COMP 4513 Assignment #1

Due Monday February 24th at midnight
Version 2.0, Feb. 11, changes in yellow

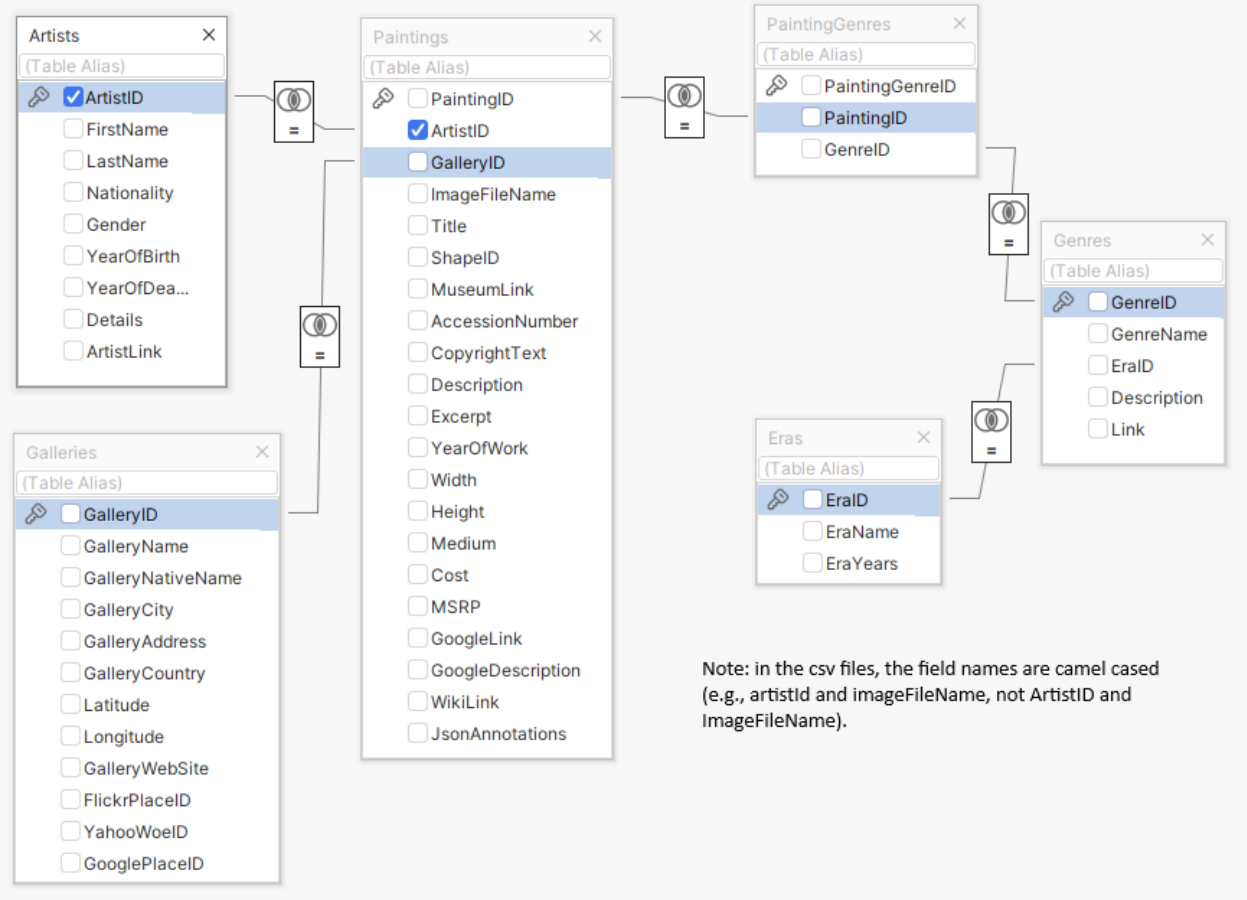
Overview

This assignment provides you with an opportunity to create an API in Node. You will also be required to provision a database on supabase.

Files

The art database represents paintings, artists, genres, galleries, and so on. You will be able to find the csv files (eventually) as well at the GitHub repo for the assignment:

`https://github.com/mru-comp4513-archive/w2025-assign1`



Grading

The grade for this assignment will be broken down as follows:

Programming Design and Documentation	15%
Hosting + Correct readme	10%

Functionality (follows requirements)	75%

Recommended Workflow

I recommend you approach this assignment in the following order:

1. Complete Lab14b
2. Set up a github repo for your source code.
3. Implement the SQL for each of the API routes using the SQL Editor in supabase. Save the SQL query text in a file. While this step is not strictly necessary, it will make step 5 easier.
4. Compare your query results to someone else's query results. Again, this isn't necessary, but it might help you catch errors in your query logic for the more complex queries.
5. Implement the APIs in Express. If you have created the SQL queries first, then you can simply convert them to supabase's query builder syntax.
6. Set up the hosting. The hosting might take longer to set up then you anticipate, so be sure to leave ample time for it.
7. Construct a `readme.md` file in your github repo with the expected example API request links (see page 5).
8. Test the link APIs in the readme file after hosting!!!
9. Send me an email with the required info (see page 2).

Submitting and Hosting

You will be using Node in this assignment. This will mean your assignment will need to reside on a working host server. Static hosts (e.g., github pages) will not work.

For this assignment, I would recommend using either glitch.com or render.com, both of which provides a free option for hosting Node applications. Do note that these free projects go to sleep after a set period of inactivity, so be aware that the first request of a slept hosted node application will take some time to awaken.

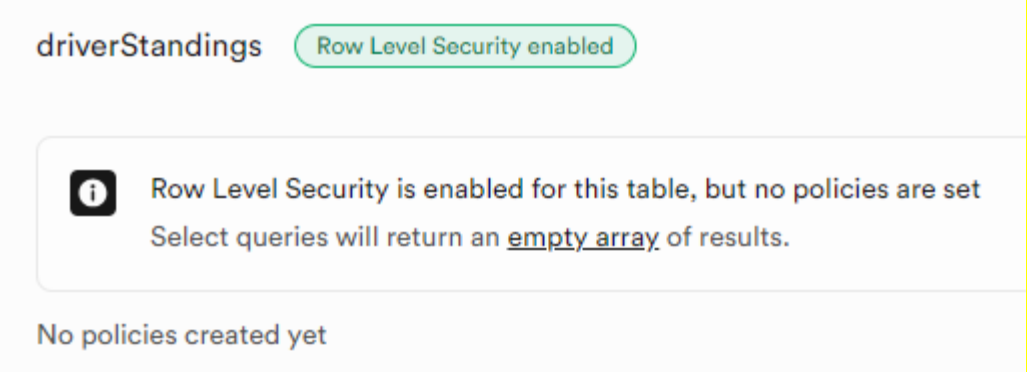
When your hosting is working and the assignment is ready to be marked, then send me an email with the following information:

- The URL of the github repo so that I can mark the source code. If your repo is private, then add me as a collaborator.
- In the `readme.md` file for your repo, provide links to the APIs on glitch/render so I can test them (see details below).

NOTE: Free tier supabase projects will go inactive after one week of inactivity (and thus your API won't work). Please login into supabase and run a test query every few days after the due date so this doesn't happen!!

Common Problems

1. API returns a blank screen. Evidently error messages are for rookies and not for supabase users. Generally this happens because there is a problem in your SQL (i.e., your field list, your filter, or your modifier). Fix it! Check for field names being incorrectly spelled or that have the wrong case. Are you including a field from a table in which relations haven't been set? **Look for trailing commas as they will mess it up.**
2. API returns an empty array. You likely haven't set a row-level security policy that allows read only access for one of the tables. For instance (ignore the table name, this is from last year):



3. You are getting results back from supabase that shouldn't be returned based on the filters. For instance, instead of a full object, you will see a **null** value instead. You will see this when you are trying to use a filter on a joined table, e.g.,

```
.select(`paintingId,yearOfWork, artists (artistId,firstName)`)
.eq(artists.artistId',12)
```

Strangely (at least to me), supabase seems to default to a FULL JOIN rather than an INNER JOIN in this situation. For instance, let's say you want to see paintings and artist data for a specific artist. When it does a FULL JOIN, you will get not only paintings for that artist, but all paintings not by that artist. But because the filter is in place, the paintings not by that artist will have a `artists` value of null.

To fix this, you need to tell supabase to make the join an inner join via:
`.select(`paintingId,yearOfWork, artists!inner (artistId,firstName)`)`

4. Your sort isn't working or is resulting in the dreaded blank screen. To sort on a field in a linked table, you have to add in a `referencedTable` property, e.g.,
`.order('yearOfWork', { referencedTable: 'paintings', ascending: false })`

API Functionality

You must create the following APIs with the specified routes and functionality. The returned data must be JSON format.

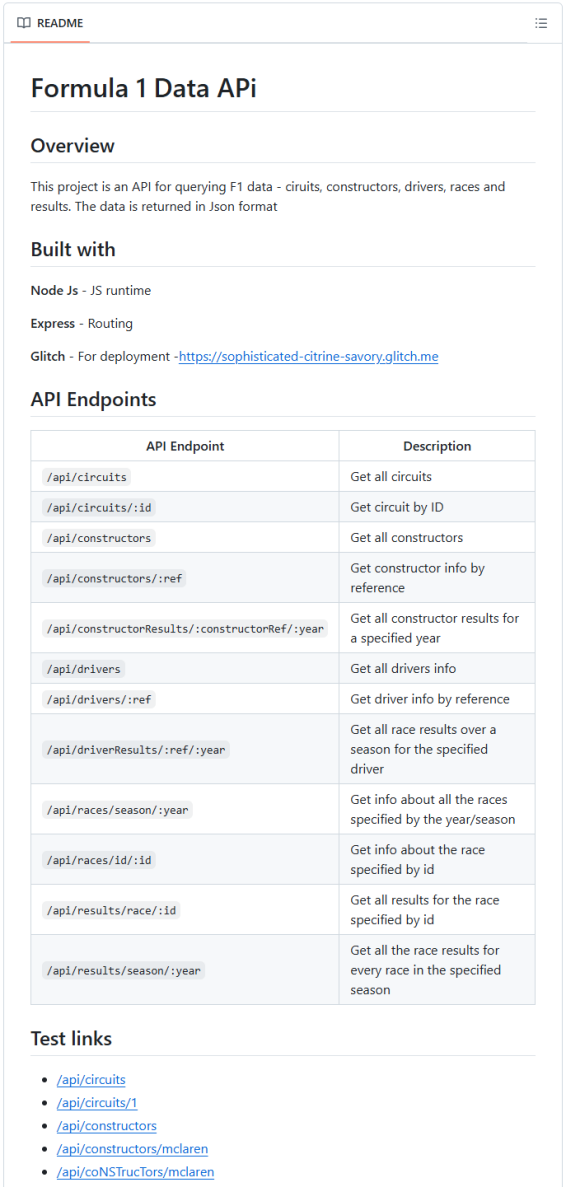
`/api/eras`	Returns all the eras
`/api/galleries`	Returns all the galleries (for this, and other `gallery` requests, return all the fields in the `galleries` table)
`/api/galleries/`*`ref`*	Returns just the specified gallery (use the `galleryId` field), e.g., `/api/galleries/30`
`/api/galleries/country/`*`substring`*	Returns the galleries whose `galleryCountry` (case insensitive) **begins** with the provided substring, e.g., `/api/galleries/country/fra`
`/api/artists`	Returns all the artists (for this, and other `artist` requests, return all the fields in the `artists` table)
`/api/artists/`*`ref`*	Returns just the specified artist (use the `artistId` field), e.g., `/api/artists/12`
`/api/artists/search/`*`substring`*	Returns the artists whose last name (case insensitive) **begins** with the provided substring, e.g., `/api/artist/search/ma`
`/api/artists/country/`*`substring`*	Returns the artists whose nationality (case insensitive) **begins** with the provided substring, e.g., `/api/artists/country/fra`
`/api/paintings`	Returns all the paintings (for this, and other `paintings` requests, return all the fields in the `paintings` table, but not the foreign keys). **For this and the following paintings requests, don't just provide the foreign keys for artist and gallery; instead provide all the artist fields and all the gallery fields. By default, for the painting requests, sort by title unless specified otherwise.**
`/api/paintings/sort/`*`title|year`*	Returns all the paintings, sorted by either `title` or `yearOfWork`.
`/api/paintings/`*`ref`*	Returns just the specified painting, e.g., `/api/paintings/63`
`/api/paintings/search/`*`substring`*	Returns the paintings whose title (case insensitive) **contains** the provided substring, e.g., `/api/paintings/search/port`
`/api/paintings/years/`*`start`*`/`*`end`*	Returns the paintings between two years (include the paintings in the provided years), ordered by `yearOfWork` e.g., `/api/paintings/years/1800/1850`

`/api/paintings/galleries/ref`	Returns all the paintings in a given gallery (use the `galleryId` field), e.g., `/api/paintings/galleries/5`
`/api/paintings/artist/ref`	Returns all the paintings by a given artist (use the `artistId` field), e.g., `/api/paintings/artist/16`
`/api/paintings/artists/country/ref`	Returns all the paintings by artists whose nationality **begins** with the provided substring, e.g., `api/paintings/artists/country/ital`
`/api/genres`	Returns all the genres. **For this and the following genres requests, don't just provide the foreign keys for era; instead provide all the era fields**
`/api/genres/ref`	Returns just the specified genre (use the `genreId` field), e.g., `/api/genres/76`
`/api/genres/painting/ref`	Returns the genres used in a given painting (order by `genreName` in ascending order), e.g., `/api/genres/painting/408`
`/api/paintings/genre/ref`	Returns all the paintings for a given genre (use the `genreId` field), e.g., `/api/paintings/genre/78` For this query, you do not have to return all the painting fields, just `paintingId` , `title`, and `yearOfWork`. Sort by `yearOfWork`.
`/api/paintings/era/ref`	Returns all the paintings for a given era (use the `eraId` field), e.g., `/api/paintings/era/2` For this query, you do not have to return all the painting fields, just `paintingId` , `title`, and `yearOfWork`. Sort by `yearOfWork`.
`/api/counts/genres`	Returns the genre name and the number of paintings for each genre, sorted by the number of paintings (fewest to most).
`/api/counts/artists`	Returns the artist name (first name space last name) and the number of paintings for each artist, sorted by the number of paintings (most to fewest).
`/api/counts/topgenres/ref`	Returns the genre name and the number of paintings for each genre, sorted by the number of paintings (most to least) for genres having over some set number of paintings, e.g., `/api/counts/topgenres/20` would show the painting counts for those genres with more than 20 paintings.

For each of the requests that take parameters, your API needs to handle a Not Found condition. For instance, if a ref or year doesn't exist, **return a JSON-formatted error message** that indicates the requested request did not return any data. For the routes with a start and end year, provide a different error message if the end year is earlier than the start year.

GitHub Readme

Remember that potential employers will look at your github repos and will want to see best practices. That is, think of your github repo readme files as if they were part of your resume/portfolio. Here are some excellent sample github readmes from a different class, but still demonstrates the level of detail that you may want to emulate.



README

Formula 1 Data APi

Overview

This project is an API for querying F1 data - ciruits, constructors, drivers, races and results. The data is returned in Json format

Built with

Node Js - JS runtime

Express - Routing

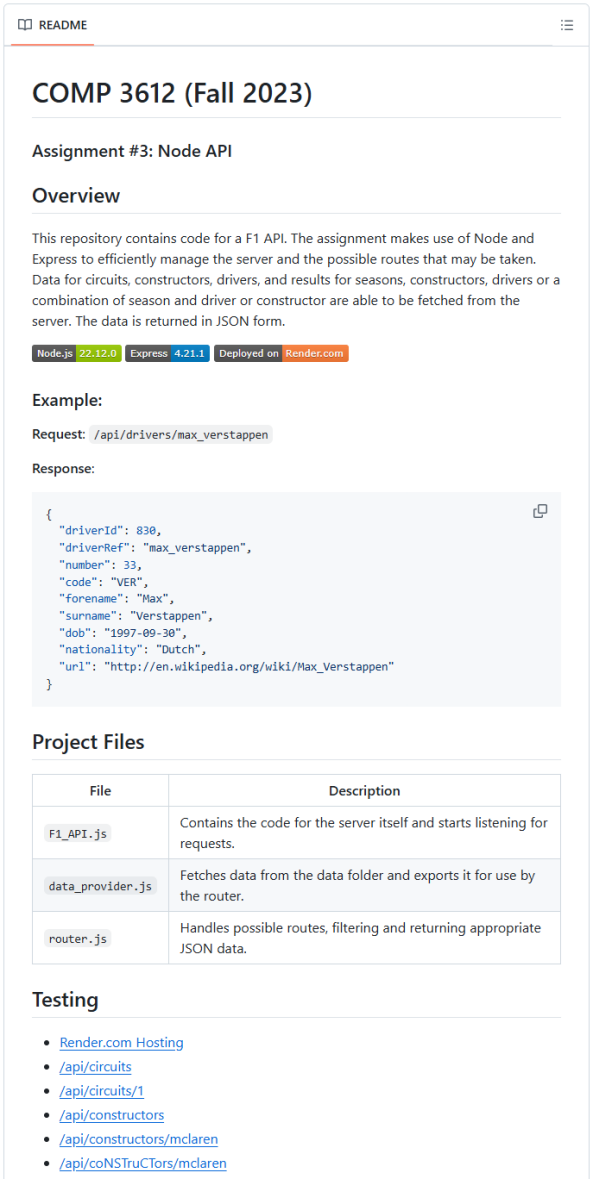
Glitch - For deployment -https://sophisticated-citrine-savory.glitch.me

API Endpoints

API Endpoint	Description
`/api/circuits`	Get all circuits
`/api/circuits/:id`	Get circuit by ID
`/api/constructors`	Get all constructors
`/api/constructors/:ref`	Get constructor info by reference
`/api/constructorResults/:constructorRef/:year`	Get all constructor results for a specified year
`/api/drivers`	Get all drivers info
`/api/drivers/:ref`	Get driver info by reference
`/api/driverResults/:ref/:year`	Get all race results over a season for the specified driver
`/api/races/season/:year`	Get info about all the races specified by the year/season
`/api/races/id/:id`	Get info about the race specified by id
`/api/results/race/:id`	Get all results for the race specified by id
`/api/results/season/:year`	Get all the race results for every race in the specified season

Test links

- /api/circuits
- /api/circuits/1
- /api/constructors
- /api/constructors/mclaren
- /api/coNSTrucTors/mclaren



README

COMP 3612 (Fall 2023)

Assignment #3: Node API

Overview

This repository contains code for a F1 API. The assignment makes use of Node and Express to efficiently manage the server and the possible routes that may be taken. Data for circuits, constructors, drivers, and results for seasons, constructors, drivers or a combination of season and driver or constructor are able to be fetched from the server. The data is returned in JSON form.

Node.js 22.12.0 | Express 4.21.1 | Deployed on Render.com

Example:

Request: `/api/drivers/max_verstappen`

Response:

```
{
  "driverId": 830,
  "driverRef": "max_verstappen",
  "number": 33,
  "code": "VER",
  "forename": "Max",
  "surname": "Verstappen",
  "dob": "1997-09-30",
  "nationality": "Dutch",
  "url": "http://en.wikipedia.org/wiki/Max_Verstappen"
}
```

Project Files

File	Description
`F1_API.js`	Contains the code for the server itself and starts listening for requests.
`data_provider.js`	Fetches data from the data folder and exports it for use by the router.
`router.js`	Handles possible routes, filtering and returning appropriate JSON data.

Testing

- Render.com Hosting
- /api/circuits
- /api/circuits/1
- /api/constructors
- /api/constructors/mclaren
- /api/coNSTruCTors/mclaren

Example API Requests

In the `readme.md` file for your assignment repo, you must supply a list of links that allow me to test each of your APIs. Please add the following test links (they must be clickable links) in this file:

/api/eras	/api/paintings/years/1800/1850
/api/galleries	/api/paintings/galleries/5
/api/galleries/30	/api/paintings/artist/16
/api/galleries/Calgary	/api/paintings/artist/666
/api/galleries/country/fra	/api/paintings/artist/country/ital
/api/artists	/api/genres
/api/artists/12	/api/genres/76
/api/artists/1223423	/api/genres/painting/408
/api/artists/search/ma	/api/genres/painting/jsdfhg
/api/artists/search/mA	/api/paintings/genre/78
/api/artists/country/fra	/api/paintings/era/2
/api/paintings	/api/counts/genres
/api/paintings/sort/year	/api/counts/artists
/api/paintings/63	/api/counts/topgenres/20
/api/paintings/search/port	/api/counts/topgenres/2034958
/api/paintings/search/pORt	
/api/paintings/search/connolly	

Note: you will need to preface the above URLs with the URL of your host. For instance, if your Glitch URL is `https:/smashing-squirrels.glitch.me`, then the URL for the second test link would be `https:/smashing-squirrels.glitch.me/api/paintings/63`