND
          UID = IFNULL(UserID, UID) AND
          Remark LIKE IFNULL(CONCAT('%', SearchString, '%'), Remark);
END ;;

DELIMITER ;
```

GPA Raw Data for Visualization

Course Fuzzy Search

```
CREATE PROCEDURE GetDetailGPAInfo(IN Term INT, Subject VARCHAR(10), Course INT,
                                  Name VARCHAR(255), CRN INT, Instructor VARCHAR(255), Num INT)
BEGIN
    SET Num = IFNULL(Num, 50);

    IF Term IS NULL AND Subject IS NULL AND Course IS NULL AND
       CRN IS NULL AND Instructor IS NULL AND Name IS NULL THEN
        SIGNAL SQLSTATE '23333'
        SET MESSAGE_TEXT = 'You must specify at least one search field!!!';
    END IF ;

    SELECT *
    FROM GPA
    WHERE TermID = IFNULL(Term, TermID) AND SubjectID = IFNULL(Subject, SubjectID) AND
          CourseID = IFNULL(Course, CourseID) AND CourseName LIKE CONCAT('%', IFNULL(Name, CourseName), '%') AND
          PrimaryInstructor LIKE CONCAT('%', IFNULL(Instructor, PrimaryInstructor), '%')
    GROUP BY TermID, SubjectID, CourseID, CRN
    ORDER BY TermID DESC
    LIMIT Num;

END ;;

DELIMITER ;
```

Technical Challenges

One of the major challenges we faced was integrating the course data from the University's official API and the crawled data from Ratemyprofessor. The University's official course API responses to API calls with XML files instead of Json. This is especially problematic since XML's tree-like structure is more suitable for NoSQL database but we have to use SQL databse for this project. In order to fetch all the course data from Course Explorer's official API, we implemented a parser in `Typescript` with `Node-fetch` and `fast-xml-parser`. These two libraries grab the response of API call and then convert XML files into Json. However, the next challenge comes here. The University's official API doesn't provide us with an XML schema. Therefore, we have to decompose the XML file from root level all the way down to the leaf level and write "schema" (Typescript's type definition in our case) that corresponds with our database schema. This process took us nearly a week since we need to flatten the document from root level (term) to leaf level (class meeting and instructor) and we need to take account into all the edge cases through trial and error. (We ended up writing over three handred lines' type definition…) After a week's work, this course crawler is capable of fetching all the course data of any semester from the University's offical API and inserting them into our database through MySQL Javascript Driver's bulk insetion function, and it can be directly invoked in our database. With the help of this crawler, we managed to insert over 300 thoudsand official course records into our database without a hassle.

Another major difficulty we had was matching third-party data from web crawler and the University's official data. For example, the University's official API only provides us with instructor's last name and first name initial but Professor Wade's GPA dataset and the data from ratemyprofessor give us full name. For exmple, Alawini A. vs. Abdu Alawni. Although this might be an overkill, we deployed an elasticsearch cluster and indexed all of the rating data and comment data from Ratemyprofessor there. We created two seperate indices for ratings and comments. The full name attribute and the comment data are indexed with a whitespace tokenizer and an Edge n-gram token filter. They will chop the name and sentences into smaller tokens when creating the search index so that we can perform fuzzy search later very efficiently. When querying the documents, the user inputs will be evaluated againt the generated index and Elasticsearch will rank the results by generating similarity scores with BM25. Finally, we implemented a wrapper in our backend API to query our Elasticsearch cluster.

```
Example Index
{
  "settings": {
    "analysis": {
      "filter": {
        "autocomplete_filter": {
          "type": "edge_ngram",
          "min_gram": 1,
          "max_gram": 20
        }
      },
      "analyzer": {
        "autocomplete": {
          "type": "custom",
          "tokenizer": "whitespace",
          "filter": [
            "lowercase",
            "autocomplete_filter"
          ]
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "teacherID": {
        "type": "keyword"
      },
      "id": {
        "index": true,
        "type": "keyword"
      },
      "legacyId": {
        "index": true,
        "type": "keyword"
      },
      "firstName": {
        "index": true,
        "type": "text",
        "similarity": "BM25",
        "analyzer": "autocomplete",
        "search_analyzer": "autocomplete"
      },
      "lastName": {
        "index": true,
        "type": "text",
        "similarity": "BM25",
        "analyzer": "autocomplete",
        "search_analyzer": "autocomplete"
      },
      "fullName": {
        "index": true,
        "type": "text",
        "similarity": "BM25",
        "analyzer": "autocomplete",
        "search_analyzer": "autocomplete"
      },
      "class": {
        "index": true,
        "type": "keyword",
        "similarity": "BM25"
      },
      "tags": {
        "index": true,
        "type": "text",
        "similarity": "BM25",
        "analyzer": "autocomplete",
        "search_analyzer": "autocomplete"
      },
      "isAttendanceMandatory": {
        "type": "boolean"
      },
      "clarity": {
        "type": "integer"
      },
      "difficultyRating": {
        "type": "integer"
      },
      "helpfulRating": {
        "type": "integer"
      },
      "wouldTakeAgain": {
        "type": "boolean"
      },
      "isForCredit": {
        "type": "boolean"
      },
      "isOnline": {
        "type": "boolean"
      },
      "grade": {
        "type": "text"
      },
      "comment": {
        "index": true,
        "type": "text",
        "similarity": "BM25",
        "analyzer": "autocomplete",
        "search_analyzer": "autocomplete"
      },
      "date": {
        "type": "date"
      }
    }
  }
}
Example Query
{
    "size": 50,
    "query": {
        "multi_match": {
            "query": "Jose Vaz",
            "fields": ["fullName", "firstName", "lastName"],
            "type": "best_fields",
            "operator": "and",
            "tie_breaker": 0.0,
            "analyzer": "autocomplete",
            "fuzziness": "AUTO",
            "fuzzy_transpositions": true,
            "lenient": true,
            "prefix_length": 0,
            "max_expansions": 50,
            "auto_generate_synonyms_phrase_query": true,
            "zero_terms_query": "none"
        }
    },
    "sort": [
        {
            "date": { "order": "desc" }
        }
    ]
}
```

There are challenges for web crawler as well. First, after analyzing the json responses from the server, we found that Ratemyprofessor utilizes both a SQL database and a NoSQL database. The SQL database is mainly for storing the information about professors and universities. And the NoSQL one stores some dynamic information like comments and ratings. The two databases use different keys for the same professor, so that we have to join the two parts of information by ourselves. Secondly, the Ratemyprofessor uses GraphQL API. We had to analyze the GraphQL of Ratemyprofessor via trial and error in order to retrieve our intended data.

Updated Plan

Since summer semester was a fast term, our team members needed to relocate, and we had timezone issues, we weren't able to accomplish all tasks in the proposal. Specifically, we did not have enough time to implement the classmate finder. Even though finding classmates from database would require only a simple SQL query, developing chatbots that automatically form group chat was time consuming. However, we did implement a prototype of chatbot that can link accounts and search for enrollments. We're happy to work on it when we have more time in the future.

Division of labor

This project can't be completed without the effort of any group member. Our final division of labor was very similar to that of the purposed. Chenlei was responsible for the frontend mobile app. She created and tested the entire mobole frontend with React Native. We won't be able to show our functionalities without her work. Yihong did a lot of work on the backend API and the course difficulty metrics, which later became our advanced function. He created complex queries to retrieve course data and designed and benchmarked the course difficulty metrics. As a by-product, he also implemented a prototype of Numpy's broadcasting mechansim because Numpy is not suitable for deploying in FaaS instances. Han implemented a complexed crawler to fetch data from ratemyprofessor. Our advanced function won't be possible without these data. Chenhui implemented the course explorer crawler, worked on our API backend and the fuzzy search functions, and implemented the chatbot. He was also responsible for the DevOps (databse management, CI/CD pipeline, and the deployment of API service) of this project.

Open Source

- Frontend
- Chatbot
- API Backend
- Course Crawler
- SQL and Elasticsearch Query Collection