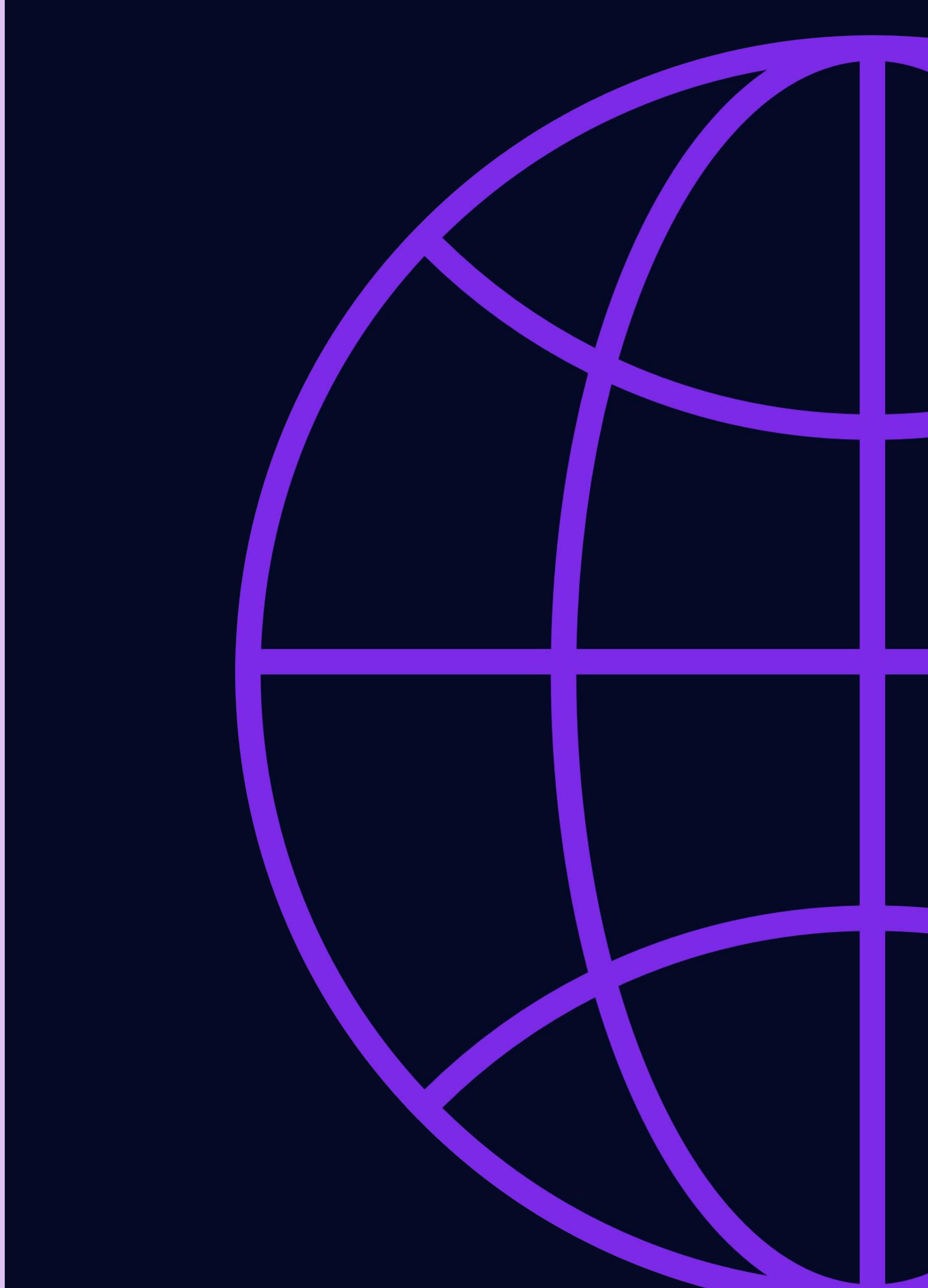
Kai Barker 241065

DV200 Semester 2 Summative Presentation

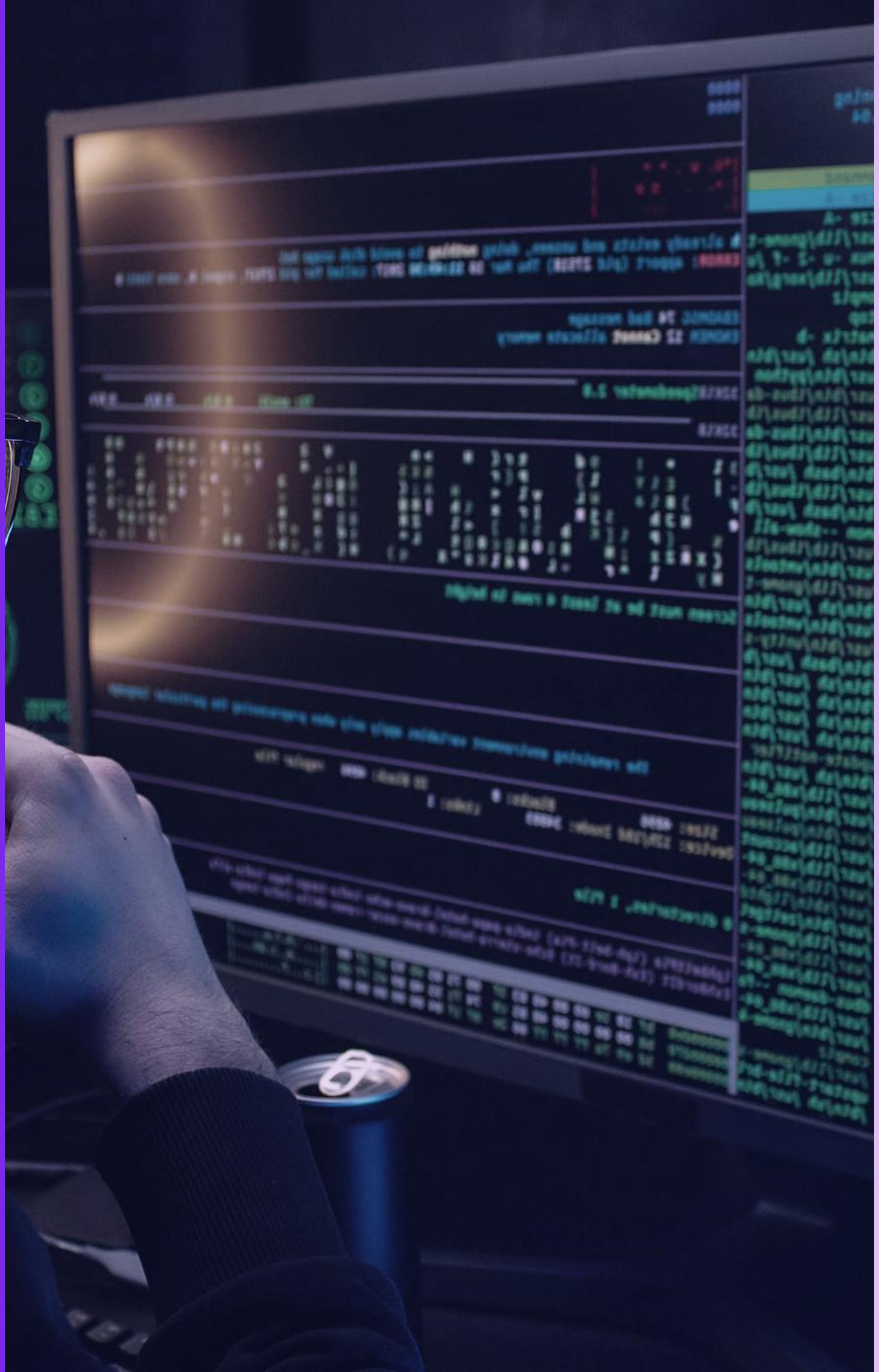
ReadyUP

What’s on the Agenda?

- Introduction
- Live Application Demonstration
- Database And Backend Explanation
- Deployment, SEO, Optimisation
- nReflection and Conclusion

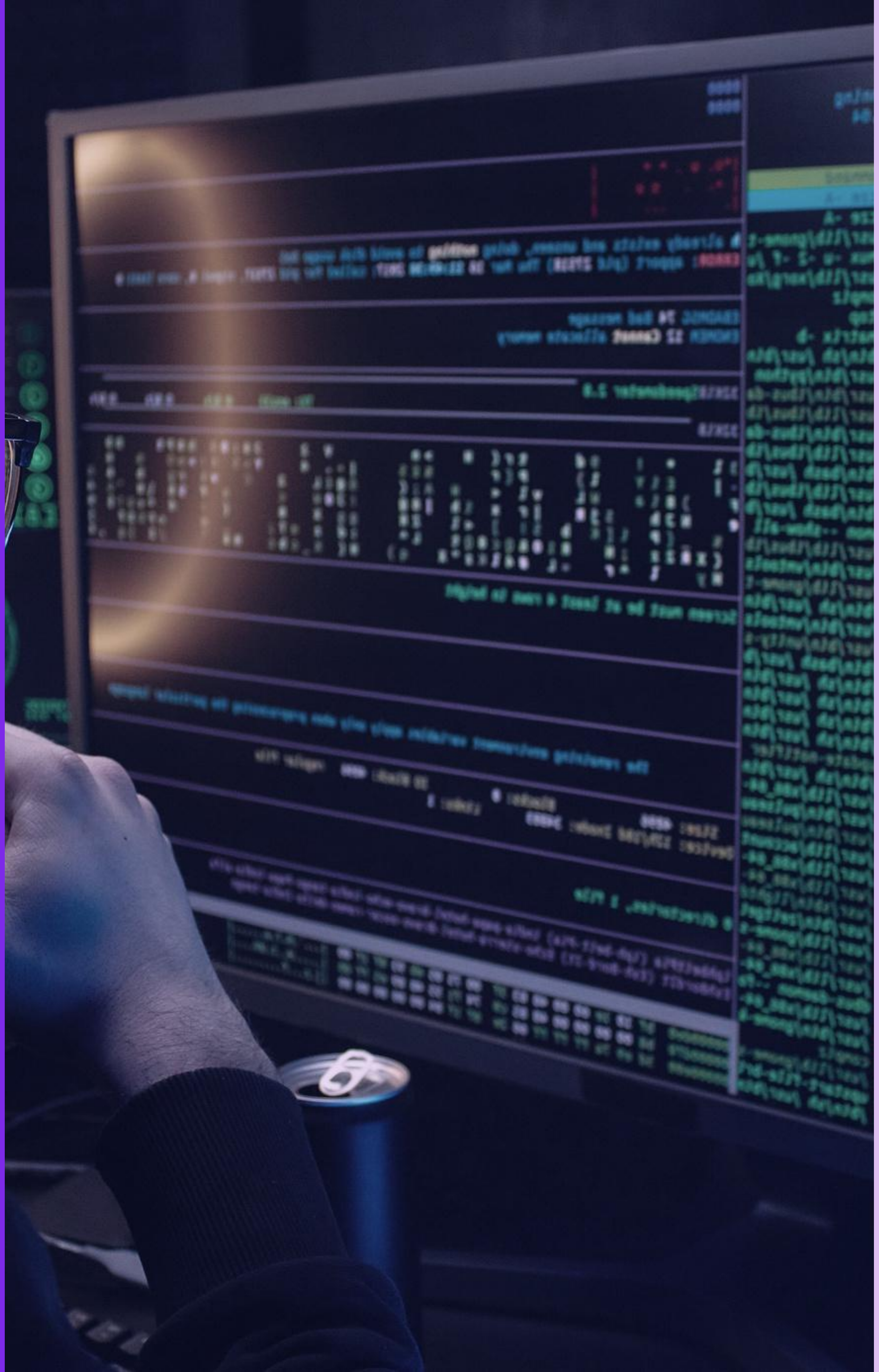


Hello there!



Hi, I'm Kai, and I'll be taking you through my summative react application. ReadyUP

ReadyUP



ReadyUP is a full stack web application designed to solve a very common issue in online gaming, your teammates.

Random matchmaking can often be toxic, frustrating, and unreliable. Some teammates troll, some are toxic, and others are just really bad

What problem requires ReadyUP

Say you want to hit masters this season, or finally escape gold in ranked, your random teammates could throw, give up, or can just be plain bad.
Say you want to just play casually, have a couple laughs and just vibe in the evening, but then you get a random who is toxic, overly competitive and judges your every mistake. Not fun.

This can be improved by finding a group of similar players that want the same experience that you do and are at a similar level. ReadyUP aims to help match players with similar interests and objectives in mind to improve their gaming experience. Current LFG solutions are fragmented over community discord servers or console-only services which makes it inconvenient for many users to use such features

So how does ReadyUP solve that?

ReadyUP is a looking for group (LFG) platform and serves as a hub for gamers to connect with others and find teammates that fit their goal and vibe. Users can make or join posts and link up with others on their dedicated platform

You want to climb to masters? Make a post saying you'd like a competitive experience and wait for other, similar players to join you.

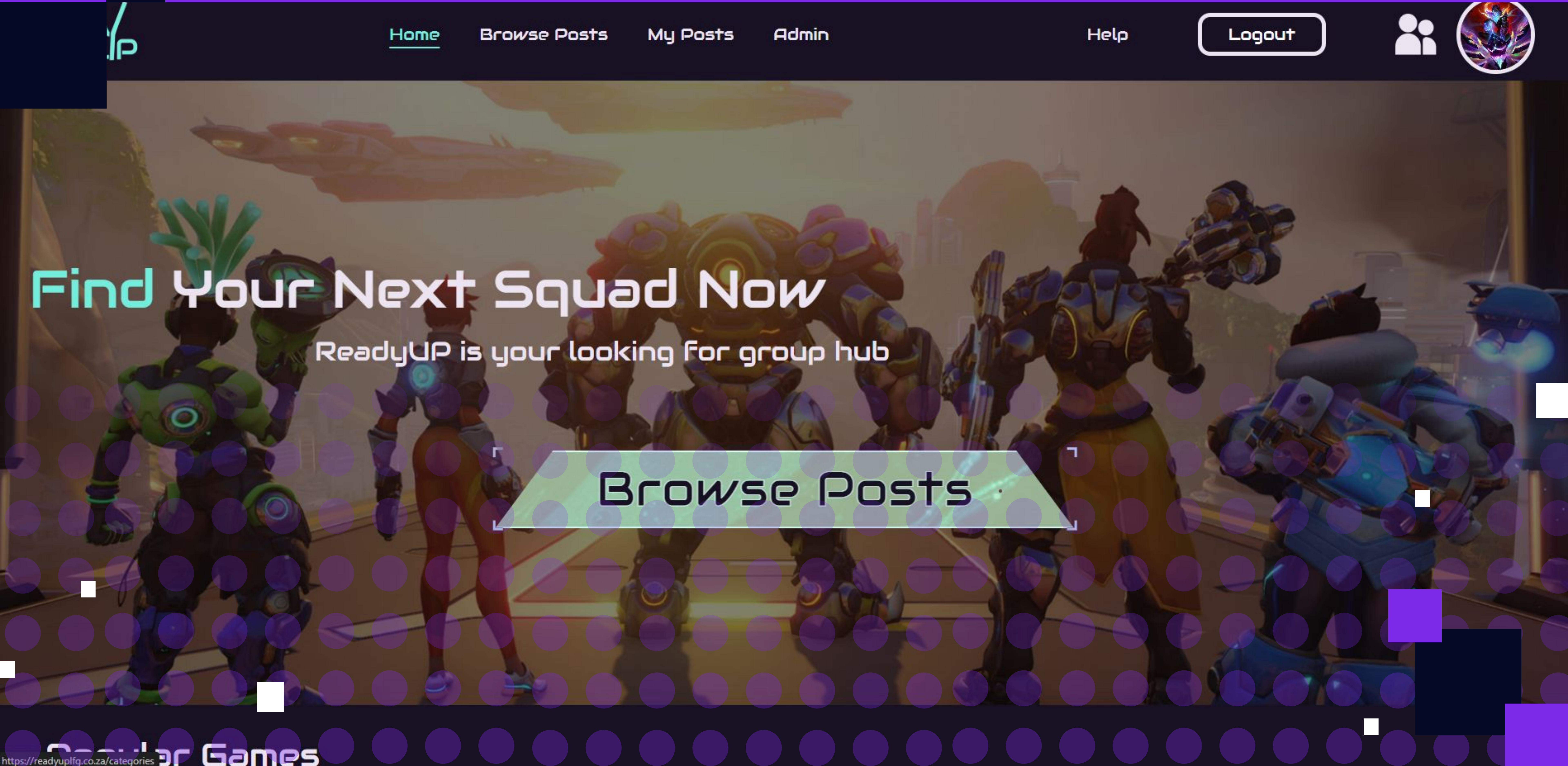
Want to try out a new game just to sit back and have fun after a long day of working on a website? Just find or make a post, then queue up and have fun.

Target market

My target audience is any gamer who plays online multiplayer titles, from the casual player who just wants to vibe and avoid a toxic lobby, to the competitive player looking for a serious, mic-required team to climb the ranks. It's built for anyone who wants to make their online gaming more social and more fun.

Whether the user plays on pc or console, there is support for all major platforms on the user’s profile

Live Application Demonstration



Find Your Next Squad Now

ReadyUP is your looking for group hub

Home Page

Here is the landing page for ReadyUP. Users are greeted with a hero section prompting for them to browse through the posts.
Below that are the top categories with the highest number of posts, followed by a brief explanation of how to use the service
As a new user, you can immediately see the core pages and browse all the game categories. This is the 'Read' part of CRUD—we're reading all categories from the database. Let's click on the current top group.

Browse Posts Page

This page is where a lot of the magic happens. At the top left the user is informed of the game category they are viewing posts for. Below that we have the Tag Filterer, Sort By Dropdown, and Search Bar. Each of these will change what posts we see, or the order we see them in. Each of these demonstrate read CRUD functionality. Signed in users will see a button to create a post, we will get to that in a bit. Below this you will see the posts users have made, to avoid long loading times these are paginated and displayed by a few at a time. Signed in users can join these posts, so lets sign in

Login Page

When we click the login button at the top right, it will take us to the login page. Here we see text fields for email and password, as well as a CheatCode section which is a secondary, gaming themed, authentication method.

Assume we do not have an account currently so lets click on the register button

Register Page

On the login page we can create a new account with our username, email, password and cheatcode. All fields are required to be filled for an account to be created.
This is one of the sections that fulfils the Create CRUD operation.

Lets create a new account named DemoUsar, purposefully misspelt, with the email DemoUser@gmail.com, password will be D3m0, cheatcode will be up up down down left right
We should see a popup saying that we are registered

Back to the Login

Now back at the login, i will enter some random incorrect details to show the login not working. I will also not fill in the cheat code.

Some feedback i recieved from external user testing was that the cheat code was easy to just skip past, and they were unsure as to what was being added or if it was actually doing something. So i added a message below it to let users know whats been pressed.

Upon successful login, a JWT is created and stored in local storage that parses the users information and access role to the website

Profile Page

Here we can see our username, bio, friends, profile picture, and our preferred platform's user id. Currently the page is mostly empty as nothing has been added, lets change that

Click on the edit profile button, where we can see a whole bunch of information related to the user in the text box's and text fields. Remember that spelling mistake? yeah lets change that.

I will fix the username and add my profile, then come back and add the bio. Users can change any of these fields. It doesn't require all to be changed. I click save changes and now the username should be changed, alongside the profile picture

Profile Page

Lets now go back and add our bio, along with our preferred platform

Click edit again and add in these fields

Now our profile is complete, and we can click on the platforms icon to copy the id we just entered.

Unfortunately, we will need to refresh the page for the nav bar's image to take affect, but everywhere else our profile will be set. This fulfils the Update CRUD operation

With all this now set up. Lets make a post. Click on Browse Posts in the nav

Categories page

Instead of taking us straight to posts, we will need to decide which game's posts we want to look at. Here we can filter categories by alphabetical and popularity, as well as search for one we want. At the bottom of the page you will see pagination. Filtering and pagination has been implemented through the backend.

I'm in the mood for a battle royale of cartoony nature, so lets click on Fortnite.

Browse Posts Page again

Here we can see all the posts currently made for fortnite. Now since we are logged in, we can see the create post button. Lets make a post.

I will fill in these options, give it a title of duos, a description related to the game, a time of 2 minutes from now, look for only 1 extra player, and add a BR tag and a custom tag of my own that does not exist yet. Then click post

We will see a little popup saying success, and we will now see our post

Our Post

Lets click on that post we just made, here you can see more information on the post asking if youd like to join. This page will tell you every player that is currently on that post.

Now if we click on join post on our own post, surely that shouldn't work right? and you are correct! it doesn't work! as the popup says, you cant join your own post silly.

Lets join someone elses post now, click on a different user’s post and click join. This will inform us we have joined that post, and our name will appear as well.

Now lets click on My Posts.

My Posts

Here we can see every post we belong to, including the ones we have just made and joined. This page is essentially just a central place to view the posts you've joined. Here we can also click on the post and click on one of the user's profiles.

We cannot join a post here and unfortunately no silly message when you try join a post you've already joined.

Lets click on the user rhirhi

Public Profile

Here we can see the entire profile page of a different user. Lets click on the add friend button at the bottom right. This is one way to send another user a friend request.

Now as we continue stalking this profile, lets click on the friends tab and view rhirhi’s friends.

Here we can see a user named Synergyy (Thats me!)

Lets add this Synergyy guy as a friend, using the friends menu.

Click on the friends menu at the top right.

Friends List

The friends menu is a sidebar, with a friends list section, and an add friends section, indicated by the icons.
As you can see, we have no friends. lets go send Synergyy a friend request. In the add a friend section, type Synergyy. This will give us a search result for the user

When we hover over them, we see a + sign pop up, lets click that and send the friend request. Cool now lets do it again. Friendships must be unique, so any user pair with an outstanding friend request or are already friends will not be able to send another

Logout and change accounts

Now lets log out and change accounts to the Synergyy account

Logging out clears the JWT

Now that i have swapped to the Synergyy account we can see a new admin page
This is because Synergyy is my master admin account which has admin access for creating and deleting posts.
But first lets check that friend request

Friend Request

If you click on the friend icon again, and navigate to the add friends tab, you should see our friend request from the demoUser account we created, among others. We can accept or reject this friend request. I will accept demoUser’s friend request and reject this other one i have in my requests section. Rejecting a friend request fulfils a Delete CRUD operation

Now we will see demo user in my friends list, and the other account not here.

And finally, lets take a look at the admin panel

Admin Page

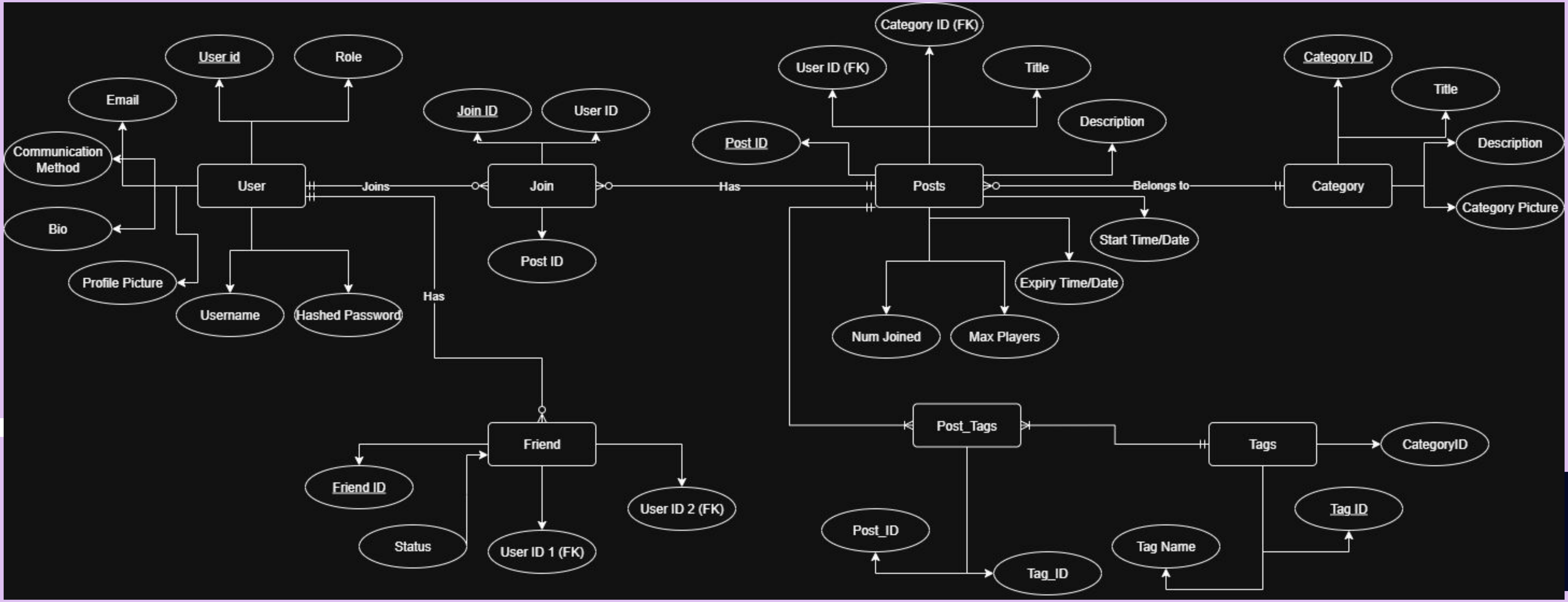
The admin page has spaces for deleting categories, posts, and adding categories.
These related backend routes are protected via the logged in users role.
Essentially, if a regular user somehow managed to run this query, it would fail as it will return 401 Unauthorized
I will demonstrate creating a category, and deleting categories and posts as well.

Database and Backend Explanation

Now let's look at the backend. Below is the ERD for the ReadyUP database.

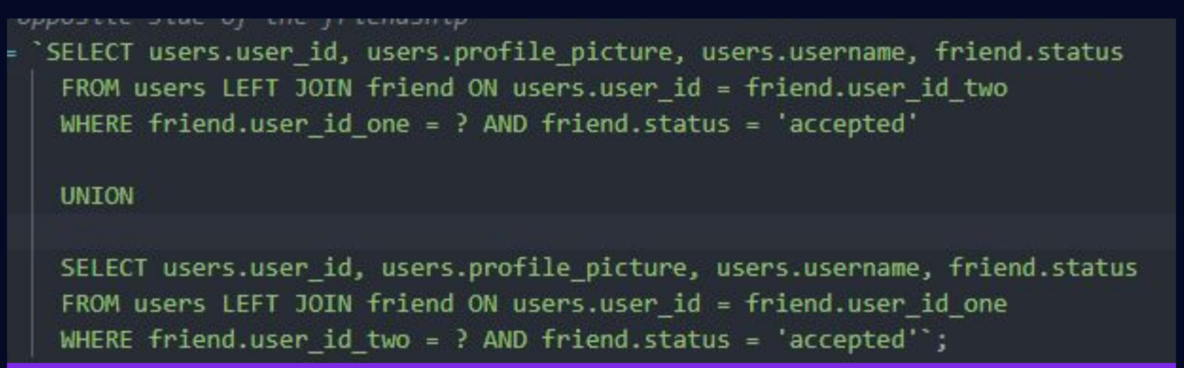
- User is the user themselves, Friend defines a friendship between 2 users, the sender is always UserID1. Category is games category that users post in, posts are the main lfg posts that are made. Tags are present within an individual category and get linked to posts.

- I used normalization to handle many-to-many relationships:
  - post_tags links posts and tags.
  - join_post links users and posts (to track who has joined).
  - friend links users to other users.



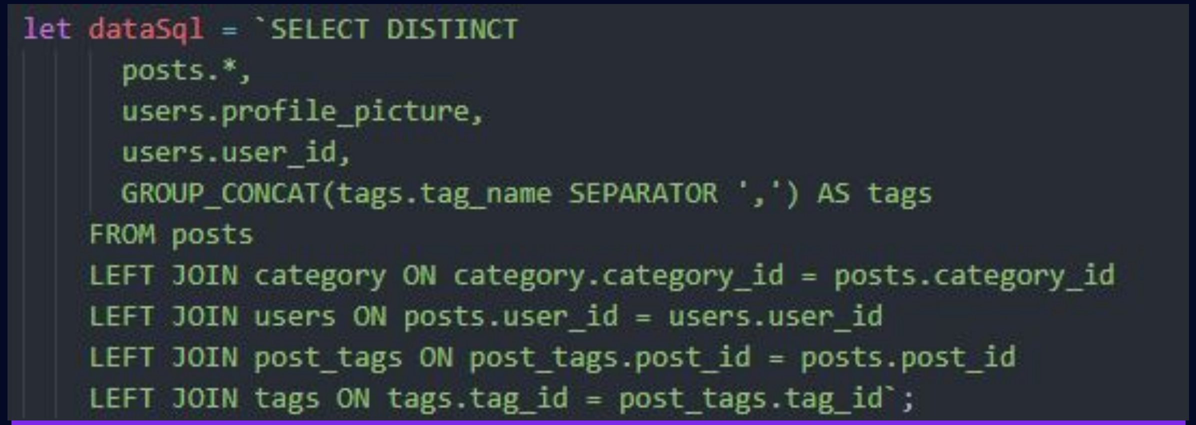
Advanced SQL Queries

JOINs: I use JOINs extensively. For example, the 'Browse Posts' page uses a single query that joins posts, users, post_tags, and tags to get all the post data, the author's profile picture, and all the post's tags in one efficient database call.



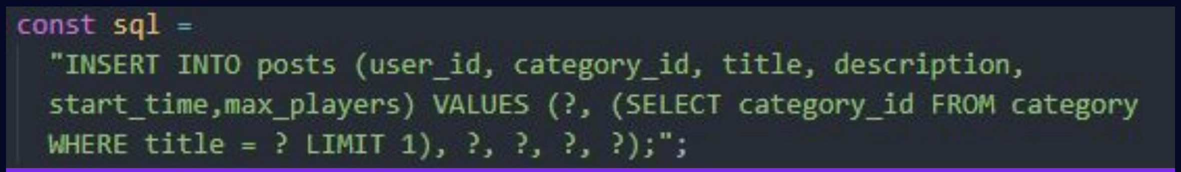
```
opposite side of the friendship
= `SELECT users.user_id, users.profile_picture, users.username, friend.status
   FROM users LEFT JOIN friend ON users.user_id = friend.user_id_two
   WHERE friend.user_id_one = ? AND friend.status = 'accepted'

   UNION

   SELECT users.user_id, users.profile_picture, users.username, friend.status
   FROM users LEFT JOIN friend ON users.user_id = friend.user_id_one
   WHERE friend.user_id_two = ? AND friend.status = 'accepted'`;
```

- UNION: My 'Friends List' query is a great example. It uses UNION to combine two SELECT statements: one for friends where the user is 'User 1' and one for friends where they are 'User 2'. This gets the full friends list regardless of who sent the request.



```
let dataSql = `SELECT DISTINCT
      posts.*,
      users.profile_picture,
      users.user_id,
      GROUP_CONCAT(tags.tag_name SEPARATOR ',') AS tags
    FROM posts
    LEFT JOIN category ON category.category_id = posts.category_id
    LEFT JOIN users ON posts.user_id = users.user_id
    LEFT JOIN post_tags ON post_tags.post_id = posts.post_id
    LEFT JOIN tags ON tags.tag_id = post_tags.tag_id`;
```

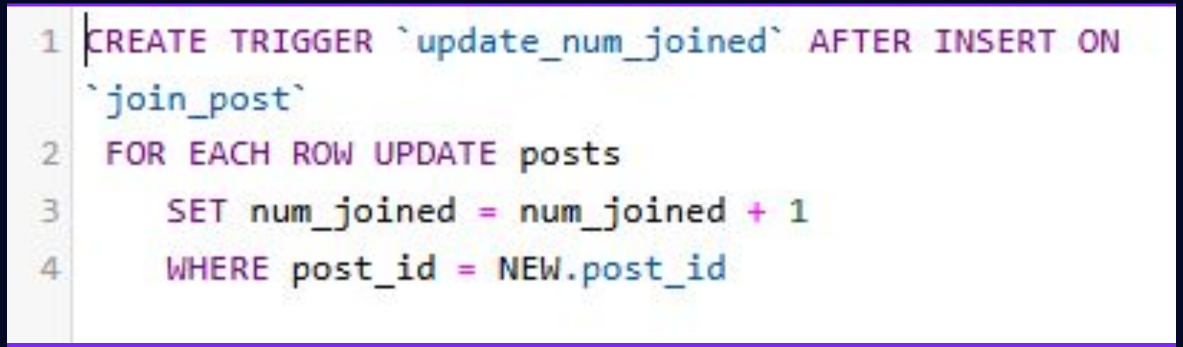
- GROUP BY & GROUP_CONCAT: When fetching a post, I use GROUP_CONCAT to bundle all its tags into a single, comma-separated string, which is very easy for React to parse and display.



```
const sql =
  "INSERT INTO posts (user_id, category_id, title, description,
  start_time,max_players) VALUES (?, (SELECT category_id FROM category
  WHERE title = ? LIMIT 1), ?, ?, ?, ?);";
```

- Subqueries: When a user creates a post, they provide the *name* of the category (e.g., "Fortnite"). My SQL query uses a subquery to insert the correct foreign key.

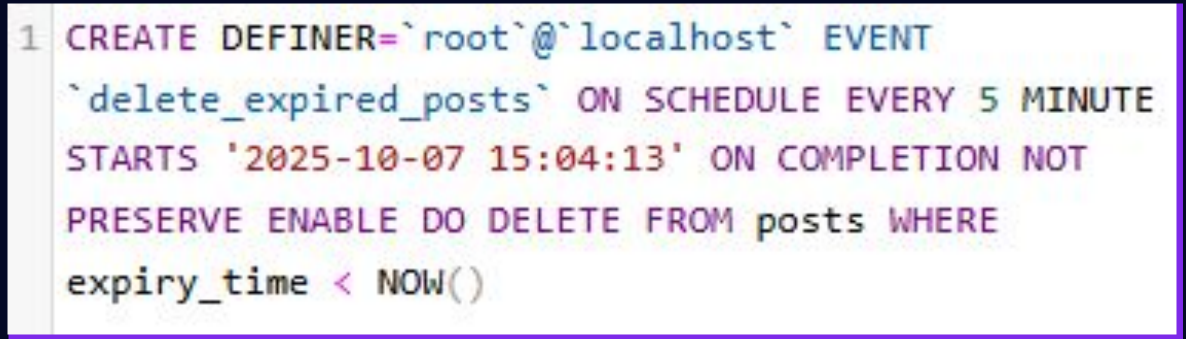
Advanced SQL Queries



```
CREATE TRIGGER `update_num_joined` AFTER INSERT ON
`join_post`
 FOR EACH ROW UPDATE posts
    SET num_joined = num_joined + 1
    WHERE post_id = NEW.post_id
```

- Triggers

I used triggers to update the number of joined players field when a user joins a post. I also have a trigger that sets the expiry time to be approx 2 hours after start time.



```
CREATE DEFINER=`root`@`localhost` EVENT
`delete_expired_posts` ON SCHEDULE EVERY 5 MINUTE
STARTS '2025-10-07 15:04:13' ON COMPLETION NOT
PRESERVE ENABLE DO DELETE FROM posts WHERE
expiry_time < NOW()
```

- Events

I used an event to automatically delete posts that are past their expiry time. I have a trigger that sets the expiry time to be approx 2 hours after start time.

Problems and Existing Alternatives

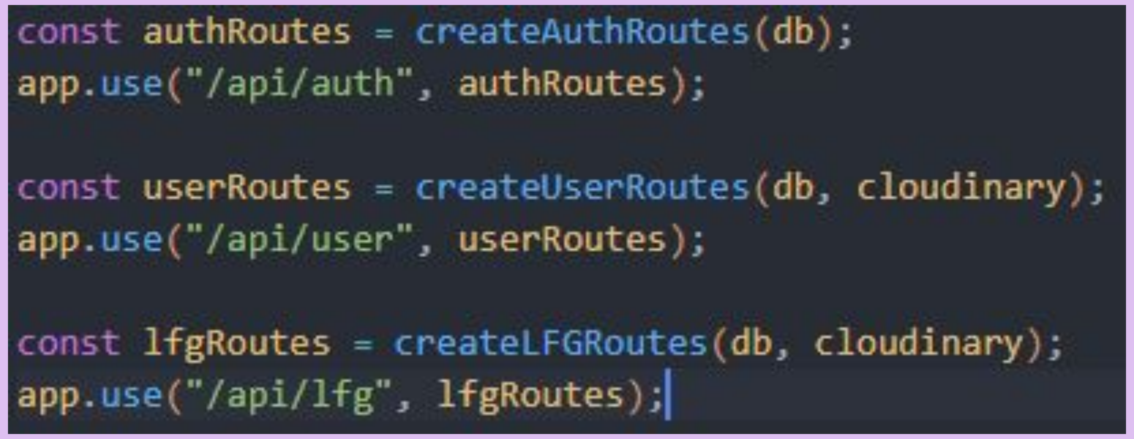
Write the problem here and elaborate on it.	Discuss an existing alternative people use to address the problem.
Write the problem here and elaborate on it.	Discuss an existing alternative people use to address the problem.
Write the problem here and elaborate on it.	Discuss an existing alternative people use to address the problem.

Frontend & Backend Integration

My frontend links to my backend through a central api that helps define all the routing to my backend. This one file utilises an environment variable stored on vercel to link to my backend securely. Additionally, it attaches the repetitive parts of my url automatically, resulting in a cleaner codebase. It uses an axios instance to make the HTTP requests to my endpoints

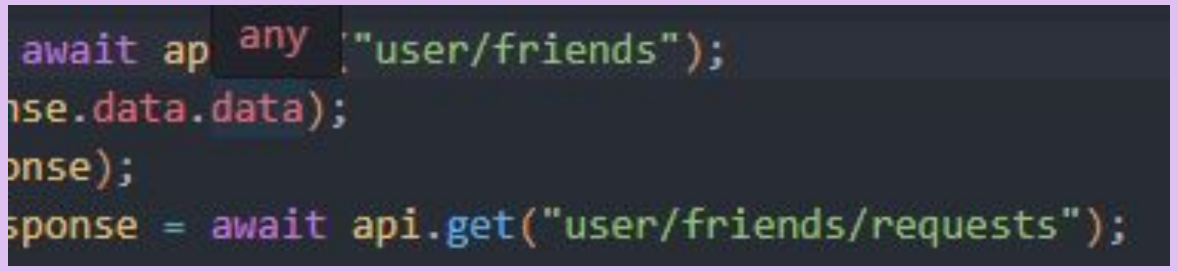
My backend routes are stored in separate files, Auth, User, and LFG. Each of these hold their respective backend routes only and are accessed by /auth, /user, or /lfg respectively. Authentication is done via JWT to securely retrieve the users details

Dynamic routes are largely handled with params e.g. /lfg/:postid/joined-users



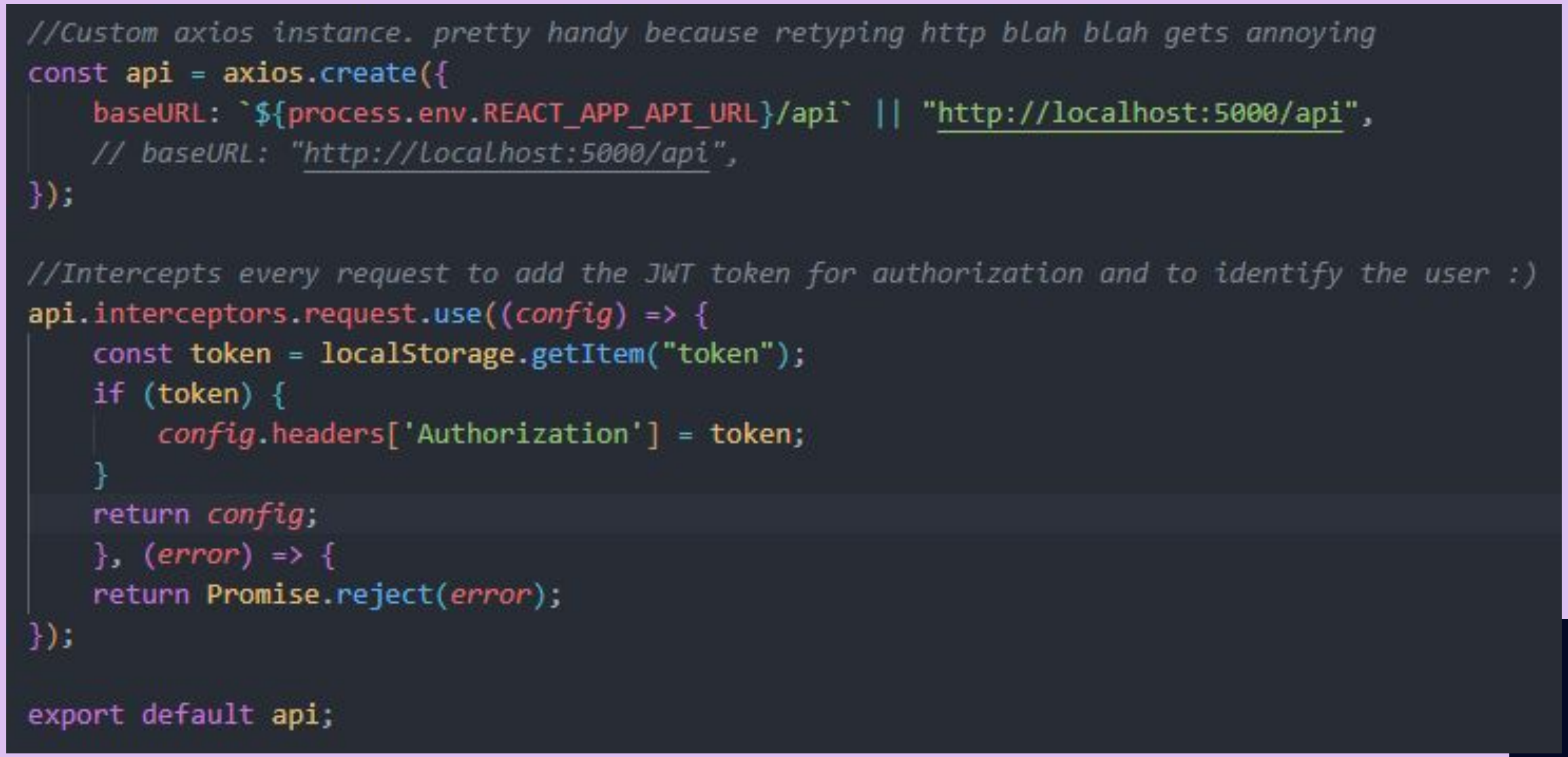
```
const authRoutes = createAuthRoutes(db);
app.use("/api/auth", authRoutes);

const userRoutes = createUserRoutes(db, cloudinary);
app.use("/api/user", userRoutes);

const lfgRoutes = createLFGRoutes(db, cloudinary);
app.use("/api/lfg", lfgRoutes);
```

Backend route definition



```
 await ap any "user/friends");
nse.data.data);
onse);
sponse = await api.get("user/friends/requests");
```

Example API calls



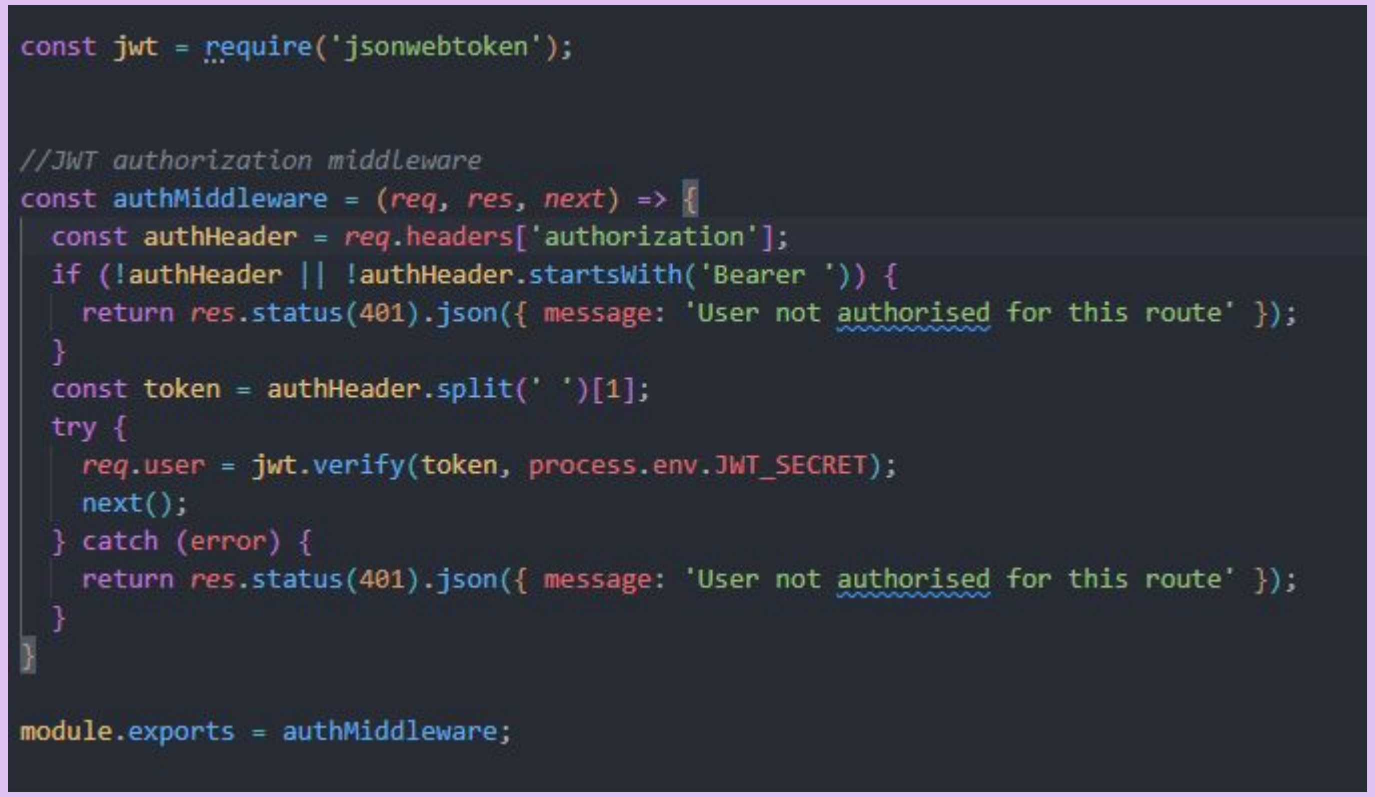
```
//Custom axios instance. pretty handy because retyping http blah blah gets annoying
const api = axios.create({
    baseURL: `${process.env.REACT_APP_API_URL}/api` || "http://localhost:5000/api",
    // baseURL: "http://localhost:5000/api",
});

//Intercepts every request to add the JWT token for authorization and to identify the user :)
api.interceptors.request.use((config) => {
    const token = localStorage.getItem("token");
    if (token) {
        config.headers['Authorization'] = token;
    }
    return config;
    }, (error) => {
    return Promise.reject(error);
});

export default api;
```

Frontend api.js

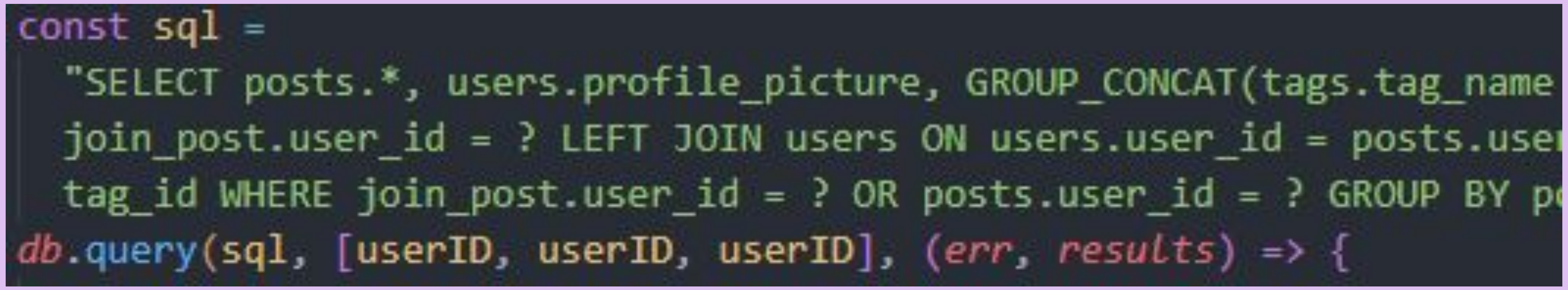
Security Measures

Finally, security was a top priority and always played into my mind when creating a route.

- SQL Injection: All database queries use the mysql2 library's prepared statements (the ? placeholders). This automatically sanitizes all user input and makes SQL injection impossible.
- Password Hashing: I am not storing passwords in plain text. All user passwords are hashed and salted using bcrypt before being saved in the database.
- Authentication: Protected routes, like editing your profile, are secured by my custom authMiddleware. This middleware checks for a valid JWT in the request header to verify who the user is and what their role is.
- Environment Variables: All sensitive data, my database password, my JWT secret, and my Cloudinary API keys, are stored in a .env file. This file is ignored by Git, so my secrets are never exposed in the code repository."



```
const jwt = require('jsonwebtoken');


//JWT authorization middleware
const authMiddleware = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'User not authorised for this route' });
  }
  const token = authHeader.split(' ')[1];
  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET);
    next();
  } catch (error) {
    return res.status(401).json({ message: 'User not authorised for this route' });
  }
}

module.exports = authMiddleware;
```



```
const sql =
  "SELECT posts.*, users.profile_picture, GROUP_CONCAT(tags.tag_name
  join_post.user_id = ? LEFT JOIN users ON users.user_id = posts.use
  tag_id WHERE join_post.user_id = ? OR posts.user_id = ? GROUP BY p
db.query(sql, [userID, userID, userID], (err, results) => {
```

Deployment and Optimisation

Deployment Cloud Platforms

For deployment, I used a modern, decoupled CI/CD (Continuous Integration/Continuous Deployment) pipeline. This made the development process more streamlined, as all i had to do for each redeployment was push to github

Frontend

I chose Vercel because it is purpose-built for modern frontend frameworks like React. Its main advantage is the seamless GitHub integration. Every push to my main branch automatically triggers a new build and deployment with zero downtime.

Vercel

Backend

I chose Render as a modern alternative to Heroku for my application server. Like Vercel, it offers a direct GitHub integration that automates my backend deployments. It's ideal for a Node.js server because it simplifies all the infrastructure, handles the npm processes, and makes it very easy to securely manage my secret environment variables. Furthermore it has detailed logs that help me identify issues during development

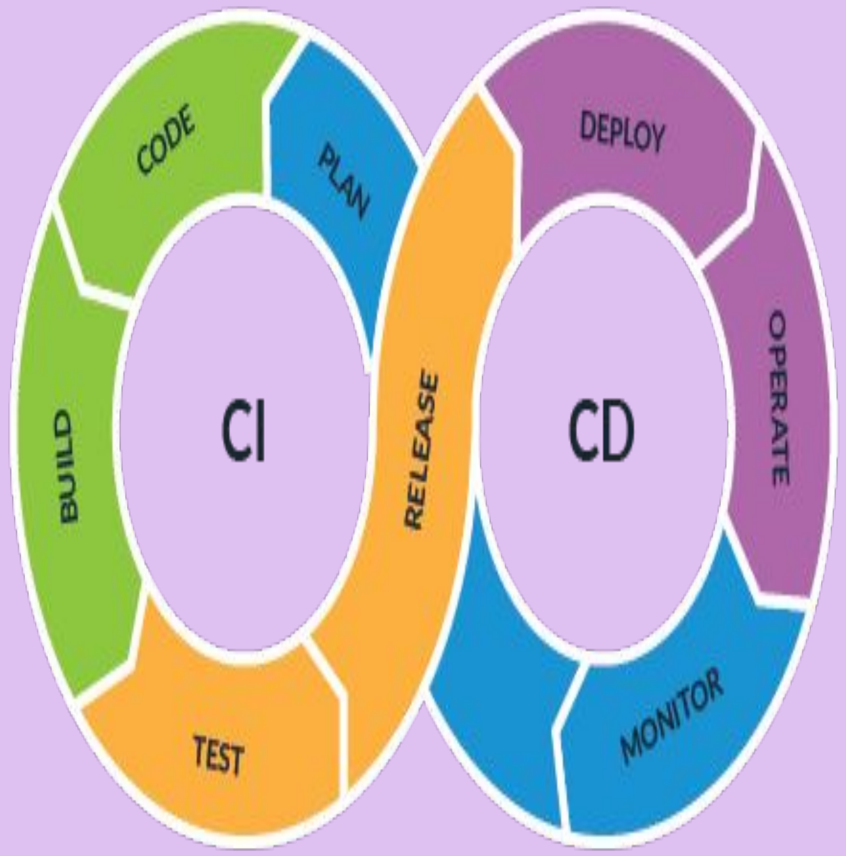
Render

Database

For my database, I chose AWS RDS. I picked AWS as I wanted high reliability and security, which led me to picking from the best in industry. Specifically, RDS keeps my database in a private network, safe from the public internet. It's also highly scalable and it's surprisingly cost-effective and easy to set up. Another option I was deciding on was between Render and Google Cloud. Render turns off with inactivity which deterred me, and I am more familiar with AWS than with Google cloud

amazon RDS

Deployment Pipeline



Trigger

The pipeline stars when i push my code to github, Render and Vercel are instantly notified of this push, and begin rebuilding their files

CI/CD

Vercel pulls the /frontend, Render pulls the /backend. Once the builds succeed my application is re-deployed without downtime

Env Variables

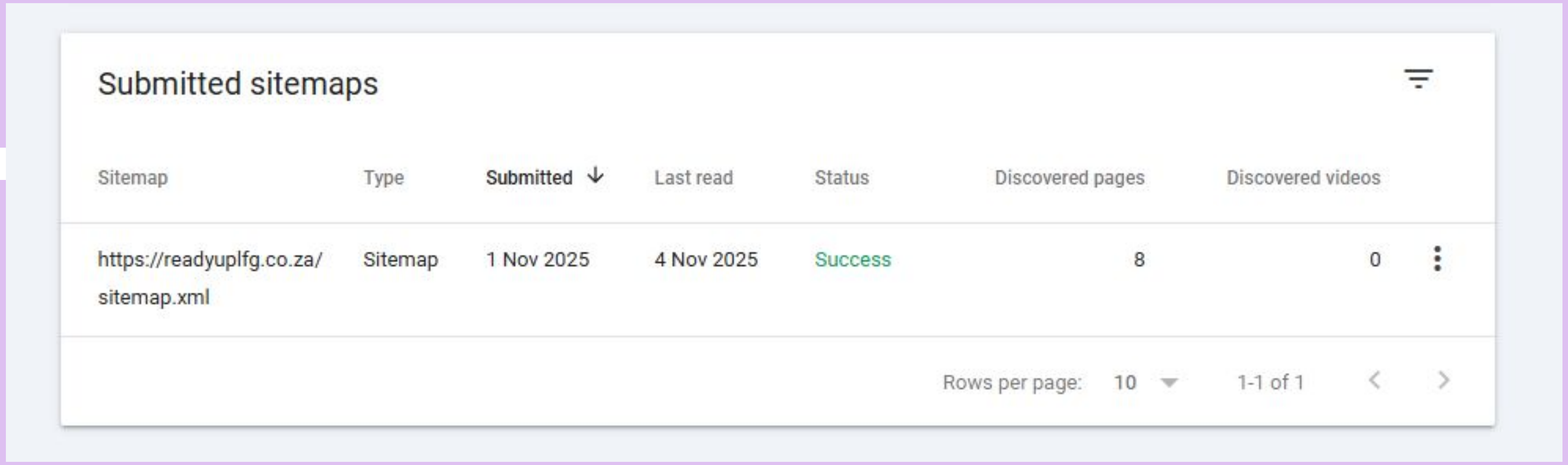
Vercel is connected with an env variable to find the render backend, Render is connected with many environment variables to connect to AWS, cloudinary and other secrets

Search Engine Optimisation

For my react-app, getting SEO to work correctly was a major challenge due to it being a SPA.

Meta Information (SEO): Standard React apps have one title for all pages. To fix this, I built a custom React hook called useSeo. This hook is used on every page, like UserProfile or Browse Posts, to dynamically update the document.title and meta tag. This gives every page a unique identity for Google's crawlers.

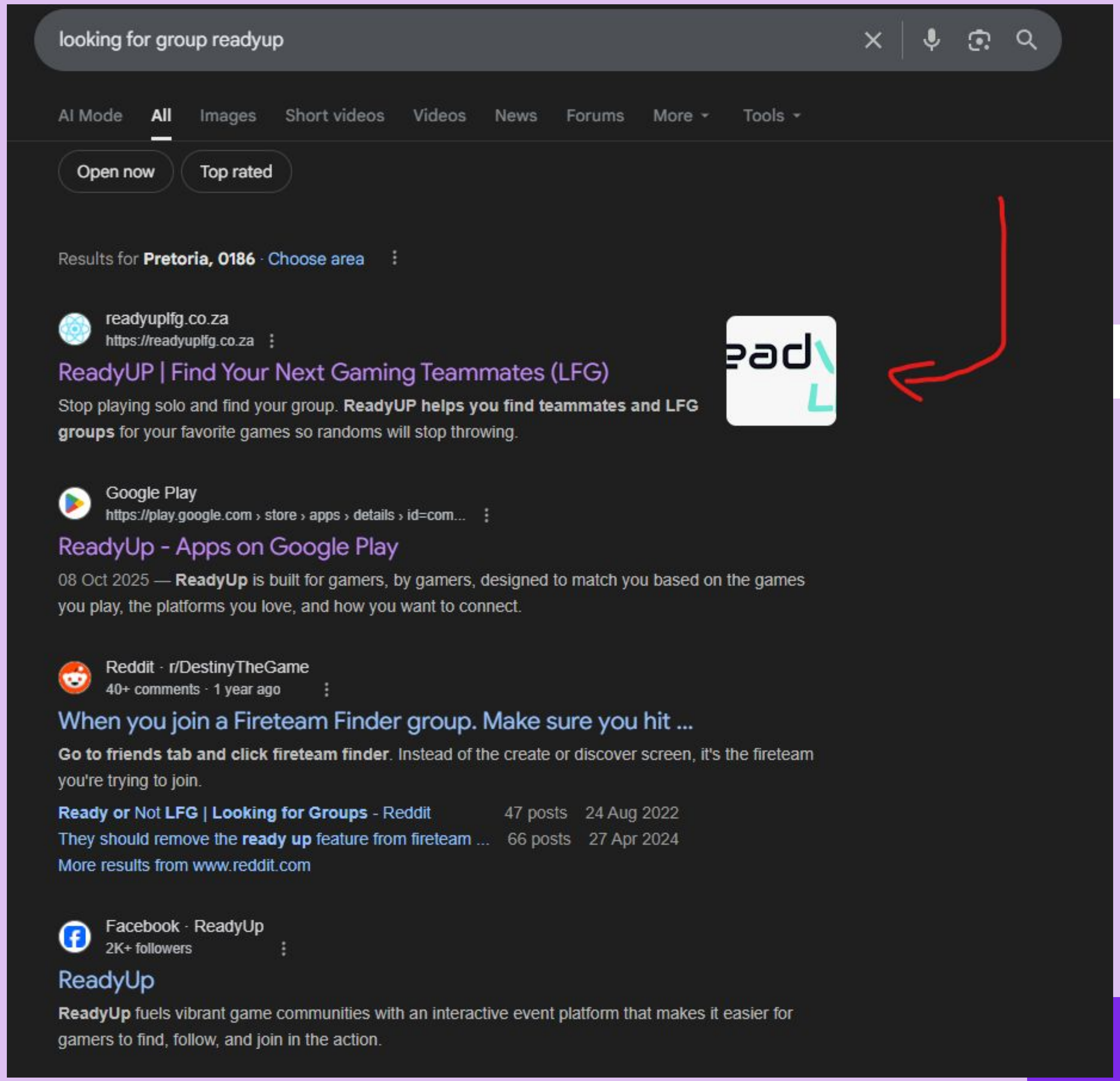
Google Indexing: I created a dynamic sitemap.xml route on my backend. This file was submitted to Google Search Console, which tells Google exactly how to find and index all my site's pages, including user profiles and categories.



Submitted sitemaps

Sitemap	Type	Submitted ↓	Last read	Status	Discovered pages	Discovered videos
https://readyuplfg.co.za/sitemap.xml	Sitemap	1 Nov 2025	4 Nov 2025	Success	8	0

Rows per page: 10 1-1 of 1



looking for group readyup

Results for **Pretoria, 0186** · Choose area

readyuplfg.co.za
https://readyuplfg.co.za
ReadyUP | Find Your Next Gaming Teammates (LFG)
Stop playing solo and find your group. **ReadyUP helps you find teammates and LFG groups** for your favorite games so randoms will stop throwing.

Google Play
https://play.google.com › store › apps › details › id=com...
ReadyUp - Apps on Google Play
08 Oct 2025 — **ReadyUp** is built for gamers, by gamers, designed to match you based on the games you play, the platforms you love, and how you want to connect.

Reddit · r/DestinyTheGame
40+ comments · 1 year ago
When you join a Fireteam Finder group. Make sure you hit ...
Go to friends tab and click fireteam finder. Instead of the create or discover screen, it's the fireteam you're trying to join.

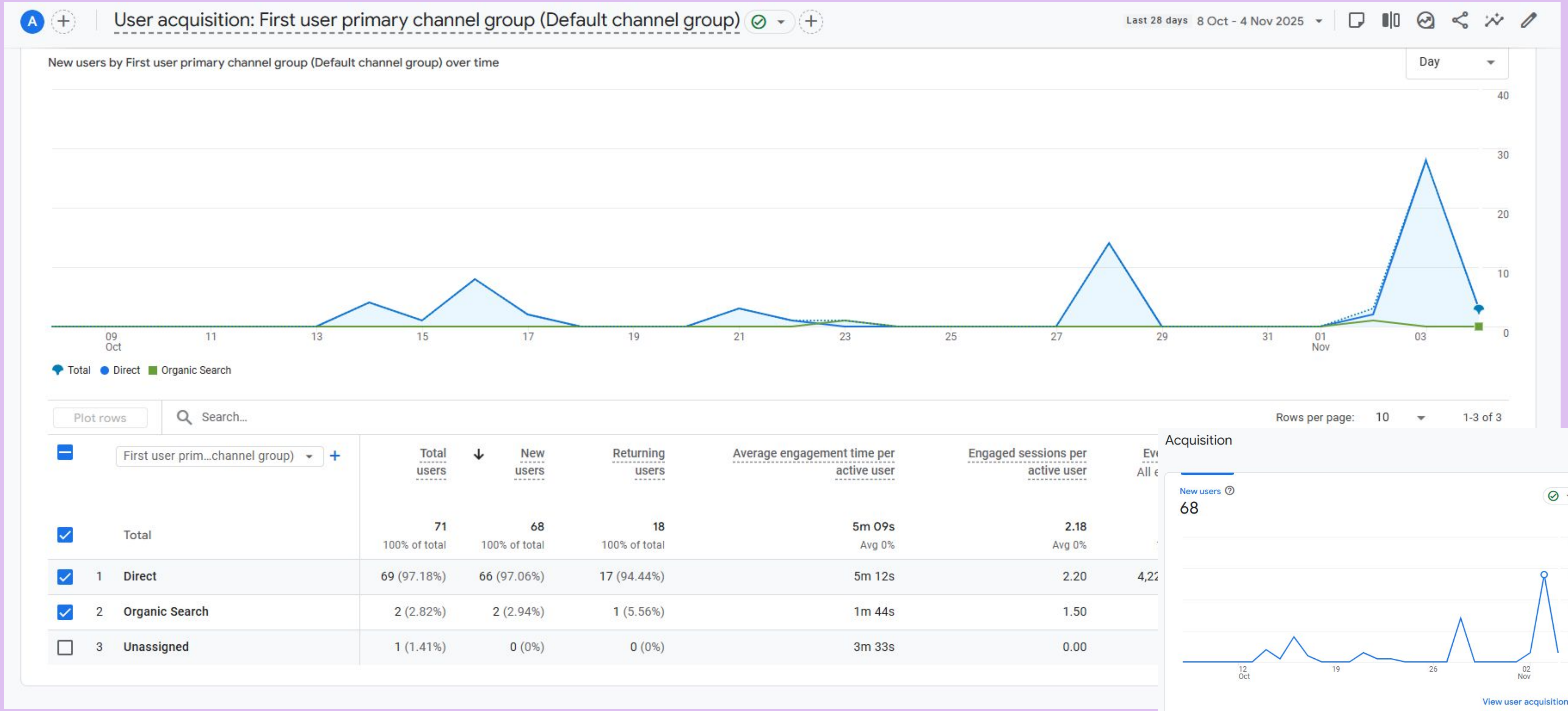
Ready or Not **LFG | Looking for Groups** - Reddit	47 posts	24 Aug 2022
They should remove the **ready up** feature from fireteam ...	66 posts	27 Apr 2024

More results from www.reddit.com

Facebook · ReadyUp
2K+ followers
ReadyUp
ReadyUp fuels vibrant game communities with an interactive event platform that makes it easier for gamers to find, follow, and join in the action.

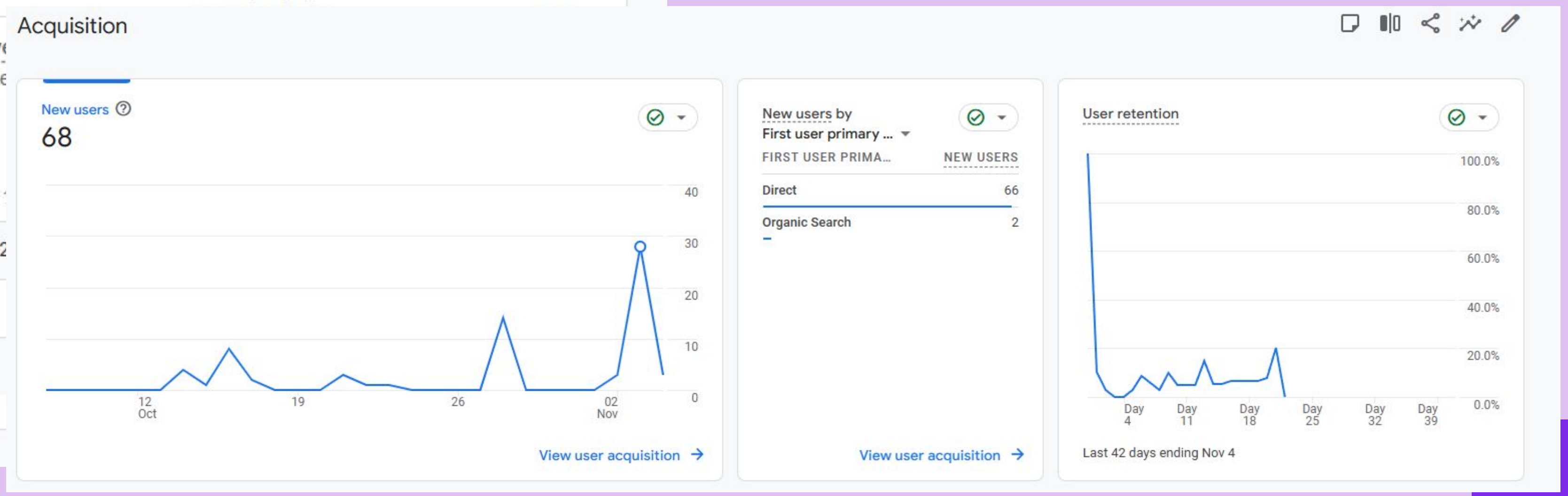
GA4 Insights and Results

Below are some of my user acquisition results, these show that an overwhelming majority of users come from Direct. This is expected due to referrals and user testing which had me send out my link to others. However, i have had 2 users visit my website via organic search results. This means 2.8% of my users thus far have been via organic search. Unfortunately this shows that i also have a lot more work to do for SEO and that my current methods are not as effective as I had hoped they'd be. But it also means that there is a foundation for my SEO that is however so slightly effective.



User acquisition: First user primary channel group (Default channel group)
Last 28 days 8 Oct - 4 Nov 2025
New users by First user primary channel group (Default channel group) over time
Day
40
30
20
10
0
09 Oct
11
13
15
17
19
21
23
25
27
29
31
01 Nov
03
Total
Direct
Organic Search
Plot rows
Search...
Rows per page: 10
1-3 of 3

	First user prim...channel group)	Total users	↓ New users	Returning users	Average engagement time per active user	Engaged sessions per active user	Eve All e
✓	Total	71 100% of total	68 100% of total	18 100% of total	5m 09s Avg 0%	2.18 Avg 0%	
✓	1 Direct	69 (97.18%)	66 (97.06%)	17 (94.44%)	5m 12s	2.20	4,22
✓	2 Organic Search	2 (2.82%)	2 (2.94%)	1 (5.56%)	1m 44s	1.50	
	3 Unassigned	1 (1.41%)	0 (0%)	0 (0%)	3m 33s	0.00	



New users by First user primary ...

FIRST USER PRIMA...	NEW USERS
Direct	66
Organic Search	2

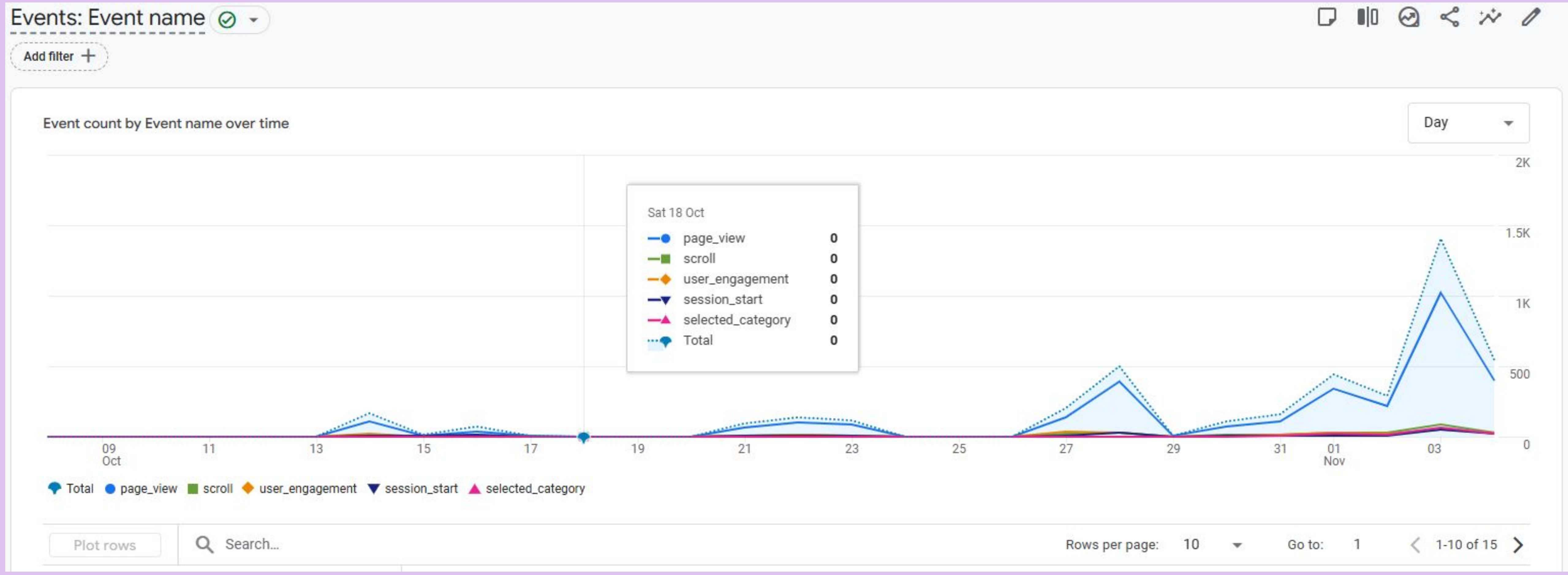
View user acquisition

GA4 Insights and Results

My most useful analytics are my events. The first few events, like page_view, scroll, user_engagement, and session_start, are automatically collected by Google. They're useful for seeing general activity—the spikes show when I was doing heavy testing—but the real insights come from my custom events.

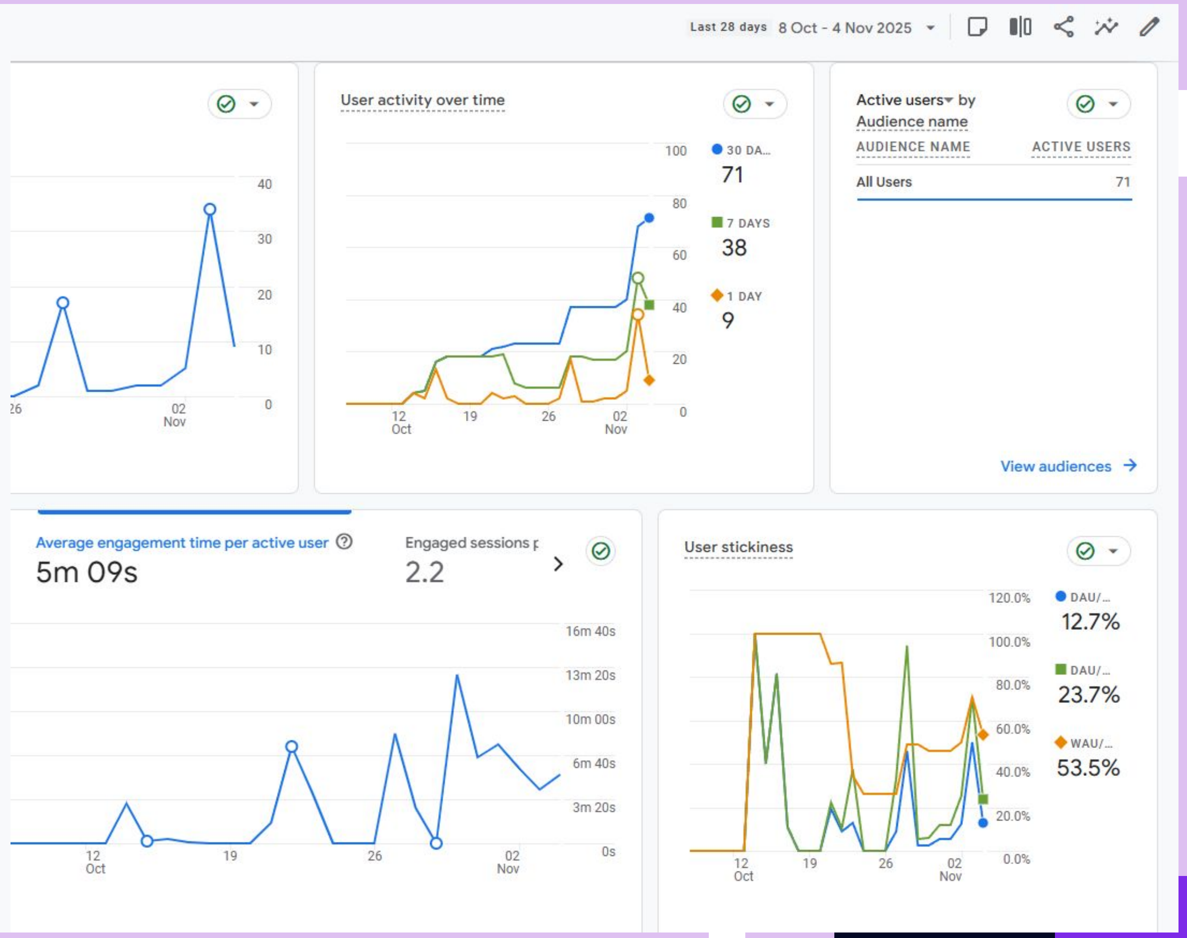
This is where the value is. I've implemented a full suite of custom event tracking to measure the *entire* user journey.

- User Registration:
  - registered_user (20 events) is a key conversion metric, showing 20 new user signups. This is pretty decent in comparison to the amount of users that used my site. Approximately 30%
  - logged_in_user (40 events) proves my login system is being used and that there are repeat logins.



Sat 18 Oct

page_view	0
scroll	0
user_engagement	0
session_start	0
selected_category	0
Total	0


Last 28 days 8 Oct - 4 Nov 2025
User activity over time
Active users by Audience name
View audiences →
User stickiness

User activity over time: 30 DAYS 71; 7 DAYS 38; 1 DAY 9

AUDIENCE NAME	ACTIVE USERS
All Users	71

Average engagement time per active user 5m 09s

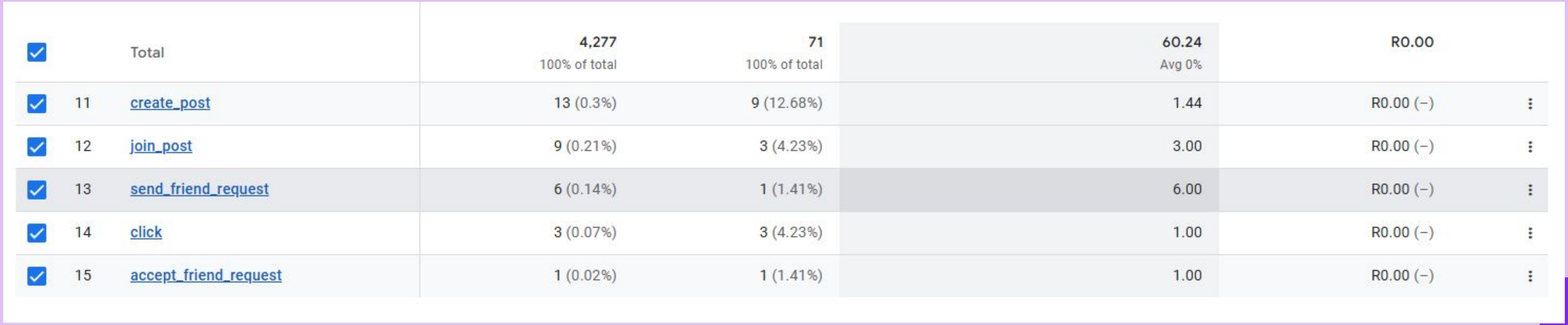
Engaged sessions per active user 2.2

User stickiness: DAU/MAU 12.7%; DAU/WAU 23.7%; WAU/MAU 53.5%

GA4 Insights and Results

- Core App Engagement:
  1. selected_category (136 events) is a great insight. It shows users are actively browsing and clicking on game categories, which is the main user flow.
  2. create_post (13 events) and join_post (9 events) are my most important core metrics. This proves that users are successfully using the LFG system's primary functions: creating and joining groups.
- Social & Profile Features:
  1. updated-profile (20 events) shows that users are engaging with their profiles after signing up.
  2. send_friend_request (6 events) and accept_friend_request (1 event) confirm that my social friend system is fully tracked and functional.

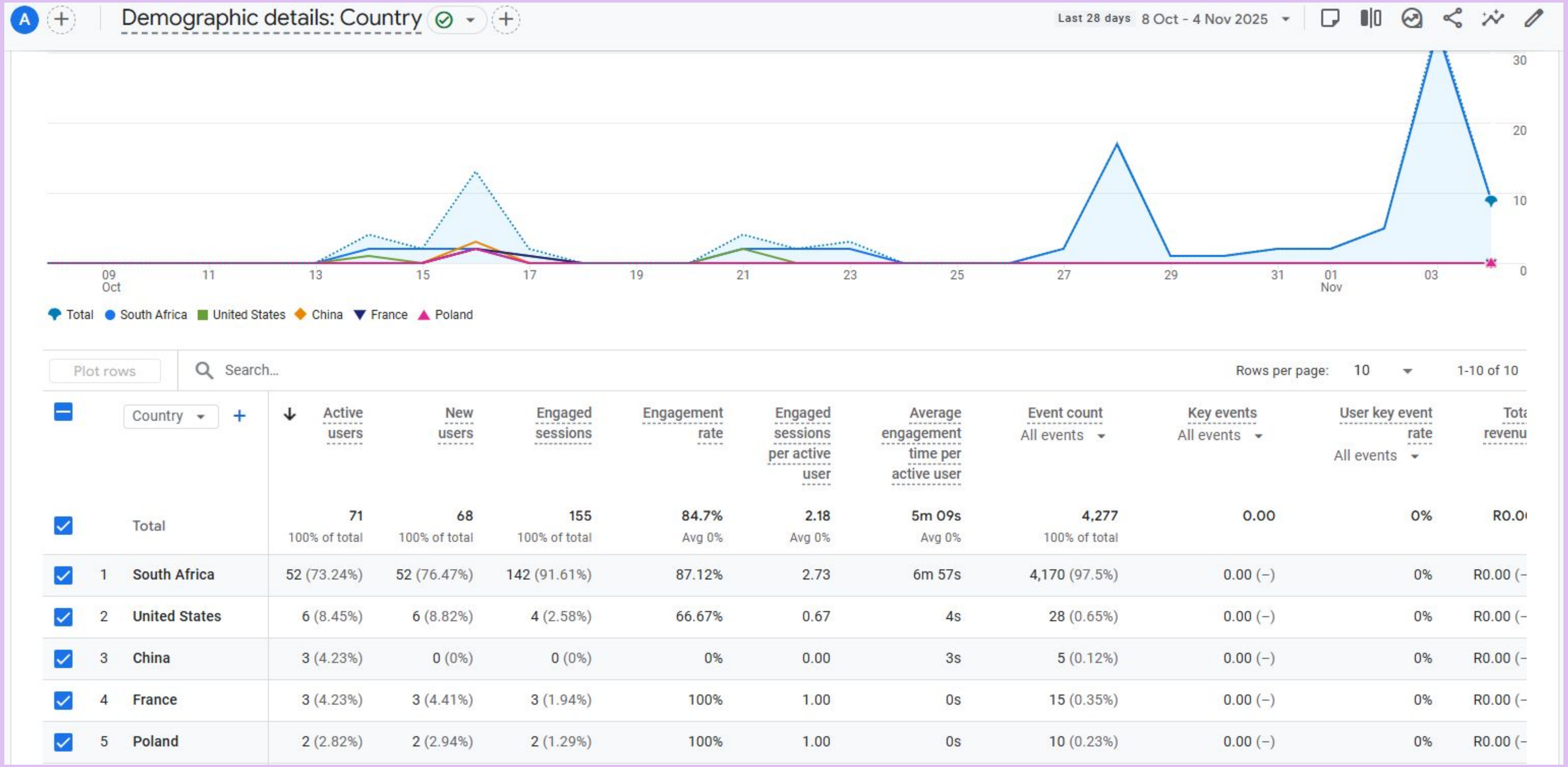
These insights pointed me towards a bug with accepting friend requests, hence its low value



✓		Total	4,277 100% of total	71 100% of total	60.24 Avg 0%	R0.00
✓	11	create_post	13 (0.3%)	9 (12.68%)	1.44	R0.00 (–)
✓	12	join_post	9 (0.21%)	3 (4.23%)	3.00	R0.00 (–)
✓	13	send_friend_request	6 (0.14%)	1 (1.41%)	6.00	R0.00 (–)
✓	14	click	3 (0.07%)	3 (4.23%)	1.00	R0.00 (–)
✓	15	accept_friend_request	1 (0.02%)	1 (1.41%)	1.00	R0.00 (–)

GA4 Insights and Results

Majority of my users are from South africa, with the second highest being the united states. The information from South Africa is the most useful by far, the remainder have not given me many key metrics. This popularity is due to me sending the link to other South Africans and the domain being a .co.za



● Total ● South Africa 🟩 United States ◆ China ▼ France ▲ Poland

Rows per page: 10 · 1-10 of 10

	Country	Active users	New users	Engaged sessions	Engagement rate	Engaged sessions per active user	Average engagement time per active user	Event count All events	Key events All events	User key event rate All events	Tota revenu
	Total	71 100% of total	68 100% of total	155 100% of total	84.7% Avg 0%	2.18 Avg 0%	5m 09s Avg 0%	4,277 100% of total	0.00	0%	R0.0
1	South Africa	52 (73.24%)	52 (76.47%)	142 (91.61%)	87.12%	2.73	6m 57s	4,170 (97.5%)	0.00 (–)	0%	R0.00 (–
2	United States	6 (8.45%)	6 (8.82%)	4 (2.58%)	66.67%	0.67	4s	28 (0.65%)	0.00 (–)	0%	R0.00 (–
3	China	3 (4.23%)	0 (0%)	0 (0%)	0%	0.00	3s	5 (0.12%)	0.00 (–)	0%	R0.00 (–
4	France	3 (4.23%)	3 (4.41%)	3 (1.94%)	100%	1.00	0s	15 (0.35%)	0.00 (–)	0%	R0.00 (–
5	Poland	2 (2.82%)	2 (2.94%)	2 (1.29%)	100%	1.00	0s	10 (0.23%)	0.00 (–)	0%	R0.00 (–

Optimisations

Cloudinary: Instead of storing images on my server, I offload them to Cloudinary. This uses their fast global CDN, making image loading *much* faster and reducing the load on my backend.

Debounced Search: My search bars use a useDebounce hook. This prevents spamming the API on every keystroke and only sends one request after the user stops typing, which saves database resources.

Further i have opted to use as few database queries as possible, to save on database resources and avoid spamming the database numerous times

Conclusion

To conclude, this project was a fantastic journey. It evolved from a simple LFG concept into a full-stack, deployed application.

I overcame several challenges:

Media Uploads: I taught myself the full multer and Cloudinary pipeline, including how to extract public IDs from URLs to delete orphaned images.

React SEO: I built a custom useSeo hook to dynamically add meta tags, which, combined with a sitemap, made my app indexable by Google.

Complex SQL: I learned to write efficient JOINs and UNIONs to handle the friends list and post data in a single call.

Reflecting on this semester, my biggest takeaways are a deep understanding of:

1. Security: Utilising JWT to verify users without making extra queries
2. SQL Database Design: How a normalized SQL database is the backbone of a complex app and the benefits of a decoupled deployment
3. Optimization: How difficult and how much work SEO can be, as well as the importance of analytics and feedback for the user journey

For the future, I plan to add live messaging with WebSockets, a clan/faction system, and more user safety features like blocking others.

Thank you.