Indy-7 - “League Coaching Platform”

Final Report

CS 4850 - Section 01 - Sharon Perry - Fall 2025 - December 8th, 2025

Team members:

Name:	Role	Cell / Email
Konrad (Team Lead)	Documentation	762.887.0844 Konkrad@proton.me
Jose	Developer	706.618.1178 genjose1231@gmail.com
Nahum	Developer	706.980.1957 nahummancera1014@gmail.com
Erik	Test	706.913.6446 ErikVasquez121@gmail.com
John	Documentation	470.557.2025 jtran50@students.kennesaw.edu
Sharon	Owner / Advisor	770.329.3895 Sperry46@kennesaw.edu

Project website link: https://ksu-lcp.github.io/

GitHub repository: https://github.com/KSU-LCP/LCP

STATS AND STATUS		
LOC	4500	
Components/Tools	React/Next.js, Supabase, TailwindCSS, Vercel	
Hours Estimate	450	
Hours Actual	500	
Status	Project is 85% complete, mostly functional with some features needing refinement	

Table of contents

1. Introduction

Many people have had a hard time learning to play the popular game “League of Legends.” With our platform, we will make it easier than ever for new players to learn how to improve their skills in the game. We will be building a modern web application that gives you individualized advice learned from a large dataset. Users will be able to enter their username and receive advice based on their recent games.

2.1 Definitions and Acronyms

LCP	League Coaching Platform
TFT	Teamfight Tactics
API	Application Programming Interface
AWS	Amazon Web Services
RDS	Relational Database Service
Baas	Backend as a Service
KDA	Kill Death Assist ratio
CS	Creep score

2. Challenges and Assumptions

3.1 Assumptions

- Users will have access to a modern web browser (Chrome, Edge, Firefox, Opera, Zen)
- User has basic familiarity of League of Legends
- Users who wish to create an account on our platform will also have a Riot Games account so their match history and ranked data can be fetched
- Riot API remain accessible and reliable for pulling player game data
- The Supabase cloud platform and its services remain available and have acceptable uptimes and performance

3.2 Challenges

3.2.1 API Rate Limit and Reliability

The backend system heavily relies on the Riot API. To relieve this, we designed our edge functions to avoid going over the rate limit and handle any outages gracefully. This also includes request retries avoiding infinite loops, a user friendly error message, and detailed logging.

Rate Limits:

- 20 requests every 1 seconds(s)
- 100 requests every 2 minutes(s)

3.2.2 Database Normalization and Scalability

When designing the initial database schema, it initially began with cardinality issues and poor normalization. As more requirements were introduced, it made it difficult to maintain causing inefficient queries. So, we had to redesign the schema to achieve a rough third normal form, improving overall query performance and scalability.

3.2.3 Migrating from AWS to Supabase

In the beginning, the initial architecture was comprised of several AWS services including RDS, Lambda, API Gateway, VPC, etc. The issue was there was too much time configuring our services and the expensive fees for daily development. Then Supabase was introduced providing its generous free

tier, integrated authentication, and ability to accelerate development without the concern of the configuration.

3.2.4 Aggregated Analytics

Analytics regarding player data is one of few most important datasets we need to provide metrics, and it would be too expensive to compute when requested. So, using one of Supabase unique features, we use Views to provide precomputed metrics for faster and easier requests.

3.2.5 Scope and Limited Time

Given a semester, not every planned feature could be implemented; this includes the TFT game mode and its player data. Due to the limited time, we decided to abandon the feature and solely focus on the primary game mode, having a larger player base and more users to support.

3.2.6 Extra Scope for Teams Larger than 4

Due to the larger group size within our team, we had two individuals working on documentation. One of the features we brainstormed was including a wiki section within our platform which provided useful tutorials and guides on how to navigate the website and interface. This was placed within the extra scope because it was not required in order for our requirements to be met.

3.2.7 Providing Meaningful Coaching Advice

Recording raw metrics is straightforward like KDA, CS, gold, etc. but turning it into clear, coaching tips was challenging. We had to define how we were going to set a baseline for “good” performance that accounted for different roles and champions. So, how we approached this, we implemented a comparative analysis system that compares a user against the average statistics of a high-ranking player within division 1 out of 4 being the best. This resulted in the application being able to identify performance gaps in a player overall of a specific match, e.g. player had low vision score compared to the average high-ranking player.

3.3 Risks

Risks are still present depending on the setting but mainly occur within our Supabase backend or Riot API.

3.3.1 Dependency Risks

Outages or significant changes in the Riot API or Data Dragon endpoints could break our analytics, resulting in a pause without our data ingestion. Furthermore, outages in Supabase would result in our application being broken until resolved.

3.3.2 Cost and Scalability Risks

As more users and matches are ingested, the database will grow to the extent that exceeds free tier. This will then require monitoring and cost management plans to keep the billing within our budget.

3.3.3 Security Risks

Storing user accounts including their sensitive data requires proper encryption, access control, and secure handling of API keys.

3. Requirements

4.1 Functional Requirements

4.1.1 User Accounts and Authentication

- Users can create an LCP account using their email, username and password
- Users can create an LCP account using their Google account via OAuth
- Users can login to LCP using their credentials or Google account
- Users can recover a password via reset flow

4.1.2 Homepage

- Home page displays a search bar for riot summoner lookup
- A login button is displayed
- A navigation menu is displayed
- A post that redirects to League of legends patch notes

4.1.3 Navigation

- Navigation should display options containing:
    - Home
    - Global Metrics
    - Blogs

4.1.4 Global Metrics

- Global Metrics page is to display metrics regarding all our data including:
    - Total Matches collected
    - Most Played Queue/ Game mode
    - Most played Champion
    - Top 5 Most played Champions
    - Objectives Captured
    - Rank Distribution Pie chart

4.1.5 Riot Account Page

The Riot Account page displays an overview of the players including their basic stats, match history, and personalized coaching stats.

- Display basic account information including name, rank, region, win rate and total matches.
- Display ranked stats of information including this year's wins, losses and league points.

- Display most recent matches and the data pertaining such as champions, items, runes, kda, summoner spells, date, duration, and metrics for each participant in a single match
- Display personalized coaching insight/tips if the user has at least 1 ranked match in in the previous 20 matches and display performance analysis comparing stats to high ranked players.

4.1.6 Coaching Engine

- The coaching engine computes metrics across recent matches including CS per minute, KDA, vision, gold, and objectives captured among high ranked players specifically in the rank 1 division.
- A threshold is to be set, to compare other ranked individuals, and return an output that recommends areas for improvement

4.2 Non-Functional Requirements

4.2.1 Security:

- Data Protection: All user data should be stored in a database and fully secured to only allow admins to and lambda function to created delete or update user information. This should also include encryption for function calls carrying sensitive user information such as passwords and log-in information.
- Access Control: The website should also only allow users to view and change only their own account information. For administrators, they will only have access to cloud services they require.
- Web Security: The website should have standard web application security to prevent common basic attacks such as XSS and SQL injections

4.2.2 Usability:

- The website should provide a responsive layout across desktops and mobile devices.
- The website should be compatible with all major browsers
- The website’s navigation menu and styling must be clear, consistent

4.2.3 Performance:

- Dashboard and player history should load within a reasonable time of 5 seconds

4.2.4 Reliability

- No unexpected crashed during repeated requests for all endpoints

4. Analysis / Design

5.1 Design Considerations

5.1.1 General Constraints

- We will need to design our system to not go over the Riot Games API Rate Limits.
- Our design will need to be scalable as our user base grows.

5.1.2 Development Methods

- Version Control: GitHub
- IDE: Visual Studio Code, WebStorm
- Communication: Discord , GitHub , Messager

5.1.3 Architectural Strategies

- Programming: Typescript, HTML, CSS, React (JSX/TSX)
- Database: PostgreSQL
- Storage: Supabase Storage Bucket
- Cloud Service: Supabase(Baas)

5.2 System Architecture

- Data Pipeline: feeds match data through Riot API inserts into PostgreSQL
- Storage Layer: PostgreSQL schema for matches, champions, items, etc.
- Backend: edge functions, views, and cron jobs are used to ingest and record data
- Frontend: interface to showcase searchable user, champion, item, comp, augment statistics

5.3 Detailed System Design

1. Classification

   Data Pipeline, Database, Backend, Frontend

2. Definition

- Data Pipeline: Feeds in the Riot Data

- Database: Keep match data
- Backend: Provide data access through APIs
- Frontend: Display statistics and data

3. Constraints

- Riot API limits
- Scalability: Our system will need to be able to scale up to handle load
- Cost concerns: We would like to keep the costs as close to 0 as possible.

4. Resources

- AWS RDS for PostgreSQL (Supabase for backup)
- AWS Amplify Webhosting
- Lambda for API Hosting / Backend Services

5. Interface/Exports

- Website endpoint
- API endpoints:
    - Search: redirects to user profile and returns a Json response

GET	/summoner/{username}/{tag}

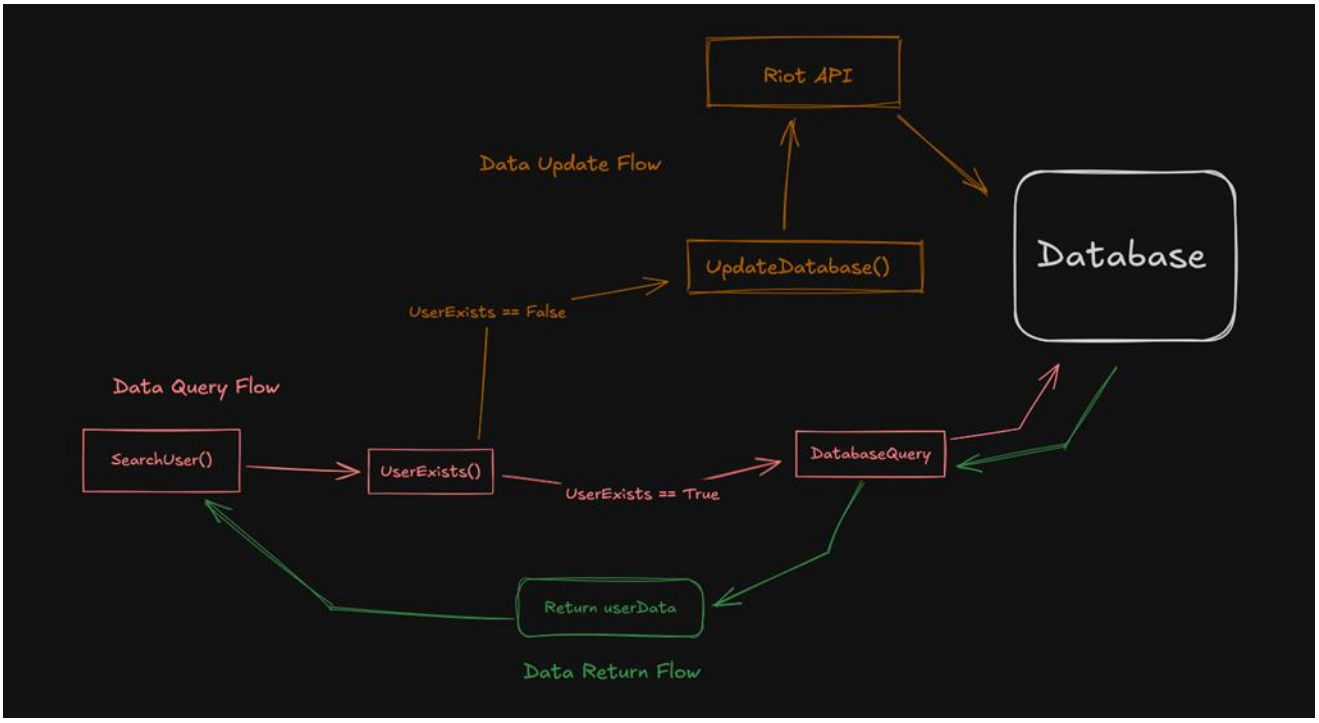
    - Global Metrics endpoints: redirects to global metrics page

GET	/global

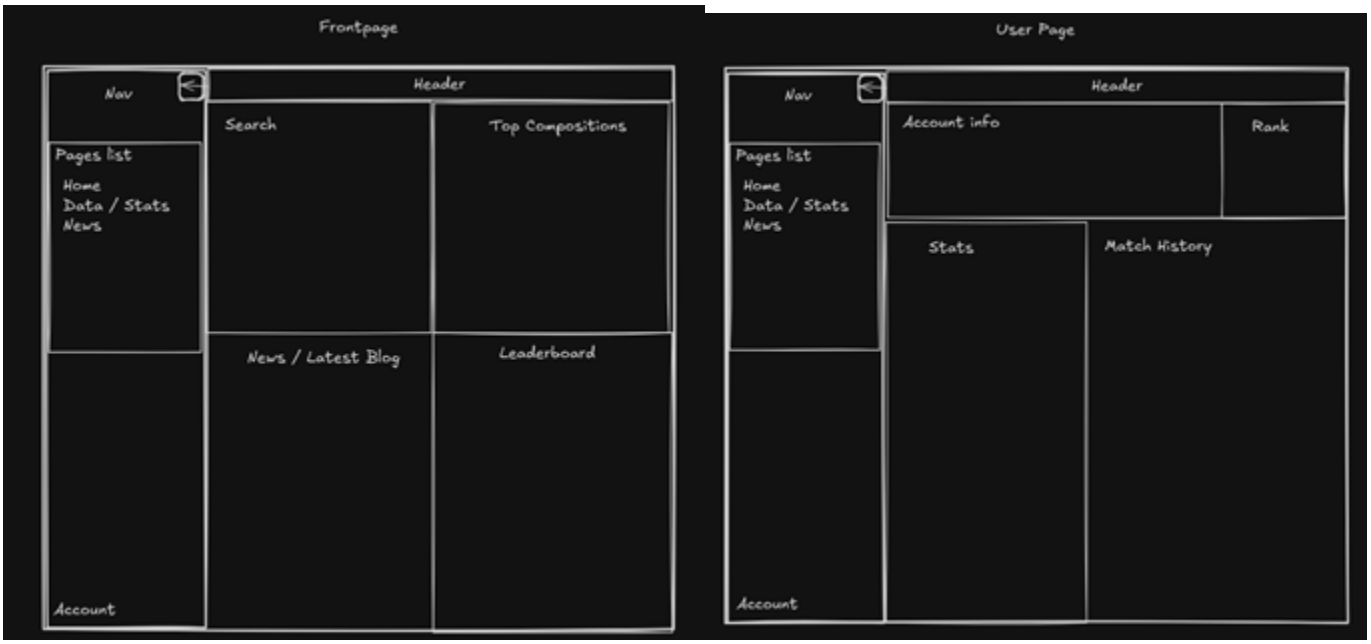
    - Authentication and Account: redirects to login page and returns a Json response

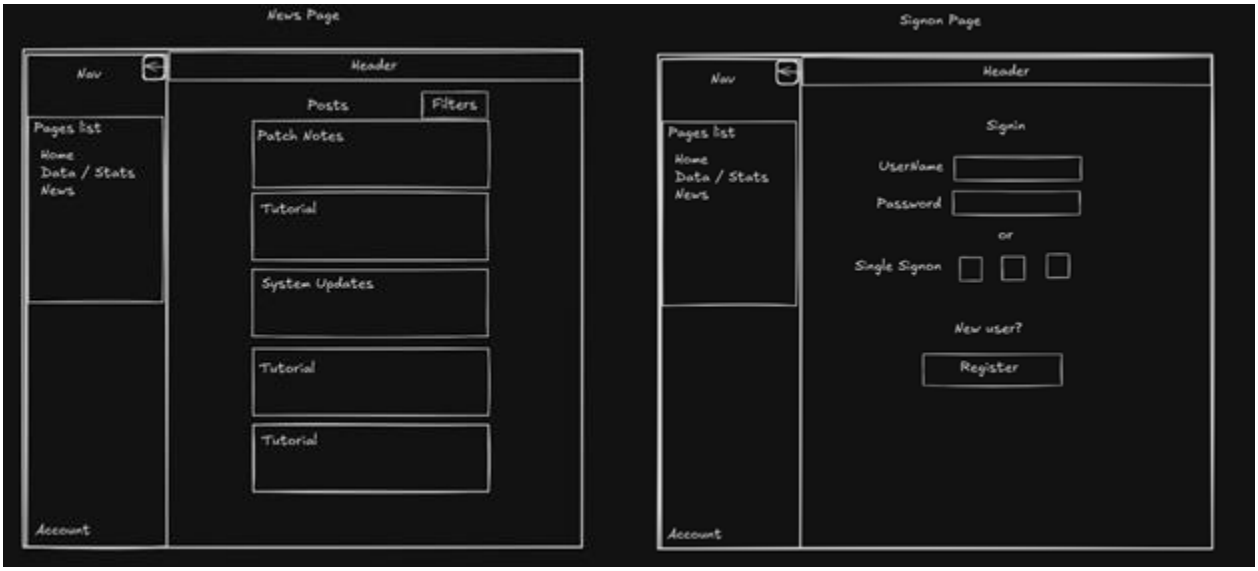
POST	/register
POST	/login
POST	/forgot-password
POST	/reset-password

5.4 Data Retrieval Design



5.5 Wireframes




News Page
Nav
Header
Posts
Filters
Pages list
Home
Data / Stats
News
Patch Notes
Tutorial
System Updates
Tutorial
Tutorial
Account
Signon Page
Nav
Header
Pages list
Home
Data / Stats
News
Signin
UserName
Password
or
Single Signon
New user?
Register
Account

5. Development

6.1 Outline with details about building each concept

- Storing our 3 data types: Static data, static image icons, player game data.
  - We first designed our schema for static data assets from League of legends, this includes entities such as champion, item, rune and summoner spell tables. For each table, we only store the necessary data we want from each entity like their name, description, and icon URL that stores a link redirecting to our storage bucket of that image.
  - Then a Supabase storage bucket is created allowing us to store images into a bucket directory.
  - As for player game data, we have a summoner table as our main player profile, storing details like their rank, summoner level, wins, losses, PUUID, top champions, etc. We also have the match table for storing game details, the team table for team statistics, and the participant table that connects players to their matches with all their performance stats. We also have tables like participant_item, participant_rune, and participant_summoner_spell that refer to our static data tables giving us match details of their selected items, runes and summoner spells

- Edge functions
  - Function get_all_static_data, makes external requests to Riots Data Dragon endpoints to collect and store the most up to date data from champions, items, runes, and summoner spells.
  - Function get_all_static_image, also makes external requests to Riots Data Dragon endpoints to retrieve champion, item, rune, and summoner spell icons in a .PNG format and store them in our Supabase storage bucket.
  - Function get_summoner, takes in a player's Riot name and tag and fetches their account details from Riot's API, including their rank, summoner level, PUUID, top 3 most played champions and more. This data gets upserted into the summoner table.
  - Function fetch_matches, takes in a players PUUID and fetches their match data to store the game details in the match table, team statistics in the team table, and individual player performance in the participant table. We also store what items, runes, and summoner spells each player, which all reference back to our static data entities mentioned earlier.

- Creating a front-end design while still providing useful data for users trying to learn the game.

- Implementing our database design.
  - Super Base provides api keys that we are able to use for retrieving data.

    - This allows us to display and calculate statical data that users would like to know to see current trends, progression, and other useful information.

- What types of analysis and advice will we provide for the end users.
    - Based on the Data received, we discussed what analysis could be made from it in the form of Supabase views. Supabase views act in the form of a database query to analyze specific information and their relation.
    - We are able to analyze game performance metrics from multiple matches collected. Metrics that include gold earned per game, specific champion win rate over hundreds of games, commonly built items win rates, etc. Analysis of such metrics allows us to give general advice based off these statistics.
    - Profile specific metrics are also available when the data from a single user is retrieved from Riot Api to give a personalized summary of their game performance compared to players from other ranks. They can see how they performed compared to pro players and see which areas they need to improve.
    - Tags based of their recent match performances are set off certain metric thresholds that notify the user what they are doing good and what they are doing bad in a badge system.

- Migrating from AWS to Supabase:
    - Upon working on the project, we decided that working with Supabase off the cloud was better primarily for its simplicity and generous free tier giving access to many tools which lead the group to decide against developing our own AWS deployment.
    - Redefining our data around new requirements, as we worked on the project, we realized that certain parts of the initial database design wouldn't be scalable.
    - The initial design of our database had issues with cardinality and poor normalization. As we added more requirements, we saw it would be difficult and very inefficient to query data. So, we redesigned our database schema which would result in achieving a third level of normalization.

- Connecting our frontend to the backend:
    - We largely use the nextjs as a middleman between our edge functions and the database. This allows us to setup functions that don't require auth to our back end and we use this for functions we know are safe to be accessible to the public.

- Creating aggregated stats for the frontend
- Created aggregated stats that would set up the basis of our coaching insights. This includes taking data already stored in our database and analyzing specific statistics to put in a separate table. This allows us to use these computed statistics to base our coaching insights off of.

- These aggregated stats make it easier to query and more accessible to the front end when queried through a Supabase view. This makes it possible to compare our computed stats against the twenty most recent matches available in a user's profile, giving them insight into their performance.

6.2 Information about the database connection

The primary method of connecting to the database is by making http requests to the Supabase edge functions used as a backend layer between the frontend client application and the database. Instead of the frontend connecting directly to the database, it instead happens on server side through Supabase edge functions.

These edge functions work by using the Supabase JavaScript client to perform actions such as queries, inserts, upserts into our PostgreSQL database. When a http request is made in one of our functions, the function authenticates internally using the service role key, which initializes the Supabase Client and executes SQL operations on relevant tables such as match, participant, summoner, and team that are relevant to the fetch matches edge function.

Edge functions are sent to Supabase's edge network, allowing for low latency access to both the database and the Riot API. This also makes it where the front end never has to see any secret keys because it isolates database credentials, making it easier to schedule background jobs such as updating player summary statistics or computing champion win rates.

Each of our functions follow a similar pattern of

```
import { createClient } from "https://esm.sh/@supabase/supabase-js@2";

const supabase = createClient(

Deno.env.get("SUPABASE_URL")!,

Deno.env.get("SUPABASE_SERVICE_ROLE_KEY")!

);
```

This is to securely initialize the connection using environment variables stored in Supabase. Our edge functions serve different purposes but still have this common architecture. Fetch matches and connects to the database to upsert the most recent matches fetched from Riot API, while champion stats can compute global champion win rates for analytics or scoreboard purposes. This system is a secure, scalable, and maintainable way to handle a connection to the database through Supabase.

6.3 Project set up

We have the project setup as a mono repo on GitHub which contains our frontend, backend and database information. Within the mono repo contains folders separating each component.

Frontend:

- Download the repo from GitHub
- Open up the repo
3. Enter the “frontend” directory
4. Run “npm install”
5. Run “npm run dev”

Backend & Database:

1. Setup and configure Supabase Account
2. Create Organization and invite team members
3. Research Riot API
4. Create Riot Developer Account
5. Obtain Riot API Developer Key
6. Plan out Database design
7. Create database or run our migration scripts.
8. Create database structure/schema
9. Input Database entities, fields, cardinalities, etc.
10. Input API secret keys
11. Test connection to the Riot API using Supabase client
12. Create edge functions that retrieve data from the Riot API and Data Dragon assets
13. Create Supabase views that allow one to analyze the data and be accessible to the front end.
14. Test Edge functions to ensure that the data was collected without error.

6. Test Plan

7.1 Purpose of Testing

To ensure complete functionality of our coaching platform, we created a list of functionalities that need to be in working order before the final version is complete. As we tested different features of our platform, each test case was noted within our software testing plan document.

7.2 Scope

7.2.1 In-Scope

- User Interface
- Database Schema and Integrity
- Supabase Edge Functions, Views, and Storage
- Riot API Functionality
- League Asset Ingestion
- Authentication and Access Control

7.2.2 Out-of-Scope

- Usability (done by users informally)
- Monetization features (not implemented)
- Riot API Uptime
- Request Rate Limiting

7.3 Analysis of Scope and Test Focus Areas

7.3.1 Release Content

- Edge Function: Fetch last 20 league matches
- Edge Function: Incorporate League assets (champions, items, runes, augments)
- Analytics engine
- Coaching module to evaluate player weaknesses based of rank averages
- Player dashboards and graphs
- Match history review page

7.3.2 Regression Testing

- New data is added affecting analytics
- Riot API version changes
- Database schema changes
- Coaching formulas are updated

7.3.3 Platform Testing

- Supabase Postgres: v15
- Supabase Edge Runtime: Deno runtime
- Browsers: Chrome, Firefox
- Riot API platform: AMERICAS/Na1
- Github Actions CI environment

7.4 Functional vs Non – Functional Test Cases

In addition to categorizing our test cases, we also divided them into functional and non-functional cases which added further organization to our process. A list of examples below will demonstrate the difference between these two.

7.4.1 Functional Test Case

- Ensuring the search feature works as intended when the user inputs their player's name.

7.4.2 Non – Functional Test Case

- Analyzing the responsiveness of the website and determining if it aligns with our requirements.

7.5 Test Cases

Test Case 001:

A. Visit all of the following routes
    - https://www.ksulcp.com/
    - https://www.ksulcp.com/{Summoner}/{TagLine}
    - https://www.ksulcp.com/login
    - https://www.ksulcp.com/register
B. All routes should successfully load including data, metrics, images, etc.

Test Case 002:

A. Verify login and registration with valid inputs
B. The account is created in the database, then redirecting the user to the homepage. Session is destroyed upon logging out.

Test Case 003:

A. Account Linking using Google and Riot OAuth
B. External Accounts are successfully linked or unlinked, and auth tokens are received without redirection

Test Case 004:

A. Fetch last 20 matches for a valid puuid
B. API call succeeds and returns 20 match IDs each in the correct format.

Test Case 005:

A. Fetch matches with an invalid or empty puuid
B. API returns a clear error, does not crash, and gives response message

Test Case 006:

A. Store raw match data in Supabase
B. Raw match JSON is stored and normalized in the correct tables

Test Case 007:

A. Compute metrics for a single match
B. CS/min, KDA, kill participation, vision score, gold details, etc, are computed correctly and saved for the single match

Test Case 008:

A. Compute metrics for all 20 recent matches
B. CS/min, KDA, kill participation, vision score, gold details, etc, are computed correctly and saved for all 20 matches

Test Case 009:

A. Generate coaching recommendations based of recent metrics
B. Coaching text tags generated based off thresholds, saved and linked to player

Test Case 010:

A. Display dashboard for a selected player

B. Dashboard loads without errors, graphs show, values on charts match data base analytics

Test Case 011:

A. Display champion specific insights
B. Champion breakdowns appear and stats relating to that specific champion

Test Case 012:

A. League asset ingestion pipeline
B. All expected assets in the league's assets bucket are saved in the database. All icons are loaded correctly on the front end.

Test Case 013:

A. Handle missing or partially ingested assets UI shows a fallback icon/text.
B. No crashes

Test Case 014:

A. Error handling when Riot API rate limit is hit Edge function returns clear error when a time out or rate limit is hit.
B. No infinite retries or crashes.

Test Case NF01:

A. Website displays data within an acceptable performance
B. Loads dashboard with recent matches Dashboard fully loads in under 5 seconds under normal conditions

Test Case NF02:

A. Reliability in repeated match fetching operations
B. No unexpected crashes or memory leaks

Test Case NF03:

A. Easy and simple coaching advice readability
B. Coaching messages are concise and readable. Formats are structured for user awareness.

Test Case NF04:

A. Ensure Logging and monitoring are sufficient for debugging
B. Key operations write meaningful logs for debugging

7.6 Test Procedures

TC001:

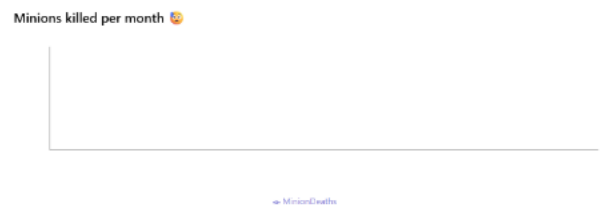
Visit the routes on the following modern web browsers:

- Chrome
    - https://www.ksulcp.com/
    - https://www.ksulcp.com/{Summoner}/{TagLine}
    - https://www.ksulcp.com/login
    - https://www.ksulcp.com/register
- Firefox
    - https://www.ksulcp.com/
    - https://www.ksulcp.com/{Summoner}/{TagLine}
    - https://www.ksulcp.com/login
    - https://www.ksulcp.com/register
- Safari
    - https://www.ksulcp.com/
    - https://www.ksulcp.com/{Summoner}/{TagLine}
    - https://www.ksulcp.com/login
    a. https://www.ksulcp.com/register

Verify charts load on web browsers

1. Pass:

2. Fail:



TC002

1. Navigate to Register page
2. Enter valid username, email, password and click Register
3. Verify you are directed to homepage

Expected:

- HTTP Status code is 200
- New Account created and recorded in Database
- On logout, session is destroyed

TC003

1. Log in with valid Google or Riot account
2. Navigate to Account linking
3. Click Google, complete Oauth flow
4. Verify UI showing google is linked
5. Repeat same for Riot Account

Expected:

- Google or Riot account is successfully linked or unlinked
- User remains within app flow

TC004

1. Open postman
2. Set method to Get
3. Enter endpoint URL
4. Ensure required headers are set
5. Send the request
6. Observe the HTTP status code and response body

Expected:

- HTTP status code is 200
- Response Body displays 20 match IDs
- Each match ID is in the correct riot format

TC005

1. Open Postman
2. Set method to GET
3. Use the same endpoints but
    a. Either leave puuid empty
    b. Set puuid to and invalid puuid
4. Send the request
5. Await HTTP status code and response body
6. Check the logs

Expected:

- HTTP status code is 400
- Response contains a clear error message
- No unhandled exceptions

TC006

1. Execute Test Case 004 to fetch matches
2. Open Supabase dashboard
3. Go to matches table where raw match data is stored
4. Filter by test puuid
5. Inspect all fields

Expected:

- A record exists for each match ID returned from the Riot API
- Raw JSON is stored correctly
- Normalized fields
- No duplication or inconsistency

TC007

1. Identify one specific match stored in the database from Test Case 007
2. Trigger the metrics function
3. Wait for the function to complete and check the response or logs
4. Open Supabase and open match metric tables
5. Filter by puuid and match id

Expected:

- A metrics row exists for that match and player

- Values such as CS/min, KDA, vision score, etc.
- Manually calculated metrics based on raw match data

TC008

1. Confirm that at least 20 matches for a test puuid are stored
2. Trigger multiple metric computation for all 20 matches
3. In Supabase query the metrics table for that puuid across all 20 match IDs
4. Count the number of metric rows

Expected:

- Metrics entries exist for all 20 matches
- No matches are missing metrics
- No duplicates
- Accurate metrics

TC009

1. Ensure metrics exists for all recent matches done in Test Case 008
2. Call the coaching function
3. Execute function and check response and logs
4. In Supabase open coaching recommendation table
5. Filter by test PUUID
6. Open front end Coaching view for that player

Expected:

- One or more coaching tags for that player
- Each tag includes test
- Recommendations line up with thresholds set in logic
- Front end displays this recommendation clearly

TC010

1. Open the front end in a browser
2. Log in with a valid account
3. Go to test Summoner profile
4. Wait for all charts to load
5. Open Devtools console to watch for errors

Expected:

- Dashboard loads without fatal errors
- Key charts appear
- Numerical values shown and legible on chart
- UI is responsive

TC011

1. From the dashboard or view go to champion insights for a test player
2. Select a champion from their page
3. Observe champion specific stats

Expected:

- Champion list is visible and interactive
- Selecting a champion shows champion specific metrics
- No undefined data errors

TC012

1. Call static ingestion edge function
2. Wait for function to load
3. Check response and logs
4. In Supabase storage open league-assets bucket and verify all the icons
5. In database check tables
6. Open front-end UI and see if icons are displayed

Expected Results:

- All expected assets for the current patch exist in the bucket
- DB tables contain correct data
- Front end uses correct assets and renders correctly
- No broken image icons

TC013

1. Manually simulate a missing asset in the bucket by
    a. Deleting an icon
    b. Editing a path invalid
2. Open front-end view that uses that asset
3. Reload the page

Expected:

- UI displays a default icon or text
- No crashes

- The UI functions normally

TC014

1. Load
    a. Fetch matches repeatedly
    b. A mock rate limit response from the riot api
2. Observe response
3. Check logs

Expected:

- The edge function returns a clear error
- No infinite loops
- No crashes

7.7 Test Environment

Riot API -> Supabase Edge Function _> DB Storage -> Analytics engine -> Dashboard UI

7.7.1 Hardware

- Standard Personal Computer
- No special hardware required

7.7.2 Software Setup

- A website browser (including but not limited to: Google Chrome, Firefox, etc.)
- Postman for API endpoint testing
- Supabase Platform
    - PostgreSQL 15 database
    - Authentication
    - Storage buckets
    - Edge Functions (Deno runtime)
    - Postgres Views
- GitHub
    - Repository for storing edge functions, Postgres Views and front end
    - GitHub Actions
- Riot Games API
    - Developer key for match-fetching
    - Access to Data Dragon

7.7.3 Testing utilities

- Browser Dev tools
- Supabase SQL
- Logging Tools
    - Supabase Edge Logs
    - GitHub Actions logs
    - Browser network logs

7.8 Test Data

7.8.1 Test accounts(puuid)

- Test users must have an existing league account with some match history
    - Needed to support match history, coaching statistics, and dashboard graphs
- Different Elo accounts to test for coaching at different levels
- High Elo accounts to base the coaching data from

7.8.2 Static Data

- League of Legends Assets
    - Champion Assets
    - Item Icons
    - Rune Icons
    - Summoner Spell Icons
    - Arena Augments
- Stored in league-assets bucket
- Static Data ingestion edge function

7.8.3 Match Data

- Raw match JSON from Riot API needed for
    - Parsing logic
    - Storage logic
    - Metric computation

7.8.4 Analytics Data

- Precalculated expected values on
    - CS per minute, gold differences, vision score, etc.

7. Test Results

8.1 Non-Functional Results

Test Case NF01:

The website loads and pages are functional; the weakest part of this is the performance when searching for a new summoner.

Test Case NF02:

The API fetches match cached correctly improving performance for existing users and fetches need matches for existing users if they have played any since the last sync

Test Case NF03:

The website provides some actionable insights for players but not quite enough for this to be considered passing as of November. This could be improved by giving more match specific information instead of just player specific information as well as building out our database of information for players to use.

Test Case NF04:

The logs in supabase are sufficient for debugging. I do think that at the current time the web client could provide more useful error information to the end user if the API is down, or another issue comes up.

8.2 Functional Results

Requirement	Pass	Reason
Test Case 001	Y	
Test Case 002	Y	
Test Case 003	Y	
Test Case 004	N	Not implemented
Test Case 005	Y	
Test Case 006	Y	

Test Case 007	Y	
Test Case 008	Y*	More Data to be added
Test Case 009	Y	
Test Case 010	Y*	Additional widgets to be added
Test Case 011	Y	
Test Case 012	N	Partial functionality
Test Case 013	Y	
Test Case 014	Y	

8. Version control

For version control, we created an organization and repository on Github. We used this repository to store and maintain our primary frontend code with periodic backups of our backend from supabase. We went for this approach because supabase edge functions have their own IDE and storage solutions for our backend code but for our frontend having a traditional Git Repository essential.

Link to our organization on GitHub: https://github.com/KSU-LCP

Repo: https://github.com/KSU-LCP/LCP

Notable branches:

- Main: This branch is the deployed production branch, and updates to it will automatically be published to our website.
- Development Branches: Each of the following branches would be used by a member of the team to develop features, and the commits would be combined into a merge request with information about the feature before getting approved to move to main.
    - Erik Dev
    - Jose Testing
    - Konrad Dev
    - Nahum Dev

Conclusion or Summary

The League Coaching Platform (LCP) project, developed by Team Indy-7, is a modern web application designed to help players from all skill levels understand and improve the game “League of Legends”. The platform uses various methods such as charts, user data, and player tags that are on each individual player's profile in order to help them understand where they might be falling behind. The system is built using React/Next.js, SupaBase, and Tailwind CSS, relying on the Riot API for player game data. While the project report was 90% complete, the team faced challenges such as managing API rate limits, migrating from AWS to Supabase and defining what is “meaningful coaching advice”.

During the development of the League Coaching Platform, we successfully learned how to develop and deploy a web application from nothing, going through all the stages of the software development cycle we learned valuable lessons about working together, understanding scope and how to plan a project. Along the way we encountered multiple roadblocks, both expected and unexpected. Our software development document and requirements document helped us quickly realize we wouldn’t have time to execute our plan on backend development using AWS for deploying our backend infrastructure leading us to move to a database as a service solution Supabase. The Riot API rate limits were particularly influential for this change that would have normally been a death blow to a project but due to the plans and requirements set forth at the start of the project we were able to migrate and build a caching solution to reduce the API calls within a week and speed up development after that. Our frontend developed in React/Next.js using Tailwind CSS as styling letting us create reusable components and routes quickly, helping us quickly work on new features, allowing us to develop most of our ambitious features.

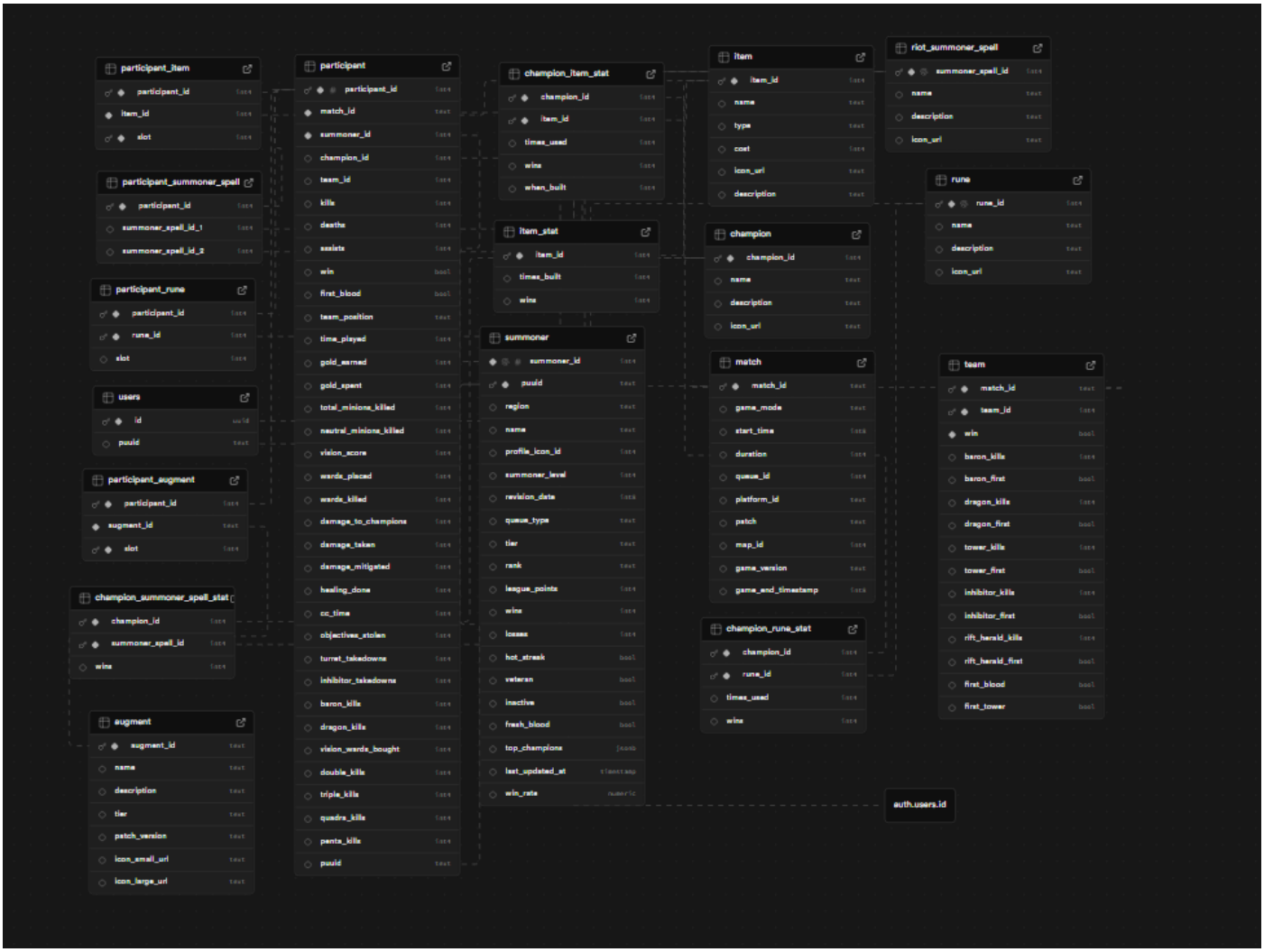
Appendix – training verification

Bibliography

- Riot Games Developer Portal: https://developer.riotgames.com
- FastAPI Documentation: https://fastapi.tiangolo.com
- Next.js Documentation: https://nextjs.org/docs/app/getting-started
- React Documentation: https://react.dev/learn
- PostgreSQL Documentation: https://www.postgresql.org/docs/current/
- AWS Documentation: https://docs.aws.amazon.com/

Supabase Images

Database schema:



Edge Functions:

NAME	URL	CREATED	LAST UPDATED	DEPLOYMENTS
fetch_matches	https://adjdvrvdekeqoonhjlcp.supabase.co/functions/v1/fetch-match…	10 Oct, 2025 17:03	a month ago	47
get_all_static_data	https://adjdvrvdekeqoonhjlcp.supabase.co/functions/v1/get_all_sta…	08 Oct, 2025 22:02	2 months ago	35
get_all_static_image	https://adjdvrvdekeqoonhjlcp.supabase.co/functions/v1/get_all_sta…	08 Oct, 2025 22:34	2 months ago	36
get_augment_assets	https://adjdvrvdekeqoonhjlcp.supabase.co/functions/v1/get-augment…	02 Nov, 2025 16:13	a month ago	15
get_coaching	https://adjdvrvdekeqoonhjlcp.supabase.co/functions/v1/get_coaching	05 Dec, 2025 13:51	2 days ago	10
get_summoner	https://adjdvrvdekeqoonhjlcp.supabase.co/functions/v1/get_summoner	07 Oct, 2025 17:36	2 months ago	45

Image of baseline metrics for coaching:

tier text	team_position text	games_sam... i...	cs_per_min_p50 float8	gold_per_min_p50 float8	dmg_per_min_p50 float8	wards_per_min_p50 float8	kda_...
CHALLENGER	EMPTY	1	0	263.81	0	0	0
CHALLENGER	BOTTOM	60	7.17	414.545	822.72	0.325	2.67
CHALLENGER	JUNGLE	151	6.31	419.9	657.5	0.16	4
CHALLENGER	MIDDLE	68	7.26	395.37	874.01	0.32	2.165
CHALLENGER	TOP	139	6.77	441.7	843.13	0.31	2.5
CHALLENGER	UTILITY	139	1.1	324.94	594.83	1.09	3.4
DIAMOND	MIDDLE	1	6.73	372.89	592.33	0.34	0.22
DIAMOND	TOP	19	6.78	379.75	815.93	0.23	1.33

Example image of a match, result of using Views

NA1_5414994595	42619	100	Minushirtus#huh	3LhRlhXpMjiwlplVar-22Qi0njwIhkaLh_cX5D_XuBga01KK05h-kdtWdkG1fLg8NYqS	Twisted Fate	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	42620	100	itsmevalky#valky	Ru07a9NWs3ovHYE42Nx40V54-W0gaRDPWIBhZEfhTG50fabBSj7G0sOxwh7hjk4OaysD	Caitlyn	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	42621	100	Kill Steal Goat#Pyked	ine-_I_njC3ZWKOU_Frc61saoedTQzmbhZoETWtN-x3v50fvvdj0w01A1gNIhbKu0G6u	Pyke	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	45711	100	70xRight#TRASH	_rSKLzATshHIpju5XPbURipEMRsP9luAM-QkpttjY3HOJtFiikBqvhJW8FNkboGhsuHR	Shen	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	45712	100	IHaveResurrected#NA1	Hh5_YfLS4XXmUgfM4uvXUBwBZWWLjrucIJWon29YZ7MTOZ8J1a63Eh__9Sjpxym_RN16	Diana	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	45713	100	Minushirtus#huh	7xq5NIGVSsu7nasrMCfhDH5VJzND0bSNGLiB6bphYZOV1pwBfjQYvB8oAKTI7LmbdsVe	Twisted Fate	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	45714	100	itsmevalky#valky	_M1HLCMQDOhALCf-Zvevysb-6URYsEC0dITsssDjdvZwQh0-9h1-43rca1q9auh6s_dw	Caitlyn	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	45715	100	Kill Steal Goat#Pyked	AK-JmXSs-EmlsuPy3lL53_3CY2Y2fEYYxFBTgXjCHgsD8zkTjGk9lj4T7LE0hyiX2Xk9	Pyke	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/
NA1_5414994595	42617	100	70xRight#TRASH	TTcCe9jIBntAEc_Ve8yHrTSofegQ97JuHDmw9O2Ugj925pSnjWayMDEP1fUaEzMLfNcb	Shen	https://ddragon.leagueoflegends.com/cdn/15.24.1/img/champion/

Example image of match detail, result of using Views

kills	deaths	assists	kda	win	cs	gold_earned	damage_to_cha	vision_score	items	runes
9	9	4	1.44	false	167	14806	15811	19	[{"name":"Infinity Edge","item_id":3031,"icon_url":"https://ddragon.	[{"name":"First S
3	9	10	1.44	false	176	12481	17606	19	[{"name":"Doran's Blade","item_id":1055,"icon_url":"https://ddragon.	[{"name":"Lethal
12	13	9	1.62	false	28	12798	18708	75	[{"name":"Bloodsong","item_id":3877,"icon_url":"https://ddragon.leag	[{"name":"Hail of
1	7	7	1.14	false	147	9399	10686	22	[{"name":"Titanic Hydra","item_id":3748,"icon_url":"https://ddragon.	[{"name":"Grasp o
7	12	9	1.33	false	217	14419	24598	33	[{"name":"Zhonya's Hourglass","item_id":3157,"icon_url":"https://ddr	[{"name":"Conquer
9	9	4	1.44	false	167	14806	15811	19	[{"name":"Infinity Edge","item_id":3031,"icon_url":"https://ddragon.	[{"name":"First S
3	9	10	1.44	false	176	12481	17606	19	[{"name":"Doran's Blade","item_id":1055,"icon_url":"https://ddragon.	[{"name":"Lethal
12	13	9	1.62	false	28	12798	18708	75	[{"name":"Bloodsong","item_id":3877,"icon_url":"https://ddragon.leag	[{"name":"Hail of
1	7	7	1.14	false	147	9399	10686	22	[{"name":"Titanic Hydra","item_id":3748,"icon_url":"https://ddragon.	[{"name":"Grasp o