Team KASS Project Reflection

1. Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).
There have been no changes to our project between our stage 1 proposal submission and our final project submission. We kept our final submission identical to the original project proposal since our original plan accounted for any potential pitfalls, so we were able to avoid having to make drastic changes throughout the semester.

One thing we proposed in Stage 1, which we did not add to our final project, was https://www.kaggle.com/datasets/llui85/tokyo-2021-olympics-complete-grouped-by-type was another dataset that we wanted to integrate onto our 10000+ row TA-approved dataset. We decided against this as the data was incompatible with our dataset, and it did not add much functionality. The joins etc. would have been too complicated due to the different data types (CSV vs. JSON) and many more problems, which all would have added to unnecessary problems.

We also did not implement this additional feature as it would be beyond the scope as the dataset did not have booking or event spaces. We did not have this data on the TA-approved dataset, but we did find data from another source. We added filtering and ticket count in our App.

Another thing we wanted to do was role-based authentication to do CRUD operations, but this was again an add-on feature. Instead, we directly do Authentication based access to perform CRUD operations.

Again, note that all the changes that were not implemented above were categorized as add-ons in our application. We delivered all the core functionality and added more features, such as visualization tools.

---

2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.
Our application has successfully created a place where a user can sort through and analyze the 2020 Tokyo Olympics data. All while being able to add and delete athletes to the data set, update the medal counts of different countries, and filter through different events. Additionally, the visualizations provide easy-to-understand readability to the large sets of data in our application's athlete and medal sections.

We have succeeded on multiple fronts, and we do this in the following ways - Data and Functionality.

Data
We have different kinds of data, workflows, and access controls

- We cleaned and set up data for our application
- User data for authentication, authorization, and access control. It would include attributes such as username and password.

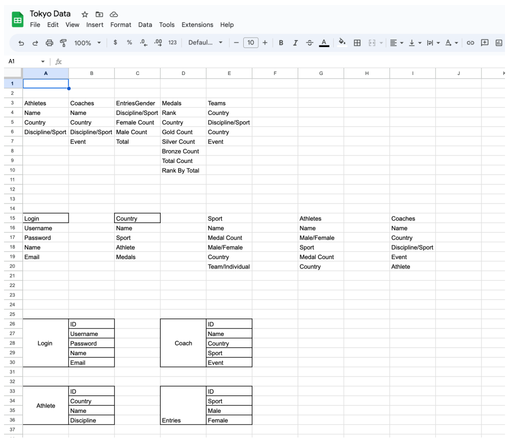
Functions
Any user has the following abilities that they can leverage from our website.

- One can log in to our website, and we process authentication and authorization.
- Anyone can leverage the search sort functionality of our website.
- Anyone can leverage the visualizations on the website.
- Anyone can request that certain athletes' data be added, updated, or deleted after authentication.
- Data enforcement for certain fields, such as countries, so random Non-existent countries can't be added.
- Have Stored Procedures and triggers to update stats and add functionality via UI, and we were able to visualize through the same.

We set out to create an ultimate best-case website that functions better than the official website, and we were able to create a wonderful user experience for the same.

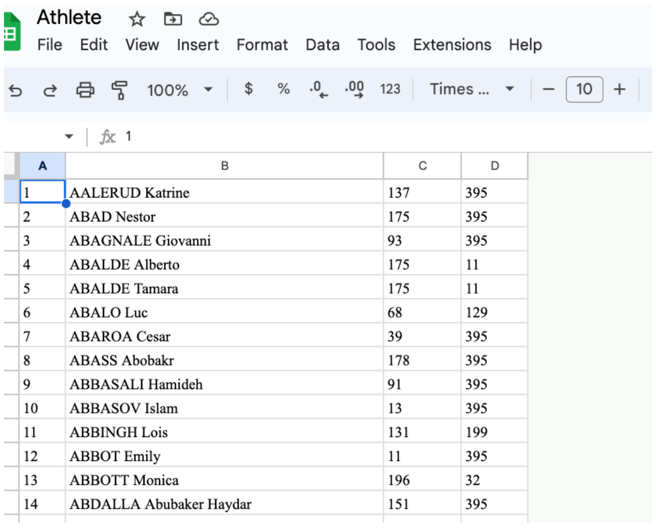
---

3. Discuss if you changed the schema or source of the data for your application
We had to modify the TA-approved dataset by introducing IDs to ensure that the tables have meaning between themselves, and this way, we were able to import the dataset directly into our backend and did not have to worry about complex queries and indexing concerns.


Tokyo Data

	A	B	C	D	E	F	G	H	I	J
1										
2										
3	Athletes	Coaches	EntriesGender	Medals	Teams					
4	Name	Name	Discipline/Sport	Rank	Country					
5	Country	Country	Female Count	Country	Discipline/Sport					
6	Discipline/Sport	Discipline/Sport	Male Count	Gold Count	Country					
7		Event	Total	Silver Count	Event					
8				Bronze Count						
9				Total Count						
10				Rank By Total						
11										
12										
13										
14										
15	Login		Country		Sport		Athletes		Coaches	
16	Username		Name		Name		Name		Name	
17	Password		Sport		Medal Count		Male/Female		Country	
18	Name		Athlete		Male/Female		Sport		Discipline/Sport	
19	Email		Medals		Country		Medal Count		Event	
20					Team/Individual		Country		Athlete	
21										
22										
23										
24										
25										
26		ID			ID					
27		Username			Name					
28	Login	Password		Coach	Country					
29		Name			Sport					
30		Email			Event					
31										
32										
33		ID			ID					
34	Athlete	Country			Sport					
35		Name			Male					
36		Discipline		Entries	Female					
37										

Above is the original schema, and we have the data to be mapped only via FKs across wherever applicable



A	B	C	D
1	AALERUD Katrine	137	395
2	ABAD Nestor	175	395
3	ABAGNALE Giovanni	93	395
4	ABALDE Alberto	175	11
5	ABALDE Tamara	175	11
6	ABALO Luc	68	129
7	ABAROA Cesar	39	395
8	ABASS Abobakr	178	395
9	ABBASALI Hamideh	91	395
10	ABBASOV Islam	13	395
11	ABBINGH Lois	131	199
12	ABBOT Emily	11	395
13	ABBOTT Monica	196	32
14	ABDALLA Abubaker Haydar	151	395

But this can be categorized as cleaning, and the original data was kept intact. However, we had to generate data for User tables synthetically. Additionally, we had to add a new dataset for the event table as it did not come under the original table or dataset.

---

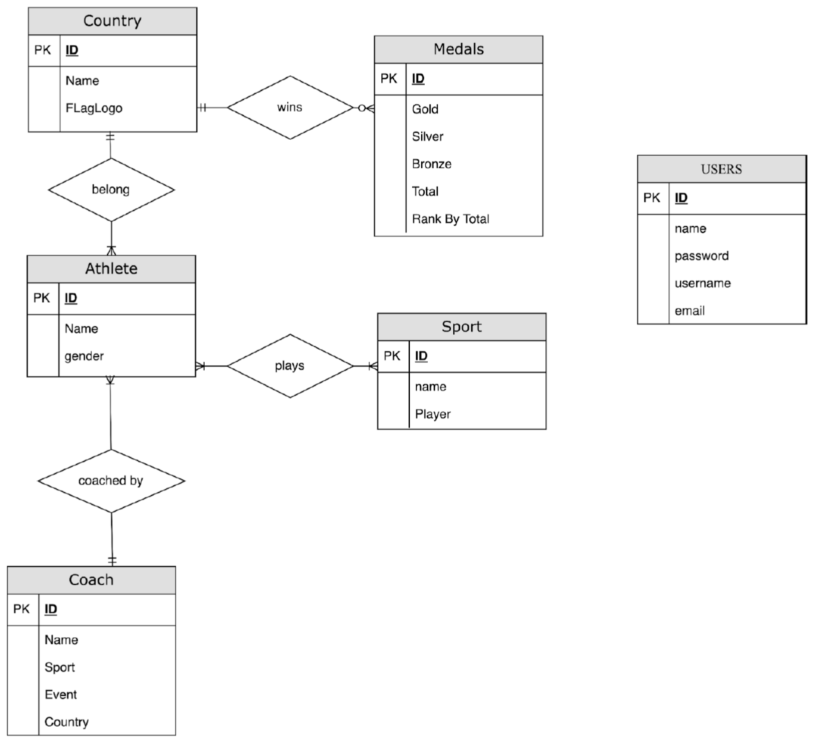
4. Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

We spent a lot of time designing the ER diagram perfectly the first time, and we discussed a lot about it. Our ER diagram is not changed from the proposal and is maintained intact as is. By making sure that we spent a lot of time designing the proper schema, we could design the entire schema and app properly the first time without any modifications. The DDL statements also remained intact. The only change in the databases is that we had to modify the athlete table once

```
CREATE TABLE Athlete (
ID INT PRIMARY KEY,
NAME VARCHAR(100) ,
COUNTRYID INT,
COACHID INT,
FOREIGN KEY (COUNTRYID) REFERENCES Country(ID),
FOREIGN KEY (COACHID) REFERENCES Coach(ID)
);
```

As ID did not have auto-increment and we had to alter the table to add AUTO_INCREMENT.

ER Diagram



ER is the same as the beginning, as explained above.

---

5. Discuss what functionalities you added or removed. Why?

We added the following functionalities overall to our project. Anything outside the scope of the minimum requirement or any changes from the proposal will be explained in line.

Added functionality based on Proposal

- Inserting Athlete Information (CRUD)
- Deleting Athlete Information (CRUD)
- Searching Athlete Information (CRUD)
- Visualising World Map with Athlete Count
- Stored Procedure + Visualization Bubble chart showing Athlete count based on country/sport
- Filtered view to find athletes based on sport/countries (This was an implementation of an advanced query outside the Stored Procedure - Filtering on the front end)
- Stored Procedure + Filtered view to find athletes based on sport/countries
- The visualization that finds athletes based on sport/countries (This was an implementation of an advanced query outside the Stored Procedure - Filtering on the front end but is done on Python Plotly) - Analytics
- Medal ranking table from a stored procedure

- Medal stats making country ranking visualization (This is a visualization of Trigger)
- Update Medal (CRUD)
- Add/Delete Trigger
- Event Page (Filtered View)

Removed Functionalities

- Based on feasibility & time constraints, the potential to book event spaces (Role/Privilege based) - This was beyond the scope of original requirements and was an add-on, and we wanted to offer functionality within the scope of a conventional Olympic website for now. We had to drop this as it was demanding and we would not be able to complete this on time as it was already beyond the scope during the proposal

---

6. Explain how you think your advanced database programs complement your application.
Our advanced programs allow us to filter through all of our data. If a user wanted to find athletes from a certain country or sport, they would not have a convenient way to search for that information. Our first advanced query allows us to filter by country and sport to find all the athletes that fit those categories. Our second, advanced query allows us to display countries according to their medal count. This is especially useful in this dataset because each country places importance on its overall placement.

For the Advanced Query 1 + Stored Procedure:
We implemented this functionality in two places. Advanced Query 1 lets me search for Athletes based on Country or Sport Name, and this query forms the backbone of my filtering processes as it lets us easily filter data either from the front or back end.
The AQ1 implemented a standalone API that lets us filter athletes from the front end only, where we will pass over 10000+ rows of data, and we could filter on React directly.
The AQ1 was also implemented in Python to let us visualize the filters directly and as a Stored Procedure where the user is able to select filters and conduct the search through that feature.

For the Advanced Query 2 + Stored Procedure:
We implemented this functionality to let us visualize the athletes and countries vs. sports. This allowed us to visualize the distribution of athletes based on their countries and sports or displayed completely on a map.

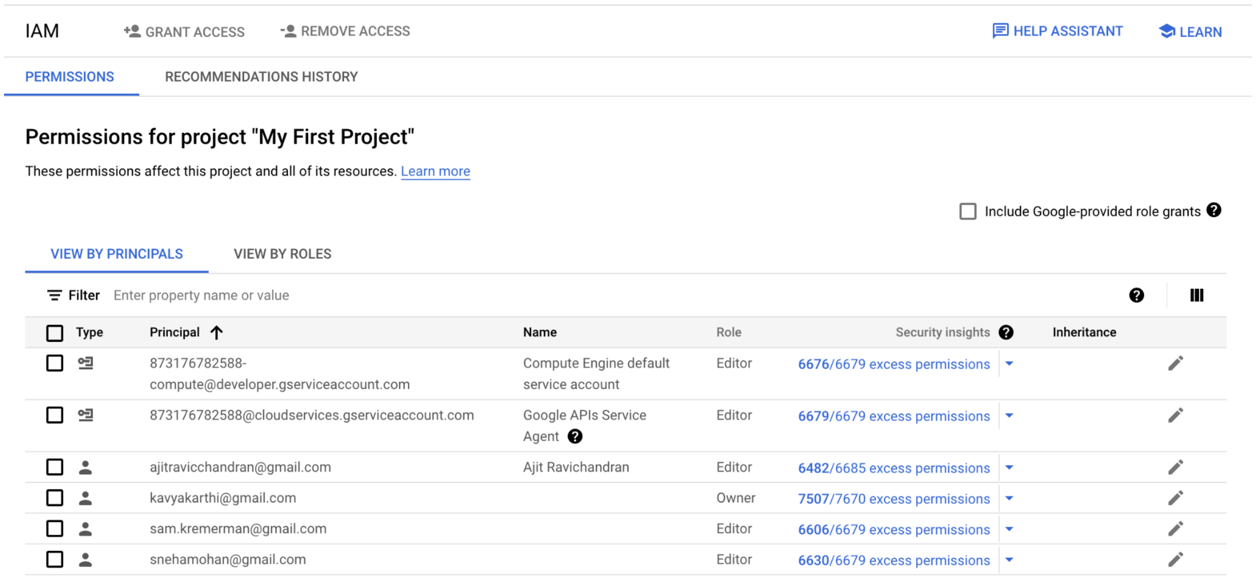
For the Trigger
We implemented a way to ensure that the medal count is kept intact, and by doing this, we were able to ensure rankings are correct when updates happen. It is critical that these filters add more functionality as they let us filter effectively, visualize more and keep data integrity in check.

---

7. Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

<u>Kavya</u>

A technical challenge that I encountered was the initial setup of the project. The documentation helped us set up the MySQL instance on GCP. However, when we moved on to adding teammates to the project, I was having trouble allowing them to access and edit the SQL instance. To solve this issue, I had to add my team members to the IAM users as editors to grant them full access.



Permissions for project "My First Project"

These permissions affect this project and all of its resources. Learn more

Include Google-provided role grants

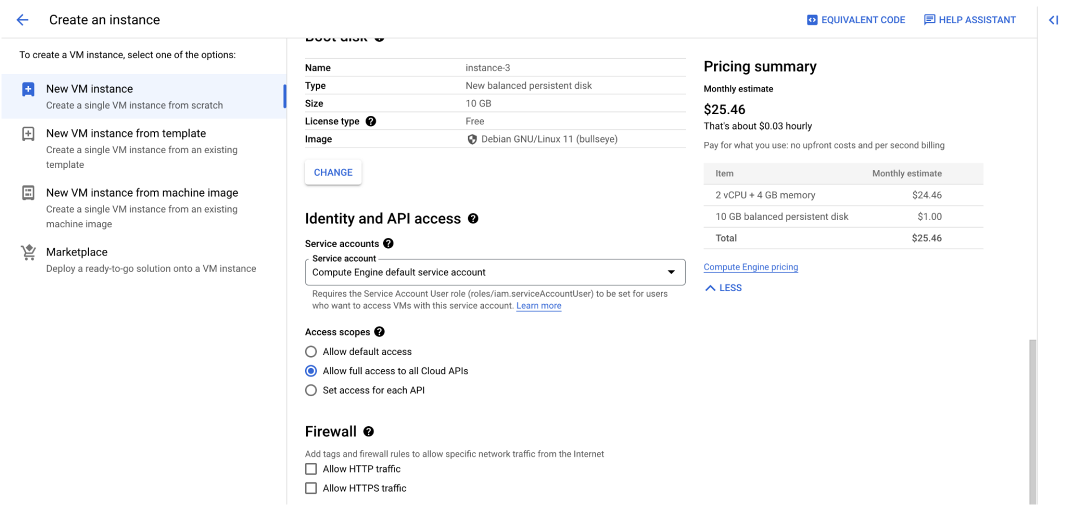
VIEW BY PRINCIPALS   VIEW BY ROLES

Filter Enter property name or value

Type	Principal	Name	Role	Security insights	Inheritance
	873176782588-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor	6676/6679 excess permissions	
	873176782588@cloudservices.gserviceaccount.com	Google APIs Service Agent	Editor	6679/6679 excess permissions	
	ajitravicchandran@gmail.com	Ajit Ravichandran	Editor	6482/6685 excess permissions	
	kavyakarthi@gmail.com		Owner	7507/7670 excess permissions	
	sam.kremerman@gmail.com		Editor	6606/6679 excess permissions	
	snehamohan@gmail.com		Editor	6630/6679 excess permissions	

One more issue was that GCP APIs were not working properly through the VM. The VM setup screen API access defaults to minimum API Access. This won't allow the VM to access API methods from backend services that invoke GCP APIs. To connect to the backend SQL server, I had to ensure that the main server had access to this endpoint.

One more issue that we encountered was the billing issues encountered, as we missed terminating the instance and had to add the promo codes from other members to counter the same.



To create a VM instance, select one of the options:

- New VM instance — Create a single VM instance from scratch
- New VM instance from template — Create a single VM instance from an existing template
- New VM instance from machine image — Create a single VM instance from an existing machine image
- Marketplace — Deploy a ready-to-go solution onto a VM instance

Name	instance-3
Type	New balanced persistent disk
Size	10 GB
License type	Free
Image	Debian GNU/Linux 11 (bullseye)

CHANGE

Identity and API access

Service accounts

Service account: Compute Engine default service account

Requires the Service Account User role (roles/iam.serviceAccountUser) to be set for users who want to access VMs with this service account. Learn more

Access scopes

( ) Allow default access
[✓ Allow full access to all Cloud APIs]
( ) Set access for each API

Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

☐ Allow HTTP traffic
☐ Allow HTTPS traffic

Pricing summary

Monthly estimate

$25.46

That's about $0.03 hourly

Pay for what you use: no upfront costs and per second billing

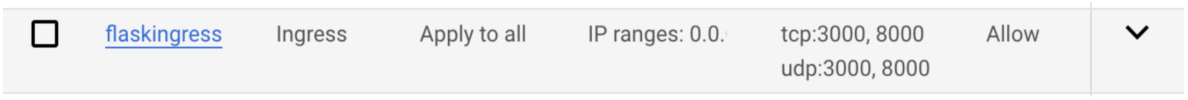
Item	Monthly estimate
2 vCPU + 4 GB memory	$24.46
10 GB balanced persistent disk	$1.00
Total	$25.46

Compute Engine pricing

LESS

In addition, we had faced issues integrating our visualizations to React front end, for which we had to modify the running of Python files to run as a standalone service that was later put onto the React framework.

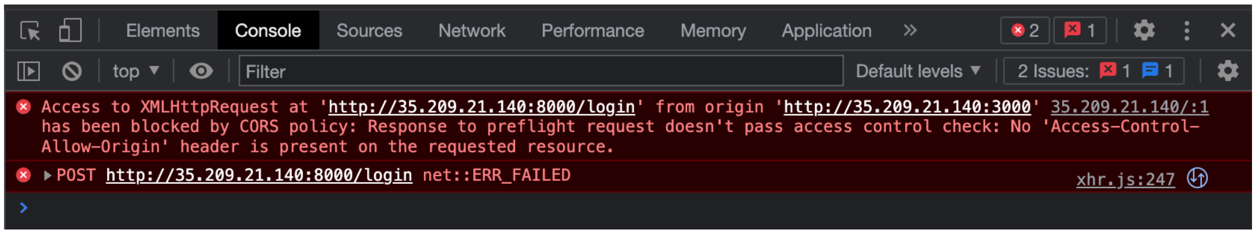
Additionally, the firewall can cause a lot of issues, for which we allowed all IP ranges access across through the firewall over the react port and backend port.



flaskingress	Ingress	Apply to all	IP ranges: 0.0.	tcp:3000, 8000 udp:3000, 8000	Allow

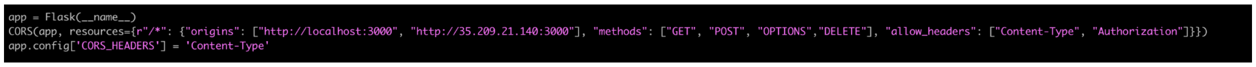
Ajit

When designing a backend service, a common problem that anyone will encounter is the CORS issue which would prevent other hosts or ports from utilizing any other port other than the one it is hosted from.
In our project, we host Flask in our backend from port 8000, and when we make requests from any other port other than 8000, we will encounter CORS issues. We have to fix this as the browser level ignoring this setting is complicated and not recommended.

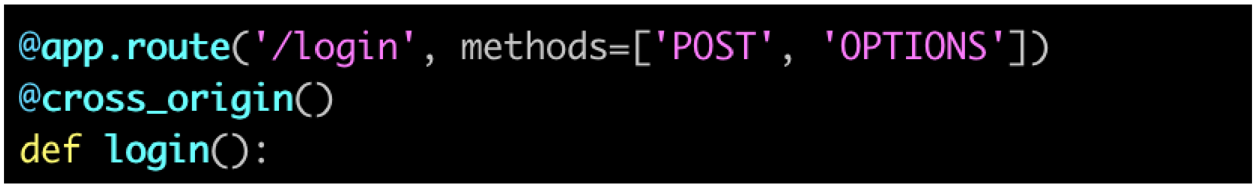


```
Access to XMLHttpRequest at 'http://35.209.21.140:8000/login' from origin 'http://35.209.21.140:3000' 35.209.21.140/:1
has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-
Allow-Origin' header is present on the requested resource.
▶POST http://35.209.21.140:8000/login net::ERR_FAILED                                              xhr.js:247
```

We had to allow CORS login in the backend, especially for POST methods as user login. Updates would not have worked if not, we fixed the issue, which was added in the backend to counter the same.



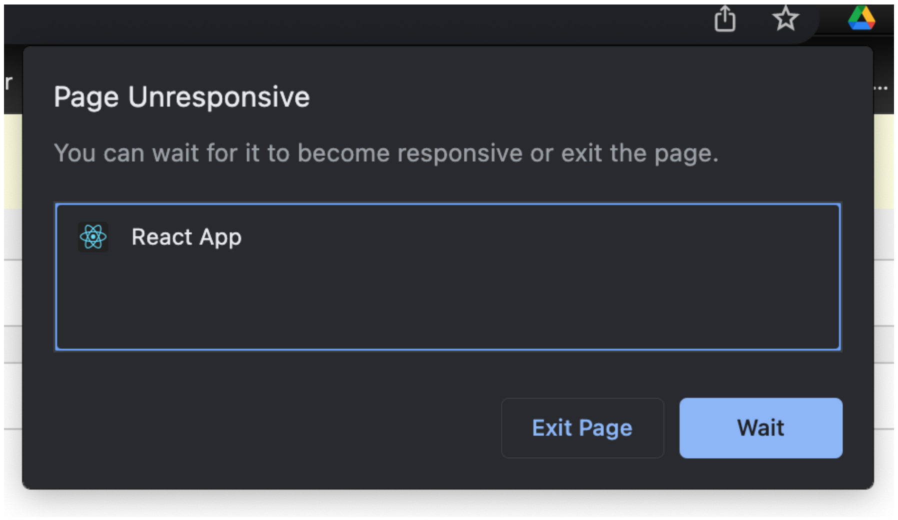
```
app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": ["http://localhost:3000", "http://35.209.21.140:3000"], "methods": ["GET", "POST", "OPTIONS","DELETE"], "allow_headers": ["Content-Type", "Authorization"]}})
app.config['CORS_HEADERS'] = 'Content-Type'
```



```
@app.route('/login', methods=['POST', 'OPTIONS'])
@cross_origin()
def login():
```

We added the methods above to counter the CORS issue.

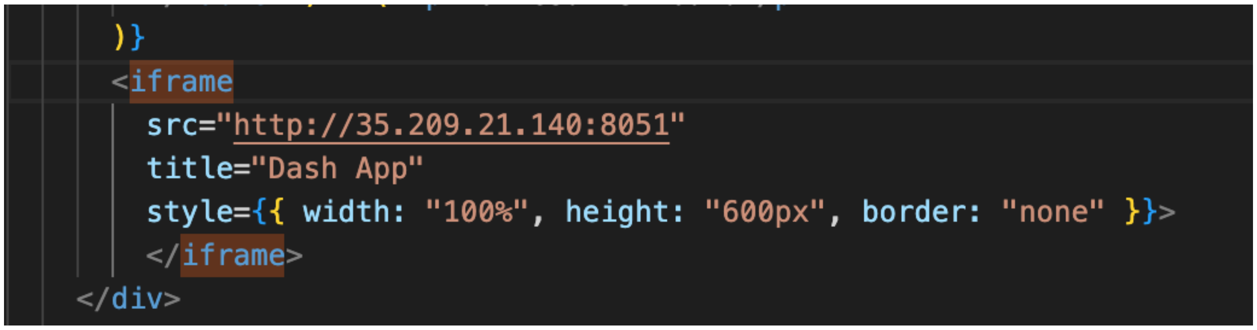
Another major issue we ended up encountering was that the backend was returning 10000+ rows in some stages. React had issues rendering the data on screen and straight up conked up the entire website, and we then, as a team, decided to shift to Python for rendering wherever a large number of rows.



Page Unresponsive

You can wait for it to become responsive or exit the page.

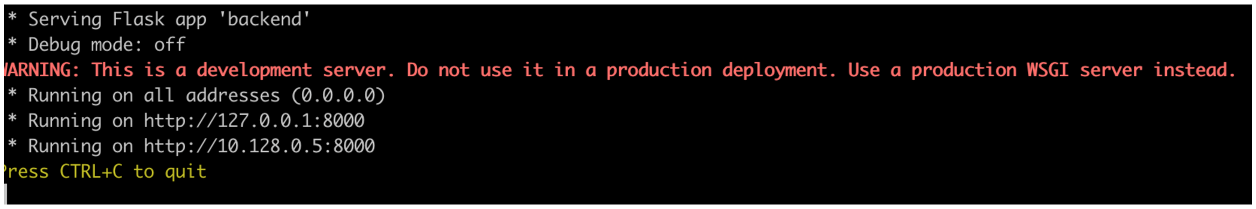
Python is very good at handling large numbers of data, and we pass over this react component as an iframe to front end.



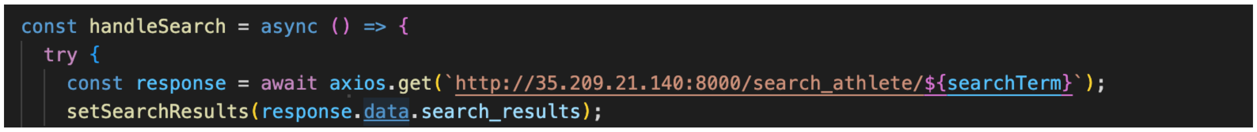
```
        )}
        <iframe
          src="http://35.209.21.140:8051"
          title="Dash App"
          style={{ width: "100%", height: "600px", border: "none" }}>
          </iframe>
      </div>
```

<u>Sneha</u>

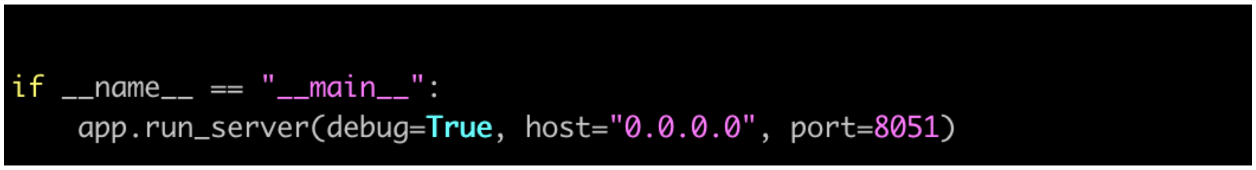
When running locally, the code runs on localhost or the loopback IP address 127.0.0.1 or on an internal IP.


```
* Serving Flask app 'backend'
* Debug mode: off
ARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8000
* Running on http://10.128.0.5:8000
ress CTRL+C to quit
```

While the code can work locally when using localhost, the API would not run on the server. When deployed to the server, the APIs had to be changed everywhere to the server's external IP, as putting localhost did not go well once the server was deployed.
Here one can see how the server IP address we have put up to ensure that it can communicate effectively with the backend server.


```
const handleSearch = async () => {
  try {
    const response = await axios.get(`http://35.209.21.140:8000/search_athlete/${searchTerm}`);
    setSearchResults(response.data.search_results);
```

We had to test locally with another project copy to ensure the copy runs fine locally and when it is deployed in the server.

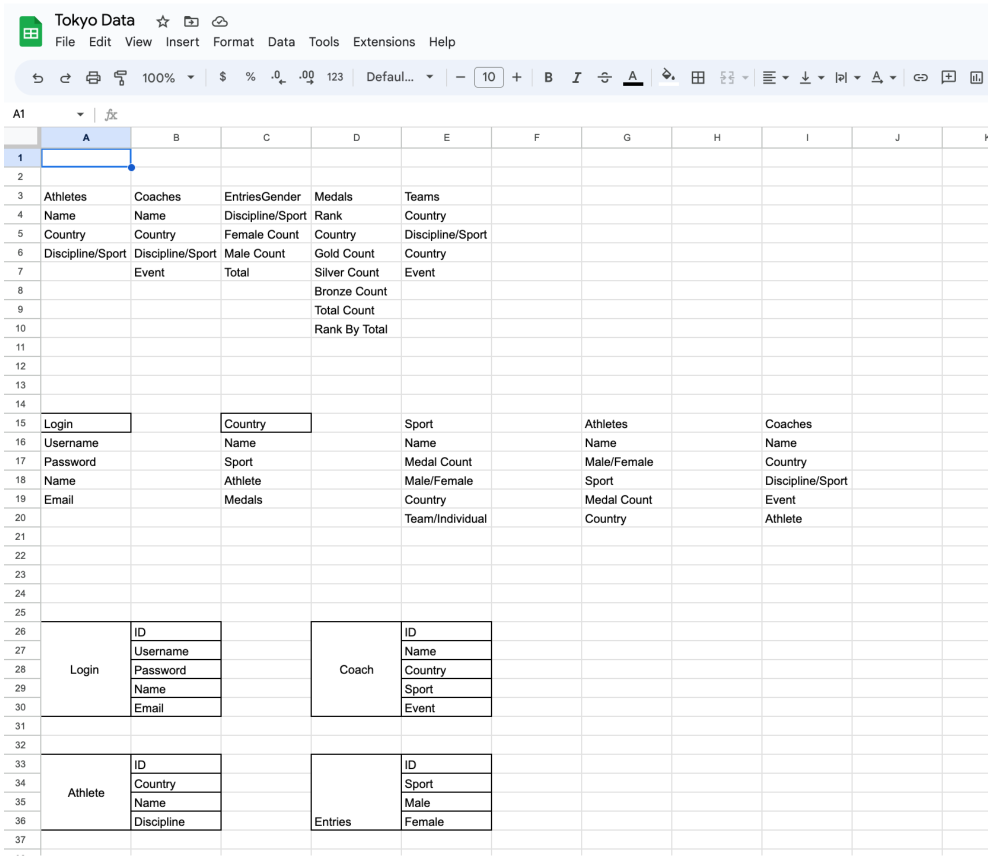

```
if __name__ == "__main__":
    app.run_server(debug=True, host="0.0.0.0", port=8051)
```

Additionally, when we implemented the Visualisations, we had to run it on port 0.0.0.0 so that it could accept incoming connections from any port.

Sam

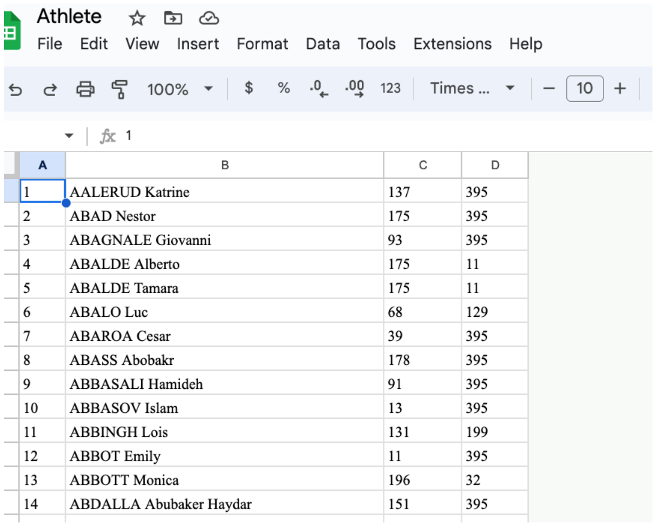
We faced lots of issues handling the dataset provided, our dataset involved was https://www.kaggle.com/datasets/arjunprasadsarkhel/2021-olympics-in-tokyo which consists of data that had no connection to one other.

The data looked like this.

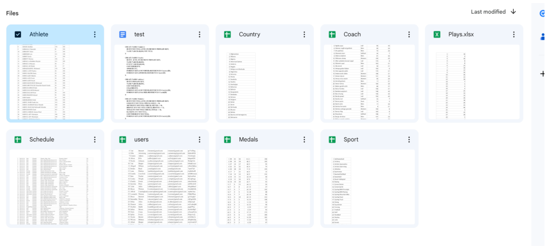


	A	B	C	D	E	F	G	H	I	J
1										
2										
3	Athletes	Coaches	EntriesGender	Medals	Teams					
4	Name	Name	Discipline/Sport	Rank	Country					
5	Country	Country	Female Count	Country	Discipline/Sport					
6	Discipline/Sport	Discipline/Sport	Male Count	Gold Count	Country					
7		Event	Total	Silver Count	Event					
8				Bronze Count						
9				Total Count						
10				Rank By Total						
11										
12										
13										
14										
15	Login		Country		Sport		Athletes		Coaches	
16	Username		Name		Name		Name		Name	
17	Password		Sport		Medal Count		Male/Female		Country	
18	Name		Athlete		Male/Female		Sport		Discipline/Sport	
19	Email		Medals		Country		Medal Count		Event	
20					Team/Individual		Country		Athlete	
21										
22										
23										
24										
25										
26		ID			ID					
27		Username			Name					
28	Login	Password		Coach	Country					
29		Name			Sport					
30		Email			Event					
31										
32										
33		ID			ID					
34		Country			Sport					
35	Athlete	Name			Male					
36		Discipline		Entries	Female					
37										

As described above, the provided data did not have any direct way to join. We added IDs directly on the CSV file and then added columns.



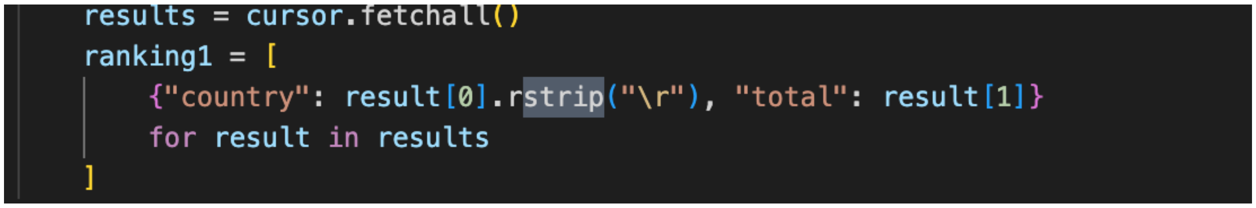
	A	B	C	D
1	1	AALERUD Katrine	137	395
2	2	ABAD Nestor	175	395
3	3	ABAGNALE Giovanni	93	395
4	4	ABALDE Alberto	175	11
5	5	ABALDE Tamara	175	11
6	6	ABALO Luc	68	129
7	7	ABAROA Cesar	39	395
8	8	ABASS Abobakr	178	395
9	9	ABBASALI Hamideh	91	395
10	10	ABBASOV Islam	13	395
11	11	ABBINGH Lois	131	199
12	12	ABBOT Emily	11	395
13	13	ABBOTT Monica	196	32
14	14	ABDALLA Abubaker Haydar	151	395



We used Excel Vlookups and Python panda frames to load data to manipulate and join data which helped us easily do our DDL statements later on.

One more issue was when playing around with CSV data, sometimes the data in the column ended with /r in the CSV, for which we changed DB to strip out /r in the data.

The front and backend had to encounter this proactively and handle this character



```
    results = cursor.fetchall()
    ranking1 = [
        {"country": result[0].rstrip("\r"), "total": result[1]}
        for result in results
    ]
```

---

8. Are there other things that changed comparing the final application with the original proposal?

We made no other changes between our original proposal and our final deliverables.

We were able to deliver all the functionality that we promised initially.

We had to remove one functionality as we initially stated this was an addon at best, and we had to exclude this as it would've eaten more into our time and would have added more unnecessary effort.

- Based on feasibility & time constraints, the potential to book event spaces (Role/Privilege based) - This was beyond the scope of original requirements and was an add-on, and we wanted to offer functionality within the scope of a conventional Olympic website for now, and we had to drop this as it was demanding and we would not be able to complete this on time as it was already beyond the scope during the proposal

---

9. Describe future work that you think, other than the interface, that the application can improve on

Other than the interface, one improvement that we could have made was improving interactive components within the application. While we have the functionality to add and delete athletes and update medal counts, we could have found other ways to allow users to interact with the data through creative components. Additionally, we could find additional datasets to allow users to see where specific countries are earning their medals and the outcomes of highly anticipated events. There would have been an additional page that would allow users to scroll through the events for the scores, highlights, and the next time those athletes will compete.

We were bound severely by the dataset as it did not have many fields we would have preferred, such as images and descriptions and many more details from the athlete's perspective, which could have added much more depth to our project.

In the future, we hope to have better datasets that will let us better display more stuff on our App.

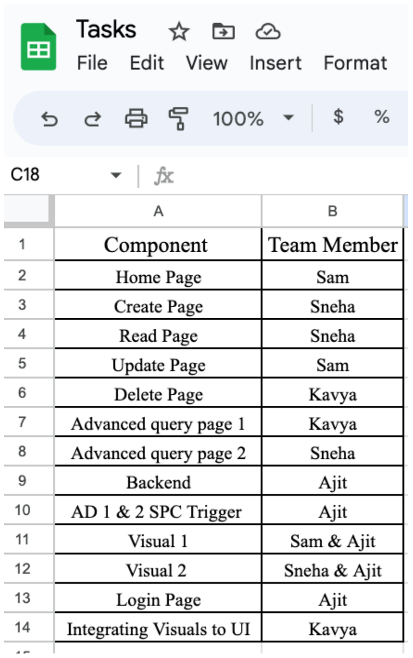
Additionally, we would like to have another backend database like Mongo or similar that can let us add more images which we can add for countries, athletes, etc., as SQL is limited when it comes to adding images. It is not straightforward if not done correctly very prior, and even then, we have to upload all of those images onto GCP and map them effectively, which is a hassle that Mongo could have bypassed.

---

10. Describe the final division of labor and how well you managed teamwork.
We maintained a spreadsheet containing all the components we envisioned in the proposal stages, and we added or removed functionality based on comforts and strengths.

In terms of the division of labor, we initially sat down in Grainger to review everyone's understanding of concepts since many of the concepts were new to many people on the team. They all sat together, leaned on each other to learn these concepts, and quickly ramped up. Parallelly everyone was working on their individual components and frequently discussed over iMessage, Zooms, or in person accordingly. Everyone actively participated in all of the above as needed.

The final spreadsheet is as follows:



Component	Team Member
Home Page	Sam
Create Page	Sneha
Read Page	Sneha
Update Page	Sam
Delete Page	Kavya
Advanced query page 1	Kavya
Advanced query page 2	Sneha
Backend	Ajit
AD 1 & 2 SPC Trigger	Ajit
Visual 1	Sam & Ajit
Visual 2	Sneha & Ajit
Login Page	Ajit
Integrating Visuals to UI	Kavya

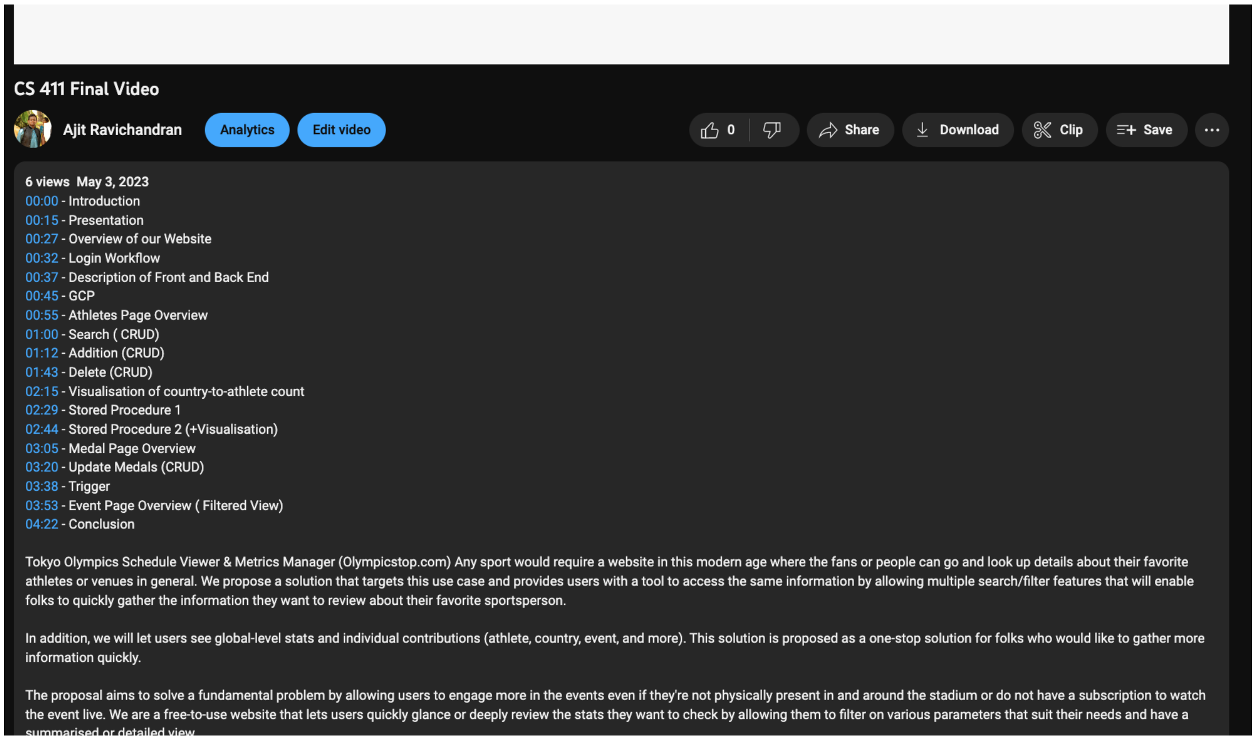
Sam took care of creating the home and update page and contributed to the creation of the first visual.
Sneha took care of creating the create page, search page, and the second advanced query page and contributed to the creation of the second visual.

Kavya created the delete page, the first advanced query page, and integrated the visual into the UI.
Ajit took care of the backend alongside all the APIs needed, The Stored Procedures & Triggers for the Advanced Queries, assisting with both the visuals and creating the login page.

While the components were developed in tandem, everyone could push and pull code directly from the server using the stored GitHub SSH keys setup in the server. This ensured that everyone could push and pull code as needed and quickly integrate the code. We coordinated when someone pushed their code, and everyone could work seamlessly without having to fix merge commits that could have emerged from working this way.

Project Demo Video

4:41 seconds - The video comprehensively outlines all the workflows and visualizations and explains the tech stack. The video contains chapter marking points covering all the key components that have to be covered.



CS 411 Final Video

6 views May 3, 2023
00:00 - Introduction
00:15 - Presentation
00:27 - Overview of our Website
00:32 - Login Workflow
00:37 - Description of Front and Back End
00:45 - GCP
00:55 - Athletes Page Overview
01:00 - Search (CRUD)
01:12 - Addition (CRUD)
01:43 - Delete (CRUD)
02:15 - Visualisation of country-to-athlete count
02:29 - Stored Procedure 1
02:44 - Stored Procedure 2 (+Visualisation)
03:05 - Medal Page Overview
03:20 - Update Medals (CRUD)
03:38 - Trigger
03:53 - Event Page Overview (Filtered View)
04:22 - Conclusion

Tokyo Olympics Schedule Viewer & Metrics Manager (Olympicstop.com) Any sport would require a website in this modern age where the fans or people can go and look up details about their favorite athletes or venues in general. We propose a solution that targets this use case and provides users with a tool to access the same information by allowing multiple search/filter features that will enable folks to quickly gather the information they want to review about their favorite sportsperson.

In addition, we will let users see global-level stats and individual contributions (athlete, country, event, and more). This solution is proposed as a one-stop solution for folks who would like to gather more information quickly.

The proposal aims to solve a fundamental problem by allowing users to engage more in the events even if they're not physically present in and around the stadium or do not have a subscription to watch the event live. We are a free-to-use website that lets users quickly glance or deeply review the stats they want to check by allowing them to filter on various parameters that suit their needs and have a summarised or detailed view

Link: https://youtu.be/aPps5rGYSyQ