Project Sprint 1 (d1)

Campus Explorer Query Service

Universities have a wide variety of courses. This deliverable will focus on importing data about these courses into your project and enabling flexible querying over these data with a flexible domain-specific query language.

This deliverable can be completed in pairs or individually. It is worth 25% of your capstone grade. You will not have to hand anything in, though you will need to complete the open response assessment on the edX platform to receive your final deliverable grade. To provision your repository you must achieve 60% on d0; the repo can be provisioned using the SDMM portal. All work must take place in this repo (for d1, d2, and d3). If you choose a partner, you must do so before you start d1 and you must work with the same partner for d1, d2, and d3; to select a partner you must know their GitHub username and they must also have > 60% on d0. If you choose to work individually, you must continue to work individually for d1, d2, and d3. We will run MOSS on all submissions so please make sure your work is your own.

You are responsible for the software design and implementation. You cannot use any library package that is not already specified in `package.json`. Your implementation must be in TypeScript.

Dataset

This data has been obtained from UBC PAIR and has not been modified in any way. The data is provided as a zip file: inside of the zip you will find a file for each of the courses offered at UBC. Each of those file is a comma-separated values (CSV) file containing the information about each offering of the course.

The dataset zip file can be found here: sdmm.courses.1.0.zip

Checking the validity of the dataset

- A valid dataset has to be a valid zip file; this zip will contain many files under a folder called `courses/`.
- Valid courses will always be in CSV format.
- Each CSV file represents a course and can contain zero or more course sections.
- A valid dataset has to contain at least one valid course section that meets the requirements above.

Reading and Parsing the Dataset

You will need to parse valid input files into internal objects or other data structures. You are

not allowed to store the data in a database. You must also persist (cache) the model to disk for quicker access. Do not commit this cached file to version control, or AutoTest will fail in unpredictable ways.

There is a provided package called JSZip that you should use to process/unzip the data you are passed in your addDataset method (described below).

Query Engine

The goal of the deliverable is to build the backend to reply to query about the dataset. The query will be based on the EBNF described below.

EBNF

Note: this EBNF is not complete and will be extended in future deliverables

```
QUERY   ::= DATASET + ', ' + FILTER + '; ' + DISPLAY(+ '; ' + ORDER)? + '.'
DATASET ::= 'in ' + KIND + ' dataset ' + INPUT
FILTER  ::= 'find all entries' || 'find entries whose ' + (CRITERIA || (CRITERIA +
((' and ' || ' or ') + CRITERIA)*)
DISPLAY ::= 'show ' + KEY (+ MORE_KEYS)?
ORDER   ::= 'sort in ascending order by ' + KEY

CRITERIA   ::= M_CRITERIA || S_CRITERIA
M_CRITERIA ::= M_KEY + M_OP + NUMBER
S_CRITERIA ::= S_KEY + S_OP + STRING

NUMBER   ::= [0-9] + (.)? + [0-9]
STRING   ::= '"' + [^*"]* + '"' // any string without * or " in it, enclosed by
double quotation marks

RESERVED ::= KEYWORD || M_OP || S_OP || AGGREGATOR || KIND
KEYWORD  ::= 'In' || 'dataset' || 'find' || 'all' || 'show' || 'and' || 'or' ||
'sort' || 'by' || 'entries' || 'is' || 'the' || 'of' || 'whose'
M_OP     ::= 'is ' + ('not ' +)? ('greater than ' || 'less than ' || 'equal to ') +
NUMBER
S_OP     ::= (('is ' + ('not ' +)?) || (('includes ' || 'does not include ') ||
('begins' || 'does not begin' || 'ends' || 'does not end' + ' with '))) + STRING
KIND     ::= 'courses'

INPUT    ::= string of one or more characters. Cannot contain spaces, underscores or
equal to RESERVED strings

KEY   ::= M_KEY || S_KEY
MORE_KEYS   ::= ((', ' + KEY +)* ' and ' + KEY)
M_KEY ::= 'Average' || 'Pass' || 'Fail' || 'Audit'
S_KEY ::= 'Department' || 'ID' || 'Instructor' || 'Title' || 'UUID'
```

Syntactic Checking (Parsing)

Your query engine must test the validity of input queries against the grammar. It must then store the query in a structure (related objects, objects in an AST, or other data structure of your choice) such that you can perform the query as indicated by the semantics below. The hierarchy of that structure should likely match that of the incoming CSV.

Error responses for failed parsing are provided below in the specification for the `performQuery` method

Semantic Checking

Semantic checks are typically performed on the existing (validated) AST. Type checking is an example of a semantic check. In this project you must perform the following semantic check:

`'ORDER': key` where key (a string) is the column name to sort on; the key must be in the COLUMNS array or the query is invalid

Valid keys

In order to maintain readability of the queries, you are not allowed to use the key contained in data. Rather, you will have to build a vocabulary that will translate the keys in the query to the keys that you will use to query the data.

Valid keys are composed by two parts, separated by an underscore: `<id>_<key>`

- `<id>` is provided by the user and will be received through the `addDataset()` method, check the API spec to better understand how it should work.
- `<key>` is the key that represents a given piece of information. For this deliverable you will parse the following keys: `dept`, `id`, `instructor`, `title`, `pass`, `fail`, `audit`, `uuid`, and `avg`.

For instance, if the `id` sent by the user is `courses`, then the queries you will run will be using the following keys:

- courses_dept: `string`; The department that offered the course.
- courses_id: `string`; The course number (will be treated as a string (e.g., 499b)).
- courses_avg: `number`; The average of the course offering.
- courses_instructor: `string`; The instructor teaching the course offering.
- courses_title: `string`; The name of the course.
- courses_pass: `number`; The number of students that passed the course offering.
- courses_fail: `number`; The number of students that failed the course offering.
- courses_audit: `number`; The number of students that audited the course offering.
- courses_uuid: `string`; The unique id of a course offering.

Note: these keys are different than the ones present in the raw data. Since you are not allowed to modify the data, you will have to come up with a way to translate them.

Query example

Two simple queries are shown below (these correspond to `InsightResponse.body`):

Simple query

```
In courses dataset courses, find entries whose Average is greater than 97; show
Department and Average; sort in ascending order by Average.
```

The result for this would look like:

```
{ result:
   [ { courses_dept: 'epse', courses_avg: 97.09 },
     { courses_dept: 'math', courses_avg: 97.09 },
     { courses_dept: 'math', courses_avg: 97.09 },
     { courses_dept: 'epse', courses_avg: 97.09 },
     { courses_dept: 'math', courses_avg: 97.25 },
     { courses_dept: 'math', courses_avg: 97.25 },
     { courses_dept: 'epse', courses_avg: 97.29 },
     { courses_dept: 'epse', courses_avg: 97.29 },
     { courses_dept: 'nurs', courses_avg: 97.33 },
     { courses_dept: 'nurs', courses_avg: 97.33 },
     { courses_dept: 'epse', courses_avg: 97.41 },
     { courses_dept: 'epse', courses_avg: 97.41 },
     { courses_dept: 'cnps', courses_avg: 97.47 },
     { courses_dept: 'cnps', courses_avg: 97.47 },
     { courses_dept: 'math', courses_avg: 97.48 },
     { courses_dept: 'math', courses_avg: 97.48 },
     { courses_dept: 'educ', courses_avg: 97.5 },
     { courses_dept: 'nurs', courses_avg: 97.53 },
     { courses_dept: 'nurs', courses_avg: 97.53 },
     { courses_dept: 'epse', courses_avg: 97.67 },
     { courses_dept: 'epse', courses_avg: 97.69 },
     { courses_dept: 'epse', courses_avg: 97.78 },
     { courses_dept: 'crwr', courses_avg: 98 },
     { courses_dept: 'crwr', courses_avg: 98 },
     { courses_dept: 'epse', courses_avg: 98.08 },
     { courses_dept: 'nurs', courses_avg: 98.21 },
     { courses_dept: 'nurs', courses_avg: 98.21 },
     { courses_dept: 'epse', courses_avg: 98.36 },
     { courses_dept: 'epse', courses_avg: 98.45 },
     { courses_dept: 'epse', courses_avg: 98.45 },
     { courses_dept: 'nurs', courses_avg: 98.5 },
     { courses_dept: 'nurs', courses_avg: 98.5 },
     { courses_dept: 'epse', courses_avg: 98.58 },
     { courses_dept: 'nurs', courses_avg: 98.58 },
     { courses_dept: 'nurs', courses_avg: 98.58 },
     { courses_dept: 'epse', courses_avg: 98.58 },
     { courses_dept: 'epse', courses_avg: 98.7 },
     { courses_dept: 'nurs', courses_avg: 98.71 },
     { courses_dept: 'nurs', courses_avg: 98.71 },
     { courses_dept: 'eece', courses_avg: 98.75 },
     { courses_dept: 'eece', courses_avg: 98.75 },
     { courses_dept: 'epse', courses_avg: 98.76 },
     { courses_dept: 'epse', courses_avg: 98.76 },
     { courses_dept: 'epse', courses_avg: 98.8 },
     { courses_dept: 'spph', courses_avg: 98.98 },
     { courses_dept: 'spph', courses_avg: 98.98 },
     { courses_dept: 'cnps', courses_avg: 99.19 },
     { courses_dept: 'math', courses_avg: 99.78 },
     { courses_dept: 'math', courses_avg: 99.78 } ] }
```

Complex query

In courses dataset courses, find entries whose Average is greater than 90 and Department is \"adhe\" or Average is equal to 95; show Department, ID and Average; sort in ascending order by Average.

The result of this query would be:


```
{ result:
   [ { courses_dept: 'adhe', courses_id: '329', courses_avg: 90.02 },
     { courses_dept: 'adhe', courses_id: '412', courses_avg: 90.16 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 90.17 },
     { courses_dept: 'adhe', courses_id: '412', courses_avg: 90.18 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 90.5 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 90.72 },
     { courses_dept: 'adhe', courses_id: '329', courses_avg: 90.82 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 90.85 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 91.29 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 91.33 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 91.33 },
     { courses_dept: 'adhe', courses_id: '330', courses_avg: 91.48 },
     { courses_dept: 'adhe', courses_id: '329', courses_avg: 92.54 },
     { courses_dept: 'adhe', courses_id: '329', courses_avg: 93.33 },
     { courses_dept: 'rhsc', courses_id: '501', courses_avg: 95 },
     { courses_dept: 'bmeg', courses_id: '597', courses_avg: 95 },
     { courses_dept: 'bmeg', courses_id: '597', courses_avg: 95 },
     { courses_dept: 'cnps', courses_id: '535', courses_avg: 95 },
     { courses_dept: 'cnps', courses_id: '535', courses_avg: 95 },
     { courses_dept: 'cpsc', courses_id: '589', courses_avg: 95 },
     { courses_dept: 'cpsc', courses_id: '589', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'crwr', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'sowk', courses_id: '570', courses_avg: 95 },
     { courses_dept: 'econ', courses_id: '516', courses_avg: 95 },
     { courses_dept: 'edcp', courses_id: '473', courses_avg: 95 },
     { courses_dept: 'edcp', courses_id: '473', courses_avg: 95 },
     { courses_dept: 'epse', courses_id: '606', courses_avg: 95 },
     { courses_dept: 'epse', courses_id: '682', courses_avg: 95 },
     { courses_dept: 'epse', courses_id: '682', courses_avg: 95 },
     { courses_dept: 'kin', courses_id: '499', courses_avg: 95 },
     { courses_dept: 'kin', courses_id: '500', courses_avg: 95 },
     { courses_dept: 'kin', courses_id: '500', courses_avg: 95 },
     { courses_dept: 'math', courses_id: '532', courses_avg: 95 },
     { courses_dept: 'math', courses_id: '532', courses_avg: 95 },
     { courses_dept: 'mtrl', courses_id: '564', courses_avg: 95 },
     { courses_dept: 'mtrl', courses_id: '564', courses_avg: 95 },
     { courses_dept: 'mtrl', courses_id: '599', courses_avg: 95 },
     { courses_dept: 'musc', courses_id: '553', courses_avg: 95 },
     { courses_dept: 'musc', courses_id: '553', courses_avg: 95 },
     { courses_dept: 'musc', courses_id: '553', courses_avg: 95 },
     { courses_dept: 'musc', courses_id: '553', courses_avg: 95 },
     { courses_dept: 'musc', courses_id: '553', courses_avg: 95 },
     { courses_dept: 'musc', courses_id: '553', courses_avg: 95 },
     { courses_dept: 'nurs', courses_id: '424', courses_avg: 95 },
```

```
    { courses_dept: 'nurs', courses_id: '424', courses_avg: 95 },
    { courses_dept: 'obst', courses_id: '549', courses_avg: 95 },
    { courses_dept: 'psyc', courses_id: '501', courses_avg: 95 },
    { courses_dept: 'psyc', courses_id: '501', courses_avg: 95 },
    { courses_dept: 'econ', courses_id: '516', courses_avg: 95 },
    { courses_dept: 'adhe', courses_id: '329', courses_avg: 96.11 } ] }
```

API

Overview:

The API is comprised of three interfaces. You must not change the interface specifications.

- `InsightResponse` is the interface for the objects your methods will fulfill with.
- `IInsightFacade` is the front end (wrapper) for the query engine. In practice, it defines the endpoints for the deliverable. It provides several methods:
- `addDataset(id: string, content: string, kind: InsightDatasetKind): Promise<InsightResponse>` adds a dataset to the internal model, providing the id of the dataset, the string of the content of the dataset, and the kind of the dataset. For this deliverable the dataset kind will be *courses*.
- `removeDataset(id: string): Promise<InsightResponse>` removes a dataset from the internal model, given the id.
- `performQuery(query: any): Promise<InsightResponse>` performs a query on the dataset. It first should parse and validate the input query, then perform semantic checks on the query, and finally evaluate the query if it is valid.
- `listDatasets(): Promise<InsightResponse>` returns InsightResponse containing the list of added datasets. This list contains the id, kind, and number of rows of each added dataset.

To implement the API you will likely have to create your own additional methods and classes.

The high-level API you must support is shown below; these are methods we will provide you in your bootstrap project in `src/controller/` (there is also code in `src/rest/` but you do not need to worry about this yet).

```
/*
 * This is the primary high-level API for the project. In this folder there should
be:
 * A class called InsightFacade, this should be in a file called InsightFacade.ts.
 * You should not change this interface at all or the test suite will not work.
 */
export interface InsightResponse {
    code: number;
    body: InsightResponseSuccessBody | InsightResponseErrorBody; // The actual
response
}

export interface InsightResponseSuccessBody {
    result: any[] | string;
}
```

```
export interface InsightResponseErrorBody {
    error: string;
}

export enum InsightDatasetKind {
    Courses = "courses",
    Rooms = "rooms"
}

export interface InsightDataset {
    id: string;
    kind: InsightDatasetKind;
    numRows: number;
}

export interface IInsightFacade {

    /**
     * Add a dataset to UBCInsight.
     *
     * @param id  The id of the dataset being added.
     * @param content  The base64 content of the dataset. This content should be in
the form of a serialized zip file.
     * @param kind  The kind of the dataset
     *
     * @return Promise <InsightResponse>
     *
     * The promise should return an InsightResponse for both fulfill and reject.
     *
     * Fulfill should be for 2XX codes and reject for everything else.
     *
     * After receiving the dataset, it should be processed into a data structure of
     * your design. The processed data structure should be persisted to disk; your
     * system should be able to load this persisted value into memory for answering
     * queries.
     *
     * Ultimately, a dataset must be added or loaded from disk before queries can
     * be successfully answered.
     *
     * Response codes:
     *
     * 204: the operation was successful
     * 400: the operation failed. The body should contain {"error": "my text"}
     * to explain what went wrong. This should also be used if the provided dataset
     * is invalid or if it was added more than once with the same id.
     *
     */
    addDataset(id: string, content: string, kind: InsightDatasetKind):
Promise<InsightResponse>;

    /**
     * Remove a dataset from UBCInsight.
     *
     * @param id  The id of the dataset to remove.
     *
     * @return Promise <InsightResponse>
     *
```

```
     * The promise should return an InsightResponse for both fulfill and reject.
     *
     * Fulfill should be for 2XX codes and reject for everything else.
     *
     * This will delete both disk and memory caches for the dataset for the id
meaning
     * that subsequent queries for that id should fail unless a new addDataset
happens first.
     *
     * Response codes:
     *
     * 204: the operation was successful.
     * 404: the operation was unsuccessful because the delete was for a resource
that
     * was not previously added.
     *
     */
    removeDataset(id: string): Promise<InsightResponse>;

    /**
     * Perform a query on UBCInsight.
     *
     * @param query  The query to be performed. This is the same as the body of the
POST message.
     *
     * @return Promise <InsightResponse>
     *
     * The promise should return an InsightResponse for both fulfill and reject.
     *
     * Fulfill should be for 2XX codes and reject for everything else.
     *
     * Return codes:
     *
     * 200: the query was successfully answered. The result should be sent in JSON
according in the response body.
     * 400: the query failed; body should contain {"error": "my text"} providing
extra detail.
     */
    performQuery(query: any): Promise<InsightResponse>;

    /**
     * List a list of datasets and their types.
     *
     * @return Promise <InsightResponse>
     * The promise should return an InsightResponse and will only fulfill.
     * The body of this InsightResponse will contain an InsightDataset[]
     *
     * Return codes:
     * 200: The list of added datasets was sucessfully returned.
     */
    listDatasets(): Promise<InsightResponse>;
}
```

Testing

The best way to test your system is via your own unit test suite. You can write these unit

tests by following the examples in `test/` and running them with `yarn test`. This will be the quickest and easiest way to ensure your system is behaving correctly and to make sure regressions are not introduced as you proceed further in the project. We are currently also providing a UI for our solution for this deliverable so you can see what the expected values should be for the queries you are trying for your query.

To ensure your code conforms with the API our marking suite expects you can run your code against AutoTest. You will nothave source-level access this suite. For verified learners, you will be able to request to run it against your implementation every 12 hours by invoking the `@ubcbot` Github bot, while audit learners may do so every 60 hours; full details will be available in the AutoTest documentation.

Getting started

This deliverable might seem intimidating, but keep in mind that this project has the same interaction mechanism as most software systems:

1. It consumes input data (the zip file).
2. It transforms the data (according to the query).
3. It returns a result.

There is no best way to get started, but you can consider each of these in turn. Some possible options that could be pursued in any order (or skipped entirely):

- Start by looking at the data file we have provided and understanding what kind of data you will be analyzing and manipulating. This will help you think about the kind of data structure you want to create (this is the precursor to step 1 above).
- Look at the sample queries in the deliverable description. From these queries, figure out how you would want the data arranged so you can answer these queries (this is the precursor to step 2 above).
- Ignoring the provided data, create some fake data (maybe for one section of one course). Write the portion of the system that queries this data (this is step 2 above).
- Like the above, using some fake data and a fake query processor, write the code that would return the data correctly and with the correct error codes (this is step 3 above).

Trying to keep all of the requirements in mind at once is going to be overwhelming. Tackling a single task that you can accomplish in an hour is going to be much more effective than worrying about the whole deliverable at once. Iteratively growing your project from small task to small task is going to be the best way to make forward progress.

Getting AutoTest to work

1. Create a file `src/controller/InsightFacade` that contains a class called `InsightFacade` that implements the interface `IInsightFacade`.
2. Make sure the three methods in your `InsightFacade` class return a promise and that this promise *always* settles. If all three methods do not always settle, the AutoTest suite will timeout. Your D0 methods should have provided some insight into

how to do this.

3. We will run your unit test suite and send you all of the output for any of your tests that fail. This is a great way to figure out why code that works on your machine fails on our infrastructure. It is also a good way for you to ensure your promises always settle.
4. Your test suite should *not* need to use `JSZip`. Your test code can use `fs` to read the zip from disk and send the base64 string representation to `InsightFacade` which will *then* use `JSZip`. Also, your product (non-test) code should not read or write the zip file to disk, it should only read and write your data structure.

Assessment

Please refer to the Project Description for more information on grading.