

Rift Pulse

Aplicație web pentru pasionații de League of Legends esports

Candidat: Mario Răzvan CIOARA

Coordonator științific: Adriana ALBU-HĂRȘIAN

Sesiunea: Februarie 2026

REZUMAT

League of legends este cel mai popular esport de pe planetă[1].Milioane de oameni se uită săptămânal la competițiile de League of Legends și vor să fie conectați tot timpul la ce se întâmplă. În lipsa unei aplicații oficiale dezvoltată de compania din spatele jocului(Riot Games),fanii recurg la aplicații de alți fani sau alte companii.Aceste soluții nu oferă tot timpul ceea ce fanii vor și de multe ori fanii sunt nevoiți să utilizeze două sau chiar trei aplicații pentru a putea accesa toate informațiile de care au nevoie.Am dezvoltat această aplicație pentru a oferi o soluție completă acestei probleme.

Aplicația Rift Pulse oferă date complete despre principalele turnee din scena competitivă de League of Legends.Pe platformă se pot găsi profile detaliate ale echipelor cu informații despre jucători si rezultatele lor din fiecare turneu în care participă.

Sistemul dispune de un pipeline automat care interoghează API-ul LoL Esports al companiei Riot Games la intervale regulate,menținând baza de date actualizată,cu intervenție manuală puțină.Utilizatorii pot crea conturi cu autentificare bazată pe JWT,își pot alege echipele și meciurile favorite și primesc notificări prin email înainte de începerea meciurilor.

Pentru realizarea proiectului, am dezvoltat un sistem client-server compus dintr-un backend REST API construit cu Django si Django REST Framework, un frontend construit cu React, un sistem de task-uri periodice bazat pe Django-Q și un canal Websocket implementat cu Django Channels.

CUPRINS

1. INTRODUCERE	4
1.1 INFORMAȚII GENERALE.....	4
1.2 IDEEA ȘI SCOPUL PROIECTULUI	4
1.3 STRUCTURA PROIECTULUI	5
1.4 STRUCTURA LUCRĂRII	5
2. TEHNOLOGII FOLOSITE	7
2.1 PYTHON ȘI DJANGO	7
2.2 DJANGO REST FRAMEWORK	7
2.3 AUTENTIFICAREA CU JSON WEB TOKENS(JWT)	8
2.4 DJANGO CHANNELS ȘI WEBSOCKET	9
2.5 SISTEM DE TASK-URI PERIODICE DJANGO-Q.....	10
2.6 POSTGRESQL.....	10
2.7 REACT	11
2.8 BOOTSTRAP 5	12
2.9 AXIOS	12
2.10 API-UL LOL ESPORTS.....	13
3. ARHITECTURA SISTEMULUI.....	14
3.1 ARHITECTURA GENERALĂ CLIENT-SERVER.....	14
3.2 ARHITECTURA BACKEND-ULUI DJANGO	16
3.3 ARHITECTURA FRONTEND-ULUI REACT	17
3.4 PIPELINE-UL AUTOMAT DE DATE	18
3.5 SISTEMUL DE NOTIFICĂRI PRIN EMAIL	19
3.6 COMUNICAREA ÎN TIMP REAL PRIN WEBSOCKET.....	19
4. DETALII DE IMPLEMENTARE	21
4.1 MODELELE DE DATE	21
4.2 IMPLEMENTAREA API-ULUI REST	24
4.3 IMPLEMENTAREA AUTENTIFICĂRII.....	27
4.4 SERVICIUL DE INTEGRARE CU API-UL EXTERN	28
4.5 TASK-URILE PERIODICE	29
4.6 SEMNALELE DJANGO ȘI BROADCAST-UL WEBSOCKET.....	30
4.7 IMPLEMENTAREA FRONTEND-ULUI REACT	30
5.UTILIZAREA APLICAȚIEI.....	34
5.1 DESCRIEREA APLICAȚIEI.....	34
5.2 SCENARII DE UTILIZARE	34
5.3 LANSARE ÎN PRODUCȚIE.....	51
6. CONCLUZII	53
7. BIBLIOGRAFIE	55

1. INTRODUCERE

1.1 INFORMAȚII GENERALE

Jocul League of Legends, dezvoltat de compania Riot Games, este cel mai popular esport de pe planetă[1]. Milioane de oameni urmăresc în fiecare săptămână competițiile de League of Legends care se organizează peste tot în lume. Cele mai urmărite competiții sunt cele din regiunile principale: LCK(Coreea de Sud), LPL(China), LCP (Taiwan,Hong Kong,Australia,Japonia,Filipine), LEC(Europa) și LCS(America de Nord și Centrală).

Fiecare regiune are propriul calendar competițional, cu trei tururi(split-uri) principale pe an(iarnă, primăvară, vară). La finalul fiecărui tur se organizează o competiție internațională unde sunt invitate cele mai bune echipe din fiecare regiune în turul cel mai recent terminat. Turneul First Stand se organizează după încheierea turului de iarnă, unde sunt invitate campioana și vicecampioana din LCK și LPL(LCK și LPL sunt regiunile cele mai bune din istoria acestui esport și de aceea sunt invitate 2 echipe din fiecare regiune) și campioana din celelalte regiuni. MSI(Mid Season Invitational) se organizează după încheierea turului de primăvară unde se reunesc campioanele și vicecampioanele din toate regiunile. După încheierea turului din vară se organizează Worlds(Campionatul Mondial) unde sunt invitate cele mai bune patru echipe din LCK și LPL și cele mai bune trei echipe din LEC, LCS și LCP.

Popularitatea jocului nu a adus și la crearea unei platforme oficiale care să satisfacă cerințele fanilor. Dezvoltatorul jocului a preferat să lase acest aspect în mână fanilor sau al altor companii. Informațiile despre echipe, meciuri, jucători și turnee sunt dispersate pe mai multe platforme precum lolesports.com, liquidpedia.net, lol.fandom.com și altele.

1.2 IDEEA ȘI SCOPUL PROIECTULUI

Idea proiectului Rift Pulse mi-a venit deoarece sunt un mare fan al jocului și al esportului League of Legends. Joc League of Legends și urmăresc competițiile de peste 16 ani. În toți acești ani am utilizat multe platforme pentru a sta la curent cu rezultatele meciurilor și statisticile jucătorilor și echipelor. Unele platforme au dispărut deoarece erau create de fani care nu dispuneau de fonduri pentru a întreține platforma sau nu mai aveau timp să se ocupe de menținerea ei. Aceste platforme foarte rar produc și bani, iar majoritatea celor care se implică o fac voluntar, din pasiune.

Obiectivele proiectului sunt:

- Agregarea datelor despre echipe(prezentare generală a fiecărei echipe și rezultate),jucători(profil,statistici,echipă curentă),turnee(clasamente,meciuri) și meciuri(programurile meciurilor,rezultate,link-uri spre VOD-uri)
- Actualizarea automată prin implementarea unui pipeline automat care folosește API-ul LoL Esports pentru a menține baza de date actualizată cu cele mai recente actualizări.
- Permitearea utilizatorului de a-și crea cont,să-și marcheze echipele și meciurile favorite și să primească notificări prin email înainte de începerea meciurilor echipelor favorite.
- Interfață modernă,receptivă,cu suport pentru tema dark/light,filtre de căutare avansate și o experiență de navigare fluidă.

1.3 STRUCTURA PROIECTULUI

Proiectul a fost structurat în două componente principale,corespunzătoare arhitecturii client-server.

Backend-ul(Django) reprezintă serverul aplicației și include modelele de date(ORM Django),API-ul REST(Django Framework cu ViewSets și serializare), sistemul de autentificare (JWT cu SimpleJWT), serviciile de integrare cu API-ul extern (client HTTP,mapper de date), task-urile periodice (fetch meciuri,notificări prin email),comunicare websocket(django channels) și panoul de administrare (Django Admin).

Frontend-ul reprezintă interfața utilizatorului și include componentele de navigație și layout (Navigation,Footer), context providers (AuthContext, RegionContext, Theme Context), doisprezece pagini (Home,Regions,Teams, TeamDetail, Players, PlayerDetail, Tournaments, Matches, Login, Register, Profile) și stilizare cu Bootstrap 5 și CSS.

1.4 STRUCTURA LUCRĂRII

Capitolele acestei lucrări sunt structurate astfel:

- Capitolul 2 prezintă în detaliu fiecare tehnologie folosită în dezvoltarea proiectului: Python și Django pentru backend,Django REST Framework pentru API,SimpleJWT pentru autentificare,Django Channels pentru websocket,Django-Q pentru task-uri periodice,React pentru frontend,Bootstrap pentru stilizare,Axios pentru comunicarea HTTP și API-ul LoL Esports ca sursă externă de date,PostgreSQL pentru baza de date.
- Capitolul 3 prezintă arhitectura generală client-server a aplicației,cu diagrame ale modulelor funcționale,arhitectura backend-ului Django cu toate componentele

sale, arhitectura frontend-ului React cu sistemul de context providers și pipeline-ul automat de date.

- Capitolul 4 este cel mai amplu capitol și prezintă în detaliu implementarea fiecărei componente: modelele de date cu structura câmpurilor, API-ul REST cu ViewSets și serializare, sistemul de autentificare JWT, serviciul de integrare cu API-ul extern (client HTTP, mapper, aliasuri), task-uri periodice, semnale Django, fiecare pagină și componentă din frontend și panoul de administrare.
- Capitolul 5 prezintă aplicația din perspectiva utilizatorului și descrierea interfeței și procesul de lansare în producție pe platforma Railway.com cu domeniul personalizat rift-pulse.com.
- Capitolul 6 prezintă concluziile

2. TEHNOLOGII FOLOSITE

2.1 PYTHON ȘI DJANGO

Python este un limbaj de programare de nivel înalt, interpretat, cu o sintaxă clară și expresivă. Este unul dintre cele mai populare limbaje de programare pentru dezvoltare web, analiza datelor și automatizare[2]. În cadrul proiectului am folosit Python 3.X ca limbaj principal pentru întreaga logică de backend.

Django este un framework web de nivel înalt, scris în Python, care încurajează dezvoltarea rapidă și un design curat și fluid. Django urmează modelul arhitectural MVT (Model-View-Template), o variantă a clasicului MVC (Model-View Controller). Principalele caracteristici ale Django pe care le-am utilizate în proiect sunt:

- **ORM (Object-Relational Mapping):** Django oferă un ORM puternic care permite definirea modelelor de date ca clase Python și interacțiunea cu baza de date fără a scrie SQL direct. Modele sunt definite în fișierul `models.py` și sunt mapate automat la tabele din baza de date. Operații precum filtrare (`.filter()`), excludere (`.exclude()`), ordonare (`.order_by()`), join-uri (`.select_related()`, `.prefetch_related()`) și agregări sunt realizate prin metode Python.
- **Sistemul de migrații:** Django generează automat fișiere de migrație care descriu modificările aduse modelelor de date. Aceste migrații pot fi aplicate secvențial pentru a actualiza schema bazei de date.
- **Panoul de administrare:** Django Admin oferă o interfață web completă pentru gestionarea datelor din baza de date, fără a fi necesară scrierea de cod suplimentar.
- **Sistemul de semnale:** Django oferă un mecanism de semnale (signals) care permite componentelor decuplate să fie notificate când au loc anumite acțiuni.
- **Middleware:** Django utilizează un sistem de middleware prin care cererile HTTP trec înainte de a ajunge la views. În proiect sunt folosite `SecurityMiddleware`, sesiuni (`SessionMiddleware`), CSRF protection (`CsrfViewMiddleware`), autentificare (`AuthenticationMiddleware`) și CORS (`CorsMiddleware`).
- **Comenzi de management:** Django permite crearea de comenzi personalizate care pot fi executate din linia de comandă. În proiect am creat două comenzi: `fetch_matches_now` (pentru fetch-ul manual al datelor) și `setup_scheduled_tasks` (pentru configurarea task-urilor periodice)

2.2 DJANGO REST FRAMEWORK

Django REST Framework (DRF) este o bibliotecă puternică și flexibilă pentru construirea API-urilor web cu Django. DRF adaugă un strat de abstractizare peste

Django, oferind mecanisme specializate pentru serializarea datelor, autentificarea cererilor și construirea de endpoint-uri RESTful.[3]

Principalele componente DRF utilizate în proiect sunt:

- **ModelSerializer:** Clase care transformă automat instanțele modelelor Django în reprezentări JSON și invers. În proiect am folosit 9 serializere: `GameSerializer`, `TeamSerializer`, `TeamDetailSerializer`, `PlayerSerializer`, `TournamentSerializer`, `TournamentDetailSerializer`, `MatchSerializer`, `RegisterSerializer` și `UserSerializer`. Fiecare serializer specifică câmpurile care sunt incluse în răspunsul JSON și poate defini câmpuri computate sau câmpuri relaționale.
- **ModelViewSet:** Clase care oferă automat implementări complete CRUD (Create, Read, Update, Delete) pentru un model. `ViewSet`-urile pe care le-am utilizat în proiect (`GameViewSet`, `TeamViewSet`, `PlayerViewSet`, `TournamentViewSet`, `MatchViewSet`) sunt înregistrate într-un router DRF care generează automat URL-urile pentru standard REST.
- **DefaultRouter:** Componenta DRF care generează automat URL-urile pentru fiecare `ViewSet` înregistrat (list, detail, custom actions). De exemplu înregistrarea `TeamViewSet` sub prefixul `teams` generează automat `/api/teams`, `/api/teams/{id}/` și orice acțiuni custom definite cu decoratorul `@action`.
- **Custom actions:** Metode suplimentare definite pe `ViewSets` cu decoratorul `@action`. În proiect, `TeamViewSet` definește acțiunea `tournament_results` (care calculează statisticile de win/loss per turneu), iar `TournamentViewSet` definește acțiunea `standings` (care calculează clasamentele bazate pe rezultatele meciurilor).
- **Permission classes:** DRF are clase de permisiuni care controlează accesul la endpoint-uri. Configurația implicită a proiectului este `AllowAny`, cu endpoint-urile de autentificare protejate individual prin decoratorul `@permission_classes`.

2.3 AUTENTIFICAREA CU JSON WEB TOKENS (JWT)

JSON Web Token (JWT) este un standard deschis care definește o modalitate compactă și autonomă de transmitere a informațiilor între părți sub forma unui obiect JSON. În contextul aplicațiilor web, JWT este utilizat frecvent pentru autentificarea și autorizarea utilizatorilor.

Un JWT este compus din trei părți, separate prin puncte:

- **Header:** conține tipul tokenului și algoritmul de semnare
- **Payload:** conține informații despre utilizator (`user_id`, `username`) și metadata (data expirării, data emiterii)
- **Signature:** O semnătură criptografică care verifică integritatea tokenului.

În proiect, autentificarea JWT este implementată prin biblioteca `django-rest-framework-simplejwt`, care pune la dispoziție:

- Două tipuri de tokenuri: access token de scurtă durată(60 de minute) care este trimis cu fiecare cerere API în header-ul Authorization: Bearer token și refresh token de lungă durată(7 zile) folosit pentru obținerea unui nou access token când cel curent expiră.
- Rotirea tokenurilor de refresh: La fiecare utilizare a refresh tokenului pentru obținerea unui nou access token,se generează și un nou refresh token, iar cel vechi este invalidat.
- Blacklist-ul tokenurilor: Modulul `token_blacklist` menține o listă de refresh a tokenurilor invalidate.Aceasta listă previne reutilizarea tokenurilor vechi.

Flow-ul complet de autentificare în aplicație este:

- Utilizatorul se înregistrează sau se autentifică,primește access și refresh token
- Tokenurile sunt stocate în `localStorage` pe frontend
- Fiecare cerere API include access token în header-ul Authorization
- Când access tokenul expiră,un interceptor Axios trimite automat refresh token-ul pentru obținerea unui nou access token
- Cererea originală este retrimisă cu noul token
- La logout,refresh token-ul este trimis la server pentru blacklisting

2.4 DJANGO CHANNELS ȘI WEBSOCKET

Django Channels este o extensie a framework-ului Django care adaugă suport pentru protocoale asincrone,în special WebSocket.În mod normal,Django funcționează prin protocolul WSGI(Web Server Gateway Interface),care este sincron și suportă doar cereri HTTP request-response.Channels înlocuiește WSGI cu ASGI (Asynchronous Server Gateway Interface),permițând:

- Conexiuni persistente(WebSocket)
- Comunicare bidirecțională în timp real
- Broadcasting către grupuri de clienți conectați.

Protocolul WebSocket oferă un canal de comunicare full-duplex peste o singură conexiune TCP.Spre deosebire de HTTP,unde clientul trebuie să inițieze fiecare cerere,WebSocket permite serverului să trimită date către client în orice moment după stabilirea conexiunii.

În proiect,Django Channels este utilizat pentru a trimite actualizările meciurilor în timp real.Componentele care fac asta sunt:

- ASGI Application(`asgi.py`): Configurează rutarea protocoalelor,cererile HTTP sunt direcționate către aplicația Django standard, iar conexiunile WebSocket către `MatchUpdatesConsumer` pe ruta `ws/matches/`.

- **MatchUpdatesConsumer:** Un consumer async(AsyncJsonConsumer) care gestionează conexiunile WebSocket. La conectare, clientul este adăugat în grupul matches. La deconectare, este eliminat din grup. Când primește un mesaj de tip match_update de la channel layer, îl trimite clientului ca JSON.
- **Channel Layer:** InMemoryChannelLayer este folosit în proiect.
- **Semnalul post_save:** Când un Match este salvat în baza de date, un semnal Django apelează async_to_sync (channel_layer.group_send) pentru a trimite actualizarea către toți clienții din grupul matches.

2.5 SISTEM DE TASK-URI PERIODICE DJANGO-Q

Django-Q este un sistem de task-uri distribuite pentru Django, care permite programarea și executarea de task-uri de background la intervale regulate. Este similar cu Celery, dar mai ușor de configurat și cu suport nativ pentru ORM-ul Django ca broker de mesaje.

Principalele caracteristici ale Django-Q utilizate în proiect:

- **Schedule:** Obiecte ORM care definesc task-uri periodice. Fiecare Schedule specifică funcția de executat, tipul de programare (minute, ore, zilnic), intervalul și numărul de repetiții (-1 pentru infinit).
- **Cluster(qcluster):** Un proces separat care monitorizează Schedule-urile și execută task-urile la momentul programat. Configurat cu 2 workeri, timeout de 60 secunde și retry după 120 de secunde.
- **ORM Broker:** În loc să necesite un serviciu extern Django-Q poate folosi baza de date Django existentă pentru stocarea și gestionarea cozii de task-uri.

În proiect sunt configurate trei task-uri periodice:

- **fetch_match_schedules** care la fiecare 15 minute: Interoghează API-ul LoL Esports pentru programările meciurilor de LoL din cele 5 regiuni.
- **fetch_match_results** care la fiecare 5 minute: Interoghează API-ul LoL Esports pentru detalii despre meciuri.
- **send_match_notifications** care la fiecare 5 minute verifică meciurile care urmează să înceapă și trimite notificări prin email utilizatorilor.

2.6 POSTGRESQL

PostgreSQL este un sistem de gestiune a bazelor de date relațional-obiectuală, open source, cunoscut pentru fiabilitate, robustețe și suport pentru funcții avansate. Este una dintre cele mai utilizate baze de date în ecosistemul Django.[4]

Proprietățile PostgreSQL relevante pentru acest proiect sunt:

- **JSONField nativ:** PostgreSQL oferă tipul de date json care permite stocarea, indexarea și interogarea datelor JSON direct în baza de date. În proiect, câmpurile stats (pe Match și Player) și social_media (pe Team și Player) folosesc JSONField,

permițând stocarea de structuri de date flexibile fără a necesita migrații suplimentare la modificarea schemei JSON.

- Indexarea și constrângerile: Câmpul `external_id` pe modelul `Match` este definit ca `unique=True`, ceea ce creează un index unic în PostgreSQL. Această constrângere asigură că nu pot exista două meciuri cu același ID extern, permițând operații de `upsert` eficiente.
- Tranzacții ACID: PostgreSQL garantează proprietățile ACID (Atomicity, Consistency, Isolation, Durability) pentru toate operațiile, ceea ce este esențial pentru integritatea datelor în contextul task-urilor concurente de `fetch` care pot actualiza același meci simultan.
- `ManyToManyField`: Relațiile many-to many (`Tournament`, `participants`, `UserProfile`, `favorite_teams`, `UserProfile.favorite_matches`) sunt implementate prin tabele intermediare generate automat de Django ORM.

Baza de date a proiectului se numește `esportsdb` și rulează pe `localhost`, portul 5432, configurată în fișierul `settings.py`

2.7 REACT

React este o bibliotecă JavaScript open-source dezvoltată de Meta (Facebook) pentru construirea interfețelor utilizatorilor. React se bazează pe conceptul de componente reutilizabile și pe un DOM virtual care optimizează actualizările interfeței prin minimizarea manipulărilor directe ale DOM-ului real [5].

Versiunea React utilizată în proiect este 19.1, cea mai recentă versiune majoră, care oferă îmbunătățiri de performanță și noi API-uri pentru managementul stării.

Principalele concepte React utilizate în proiect:

- Componente funcționale: Toate componentele din proiect sunt componente funcționale, nu componente bazate pe clase. Această abordare modernă permite utilizarea hook-urilor și rezultă în cod mai concis.
- Hooks:
 - `useState`: Pentru gestionarea stării locale a componentelor (de exemplu, lista de meciuri, starea de `loading`, filtrul selectat).
 - `useEffect`: Pentru efecte secundare precum `fetch`-ul de date din API la montarea componentei, actualizarea `localStorage` sau `debounce`-ul căutării.
 - `useContext`: Pentru accesarea datelor din Context Providers (`AuthContext`, `RegionContext`, `ThemeContext`) fără a transmite props prin mai multe nivele de componente.
 - `useRef`: Pentru păstrarea referințelor la timere de `debounce` între randări, fără a cauza re-randări inutile.
 - `useParams`: Hook de React Router extragerea parametrilor dinamici din URL (de exemplu `:id` din `/teams/:id`).
 - `useNavigate`: Hook de React Router pentru navigarea programatică.

-useLocation: Hook de React Router pentru accesarea stării de navigare(de exemplu filtrul de regiune transmis de la pagina Regions)

- Context API: React Context oferă un mecanism de a transmite date prin arborele de componente fără a fi necesar să se transmită props manual la fiecare nivel.În proiect sunt definite trei contexte:
 - ThemeContext: Gestionează tema dark/light, persistată în local storage.
 - RegionContext: Gestionează regiunea selectată,persistată în localStorage.
 - AuthContext: Gestionează întreaga stare de autentificare, tokenurile JWT,datele utilizatorului,favoritele și operațiile de login/register/logout.
- React Router: Biblioteca de rutare client-side care permite navigarea între pagini fără reîncărcarea completă a aplicației.Suportă rute cu parametri dinamici (/teams/:id), navigarea programatică(useNavigate) și transmiterea stării între pagini.

Aplicația frontend a fost inițializată cu Createa React App, care oferă configurația de build, server de dezvoltare cu hot reloading și optimizări pentru producție.

2.8 BOOTSTRAP 5

Bootstrap este cel mai populat framework CSS pentru construirea de interfețe web responsive.Versiunea 5 elimină dependența de jQuery și introduce suportul nativ pentru teme dark/light prin atributul data-bs-theme[6].

În proiect Bootstrap 5.3.6 este utilizat pentru:

- Sistemul de grid responsive(container,row,col-md-*,col-lg-*).
- Componentele UI (card, badge, table, navbar, nav-tabs, alert, form-select, spinner-border,breadcrumb,btn,input-group).
- Utilitățile de spațiere și aliniere(mb-3, d-flex, gap-2, text-center, justify-content-between).
- Temele dark/light(data-bs-theme pe document.body)
- Clasele de text(text-muted, text-success, text-danger, fw-bold)

Bootstrap Icons sunt de asemenea utilizate pentru iconițe din interfață(bi-star, bi-star-fill, bi-calendar-event, bi-trophy, bi-search, bi-person-circle, bi-box-arrow-in-right, bi-eye, bi-eye-slash etc.)

2.9 AXIOS

Axios este un client HTTP bazat pe Promise-uri pentru JavaScript,utilizat atât în browser cât și în Node.js[7].În proiect Axios este utilizat pentru toate comunicările HTTP între frontend-ul React și backend-ul Django.

Facilitățile Axios utilizate în proiect:

- Interceptori de răspuns: Un interceptor global este configurat în AuthContext care interceptează răspunsurile 401 (Unauthorized) și încearcă automat refresh-ul tokenului JWT înainte de a re-trimite cererea originală.
- Default headers: După autentificare, header-ul Authorization este setat global pe instanța Axios (axios.defaults.headers.common['Authorization']), astfel încât toate cererile ulterioare includ automat tokenul JWT.
- Cereri concurente cu Promise.all: În paginile de detaliu (TeamDetail, TournamentDetail, PlayerDetail, Profile), mai multe cereri API sunt executate în paralel folosind Promise.all pentru a reduce timpul total de încărcare.

2.10 API-UL LOL ESPORTS

API-ul LoL Esports este un API neoficial care alimentează site-ul oficial lolesports.com. Acest API oferă acces la date complexe despre competițiile profesionale de League of Legends, incluzând programări de meciuri, rezultate, detalii pe jocuri și link-uri pentru VOD.

Caracteristicile API-ului:

- URL de bază: <https://esports-api.lolesports.com/persisted/gw/>
- Autentificare: O cheie API publică, transmisă prin header-ul x-api-key
- Format: JSON
- Parametru de limbă: hl=en-US (limba engleza)
-

Endpoint-urile principale utilizate în proiect:

- getLeagues: Returnează lista tuturor ligilor cu ID-urile lor. ID-urile ligilor sunt constante.
- getSchedule: Returnează programările meciurilor pentru o ligă, inclusiv meciuri recente și viitoare. Suportă paginare prin parametrul pageToken. Datele returnate includ: echipe, scor, stare, block name (saptamana/ziua), strategie (Bo1/Bo3/Bo5) și timestamp-ul de start.
- getEventDetails: Returnează informații detaliate pentru un singur meci, inclusiv datele pentru joc (echipe cu side blue/red, stare pentru joc, link-uri VOD cu YouTube video ID). Acest endpoint oferă date mult mai bogate decât getSchedule, dar necesită un apel separat pentru fiecare meci.
- getTournamentsForLeague: Returnează metadatele turneele pentru o ligă.
- getCompletedEvents: Returnează evenimentele finalizate pentru un turneu.

Este important de menționat că acest API este neoficial și nu are o documentație publică oficială. Endpoint-urile și structura datelor au fost descoperite prin analiza traficului de rețea al site-ului lolesports.com. API-ul poate suferi modificări fără preaviz, motiv pentru care implementarea din proiect include mecanisme robuste de tratare a erorilor.

3. ARHITECTURA SISTEMULUI

3.1 ARHITECTURA GENERALĂ CLIENT-SERVER

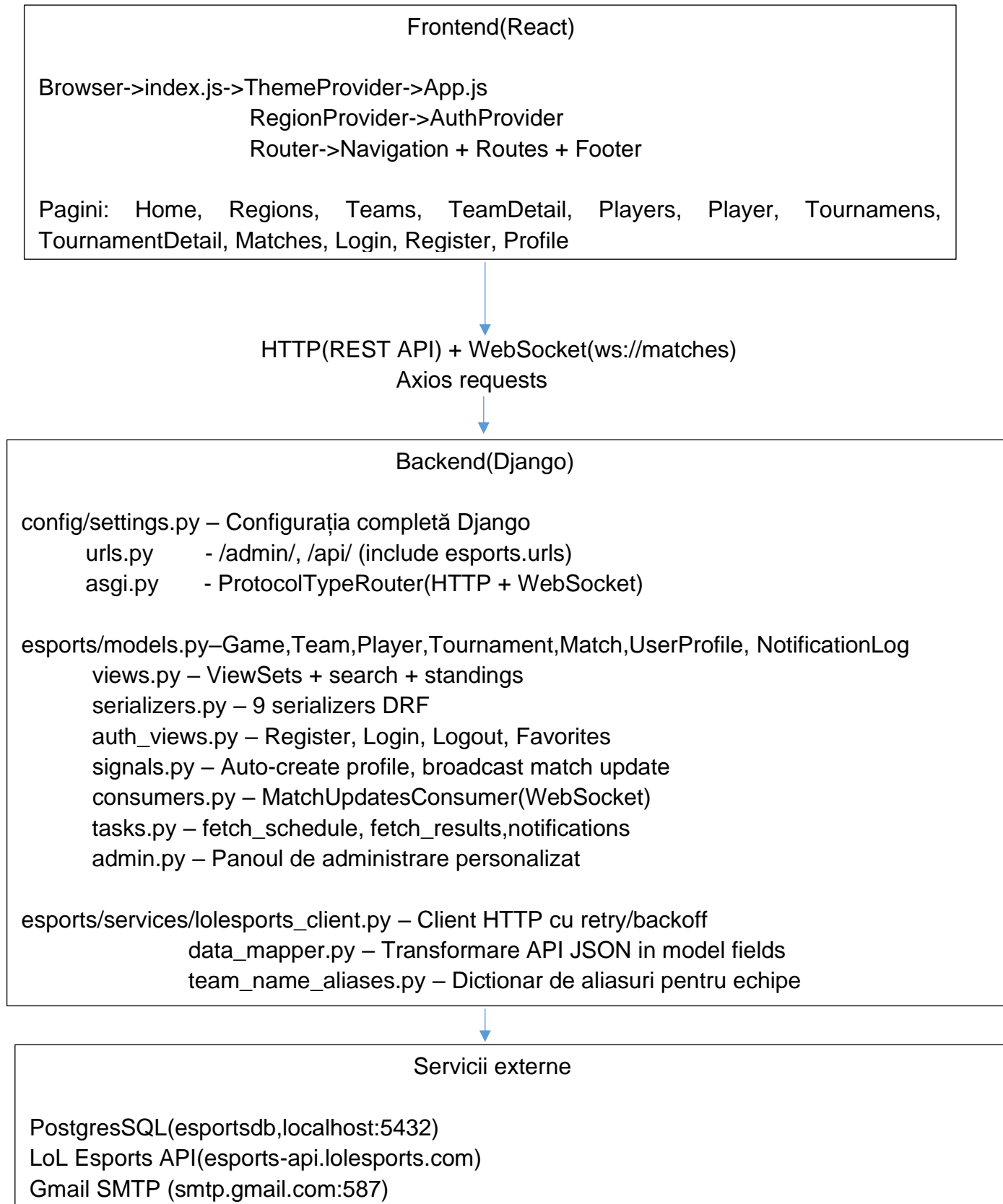
Aplicația Rift Pulse urmează o arhitectură client-server clasică, cu o separare clară între frontend (clientul) și backend (serverul).

Componenta client este o aplicație React de tip SPA (Single Page Application) care rulează în browserul utilizatorului. Aceasta comunică cu serverul exclusiv prin două protocoale:

- HTTP/HTTPS: Pentru cererile REST API standard (GET, POST, PUT, DELETE). Toate cererile sunt inițiate de client și urmează modelul request-response.
- WebSocket: Pentru primirea actualizărilor în timp real. După stabilirea conexiunii, serverul poate trimite date către client fără ca acesta să le solicite explicit.

Componenta server este o aplicație Django care expune un API REST și un endpoint WebSocket. Serverul interacționează cu trei servicii externe:

- PostgreSQL: Baza de date relațională pentru stocarea persistentă a tuturor datelor aplicației.
- API-ul LoL Esports: Sursa externă de date pentru programările și rezultatele meciurilor. Serverul interoghează periodic acest API prin task-uri de background.
- Gmail SMTP: Serviciul de email pentru trimiterea notificărilor către utilizatori.



3.2 ARHITECTURA BACKEND-ULUI DJANGO

Backend-ul Django este organizat în două pachete Python:

- `config/`: Pachetul de configurare al proiectului Django, care conține `settings.py` (toate setările), `urls.py` (rutele de nivel superior), `asgi.py` (configurația ASGI) și `wsgi.py` (configurația WSGI).
- `esports/`: Aplicația Django principală care conține toată logica de business.

Modulele funcționale ale backend-ului sunt:

a) Stratul de date (Models + ORM):

Fișierul `models.py` definește 7 modele care sunt mapate la tabele PostgreSQL. Relațiile dintre modele includ: `ForeignKey` (`Match->Tournament`, `Match->Team`, `Player->Team`, `Team->Game`, `Tournament->Game`), `ManyToManyField` (`Tournament.participants`, `UserProfile.favorite_teams`, `UserProfile.favorite_matches`) și `OneToOneField` (`Player->User`, `UserProfile->User`).

b) Stratul API (Views + Serializers + URLs):

Cererile HTTP sunt redirectionate prin `urls.py` către `ViewSets` (definite în `views.py`) care procesează cererea, interacționează cu modelele și returnează răspunsuri JSON serializate prin serializerele din `serializers.py`. Routerul DRF generează automat URL-urile standard CRUD.

c) Stratul de autentificare (Auth Views + SimpleJWT):

Endpoint-urile de autentificare sunt definite separat în `auth_views.py` și folosesc SimpleJWT pentru generarea și validarea tokenurilor JWT.

d) Stratul de servicii externe (`services/`):

Pachetul `services/` încapsulează toată logica de comunicare cu API-ul LoL Esports. Include clientul HTTP (`lolesports_client.py`), mapperul de date (`data_mapper.py`) și tabelul de aliasuri (`team_name_aliases.py`).

e) Stratul de task-uri de background (`tasks.py` + Django-Q):

Funcțiile din `tasks.py` sunt executate periodic de Django-Q. Acestea operează independent de cererile HTTP, interacționând direct cu modelele și cu serviciul de API extern.

- f) Stratul de comunicare in timp real (Channels + Consumers + Signals):
Semnalele Django declanseaza broadcast-uri WebSocket prin channel layer catre consumerii conectati.
- g) Panoul de administrare (admin.py):
Configurează interfața Django Admin cu formulare personalizate, filtre și câmpuri de căutare pentru fiecare model.

3.3 ARHITECTURA FRONTEND-ULUI REACT

Frontend-ul urmează o arhitectură bazată pe componente, cu trei straturi de context providers care oferă stare globală tuturor componentelor:

Ierarhia de context providers (de la exterior la interior):

1. ThemeProvider (ThemeContext.jsx):
 - Cel mai exterior provider, definit in index.js
 - Gestionează tema dark/light
 - Persistă selecția în localStorage
 - Setează atributul data-bs-theme pe document.body
 - Expune: theme, setTheme, toggleTheme
2. RegionProvider (RegionContext.jsx):
 - Definit în App.js
 - Gestionează regiunea selectată (all, europe, north_america, china, south_korea, apac)
 - Persistă selecția in localStorage
 - Expune: selectedRegion, setSelectedRegion, getSelectedRegionData, clearRegion, regions (lista completă)
3. AuthProvider (AuthContext.jsx):
 - Cel mai interior provider, definit in App.js
 - Cel mai complex context, gestionează:
 - Starea utilizatorului (user, loading, isAuthenticated)
 - Tokenurile JWT (stocate in localStorage)
 - Interceptorul Axios pentru refresh automat
 - Funcțiile de autentificare (login, register, logout, deleteAccount)
 - Funcțiile de favorite (toggleFavoriteTeam, toggleFavoriteMatch)

Paginile aplicației sunt organizate în directorul `pages/` și sunt randate condiționat de React Router în funcție de URL-ul curent. Fiecare pagină este o componentă funcțională care:

1. Declară starea locală cu `useState` (date, loading, error, filtre)
2. Execută `fetch`-ul de date în `useEffect` la montare
3. Aplică filtre și sortări pe datele primite
4. Randează interfața cu componente Bootstrap

Componentele partajate (Navigation, Footer) sunt randate în afara sistemului de rute, fiind vizibile pe toate paginile.

3.4 PIPELINE-UL AUTOMAT DE DATE

Pipeline-ul automat de date este responsabil cu menținerea bazei de date actualizate cu cele mai recente informații despre meciuri. Acesta funcționează în două faze:

Faza 1 - Fetch-ul programărilor (`fetch_match_schedules`):

Executat la fiecare 15 minute prin Django-Q.

Fluxul de date:

1. Se iterează prin cele 5 ID-uri de ligi
2. Se apelează `getSchedule` pentru fiecare liga
3. Se identifică turneul local corespunzător (prin căutare după nume)
4. Pentru fiecare eveniment din răspuns:
 - a. `DataManager.map_schedule_event()` extrage campurile relevante
 - b. `DataManager.resolve_team()` pune numele echipelor la obiectul `Team`
 - c. `_derive_winner()` determină câștigătorul din scor
 - d. `Match.objects.update_or_create(external_id=...)` face upsert

Faza 2 – Actualizarea rezultatelor (`fetch_match_results`):

Executat la fiecare 5 minute prin Django-Q.

Fluxul de date:

1. Se identifică meciurile care necesită actualizare:
 - Categoria 1: Meciuri din trecut care nu sunt încă marcate ca finalizate
 - Categoria 2: Meciuri finalizate cărora le lipsesc detaliile per joc
2. Pentru fiecare meci:
 - a. Se apelează `getEventDetails` cu `external_id`-ul meciului

b. `Mapper.map_event_details()` extrage datele detaliate:

- Scor final, link VOD stare derivată din stările per joc
- Detalii per joc: număr, stare, echipe cu side (blue/red), VOD-uri

c. Se actualizează Match-ul doar dacă datele s-au schimbat

3.5 SISTEMUL DE NOTIFICĂRI PRIN EMAIL

Sistemul de notificări trimite emailuri personalizate utilizatorilor înainte de începerea meciurilor care îi interesează. Funcționează prin task-ul periodic `send_match_notifications`, executat la fiecare 5 minute.

Două ferestre de notificare:

1. Fereastra de 2 ore (echipe favorite):

- Se identifică meciurile care încep în intervalul `[acum + 2h, acum + 2h5m]`
- Pentru fiecare meci, se găsesc utilizatorii care au marcat ca favorită cel puțin una dintre cele două echipe
- Se trimite un email cu subiectul "Upcoming Match: TeamA vs TeamB"

2. Fereastra de 10 minute (meciuri favorite):

- Se identifică meciurile care încep în intervalul `[acum + 10m, acum + 15m]`
- Se găsesc utilizatorii care au marcat ca favorit acel meci specific
- Se trimite un email cu subiectul "Match Starting Soon: TeamA vs TeamB"

Modelul `NotificationLog` înregistrează fiecare notificare trimisă cu o constrângere `unique_together` pe (user, match, notification_type). Înainte de trimiterea fiecărui email, se verifică dacă există deja un log pentru combinația respectivă.

3.6 COMUNICAREA ÎN TIMP REAL PRIN WEBSOCKET

Când un obiect Match este salvat în baza de date (creat sau actualizat), următorul lanț de evenimente se declanșează:

`Match.save()`

- Django emite semnalul `post_save`
- `broadcast_match_update()` este invocat
- `channel_layer.group_send("matches", {type: "match_update", ...})`
- `MatchUpdatesConsumer.match_update(event)`
- `send_json(content)` către fiecare client WebSocket conectat

Mesajul JSON trimis fiecărui client conține:

```
{  
  "match_id": 42,  
  "external_id": "110853167613028940",  
  "team1": "T1",  
  "team2": "Gen.G Esports",  
  "score": "2-1",  
  "state": "completed",  
  "date_time": "2026-02-08T12:00:00+00:00"  
}
```

Mecanismul este proiectat pentru reziliență: întreaga operație de broadcast este încadrată într-un bloc try/except care prinde toate excepțiile. Dacă channel layer-ul nu este disponibil sau conexiunea WebSocket eșuează, excepția este logată la nivel DEBUG și Match-ul este salvat normal în baza de date.

4. DETALII DE IMPLEMENTARE

4.1 MODELELE DE DATE

Modelele de date sunt definite în fișierul `esports/models.py` și sunt mapate la tabele PostgreSQL prin ORM-ul Django. Proiectul conține 7 modele, fiecare cu rolul său specific în cadrul aplicației. Istoricul migrațiilor cuprinde 9 fișiere, de la `0001_inițial` până la `0009_match_external_id`.

Modelul `Game` reprezintă un joc video (în această aplicație, `League of Legends`). Servește ca punct de referință pentru echipe și turnee.

Câmpuri:

- `name` (`CharField`, `max_length=100`): Numele jocului
- `description` (`TextField`): Descrierea jocului
- `release_date` (`DateField`): Data lansării oficiale
- `developer` (`CharField`, `max_length=100`): Dezvoltatorul jocului
- `publisher` (`CharField`, `max_length=100`): Editorul jocului
- `logo` (`ImageField`, `upload_to='game_logos/'`): Logo-ul jocului (opțional)

Exemplu de utilizare:

```
game = Game.objects.get(name='League of Legends')
```

Modelul `Team` reprezintă o echipă profesionistă de esport. Este unul dintre cele mai referențiate modele, fiind legat de `Player`, `Tournament`, `Match` și `UserProfile`.

Câmpuri:

- `name` (`CharField`, `max_length=100`): Numele echipei (ex: "T1")
- `logo` (`URLField`, `max_length=1000`): URL-ul logo-ului echipei. Am ales `URLField` în loc de `ImageField` pentru a suporta atât URL-uri externe cât și căi locale de media.
- `founded_date` (`DateField`): Data fondării organizației
- `country` (`CharField`, `max_length=100`): Țara de origine
- `region` (`CharField`, `max_length=20`, `choices=REGION_CHOICES`): Diviziunea regională, cu opțiunile: 'europe', 'north_america', 'china', 'south_korea', 'apac'
- `game` (`ForeignKey -> Game`): Jocul în care echipa concurează
- `description` (`TextField`): Biografia echipei, cuprinzând istoricul organizației, realizările și jucatorii notabili
- `social_media` (`JSONField`): Link-uri social media într-un format JSON flexibil, ex: `{"twitter": "https://twitter.com/T1LoL", "website": "https://t1.gg"}`

Decizia de design de a folosi JSONField pentru social_media si nu câmpuri separate permite adăugarea ușoară de noi platforme sociale fără a modifica schema bazei de date.

Modelul Tournament reprezintă un turneu sau split competițional.

Câmpuri:

- name (CharField, max_length=200): Numele turneului (ex: "LCK 2026 Split 1")
- game (ForeignKey -> Game): Jocul pentru care se desfasoara turneul
- start_date (DateTimeField): Data de început
- end_date (DateTimeField): Data de sfârșit
- prize_pool (DecimalField, max_digits=12, decimal_places=2): Fondul de premii in USD
- location (CharField, max_length=100): Locația geografică
- format (CharField, max_length=100): Formatul competitiei
- region (CharField, max_length=20, choices=REGION_CHOICES): Regiunea principală
- participants (ManyToManyField -> Team): Echipele participante. Relatia many-to-many permite unei echipe să participe la mai multe turnee și unui turneu să aibă mai multe echipe.
- status (CharField, max_length=10): Statusul turneului: 'upcoming', 'ongoing' sau 'completed'.

Modelul Match este cel mai complex model din aplicatie, reprezentând un meci (serie) între două echipe în cadrul unui turneu.

Câmpuri:

- tournament (ForeignKey -> Tournament): Turneul caruia ii apartine meciul
- team1 (ForeignKey -> Team, related_name='team1_matches'): Prima echipa
- team2 (ForeignKey -> Team, related_name='team2_matches'): A doua echipă.related_name-uri diferite sunt necesare pentru a preveni conflicte de reverse relation pe modelul Team.
- date_time (DateTimeField): Data si ora programată
- result (CharField, max_length=100): Numele echipei câștigătoare (ex: "T1") sau string gol pentru meciuri viitoare
- score (CharField, max_length=50): Scorul seriei (ex: "2-1", "3-0")
- vod_link (URLField): Link YouTube către înregistrarea video a meciului
- stats (JSONField): Date structurate despre meci, incluzând:
 - state: Starea meciului ("completed", "InProgress", "unstarted")
 - block_name: Contextul programului (ex: "Week 3 - Day 2")
 - best_of: Formatul seriei ("Bo1", "Bo3", "Bo5")

- games[]: Listă de obiecte per joc, fiecare conținând:
 - number: Numarul jocului (1, 2, 3...)
 - state: Starea jocului ("completed", "unneeded")
 - teams[]: Array cu cele două echipe, fiecare cu name, code, side (blue/red)
 - vods[]: Array de link-uri VOD cu parameter (YouTube ID) și locale
- external_id (CharField, max_length=200, unique=True, null=True): ID-ul extern din API-ul LoL Esports.

Structura JSON a câmpului stats este flexibilă și evoluează fără migrații. Această decizie de design permite stocarea de date diverse (meciuri cu 1, 3 sau 5 jocuri) fără a necesita modele relaționale separate pentru fiecare nivel de detaliu.

Modelul UserProfile extinde modelul built-in Django User cu funcționalități de favorite(utilizatorul poate să-și aleagă meciurile și echipele favorite).

Câmpuri:

- user (OneToOneField -> User, related_name='profile'): Utilizatorul asociat
- favorite_teams (ManyToManyField -> Team, blank=True): Echipele favorite
- favorite_matches (ManyToManyField -> Match, blank=True): Meciurile favorite
- created_at (DateTimeField, auto_now_add=True): Data creării profilului
- updated_at (DateTimeField, auto_now=True): Data ultimei actualizări

Un UserProfile este creat automat când un nou User este înregistrat, prin semnalul post_save definit în signals.py. Aceasta asigură că fiecare utilizator are întotdeauna un profil asociat.

Modelul NotificationLog urmărește notificările trimise pentru a preveni duplicatele.

Câmpuri:

- user (ForeignKey -> User): Utilizatorul care a primit notificarea
- match (ForeignKey -> Match): Meciul pentru care s-a trimis notificarea
- notification_type (CharField, max_length=50): Tipul notificării: 'favorite_team' sau 'favorite_match'
- sent_at (DateTimeField, auto_now_add=True): Data trimiterii

Constrângerea unique_together pe (user, match, notification_type) previne trimiterea aceleiași notificări de două ori, chiar dacă task-ul periodic rulează de multiple ori în fereastra de notificare.

4.2 IMPLEMENTAREA API-ULUI REST

Rutarea cererilor HTTP urmează doi pași: rutele de nivel superior din `config/urls.py` și rutele specifice aplicației din `esports/urls.py`.

În `config/urls.py`:

- `/admin/` -> Django Admin
- `/api/` -> `include(esports.urls)` și toate endpoint-urile API
- `/api-auth/` -> DRF browsable API auth

În `esports/urls.py`, un `DefaultRouter` DRF înregistrează `ViewSet`-urile:

- `router = routers.DefaultRouter()`
- `router.register('games', views.GameViewSet, basename='game')`
- `router.register('teams', views.TeamViewSet, basename='team')`
- `router.register('players', views.PlayerViewSet, basename='player')`
- `router.register('tournaments', views.TournamentViewSet, basename='tournament')`
- `router.register('matches', views.MatchViewSet, basename='match')`

Această înregistrare generează automat următoarele URL-uri pentru fiecare `ViewSet`:

- `GET /api/{resource}/` -> Listează toate obiectele
- `POST /api/{resource}/` -> Creează un obiect nou
- `GET /api/{resource}/{id}/` -> Returnează detaliile unui obiect
- `PUT /api/{resource}/{id}/` -> Actualizează complet un obiect
- `PATCH /api/{resource}/{id}/` -> Actualizează parțial un obiect
- `DELETE /api/{resource}/{id}/` -> Șterge un obiect

URL-urile suplimentare definite manual includ:

- `/api/search/` -> Căutare globală
- `/api/auth/register/, login/, logout/, me/, delete-account/` -> Autentificare
- `/api/teams/{id}/favorite/, /api/matches/{id}/favorite/` -> Alege favorit

`GameViewSet`: Un `ModelViewSet` simplu fără personalizări, expunând CRUD complet pe modelul `Game`.

`TeamViewSet`: Suportă filtrare opțională după regiune (`?region=europe`). Folosește `TeamSerializer` pentru lista și `TeamDetailSerializer` (care include `current_players`) pentru detaliu. Definește o acțiune custom `tournament_results` care:

1. Găsește toate meciurile echipei cu `Q(team1=team) | Q(team2=team)`

2. Grupează meciurile per turneu
3. Parsează scorul pentru a determina victorii/înfrângeri
4. Returnează statistici per turneu, sortate după data

PlayerViewSet: Suportă filtrare după regiune (?region=), filtrarea operând pe team_region (regiunea echipei jucătorului).

TournamentViewSet: Filtrat implicit să arate doar turneele de League of Legends din 2026 (game__name__icontains='league', start_date__year=2026).
Suportă filtrare după regiune prin Q(region=region) | Q(participants__region=region) cu .distinct() pentru a preveni duplicatele.
Definește acțiunea custom standings

MatchViewSet: Implementează o logică sofisticată de filtrare:

- Exclude meciurile cu echipe TBD sau null
- Când parametrul ?tournament= este prezent: afisează toate meciurile turneului (fără limita de dată)
- Altfel: aplică o fereastră de 14 zile în viitor (date_time <= now + 14d)
- Ordonează după date_time crescător

Serializerele transformă obiectele Django în reprezentări JSON și invers. Proiectul definește 9 serializere în fișierul serializers.py:

- TeamSerializer: Include câmpul calculat region_display (numele citibil al regiunii, ex: "South Korea" în loc de "south_korea") și un SerializerMethodField get_logo care detectează dacă logo-ul este un URL extern (începe cu http://) sau o cale locală de media și construiește URL-ul absolut corespunzător.
- TeamDetailSerializer: Extinde TeamSerializer cu game_name și current_players, un SerializerMethodField care interoghează Player.objects.filter(team=obj) și returnează o listă de PlayerSerializer.
- MatchSerializer: Include câmpuri relaționale calculate: team1_name, team2_name, tournament_name și tournament_region, fiecare accesând câmpuri ale modelelor relaționate prin notația source='team1.name'.
- RegisterSerializer: Gestionează înregistrarea utilizatorilor cu validare de email unic (UniqueValidator), validare de parolă (validate_password) și confirmare parolă (password2). Metoda create() apelează User.objects.create_user() care hash-uieste automat parola.

- **UserSerializer:** Serializează utilizatorul împreună cu favoritele din profil folosind `PrimaryKeyRelatedField` cu `source='profile.favorite_teams'`, returnând listele de ID-uri ale echipelor și meciurilor favorite.

Funcția `search()` din `views.py` este un endpoint function-based (`@api_view`) care acceptă un parametru de query `q` și efectuează căutare case-insensitive (`icontains`) în paralel pe trei modele:

- **Team:** Caută în `name` și `country`
- **Player:** Caută în `nickname` și `real_name`
- **Tournament:** Caută în `name` și `location`

Fiecare căutare este limitată la 10 rezultate. Căutarea necesită minimum 2 caractere. Răspunsul returnează un obiect JSON cu trei vectori: `teams`, `players`, `tournaments`.

Pe frontend, `Home.jsx` implementează căutarea cu `debounce` de 300ms: după ce utilizatorul termină de tastat, se așteaptă 300ms înainte de a trimite cererea, prevenind cereri excesive către server în timpul tastării.

Acțiunea `standings` din `TournamentViewSet` calculează dinamic clasamentele pe baza rezultatelor meciurilor. Implementează două moduri de calcul:

Modul standard: Pentru turneele obișnuite (LCK, LEC, LCS):

1. Se iterează prin fiecare echipă participantă
2. Se găsesc toate meciurile echipei în turneu (filtrate după data de început a turneului)
3. Pentru fiecare meci finalizat, se parsează scorul pentru a determina: victorii (`matches won`), înfrangeri (`matches lost`), jocuri câștigate (`games won`), jocuri pierdute (`games lost`)
4. Se sortează după: victorii (descrescător) -> diferența de jocuri (descrescător) -> jocuri câștigate (descrescător)

Modul cu grupe (LPL): Detectat prin verificarea numelui turneului ('LPL' în `tournament.name` and '2026' and 'Split 1'):

1. Se definesc trei grupe predefinite:
 - **Group Ascend:** Bilibili Gaming, Anyone's Legend, Top Esports, Invictus Gaming, JD Gaming, Weibo Gaming
 - **Group Perseverance:** Ninjas in Pyjamas, Team WE, EDward Gaming, ThunderTalk Gaming
 - **Group Nirvana:** LGD Gaming, LNG Esports, Oh My God, Ultra Prime
2. Pentru fiecare echipă, se numără doar meciurile împotriva echipelor din aceeași grupă
3. Se sortează și se atribuie rank-uri per grupă

4. Răspunsul include câmpul group pentru fiecare intrare

4.3 IMPLEMENTAREA AUTENTIFICĂRII

Înregistrarea este gestionată de RegisterView (generics.CreateAPIView):

1. Clientul trimite request POST pe endpoint-ul /api/auth/register/ cu username, email, password, password2
2. RegisterSerializer validează ca: emailul să fie unic, parola să respecte politica Django și dacă parolele se potrivesc.
3. User.objects.create_user() creează utilizatorul cu parola hash-uită
4. Semnalul post_save pe User creează automat un UserProfile
5. RefreshToken.for_user(user) generează perechea access + refresh
6. Răspunsul include: user data, refresh token, access token

Această abordare permite autentificarea imediat după înregistrare, fără a necesita un pas suplimentar de login.

Login-ul se face în felul următor: se trimite request POST pe /api/auth/login/ și folosește TokenObtainPairView din SimpleJWT. Clientul trimite username și password și primește access și refresh tokens.

Refresh-ul se realizează printr-un request POST pe endpoint-ul /api/auth/token/refresh/ cu refresh token-ul actual. Returnează un nou access token (și un nou refresh token datorită configurației ROTATE_REFRESH_TOKENS = True).

Logout-ul se face printr-un request POST pe endpoint-ul /api/auth/logout/ cu refresh token-ul. Funcția logout_view creează un obiect RefreshToken și apelează .blacklist() care adaugă token-ul în tabelul de blacklist. Token-ul invalidat nu mai poate fi folosit pentru refresh.

AuthContext.jsx implementează pe frontend un interceptor Axios care detectează răspunsurile 401 și încearcă automat să facă refresh înainte de a re-trimite cererea originală.

Favoritele sunt implementate ca toggle-uri pe endpoint-uri dedicate:

- POST /api/teams/{id}/favorite/ -> toggle_favorite_team:
 1. Se găsește echipa după ID
 2. Se accesează profilul utilizatorului (request.user.profile)
 3. Dacă echipa este deja în profile.favorite_teams atunci se elimină, dacă nu se adaugă
 4. Se returnează {is_favorite: true/false}

La fel este implementat și toggle_favorite_match.

AuthContext re-fetch-uește datele utilizatorului, pe frontend, după fiecare toggle (/api/auth/me/) pentru a sincroniza starea cu serverul.

4.4 SERVICIUL DE INTEGRARE CU API-UL EXTERN

Fișierul `services/lolesports_client.py` definește clasa `LolesportsClient` care încapsulează toată comunicarea cu API-ul LoL Esports.

La inițializare, clientul creează un `requests.Session` cu header-urile necesare (`x-api-key`, `Accept: application/json`). Folosirea unui `Session` permite reutilizarea conexiunilor TCP (`connection pooling`).

Metoda centrală `_get(endpoint, params)` implementează logica de `retry` cu `backoff` exponențial. Pentru fiecare încercare (0, 1, 2) trimite GET request cu timeout de 15 secunde. Dacă statusul este 200 returnează JSON parsat. Dacă statusul este 429 sau 500 sau mai mare decât 500 atunci așteaptă 2ⁿ încercare secunde (1s, 2s, 4s) și continuă cu următoarea încercare. Dacă statusul este alt 4xx atunci loghează eroare, returnează `None`. Dacă excepție de rețea atunci așteaptă 2ⁿ încercare secunde și continuă cu următoarea încercare. Dacă toate încercările au eșuat atunci loghează eroare și returnează `None`.

Această abordare asigură reziliența față de erori tranzitorii (`rate limiting`, erori de server, probleme de rețea) fără a bloca execuția pe termen lung.

Fișierul `services/data_mapper.py` definește clasa `DataMapper` care transformă răspunsurile JSON ale API-ului în dicționare compatibile cu modelul `Match`.

Metoda `map_schedule_event(event)` procesează evenimentele din `getSchedule` extrage `external_id` din `event.match.id`, extrage numele echipelor, fără meciurile cu echipe TBD, parsează timestamp-ul ISO (`startTime`) în `datetime` Python, extrage starea, `block name`, scorul (din `result.gameWins`), strategia, caută link-uri VOD în `games[].vods[].parameter` și construiește obiectul stats JSON.

Metoda `map_event_details(event_detail)` procesează detaliile de la `getEventDetails`, construiește un `team_id_map` din datele echipelor la nivel de meci pentru a îmbogăți datele per joc cu numele și codurile echipelor, extrage link-urile pentru VOD-urile în limba engleză (locale începând cu "en") și derivează starea generală din stările pentru fiecare joc. Dacă toate jocurile sunt "completed" sau "unneeded" (joc care nu a fost jucat deoarece seria deja a fost decisă, de exemplu jocul 3 dintr-o serie best of 3 care s-a terminat 2-0) atunci se pun ca și "completed". Dacă vreun joc este "inProgress" atunci se pune "inProgress".

Metoda `resolve_team(api_team_name)` din `DataMapper` pune un nume de echipă din API la un obiect `Team` local, folosind o strategie în 4 pași:

- se caută în tabelul de aliasuri, se verifică dicționarul `TEAM_NAME_ALIASES`. Dacă există o mapare (de exemplu "Gen.G" -> "Gen.G Esports"), se folosește numele mapat.
- potrivire exactă: `Team.objects.filter(name=db_name).first()`
- potrivire case-insensitive: `Team.objects.filter(name__icontains=db_name).first()`

- potrivire inversă pe participanții turneului. Se verifică dacă numele vreunei echipe din turneu se află în numele din API sau invers. Aceasta rezolvă cazuri precum "Beijing JDG Intel Esports" care conține "JD Gaming" (parțial).

Rezultatele sunt păstrate într-un dicționar intern (`_team_cache`) pentru a minimiza interogările la baza de date când același nume apare de mai multe ori.

Fișierul `services/team_name_aliases.py` conține un dicționar Python, actualizat manual, `TEAM_NAME_ALIASES` care mapează variante de nume utilizate de API la numele din baza de date locală. Câteva exemple:

```
'BNK FearX' = 'FEARX'  
'NaVi' = 'Natus Vincere'  
'Giants' = 'GIANTX'  
'WeiboGaming Faw Audi' = 'Weibo Gaming'  
'Team Liquid Alienware' = 'Team Liquid'
```

4.5 TASK-URILE PERIODICE

Funcția `fetch_match_schedules()` din `tasks.py` este executată la fiecare 15 minute și are următorul flux:

- se creează o instanță `LolesportsClient`
- se definește un dicționar de fallback VOD pe ligi (ex: LPL folosește canalul YouTube @LPL_English ca fallback)
- pentru fiecare ligă (LCK, LEC, LPL, LCS, LCP) se apelează `client.get_schedule(league_id)`, se identifică turneul local prin căutare după numele ligii, se creează un `DataMapper` cu contextul turneului.
- pentru fiecare eveniment se mapează prin `mapper.map_schedule_event(event)`, se transformă echipele prin `mapper.resolve_team()`, se construiește dicționarul `defaults` cu `tournament`, `team1`, `team2`, `date_time`, `score`, `vod_link` (cu fallback pe ligă), `result` (derivat din scor), `stats`. Se face `upsert Match.objects.update_or_create(external_id=mapped['external_id'], defaults=defaults)`.
- se procesează paginile următoare (`pages.newer token`)
- se loghează sumar `created=X`, `updated=Y`, `skipped=Z`

Funcția `fetch_match_results()` din `tasks.py` este executată la fiecare 5 minute și actualizează meciurile existente cu date detaliate și are următorul flux:

- se identifică meciurile care necesită actualizare (max 50 per categorie):
 - categoria 1: Meciuri din trecut cu `stats.state != 'completed'`
 - categoria 2: Meciuri finalizate fără `stats.games` (lipsesc detalii)
- se deduplicatează listele
- pentru fiecare meci se apelează `client.get_event_details(match.external_id)`, se mapează prin `mapper.map_event_details(details)`, se actualizează câmpurile doar

- dacă s-au schimbat score, vod_link, stats, result (cu winner derivat din scor), se salvează meciul (trigger-ul post_save broadcast-eaza prin WS).
- se loghează updated=X, errors=Y, checked=Z

Funcția `send_match_notifications()` din `tasks.py` este executată la fiecare 5 minute și implementează două tipuri de notificări:

- Notificarea pentru echipe favorite (2 ore înainte): se calculează intervalul `[now + 2h, now + 2h5m]`, se găsesc meciurile în acest interval. Pentru fiecare meci se găsesc profilurile utilizatorilor care au favorite team1 sau team2, se combină și se deduplicatează profilurile. Pentru fiecare profil se verifică `NotificationLog` (dacă a fost deja trimisă), se trimite email cu `send_mail()` cu formatul Subiect: "Upcoming Match: TeamA vs TeamB" și Corp: Detalii meci, turneu, ora. Se creează `NotificationLog`.
- Notificarea pentru meciuri favorite (10 minute înainte): se calculează intervalul: `[now + 10m, now + 15m]`, se găsesc meciurile în acest interval, se găsesc utilizatorii care au marcat meciul ca favorit. Același flux de verificare, trimitere și logare ca și la primul tip de notificare.

4.6 SEMNALELE DJANGO ȘI BROADCAST-UL WEBSOCKET

Fișierul `signals.py` definește trei receptori de semnale:

- `create_user_profile`: la crearea unui User nou (`post_save`, `created=True`), se creează automat un `UserProfile` asociat. Aceasta asigură că operațiunile pe `profile.favorite_teams` nu vor eșua cu un profil inexistent.
- `save_user_profile`: la salvarea unui User, se salvează și profilul asociat (dacă există).
- `broadcast_match_update`: la salvarea unui Match (`post_save`), se trimite actualizarea prin Django Channels. Se obține `channel layer`-ul, se construiește mesajul cu: `match_id`, `external_id`, `team1`, `team2`, `score`, `state`, `date_time` și se trimite prin `async_to_sync(channel_layer.group_send)("matches", ...)`.

Fișierul `apps.py` conține metoda `ready()` a configurației `EsportsConfig` care importă `esports.signals`, asigurând că receptorii sunt conectați la pornirea aplicației.

4.7 IMPLEMENTAREA FRONTEND-ULUI REACT

Punctul de intrare al aplicației React este `index.js` care importă CSS-ul Bootstrap global, creează root-ul React cu `createRoot` și randează în `StrictMode > ThemeProvider > App`.

Componenta App (`App.js`) configurează context providers: `RegionProvider > AuthProvider`, React Router (`BrowserRouter`), Layout: Navigation (sus) + container cu Routes (mijloc) + Footer (jos) și 12 rute definite pentru toate paginile aplicației.

ThemeContext (ThemeContext.jsx) stochează tema curentă ('dark' sau 'light') în state și localStorage. La schimbare setează data-bs-theme pe document.body. Bootstrap 5 aplică automat stilurile dark/light pe toate componentele.

RegionContext (RegionContext.jsx) definește constantele REGIONS cu 6 opțiuni (All, Europe, NA, China, KR, APAC). Fiecare regiune are: id, name, code (folosit ca parametru pentru API). Selecția este păstrată în localStorage.

AuthContext.jsx este cel mai complex context provider, gestionând întreg ciclul de viață al autentificării. La montarea componentei (useEffect) se verifică localStorage pentru access_token. Dacă există, setează header-ul Authorization pe Axios. Apelează GET /api/auth/me/ pentru a încărca datele utilizatorului. Dacă primește 401, încearcă refresh-ul tokenului, apoi setează loading = false.

Interceptorul Axios (useEffect) înregistrează un interceptor de răspuns. La primirea unui 401 (și dacă nu e deja o re-încercare) se încearcă refresh-ul tokenului. Dacă reușește, cererea originală este retrimisă cu noul token. Dacă eșuează, utilizatorul este deconectat. Flag-ul _retry pe request config previne bucle infinite.

Funcțiile expuse în frontend sunt:

- login(username, password): request POST pe /api/auth/login/, stochează tokeni și face fetch user data.
- register(username, email, password, password2): request POST pe /api/auth/register/
- logout(): request POST refresh token pe /api/auth/logout/, șterge localStorage
- deleteAccount(): request DELETE /api/auth/delete-account/, șterge state
- toggleFavoriteTeam(teamId): request POST pe /api/teams/{id}/favorite/, re-fetch user
- toggleFavoriteMatch(matchId): request POST pe /api/matches/{id}/favorite/
- isAuthenticated: boolean calculat

Home.jsx este pagina de landing cu două secțiuni:

Secțiunea Hero:

- Titlu cu gradient CSS ("Welcome to Rift Pulse")
- Bara de căutare globală cu debounce de 300ms. useRef pentru timer-ul de debounce. La fiecare apăsare de tastă, se resetează timer-ul. După 300ms fără tastare, se trimite GET /api/search/?q=... Rezultatele apar într-un dropdown cu trei categorii (TEAMS, PLAYERS, TOURNAMENTS). Fiecare rezultat poate fi apăsat și navighează către pagina de detaliu.
- Butoane de acțiune rapidă: "Explore Regions", "View Tournaments"

Secțiunea Upcoming Matches:

- Se face Fetch GET /api/matches/ la montare
- Filtrează meciurile din viitor și sortează crescător
- Afișează primele 8 meciuri într-un grid de 4 coloane
- Fiecare card arată: data/ora, block name, echipe, turneu

Teams.jsx afișează echipele într-un grid responsive (1/2/3 coloane) cu filtre de țară și regiune (populate dinamic din date), card-uri cu logo (cu culoare de fundal conditionată pentru contrastul logo-urilor transparente), nume, țară, regiune. Click navighează la TeamDetail.

Players.jsx afișează jucătorii într-un grid de 4 coloane cu filtre (de echipă, țară și rol), card-uri (cu nickname, real name, rol, țară, echipă) și sortare alfabetică după nickname.

Tournaments.jsx afișează turneele în grid de 2 coloane cu filtru de regiune prin React Router state (de la Regions), sortare după an (Newest/Oldest) și game (A-Z/Z-A), card-uri cu nume, badge status (colorat), game name, date, locație, prize pool (formatat \$K/\$M), format.

TeamDetail.jsx are profil detaliat de echipă cu breadcrumb și trei tab-uri. Header cu nume, buton favorit (stea), țară, regiune, data fondării, social media, logo cu fundal dinamic. Primul tab este Overview cu descrierea echipei, al doilea tab este Current Roster cu grid de card-uri cu jucători (nickname, rol, țară, data nașterii, social media). Link-uri către PlayerDetail și al treilea tab este Tournament Results cu tabel cu performanța per turneu (nume turneu, link, status, meciuri, victorii, infrangeri, win rate colorat).

TournamentDetail.jsx este cel mai complex component, cu până la 4 tab-uri. Are header cu nume, game name, status badge, dată, locație, prize pool, format. Primul tab este Overview cu grid de echipe participante, al doilea tab este Matches care împarte meciurile în "Upcoming" și "Past". MatchCard arată data, block name, echipe (câștigător verde), scor badge, VOD button. Al treilea tab este Standings cu două moduri de afișare, cu grupe (LPL) cu tabele separate per grupă cu coloane Rank, Team, Wins, Losses, Games Won, Games Lost, Game Diff și fără grupe cu tabel unic cu toate echipele. Al patrulea tab este Bracket care apare doar dacă turneul are meciuri playoff (detectate prin block_name). Grupează meciurile pe runde și afișează card-uri cu cele două echipe, scoruri și câștigător evidențiat.

Matches.jsx afișează toate meciurile cu filtrare, sortare și detalii expandabile. Are filtre (de regiune) și sortare (după dată) și counter (showing x of y matches). Conține card-uri de meci (full-width) care au în stânga data/ora, block name, în centru echipe (câștigător în verde bold), badge de scor sau "Upcoming". În dreapta, pe card-uri, se află rezultat (numele câștigătorului) și butoane pentru Favorite (stea - toggle, necesită autentificare), Watch VOD (link extern YouTube, filtrează placeholder lolesports.com) și Show Details (doar pentru meciuri finalizate cu stats.games). Secțiunea expandabilă (la click pe Show Details) are fundal dark, titlu "Game Details", grid de card-uri per joc, număr joc ("Game 1"), badge stare (verde/gri/galben), două rânduri de echipe cu nume, badge side (albastru/rosu).

Profile.jsx conține pagina de profil (necesită autentificare, redirecționează la /login dacă nu). Are header cu salut personalizat ("Welcome back, username!"). Secțiunea Favorite Teams are grid de card-uri cu logo, nume, regiune. Fiecare are buton X de eliminare. Link "Browse teams" dacă lista e goală.

Secțiunea Favorite Matches conține grid de card-uri cu echipe, data, scor, turneu. Fiecare are buton X de eliminare. În Danger Zone Buton se află "Delete My Account" cu dialog de confirmare în doi pași și avertisment că acțiunea este permanentă.

Favoritele sunt încărcate prin fetch-uri paralele (Promise.all) folosind ID-urile din `user.favorite_teams` și `user.favorite_matches`.

Fișierul `admin.py` configurează interfața Django Admin pentru gestionarea datelor. Formulare personalizate: Câmpurile `JSONField` (`social_media`, `stats`) sunt randate ca `Textarea` cu dimensiuni personalizate și `help text` explicativ. Aceasta rezolvă problema widgetului implicit care randează JSON pe o singură linie, făcându-l dificil de editat. Configurări per model:

- `GameAdmin`: `list_display=['name', 'developer', 'publisher'], search_fields=['name']`
- `TeamAdmin`: `list_display=['name', 'region', 'country', 'founded_date'], list_filter=['region', 'country'], search_fields=['name', 'country']`
- `PlayerAdmin`: `list_display=['nickname', 'real_name', 'team', 'role', 'country'], list_filter=['role', 'country', 'team']`
- `TournamentAdmin`: `list_display=['name', 'game', 'region', 'start_date', 'end_date', 'status'], date_hierarchy='start_date'`
- `MatchAdmin`: `list_display=['id', 'tournament', 'team1', 'team2', 'date_time', 'score', 'result'], date_hierarchy='date_time'`

5.UTILIZAREA APLICAȚIEI

5.1 DESCRIEREA APLICAȚIEI

Rift Pulse este accesibil prin intermediul unui browser web modern. Interfața este construită cu Bootstrap 5 și oferă un design responsive care se adaptează la diferite dimensiuni de ecran (desktop, tableta, telefon mobil). Tema implicită este dark, cu posibilitatea de a comuta la tema light prin butonul din bara de navigare.

Bara de navigare superioară conține:

- Logo-ul "Rift Pulse" (link către pagina principală)
- Link-uri de navigare: Regions, Teams, Players, Tournaments, Matches
- Buton de comutare tema (dark/light)
- Butoane de autentificare: Login și Register (pentru vizitatori) sau numele utilizatorului și Logout (pentru utilizatorii autentificați)

Footer-ul se află pe fiecare pagină și conține textul "Rift Pulse - Your premier source for League of Legends esports" și "Developed by Cioara Mario Răzvan".

Aplicația nu necesită autentificare pentru a vizualiza datele (echipe, jucatori, turnee, meciuri, clasamente). Autentificarea este necesară doar pentru funcționalitățile de personalizare: marcarea favoritelor și primirea notificărilor prin email.

5.2 SCENARII DE UTILIZARE

Scenariul 1: Vizitator explorează turneele pe regiuni

Un vizitator nou accesează aplicația și vede pagina principală cu bara de căutare și meciurile viitoare. (Figura 5.1)

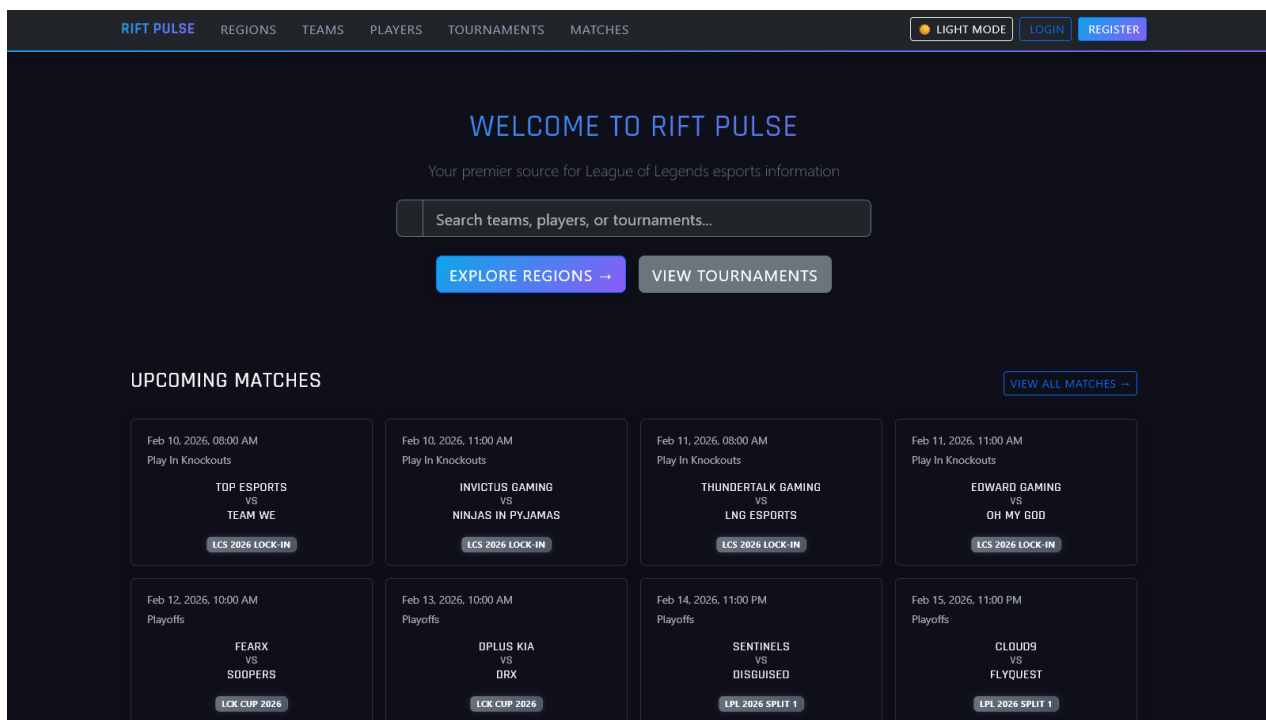


Figura 5.1 - Pagina Home

Apasă pe "Explore Regions" și vede 6 card-uri cu regiunile disponibile (cu emoji-uri de steag). (Figura 5.2)

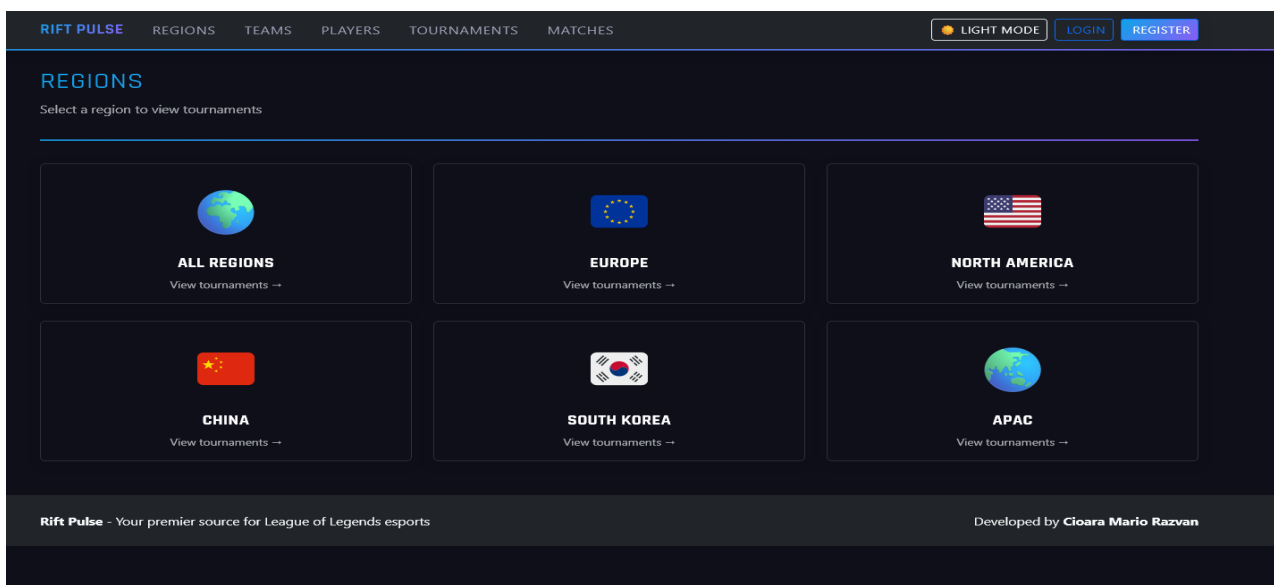


Figura 5.2 - Pagina Regions

Alege "South Korea" și este redirecționat la pagina de turnee, filtrată automat pentru această regiune. Vede turneul "LCK 2026 Split 1" cu badge-ul "ongoing", datele, locația și fondul de premii. (Figura 5.3)

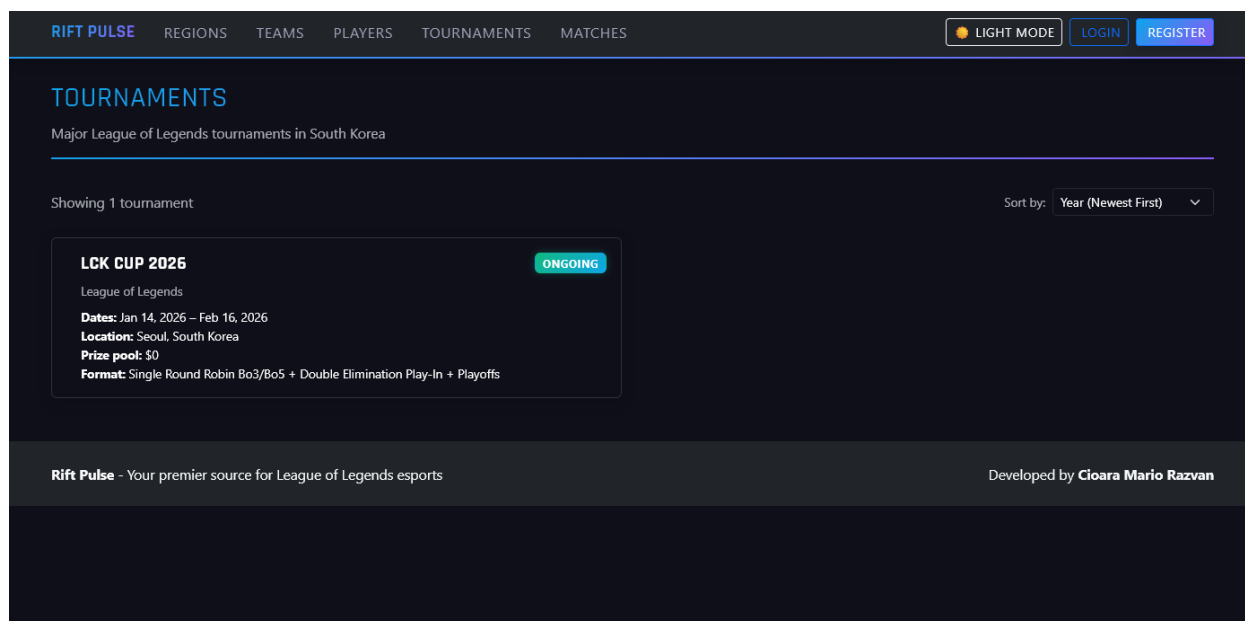


Figura 5.3 - Pagina de Turnee pentru Coreea de Sud

Dă click pe turneu și vede pagina de detaliu cu 4 tab-uri: Overview (echipele participante), Matches (meciuri viitoare și trecute), Standings (clasamentul curent calculat din rezultate) și Bracket (meciurile playoff, dacă există).(Figurile 5.4,5.5,5.6,5.7)

RIFT PULSE [REGIONS](#) [TEAMS](#) [PLAYERS](#) [TOURNAMENTS](#) [MATCHES](#)

LIGHT MODE

LOGIN

REGISTER

[Tournaments](#) / [LCK Cup 2026](#)

LCK CUP 2026

ONGOING

League of Legends

DATES:
Jan 14, 2026 – Feb 16, 2026

LOCATION:
Seoul, South Korea

PRIZE POOL:
\$0

FORMAT:
Single Round Robin Bo3/Bo5 + Double Elimination Play-In + Playoffs

OVERVIEW

MATCHES

STANDINGS

BRACKET

PARTICIPATING TEAMS

DPLUS KIA
South Korea

GEN.G ESPORTS
South Korea

FEARX
South Korea

HANWHA LIFE ESPORTS
South Korea

KT ROLSTER
South Korea

NONGSHIM REDFORCE
South Korea

SOOPERS
South Korea

T1
South Korea

BRION
South Korea

DRX
South Korea

Rift Pulse - Your premier source for League of Legends esports

Developed by **Cioara Mario Razvan**

Figura 5.4 - Pagina turneului LCK Cup 2026 cu tab-ul Overview

- 37 -

[RIFT PULSE](#) [REGIONS](#) [TEAMS](#) [PLAYERS](#) [TOURNAMENTS](#) [MATCHES](#)

[LIGHT MODE](#) [LOGIN](#) [REGISTER](#)

[Tournaments](#) / [LCK Cup 2026](#)

LCK CUP 2026

League of Legends

DATES:
Jan 14, 2026 – Feb 16, 2026

LOCATION:
Seoul, South Korea

PRIZE POOL:
\$0

FORMAT:
Single Round Robin Bo3/Bo5 + Double
Elimination Play-In + Playoffs

ONGOING

[OVERVIEW](#) [MATCHES](#) [STANDINGS](#) [BRACKET](#)

UPCOMING MATCHES

Feb 12, 2026, 10:00 AM
Playoffs

FEARX VS SOOPERS

0-0

unstarted

Feb 13, 2026, 10:00 AM
Playoffs

DPLUS KIA VS DRX

0-0

unstarted

PAST MATCHES

Feb 8, 2026, 11:00 AM
Play-In - Losers Bracket

DRX VS BRION

WATCH VOD

Feb 8, 2026, 10:00 AM
Play-Ins

NONGSHIM REDFORCE VS DRX

0-3

DRX

Feb 7, 2026, 12:00 PM
Play-Ins

SOOPERS VS NONGSHIM REDFORCE

2-1

SOOPers

Figura 5.5 - Pagina turneului LCK Cup 2026 cu tab-ul Matches

RIFT PULSE
REGIONS
TEAMS
PLAYERS
TOURNAMENTS
MATCHES
LIGHT MODE
LOGIN
REGISTER

Tournaments / LCK Cup 2026

LCK CUP 2026

League of Legends

DATES:
Jan 14, 2026 – Feb 16, 2026
LOCATION:
Seoul, South Korea
PRIZE POOL:
\$0
FORMAT:
Single Round Robin Bo3/Bo5 + Double Elimination Play-In + Playoffs

OVERVIEW
MATCHES
STANDINGS
BRACKET

CURRENT STANDINGS

#	TEAM	WINS	LOSSES	GAMES WON	GAMES LOST	GAME DIFF
1	Gen.G Esports	11	0	24	0	+24
2	T1	10	2	22	10	+12
3	Dplus KIA	7	4	15	12	+3
4	DRX	6	6	17	19	-2
5	Nongshim RedForce	6	8	21	25	-4
6	FEARX	5	4	14	13	+1
7	SOOPers	5	7	16	20	-4
8	KT Rolster	3	8	13	21	-8
9	Hanwha Life Esports	3	8	11	21	-10
10	BRION	1	10	11	23	-12

Figura 5.6 - Pagina turneului LCK Cup 2026 cu tab-ul Standings

The screenshot displays the 'LCK CUP 2026' tournament page. The header includes navigation links: RIFT PULSE, REGIONS, TEAMS, PLAYERS, TOURNAMENTS, and MATCHES. There are also buttons for LIGHT MODE, LOGIN, and REGISTER. The main content area shows the tournament details: LCK CUP 2026 (League of Legends), DATES: Jan 14, 2026 – Feb 16, 2026, LOCATION: Seoul, South Korea, PRIZE POOL: \$0, and FORMAT: Single Round Robin Bo3/Bo5 + Double Elimination Play-In + Playoffs. Below this, there are tabs for OVERVIEW, MATCHES, STANDINGS, and BRACKET. The BRACKET tab is selected, showing the PLAYOFF BRACKET. The bracket is divided into two columns. The left column shows FEARX and SOOPers, both with a score of 0, and the date Feb 12, 2026. The right column shows Dplus KIA and DRX, both with a score of 0, and the date Feb 13, 2026. The footer contains the text 'Rift Pulse - Your premier source for League of Legends esports' and 'Developed by Cioara Mario Razvan'.

RIFT PULSE REGIONS TEAMS PLAYERS TOURNAMENTS MATCHES LIGHT MODE LOGIN REGISTER

Tournaments / LCK Cup 2026

LCK CUP 2026 ONGOING

League of Legends

DATES: Jan 14, 2026 – Feb 16, 2026

LOCATION: Seoul, South Korea

PRIZE POOL: \$0

FORMAT: Single Round Robin Bo3/Bo5 + Double Elimination Play-In + Playoffs

OVERVIEW MATCHES STANDINGS **BRACKET**

PLAYOFF BRACKET

PLAYOFFS

FEARX	0	Dplus KIA	0
SOOPers	0	DRX	0
Feb 12, 2026		Feb 13, 2026	

Rift Pulse - Your premier source for League of Legends esports

Developed by Cioara Mario Razvan

Figura 5.7 - Pagina turneului LCK Cup 2026 cu tab-ul Bracket

Scenariul 2: Utilizator cauta un jucator specific

Un utilizator tastează "Faker" în bara de căutare de pe pagina principală. După 300ms, apare un dropdown cu rezultatele. În secțiunea PLAYERS vede "Faker - Lee Sang-hyeok - Mid - T1". (Figura 5.8)

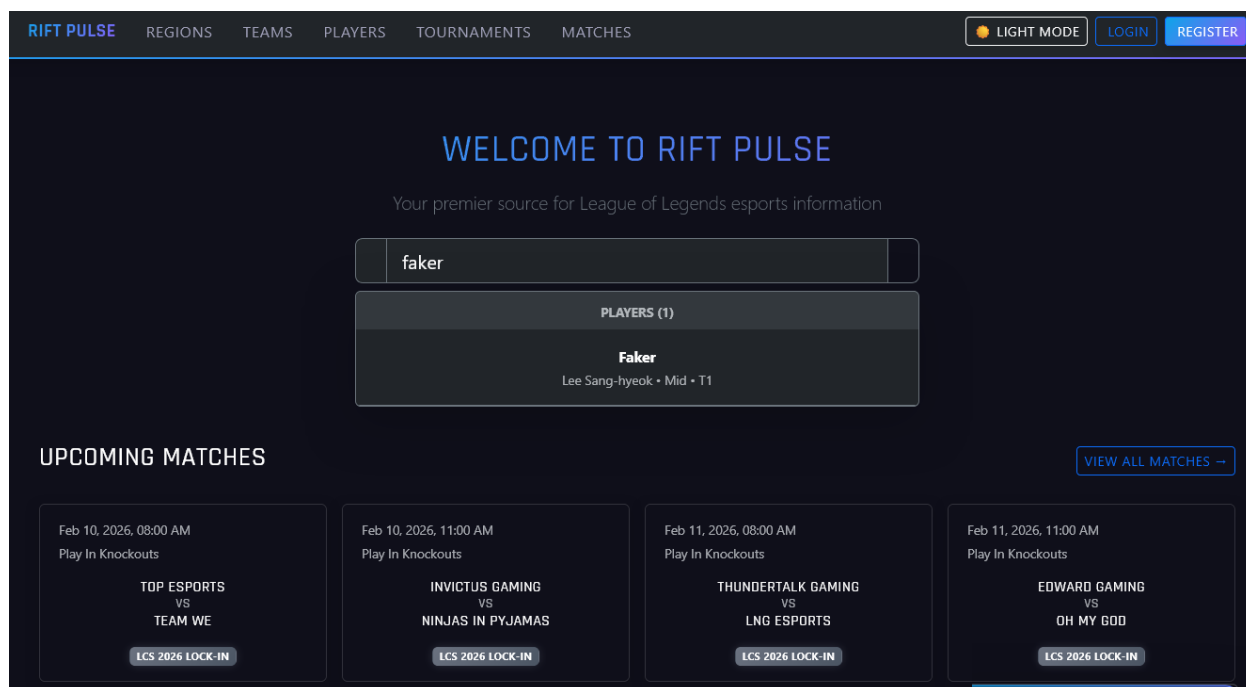


Figura 5.8 - Căutare jucător pe pagina principală

Dă click și ajunge pe pagina de profil a jucătorului. Tab-ul "Current Team" arată echipa T1 cu logo și detalii. Tab-ul "Statistics" afișează statisticile detaliate: KDA, Win Rate, CSM, DPM etc., organizate pe categorii. Tab-ul "Career Overview" arată informații profesionale. (Figurile 5.9, 5.10, 5.11)

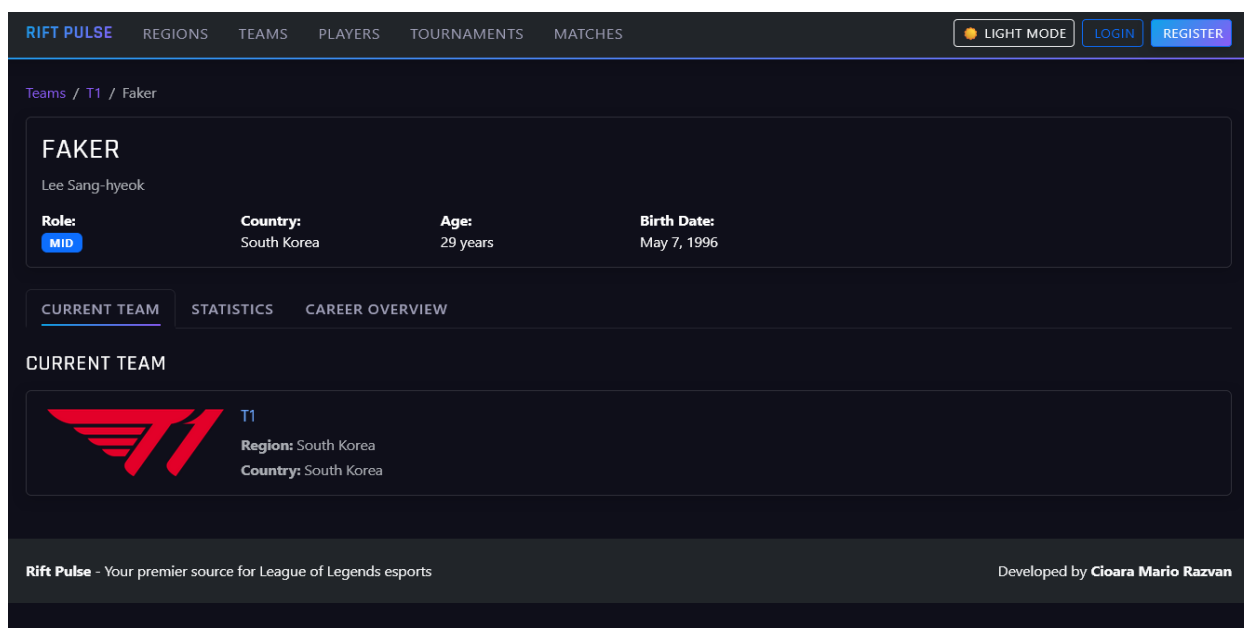


Figura 5.9 - Pagina jucătorului Faker cu tab-ul Current Team

RIFT PULSE

REGIONS

TEAMS

PLAYERS

TOURNAMENTS

MATCHES

LIGHT MODE

LOGIN

REGISTER

Teams / T1 / Faker

FAKER

Lee Sang-hyeok

Role:

MID

Country:

South Korea

Age:

29 years

Birth Date:

May 7, 1996

CURRENT TEAM

STATISTICS

CAREER OVERVIEW

PLAYER STATISTICS

GENERAL STATS

Games	32.00
Win Rate	0.72
KDA	3.60
AVG Kills	4.20
AVG Deaths	3.10
AVG Assists	7.00

COMBAT STATS

KP%	53.7%
DMG%	21.7%
DPM	707.00
FB%	15.6%
FB Victim	0.19
Penta Kills	0.00
Solo Kills	10.00

FARMING STATS

CSM	8.80
GPM	433.00
CSG@15	-4.00
GD@15	-84.00
XPD@15	-142.00

VISION STATS

VS%	13.9%
VSPM	1.26
AVG WPM	0.49
AVG WCPM	0.23
AVG VWPM	0.15

ECONOMY STATS

GOLD%	20.8%
-------	-------

Figura 5.10 - Pagina jucătorului Faker cu tab-ul Statistics

The screenshot displays the 'Faker' player profile on the 'Rift Pulse' website. The navigation bar at the top includes 'RIFT PULSE', 'REGIONS', 'TEAMS', 'PLAYERS', 'TOURNAMENTS', and 'MATCHES', along with 'LIGHT MODE', 'LOGIN', and 'REGISTER' buttons. The breadcrumb trail shows 'Teams / T1 / Faker'. The player's name 'FAKER' is prominently displayed, followed by 'Lee Sang-hyeok'. Key details include 'Role: MID', 'Country: South Korea', 'Age: 29 years', and 'Birth Date: May 7, 1996'. Below this, three tabs are visible: 'CURRENT TEAM', 'STATISTICS', and 'CAREER OVERVIEW' (which is active). The 'CAREER OVERVIEW' section contains a 'TEAM HISTORY' table with one entry: 'T1' (Current), 'Playing as Mid'. Below the table is a 'PROFESSIONAL INFORMATION' section with fields for 'In-game Name: Faker', 'Nationality: South Korea', 'Real Name: Lee Sang-hyeok', and 'Primary Role: MID'. The footer states 'Rift Pulse - Your premier source for League of Legends esports' and 'Developed by Cioara Mario Razvan'.

Figura 5.11 - Pagina jucătorului Faker cu tab-ul Career Overview

Scenariul 3: Utilizator autentificat marcheaza favorite si primeste notificari

Un utilizator se înregistrează cu username, email și parolă. După înregistrare, este autentificat automat și redirecționat la pagina principală. Navighează la pagina echipei "Team WE" și apasă butonul "Add to Favorites" (stea). Apoi navighează la pagina de meciuri, găsește un meci viitor Top Esports vs Team WE și apasă "Favorite" pe acel meci. (Figura 5.12)

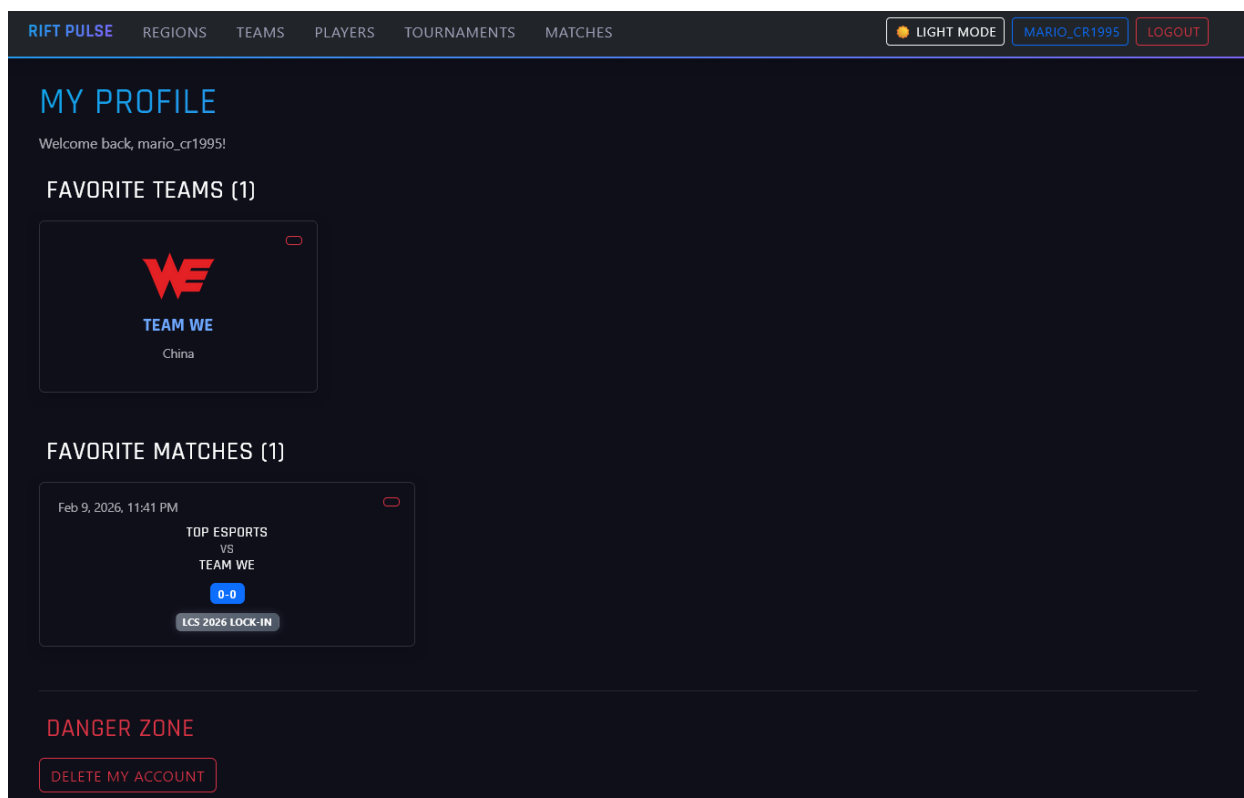


Figura 5.12 - Pagina utilizatorului cu echipa și meciul favorit

Cu 2 ore înainte de meci, primește un email: " Upcoming Match: Top Esports vs Team WE ". (Figura 5.13)

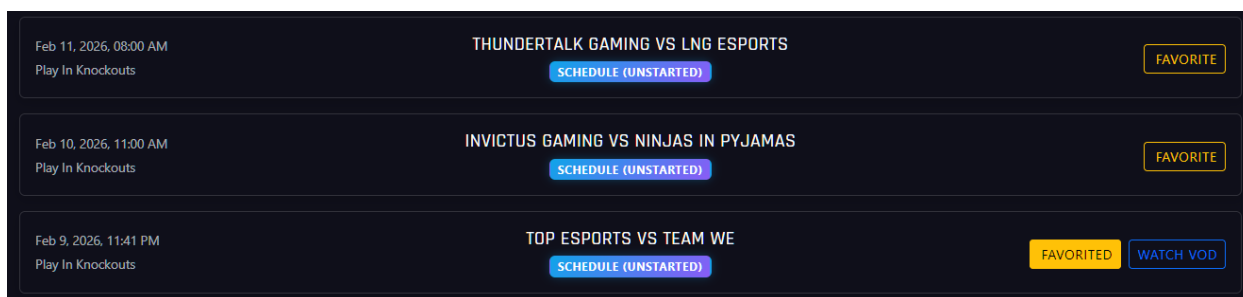


Figura 5.13 - Pagina cu meciul Top Esports vs Team setat la ora 23:41

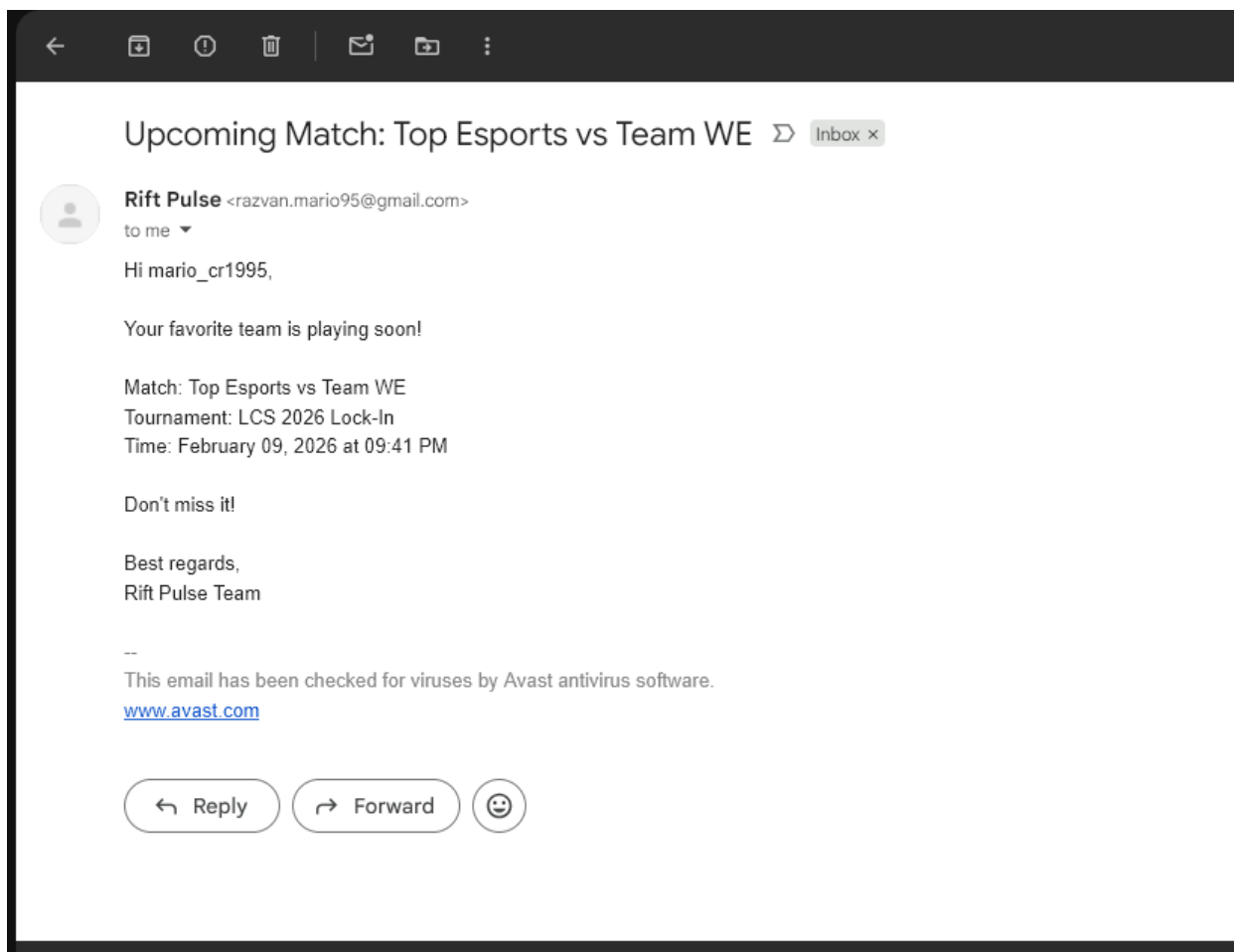


Figura 5.13 - Notificare primită prin email cu 2 ore înainte de începerea meciului

Cu 10 minute înainte, se primește un al doilea email: "Match Starting Soon: Top Esports vs Team WE ". (Figura 5.14)

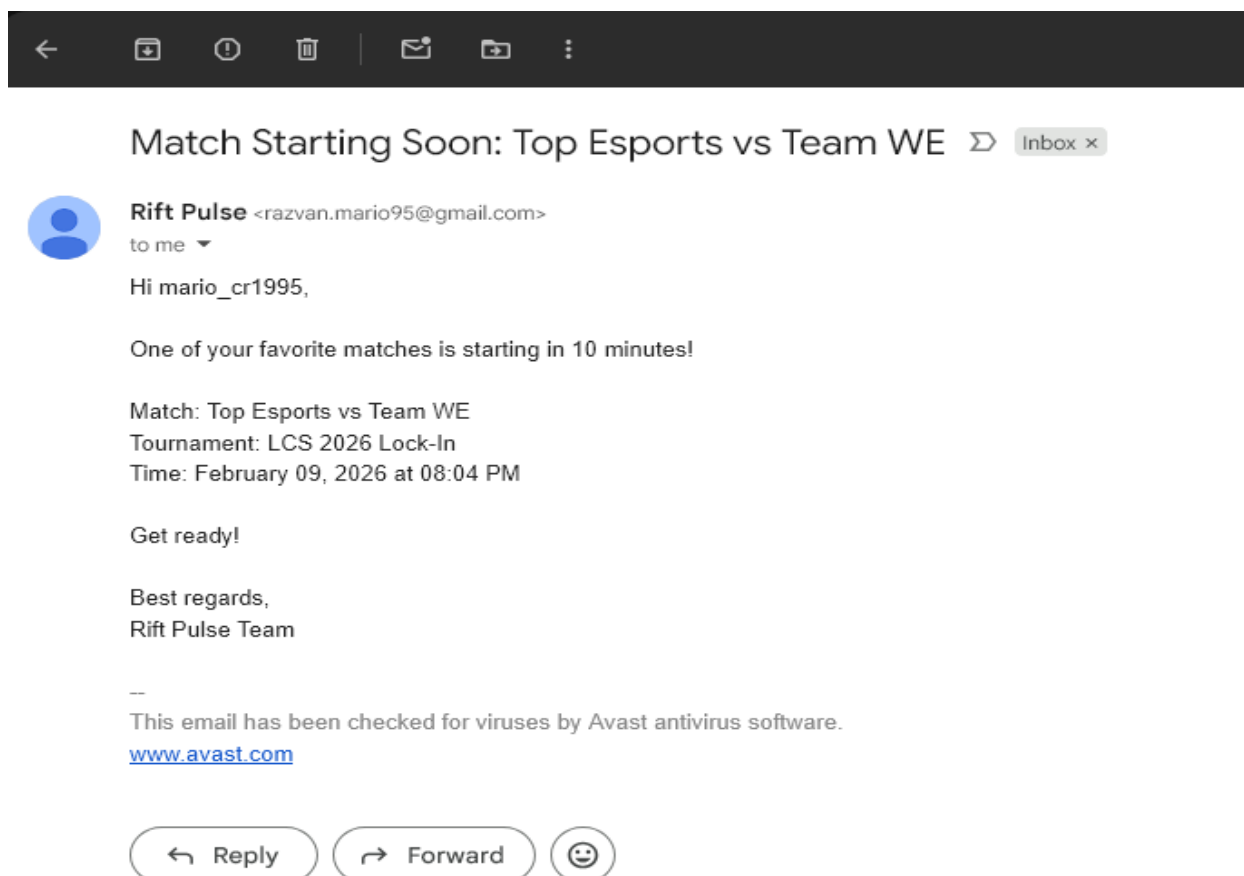


Figura 5.14 - Notificare primită prin email cu 10 min înainte de începerea meciului

Scenariu 4: Utilizator urmărește rezultatele meciurilor

Un utilizator navighează la pagina de meciuri și filtrează după regiune "Europe". Vede meciurile LPL sortate după dată. Pentru meciurile finalizate, numele castigatorului apare în verde bold, iar scorul este afisat ca badge (ex: "2-1"). Apasă "Watch VOD" pentru a vedea inregistrarea pe YouTube. Apasă "Show Details" pe un meci pentru a vedea detaliile per joc informațiile despre side (blue/red) sunt afișate cu badge-uri colorate. (Figura 5.15)

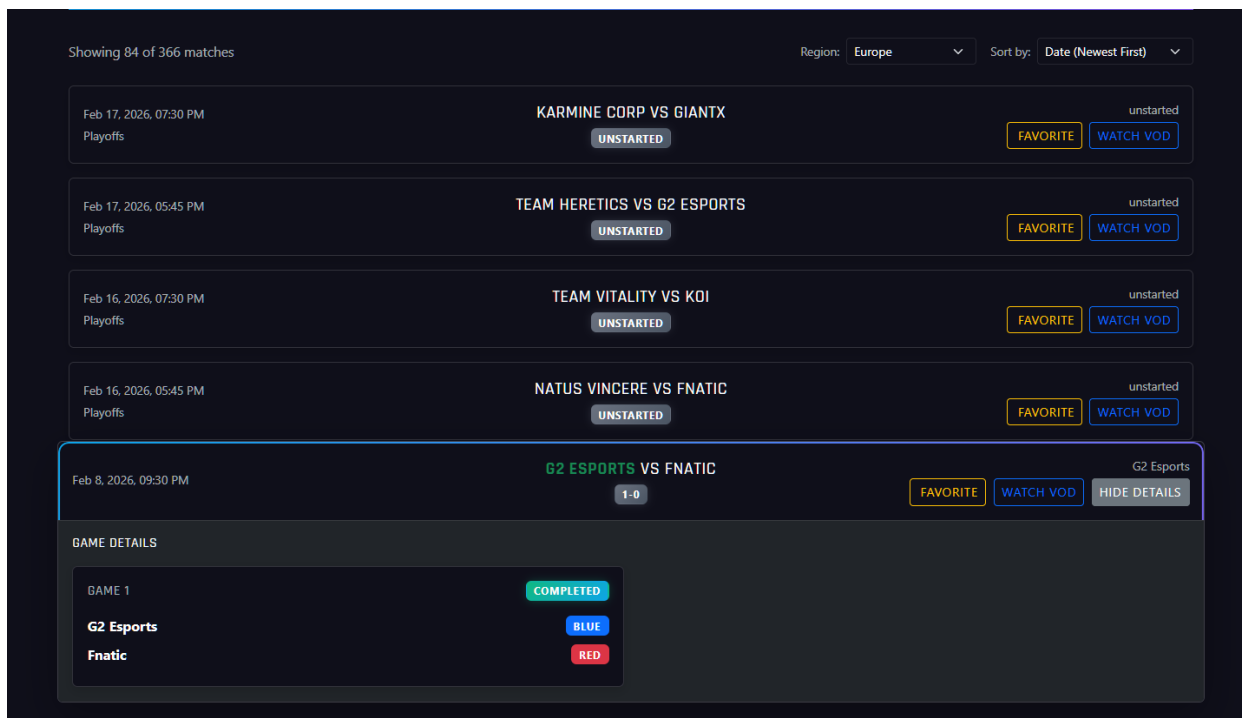


Figura 5.15 - Pagina cu meciurile filtrate după regiunea Europe

Scenariu 5: Utilizator verifică clasamentele unui turneu

Un utilizator navighează la turneul "LPL 2026 Split 1" și selectează tab-ul "Standings". Vede trei tabele separate, unul pentru fiecare grupă (Group Ascend, Group Perseverance, Group Nirvana). Fiecare tabel arată: poziția, echipa, victorii (verde), înfrângeri (roșu), jocuri câștigate, jocuri pierdute și diferența de jocuri (colorată verde/roșu). Echipele sunt sortate după victorii, apoi după diferența de jocuri. Pentru turneele din alte regiuni (LCK, LEC, LCS), clasamentul este un singur tabel fara grupe. (Figura 5.16 și Figura 5.17)

RIFT PULSE

REGIONS

TEAMS

PLAYERS

TOURNAMENTS

MATCHES

LIGHT MODE

MARIO_CR1995

LOGOUT

Tournaments / LPL 2026 Split 1

LPL 2026 SPLIT 1

League of Legends

ONGOING

DATES:

LOCATION:

PRIZE POOL:

FORMAT:

Jan 14, 2026 – Mar 8, 2026

China (Shanghai, Suzhou, Shenzhen, Xi'an, Beijing)

\$230.6K

Double round-robin regular season followed by double elimination playoffs

OVERVIEW

MATCHES

STANDINGS

BRACKET

CURRENT STANDINGS

GROUP ASCEND

#	TEAM	WINS	LOSSES	GAMES WON	GAMES LOST	GAME DIFF
1	Bilibili Gaming	5	2	12	7	+5
2	Anyone's Legend	3	2	8	5	+3
3	Top Esports	2	1	5	3	+2
4	JD Gaming	2	2	5	5	+0
5	Weibo Gaming	2	2	5	6	-1
6	Invictus Gaming	0	5	1	10	-9

GROUP PERSEVERANCE

#	TEAM	WINS	LOSSES	GAMES WON	GAMES LOST	GAME DIFF
1	Team WE	2	1	5	2	+3
2	Ninjas in Pyjamas	1	0	2	1	+1
3	EDward Gaming	1	1	2	2	+0
4	ThunderTalk Gaming	0	2	0	4	-4

GROUP NIRVANA

#	TEAM	WINS	LOSSES	GAMES WON	GAMES LOST	GAME DIFF
1	LNG Esports	3	0	6	2	+4
2	LGD Gaming	1	1	3	3	+0
3	Oh My God	1	1	2	3	-1
4	Ultra Prime	0	3	3	6	-3

Figura 5.16 - Turneul LPL cu tab-ul Standings

RIFT PULSE	REGIONS	TEAMS	PLAYERS	TOURNAMENTS	MATCHES	LIGHT MODE	MARIO_CR1995	LOGOUT
------------	---------	-------	---------	-------------	---------	------------	--------------	--------

Tournaments / LCK Cup 2026	
LCK CUP 2026	
League of Legends	
DATES: Jan 14, 2026 – Feb 16, 2026	LOCATION: Seoul, South Korea
PRIZE POOL: \$0	FORMAT: Single Round Robin Bo3/Bo5 + Double Elimination Play-In + Playoffs

OVERVIEW	MATCHES	STANDINGS	BRACKET
----------	---------	-----------	---------

CURRENT STANDINGS						
#	TEAM	WINS	LOSSES	GAMES WON	GAMES LOST	GAME DIFF
1	Gen.G Esports	11	0	24	0	+24
2	T1	10	2	22	10	+12
3	Dplus KIA	7	4	15	12	+3
4	DRX	6	6	17	19	-2
5	Nongshim RedForce	6	8	21	25	-4
6	FEARX	5	4	14	13	+1
7	SOOPers	5	7	16	20	-4
8	KT Rolster	3	8	13	21	-8
9	Hanwha Life Esports	3	8	11	21	-10
10	BRION	1	10	11	23	-12

Figura 5.17 - Turneul LCK Cup cu tab-ul Standings

Scenariu 6: Administratorul gestionează datele

Administratorul accesează <http://localhost:8000/admin/> și se autentifică. Poate vedea și edita toate modelele: Games, Teams, Players, Tournaments, Matches, User Profiles, Notification Logs. Poate filtra meciurile după turneu sau data, căuta echipe după nume, vizualiza ierarhia de date pe meciuri. Câmpurile JSON (stats, social_media) sunt randate ca textarea-uri editabile cu help text. (Figura 5.18)

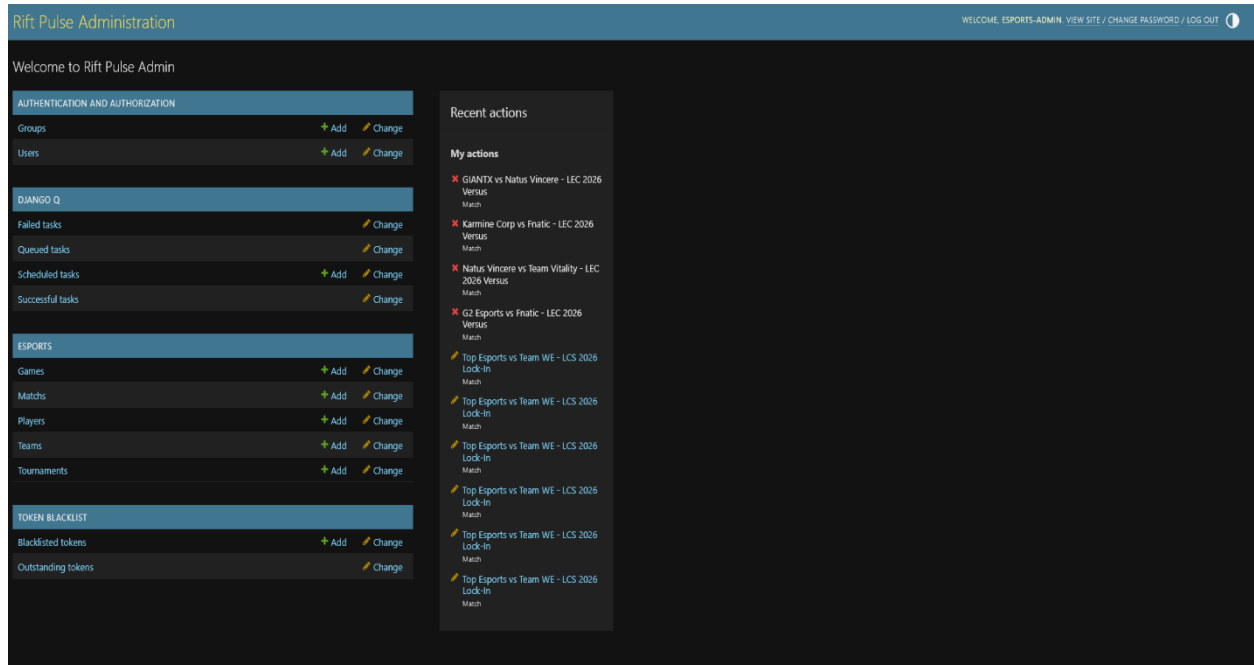


Figura 5.18 - Panoul de administrare al bazei de date

Scenariu 7: Pipeline-ul automat actualizeaza datele

Administratorul rulează 'python manage.py setup_scheduled_tasks' pentru a configura task-urile periodice, apoi 'python manage.py qcluster' pentru a porni worker-ul Django-Q. La fiecare 15 minute, fetch_match_schedules interoghează API-ul LoL Esports pentru cele 5 ligi și creează/actualizează meciurile. La fiecare 5 minute, fetch_match_results actualizează meciurile finalizate cu detalii per joc și link-uri VOD. Tot la fiecare 5 minute, send_match_notifications trimite emailuri utilizatorilor înainte de meciuri. Toate aceste operații se desfășoară în background, fără intervenție manuală. (Figura 5.19)

```
INFO 2026-02-09 22:35:49,883 tasks fetch_match_schedules: LCS returned 80 events
INFO 2026-02-09 22:35:50,049 tasks fetch_match_schedules: created=0, updated=244, skipped=76
22:35:50 [Q] INFO Processed 'esports.tasks.fetch_match_schedules' (ohio-item-tennessee-venus)
INFO 2026-02-09 22:36:09,765 tasks fetch_match_results: updated=50, errors=0, checked=50
22:36:09 [Q] INFO Processed 'esports.tasks.fetch_match_results' (winter-robert-coffee-beer)
22:37:46 [Q] INFO Enqueued [esports_notifications] 971
22:37:46 [Q] INFO Process-5173b89ca6b449d896449b0838e42ed6 created task robin-football-may-steak from schedule [send_match_notifications]
22:37:46 [Q] INFO Process-able2aa1dbf24abb8a7777d3b3d641e6 processing robin-football-may-steak 'esports.tasks.send_match_notifications' [send_match_notifications]
22:37:47 [Q] INFO Processed 'esports.tasks.send_match_notifications' (robin-football-may-steak)
```

Figura 5.19 - Task-urile rulând

5.3 Lansare în producție

Aplicația Rift Pulse a fost lansată în producție pe platforma Railway.com, un serviciu de cloud hosting care suportă lansarea aplicațiilor direct din repository-uri GitHub.

Arhitectura de deployment este compusă din patru servicii separate, toate gestionate în cadrul aceluiasi proiect Railway:

- 1) Serviciul web backend - rulează serverul ASGI Daphne, care gestionează atât cererile HTTP pentru API-ul REST, cât și conexiunile WebSocket pentru actualizările în timp real. La fiecare pornire a serviciului, se execută automat migrările bazei de date și colectarea fișierelor statice înainte de lansarea serverului.
- 2) Serviciul qcluster - rulează procesul Django-Q (qcluster) responsabil cu executarea task-urilor periodice: fetch-ul programărilor meciurilor la fiecare 15 minute, actualizarea rezultatelor la fiecare 5 minute și trimiterea notificărilor prin email. Acest serviciu utilizează aceeași bază de date ca serviciul web.
- 3) Baza de date PostgreSQL - aprovizionată automat de Railway ca plugin, cu conexiunea configurată prin variabila de mediu DATABASE_URL. Datele din baza de date locală de dezvoltare au fost migrate în producție folosind utilitarele pg_dump și pg_restore.
- 4) Serviciul frontend - servește build-ul static al aplicației React folosind pachetul npm 'serve'. Fișierul nixpacks.toml configurează explicit mediul Node.js pentru fazele de build și runtime, asigurând disponibilitatea interpretorului în ambele etape.

Pregătirea codului pentru producție a necesitat mai multe modificări:

Externalizarea configurărilor sensibile: Toate secretele și configurările dependente de mediu au fost mutate din fișiere hardcodate în variabile de mediu. SECRET_KEY, DEBUG, ALLOWED_HOSTS, CORS_ALLOWED_ORIGINS, credențialele email și cheia API LoL Esports sunt citite din os.environ cu valori implicite pentru dezvoltarea locală. Acest lucru permite rularea aceluiasi cod atât local cât și în producție, fără modificări.

Configurarea bazei de date: Biblioteca dj-database-url parsează automat variabila DATABASE_URL furnizată de Railway, eliminând necesitatea configurării

individuale a host-ului, portului, numelui bazei de date și credențialelor. Local, se folosește un URL PostgreSQL implicit către baza de date de dezvoltare.

Servirea fișierelor statice: Middleware-ul WhiteNoise a fost adăugat imediat după SecurityMiddleware în lanțul de middleware Django. Acesta servește fișierele statice (CSS, JavaScript, imagini) direct din aplicația Python. Fișierele media (logo-urile echipelor) sunt servite printr-un URL pattern dedicat care funcționează indiferent de valoarea flagului DEBUG.

Securitate în producție: Când DEBUG este dezactivat, aplicația activează automat mai multe măsuri de securitate: redirectarea automată HTTP către HTTPS (SECURE_SSL_REDIRECT), marcarea cookie-urilor de sesiune și CSRF ca secure (transmise doar prin HTTPS), setarea headerului SECURE_PROXY_SSL_HEADER pentru detectarea corectă a protocolului din spatele proxy-ului Railway și configurarea CSRF_TRUSTED_ORIGINS pentru a accepta cereri de la domeniile frontend și backend.

Centralizarea configurării API pe frontend: Toate cele 26 de apeluri HTTP hardcodate către `http://localhost:8000` din cele 10 fișiere componente React au fost înlocuite cu o instanță Axios centralizată. Fișierul `api.js` citește variabila de mediu `REACT_APP_API_URL` (setată la build time prin fișierul `.env.production`) și creează o instanță Axios cu `baseUrl`-ul corespunzător. Astfel, aceeași bază de cod funcționează atât local (către `localhost:8000`) cât și în producție (către `api.rift-pulse.com`).

Serverul ASGI Daphne: În locul serverului de dezvoltare Django, producția utilizează Daphne, un server ASGI performant care suportă nativ atât protocolul HTTP cât și WebSocket. Daphne este înregistrat în `INSTALLED_APPS` și configurat ca punct de intrare în `Procfile`-ul backend-ului.

Channel layers cu Redis: În dezvoltare, Django Channels utilizează un `InMemoryChannelLayer` pentru simplitate. În producție, când variabila de mediu `REDIS_URL` este prezentă (furnizată automat de plugin-ul Redis din Railway), aplicația comută automat pe `RedisChannelLayer`, care permite comunicarea între procese și suportă scalarea pe mai multe instanțe.

Migrarea datelor: Transferul bazei de date din mediul de dezvoltare în producție a fost realizat prin exportul cu `pg_dump` în format custom (`-Fc`) și restaurarea cu `pg_restore` cu flagul `--no-owner` pe URL-ul public al bazei de date Railway. Această abordare a copiat integral structura și datele (echipe, jucatori, turnee, meciuri, profiluri de utilizator) într-o singură operație.

6. CONCLUZII

Dezvoltarea aplicației Rift Pulse a necesitat utilizarea și integrarea multor tehnologii și concepte învățate pe parcursul facultății, precum și acumularea unor cunoștințe noi specifice dezvoltării web moderne.

Din punct de vedere al backend-ului, proiectul mi-a permis să aprofundez framework-ul Django și ecosistemul său extins. Am învățat să proiectez modele de date cu relații complexe (ForeignKey, ManyToManyField, OneToOneField) și câmpuri flexibile (JSONField), să construiesc un API REST complet cu Django REST Framework utilizând ViewSets, serializare și routere, și să implementez un sistem de autentificare bazat pe JWT cu tokenuri de acces și refresh, rotație automată și blacklisting.

Implementarea pipeline-ului automat de date a ridicat provocări interesante: comunicarea cu un API extern neoficial (fără documentație publică), tratarea erorilor de rețea cu retry și backoff exponențial și operațiile de upsert idempotente folosind external_id drept cheie unică.

Integrarea Django Channels pentru WebSocket m-a introdus în programarea asincronă în contextul Django, utilizând ASGI în loc de WSGI, channel layers pentru comunicarea între procese și semnale Django pentru declanșarea broadcast-urilor. Sistemul a fost proiectat cu reziliență: eșecul broadcast-ului nu afectează niciodată salvarea datelor.

Sistemul de notificări prin email a combinat task-uri periodice (Django-Q), interogări complexe la baza de date (filtrare pe relații many-to-many cu ferestre temporale) și integrare SMTP, cu un mecanism robust de prevenire a duplicatelor prin NotificationLog.

Pe partea de frontend, am aprofundat React 19 cu hooks și Context API, construind o aplicație SPA cu 12 pagini, 3 context providers, căutare globală cu debounce, rutare client-side cu parametri dinamici și interceptori Axios pentru refresh automat al tokenurilor JWT. Fiecare pagină implementează un pattern consistent: declararea stării, fetch la montare, aplicare filtre/sortari, randare condiționată cu stări de loading/error/empty.

Procesul de lansare în producție a adus un set distinct de provocări și cunoștințe. Am învățat să externalizez configurările sensibile prin variabile de mediu, să configurez serverul ASGI Daphne pentru traficul de producție, să utilizez WhiteNoise pentru servirea fișierelor statice și să gestionez migrarea bazei de date între medii folosind pg_dump și pg_restore. Configurarea platformei Railway cu servicii separate pentru web, worker și baza de date mi-a oferit experiența practică în gestionarea unei arhitecturi de microservicii, iar integrarea domeniului personalizat cu certificate SSL automate a completat ciclul complet de la dezvoltare la producție.

Provocările principale întâmpinate au fost:

- discrepanțele de numire ale echipelor între API și baza de date locală, rezolvate prin tabelul de aliasuri și strategia de rezolvare în 4 pași
- derivarea stării generale a meciului din stările individuale per joc, inclusiv tratarea stării "unneeded" (jocuri nedeșfășurate în serii câștigate anticipat)
- îmbogățirea datelor din API-ul de programari (minimal) cu date din API-ul de detalii (complet), necesitând două faze de fetch separate
- gestionarea logo-urilor echipelor cu fundal transparent, necesitând culori de fundal condiționate per echipă.
- adaptarea codului pentru producție: diferențele între mediul de dezvoltare și cel de producție (variabile de mediu, servirea fișierelor statice și media, securitatea HTTPS, configurarea CORS)

7. BIBLIOGRAFIE

Django Software Foundation, Django Documentation (v5.2)
<https://docs.djangoproject.com/en/5.2/> accesare februarie 2026

Tom Christie, Django REST Framework Documentation
<https://www.django-rest-framework.org/> accesare februarie 2026

David Amos, SimpleJWT Documentation
<https://django-rest-framework-simplejwt.readthedocs.io/> accesare februarie 2026

Andrew Godwin, Django Channels Documentation
<https://channels.readthedocs.io/> accesare februarie 2026

Ilan Steemers, Django-Q Documentation
<https://django-q.readthedocs.io/> accesare februarie 2026

The PostgreSQL Global Development Group, PostgreSQL Documentation
<https://www.postgresql.org/docs/> accesare februarie 2026

Meta (Facebook), React Documentation (v19)
<https://react.dev/> accesare februarie 2026

Remix Software, React Router Documentation (v7)
<https://reactrouter.com/> accesare februarie 2026

Matt Zabriskie, Axios Documentation
<https://axios-http.com/> accesare februarie 2026

Bootstrap Team, Bootstrap Documentation (v5.3)
<https://getbootstrap.com/docs/5.3/> accesare februarie 2026

Internet Engineering Task Force, RFC 7519 - JSON Web Token (JWT)
<https://datatracker.ietf.org/doc/html/rfc7519> accesare februarie 2026

Internet Engineering Task Force, RFC 6455 - The WebSocket Protocol
<https://datatracker.ietf.org/doc/html/rfc6455> accesare februarie 2026

Roy T. Fielding, Architectural Styles and the Design of Network-based Software Architectures (REST), 2000

<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.html> accesare februarie 2026

Kenneth Reitz, Requests: HTTP for Humans Documentation

<https://docs.python-requests.org/> accesare februarie 2026

Railway Corporation, Railway Documentation

<https://docs.railway.com/> accesare februarie 2026

David Cramer, WhiteNoise Documentation

<https://whitenoise.readthedocs.io/> accesare februarie 2026

Kenneth Reitz, dj-database-url Documentation

<https://pypi.org/project/dj-database-url/> accesare februarie 2026

Andrew Godwin, Daphne - Django ASGI Server

<https://github.com/django/daphne>

*** <https://lolesports.com/> accesare februarie 2026

*** <https://liquipedia.net/> accesare februarie 2026

*** https://lol.fandom.com/wiki/League_of_Legends_Esports_Wiki accesare februarie 2026

[1] ***<https://escharts.com/top-games?order=peak&year=2025> accesare februarie 2026

[2] Conținut generat cu IA

[3] Conținut generat cu IA

[4] Conținut generat cu IA

[5] Conținut generat cu IA

[6] Conținut generat cu IA

[7] Conținut generat cu IA

DECLARAȚIE DE AUTENTICITATE A
LUCRĂRII DE FINALIZARE A STUDIILOR*

Subsemnatul CIOARA MARIO RĂZVAN

legitimat cu CI seria HD nr. 984220
CNP 1350514205569
autorul lucrării RIFT PULSE - APLICAȚIE WEB PENTRU PASIONAȚII
DE LEAGUE OF LEGENDS

elaborată în vederea susținerii examenului de finalizare a studiilor de LICENȚĂ organizat de către Facultatea AUTOMATICĂ ȘI CALCULATOARE din cadrul Universității Politehnica Timișoara, sesiunea FEBRUARIE 2026 a anului universitar 2025-2026, coordonator ALBU-MARȘIAN ADRIANA, luând în considerare art. 34 din Regulamentul privind organizarea și desfășurarea examenelor de licență/diplomă și disertație, aprobat prin HS nr. 109/14.05.2020 și cunoscând faptul că în cazul constatării ulterioare a unor declarații false, voi suporta sancțiunea administrativă prevăzută de art. 146 din Legea nr. 1/2011 – legea educației naționale și anume anularea diplomei de studii, declar pe proprie răspundere, că:

- această lucrare este rezultatul propriei activități intelectuale;
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să nu fie citate, inclusiv situația în care sursa o reprezintă o altă lucrare/alte lucrări ale subsemnatului;
- sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor;
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen/prezentată public/publicată de licență/diplomă/disertație;
- În elaborarea lucrării am utilizat instrumente specifice inteligenței artificiale (IA) și anume CLAUDE (denumirea) CLAUDE.AI (sursa), pe care le-am citat în conținutul lucrării nu am utilizat ~~instrumente specifice inteligenței artificiale (IA)~~¹.

Declar că sunt de acord ca lucrarea să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Timișoara,

Data

09.02.2026

Semnătura

M Cioara

¹Declarația se completează de student, se semnează olograf de acesta și se inserează în lucrarea de finalizare a studiilor, la sfârșitul lucrării, ca parte integrantă.

² Se va păstra una dintre variante: 1 - s-a utilizat IA și se menționează sursa 2 – nu s-a utilizat IA