**RĪGAS VALSTS TEHNIKUMS**

DATORIKAS NODAĻA

Izglītības programma: Programmēšana

**KVALIFIKĀCIJAS DARBS**

**“Sporta spēles derību tīmekļu vietne”**

Paskaidrojošais raksts 154 lpp.

Audzēknis: Danila Goļcovs

Prakses vadītājs: Ilona Demčenko

Nodaļas vadītājs: Normunds Barbāns

**Rīga 2024**

**ANOTāCIJA**

Kvalifikācijas darbā ir aprakstīts kibersporta derību vietnes izstrādes process. Sistēma nodrošina iespēju vietnes lietotājiem veikt likmes uz dažādiem kibersporta notikumiem, kā arī sniedz plašu informāciju par spēļu statistiku un tiešraides straumēm. Šī platforma palīdz efektīvi pārvaldīt lietotāju kontus, kā arī nodrošina drošu un ātru darījumu apstrādi. Izstrādātā sistēma piedāvā inovatīvas funkcijas, kas uzlabo lietotāju pieredzi. Sistēma tika izstrādāta ar Java Script programmēšanas valodu, izmantojot Node.js un React.js ietvaru, un MySQL relāciju datubāzu vadības sistēmu.

Kvalifikācijas darbs ietver ievadu, uzdevumu nostādni, prasību specifikāciju, uzdevuma risināšanas līdzekļu izvēles pamatojumu, programmatūras produkta modelēšanas un projektēšanas aprakstu, datu struktūru aprakstu, lietotāju ceļvedi, nobeigumu un pielikumus. Kvalifikācijas darba ievadā ir sporta spēles derībasv vietnes aktuālā problēma un tās risinājums. Uzdevumu nostādnē ir norādīti uzdevumi, kurus sistēmā būs nepieciešams veikt. Prasību specifikācija sastāv no ieejas un izejas informācijas, kā arī no sistēmas funkcionālajām un nefunkcionālajām prasībām. Uzdevuma risināšanas līdzekļu izvēles pamatojumā ir norādīti, kādi līdzekļi tiks izmantoti izstrādāšanai un kādiem nolūkiem tie tiek izmantoti. Programmatūras produkta modelēšanas un projektēšanas apraksts sastāv no sistēmas struktūra modeļa, kas ietver sistēmas arhitektūru un ER modeli, un funkcionālās sistēmas modeļa, kas satur datu plūsmas modeli. Datu struktūru aprakstā tiek parādīta datu bāzes relāciju shēma, kā arī tabulu struktūra ar aprakstu, kas ietver datu tipu un datu garumu norādīšanu. Lietotāju ceļvedī ir norādītas nepieciešamās sistēmas prasības aparatūrai un programmatūrai, sistēmas instalācija un palaišana, kā arī programmas apraksts, kas paskaidro, kā pareizi lietot sistēmu. Testa piemēri ir dots detalizēt likems likšanas apraksts ar vizuāliem attēliem.

Kvalifikācijas darbs sastāv no 154 lappusēm, kurā ietilpst 28 attēli, 10 tabulas un 2 pielikumiem. Pielikumi satur funkcionālās dekompozīcijas diagrammu, ER diagrammu, pasūtījumu datu plūsmas diagrammu un datu bāzes relāciju shē

**ANNotation**

The qualification paper describes the development process of an esports betting website. The system provides users with the ability to place bets on various esports events and offers extensive information on game statistics and live streams. This platform helps effectively manage user accounts and ensures secure and fast transaction processing. The developed system offers innovative features that enhance the user experience. The system was developed using the JavaScript programming language, utilizing the Node.js and React.js frameworks, and the MySQL relational database management system.

The qualification paper includes an introduction, task statement, requirements specification, justification for the choice of problem-solving tools, description of software product modeling and design, description of data structures, user guide, conclusion, and appendices. The introduction of the qualification paper outlines the current problem of the sports betting website and its solution. The task statement specifies the tasks that need to be performed in the system. The requirements specification consists of input and output information, as well as the system's functional and non-functional requirements. The justification for the choice of problem-solving tools indicates which tools will be used for development and their purposes. The software product modeling and design description includes the system structure model, which encompasses the system architecture and ER model, and the functional system model, which contains the data flow model. The data structure description presents the database relationship schema and the table structure description, including data types and data length specifications. The user guide outlines the necessary system requirements for hardware and software, system installation and launch, and provides a program description explaining how to use the system correctly. Test examples include a detailed betting placement description with visual illustrations.

The qualification paper consists of 154 pages, which include 28 images, 10 tables, and 2 appendices. The appendices contain the functional decomposition diagram, ER diagram, order data flow diagram, and database relationship schem

**Saturs**

[ievads 5](#_Toc168843331)

[1. UZDEVUMA NOSTĀDNE 7](#_Toc168843332)

[2. PRASĪBU SPECIFIKĀCIJA 8](#_Toc168843333)

[2.1. Ieejas un izejas informācijas apraksts 8](#_Toc168843334)

[2.1.1.Ieejas informācijas apraksts 8](#_Toc168843335)

[2.1.2. Izejas informācijas apraksts 9](#_Toc168843336)

[2.2. Funkcionālās prasības 10](#_Toc168843337)

[2.3. Nefunkcionālās prasības 13](#_Toc168843338)

[3. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS 16](#_Toc168843339)

[4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA 18](#_Toc168843340)

[4.1. Sistēmas struktūras modelis 18](#_Toc168843341)

[4.1.1. Sistēmas arhitektūra 18](#_Toc168843342)

[4.2. Sistēmas ER modelis 20](#_Toc168843343)

[4.3. Datu plūsmu modelis 22](#_Toc168843344)

[5. DATU STRUKTŪRU APRAKSTS 30](#_Toc168843345)

[6. LIETOTĀJA CEĻVEDIS 34](#_Toc168843346)

[6.1. Sistēmas prasības aparatūrai un programmatūrai 34](#_Toc168843347)

[6.1. Sistēmas instalācija un palaišana 35](#_Toc168843348)

[6.2. Programmas apraksts 36](#_Toc168843349)

[6.3. Testa piemērs 57](#_Toc168843350)

[NOBEIGUMS 59](#_Toc168843351)

[INFORMĀCIJAS AVOTI 60](#_Toc168843352)

[PIELIKUMI 61](#_Toc168843353)

[1.pielikums Lietojumgadījuma diagramma 62](#_Toc168843354)

[2.pielikums Programmas pirmkods 64](#_Toc168843358)

# ievads

Tiešsaistes azartspēļu industrija strauji pieaug, radot lielu konkurenci. Viena no vadošajām nozarem ir sporta likmes, kas ir ieguvusi plašu popularitāti tiešsaistes derību entuziastu vidū. Šī kvalifikācijas darba mērķis ir izveidot sporta spēles derību tīmekļu vietni, kura pamatvirziens būs kibersports.

Sporta derību vietne nodrošina lietotājiem iespēju baudīt azartspēles, piedaloties likmju veikšanā uz dažādiem sporta notikumiem. Tas piedāvā izklaides un iespējas laimēt naudu, nodrošinot ērtības un pieejamību klientiem, kuri var izbaudīt spēli mājās vai ceļojot. Lai saglabātu savu pozīciju tirgū, Sporta derību vietnei ir nepieciešams pielāgoties tirgus prasībām un piedāvāt inovatīvus risinājumus klientu vajadzībām apmierināšanai.

Kibersports ir viens no visstraujāk augošajiem sporta veidiem pasaulē, un tā popularitāte pieaug ik dienu. Bukmeikeru vietnei, kas specializējas kibersportā, ir ievērojamas priekšrocības salīdzinājumā ar tradicionālajām derību platformām. Kibersports piesaista jaunāku un tehnoloģiski prasmīgāku auditoriju, kura meklē jaunas un inovatīvas derību iespējas. Tā kā kibersports ir salīdzinoši jauns un dinamiskā attīstībā esošs tirgus segments, tas piedāvā milzīgu izaugsmes potenciālu un mazāku konkurenci, salīdzinot ar tradicionālajiem sporta veidiem.

Specializējoties kibersportā, bukmeikeru vietne var piedāvāt pielāgotus pakalpojumus, kas atbilst šīs specifiskās auditorijas vajadzībām un vēlmēm, piemēram, detalizētus spēļu analīzes, tiešraides straumēšanu un ekskluzīvas likmju iespējas. Tas palīdzēs vietnei ne tikai piesaistīt, bet arī noturēt klientus, kuri meklē unikālu un specializētu pieredzi. Turklāt, ņemot vērā kibersporta eksponenciālo izaugsmi un arvien pieaugošo globālo atpazīstamību, šāda specializācija ievērojami palielinās vietnes konkurētspēju un spēju izcelties tirgū.

Bukmeikeru vietnei, kas specializējas kibersportā, mērķauditorija ir daudzveidīga un ietver:

* jaunieši (18-35 gadi): tehnoloģiski prasmīgi kibersporta sekotāji un spēlētāji, kas meklē modernus un intuitīvus risinājumus;
* kibersporta fani: regulāri turnīru un sacensību skatītāji, kas vēlas informāciju par spēlēm un komandu statistiku;
* geimeri: tiešsaistes spēļu kopienas locekļi, kas vēlas izmantot savas zināšanas, lai veiktu likmes;
* rehnoloģiju entuziasti: cilvēki, kas seko jaunākajām tehnoloģiju un kibersporta tendencēm;
* sporta derību cienītāji: pieredzējuši likmju veicēji, kas meklē jaunas un aizraujošas iespējas;
* starptautiskā auditorija: kibersporta globālā pievilcība piesaista lietotājus no visas pasaules.

Šī daudzveidīgā auditorija palīdzēs vietnei izcelties tirgū un piesaistīt plašu lietotāju bāzi.

Kibersporta derību tirgū jau darbojas vairāki nozīmīgi spēlētāji, kuri piedāvā plašu pakalpojumu klāstu un inovatīvas funkcijas. Daži no galvenajiem konkurentiem ir:

* Betway Esports: vietne ir labi pazīstama ar savu plašo kibersporta piedāvājumu, ieskaitot populāras spēles kā "Counter-Strike: Global Offensive", "Dota 2" un "League of Legends". Betway izceļas ar lieliskiem bonusiem jaunajiem lietotājiem un intuitīvu lietotāja interfeisu, kas padara derību procesu vienkāršu un pieejamu.
* GG.Bet: specializējas kibersportā un piedāvā plašu likmju klāstu uz dažādām spēlēm un turnīriem. GG.Bet priekšrocības ir ātras izmaksas, plaša likmju daudzveidība un tiešraides straumēšanas iespējas, kas ļauj lietotājiem sekot līdz spēlēm un veikt likmes reāllaikā.

# UZDEVUMA NOSTĀDNE

Kvalifikācijas darba uzdevums ir izveidot Sporta spēles derību tīmekļu vietni ar pamatvirzieni kibersports. Sistēmā nepieciešams realizēt iespēju veikt likmes uz sporta spēlēm.

Bukmeikeru vietne piedāvās vairākas būtiskas funkcijas, lai nodrošinātu lietotājiem visaptverošu un augstvērtīgu kibersporta derību pieredzi. Lietotāji varēs viegli reģistrēties, pieteikties un pārvaldīt savus kontus, ieskaitot depozītu veikšanu un laimestu izņemšanu. Piedāvāsim derības uz populārākajiem kibersporta veidiem, kā arī uz mazāk zināmām, bet strauji augošām spēlēm. Lietotāji varēs veikt likmes reāllaikā, skatoties spēļu tiešraides, lai iegūtu dinamisku un aizraujošu pieredzi. Vietnē būs ieviesti augsti drošības standarti un pieejams klientu atbalsts, lai nodrošinātu drošu un uzticamu lietotāja pieredzi. Ar šīm funkcijām vietne spēs nodrošināt augstas kvalitātes pakalpojumus un izcelties kibersporta derību tirgū.

Ir plānotas vairākas funkcijas (skatīt lietojumgadījuma diagramma 1. pielikuma.):

* likmju veikšana;
* lietotāju reģistrācija un pieteikšanās;
* likmju un derību iespēju skatīšana;
* konta informācijas pārvaldīšana;
* līdzekļu iemaksa un izņemšana;
* piekļuve spēļu statistikai;
* paziņojumu saņemšana par spēļu atjauninājumiem un reklāmas piedāvājumiem.

# PRASĪBU SPECIFIKĀCIJA

## 2.1. Ieejas un izejas informācijas apraksts

### ***Ieejas informācijas apraksts***

Sistēmā tiks nodrošināta šādas ieejas informācijas apstrāde.

1. Informācija par **lietotājiem** sastāvēs no šādiem datiem:

* lietotājvārds – burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Danila”);
* parole– burtu teksts ar izmēru līdz 144 rakstzīmēm. (piem., “qwerty123@”);
* epasta adrese– burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “danila@gmail.com”);
* loma – dati no tabulas lomas admin/user. (piem., “Admin”);
* atlikums – datu tips float. (piem., “100,12”).

2. Informācija par **lomas** sastāvēs no šādiem datiem:

* lomas nosaukums – burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Admin”).

3. Informācija par **kategorijā** sastāvēs no šādiem datiem:

* kategorijās nosaukums – burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Valorant”);
* kanāla nosaukums - burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “ESL”).

4. Informācija par **komandam** sastāvēs no šādiem datiem:

* komandas nosaukums – burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Faze clan”).

5. Informācija par **komandu spēles** sastāvēs no šādiem datiem:

* komanda – burtu teksts ar izmēru līdz 30 rakstzīmēm, dati no tabulas komandas. (piem., “Faze clan”);
* koeficients – datu tips float. (piem., “1.8”);
* likmes summa – daut tips float. (piem., “100,50”).

6. Informācija par **speles** sastāvēs no šādiem datiem:

* kategorijā– burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Valorant”);
* 1 komanda– burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Faze clan”);
* 2 komanda– burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Spirit”);
* speles tips – datu izvele manual/auto. (piem., “Prematch”);
* aktiva spēle- datu tips Boolean. (piem., “true”);
* atvērta spēle- datu tips Boolean. (piem., “true”);
* spēles sakums – datu tips DateTime. (piem., “11/03/2005 15min”);
* spēles beigums- datu tips DateTime. (piem., “11/03/2005 18min”);
* uzvarētājs- burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “Faze clan”);
* kanāla nosaukums - burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “ESLCS”).

7. Informācija par **likmes** sastāvēs no šādiem datiem:

* summa– datu tips float. (piem., “100,50”);
* spēle – burtu teksts ar izmēru līdz 30 rakstzīmēm. (piem., “CS2”);
* komandas izvēle– izvēle team1/team2. (piem., “team1”);
* atgrieztas likmes – datu tips Boolean. (piem., “true”).

Lietotāja informācija tiks ievadīta reģistrācijas laikā vai lietotāja lapā, izmantojot tastatūru. Maksājuma informāciju lietotājs ievadīs arī savā profila lapā. Datus par likmēm administrators ievadīs administratora panelī.

### ***2.1.2. Izejas informācijas apraksts***

1. **Darījumu vēsture**: iepriekšējo darījumu kopsavilkums, tostarp iemaksas, izņemšana un veiktās likmes. Darījumu datumi un summas. Darījumos izmantotās norēķinu metodes.
2. **Konta atlikums**: pašreizējais atlikums lietotāja derību kontā. Sīkāka informācija par nesenajiem noguldījumiem un izņemšanu, kas ietekmē atlikumu.
3. **Derību rezultāti**: sporta notikumu vai spēļu rezultāti, uz kuriem lietotājs ir veicis likmes. Uzvaras vai zaudējumi no likmēm, kas veiktas uz konkrētiem notikumiem.

## 2.2. Funkcionālās prasības

1. Reģistrācija.

1.1. Sistēmai jānodrošina jauna lietotāja reģistrācijas iespēja, ievadot lietotājvārdu, e-pastu un paroli.

1.2. Ja ievadītā e-pasta adrese jau ir reģistrēta, sistēmai jāinformē lietotājs par esošo reģistrāciju un jāiesaka atjaunot paroli.

1.3. Visi obligātie reģistrācijas formas lauki jāaizpilda. Ja lauks nav aizpildīts, sistēmai jāizdod attiecīgs kļūdas ziņojums.

1.4. Sistēmai jāpārbauda ievadītā e-pasta un paroles formāta pareizība. Ja formāts ir nepareizs, sistēmai jāizdod attiecīgs kļūdas ziņojums.

2. Autorizācija.

2.1. Lietotāja autorizācijai nepieciešams ievadīt pareizu lietotājvārdu (vai e-pastu) un paroli.

2.2. Ja ievadīts nepareizs lietotājvārds vai parole, sistēmai jāizdod kļūdas ziņojums.

2.3. Ja lietotājs aizmirst paroli, sistēmai jānodrošina paroles atjaunošanas iespēja caur e-pastu.

2.4. Pēc veiksmīgas lietotāja autorizācijas sistēmai jānovirza lietotājs uz galveno lapu.

3. Logout poga galvenajā lapā.

3.1. Galvenajā lapā jābūt pieejamai "Logout" pogai autorizētiem lietotājiem.

3.2. Nospiežot "Logout" pogu, sistēmai jāizlogojas lietotājam un jānovirza viņš uz autorizācijas lapu.

3.3. Sistēmai jānodrošina logout operācijas drošība, novēršot nesankcionētu piekļuvi pēc izlogēšanās.

4. Sistēmai jānodrošina iespēja skatīties tiešraidi pleijerī galvenajā lapā.

5. Lietotājam jābūt iespējai izvēlēties pieejamās kategorijas no saraksta.

6. Meklēšana pēc kategorijām.

6.1. Sistēmai jānodrošina meklēšanas funkcija, lai lietotājs varētu meklēt pēc kategorijām.

6.2. Meklēšanai jādarbojas pēc atslēgvārda pat tad, ja pieprasījums nav pabeigts.

6.3. Vaicājumā var izmantot visas rakstzīmes.

7. Lietotājam jābūt iespējai izvēlēties likmju veidu: prematch vai live.

7.1 Pēc kategorijas izvēles lietotājs redzēs filtrētas spēles.

7.2. Sistēmai jānodrošina maču šķirošanas funkcija pēc dažādiem kritērijiem (datuma, popularitātes u.c.).

8. Likmes uzlikšanas iespēja uz komandu.

8.1. Lietotājam jābūt iespējai uzlikt likmi uz izvēlēto komandu.

8.2. Lietotājs var izvēlēties pogu ar koeficientu komandai, uz kuru viņš vēlas likt likmes

8.3. Lietotājs varēs izvēlēties summu, kas ir pieejama viņa kontā, ja viņš izvēlēsies lielāku summu, likme tiks atcelta un tiks parādīta kļūda.

9. Lietotājam jābūt iespējai skatīt savas aktīvās likmes.

10. Galvenajā lapā jābūt pieejamai "Kontakti" pogai, kas novirza uz kontaktu lapu.

11. Galvenajā lapā jābūt pieejamai "Par mums" pogai, kas novirza uz lapu ar informāciju par uzņēmumu.

12. Galvenajā lapā jābūt pieejamai "Profils" pogai, kas novirza uz lietotāja profila lapu.

13. Lietotājs, nospiežot uzņēmuma logotipu, tiek automātiski novirzīts uz galveno lapu.

14. Lietotāja personīgā informācija.

14.1. Lietotājam jābūt iespējai skatīt savu personīgo informāciju, tostarp vārdu, uzvārdu, e-pasta adresi.

14.2. Lietotājam jābūt iespējai rediģēt savu personīgo informāciju.

15. Paroles maiņas iespēja.

15.1. Lietotājam jābūt iespējai nomainīt paroli savā profilā, ievadot pašreizējo paroli un jauno paroli.

15.2. Jaunajai parolei jāatbilst noteiktajām drošības prasībām (garums, speciālie simboli, utt.).

15.3. Ja ievadītā pašreizējā parole ir nepareiza, sistēmai jāizdod attiecīgs kļūdas ziņojums.

16. Atlikuma Papildināšana.

16.1. Lietotājam jābūt iespējai ievadīt savas kartes datus (kartes numurs, derīguma termiņš, CVV kods) depozīta veikšanai.

16.2. Lietotājam jābūt iespējai izvēlēties depozīta summu.

16.3. Pēc depozīta veikšanas sistēmai jāsniedz apstiprinājums par veiksmīgu darījumu.

16.4. Lietotājam jābūt iespējai lejupielādēt kvīti par veikto depozītu PDF formātā.

16.5. Ja depozīts neizdodas, sistēmai jāsniedz attiecīgs kļūdas ziņojums un jānorāda iespējamie risinājumi.

17. Izņemšanas sadaļa.

17.1. Lietotājam jābūt iespējai izvēlēties izņemšanas summu no sava konta.

17.2. Lietotājam jābūt iespējai ievadīt kartes datus, uz kuru vēlas saņemt izņemto summu.

17.3. Pēc izņemšanas pieprasījuma iesniegšanas sistēmai jānosūta apstiprinājuma e-pasts lietotājam.

17.4. Ja izņemšana neizdodas, sistēmai jāsniedz attiecīgs kļūdas ziņojums un jānorāda iespējamie risinājumi.

17.5. Lietotājam jābūt iespējai skatīt visu izņemšanas pieprasījumu vēsturi.

18. Transakciju vēstures skatīšanās iespēja.

18.1. Lietotājam jābūt iespējai skatīt visu veikto transakciju vēsturi, tostarp depozītus un izņemšanas.

18.2. Transakciju vēsturei jāiekļauj transakciju datumi, summas, transakciju veidi un statuss (veiksmīga, neizdevusies).

19. Lietotāju tabulā administrators var pievienot lietotājam atlikumu.

19.1. Administratoram jābūt iespējai pievienot vai mainīt lietotāja atlikumu lietotāju tabulā.

19.2. Administratoram jābūt iespējai meklēt lietotāju pēc lietotājvārda, e-pasta vai lietotāja ID.

19.3. Administratoram jābūt iespējai redzēt pašreizējo lietotāja atlikumu.

19.4. Administratoram jābūt iespējai ievadīt summu, kuru pievienot vai atņemt no lietotāja atlikumu.

20. Administrators var bloķēt lietotāju.

20.1. Administratoram jābūt iespējai bloķēt lietotāju no lietotāju tabulas.

20.3. Administratoram jābūt iespējai atbloķēt lietotāju, ja nepieciešams.

21. Administratoram jābūt iespējai skatīt likmju vēsturi un atgriezt tās tehnisku problēmu gadījumā.

3.1. Administratoram jābūt iespējai skatīt lietotāju likmju vēsturi, tostarp likmes datumu, summu un iznākumu.

3.2. Administratoram jābūt iespējai filtrēt likmju vēsturi pēc datuma, lietotāja vai spēles.

3.3. Ja likme jāatgriež tehnisku problēmu dēļ, administratoram jābūt iespējai manuāli atgriezt likmes summu lietotājam.

22. Komandu tabulā administrators var izveidot un dzēst komandas.

22.1. Administratoram jābūt iespējai izveidot jaunu komandu, ievadot komandas nosaukumu un citu nepieciešamo informāciju.

22.2. Administratoram jābūt iespējai rediģēt esošās komandas informāciju.

22.3. Administratoram jābūt iespējai dzēst komandas no sistēmas.

22.4. Pirms komandas dzēšanas sistēmai jābrīdina administrators par sekām (piemēram, saistīto spēļu dzēšanu).

23. Kategoriju tabulā administrators var izveidot un dzēst kategorijas.

23.1. Administratoram jābūt iespējai izveidot jaunu kategoriju, ievadot kategorijas nosaukumu un citu nepieciešamo informāciju.

23.2. Administratoram jābūt iespējai rediģēt esošās kategorijas informāciju.

23.3. Administratoram jābūt iespējai dzēst kategorijas no sistēmas.

24. Spēļu tabulā administrators var izveidot un dzēst spēles.

24.1. Administratoram jābūt iespējai izveidot jaunu spēli, ievadot spēles nosaukumu, datumu, komandas un citus nepieciešamos parametrus.

24.2. Administratoram jābūt iespējai rediģēt esošās spēles informāciju.

24.3. Administratoram jābūt iespējai dzēst spēles no sistēmas.

25. Spēļu tabulā administrators var šķirot spēles.

25.1. Administratoram jābūt iespējai šķirot spēles pēc dažādiem kritērijiem, piemēram, pēc datuma, popularitātes vai statusa (aktīva, beigusies).

25.2. Administratoram jābūt iespējai izmantot vairākus šķirošanas filtrus vienlaikus.

26. Aktīvo spēļu skatīšanās iespēja ar koeficientu mainīšanas un uzvarētāju izvēles iespējām.

26.1. Administratoram jābūt iespējai skatīt visas aktīvās spēles reāllaikā.

26.2. Administratoram jābūt iespējai mainīt spēļu koeficientus atbilstoši aktuālajai situācijai.

26.3. Administratoram jābūt iespējai manuāli izvēlēties un atzīmēt spēles uzvarētājus pēc spēles beigām.

26.4. Pēc uzvarētāju izvēles sistēmai automātiski jāaprēķina un jāizmaksa laimesti lietotājiem.

## 2.3. Nefunkcionālās prasības

1. Sistēmas veiktspējas prasības sistēmai jāspēj apstrādāt vismaz 100 vienlaicīgus lietotājus bez veiktspējas samazināšanās. Maksimālais atbildes laiks no servera nedrīkst pārsniegt 2 sekundes, lai nodrošinātu ātru lietotāju pieredzi.

2. Sistēmas pieejamība un uzticamība Sistēmai jābūt pieejamai 95% laika, lai nodrošinātu nepārtrauktu pakalpojuma sniegšanu. Jābūt izveidotai automātiskai rezerves kopēšanas sistēmai, kas nodrošina datu drošību un atjaunošanu.

3. Drošības prasības sistēmai jānodrošina datu šifrēšana gan pārsūtīšanas laikā, gan datu glabāšanas laikā. Lietotājiem jābūt iespējai izveidot stipru paroli, kas atbilst noteiktiem drošības kritērijiem (vismaz 8 rakstzīmes, speciālie simboli, cipari).

4. Lietotāja interfeisa prasības sistēmas saskarnes valodai jābūt angļu valodā. Dizainam jāatbilst uzņēmuma stila vadlīnijām, izmantojot noteiktās krāsas un fontus.

5. Pielāgojamība un mobilitāte sistēmai jābūt pielāgojamai jebkuram ekrāna izmēram, lai to varētu izmantot gan datorā, gan planšetdatorā, gan viedtālrunī. Jānodrošina, ka visas funkcionalitātes ir pieejamas mobilajās ierīcēs.

6. Lietotāju atbalsta prasības Jābūt pieejamam 24/7 klientu atbalstam, lai palīdzētu lietotājiem ar jebkādām problēmām. Jānodrošina tiešsaistes čats, e-pasta atbalsts un tālruņa līnija.

7. Pārvaldības prasības sistēmai jābūt iespējai izsekot lietotāju aktivitātēm, lai novērstu krāpnieciskas darbības. Administratīvā paneļa piekļuvei jābūt ierobežotai un drošai, izmantojot divfaktoru autentifikāciju.

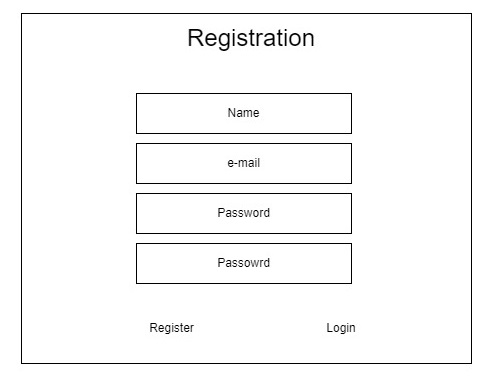
8. Datu integritātes prasības visām transakcijām jābūt precīzām un jānodrošina, ka dati netiek zaudēti vai bojāti. Jānodrošina, ka lietotāju dati tiek glabāti atbilstoši visiem tiesiskajiem un regulatīvajiem standartiem.

9. Lietojamības prasības sistēmai jābūt intuitīvai un viegli lietojamai, lai lietotāji varētu ātri atrast un izmantot nepieciešamās funkcijas. Jānodrošina visaptveroša palīdzības sadaļa ar biežāk uzdotajiem jautājumiem un atbildēm.

10. Pārejamības prasības jauna sistēmas versija jāievieš bez lietotāju pieredzes traucējumiem. Jānodrošina, ka visa iepriekšējā versijā saglabātā informācija tiek veiksmīgi pārnesta uz jauno versiju. Šīs prasības nodrošinās sistēmas drošību, uzticamību, ātrumu un lietojamību, kā arī nodrošinās labu lietotāju pieredzi.

Sistēmas ekrānu skices:

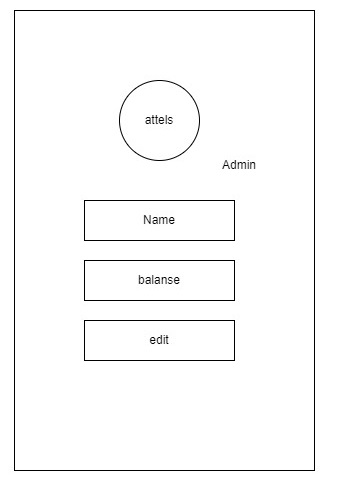
* sistēmas reģistrācijas skice (skat 2.1. att.);



2.1.att. Sistēmas reģistrācijas skice

Šī skice attēlo sistēmas interfeisu, kurā lietotāji var reģistrēties.

* sistēmas profila skice (skat 2.2. att.).



2.2.att. Sistēmas profila skice

Šī skice attēlo profila interfeisu. Lietotājs var mainīt savus datus un pievienot balansi.

1. **UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS**

Aplīkācija sludinājumu portāla sistēma būs web aplikācija, tāpēc bija izveletie rīki priekš web aplikācijas izstradei.

**Node.js(express):**

* versija: 4.18.3;
* raksturojums: Express.js vai vienkārši Express ir tīmekļa lietojumprogrammu sistēma Node.js, kas ieviesta kā bezmaksas un atvērtā pirmkoda programmatūra saskaņā ar MIT licenci;
* izmantošanas iemesls: Express for Node.js tiek izmantots ātrai tīmekļa lietojumprogrammu un API izstrādei, nodrošinot maršrutēšanas funkcionalitāti, pieprasījumu apstrādi un ērtu integrāciju ar dažādu starpprogrammatūru. Tas atvieglo mērogojamu servera puses risinājumu izveidi JavaScript.

**MySQL:**

* versija: 8.0;
* raksturojums: MySQL ir bezmaksas relāciju datu bāzes pārvaldības sistēma;
* izmantošanas iemesls: MySQL tiek izmantota kā datu bāzes pārvaldības sistēma (DBMS) lietojumprogrammu izmantoto datu glabāšanai, pārvaldībai un izguvei, nodrošinot efektīvu datu pārvaldību un atbalstu transakciju operācijām.

**Docker:**

* Versija: 6.4;
* raksturojums: Docker ir programmatūra lietojumprogrammu izvietošanas un pārvaldības automatizēšanai konteinerizētās vidēs, lietojumprogrammu konteineri;
* izmantošanas iemesls: Docker tiek izmantots, lai izstrādātu, piegādātu un palaistu lietojumprogrammas, izmantojot konteinerus, ļaujot jums pakotēt lietojumprogrammu ar visu tās vidi un atkarībām standartizētā vienībā, tādējādi nodrošinot vides konsekvenci visos izstrādes un darbības posmos.

**Sass:**

* versija: 1.72.0;
* raksturojums: Sass ir uz CSS balstīta metavaloda, kas izstrādāta, lai palielinātu CSS koda abstrakcijas līmeni un vienkāršotu kaskādes stila lapu failus;
* SASS (Syntactically Awesome Style Sheets) tiek izmantots, lai vienkāršotu un paplašinātu CSS jaudu, nodrošinot tādas funkcijas kā mainīgie, ligzdošana, mixins un funkcijas, atvieglojot stilu organizēšanu un uzturēšanu lielos projektos.

**HTML, CSS, JavaScript:**

* versija: HTML5, CSS3, ECMAScript 6+;
* raksturojums: šīs tehnoloģijas ir standarts tīmekļa lietotņu front-end izstrādē, nodrošinot struktūru, stilu un interaktivitāti;
* izmantošanas iemesls: tās tika izvēlētas, lai nodrošinātu lietotāja saskarnes struktūru, stilu un interaktivitāti, kas ir nepieciešama dinamiskai un reaģējošai tīmekļa lietojumprogrammai.

**React:**

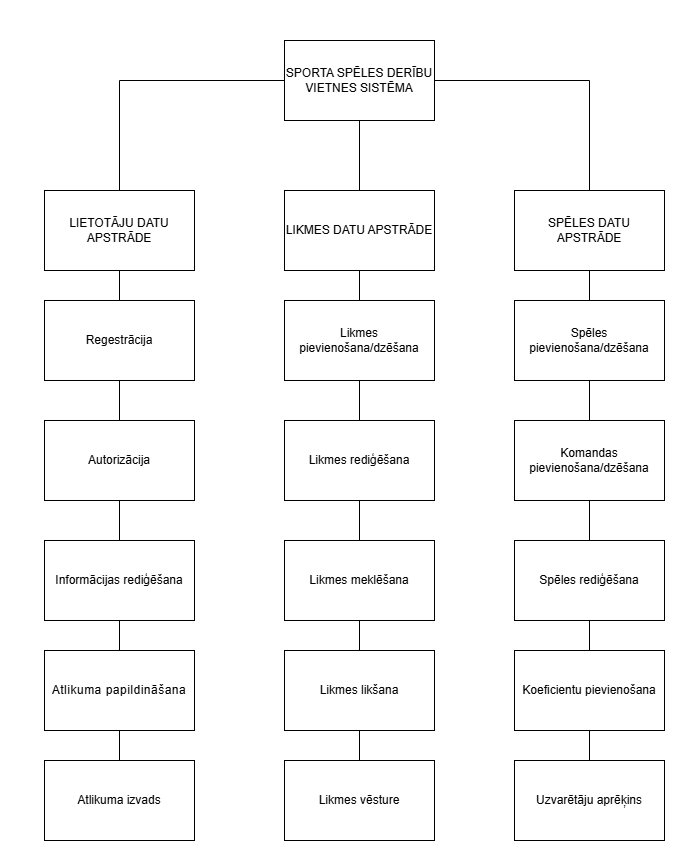
* versija: 18.2;
* raksturojums: React ir deklaratīva, efektīva un elastīga JavaScript bibliotēka lietotāja saskarnes izstrādei;
* izmantošanas iemesls: React tika izvēlēts, lai izstrādātu interaktīvu un dinamisku lietotāja saskarni, izmantojot komponentu bāzētu pieeju, kas atvieglo uzturēšanu un paplašināšanu.

# PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA

## . Sistēmas struktūras modelis

### ***4.1.1.*** ***Sistēmas arhitektūra***

Sistēma ir sadalīta uz 3 daļam: lietotāji dati, likmes dati, spēles dati. katrā daļa izplidā savas darbības un atbild par savu jomu. Funkcionālās dekompozicijas diagramma (skat. 3. att.).



3.att. Funkcionālās dekompozīcijas diagramma

**1. Lietotāju datu apstrāde**

Šī apakšsistēma nodrošina visu, kas saistīts ar lietotāju reģistrāciju, autorizāciju un informācijas pārvaldību:

* reģistrācija: nodrošina jaunu lietotāju pievienošanu sistēmai;
* autorizācija: veic lietotāju autentifikāciju un piekļuves piešķiršanu;
* informācijas rediģēšana: ļauj lietotājiem atjaunināt savu personīgo informāciju;
* atlikuma papildināšana: nodrošina iespēju papildināt lietotāja konta atlikumu;
* atlikuma izvadīšana: ļauj lietotājiem izņemt līdzekļus no sava konta.

**2. Likmes datu apstrāde**

Šī apakšsistēma ir atbildīga par visu darbību, kas saistīta ar likmēm:

* likmes pievienošana/rediģēšana;
* nodrošina jaunu likmju ievietošanu un esošo likmju atjaunināšanu;
* likmes rediģēšana: ļauj rediģēt esošās likmes;
* likmes meklēšana: nodrošina iespēju meklēt likmes pēc noteiktiem kritērijiem;
* likmes likšana: ļauj lietotājiem veikt likmes;
* likmes vēsture: nodrošina piekļuvi likmju vēsturei.

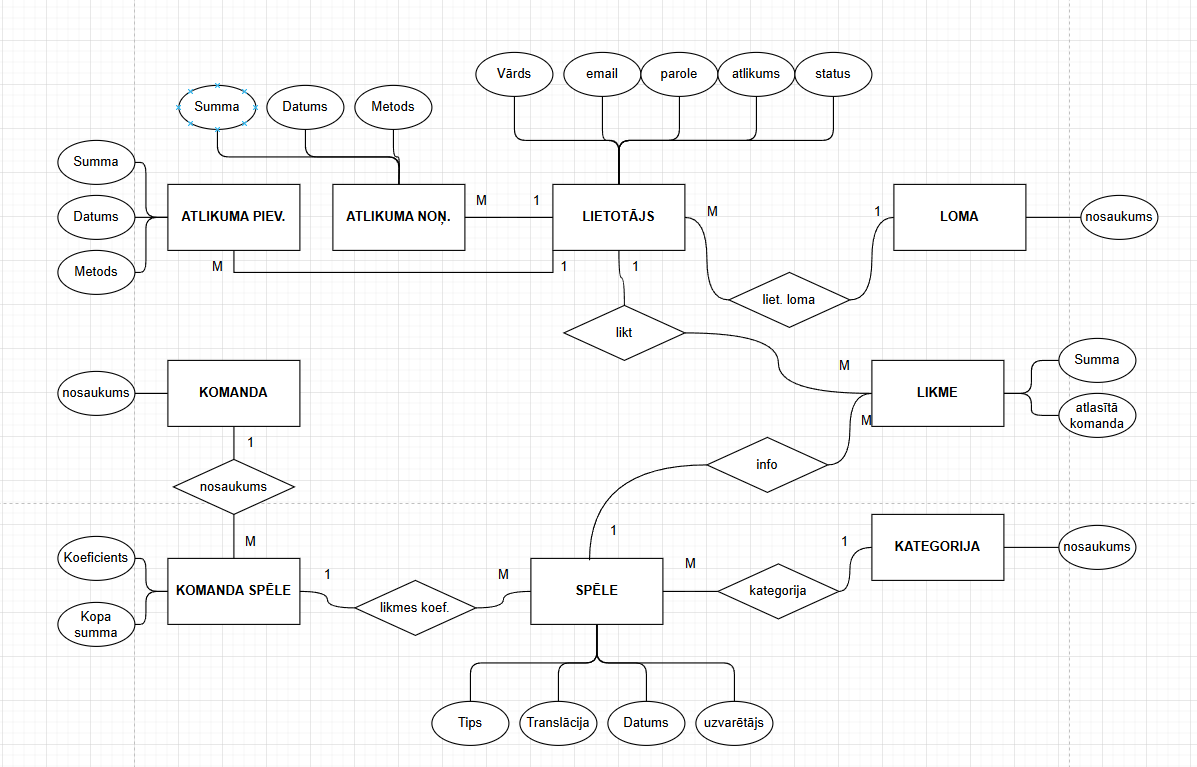
**3. Spēles datu apstrāde**

Šī apakšsistēma pārvalda visu informāciju, kas saistīta ar sporta spēlēm un komandām:

* spēles pievienošana/rediģēšana: nodrošina jaunu spēļu pievienošanu un esošo spēļu rediģēšanu;
* komandas pievienošana/rediģēšana: ļauj pievienot un rediģēt komandas;
* spēles rediģēšana: nodrošina spēļu datu rediģēšanu;
* koeficientu pievienošana: ļauj pievienot un rediģēt spēļu koeficientus;
* uzvarētāju aprēķins: nodrošina uzvarētāju noteikšanu un attiecīgo izmakšu aprēķinu;
* šis apraksts sniedz pārskatu par katras apakšsistēmas funkcionalitāti un tajās ietvertajiem moduļiem, balstoties uz diagrammu.

## 

## 4.2. Sistēmas ER modelis

 Sistēmas ER-modelis sastāv no 9 entitijām (skat. 4. att.), kas nodrošina pamat informācijas uzglabāšanu un apstrādi. Tie ir lietotājs, loma, likme, kategorija, spēle, komanda spēle, komanda, atlikuma pievienošana, atlikuma noņemšana.

4.att. Sistēmas ER-diagramma

Šis apraksts atspoguļo savstarpējās saites starp dažādām entītijām ER-modeļa diagrammā, nodrošinot saprotamu skaidrojumu par katras entītijas attiecībām.

1. Lietotājs - atlikuma papildināšana.

Starp entītijām "lietotājs" un "atlikuma papildināšana" ir attiecība viens pret daudziem, jo vienam lietotājam var būt vairākas atlikuma papildināšanas darbības, bet katra atlikuma papildināšana attiecas uz vienu lietotāju.

2. Lietotājs - atlikuma noņemšana .

Starp entītijām "lietotājs" un "atlikuma noņemšana" ir attiecība viens pret daudziem, jo vienam lietotājam var būt vairākas atlikuma noņemšanas darbības, bet katra atlikuma noņemšana attiecas uz vienu lietotāju.

3. Lietotājs – likme.

Starp entītijām "lietotājs" un "likme" ir attiecība viens pret daudziem, jo vienam lietotājam var būt vairākas likmes, bet katra likme attiecas uz vienu lietotāju.

4. Likme – spēle.

Starp entītijām "likme" un "spēle" ir attiecība daudzi pret vienu, jo vienai spēlei var būt vairākas likmes, bet katra likme attiecas uz vienu spēli.

5. Likme - komandas spēle.

Starp entītijām "likme" un "komanda spēle" ir attiecība daudzi pret vienu, jo viena komandas spēle var attiekties uz vairākām likmēm, bet katra likme ir saistīta ar vienu komandas spēli.

6. Spēle - komandas spēle.

Starp entītijām "spēle" un "komanda spēle" ir attiecība viens pret daudziem, jo vienai spēlei var būt vairākas komandu spēles, bet katra komandas spēle attiecas uz vienu spēli.

7. Komanda - komanda spēle.

Starp entītijām "komanda" un "komanda spēle" ir attiecība viens pret daudziem, jo viena komanda var piedalīties vairākās spēlēs, bet katra komandas spēle attiecas uz vienu komandu. 8. 8. Spēle - kategorija.

Starp entītijām "spēle" un "kategorija" ir attiecība daudzi pret vienu, jo vienai kategorijai var būt vairākas spēles, bet katra spēle ir saistīta ar vienu kategoriju.

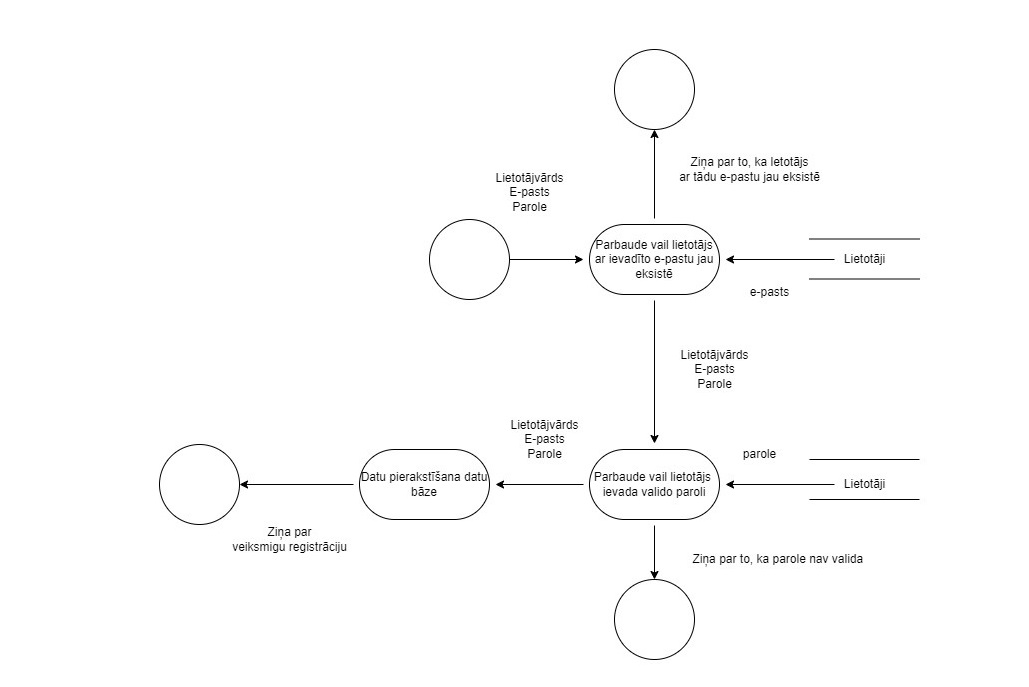
9. Lietotājs - loma.

Starp entītijām "lietotājs" un "loma" ir attiecība daudzi pret daudziem, jo viena loma var būt vairākiem lietotājiem, bet, no otras puses, vienam lietotājam var būt vairākas lomas.

**4.3. Datu plūsmu modelis**

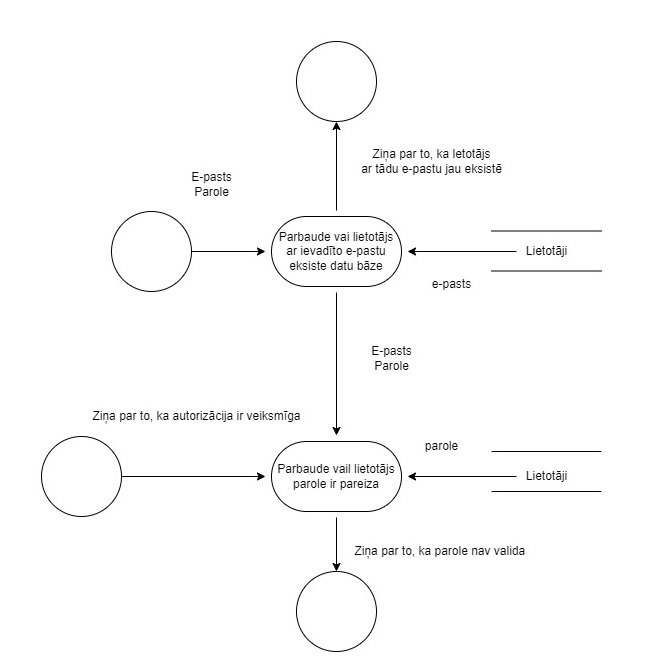
1. **Lietotāja registrācija** (skat. 5.1. att.)

Pēc lietotāja datu ievades sistēma veic trīs pārbaudes. Vispirms sistēma pārbauda, vai lietotājs ar tādu e-pastu jau eksistē, un tad pārbauda, vai lietotājs ir ievadījis validus paroles. Ja dati veiksmīgi pārbaudīti, tie tiek pievienoti datu bāzei, un sistēma informē lietotāju par veiksmīgu reģistrāciju. Ja kāda no pārbaudēm nav izpildīta, sistēma informē lietotāju par kļūdam un izvada ziņus.



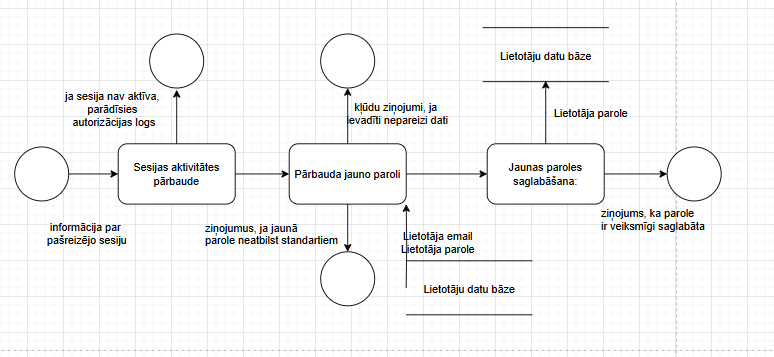
5.1.att. Datu plūsmu diagramma – reģistrācija

1. **Lietotaja autorizācija**(skat. 5.2. att.)

Pēc lietotāja datu ievades tie veic divas pārbaudes. Vispirms sistēma pārbauda, vai eksistē lietotājs ar tādu e-pastu, un vai lietotājs ir ievadījis pareizo paroli savam e-pastam. Ja dati ir veiksmīgi pārbaudīti, sistēma informē lietotāju par veiksmīgu autorizāciju. Ja kāda no pārbaudēm nav izpildīta, sistēma informē lietotāju par kļūdam un izvada ziņus.

5.2.att. Datu plūsmu diagramma – autorizācija

1. **Profila paroles maiņa** (skat. 5.3. att.)

Šī funkcija ļauj lietotājiem mainīt savu paroli. Process sākas, pārbaudot, vai lietotāja sesija ir aktīva. Ja lietotājs nav pieteicies, viņš tiek novirzīts uz sākumlapu. Pēc tam sistēma pārbauda, ​​vai lietotājs ir norādījis jaunu paroli. Ja jā, tad tiek pārbaudīts, vai ievadītās paroles atbilst noteiktajām drošības prasībām.

5.3.att. Datu plūsmu diagramma - Biļešu rediģēšanas process

1. **Atlikuma papildināšana** (skat. 5.4. att.).

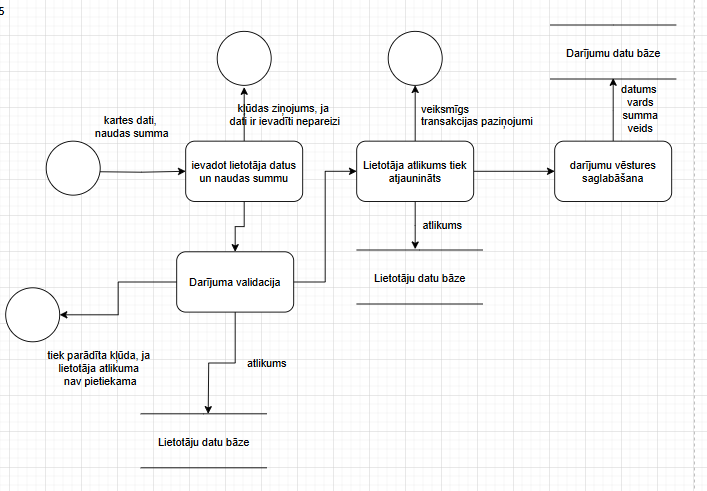
Šī funkcija ļauj lietotājiem papildināt savu bilanci sporta derību vietnē. Process sākas ar to, ka lietotājs ievada papildināšanas summu un izvēlas maksājuma veidu. Tālāk sistēma pārbauda ievadīto datu pareizību un veic darījumu. Pēc veiksmīgas darījuma pabeigšanas lietotāja atlikums tiek atjaunināts un tiek parādīts papildināšanas apstiprinājums. Ja rodas kļūda, tiek parādīts atbilstošs ziņojums.



5.4.att. Datu plūsmu diagramma – likmes izveide

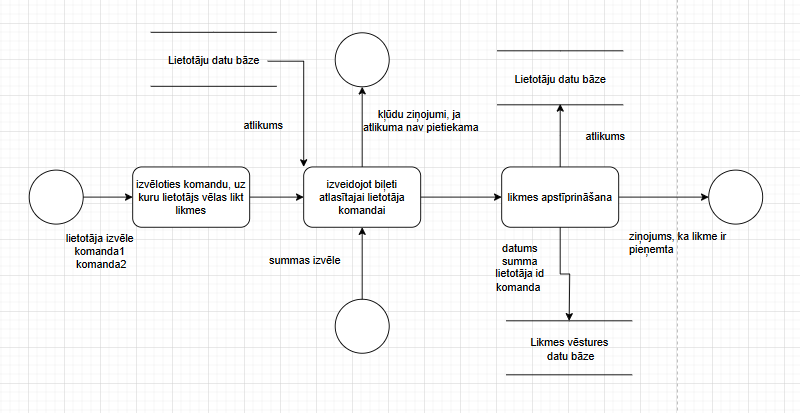
1. **Atlikuma izvāds** (skat. 5.5. att.)

Šī funkcija ļauj lietotājiem izņemt līdzekļus no sava sporta derību vietnes konta. Process sākas ar to, ka lietotājs ievada izņemšanas summu un izvēlas izņemšanas metodi. Tālāk sistēma pārbauda ievadīto datu pareizību un veic darījumu. Kad darījums ir veiksmīgi pabeigts, lietotāja atlikums tiek atjaunināts un tiek parādīts izņemšanas apstiprinājums. Ja rodas kļūda, tiek parādīts atbilstošs ziņojums.



5.5.att. Datu plūsmu diagramma – likmes meklēšanas process

1. **Likmes likšana** (skat. 5.6. att.)

Šī funkcija ļauj lietotājiem veikt likmes uz izvēlētu komandu sporta pasākumos. Process sākas ar to, ka lietotājs izvēlas komandu un ievada likmes summu. Tālāk sistēma pārbauda ievadīto datu pareizību un to, vai lietotājam ir pietiekams atlikums. Pēc veiksmīgas pārbaudes piedāvājums tiek pieņemts un dati tiek atjaunoti datu bāzē. Ja rodas kļūda, tiek parādīts atbilstošs ziņojums.

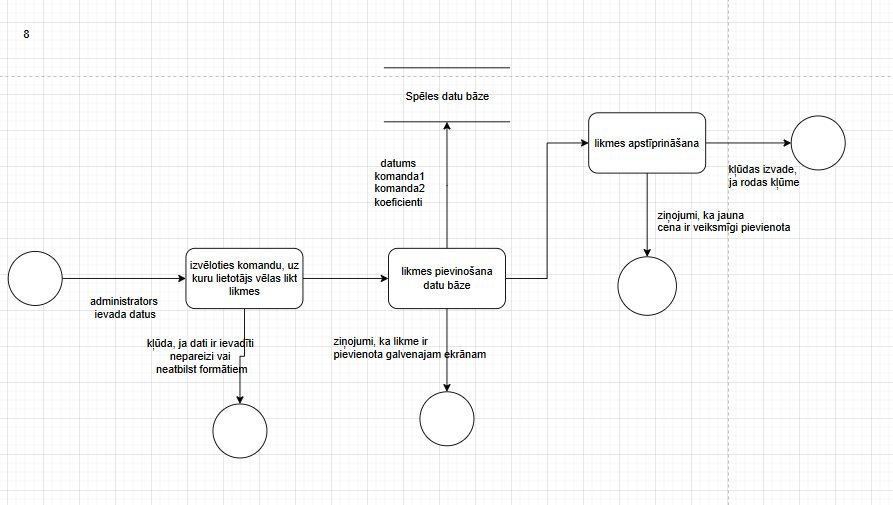
5.6.att. Datu plūsmu diagramma – likmes meklēšanas process

1. **Likmes vēsture** (skat. 5.7. att.).

Šī funkcija ļauj lietotājiem skatīt savu likmju vēsturi sporta derību vietnē. Process sākas ar to, ka lietotājs pieprasa savu derību vēsturi. Sistēma apstrādā pieprasījumu un izgūst cenu datus no datu bāzes. Pēc tam dati tiek filtrēti un formatēti, lai tie parādītu lietotājam. Lietotājs saņem savu iepriekšējo likmju sarakstu ar informāciju par katru likmi.

5.7.att. Datu plūsmu diagramma – likmes meklēšanas process

1. **Spēles pievienošana/dzēšana** (skat. 5.8. att.).

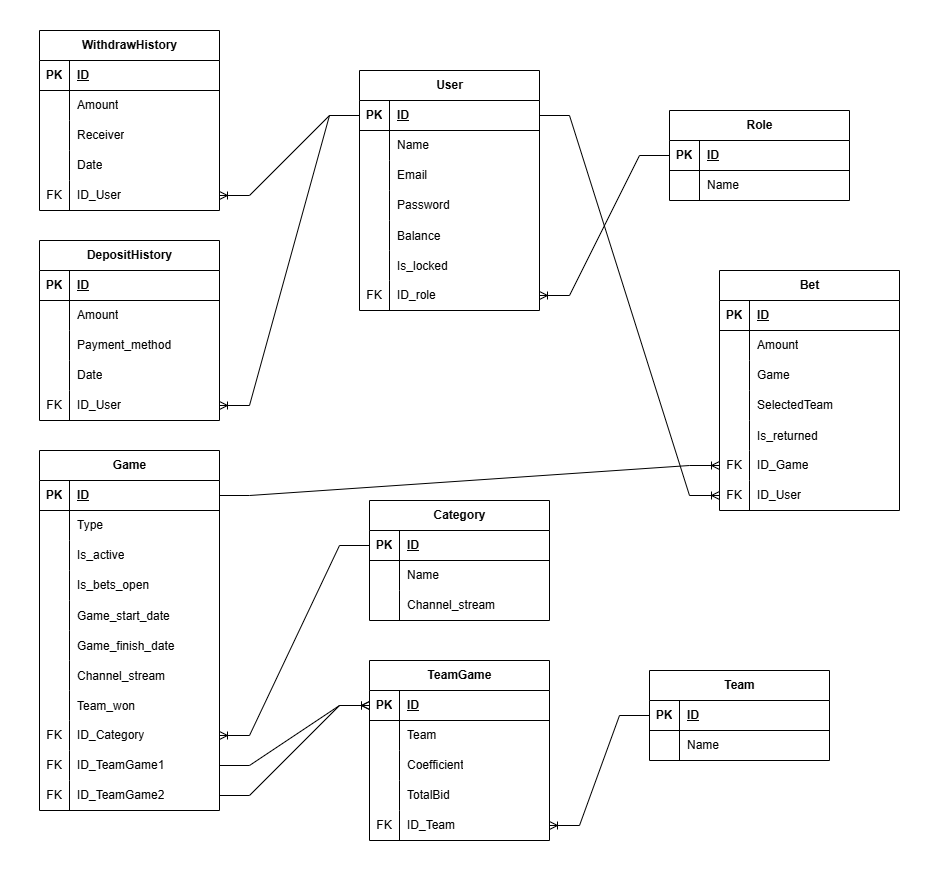
Šī funkcija ļauj administratoram pievienot jaunas spēles sporta derību vietnei. Process sākas ar to, ka administrators ievada jaunās spēles datus, piemēram, komandas nosaukumu, spēles datumu un laiku, koeficientus un citu nepieciešamo informāciju. Tālāk sistēma pārbauda ievadīto datu pareizību. Pēc veiksmīgas verifikācijas spēle tiek pievienota datu bāzei un lietotājiem tiek parādīta vietnē. Ja rodas kļūda, tiek parādīts atbilstošs ziņojums.

5.8.att. Datu plūsmu diagramma – likmes meklēšanas process

# DATU STRUKTŪRU APRAKSTS

Visi dati tiks saglabāti servera datu bāzē. Datu bāzē sastāv no 9 tabulām.

Tabulas datu bāzē:

* tabula "**User**” - glabā lietotāja kontus;
* tabula “**Role**” - glabā datus par lomam;
* tabula “**Category**” - tabula glabā informāciju par spēļu kategorijām;
* tabula “**Team**”- glaba datus par komandam;
* tabula “**TeamGame**” - tabula ievieš relāciju daudz pret daudz un tiek izmantota, lai izveidotu koeficientu 2 komandām spēlē;
* tabula “**Game**” - glabā informāciju par spēlēm;
* tabula “**Bet**”- glabā informāciju par lietotāju veiktajām likmēm;
* tabula “**DepositHistory” -** glabā informāciju par atlikuma papildināšanu;
* tabula “**WithdrawHistory**” - glabā informāciju par atlikuma izvadīšanu.

6.att. Tabulu saišu shēma

1.tabula

Tabulas **“User”** sturktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | name | varchar | 256 | lietotājvārds |
| 3. | email | varchar | 256 | lietotāja e-pasts |
| 4. | password | varchar | 256 | lietotāja parole |
| 5. | balance | float | - | lietotāja atlikums |
| 6. | Is\_locked | boolean | - | lauks rāda, vai lietotājs ir bloķēts |
| 7. | ID\_role | int | 11 | sveša atslēga Role |

Tabula ”User” ir saistīta ar “Role” caur primāru atslēgu ar ID\_role. Tabula ”User” ir saistīta ar “WithdrawHistory” caur primāru atslēgu ar ID\_User. Tabula ”User” ir saistīta ar “DepositHistory” caur primāru atslēgu ar ID\_DepositHistory. Tabula ”User” ir saistīta ar “Bet” caur primāru atslēgu ar ID\_User.

2.tabula

Tabulas **“Role”** struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | name | varchar | 100 | lomas nosaukumi |

Tabula ”Role” ir saistīta ar “User” caur primāru atslēgu ar ID\_role.

3.tabula

Tabulas **“Category”** struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | name | varchar | 256 | kategorijas nosaukumi |
| 3. | Channel\_stream | varchar | 256 | kanāla nosaukums, kurā būs spēles translācija |

Tabula ”Category” ir saistīta ar “Game” caur primāru atslēgu ar ID\_Category.

4.tabula

Tabulas “**Team**” struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | name | varchar | 256 | kanāla virsraksti |

Tabula ”Team” ir saistīta ar “TeamGame” caur primāru atslēgu ar ID\_TeamGame.

5.tabula

Tabulas “**TeamGame**” struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | coefficient | float | - | likmes koeficients |
| 3. | TotalBid | float | - | visu likmju summa |
| 4. | ID\_Team | in | 11 | sveša atslēga Team |

Tabula ”TeamGame” ir saistīta ar “Game” caur primāru atslēgu ar ID\_Game.

6.tabula

Tabulas “**Game**” struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | type | choice | - | likmes tipa izvēle |
| 3. | Is\_active | boolean | - | lauks rādīs, vai likme ir aktīva |
| 4. | Is\_bets\_open | boolean | - | lauks rāda likme atvērta vai slēgta |
| 5. | Game\_start\_date | DateTime | - | spēles sākuma datums |
| 6. | Game\_finish\_date | DateTime | - | spēles beigu datums |
| 7. | Channel\_stream | varchar | 256 | kanāla nosaukums, kurā būs spēles translācija |
| 8. | Team\_won | varchar | 256 | dati par komandu, kas uzvarēja |
| 9. | ID\_Category | int | 11 | sveša atslēga Category |
| 10. | ID\_TeamGame1 | int | 11 | sveša atslēga 1. komandai |
| 11. | ID\_TeamGame2 | int | 11 | sveša atslēga 2. komandai |

Tabula “Game” ir saistīta ar ”Bet” caur primāru atslēgu un ID\_Bet, un ar ”TeamGame” caur tabulas “TeamGame”primāru atslēgu ar ID\_TeamGame1 un ID\_TeamGame2. Tabula “Game” ir saistīta ar ”Category” caur un ID\_Category un tabulas “Category”primāru atslēgu.

7.tabula

Tabulas “**Bet**” struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | Unikālāis kods |
| 2. | amount | varchar | 100 | likmes summa |
| 3. | selectedTeam | choice | - | 1. vai 2. komandas atlase |
| 4. | Is\_returned | boolean | - | lauks parādīs, vai likme ir atgriezta |
| 5. | ID\_Game | int | 11 | sveša atslēga Game |
| 6. | ID\_User | int | 11 | sveša atslēga User |

Tabula “Bet**”,** kas saista saistīta ar “Game” primāra atslēga un ID\_game, un saistīta ar “User” primāra atslēga un ID\_user.

8.tabula

Tabulas “**DepositHistory**” struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | amount | float | - | atlikuma papildināšanas summa |
| 3. | payment\_method | varchar | 256 | papildināšanas metode |
| 4. | date | DateTime | - | papildināšanas datums |
| 5. | ID\_user | int | 11 | sveša atslēga User |

Tabula “DepositHistory**” ir** saistīta ar “User” caur primāru atslēgu un ID\_user.

9.tabula

Tabulas “**WithdrawHistory**” struktūra

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| 1. | id | int | 11 | unikālāis kods |
| 2. | amount | float | - | atlikuma izvades daudzums |
| 3. | receiver | varchar | 256 | adresāta dati |
| 4. | date | DateTime | - | atlikuma izvades datums |
| 5. | ID\_user | int | 11 | sveša atslēga User |

Tabula **“WithdrawHistory” ir** saistīta ar “User” caur primāru atslēgu un ID\_user.

1. **LIETOTĀJA CEĻVEDIS**

**6.1. Sistēmas prasības aparatūrai un programmatūrai**

Lai veiksmīgi darbotos bukmeikeru vietne, nepieciešams ievērot šādas sistēmas prasības.

Vietne ir optimizēta darbam ar šādām pārlūkprogrammu versijām:

* Safari – 13. versija un jaunākas;
* Windows Edge – 98.0.1108.43 un jaunākas;
* Google Chrome – 98.0.4758.107 un jaunākas;
* Opera – 84.0.4316.31 un jaunākas;
* Mozilla Firefox – 98.0.1 un jaunākas.

Lai nodrošinātu stabilu un efektīvu vietnes darbību, ir nepieciešams stabils interneta savienojums ar minimālo ātrumu 10 Mbps.

Lai nodrošinātu vietnes darbību, ir nepieciešams:

* dators vai mobilā ierīce ar vismaz 2 GB brīvas vietas cietajā diskā.;
* procesors ar minimālo takts frekvenci 1 GHz;
* vismaz 2 GB RAM operatīvā atmiņa.

Lai nodrošinātu vietnes darbību, ir nepieciešamsšādās operētājsistēmas**:**

* Windows – Windows 10 vai jaunāka;
* macOS – macOS 10.15 (Catalina) vai jaunāka;
* Linux – modernās Linux distribūcijas ar atjauninātām bibliotēkām.

Šīs prasības nodrošinās, ka vietne darbosies raiti un bez tehniskām problēmām, sniedzot lietotājiem labu pieredzi.

**6.1. Sistēmas instalācija un palaišana**

Lai veiksmīgi instalētu un palaistu bukmeikeru vietni, kas izmanto Express.js un Docker kā backend un React.js kā frontend, sekojiet šiem soļiem:

prasības:

* Node.js un npm (Node Package Manager);
* Docker un Docker Compose;
* Git.

Soli pa solim instrukcija.

1. Koda klonēšana no Git repozitorija. Atveriet termināli un izpildiet komandu: bash copy code git clone <https://github.com/danila-lab/bookmaker> cd bookmaker.
2. Backend (Express.js) uzstādīšana un palaišana:

* lejupielādējiet un uzstādiet nepieciešamos pakotnes: bash copy code cd backend npm install;
* palaižiet Docker konteinerus: bash copy code docker-compose up.

1. Frontend (React.js) uzstādīšana un palaišana:

* lejupielādējiet un uzstādiet nepieciešamos pakotnes: bash copy code cd frontend npm install;
* Palaižiet React serveri: bash Copy code npm start.

1. Sistēmas reģistrācija un piekļuve:

* pārlūkprogrammā atveriet vietni: http://localhost:3000;
* reģistrējieties vietnē, izveidojot lietotāja kontu, lai piekļūtu visām iespējām un funkcionalitātei;
* imantotie pieteikšanās dati atkarīgi no sistēmā iepriekš definētām lomām;

Piemēram:

* + admins: admin@example.com / parole: admin123
  + pietotājs: user@example.com / parole: user123

1. Papildus soļi:
   * + Docker konteineru pārvaldība: lai apturētu Docker konteinerus, izmantojiet komandu: bash Copy code docker-compose down;
     + vides mainīgie: pārliecinieties, ka visi nepieciešamie vides mainīgie ir definēti .env failā gan frontend, gan backend daļām.

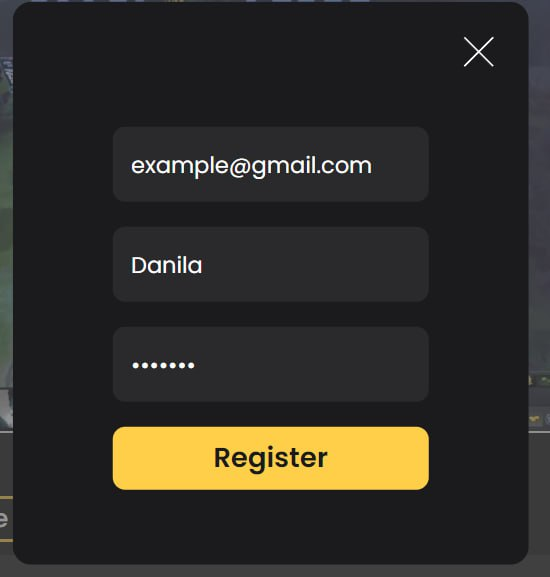
## 6.2. Programmas apraksts

1. **Reģistrācija**

Reģistrācijas forma ir paredzēta jauna lietotāja konta izveidei vietnē. Tā ietver šādus elementus: lauks "email" - lietotājs ievada savu e-pasta adresi reģistrācijai. Lauks "name" - lietotājs ievada savu vārdu. Lauks "password" - lietotājs izveido paroli savam kontam. Poga "register" - pēc visu lauku aizpildīšanas lietotājs nospiež šo pogu, lai pabeigtu reģistrācijas procesu (skat 7. att.).

Piemērs aizpildītai formai:

* email: [example@mail.com](mailto:example@mail.com)
* name: Danila
* password: qwerty@123



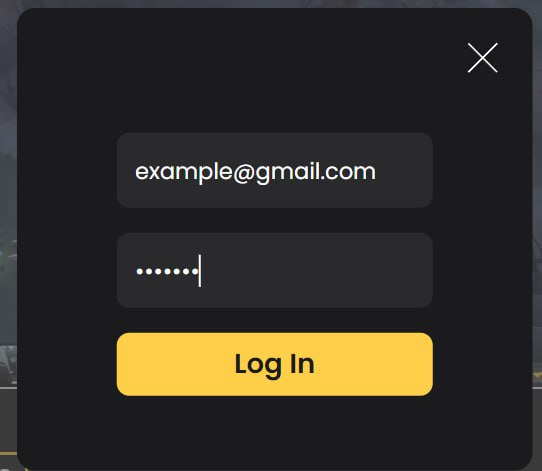
7.att. registrācija

1. **Autorizācija**

Autorizācijas forma ļauj reģistrētiem lietotājiem ieiet savā kontā. Tā ietver šādus elementus: Lauks "email" - lietotājs ievada savu reģistrēto e-pasta adresi. Lauks "password" - lietotājs ievada savu paroli. Poga "log in" - pēc visu lauku aizpildīšanas lietotājs nospiež šo pogu, lai ieietu savā kontā (skat 8. att.).

Piemērs aizpildītai formai:

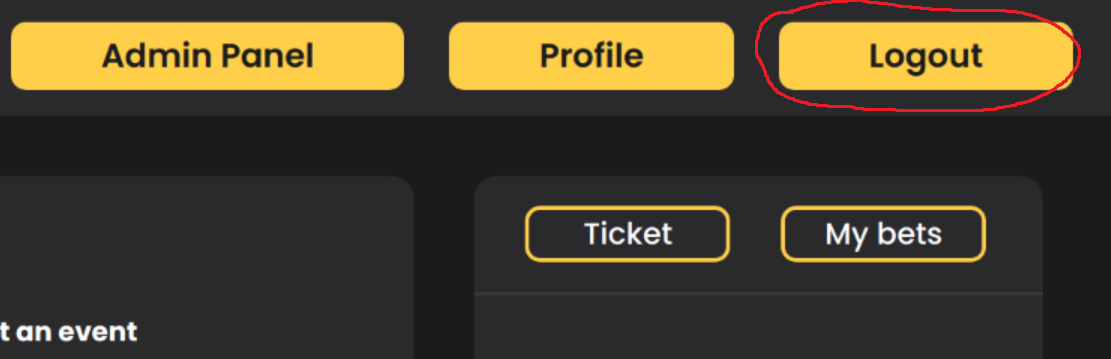
* email: [example@mail.com](mailto:example@mail.com) ;
* password: qwerty123#



8.att. autorizācija

1. **Poga "Logout"**

Poga "logout" ir paredzēta lietotāja izrakstīšanai no sava konta. Tā atrodas vietnes augšējā panelī un nodrošina drošu lietotāja sesijas beigas. Nospiežot šo pogu, notiek pašreizējās sesijas izbeigšana, un lietotājs tiek novirzīts uz autorizācijas vai galveno vietnes lapu (skat 9. att.).



9.att. logout

1. **Kategoriju izvēle**

Kategoriju izvēles logs ļauj lietotājam filtrēt un skatīt likmes tikai pēc izvēlētās spēles kategorijas (skat 10. att.).

**Kategoriju saraksts**:

* katra kategorija tiek attēlota ar atbilstošo ikonu un spēles nosaukumu;
* noklikšķinot uz jebkuras kategorijas (piemēram, Dota2, Valorant, CS2, League of Legends), likmju filtrācija tiek atjaunināta, un lietotājam tiek rādītas tikai izvēlētās kategorijas spēles;
* izvēlētā kategorija tiek vizuāli izcelta (piemēram, ar fona krāsu), lai lietotājs viegli identificētu pašreizējo izvēli.

**Meklēšanas lauks**:

* meklēšanas lauks ļauj lietotājam ātri atrast nepieciešamo spēles kategoriju;
* Ievadot tekstu meklēšanas laukā, kategoriju saraksts tiek atjaunināts reāllaikā, parādot tikai tās kategorijas, kas atbilst ievadītajam pieprasījumam.



10.att. kategoriju izvēlne

1. **Likmes veida izvēles pogas**

Šīs pogas ļauj lietotājam filtrēt likmes pēc to pieejamības statusa: pirms spēles sākuma vai tiešraidē (skat 11. att.).

**Prematch poga**:

* noklikšķinot uz pogas "Prematch", lietotājam tiek parādītas likmes, kuras var veikt tikai pirms spēles sākuma;
* poga tiek vizuāli izcelta, lai parādītu, ka ir aktīva.

**Live poga**:

* noklikšķinot uz pogas "Live", lietotājam tiek parādītas likmes, kuras var veikt tiešraidē, spēles laikā;
* poga tiek vizuāli izcelta, lai parādītu, ka ir aktīva.



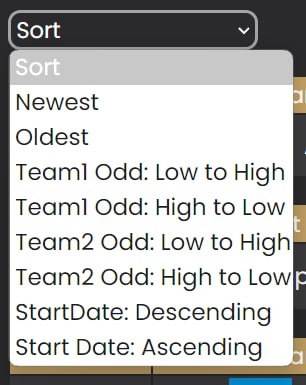
11.att. Limes veida izvēles pogas

1. **Filtra poga**

Sākotnes poga ļauj lietotājam kārtot likmes pēc dažādiem kritērijiem, lai vieglāk atrastu interesējošās likmes (skat 12. att.).

Darbības piemērs:

* **newest:** lietotājs izvēlas "Newest", un likmes tiek sakārtotas, rādot vispirms jaunākās likmes;
* **Team1 Odd:** lietotājs izvēlas "Team1 Odd: Low to High", un likmes tiek sakārtotas pēc pirmās komandas koeficientiem, sākot ar zemākajiem.



12.att. Filtra poga

1. **Spēles logs un uzvarētāja izvēle**

Šis panelis parāda detalizētu informāciju par konkrētu spēli un tās likmēm. (skat 13. att.).

**Game (spēle)**:

* parāda spēles nosaukumu, piemēram, Dota2.

**Start date (sākuma datums)**:

* norāda spēles sākuma datumu un laiku, piemēram, 23/06 12:16.

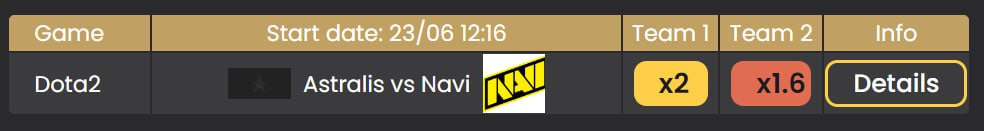
**Team 1 un Team 2**:

* parāda abu komandu nosaukumus un logotipus, piemēram, Astralis vs Navi;
* attēlo katras komandas koeficientus.

**Game (Spēle):** kolonna ar spēles nosaukumu.

**Start date (Sākuma datums):** kolonna ar spēles sākuma datumu un laiku.

**Team 1 un Team 2**: kolonnas ar komandu nosaukumiem, logotipiem un koeficientiem.

Šis panelis sniedz lietotājam ātru pārskatu par spēli un tās likmju iespējām, nodrošinot visu nepieciešamo informāciju likmju veikšanai.

13.att. Spēles logs un uzvarētāja izvēle

1. **Likmes apstiprināšanas logs**

Šis logs ļauj lietotājam izvēlēties summu likmei uz izvēlēto komandu un apstiprināt likmi (skat 14. att.).

**Komandu informācija**:

* Astralis vs Navi: norāda spēles komandas;
* selected team (izvēlētā komanda): parāda izvēlēto komandu, piemēram, Navi;
* odd (koeficients): parāda izvēlētās komandas koeficientu, piemēram, x1.6.

**Konts un likmes informācija**:

* current balance (pašreizējais atlikums): norāda lietotāja konta atlikumu, piemēram, 198.15€;
* bet amount (likmes summa): ievades lauks, kur lietotājs ievada likmes summu, piemēram, 5;
* possible winnings (iespējamie laimesti): parāda iespējamā laimesta summu, kas aprēķināta, balstoties uz ievadīto likmes summu un koeficientu.

**Darbību pogas**:

* place bet (veikt likmi): poga, kas apstiprina likmi ar izvēlēto summu un komandu.



14.att. Likmes apstiprināšanas logs

1. **Modālais logs par aktīvām likmem**

Šis logs ļauj lietotājam pārvaldīt savas likmes, skatot aktīvās likmes un to vēsturi (skat 15. att.).

**Pārslēgšanas pogas**:

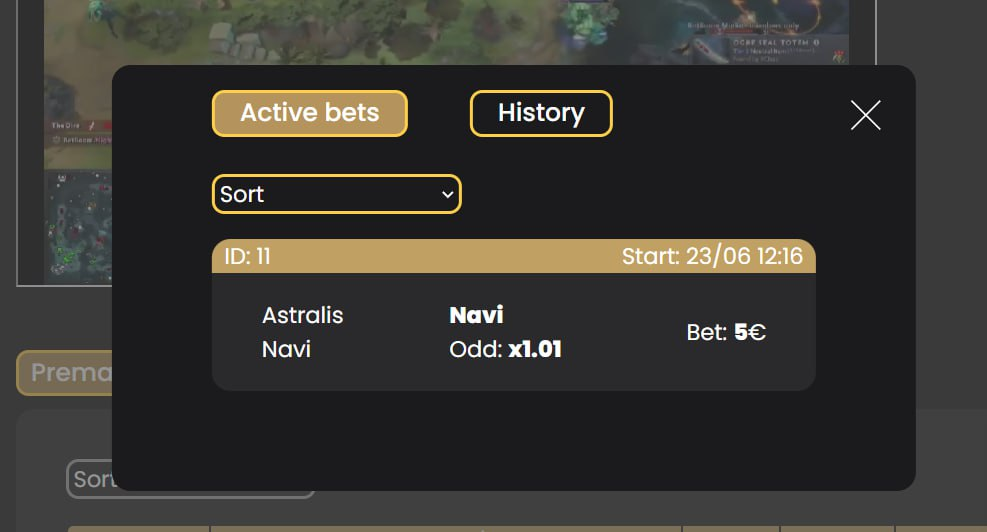
* active bets (aktīvās likmes): parāda lietotāja pašreizējās aktīvās likmes;
* history (vēsture): parāda lietotāja likmju vēsturi.

**Kārtošanas iespējas**.

* sort (kārtošana): nolaižamā izvēlne, kas ļauj lietotājam kārtot likmes pēc dažādiem kritērijiem, piemēram, datuma vai likmes lieluma.

**Likmju saraksts**:

* ID: likmes identifikācijas numurs, piemēram, ID: 11;
* start date (sākuma datums): spēles sākuma datums un laiks, piemēram, 23/06 12:16;
* komandas: abas spēlējošās komandas, piemēram, Astralis vs Navi;
* odd (koeficients): likmes koeficients, piemēram, x1.01;
* bet (likme): likmes summa, piemēram, 5€.



15.att. Modālais logs part aktīvām likmem

1. **Lietotāja informācija**

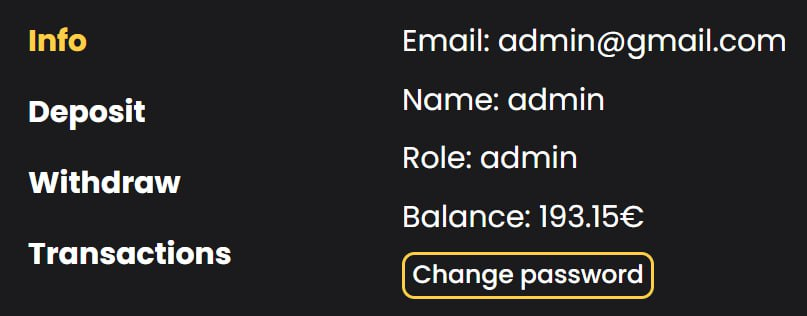
Šī lapa atveras pēc noklikšķināšanas uz pogas "Profils" galvenajā lapā, ļaujot lietotājam redzēt savu informāciju un pārvaldīt kontu (skat 16. att.).

**Lietotāja informācija:**

* rmail: lietotāja e-pasta adrese, piemēram, [admin@gmail.com](mailto:admin@gmail.com);
* name: lietotāja vārds, piemēram, admin;
* role: lietotāja loma, piemēram, admin;
* balance: konta atlikums, piemēram, 193.15€.

**Darbību poga**:

* change password: poga, lai mainītu paroli.



16.att. Lietotāja informācija

1. **Paroles maiņas logs**

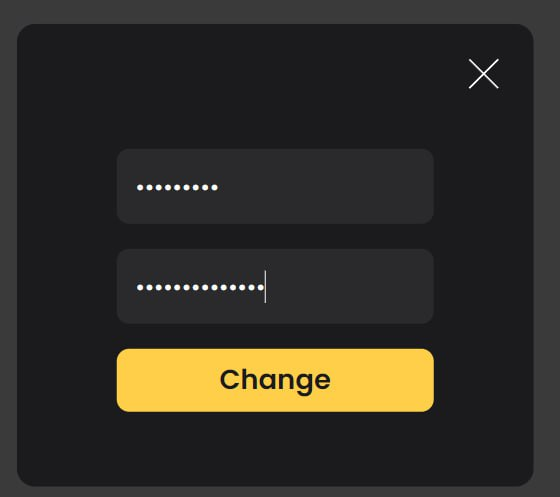
Šis logs atveras pēc noklikšķināšanas uz pogas "Change password" (mainīt paroli) lietotāja profila lapā. Logs ļauj lietotājam ievadīt pašreizējo paroli un jauno paroli, lai veiktu paroles maiņu (skat 17. att.).

**Ievades lauki**:

* pašreizējā parole: pirmajā laukā lietotājam jāievada pašreizējā parole;
* jaunā parole: otrajā laukā lietotājam jāievada jaunā parole.

**Apstiprināšanas poga**:

* Change (mainīt): poga, kas apstiprina paroles maiņu pēc abu paroļu ievadīšanas.



17.att. Paroles maiņas logs

1. **Papildināšanas logs**

Šis logs ļauj lietotājam papildināt savu konta bilanci, izvēloties summu un maksājuma metodi.

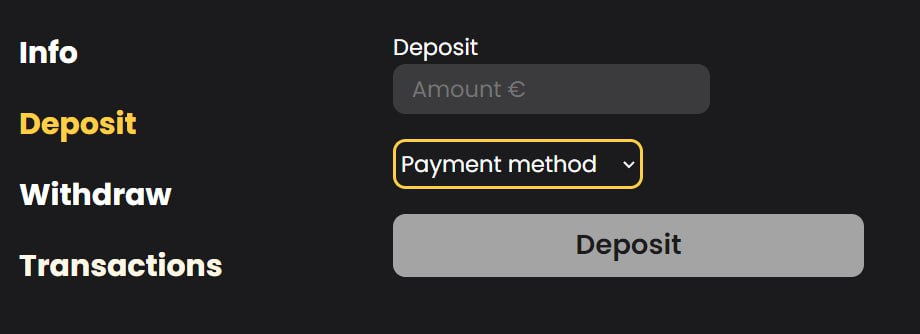
**Sānu izvēlne**:

* info: parāda lietotāja informāciju;
* deposit: izvēlne bilances papildināšanai;
* withdraw: izvēlne naudas izņemšanai;
* transactions: parāda lietotāja transakciju vēsturi.

**Papildināšanas informācija**:

* amount (Summa): ievades lauks, kur lietotājs ievada papildināšanas summu eiro;
* payment method (maksājuma metode): nolaižamā izvēlne, kur lietotājs izvēlas maksājuma metodi.

**Apstiprināšanas poga**:

* deposit (Papildināt): poga, kas apstiprina bilances papildināšanu pēc summas un maksājuma metodes ievadīšanas (skat 18. att.).

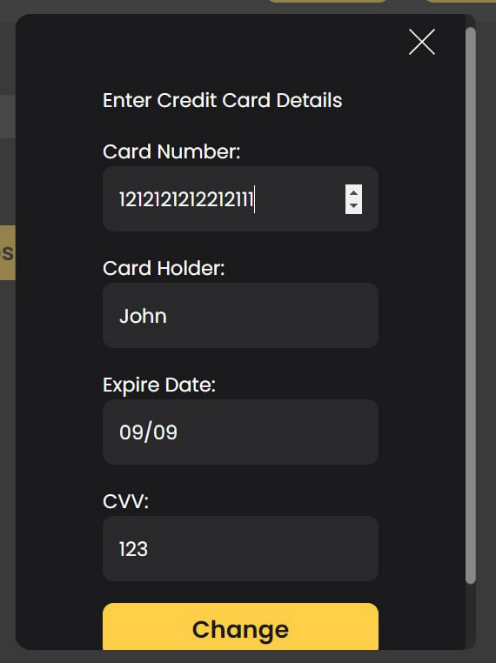
18.att. Papildināšanas logs

1. **Kredītkartes forma**

Šis logs atveras pēc noklikšķināšanas uz pogas "Deposit" (Papildināt) bilances papildināšanas lapā. Logs ļauj lietotājam ievadīt kredītkartes informāciju, lai veiktu iemaksu (skat 19. att.).

**Ievades lauki kredītkartes informācijai**:

* card Number (Kartes numurs): lauks kredītkartes numura ievadīšanai;
* card Holder (Kartes īpašnieks): lauks kartes īpašnieka vārda ievadīšanai;
* expire Date (Derīguma termiņš): lauks kartes derīguma termiņa ievadīšanai (mēnesis/gads);
* CVV: lauks kartes drošības koda (CVV) ievadīšanai.



19.att. Kredītkartes forma

1. **Lietotāja darījumu logs**

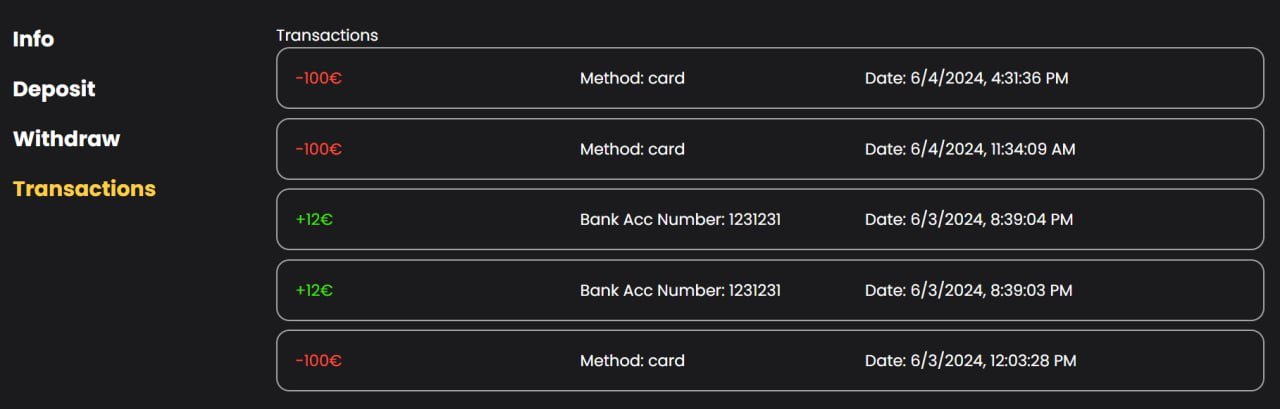
Šis logs parāda visas transakcijas, ko lietotājs ir veicis savā kontā. (skat 20. att.).

**Sānu izvēlne**:

* info: parāda lietotāja informāciju;
* deposit: iespēja iemaksāt naudu kontā;
* withdraw: iespēja izņemt naudu no konta;
* transactions: parāda lietotāja transakciju vēsturi.

**Transakciju saraksts**:

* summas: pozitīvas summas (zaļā krāsā) norāda uz iemaksām. Negatīvas summas (sarkanā krāsā) norāda uz izmaksām;
* maksājuma metode: norāda, kāda metode tika izmantota transakcijai, piemēram, "Method: card" vai "Bank Acc Number";
* datums: norāda transakcijas datumu un laiku.



20.att. Lietotāja darījumu logs

1. **Informācija par spēlēm administratoram**

Šis logs ir redzams tikai lietotājiem ar administratora lomu pēc noklikšķināšanas uz pogas "Admin panel". Tas ļauj pārraudzīt visas likmes vietnē, gan aktīvās, gan noslēgtās (skat 21. att.).

**Filtrēšanas un kārtošanas iespējas**:

* prematch/live: filtrē likmes pēc statusa (pirms spēles vai tiešraidē);
* category: filtrē likmes pēc kategorijas;
* active Games: filtrē pēc spēles statusa (aktīvas spēles);
* sort: nolaižamā izvēlne likmju kārtošanai.

**Likmju saraksts**:

* ID: likmes identifikācijas numurs;
* category: spēles kategorija;
* game: spēles nosaukums, piemēram, Astralis vs Navi;
* team-1 odd: pirmās komandas koeficients;
* team-2 odd: otrās komandas koeficients;
* start date: spēles sākuma datums un laiks;
* finish date: spēles beigu datums un laiks;
* bets open: norāda, vai likmes ir atvērtas;
* action: saites uz detalizētu informāciju par likmi.

**Darbību pogas**:

* awaiting result: filtrē likmes, kurām tiek gaidīti rezultāti;
* create: poga jaunas likmes izveidei.



21.att. Informācija par spēlēm administratoram

1. **Spēļu izveide**

Šis logs ļauj administratoram izveidot jaunu likmi, ievadot spēles un likmju informāciju (skat 22. att.).

**Komandu izvēle un koeficienti**:

* pirmā komanda: nolaižamā izvēlne, kurā administrators izvēlas pirmo komandu, piemēram, Cloud9, un ievada tās koeficientu, piemēram, 2;
* otrā komanda: nolaižamā izvēlne, kurā administrators izvēlas otro komandu, piemēram, Cloud9, un ievada tās koeficientu, piemēram, 1.7.

**Spēles datumi**:

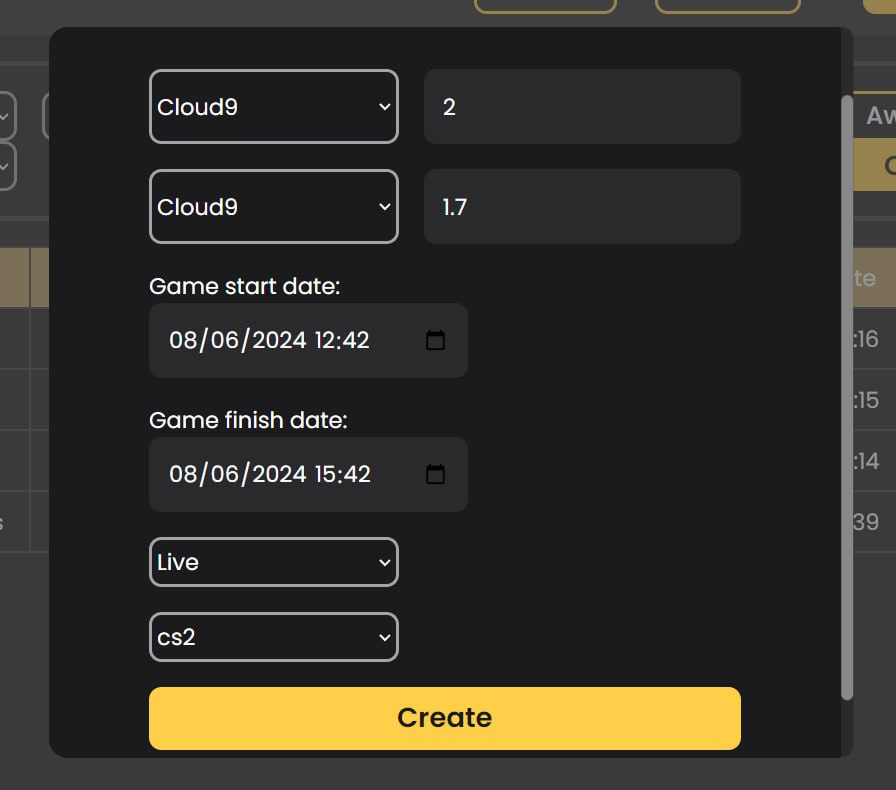
* spēles sākuma datums: datuma un laika ievades lauks, kurā administrators ievada spēles sākuma datumu un laiku, piemēram, 08/06/2024 12:42;
* spēles beigu datums: datuma un laika ievades lauks, kurā administrators ievada spēles beigu datumu un laiku, piemēram, 08/06/2024 15:42.

**Spēles statuss un kategorija**:

* spēles statuss: nolaižamā izvēlne, kurā administrators izvēlas spēles statusu, piemēram, Live (tiešraidē) vai Prematch (pirms spēles);
* spēles kategorija: nolaižamā izvēlne, kurā administrators izvēlas spēles kategoriju, piemēram, cs2.

**Apstiprināšanas poga**:

* create (Izveidot): poga, kas apstiprina jaunas likmes izveidi ar ievadīto informāciju.



22.att. Spēļu izveide

1. **Informācija par kategorijām administratoram**

Šis logs ļauj administratoram pārvaldīt spēļu kategorijas, pievienojot, rediģējot un dzēšot tās. (skat 23. att.).

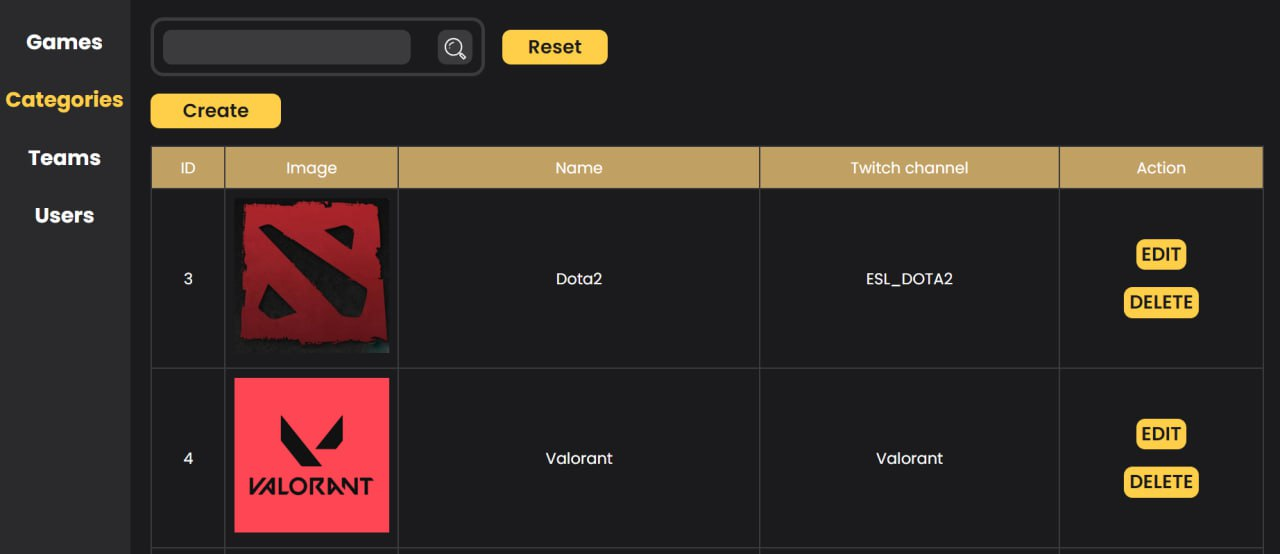
**Darbību pogas**:

* create (izveidot): poga jaunas kategorijas pievienošanai;
* edit (rediģēt): poga kategorijas rediģēšanai;
* delete (dzēst): poga kategorijas dzēšanai.

**Kategoriju saraksts**:

* ID: kategorijas identifikācijas numurs;
* image: kategorijas attēls (ikona);
* name: kategorijas nosaukums, piemēram, Dota2, Valorant;
* twitch channel: saistītais Twitch kanāls, piemēram, ESL\_DOTA2.

**Action**: darbības pogas (edit, delete) katrai kategorijai:

* meklēšanas un atiestatīšanas iespējas;
* meklēšanas lauks: lauks kategoriju meklēšanai pēc nosaukuma.
* reset (Atiestatīt): poga meklēšanas rezultātu atiestatīšanai.

23.att. informācija par kategorijām administratoram

1. **Kategorijas izveide**

Šis logs atveras pēc noklikšķināšanas uz pogas "Create" kategoriju pārvaldības lapā, ļaujot administratoram pievienot jaunu kategoriju (skat 24. att.).

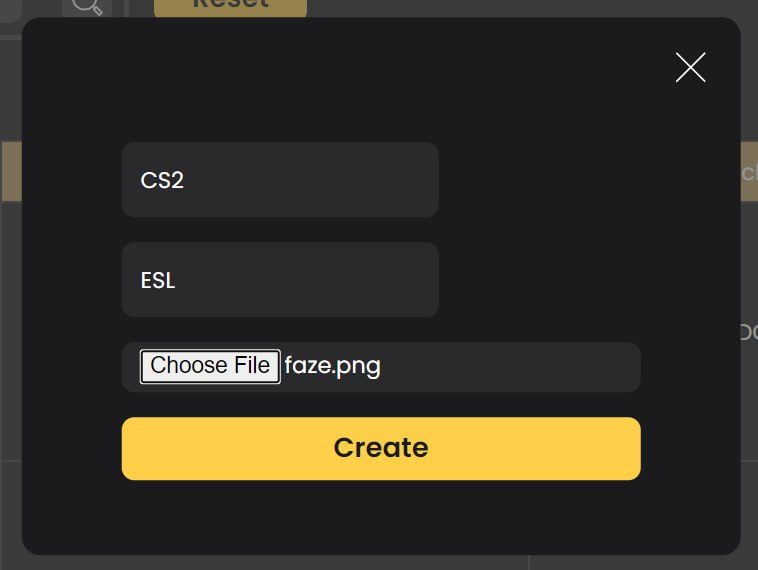
**Ievades lauki kategorijas informācijai**:

* name (nosaukums): ievades lauks, kurā administrators ievada jaunās kategorijas nosaukumu, piemēram, CS2;
* twitch channel (Twitch kanāls): ievades lauks, kurā administrators ievada saistītā Twitch kanāla nosaukumu, piemēram, ESL.

**Attēla izvēle**:

* choose File (izvēlēties failu): poga attēla augšupielādei, kas pārstāv jauno kategoriju, piemēram, faze.png.

**Apstiprināšanas poga**:

* create (izveidot): poga, kas apstiprina jaunās kategorijas izveidi ar ievadīto informāciju.

24.att. kategorijas izveide

1. **Informācija par komandam administratoram**

Šis logs ļauj administratoram pārvaldīt komandas, pievienojot, rediģējot un dzēšot tās (skat 25. att.).

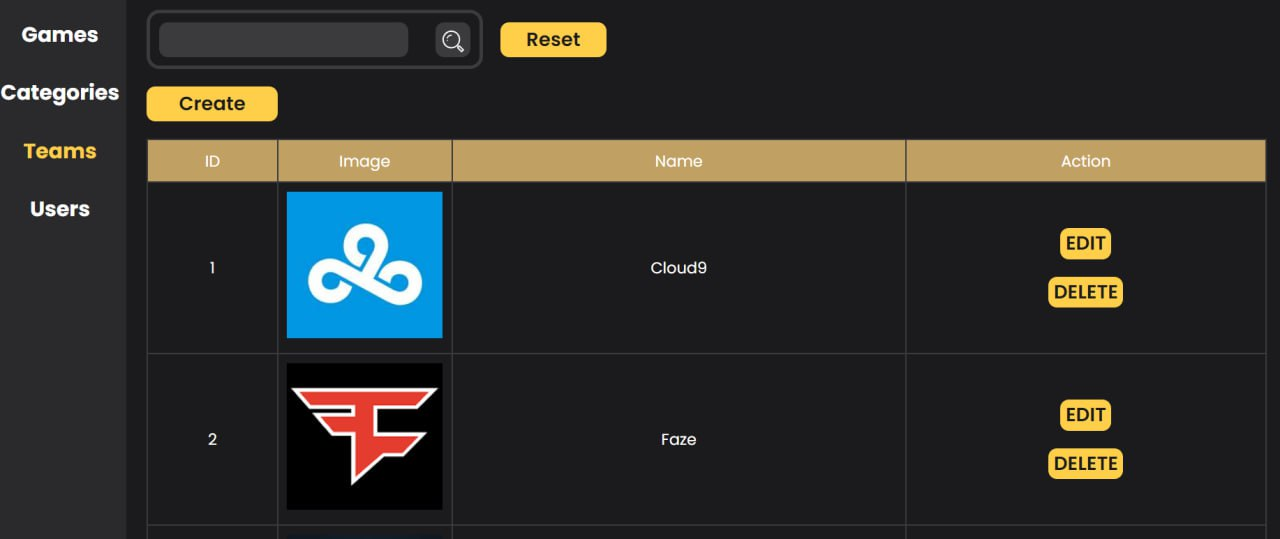
**Darbību pogas**:

* create (izveidot): poga jaunas komandas pievienošanai;
* edit (rediģēt): poga komandas rediģēšanai;
* delete (dzēst): poga komandas dzēšanai.

**Komandu saraksts**:

* ID: komandas identifikācijas numurs;
* image: komandas attēls (logotips);
* name: komandas nosaukums, piem., Cloud9, Faze;
* action: darbības pogas (edit, delete) katrai komandai.

**Meklēšanas un atiestatīšanas iespējas**:

* meklēšanas lauks: lauks komandu meklēšanai pēc nosaukuma;
* reset (atiestatīt): poga meklēšanas rezultātu atiestatīšanai.

25.att. komandas administrators

1. **Komandas veidošana**

Šis logs atveras pēc noklikšķināšanas uz pogas "Create" komandu pārvaldības lapā, ļaujot administratoram pievienot jaunu komandu (skat 26. att.).

**Ievades lauks komandas nosaukumam**:

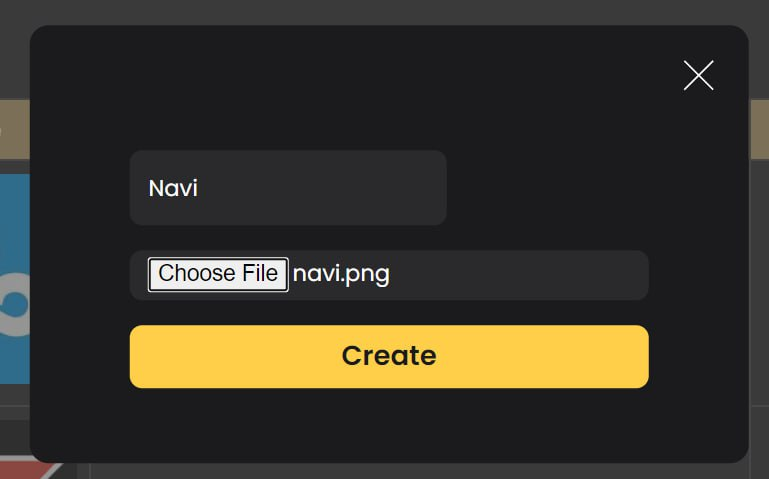
* name (nosaukums): ievades lauks, kurā administrators ievada jaunās komandas nosaukumu.

**Attēla izvēle**:

* choose File (izvēlēties failu): poga komandas logotipa attēla augšupielādei.

**Apstiprināšanas poga**:

* create (izveidot): poga, kas apstiprina jaunās komandas izveidi ar ievadīto informāciju.



26.att. komandas veidošana

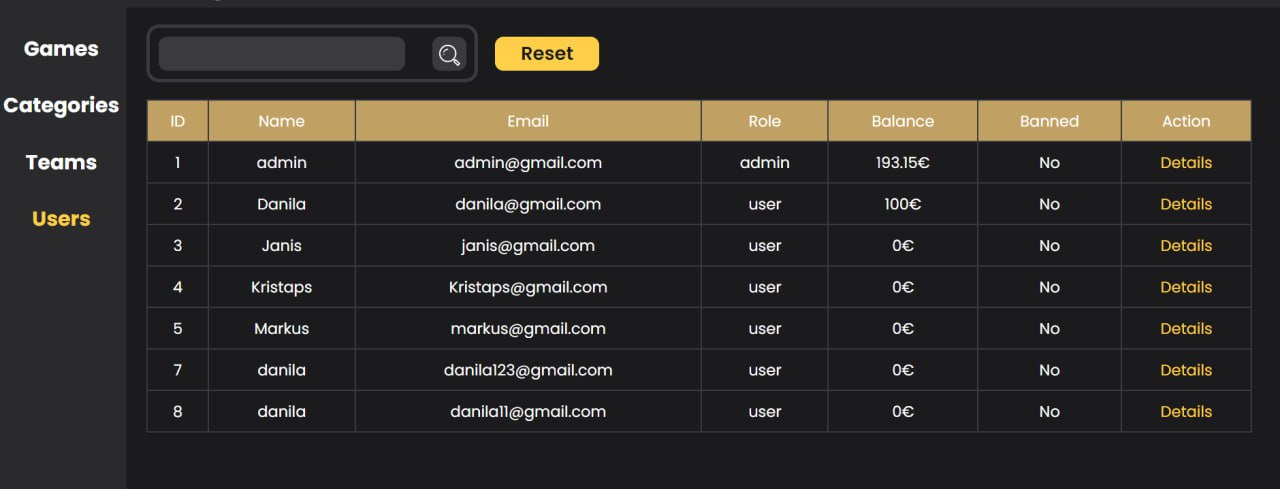
1. **Informācija par lietotājiem**

Šis logs ļauj administratoram pārvaldīt lietotājus, sniedzot detalizētu informāciju par katru lietotāju. (skat 27. att.).

**Lietotāju saraksts**.

* **ID**: lietotāja identifikācijas numurs;
* **name (vārds)**: lietotāja vārds;
* **email**: lietotāja e-pasta adrese;
* **role (loma)**: lietotāja loma, piemēram, admin vai user;
* **balance (atlikums)**: lietotāja konta atlikums;
* **banned (aizliegts)**: norāda, vai lietotājs ir aizliegts, piemēram, No;
* **action (darbība)**: poga "Details" katram lietotājam, lai apskatītu detalizētu informāciju.

**Meklēšanas un atiestatīšanas iespējas**.

* **meklēšanas lauks**: lauks lietotāju meklēšanai pēc vārda vai e-pasta adreses;
* **reset (atiestatīt)**: poga meklēšanas rezultātu atiestatīšanai.

27.att. informācija par lietotājiem

1. **Detalizēta informācija par katru lietotāju**

Šis logs atveras pēc noklikšķināšanas uz pogas "Details" blakus konkrētam lietotājam lietotāju pārvaldības lapā. Logs sniedz detalizētu informāciju par lietotāju un viņa likmēm (skat 28. att.).

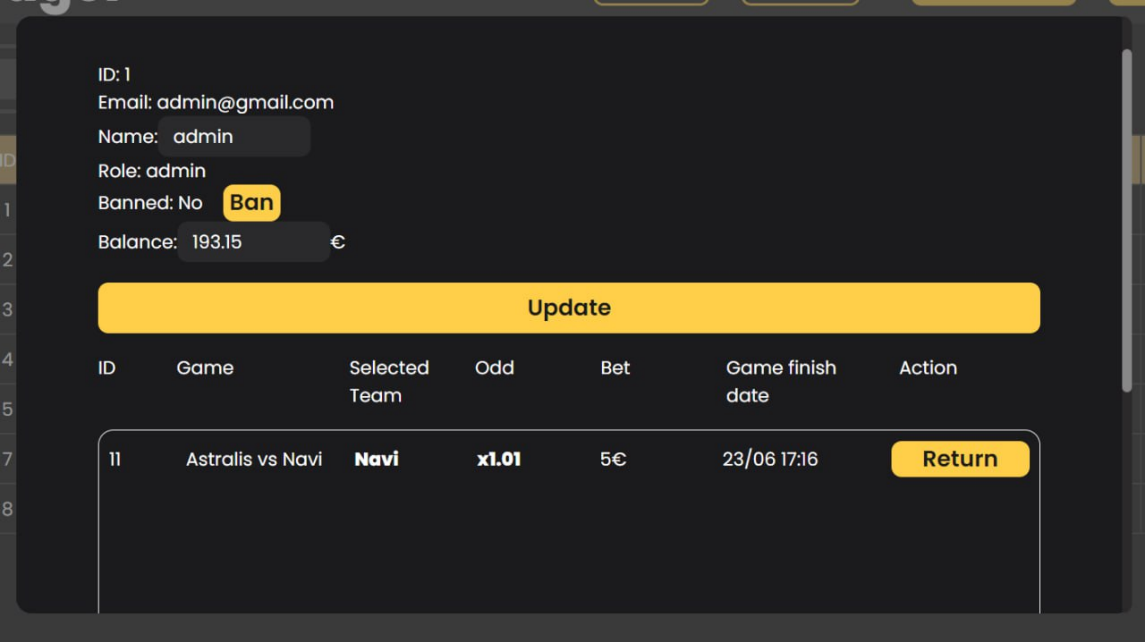
**Lietotāja informācija**:

* ID: lietotāja identifikācijas numurs;
* rmail: lietotāja e-pasta adrese, piemēram, [admin@gmail.com](mailto:admin@gmail.com);
* name (vārds): lietotāja vārds, piemēram, admin;
* role (loma): lietotāja loma, piemēram, admin;
* banned (bloķets): norāda, vai lietotājs ir aizliegts. Pogas "Ban" nospiešana maina lietotāja statusu;
* balance (atlikums): lietotāja konta atlikums, piemēram, 193.15€.

**Darbību poga**:

* update (atjaunināt): poga, kas apstiprina izmaiņas lietotāja informācijā.

**Lietotāja likmju saraksts**:

* ID: likmes identifikācijas numurs;
* game (spēle): spēles nosaukums, piemēram, Astralis vs Navi;
* selected Team (izvēlētā komanda): komanda, uz kuru veikta likme, piemēram, Navi;
* action (darbība): poga "Return" katrai likmei, lai atgrieztu likmi.

28.att. informācija par lietotājiem

## 6.3. Testa piemērs

Testēšanas laikā tika notestētas vairakas prasības no funkcionālajām prasībām (skat 2.3. punktu). Testēšanas parskāts tiek dots 10. tabula.

10.tabula

Testēšanas rezultātu pārskāts

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr.** | **Prasības numurs** | **Prasības nosaukums** | **Ievaddati/situācijas apraksts** | **Sagaidāmais rezultāts** | **Statuss** |
| 1. | 2. | Esoša lietotāja autorizācija | Pareizi dati: lietotāja e-pasts, parole | Tiks pārbaudīta lietotāja esamība un tad lietotājs varēs izmantot vietnes funkcijas | Pareizi |
| 2. | Lietotāja e-pasts = danila.golcovs%gmail.com | Paziņojums par nepareizo formātu | Pareizi |
| 3. | Nepareiza parole | Paziņojums, ka daži dati ir ievadīti nepareizi | Pareizi |
| 4 | Visi dati tiks ievadīti pareizi, bet lietotājs būs bloķēts no vietnes | Ziņojums, ka lietotājs ir bloķēts šajā vietnē | Pareizi |
| 5. | 15. | Lietotāja paroles maiņa | Lietotāja ievadītā parole atbilst visām normām un noteikumiem un ir pareizi dublēts | Ziņojums par veiksmīgu paroles maiņu | Pareizi |
| 6. | Vienā rindā ievadītā parole nesakrīt ar otro ievadīto paroli | Ziņojums, ka parole nesakrīt | Pareizi |
| 7. | Parole sastāv tikai no 4 rakstzīmēm un neatbilst normai | Ziņojums, ka parole neatbilst standartiem un tai jāsastāv no vismaz 8 cipariem | Pareizi |
| 8. | 8. | Likt likmi | Lietotājs izvēlējās komandu, uz kuru viņš liek likmes, un ievadīja summu, kas vienāda ar 10 eiro, ja viņa atlikums ir 100 eiro | Ziņojums, ka piedāvājums bija veiksmīgs | Pareizi |
| 9. | Lietotājs izvēlējās komandu, uz kuru viņš liek likmes, un ievadīja summu, kas vienāda ar 100 eiro, ja viņa atlikums ir 10 eiro | Ziņojums, ka lietotājam nav pietiekami daudz atlikuma | Pareizi |
| 10. | Lietotājs likmes summas rindā ievadīja burtus, nevis ciparus | Tiek parādīts ziņojums, ka rindā var ievadīt tikai ciparus | Pareizi |

Testēšana bija veiksmīga, tika izmantoti daudzi ievades dati, un visi rezultāti bija pozitīvi

**NOBEIGUMS**

Kvalifikācijas darba izstrādes laikā tika izveidota vietne, kurā lietotāji var veikt likmes uz kiberporta mačiem. Projekts tika realizēts angļu valodā, un tika panākts augsts funkcionālais līmenis. Vietne ļauj veikt likmes uz dažādiem kiberporta mačiem, piedāvājot vienkāršu un lietotājam draudzīgu saskarni. Izstrādes gaitā tika pielietotas modernas tīmekļa tehnoloģijas, lai nodrošinātu stabilu un ātru vietnes darbību.

Pašlaik šī vietne ir piemērota salīdzinoši neliela likmju skaita apkalpošanai. Nākotnē vietni var attīstīt, integrējot trešo pušu API un datu bāzes, lai automātiski iegūtu informāciju par spēlēm un to statistiku. Tas palīdzēs palielināt vietnes veiktspēju un efektivitāti. Turklāt vietnei var pievienot plašāku likmju variāciju klāstu, tādējādi piesaistot plašāku lietotāju loku un nodrošinot lielāku funkcionalitāti. Vietne ir ar potenciālu attīstīties par lielāku un sarežģītāku platformu, kas spēj apkalpot lielāku lietotāju skaitu un piedāvāt daudzveidīgākas likmju iespējas.

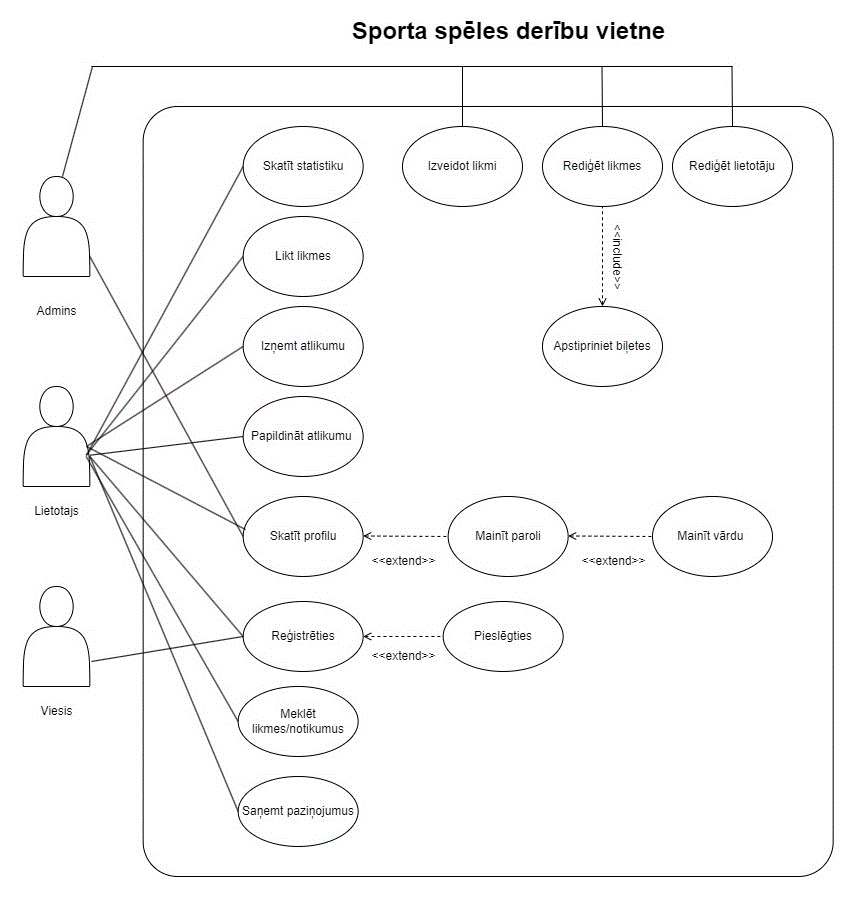
**INFORMĀCIJAS AVOTI**

1. MDN Web Docs - Detalizēti un uzticami resursi par HTML, CSS, JavaScript un citām tīmekļa tehnoloģijām. <https://developer.mozilla.org/en-US/> (Resurss apskatīts 11.09.2023.).
2. Stack Overflow – formus par dažādām programmēšanas problēmām. <https://stackoverflow.com/> (Resurss apskatīts 15.09.2023.).
3. Node.js oficiālā dokumentācija - Visaptveroša informācija un piemēri, kas palīdz ar Node.js. <https://nodejs.org/en> (Resurss apskatīts 22.09.2023.).
4. React oficiālā dokumentācija - Oficiālie resursi un apmācības, lai apgūtu un izmantotu React bibliotēku. <https://react.dev/> (Resurss apskatīts 22.09.2023.).
5. Docker oficiālā dokumentācija - Detalizēti norādījumi un apmācības par Docker izmantošanu konteinerizācijā. <https://docs.docker.com/> (Resurss apskatīts 22.09.2023.).
6. CSS-Tricks - Raksti un pamācības par CSS, SASS un citiem front-end izstrādes rīkiem. <https://css-tricks.com/> (Resurss apskatīts 07.02.2024.).
7. GitHub (github.com) - Platforma koda mitināšanai un versiju kontrolei, kā arī vieta, kur atrast atvērtā pirmkoda projektus un piemērus. <https://github.com/> (Resurss apskatīts 11.02.2024.).
8. FreeCodeCamp - raksti par dažādām programmēšanas valodām un tehnoloģijām. <https://www.freecodecamp.org/>(Resurss apskatīts 11.02.2024.).
9. Express.js Dokumentācija Express.js oficiālā dokumentācija: <https://expressjs.com/en/guide/> (Resurss apskatīts 22.09.2023.).

# PIELIKUMI

1. pielikums

**Lietojumgadījuma diagramma**



1. pielikums

**Programmas pirmkods**

import {formatGameDate} from "../../util/format";

const ActiveBets = ({ bets }) => {

    return (

        <div className="MyBets-Content" >

            {bets && bets.length > 0 &&

                bets.map(bet =>

                    <div

                        className="MyBets-Content-Bet"

                        style={bet.is\_returned ? { backgroundColor: "#5e5d5d" } : {}}

                        key={bet.id}

                    >

                        <div className="MyBets-Content-Bet-Header">

                            <p>ID: {bet.id}</p>

                            <p>Start: {formatGameDate(bet.game.game\_start\_date)}</p>

                        </div>

                        <div className="MyBets-Content-Bet-Content">

                            <div className="MyBets-Content-Bet-Content-Game">

                                <p>{bet.game.teamGame1.team.name}</p>

                                <p>{bet.game.teamGame2.team.name}</p>

                            </div>

                            <div className="MyBets-Content-Bet-Content-Bet">

                                <p className="fw-800">

                                    {bet.selectedTeam === 'team1' ? bet.game.teamGame1.team.name : bet.game.teamGame2.team.name}

                                </p>

                                <p>Odd: <span

                                    className="fw-800">x{bet.selectedTeam === 'team1' ? bet.game.teamGame1.coefficient : bet.game.teamGame2.coefficient}</span>

                                </p>

                            </div>

                            <div className="MyBets-Content-Bet-Content-Amount" style={bet.is\_returned ? {display: "flex", flexDirection: "column"} : {}}>

                                <p>Bet: <span className="fw-800">{bet.amount}</span>€</p>

                                {bet.is\_returned && <p className="fw-800" style={{color: "yellow"}}>RETURNED</p>}

                            </div>

                        </div>

                    </div>

                )

            }

        </div>

    );

}

export default ActiveBets;

// import AdminGamesCreate from './AdminGamesCreate';

import { useContext, useEffect, useState } from 'react';

import './AdminCategories.scss';

import axios from 'axios';

import Modal from "../Modal/Modal";

import AdminCategoriesCreate from './AdminCategoriesCreate';

import { AuthContext } from '../../App';

import { handleDeleteImage } from '../../util/filesApi';

const AdminCategories = () => {

    const [categories, setCategories] = useState([]);

    const [filteredCategories, setFilteredCategories] = useState([]);

    const [isCreateModalOpen, setIsCreateModalOpen] = useState(false);

    const { authData } = useContext(AuthContext);

    const [categoryToEdit, setCategoryToEdit] = useState(null);

    const [search, setSearch] = useState('');

    const openCreate = () => {

        setIsCreateModalOpen(true);

    }

    const closeCreate = () => {

        updateTeams();

        setIsCreateModalOpen(false);

        setCategoryToEdit(null);

    }

    useEffect(() => {

        updateTeams();

    }, []);

    const updateTeams = () => {

        axios.get('http://localhost:8080/categories/')

            .then(r => {

                console.log(r.data)

                setCategories(r.data);

                setFilteredCategories(r.data);

            }).catch(err => {

                console.log(err);

            });

    };

    const handleDelete = (category) => {

        if (category?.img) {

            handleDeleteImage(category.img);

        }

        axios.post('http://localhost:8080/categories/delete', {

            id: category.id,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            updateTeams();

        }).catch(err => {

            console.log(err);

        });

    };

    const handleEdit = (team) => {

        setCategoryToEdit(team);

        setIsCreateModalOpen(true);

    }

    const handleSearch = () => {

        if (search === '') {

            setFilteredCategories(categories);

            return;

        }

        const filtered = categories.filter(category => {

            return category.name.toLowerCase().includes(search.toLowerCase())

        })

        setFilteredCategories(filtered);

    }

    const handleResetSearch = () => {

        setSearch('');

        setFilteredCategories(categories);

    }

    return (

        <div className="AdminTeams">

            <Modal isOpen={isCreateModalOpen} close={closeCreate} content={<AdminCategoriesCreate close={closeCreate} category={categoryToEdit} />}/>

            <div style={{display: 'flex', flexDirection: "row", alignItems: "center"}}>

                <div className="AdminTeams-Header">

                    <input

                        type="text"

                        value={search}

                        onChange={(e) => setSearch(e.target.value)}

                        onKeyDown={(e) => {

                            if (e.key === 'Enter') {

                                handleSearch();

                            }

                        }}

                    />

                    <button onClick={handleSearch} className="AdminTeams-Search" />

                </div>

                <button onClick={handleResetSearch} className='Button' style={{height: "40px", marginBottom: "20px", marginLeft: "20px", paddingInline: "30px"}}>Reset</button>

            </div>

            <div className="AdminTeams-Create">

                <button className="Button AdminTeams-Create-button" onClick={openCreate}>Create</button>

            </div>

            <table className="AdminTeams-Table">

                <thead className="AdminTeams-Table-Header">

                    <tr>

                        <td>ID</td>

                        <td>Image</td>

                        <td>Name</td>

                        <td>Twitch channel</td>

                        <td>Action</td>

                    </tr>

                </thead>

                <tbody className="AdminTeams-Table-Content">

                    {filteredCategories.length > 0 && filteredCategories.map(category =>

                        <tr>

                            <td>{category.id}</td>

                            <td style={{width: "200px"}}><img src={category.img} alt="" /></td>

                            <td>{category.name}</td>

                            <td>

                                {category.channel\_stream || 'null'}

                            </td>

                            {/\* <td className="AdminTeams-Table-Content-Action" onClick={() => handleDelete(category.id)}>DELETE</td> \*/}

                            <td>

                                <div style={{marginBottom: "20px"}}>

                                    <button className='Button' onClick={() => handleEdit(category)}>EDIT</button>

                                </div>

                                <div>

                                    <button className='Button' onClick={() => handleDelete(category)}>DELETE</button>

                                </div>

                            </td>

                        </tr>

                    )}

                </tbody>

            </table>

        </div>

    );

}

export default AdminCategories;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

export const AdminCategoriesCreate = ({ close, category }) => {

    const [form, setForm] = useState({

        name: '',

        channel\_stream: '',

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    const [img, setImg] = useState(null);

    useEffect(() => {

        setForm({

            name: category?.name,

            channel\_stream: category?.channel\_stream,

        });

        setImg(null);

    }, [category])

    const uploadImage = (e) => {

        const file = e.target.files[0];

        setImg(file);

    }

    const handleSaveImage = async () => {

        const formData = new FormData();

        formData.append('image', img);

        try {

            const response = await axios.post('http://localhost:8080/upload', formData, {

                headers: {

                    'Content-Type': 'multipart/form-data',

                },

            });

            return response.data;

        } catch (error) {

            console.error('Error:', error);

        }

    }

    const handleDeleteImage = async (path) => {

        try {

            await axios.post('http://localhost:8080/deleteImg', {

                path: path,

            });

        } catch (error) {

            console.error('Error:', error);

        }

    }

    useEffect(() => {

        if(form.name) {

            setIsFormReady(true);

        } else {

            setIsFormReady(false);

        }

    }, [form]);

    const changeName = (e) => {

        const name = e.target.value;

        setForm({

            ...form,

            name: name,

        });

    }

    const changeChannel = (e) => {

        const channel\_stream = e.target.value;

        setForm({

            ...form,

            channel\_stream: channel\_stream,

        });

    }

    const handleCreate = async () => {

        const path = await handleSaveImage();

        axios.post('http://localhost:8080/categories', {

            name: form.name,

            channel\_stream: form.channel\_stream,

            image: path,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                console.log(r.data);

                setError('');

                close();

            }).catch(err => {

                if (err && err.response) {

                    setError(err.response.data.error);

                }

            });

    }

    const handleEdit = async () => {

        let newImg;

        if (category?.img && img) {

            handleDeleteImage(category.img);

        }

        if (img) {

            newImg = await handleSaveImage(img);

            console.log(newImg)

        }

        axios.post('http://localhost:8080/categories/edit', {

            id: category.id,

            name: form.name,

            channel\_stream: form.channel\_stream,

            image: newImg,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                console.log(r.data);

                setImg(null);

                setForm({

                    name: '',

                    channel\_stream: '',

                });

                setError('');

                close();

            }).catch(err => {

                if (err && err.response) {

                    setError(err.response.data.error);

                }

            });

    }

    const handleSubmit = async () => {

        if (category?.name) {

            handleEdit();

        } else {

            handleCreate();

        }

    }

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <input

                    type="text"

                    placeholder="Name"

                    value={form.name}

                    onChange={changeName}

                />

            </div>

            <div className="Modal-input">

                <input

                    type="text"

                    placeholder="Twitch channel"

                    value={form.channel\_stream}

                    onChange={changeChannel}

                />

            </div>

            <div style={{marginBottom: "20px"}}>

                <input type="file" name="avatar" accept="image/\*" onChange={uploadImage} />

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                {category?.name ? 'Edit' : 'Create'}

            </button>

        </div>

    );

}

export default AdminCategoriesCreate;

import './AdminGames.scss';

import React, {useEffect, useState} from 'react';

import axios from "axios";

import Modal from "../Modal/Modal";

import {formatGameDate} from "../../util/format";

import AdminGamesCreate from './AdminGamesCreate';

import AdminGamesDetails from './AdminGamesDetails';

import AdminGamesAwaitingResult from './AdminGamesAwaitingResult';

const AdminGames = () => {

    const [selectedType, setSelectedType] = useState('');

    const [selectedCategory, setSelectedCategory] = useState('all');

    const [games, setGames] = useState([]);

    const [categories, setCategories] = useState([]);

    const [isCreateModalOpen, setIsCreateModalOpen] = useState(false);

    const [isDetailsModalOpen, setIsDetailsModalOpen] = useState(false);

    const [isAwaitingResultOpen, setIsAwaitingResultOpen] = useState(false);

    const [gamesStatus, setGamesStatus] = useState("1");

    const [selectedGame, setSelectedGame] = useState(null);

    const [sort, setSort] = useState(null);

    const openCreate = () => {

        setIsCreateModalOpen(true);

    }

    const closeCreate = () => {

        fetchGames();

        setIsCreateModalOpen(false);

    }

    const openDetails = (game) => {

        setSelectedGame(game);

        setIsDetailsModalOpen(true);

    }

    const closeDetails = () => {

        fetchGames();

        setIsDetailsModalOpen(false);

    }

    const openAwaitingResults = () => {

        setIsAwaitingResultOpen(true);

    }

    const closeAwaitingResults = () => {

        setIsAwaitingResultOpen(false);

    }

    useEffect(() => {

        fetchCategories();

    }, []);

    useEffect(() => {

        fetchGames();

    }, [selectedCategory, selectedType, gamesStatus, sort]);

    const fetchGames = () => {

        if(selectedType) {

            const s = sort?.split('\_') || null;

            const sortBy = s && s[0];

            const order = s && s[1];

            axios.get(`http://localhost:8080/game?category=${selectedCategory}&type=${selectedType}&is\_active=${gamesStatus}&sort=${sortBy}&order=${order}`)

            .then((r) => {

                setGames(r.data);

                console.log(r.data);

            }).catch((err) => {

            console.log(err);

        })

        }

    };

    const fetchCategories = () => {

        axios.get('http://localhost:8080/categories')

            .then(r => {

                setCategories(r.data);

            }).catch(err => {

            console.log(err);

        });

    }

    const handleCategoryChange = (e) => {

        setSelectedCategory(e.target.value);

    }

    const handleGameTypeChange = (e) => {

        setSelectedType(e.target.value);

    }

    const handleGameStatusChange = (e) => {

        setGamesStatus(e.target.value);

    }

    const handleSortChange = (e) => {

        setSort(e.target.value)

    }

    return (

        <div className="AdminGames">

            <Modal isOpen={isCreateModalOpen} close={closeCreate} content={<AdminGamesCreate close={closeCreate} />}/>

            <Modal isOpen={isDetailsModalOpen} close={closeDetails} content={<AdminGamesDetails close={closeDetails} game={selectedGame} />}/>

            <Modal isOpen={isAwaitingResultOpen} close={closeAwaitingResults} content={<AdminGamesAwaitingResult close={closeAwaitingResults} />}/>

            <div className="AdminGames-Header">

                <div className="AdminGames-Header-Left">

                    <select className="Select\_secondary" onChange={handleGameTypeChange}>

                        <option value="" disabled selected>Game Type</option>

                        <option value="manual">Live</option>

                        <option value="auto">Prematch</option>

                    </select>

                    <select className="Select\_secondary" onChange={handleCategoryChange}>

                        <option value="" disabled selected>Category</option>

                        {categories.length > 0 && categories.map(category =>

                            <option value={category.name}>{category.name}</option>

                        )}

                    </select>

                    <select className="Select\_secondary" onChange={handleGameStatusChange}>

                        <option value="1" selected>Active Games</option>

                        <option value="0">History</option>

                    </select>

                    <select className='Select\_secondary' style={{width: "200px"}} value={sort} onChange={handleSortChange}>

                        <option value={null} disabled selected>Sort</option>

                        <option value="id\_desc">Newest</option>

                        <option value="id\_asc">Oldest</option>

                        <option value="team1\_asc">Team1 Odd: Low to High</option>

                        <option value="team1\_desc">Team1 Odd: High to Low</option>

                        <option value="team2\_asc">Team2 Odd: Low to High</option>

                        <option value="team2\_desc">Team2 Odd: High to Low</option>

                        <option value="startDate\_desc">StartDate: Descending</option>

                        <option value="startDate\_asc">Start Date: Ascending</option>

                        <option value="finishDate\_desc">Finish Date: Descending</option>

                        <option value="finishDate\_asc">Finish Date: Ascending</option>

                    </select>

                </div>

                <div className="AdminGames-Header-Right">

                    <button className="Button\_secondary" style={{paddingInline: "20px", width: "auto", marginRight: "40px"}} onClick={openAwaitingResults}>Awaiting result</button>

                    <button className="Button" onClick={openCreate}>Create</button>

                </div>

            </div>

            <div className="AdminGames-Content">

            <table className="AdminTeams-Table">

                <thead className="AdminTeams-Table-Header">

                    <tr>

                        <td>ID</td>

                        <td>Category</td>

                        <td>Game</td>

                        <td>Team-1 odd</td>

                        <td>Team-2 odd</td>

                        <td>Start date</td>

                        <td>Finish date</td>

                        {gamesStatus === "1" && <td>Bets open</td>}

                        <td>Action</td>

                    </tr>

                </thead>

                {!selectedType &&

                    <h1 style={{ position: "absolute", textAlign: "center", marginLeft: "50px", marginTop: "20px", fontSize: "28px"}}>Please select Game Type</h1>

                }

                <tbody className="AdminTeams-Table-Content">

                    {games.length > 0 && games.map(game =>

                        <tr>

                            <td>{game.id}</td>

                            <td>{game.category.name}</td>

                            <td>{game.teamGame1.team.name} vs {game.teamGame2.team.name}</td>

                            <td className="fw-800">x{game.teamGame1.coefficient}</td>

                            <td className="fw-800">x{game.teamGame2.coefficient}</td>

                            <td>{formatGameDate(game.game\_start\_date)}</td>

                            <td>{formatGameDate(game.game\_finish\_date)}</td>

                            {gamesStatus === "1" && <td>{game.is\_bets\_open ? 'Yes' : 'No'}</td>}

                            <td className="AdminGames-Table-Content-Details" onClick={() => openDetails(game)}>Details</td>

                        </tr>

                    )}

                </tbody>

            </table>

            </div>

        </div>

    );

};

export default AdminGames;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

import './AdminGamesDetails.scss';

import { formatGameDate } from '../../util/format';

export const AdminGamesAwaitingResult = ({ close }) => {

    const [error, setError] = useState('');

    const { authData } = useContext(AuthContext);

    const [games, setGames] = useState([]);

    useEffect(() => {

        fetchGames();

    }, [])

    const fetchGames = () => {

        axios.get('http://localhost:8080/game/awaitingresult')

        .then(r => {

            setGames(r.data);

            console.log(r.data);

        });

    }

    const handleSubmit = (id, team) => {

        axios.post('http://localhost:8080/game/setteamwon', {

            id: id,

            teamWon: team

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            setError('');

            // close();

            fetchGames();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div className='AwaitingResult'>

            <div className="Modal-Error">

                {error}

            </div>

            <div className='AwaitingResult-Games-Header'>

                    <p>ID</p>

                    <p>Category</p>

                    <p>Game</p>

                    <p>Start date</p>

                    <p>Finish Date</p>

                    <p>Team Won</p>

                </div>

            <div className='AwaitingResult-Games'>

                {games && games.length > 0 && games.map(game =>

                    <div key={game.id} className='AwaitingResult-Games-Item'>

                        <div className='AwaitingResult-Games-Item-Info'>

                            <p>{game.id}</p>

                            <p>{game.category.name}</p>

                            <p>{game.teamGame1.team.name} vs {game.teamGame2.team.name}</p>

                            <p>{formatGameDate(game.game\_start\_date)}</p>

                            <p>{formatGameDate(game.game\_finish\_date)}</p>

                        </div>

                        <div style={{marginLeft: '10px', display: "flex", flexDirection:"row", height: '40px'}}>

                            <button onClick={() => handleSubmit(game.id, 'team1')} className='Button' style={{width: "150px"}}>Team 1</button>

                            <button onClick={() => handleSubmit(game.id, 'team2')} className='Button' style={{width: "150px"}}>Team 2</button>

                        </div>

                    </div>

                )}

            </div>

        </div>

    );

}

export default AdminGamesAwaitingResult;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

export const AdminGamesCreate = ({ close }) => {

    const [form, setForm] = useState({

        team\_name\_1: '',

        coefficient\_1: 1,

        team\_name\_2: '',

        coefficient\_2: 1,

        game\_type: '',

        game\_start\_date: '',

        game\_finish\_date: '',

        category\_name: '',

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    const [categories, setCategories] = useState([]);

    const [teams, setTeams] = useState([]);

    useEffect(() => {

        fetchCategories();

        fetchTeams();

    }, []);

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        if(form.coefficient\_1 <= 1 || form.coefficient\_2 <= 1) {

            setError('Coefficient should be greater than 1');

            setIsFormReady(false);

            return;

        } else {

            setError('');

        }

        setIsFormReady(isFilled);

    }, [form]);

    useEffect(() => {

        console.log(isFormReady);

    }, [isFormReady]);

    const fetchTeams = () => {

        axios.get('http://localhost:8080/team/')

            .then(r => {

                setTeams(r.data);

                console.log(r.data);

            }).catch(err => {

                console.log(err);

        });

    };

    const fetchCategories = () => {

        axios.get('http://localhost:8080/categories')

            .then(r => {

                setCategories(r.data);

            }).catch(err => {

            console.log(err);

        });

    };

    const handleSubmit = () => {

        console.log(form);

        axios.post('http://localhost:8080/game', {

            team\_name\_1: form.team\_name\_1,

            coefficient\_1: parseFloat(form.coefficient\_1),

            team\_name\_2: form.team\_name\_2,

            coefficient\_2: parseFloat(form.coefficient\_2),

            game\_type: form.game\_type,

            game\_start\_date: form.game\_start\_date,

            game\_finish\_date: form.game\_finish\_date,

            category\_name: form.category\_name,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <select className="Select\_secondary" style={{ width: "200px", height: "60px"}} onChange={(e) => changeField(e, 'team\_name\_1')}>

                    <option value="" disabled selected>Team 1</option>

                    {teams.length > 0 && teams.map(team =>

                        <option value={team.name}>{team.name}</option>

                    )}

                </select>

                <input

                    style={{ marginLeft: "20px"}}

                    type="number"

                    placeholder="Odd 1"

                    value={form.coefficient\_1}

                    onChange={(e) => changeField(e, 'coefficient\_1')}

                />

            </div>

            <div className="Modal-input">

            <select className="Select\_secondary" style={{ width: "200px", height: "60px"}} onChange={(e) => changeField(e, 'team\_name\_2')}>

                    <option value="" disabled selected>Team 2</option>

                    {teams.length > 0 && teams.map(team =>

                        <option value={team.name}>{team.name}</option>

                    )}

                </select>

                <input

                    style={{ marginLeft: "20px"}}

                    type="number"

                    placeholder="Odd 2"

                    value={form.coefficient\_2}

                    onChange={(e) => changeField(e, 'coefficient\_2')}

                />

            </div>

            Game start date:

            <div className="Modal-input">

                <input

                    type="datetime-local"

                    placeholder="Game start date"

                    value={form.game\_start\_date}

                    onChange={(e) => changeField(e, 'game\_start\_date')}

                />

            </div>

            Game finish date:

            <div className="Modal-input">

                <input

                    type="datetime-local"

                    placeholder="Game finish date"

                    value={form.game\_finish\_date}

                    onChange={(e) => changeField(e, 'game\_finish\_date')}

                />

            </div>

            <div className="Modal-input">

                <select className="Select\_secondary" style={{ width: "200px", height: "40px"}} onChange={(e) => changeField(e, 'game\_type')}>

                    <option value="" disabled selected>Game Type</option>

                    <option value="manual">Live</option>

                    <option value="auto">Prematch</option>

                </select>

            </div>

            <div className="Modal-input">

                <select className="Select\_secondary" style={{ width: "200px", height: "40px"}} onChange={(e) => changeField(e, 'category\_name')}>

                    <option value="" disabled selected>Category</option>

                    {categories.length > 0 && categories.map(category =>

                        <option value={category.name}>{category.name}</option>

                    )}

                </select>

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                Create

            </button>

        </div>

    );

}

export default AdminGamesCreate;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

import { convertDate, format2Num, formatGameDate } from '../../util/format';

import './AdminGamesDetails.scss';

export const AdminGamesDetails = ({ close, game }) => {

    const [form, setForm] = useState({

        coefficient\_1: game.teamGame1.coefficient,

        coefficient\_2: game.teamGame2.coefficient,

        game\_start\_date: formatDateForInput(game.game\_start\_date),

        game\_finish\_date: formatDateForInput(game.game\_finish\_date),

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    const [gameInfo, setGameInfo] = useState({});

    function formatDateForInput(dateString) {

        const date = new Date(dateString);

        const year = date.getFullYear();

        let month = (date.getMonth() + 1).toString().padStart(2, '0');

        let day = date.getDate().toString().padStart(2, '0');

        let hours = date.getHours().toString().padStart(2, '0');

        let minutes = date.getMinutes().toString().padStart(2, '0');

        return `${year}-${month}-${day}T${hours}:${minutes}`;

    }

    useEffect(() => {

        fetchGame();

    }, []);

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        if(form.coefficient\_1 <= 1 || form.coefficient\_2 <= 1) {

            setError('Coefficient should be greater than 1');

            setIsFormReady(false);

            return;

        } else {

            setError('');

        }

        setIsFormReady(isFilled);

    }, [form]);

    useEffect(() => {

        console.log(isFormReady);

    }, [isFormReady]);

    const fetchGame = () => {

        axios.get('http://localhost:8080/game/info?id=' + game.id)

            .then(r => {

                setGameInfo(r.data);

                console.log(r.data);

                setForm({

                    ...form,

                    coefficient\_1: r.data.teamGame1.coefficient,

                    coefficient\_2: r.data.teamGame2.coefficient,

                })

            }).catch(err => {

            console.log(err);

        });

    };

    const handleSubmit = () => {

        console.log(form);

        axios.post('http://localhost:8080/game/update', {

            id: gameInfo.id,

            coefficient\_1: parseFloat(form.coefficient\_1),

            coefficient\_2: parseFloat(form.coefficient\_2),

            game\_start\_date: form.game\_start\_date,

            // game\_start\_date: convertDate(form.game\_start\_date),

            game\_finish\_date: form.game\_finish\_date,

            // game\_finish\_date: convertDate(form.game\_finish\_date),

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    const handleClickBets = (e) => {

        e.preventDefault();

        e.stopPropagation();

        axios.post('http://localhost:8080/game/changebetsstatus', {

            id: gameInfo.id,

            is\_bets\_open: !gameInfo.is\_bets\_open,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            setError('');

            fetchGame();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    const handleReturnAll = () => {

        axios.post('http://localhost:8080/bet/returnbygame', {

            id: gameInfo.id,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            setError('');

            fetchGame();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div className="AdminGamesDetails">

            <div className="AdminGamesDetails-Left">

                <div className="Modal-Error">

                    {error}

                </div>

                <div className="Modal-input" style={{display:"flex", flexDirection: "row", alignItems: "center"}}>

                    <p>{game.teamGame1.team.name}:&nbsp;</p>

                    {game.type === 'manual' && game.is\_active ? (

                        <input

                            style={{ marginLeft: "20px"}}

                            type="number"

                            placeholder="Odd 1"

                            value={form.coefficient\_1}

                            onChange={(e) => changeField(e, 'coefficient\_1')}

                        />

                    ) : (

                        <p> <span className='fw-800'> x{game.teamGame1.coefficient}</span></p>

                    )}

                </div>

                <div className="Modal-input" style={{display:"flex", flexDirection: "row", alignItems: "center"}}>

                    <p>{game.teamGame2.team.name}:&nbsp;</p>

                    {game.type === 'manual' && game.is\_active ? (

                        <input

                            style={{ marginLeft: "20px"}}

                            type="number"

                            placeholder="Odd 2"

                            value={form.coefficient\_2}

                            onChange={(e) => changeField(e, 'coefficient\_2')}

                        />

                    ) : (

                        <p> <span className='fw-800'> x{game.teamGame2.coefficient}</span></p>

                    )}

                </div>

                Game start date:

                <div className="Modal-input">

                    {gameInfo.is\_active ? (

                        <input

                        type="datetime-local"

                        placeholder="Game start date"

                        value={form.game\_start\_date}

                        onChange={(e) => changeField(e, 'game\_start\_date')}

                    />

                    ) : (

                        <p>{formatGameDate(form.game\_start\_date)}</p>

                    )}

                </div>

                Game finish date:

                <div className="Modal-input">

                    {gameInfo.is\_active ? (

                        <input

                        type="datetime-local"

                        placeholder="Game finish date"

                        value={form.game\_finish\_date}

                        onChange={(e) => changeField(e, 'game\_finish\_date')}

                    />

                    ) : (

                        <p>{formatGameDate(form.game\_finish\_date)}</p>

                    )}

                </div>

                <div className="Modal-input">

                    <p>Is Game Active: <span className='fw-800'>{gameInfo.is\_active === true ? 'Yes' : 'No'}</span></p>

                </div>

                <div className="Modal-input">

                    <p>Is Bets Open: <span className='fw-800'>{gameInfo.is\_bets\_open === true ? 'Yes' : 'No'}</span></p>

                </div>

                <div className="Modal-input">

                    <p>Game type: <span className='fw-800'>{game.type === 'manual' ? 'Live' : 'Prematch'}</span></p>

                </div>

                <div className="Modal-input">

                    <p>Category: <span className='fw-800'>{game.category.name}</span></p>

                </div>

                <div style={{display: "flex", flexDirection: "row", marginBottom: '20px'}}>

                    {game.is\_active && (

                        <button

                            className={`Button\_secondary Modal-button`}

                            onClick={handleClickBets}

                            style={{marginRight: '10px'}}

                        >

                            {gameInfo.is\_bets\_open === false ? 'Open Bets' : 'Close bets'}

                        </button>

                    )}

                    <button

                        className={`Button\_secondary Modal-button`}

                        onClick={handleReturnAll}

                        disabled={!isFormReady}

                        style={{marginLeft: '10px'}}

                    >

                        Return all bets

                    </button>

                </div>

                {game.is\_active && (

                <button

                    className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                    onClick={handleSubmit}

                    disabled={!isFormReady}

                >

                    Save

                </button>

                )}

            </div>

            <div className="AdminGamesDetails-Right">

                <div>

                    <p>Total bets amount: {format2Num(gameInfo.totalAmount)}€</p>

                </div>

                <div style={{marginTop: '20px'}}>

                    <p>{game.teamGame1.team.name}: {format2Num(gameInfo.totalTeam1Amount)}€ ({format2Num(gameInfo.totalTeam1Amount / gameInfo.totalAmount \* 100)}%)</p>

                </div>

                <div>

                    <p>{game.teamGame2.team.name}: {format2Num(gameInfo.totalTeam2Amount)}€ ({format2Num(gameInfo.totalTeam2Amount / gameInfo.totalAmount \* 100)}%)</p>

                </div>

                <br />

                <h1 className='fw-800'>{game.teamGame1.team.name}</h1>

                <br />

                <table className="AdminGamesDetails-Right-Bets-Header">

                    <tr>

                        <td>Game ID</td>

                        <td>User ID</td>

                        <td>Email</td>

                        <td>Amount</td>

                    </tr>

                </table>

                <div className="AdminGamesDetails-Right-Bets-Items">

                    {gameInfo.betsTeam1 && gameInfo.betsTeam1.length > 0 && gameInfo.betsTeam1.map(bet =>

                        <div key={bet.id} className="AdminGamesDetails-Right-Bets-Items-Item">

                            <p>{bet.id}</p>

                            <p>{bet.user\_id}</p>

                            <p>{bet.user.email}</p>

                            <p>{bet.is\_returned ? 'RETURNED' : `${bet.amount}€`}</p>

                        </div>

                    )}

                </div>

                <br />

                <h1 className='fw-800'>{game.teamGame2.team.name}</h1>

                <br />

                <table className="AdminGamesDetails-Right-Bets-Header">

                    <tr>

                        <td>Game ID</td>

                        <td>User ID</td>

                        <td>Email</td>

                        <td>Amount</td>

                    </tr>

                </table>

                <div className="AdminGamesDetails-Right-Bets-Items">

                    {gameInfo.betsTeam2 && gameInfo.betsTeam2.length > 0 && gameInfo.betsTeam2.map(bet =>

                        <div key={bet.id} className="AdminGamesDetails-Right-Bets-Items-Item">

                            <p>{bet.id}</p>

                            <p>{bet.user\_id}</p>

                            <p>{bet.user.email}</p>

                            <p>{bet.is\_returned ? 'RETURNED' : `${bet.amount}€`}</p>

                        </div>

                    )}

                </div>

            </div>

        </div>

    );

}

export default AdminGamesDetails;

import { useContext, useEffect, useState } from 'react';

import './AdminTeams.scss';

import axios from 'axios';

import Modal from "../Modal/Modal";

import AdminTeamsCreate from './AdminTeamsCreate';

import { AuthContext } from '../../App';

import { handleDeleteImage } from '../../util/filesApi';

const AdminTeams = () => {

    const [teams, setTeams] = useState([]);

    const [isCreateModalOpen, setIsCreateModalOpen] = useState(false);

    const { authData } = useContext(AuthContext);

    const [teamToEdit, setTeamToEdit] = useState(null);

    const [filteredTeams, setFilteredTeams] = useState([]);

    const [search, setSearch] = useState('');

    const openCreate = () => {

        setIsCreateModalOpen(true);

    }

    const closeCreate = () => {

        updateTeams();

        setIsCreateModalOpen(false);

        setTeamToEdit(null);

    }

    useEffect(() => {

        updateTeams();

    }, []);

    const updateTeams = () => {

        axios.get('http://localhost:8080/team/')

            .then(r => {

                setTeams(r.data);

                setFilteredTeams(r.data);

                console.log(r.data);

            }).catch(err => {

                console.log(err);

            });

    };

    const handleDelete = (team) => {

        if (team?.img) {

            handleDeleteImage(team.img);

        }

        axios.post('http://localhost:8080/team/delete', {

            id: team.id,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                console.log(r.data);

                updateTeams();

            }).catch(err => {

                console.log(err);

            });

    };

    const handleEdit = (team) => {

        setTeamToEdit(team);

        setIsCreateModalOpen(true);

    }

    const handleSearch = () => {

        if (search === '') {

            setFilteredTeams(teams);

            return;

        }

        const filtered = teams.filter(team => {

            return team.name.toLowerCase().includes(search.toLowerCase())

        })

        setFilteredTeams(filtered);

    }

    const handleResetSearch = () => {

        setSearch('');

        setFilteredTeams(teams);

    }

    return (

        <div className="AdminTeams">

            <Modal isOpen={isCreateModalOpen} close={closeCreate} content={<AdminTeamsCreate close={closeCreate} team={teamToEdit} />}/>

            <div style={{display: 'flex', flexDirection: "row", alignItems: "center"}}>

                <div className="AdminTeams-Header">

                    <input

                        type="text"

                        value={search}

                        onChange={(e) => setSearch(e.target.value)}

                        onKeyDown={(e) => {

                            if (e.key === 'Enter') {

                                handleSearch();

                            }

                        }}

                    />

                    <button onClick={handleSearch} className="AdminTeams-Search" />

                </div>

                <button onClick={handleResetSearch} className='Button' style={{height: "40px", marginBottom: "20px", marginLeft: "20px", paddingInline: "30px"}}>Reset</button>

            </div>

            <div className="AdminTeams-Create">

                <button className="Button AdminTeams-Create-button" onClick={openCreate}>Create</button>

            </div>

            <table className="AdminTeams-Table">

                <thead className="AdminTeams-Table-Header">

                    <tr>

                        <td>ID</td>

                        <td>Image</td>

                        <td>Name</td>

                        <td>Action</td>

                    </tr>

                </thead>

                <tbody className="AdminTeams-Table-Content">

                    {filteredTeams.length > 0 && filteredTeams.map(team =>

                        <tr>

                            <td>{team.id}</td>

                            <td style={{width: "200px"}}><img src={team.img} alt="" /></td>

                            <td>{team.name}</td>

                            <td>

                                <div style={{marginBottom: "20px"}}>

                                    <button className='Button' onClick={() => handleEdit(team)}>EDIT</button>

                                </div>

                                <div>

                                    <button className='Button' onClick={() => handleDelete(team)}>DELETE</button>

                                </div>

                            </td>

                        </tr>

                    )}

                </tbody>

            </table>

        </div>

    );

};

export default AdminTeams;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

import { handleDeleteImage, handleSaveImage } from '../../util/filesApi';

export const AdminTeamsCreate = ({ close, team }) => {

    const [form, setForm] = useState({

        name: '',

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    const [img, setImg] = useState(null);

    useEffect(() => {

        setForm({

            name: team?.name,

        });

        setImg(null);

    }, [team])

    useEffect(() => {

        if(form.name) {

            setIsFormReady(true);

        } else {

            setIsFormReady(false);

        }

    }, [form]);

    const uploadImage = (e) => {

        const file = e.target.files[0];

        setImg(file);

    }

    const changeName = (e) => {

        const name = e.target.value;

        setForm({

            ...form,

            name: name,

        });

    }

    const handleSubmit = async () => {

        if (team?.name) {

            handleEdit();

        } else {

            handleCreate();

        }

    }

    const handleEdit = async () => {

        let newImg;

        if (team?.img && img) {

            handleDeleteImage(team.img);

        }

        if (img) {

            newImg = await handleSaveImage(img);

            console.log(newImg)

        }

        axios.post('http://localhost:8080/team/edit', {

            id: team.id,

            name: form.name,

            image: newImg,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                console.log(r.data);

                setImg(null);

                setForm({

                    name: '',

                });

                setError('');

                close();

            }).catch(err => {

                if (err && err.response) {

                    setError(err.response.data.error);

                }

            });

    }

    const handleCreate = async () => {

        const path = await handleSaveImage(img);

        console.log(path)

        axios.post('http://localhost:8080/team', {

            name: form.name,

            image: path,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                console.log(r.data);

                setImg(null);

                setForm({

                    name: '',

                });

                setError('');

                close();

            }).catch(err => {

                if (err && err.response) {

                    setError(err.response.data.error);

                }

            });

    }

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <input

                    type="text"

                    placeholder="Name"

                    value={form.name}

                    onChange={changeName}

                />

            </div>

            <div style={{marginBottom: "20px"}}>

                <input type="file" name="avatar" accept="image/\*" onChange={uploadImage} />

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                {team?.name ? 'Edit' : 'Create'}

            </button>

        </div>

    );

}

export default AdminTeamsCreate;

import { useContext, useEffect, useState } from 'react';

import './AdminUsers.scss';

import axios from 'axios';

import Modal from "../Modal/Modal";

import { AdminUsersDetails } from './AdminUsersDetails';

import { AuthContext } from '../../App';

import { format2Num } from '../../util/format';

const AdminUsers = () => {

    const [users, setUsers] = useState([]);

    const [filteredUsers, setFilteredUsers] = useState([]);

    const [isDetailsModalOpen, setIsDetailsModalOpen] = useState(false);

    const { authData } = useContext(AuthContext);

    const [selectedUser, setSelectedUser] = useState({});

    const [search, setSearch] = useState('');

    const openDetails = () => {

        setIsDetailsModalOpen(true);

    }

    const closeDetails = () => {

        updateUsers();

        setIsDetailsModalOpen(false);

    }

    useEffect(() => {

        updateUsers();

    }, []);

    const updateUsers = () => {

        console.log(authData.jwt)

        axios.get('http://localhost:8080/users',{

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            }

        })

            .then(r => {

                setFilteredUsers(r.data);

                setUsers(r.data);

                console.log(r.data);

            }).catch(err => {

                console.log(err);

            });

    };

    const handleDetails = (id) => {

        axios.get('http://localhost:8080/users/info?id=' + id,

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                setSelectedUser(r.data);

                openDetails();

            }).catch(err => {

                console.log(err);

            });

    };

    const handleSearch = () => {

        if (search === '') {

            setFilteredUsers(users);

            return;

        }

        const filtered = users.filter(user => {

            return user.name.toLowerCase().includes(search.toLowerCase())

                || user.email.toLowerCase().includes(search.toLowerCase());

        })

        console.log(filtered);

        setFilteredUsers(filtered);

    }

    const handleResetSearch = () => {

        setSearch('');

        setFilteredUsers(users);

    }

    return (

        <div className="AdminTeams">

            <div style={{display: 'flex', flexDirection: "row", alignItems: "center"}}>

                <div className="AdminTeams-Header">

                    <input

                        type="text"

                        value={search}

                        onChange={(e) => setSearch(e.target.value)}

                        onKeyDown={(e) => {

                            if (e.key === 'Enter') {

                                handleSearch();

                            }

                        }}

                    />

                    <button onClick={handleSearch} className="AdminTeams-Search" />

                </div>

                <button onClick={handleResetSearch} className='Button' style={{height: "40px", marginBottom: "20px", marginLeft: "20px", paddingInline: "30px"}}>Reset</button>

            </div>

            <Modal isOpen={isDetailsModalOpen} close={closeDetails} content={<AdminUsersDetails close={closeDetails} user={selectedUser} />} />

            <table className="AdminTeams-Table">

                <thead className="AdminTeams-Table-Header">

                    <tr>

                        <td>ID</td>

                        <td>Name</td>

                        <td>Email</td>

                        <td>Role</td>

                        <td>Balance</td>

                        <td>Banned</td>

                        <td>Action</td>

                    </tr>

                </thead>

                <tbody className="AdminTeams-Table-Content">

                    {filteredUsers.length > 0 && filteredUsers.map(user =>

                        <tr>

                            <td>{user.id}</td>

                            <td>{user.name}</td>

                            <td>{user.email}</td>

                            <td>{user.role.name}</td>

                            <td>{user.balance}€</td>

                            <td>{user.is\_locked ? 'Yes' : 'No'}</td>

                            <td className="AdminTeams-Table-Content-Action" onClick={() => handleDetails(user.id)}>Details</td>

                        </tr>

                    )}

                </tbody>

            </table>

        </div>

    );

};

export default AdminUsers;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

import { format2Num, formatGameDate } from '../../util/format';

export const AdminUsersDetails = ({ close, user }) => {

    const [form, setForm] = useState({

        name: user.name,

        balance: user.balance,

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    const [activeBets, setActiveBets] = useState([]);

    const [historyBets, setHistoryBets] = useState([]);

    useEffect(() => {

        fetchBets();

    }, []);

    const fetchBets = () => {

        axios.get(`http://localhost:8080/bet?active=1&email=${user.email}`,

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                setActiveBets(r.data);

                console.log(r.data);

            }).catch(err => {

                if (err && err.response) {

                    setError(err.response.data.error);

                }

            });

        axios.get(`http://localhost:8080/bet?active=0&email=${user.email}`,

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                setHistoryBets(r.data);

                console.log(r.data);

            }).catch(err => {

                if (err && err.response) {

                    setError(err.response.data.error);

                }

            });

    }

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        if(form.balance < 0) {

            setError('Balance should be more or equal 0');

            setIsFormReady(false);

            return;

        } else {

            setError('');

        }

        setIsFormReady(isFilled);

    }, [form]);

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    const handleSubmit = () => {

        axios.post('http://localhost:8080/users/update', {

            id: user.id,

            name: form.name,

            balance: parseFloat(form.balance),

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    const handleReturn = (id) => {

        axios.post('http://localhost:8080/bet/return', {

            id: id

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(() => {

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    const handleBanClick = () => {

        const action = user.is\_locked ? 'unban' : 'ban';

        axios.post('http://localhost:8080/users/' + action, {

            id: user.id

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(() => {

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div style={{width: "60vw"}}>

            <div className="Modal-Error">

                {error}

            </div>

            <div>

                <p>ID: {user.id}</p>

            </div>

            <div>

                <p>Email: {user.email}</p>

            </div>

            <div>

                <p>Name:

                    <input

                        type="text"

                        value={form.name}

                        onChange={(e) => changeField(e, 'name')}

                        style={{width: "150px"}}

                    />

                </p>

            </div>

            <div>

                <p>Role: {user.role.name}</p>

            </div>

            <div style={{display: "flex", alignItems: "center"}}>

                <p style={{marginRight:"20px"}}>Banned: {user.is\_locked ? 'Yes' : 'No'}</p>

                <button onClick={handleBanClick} className='Button'>{user.is\_locked ? 'Unban' : 'Ban'}</button>

            </div>

            <div>

                <p>Balance:

                    <input

                        type="number"

                        value={form.balance}

                        onChange={(e) => changeField(e, 'balance')}

                        style={{width: "150px"}}

                    />

                    €

                </p>

            </div>

            <button

                style={{ marginTop: "20px"}}

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                Update

            </button>

            <div style={{display: "flex", flexDirection: "row", marginTop: '20px', justifyContent:"space-between"}}>

                <p style={{width: "5%"}}>ID</p>

                <p style={{width: "15%"}}>Game</p>

                <p style={{width: "10%"}}>Selected Team</p>

                <p style={{width: "10%"}}>Odd</p>

                <p style={{width: "10%"}}>Bet</p>

                <p style={{width: "15%"}}>Game finish date</p>

                <p style={{width: "15%"}}>Action</p>

            </div>

            <div style={{height: "200px", border: "1px solid white", marginTop: "20px", borderRadius: "15px", overflowY: "auto"}}>

                {activeBets && activeBets.length > 0 && activeBets.map(bet =>

                    <div style={{display: "flex", flexDirection: "row", justifyContent:"space-between", marginBottom:"10px", alignItems: "center", padding: "10px"}}>

                        <p style={{width: "5%"}}>{bet.id}</p>

                        <p style={{width: "15%"}}>{bet.game.teamGame1.team.name} vs {bet.game.teamGame2.team.name}</p>

                        <p style={{width: "10%"}} className='fw-800'>{bet.selectedTeam === 'team1' ? bet.game.teamGame1.team.name : bet.game.teamGame2.team.name}</p>

                        <p style={{width: "10%"}} className='fw-800'>x{bet.selectedTeam === 'team1' ? bet.game.teamGame1.coefficient : bet.game.teamGame2.coefficient}</p>

                        <p style={{width: "10%"}}>{bet.amount}€</p>

                        <p style={{width: "15%"}}>{formatGameDate(bet.game.game\_finish\_date)}</p>

                        {bet.is\_returned ? (

                            <p>RETURNED</p>

                        ) : (

                            <button onClick={() => handleReturn(bet.id)} className='Button' style={{width: "15%"}}>Return</button>

                        )}

                    </div>

                )}

            </div>

            <div style={{display: "flex", flexDirection: "row", marginTop: '20px', justifyContent:"space-between"}}>

                <p style={{width: "5%"}}>ID</p>

                <p style={{width: "15%"}}>Game</p>

                <p style={{width: "10%"}}>Selected Team</p>

                <p style={{width: "10%"}}>Odd</p>

                <p style={{width: "10%"}}>Bet</p>

                <p style={{width: "10%"}}>Result</p>

                <p style={{width: "15%"}}>Game finish date</p>

                <p style={{width: "15%"}}>Action</p>

            </div>

            <div style={{height: "200px", border: "1px solid white", marginTop: "20px", borderRadius: "15px", overflowY: "auto"}}>

                {historyBets && historyBets.length > 0 && historyBets.map(bet =>

                    <div style={{display: "flex", flexDirection: "row", justifyContent:"space-between", marginBottom:"10px", alignItems: "center", padding: "10px"}}>

                        <p style={{width: "5%"}}>{bet.id}</p>

                        <p style={{width: "15%"}}>{bet.game.teamGame1.team.name} vs {bet.game.teamGame2.team.name}</p>

                        <p style={{width: "10%"}} className='fw-800'>{bet.selectedTeam === 'team1' ? bet.game.teamGame1.team.name : bet.game.teamGame2.team.name}</p>

                        <p style={{width: "10%"}} className='fw-800'>x{bet.selectedTeam === 'team1' ? bet.game.teamGame1.coefficient : bet.game.teamGame2.coefficient}</p>

                        <p style={{width: "10%"}}>{bet.amount}€</p>

                        <p style={{width: "10%"}}>

                            <span style={bet.game.team\_won === bet.selectedTeam ? {color: "#40e00f"} : {color: "#ff4c3f"}}>

                                {bet.game.team\_won === bet.selectedTeam ? (

                                    '+'

                                ) : (

                                    '-'

                                )}

                                <span className="fw-800" style={bet.game.team\_won === bet.selectedTeam ? {color: "#40e00f"} : {color: "#ff4c3f"}}>

                                    {bet.game.team\_won === bet.selectedTeam ? (

                                        bet.selectedTeam === 'team1' ? (

                                            format2Num(bet.amount \* bet.game.teamGame1.coefficient)

                                        ) : (

                                            format2Num(bet.amount \* bet.game.teamGame2.coefficient)

                                        )

                                    ) : (

                                        bet.amount

                                    )}

                                </span>

                                €

                            </span>

                        </p>

                        <p style={{width: "15%"}}>{formatGameDate(bet.game.game\_finish\_date)}</p>

                        {bet.is\_returned ? (

                            <p>RETURNED</p>

                        ) : (

                            <button onClick={() => handleReturn(bet.id)} className='Button' style={{width: "15%"}}>Return</button>

                        )}

                    </div>

                )}

            </div>

        </div>

    );

}

export default AdminUsersDetails;

import { useEffect } from "react";

import {format2Num, formatGameDate} from "../../util/format";

const BetsHistory = ({bets}) => {

    useEffect(() => {

        console.log(bets);

    }, [bets]);

    return (

        <div className="MyBets-Content">

            {bets && bets.length > 0 &&

                bets.map(bet =>

                    <div

                        className="MyBets-Content-Bet"

                        style={bet.is\_returned ? { backgroundColor: "#5e5d5d" } : {}}

                        key={bet.id}

                    >

                        <div className="MyBets-Content-Bet-Header">

                            <p>ID: {bet.id}</p>

                            <p>Finished: {formatGameDate(bet.game.game\_finish\_date)}</p>

                        </div>

                        <div className="MyBets-Content-Bet-Content">

                            <div className="MyBets-Content-Bet-Content-Game">

                                <p>{bet.game.teamGame1.team.name}</p>

                                <p>{bet.game.teamGame2.team.name}</p>

                            </div>

                            <div className="MyBets-Content-Bet-Content-Bet">

                                <p className="fw-800">

                                    {bet.selectedTeam === 'team1' ? bet.game.teamGame1.team.name : bet.game.teamGame2.team.name}

                                </p>

                                <p>Odd: <span

                                    className="fw-800">x{bet.selectedTeam === 'team1' ? bet.game.teamGame1.coefficient : bet.game.teamGame2.coefficient}</span>

                                </p>

                            </div>

                            <div className="MyBets-Content-Bet-Content-Result">

                                <p>Bet: <span className="fw-800">{bet.amount}</span>€</p>

                                {bet.is\_returned ? (

                                    <p className="fw-800" style={{color: "yellow"}}>RETURNED</p>

                                ) : (

                                    <p>

                                        Result:&nbsp;

                                        <span style={bet.game.team\_won === bet.selectedTeam ? {color: "#40e00f"} : {color: "#ff4c3f"}}>

                                            {bet.game.team\_won === bet.selectedTeam ? (

                                                '+'

                                            ) : (

                                                '-'

                                            )}

                                            <span className="fw-800" style={bet.game.team\_won === bet.selectedTeam ? {color: "#40e00f"} : {color: "#ff4c3f"}}>

                                                {bet.game.team\_won === bet.selectedTeam ? (

                                                    bet.selectedTeam === 'team1' ? (

                                                        format2Num(bet.amount \* bet.game.teamGame1.coefficient)

                                                    ) : (

                                                        format2Num(bet.amount \* bet.game.teamGame2.coefficient)

                                                    )

                                                ) : (

                                                    bet.amount

                                                )}

                                            </span>

                                            €

                                        </span>

                                    </p>

                                )}

                            </div>

                        </div>

                    </div>

                )

            }

        </div>

    );

}

export default BetsHistory;

import React, {useEffect, useState} from 'react';

import './Categories.scss';

import axios from "axios";

const Categories = ({selectedCategory, changeCategory}) => {

    const [categories, setCategories] = useState([]);

    const [search, setSearch] = useState('');

    const [filteredCategories, setFilteredCategories] = useState([]);

    useEffect(() => {

        fetchCategories()

            .then(r => {

                filter(r);

            });

    }, []);

    const fetchCategories = () => {

        return axios.get('http://localhost:8080/categories')

        .then(r => {

            setCategories(r.data);

            console.log(r.data)

            return r.data;

        }).catch(err => {

        console.log(err);

    });

    }

    const handleChangeCategory = (category) => {

        if(category.name === 'all' && categories.length > 0) {

            category.channel\_stream = categories[0].channel\_stream;

        }

        changeCategory(category);

    }

    const handleSearchChange = (e) => {

        setSearch(e.target.value);

    }

    const handleSearchClick = () => {

        filter(categories);

    }

    const filter = (c) => {

        if (search.length > 0) {

        setFilteredCategories(c.filter(category => category.name.toLowerCase().startsWith(search.toLowerCase())));

        } else {

            setFilteredCategories(c);

        }

    }

    return (

        <div className="Categories">

            <div className="Categories-Search">

                <input

                    className="Categories-Search-input"

                    type="text"

                    value={search}

                    onChange={handleSearchChange}

                />

                <button onClick={handleSearchClick} className="Categories-Search-button"/>

            </div>

            <div className="Categories-List">

                <ul>

                <li

                        className={selectedCategory.name === 'all' ? 'Categories-List\_selected-item' : ''}

                        onClick={() => handleChangeCategory({name: 'all'})}

                    >

                        All

                    </li>

                    {filteredCategories.length > 0 && filteredCategories.map(category =>

                        <li

                            onClick={() => handleChangeCategory(category)}

                            key={category.id}

                            className={selectedCategory.name === category.name ? 'Categories-List\_selected-item' : ''}

                            style={{display: "flex", alignItems: "center"}}

                        >

                            <img src={category.img} width='70px' style={{marginRight: '20px', background: 'none'}} />

                            {category.name}

                        </li>

                    )}

                </ul>

            </div>

        </div>

    );

};

export default Categories;

import React, {useCallback, useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

export const ChangePassword = ({ close }) => {

    const [form, setForm] = useState({

        oldPassword: '',

        newPassword: ''

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        setIsFormReady(isFilled);

    }, [form]);

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    const handleSubmit = () => {

        axios.post('http://localhost:8080/users/changepassword', {

            old\_password: form.oldPassword,

            new\_password: form.newPassword,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            setError('');

            close();

        }).catch(err => {

            console.log(err)

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <input

                    type="password"

                    placeholder="Old password"

                    value={form.oldPassword}

                    onChange={(e) => changeField(e, 'oldPassword')}

                />

            </div>

            <div className="Modal-input">

                <input

                    type="password"

                    placeholder="New password"

                    value={form.newPassword}

                    onChange={(e) => changeField(e, 'newPassword')}

                />

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                Change

            </button>

        </div>

    );

}

export default ChangePassword;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

import { saveAs } from 'file-saver';

export const Card = ({ close, amount }) => {

    const [form, setForm] = useState({

        number: '',

        holder: '',

        date: '',

        cvv: ''

    });

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    const { authData } = useContext(AuthContext);

    const [isLastStep, setIsLastStep] = useState(false);

    const [invoice, setInvoice] = useState(null);

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        if(!isFilled) {

            setIsFormReady(false);

            return;

        }

        if(form.number.toString().length !== 16) {

            setError('Card number should have 16 numbers');

            setIsFormReady(false);

            return;

        } else {

            setError('')

        }

        if(form.cvv.toString().length !== 3) {

            setError('CVV should have 16 numbers');

            setIsFormReady(false);

            return;

        } else {

            setError('')

        }

        setIsFormReady(isFilled);

    }, [form]);

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    const handleDownload = () => {

        saveAs(invoice, 'invoice.pdf');

    }

    const handleSubmit = () => {

        axios.post('http://localhost:8080/users/deposit', {

            id: authData.id,

            amount: amount,

            payment\_method: "card",

        },

        {

            responseType: 'blob',

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

        setIsLastStep(true);

        const blob = new Blob([r.data], { type: 'application/pdf' });

        setInvoice(blob);

        setError('');

    }).catch(err => {

        console.log(err)

        if (err && err.response) {

            setError(err.response.data.error);

        }

    });

    }

    return (

        isLastStep ? (

        <div>

            <h1>Thank you for your deposit</h1>

            <button

                className={`Button Modal-button`}

                onClick={handleDownload}

                style={{marginTop: "20px"}}

            >

                Download Invoice

            </button>

        </div>

        ) : (

        <div>

            <h1>Enter Credit Card Details</h1>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <p>Card Number:</p>

                <input

                    type="number"

                    placeholder="1234 5678 9012 3456"

                    value={form.number}

                    onChange={(e) => changeField(e, 'number')}

                />

            </div>

            <div className="Modal-input">

                <p>Card Holder:</p>

                <input

                    type="text"

                    placeholder="John Doe"

                    value={form.holder}

                    onChange={(e) => changeField(e, 'holder')}

                />

            </div>

            <div className="Modal-input">

                <p>Expire Date:</p>

                <input

                    type="text"

                    placeholder="MM/YY"

                    value={form.date}

                    onChange={(e) => changeField(e, 'date')}

                />

            </div>

            <div className="Modal-input">

                <p>CVV:</p>

                <input

                    type="number"

                    placeholder="123"

                    value={form.cvv}

                    onChange={(e) => changeField(e, 'cvv')}

                />

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                Change

            </button>

        </div>

        )

    );

}

export default Card;

import React, { useContext, useEffect, useState } from 'react';

import './Deposit.scss';

import { AuthContext } from '../../App';

import axios from 'axios';

import Modal from '../Modal/Modal';

import ChangePassword from '../ChangePassword/ChangePassword';

import Card from './Card';

const Deposit = () => {

    const [form, setForm] = useState({

        amount: '',

        type: '',

    });

    const { authData } = useContext(AuthContext);

    const [isModalOpen, setIsModalOpen] = useState(false);

    const [user, setUser] = useState(null);

    const [isFormReady, setIsFormReady] = useState(false);

    const openModal = () => {

        setIsModalOpen(true);

    };

    const closeModal = () => {

        setIsModalOpen(false);

    }

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        setIsFormReady(isFilled);

    }, [form]);

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    useEffect(() => {

        axios.get('http://localhost:8080/users/info?id=' + authData.id, {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

            setUser(r.data);

        }).catch(e => {

            console.log(e);

        })

    }, []);

    return (

        <div className="Payments">

            <Modal isOpen={isModalOpen} close={closeModal} content={<Card close={closeModal} amount={form.amount}/>} />

            <h1>Deposit</h1>

            <div>

                <input

                    type="text"

                    className="Input"

                    placeholder='Amount €'

                    value={form.amount}

                    onChange={(e) => changeField(e, 'amount')}

                />

            </div>

            <div>

                <select className='Select' style={{width: '200px', height: "40px", marginTop: "20px"}} value={form.type}

                    onChange={(e) => changeField(e, 'type')}>

                    <option value="" disabled selected>Payment method</option>

                    <option value="card">Credit card</option>

                </select>

            </div>

            <div style={{marginTop: "20px"}}>

                <button

                    style={{paddingInline: "20px", width: "377px"}}

                    className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                    onClick={openModal}

                    disabled={!isFormReady}

                >

                    Deposit

                </button>

            </div>

        </div>

    );

};

export default Deposit;

import React, {useEffect, useState, useContext} from 'react';

import './Games.scss';

import axios from "axios";

import BetsPanel from "../BetsPanel/BetsPanel";

import {formatGameDate} from "../../util/format";

import { Navigate, useNavigate } from 'react-router-dom';

import {AuthContext} from "../../App";

const Games = ({type, category}) => {

    const [games, setGames] = useState([]);

    const [selectedGame, setSelectedGame] = useState(null);

    const [selectedType, setSelectedType] = useState('auto');

    const [selectedBet, setSelectedBet] = useState(null);

    const [sort, setSort] = useState(null);

    const { authData, setIsLoginOpen } = useContext(AuthContext);

    const navigate = useNavigate();

    useEffect(() => {

        fetchGames()

    }, [category, selectedType, sort]);

    const fetchGames = () => {

        console.log(category.name);

        console.log(selectedType);

        const s = sort?.split('\_') || null;

        const sortBy = s && s[0];

        const order = s && s[1];

        axios.get(`http://localhost:8080/game?category=${category.name}&type=${selectedType}&is\_active=1&sort=${sortBy}&order=${order}`)

        .then((r) => {

            setGames(r.data);

            console.log(r.data);

        }).catch((err) => {

        console.log(err);

        });

    }

    const handleBetClick = (e, game, teamGame) => {

        const clickedButton = e.target;

        if (selectedBet) {

            selectedBet.style.backgroundColor = '#ffcf49';

        }

        clickedButton.style.backgroundColor = '#e06c51';

        setSelectedBet(clickedButton);

        const updatedGame = {

            ...game,

            selectedTeam: teamGame,

        }

        setSelectedGame(updatedGame);

        console.log(updatedGame)

    }

    const handleTypeChange = (type) => {

        setSelectedType(type);

    }

    const clearBet = () => {

        fetchGames();

        setSelectedBet(null);

        if (selectedBet) {

            selectedBet.style.backgroundColor = '#ffcf49';

        }

        setSelectedGame(null);

    }

    const handleSortChange = (e) => {

        setSort(e.target.value)

    }

    const handleDepositClick = () => {

        if(!authData.id) {

            setIsLoginOpen(true);

            return;

        }

        navigate('/profile?deposit=true')

    }

    return (

        <div className="Games">

            <div className="Games-Main">

                <div className="Games-Main-Top">

                    <div className="Games-Main-Top-Stream">

                        <iframe

                            title='twitch'

                            src={`https://player.twitch.tv/?channel=${category.channel\_stream}&parent=localhost`}

                            allowFullScreen

                        ></iframe>

                    </div>

                    <div className="Games-Main-Top-info" style={{ display: "flex", flexDirection: "column", alignItems: "center", justifyContent: "space-around"}}>

                        <div>

                        <div style={{display:"flex"}}>

                            <div style={{ display:"flex", justifyContent: "center", alignItems: "center", color: "black", fontWeight: 800, width: "40px", height: "40px", borderRadius: "20px", backgroundColor: "#ffcf49"}}>1</div>

                            <div style={{marginLeft: "20px", width: "80%"}}>

                                <p style={{ fontWeight: 700 }}>Select an event</p>

                                <p style={{textWrap: "wrap"}}>Select an event and click on the odds. Example: “Win1” means the home team wins.</p>

                            </div>

                        </div>

                        <div style={{display:"flex", marginTop: "50px"}}>

                            <div style={{ display:"flex", justifyContent: "center", alignItems: "center", color: "black", fontWeight: 800, width: "40px", height: "40px", borderRadius: "20px", backgroundColor: "#ffcf49"}}>2</div>

                            <div style={{marginLeft: "20px", width: "80%"}}>

                                <p style={{ fontWeight: 700 }}>Place your bet</p>

                                <p style={{textWrap: "wrap"}}>Enter the amount and the system will calculate the possible winnings, then click “Place a bet”.</p>

                            </div>

                        </div>

                        </div>

                        <button onClick={handleDepositClick} className='Button' style={{width: "80%", height: "40px"}}>Deposit</button>

                    </div>

                </div>

                <div className="Games-Main-Type">

                    <button

                        className={`Button\_secondary ${selectedType === 'auto' ? 'Button\_secondary\_selected' : ''}`}

                        onClick={() => handleTypeChange('auto')}

                    >

                        Prematch

                    </button>

                    <button

                        className={`Button\_secondary ${selectedType === 'manual' ? 'Button\_secondary\_selected' : ''}`}

                        onClick={() => handleTypeChange('manual')}

                    >

                        Live

                    </button>

                </div>

                <div className="Games-Main-Items">

                <select className='Select\_secondary' style={{width: "200px", marginBottom: "20px"}} value={sort} onChange={handleSortChange}>

                        <option value={null} disabled selected>Sort</option>

                        <option value="id\_desc">Newest</option>

                        <option value="id\_asc">Oldest</option>

                        <option value="team1\_asc">Team1 Odd: Low to High</option>

                        <option value="team1\_desc">Team1 Odd: High to Low</option>

                        <option value="team2\_asc">Team2 Odd: Low to High</option>

                        <option value="team2\_desc">Team2 Odd: High to Low</option>

                        <option value="startDate\_desc">StartDate: Descending</option>

                        <option value="startDate\_asc">Start Date: Ascending</option>

                    </select>

                {games.length > 0 && games.map(game =>

                        <table key={game.id} className="Games-Main-Items-Game">

                            <tr className="Games-Main-Items-Game-Header">

                                <td>Game</td>

                                <td>Start date: {formatGameDate(game.game\_start\_date)}</td>

                                <td>Team 1</td>

                                <td>Team 2</td>

                                <td>Info</td>

                            </tr>

                            <tr className="Games-Main-Items-Game-Content">

                                <td>

                                    {game.category.name}

                                </td>

                                <td style={{display: "flex", justifyContent: "center", alignItems: "center"}}>

                                    <img src={game.teamGame1.team.img} alt="" style={{ width: "50px", marginRight: "10px" }} />

                                    <p>{game.teamGame1.team.name} vs {game.teamGame2.team.name}</p>

                                    <img src={game.teamGame2.team.img} alt="" style={{ width: "50px", marginLeft: "10px" }} />

                                </td>

                                <td>

                                    <button

                                        onClick={(e) => handleBetClick(e, game, game.teamGame1)}

                                        className="Button Games-Main-Items-Game-Content-Coefficient"

                                    >

                                        x{parseFloat(game.teamGame1.coefficient.toFixed(2))}

                                    </button>

                                </td>

                                <td>

                                    <button

                                        onClick={(e) => handleBetClick(e, game, game.teamGame2)}

                                        className="Button Games-Main-Items-Game-Content-Coefficient"

                                    >

                                        x{parseFloat(game.teamGame2.coefficient.toFixed(2))}

                                    </button>

                                </td>

                                <td>

                                    <button className="Button\_secondary">Details</button>

                                </td>

                            </tr>

                        </table>

                    )}

                </div>

            </div>

            <div className="Games-BetsPanel">

                <BetsPanel bet={selectedGame} close={clearBet}/>

            </div>

        </div>

    );

};

export default Games;

import React, {useContext, useState} from 'react';

import './Header.scss';

import Modal from "../Modal/Modal";

import Register from "../Register/Register";

import Login from "../Login/Login";

import {AuthContext} from "../../App";

import { Link, Navigate, useNavigate } from 'react-router-dom';

const Header = () => {

    const navigate = useNavigate();

    const [isRegisterModalOpen, setIsRegisterModalOpen] = useState(false);

    const { authData, updateAuthData, isLoginOpen, setIsLoginOpen } = useContext(AuthContext);

    const openLogin = () => {

        setIsLoginOpen(true);

    }

    const closeLogin = () => {

        setIsLoginOpen(false);

    }

    const openRegister = () => {

        setIsRegisterModalOpen(true);

    }

    const closeRegister = () => {

        setIsRegisterModalOpen(false);

    }

    const logout = () => {

        localStorage.removeItem('user');

        updateAuthData({});

    }

    const handleProfile = () => {

        navigate('/profile');

    }

    const renderButtons = () => {

        if (!authData.id) {

            return (

                <div>

                    <button

                        className="Button Header-Buttons-Button"

                        onClick={openRegister}

                    >

                        Register

                    </button>

                     <Modal isOpen={isRegisterModalOpen} close={closeRegister} content={<Register close={closeRegister} />}/>

                    <button

                        className="Button\_secondary Header-Buttons-Button"

                        onClick={openLogin}

                    >

                        Login

                    </button>

                    <Modal isOpen={isLoginOpen} close={closeLogin} content={<Login close={closeLogin} />} />

                </div>

            );

        }

        return (

            <div>

                {authData.role === 'admin' &&

                    <button className="Button Header-Buttons-Button" onClick={() => {navigate('/admin')}}>

                        Admin Panel

                    </button>

                }

                <button

                    className="Button Header-Buttons-Button"

                    onClick={handleProfile}

                >

                    Profile

                </button>

                <button

                    className="Button Header-Buttons-Button"

                    onClick={logout}

                >

                    Logout

                </button>

            </div>

        );

    }

    return (

        <div className="Header">

            <div className="Header-Logo" onClick={() => navigate('/games')}>

                <div className="Header-Logo-Img"></div>

                <h1 className="Header-Logo-Title">eWager</h1>

            </div>

            <div className="Header-Buttons" style={{display: "flex"}}>

                <button className="Button\_secondary Header-Buttons-Button" onClick={() => {navigate('/about')}}>

                    About us

                </button>

                <button style={{marginRight: "50px"}} className="Button\_secondary Header-Buttons-Button" onClick={() => {navigate('/contacts')}}>

                    Contacts

                </button>

                { renderButtons() }

            </div>

        </div>

    );

};

export default Header;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

const Login = ({close}) => {

    const [form, setForm] = useState({

        email: '',

        password: '',

    });

    const { updateAuthData } = useContext(AuthContext);

    const [error, setError] = useState('');

    const [isFormReady, setIsFormReady] = useState(false);

    useEffect(() => {

        if(form.email && form.password) {

            setIsFormReady(true);

        } else {

            setIsFormReady(false);

        }

    }, [form]);

    const changeEmail = (e) => {

        const email = e.target.value;

        setForm({

            ...form,

            email: email,

        });

    }

    const changePassword = (e) => {

        const password = e.target.value;

        setForm({

            ...form,

            password: password,

        });

    }

    const handleSubmit = () => {

        axios.post('http://localhost:8080/auth/login', {

            email: form.email,

            password: form.password

        }).then(r => {

            const user = {

                email: form.email,

                id: r.data.id,

                jwt: r.data.token,

                balance: r.data.balance,

                role: r.data.role,

            };

            updateAuthData(user);

            localStorage.setItem('user', JSON.stringify(user));

            console.log(r.data);

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <input

                    type="text"

                    placeholder="Email"

                    value={form.email}

                    onChange={changeEmail}

                />

            </div>

            <div className="Modal-input">

                <input

                    type="password"

                    placeholder="Password"

                    value={form.password}

                    onChange={changePassword}

                />

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                Log In

            </button>

        </div>

    );

};

export default Login;

import React from 'react';

import './Modal.scss';

const Modal = ({isOpen, close, content}) => {

    const body = document.body;

    if(!isOpen) {

        body.classList.remove('body-lock');

        return null;

    } else {

        body.classList.add('body-lock');

    }

    const stopPropagation = (e) => {

        e.stopPropagation()

    }

    return (

        <div className="ModalWrapper" onClick={close}>

            <div className="Modal" onClick={stopPropagation}>

                <button className="Modal-Close" onClick={close} />

                {content}

            </div>

        </div>

    );

};

export default Modal;

import React, {useContext, useEffect, useState} from 'react';

import './MyBets.scss';

import {AuthContext} from "../../App";

import axios from "axios";

import ActiveBets from '../ActiveBets/ActiveBets';

import BetsHistory from '../BetsHistory/BetsHistory';

const List = {

    ACTIVE: 'active',

    HISTORY: 'history',

}

const MyBets = ({close}) => {

    const { authData } = useContext(AuthContext);

    const [selectedList, setSelectedList] = useState(List.ACTIVE);

    const [bets, setBets] = useState([]);

    const [sort, setSort] = useState(null);

    useEffect(() => {

        fetchBets();

    }, [selectedList, sort]);

    const fetchBets = () => {

        const s = sort?.split('\_') || null;

        const sortBy = s && s[0];

        const order = s && s[1];

        const url = `http://localhost:8080/bet?active=${selectedList === List.ACTIVE ? '1' : '0'}&sort=${sortBy}&order=${order}`;

        axios.get(url, {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

            setBets(r.data);

        }).catch((err) => {

            console.log(err);

        });

    }

    const handleChangeList = (list) => {

        setSelectedList(list);

    }

    const handleSortChange = (e) => {

        setSort(e.target.value)

    }

    return (

        <div className="MyBets">

            <div className="MyBets-Header">

                <button

                    className={`Button\_secondary ${selectedList === List.ACTIVE ? 'Button\_secondary\_selected' : ''}`}

                    onClick={() => handleChangeList(List.ACTIVE)}

                >

                    Active bets

                </button>

                <button

                    className={`Button\_secondary ${selectedList === List.HISTORY ? 'Button\_secondary\_selected' : ''}`}

                    onClick={() => handleChangeList(List.HISTORY)}

                >

                    History

                </button>

            </div>

            <div className="MyBets-Sort" style={{marginBottom: '20px'}}>

                <select className='Select' style={{width: "200px"}} value={sort} onChange={handleSortChange}>

                    <option value={null} disabled selected>Sort</option>

                    <option value="id\_desc">Newest</option>

                    <option value="id\_asc">Oldest</option>

                    <option value="amount\_asc">Amount: Low to High</option>

                    <option value="amount\_desc">Amount: High to Low</option>

                    <option value="date\_desc">Date: Descending</option>

                    <option value="date\_asc">Date: Ascending</option>

                </select>

            </div>

            {selectedList === List.ACTIVE ? (

                <ActiveBets bets={bets} />

            ) : (

                <BetsHistory bets={bets} />

            )}

        </div>

    );

};

export default MyBets;

import React, {useEffect, useState} from 'react';

import axios from "axios";

const Register = ({close}) => {

    const [form, setForm] = useState({

        email: '',

        name: '',

        password: '',

    });

    const [isFormReady, setIsFormReady] = useState(false);

    const [error, setError] = useState('');

    useEffect(() => {

        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

        const isEmailValid = emailRegex.test(form.email);

        if(form.email && form.password && form.name) {

            if (isEmailValid) {

                setError('');

                setIsFormReady(true);

            } else {

                setError('Incorrect email format');

            }

        } else {

            setIsFormReady(false);

        }

    }, [form]);

    const changeEmail = (e) => {

        const email = e.target.value;

        setForm({

            ...form,

            email: email,

        });

    }

    const changeName = (e) => {

        const name = e.target.value;

        setForm({

            ...form,

            name: name,

        });

    }

    const changePassword = (e) => {

        const password = e.target.value;

        setForm({

            ...form,

            password: password,

        });

    }

    const handleSubmit = () => {

        axios.post('http://localhost:8080/auth/register', {

            email: form.email,

            name: form.name,

            password: form.password

        }).then(r => {

            setError('');

            close();

        }).catch(err => {

            if (err && err.response) {

                console.log(err.response.data);

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <input

                    type="text"

                    placeholder="Email"

                    value={form.email}

                    onChange={changeEmail}

                />

            </div>

            <div className="Modal-input">

                <input

                    type="text"

                    placeholder="Name"

                    value={form.name}

                    onChange={changeName}

                />

            </div>

            <div className="Modal-input">

                <input

                    type="password"

                    placeholder="Password"

                    value={form.password}

                    onChange={changePassword}

                />

            </div>

            <button

                className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                onClick={handleSubmit}

                disabled={!isFormReady}

            >

                Register

            </button>

        </div>

    );

};

export default Register;

import React, { useEffect, useState, useContext } from 'react';

import axios from 'axios';

import { AuthContext } from '../../App';

import './Transactions.scss';

const Transactions = () => {

    const [transactions, setTransactions] = useState([]);

    const { authData } = useContext(AuthContext);

    useEffect(() => {

        fetchTransactions();

    }, []);

    const fetchTransactions = async () => {

        axios.get('http://localhost:8080/users/transactions?id=' + authData.id,

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

                console.log(r.data);

                setTransactions(r.data);

            }).catch(err => {

                console.log(err)

            });

    };

    const renderWithdraw = (transaction) => {

        return (

            <li className='Transactions-Item' key={transaction.id + transaction.receiver}>

                <p style={{color: "#40e00f", width: "30%"}}>+{transaction.amount}€&nbsp;</p>

                <p style={{width: "30%"}}>Bank Acc Number: {transaction.receiver}&nbsp;</p>

                <p style={{width: "30%"}}>Date: {new Date(transaction.date).toLocaleString()}</p>

            </li>

        )

    }

    const renderDeposit = (transaction) => {

        return (

            <li className='Transactions-Item' key={transaction.id + transaction.payment\_method}>

                <p style={{color: "#ff4c3f", width: "30%"}}>-{transaction.amount}€</p>

                <p style={{width: "30%"}}>Method: {transaction.payment\_method}</p>

                <p style={{width: "30%"}}>Date: {new Date(transaction.date).toLocaleString()}</p>

            </li>

        )

    }

    return (

        <div>

            <h1>Transactions</h1>

            <ul>

                {transactions.map((transaction) => (

                    transaction.receiver ? renderWithdraw(transaction) : renderDeposit(transaction)

                ))}

            </ul>

        </div>

    );

};

export default Transactions;

import React, { useContext, useEffect, useState } from 'react';

import './Withdraw.scss';

import { AuthContext } from '../../App';

import axios from 'axios';

import { format2Num } from '../../util/format';

const Withdraw = () => {

    const [form, setForm] = useState({

        amount: '',

        receiver: '',

    });

    const { authData } = useContext(AuthContext);

    const [user, setUser] = useState(null);

    const [isFormReady, setIsFormReady] = useState(false);

    const [error, setError] = useState('');

    useEffect(() => {

        const formValues = Object.values(form);

        const isFilled = formValues.every(value => value !== '' && value !== null);

        setIsFormReady(isFilled);

    }, [form]);

    const changeField = (e, field) => {

        setForm({

            ...form,

            [field]: e.target.value,

        });

    };

    useEffect(() => {

        fetchUser();

    }, []);

    const fetchUser = () => {

        axios.get('http://localhost:8080/users/info?id=' + authData.id, {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

            setUser(r.data);

        }).catch(e => {

            console.log(e);

        });

    }

    const handleSubmit = () => {

        axios.post('http://localhost:8080/users/withdraw', {

            id: authData.id,

            amount: form.amount,

            receiver: form.receiver,

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

    }).then(r => {

            console.log(r.data);

            fetchUser();

            setError('');

        }).catch(err => {

            console.log(err)

            fetchUser();

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div className="Payments">

            <h1>Withdraw</h1>

            <p>Current balance: {user ? user.balance : '...'}€</p>

            <div className="Modal-Error">

                {error}

            </div>

            <div>

                <input

                    type="text"

                    className="Input"

                    placeholder='Amount €'

                    value={form.amount}

                    onChange={(e) => changeField(e, 'amount')}

                />

            </div>

            <div style={{marginTop: "20px"}}>

                <input

                    type="text"

                    className="Input"

                    placeholder='Bank account number'

                    value={form.receiver}

                    onChange={(e) => changeField(e, 'receiver')}

                />

            </div>

            <div style={{marginTop: "20px"}}>

                <button

                    style={{paddingInline: "20px", width: "377px"}}

                    className={`Button Modal-button ${isFormReady ? '' : 'Button\_disabled'}`}

                    onClick={handleSubmit}

                    disabled={!isFormReady}

                >

                    Withdraw

                </button>

            </div>

        </div>

    );

};

export default Withdraw;

import React from 'react';

const About = () => {

    return (

        <div style={{display: "flex", flexDirection: "column", justifyContent: "center", alignItems: "center"}}>

            <h1 style={{fontSize: 40, fontWeight: 700}}>About Us</h1>

            <p style={{ marginTop: "80px", fontSize: 24, whiteSpace: "pre-wrap", textAlign: "justify", lineHeight: "2", width: "50%" }}>

                &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Welcome to the website of our betting company, specializing in esports betting!

                Who We Are

                We are a team of esports enthusiasts united by a shared passion for this rapidly growing world. Our mission is to provide esports fans with a unique and thrilling betting experience that not only complements your enthusiasm but also adds even more excitement and opportunities.

                <br /><br />

                &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Our Mission: We strive to become the leading provider of esports betting, offering the most competitive odds, a wide selection of events, and unique bonuses for our users. We understand that esports are not just games, but a whole world where millions of people around the globe live and breathe. We are here to help you become a part of this world, making it even more exciting and rewarding.

            </p>

            <img src='images/aboutimg.png' style={{ marginTop: "50px", width: "50%"}} />

        </div>

    );

};

export default About;

import { useState } from 'react';

import './AdminPanel.scss';

import AdminGames from '../../component/AdminGames/AdminGames';

import AdminTeams from '../../component/AdminTeams/AdminTeams';

import AdminCategories from '../../component/AdminCategories/AdminCategories';

import AdminUsers from '../../component/AdminUsers/AdminUsers';

const Tabs = {

    GAMES: {

        name: 'games',

        content: <AdminGames />

    },

    TEAMS: {

        name: 'teams',

        content: <AdminTeams />

    },

    USERS: {

        name: 'users',

        content: <AdminUsers />

    },

    CATEGORIES: {

        name: 'categories',

        content: <AdminCategories />

    }

};

const AdminPanel = () => {

    const [selectedTab, setSelectedTab] = useState(Tabs.GAMES);

    return (

        <div className="AdminPanel">

            <div className="AdminPanel-Menu">

                <p

                    className={`AdminPanel-Menu-Items ${selectedTab.name === Tabs.GAMES.name ? 'AdminPanel-Menu-Items\_selected' : ''}`}

                    onClick={() => setSelectedTab(Tabs.GAMES)}

                    >

                        Games

                </p>

                <p

                    className={`AdminPanel-Menu-Items ${selectedTab.name === Tabs.CATEGORIES.name ? 'AdminPanel-Menu-Items\_selected' : ''}`}

                    onClick={() => setSelectedTab(Tabs.CATEGORIES)}

                >

                    Categories

                </p>

                <p

                    className={`AdminPanel-Menu-Items ${selectedTab.name === Tabs.TEAMS.name ? 'AdminPanel-Menu-Items\_selected' : ''}`}

                    onClick={() => setSelectedTab(Tabs.TEAMS)}

                >

                    Teams

                </p>

                <p

                    className={`AdminPanel-Menu-Items ${selectedTab.name === Tabs.USERS.name ? 'AdminPanel-Menu-Items\_selected' : ''}`}

                    onClick={() => setSelectedTab(Tabs.USERS)}

                >

                    Users

                </p>

            </div>

            <div className="AdminPanel-Content">

                {selectedTab.content}

            </div>

        </div>

    );

};

export default AdminPanel;

import React from 'react';

import { IoMdMail } from "react-icons/io";

import { FaMapLocationDot } from "react-icons/fa6";

import { FaPhoneVolume } from "react-icons/fa6";

const Contacts = () => {

    return (

        <div style={{display: "flex", flexDirection: "column", justifyContent: "center", alignItems: "center"}}>

            <h1 style={{fontSize: 40, fontWeight: 700}}>Contacts</h1>

            <h1 style={{fontSize: 28, fontWeight: 600}}>Any questions? Contact us in any convenient way</h1>

            <div style={{ display: "flex", flexDirection: "row", width: "100%", justifyContent: "center", marginTop: "100px"}}>

                <div style={{ backgroundColor: "#121212", width: "500px", height: "350px", borderRadius: "20px", display: "flex", flexDirection: "column", alignItems: "center", justifyContent: "center"}}>

                <IoMdMail  style={{ width: "150px", height: "150px" }} size="sm"/>

                <p style={{color: 'gray'}}>Email tech. support</p>

                <p style={{ fontWeight: 700, color: "#ffcf49",fontSize: 24, marginTop: "10px" }}>support@ewager.com</p>

                </div>

                <div style={{ backgroundColor: "#121212", width: "500px", height: "350px", borderRadius: "20px", display: "flex", flexDirection: "column", alignItems: "center", justifyContent: "center", marginLeft: "50px", marginRight: "50px"}}>

                <FaPhoneVolume  style={{ width: "150px", height: "150px" }} size="sm"/>

                <p style={{color: 'gray'}}>Hotline number</p>

                <p style={{ fontWeight: 700, color: "#ffcf49", fontSize: 24, marginTop: "10px" }}>+371 50 43 43 12</p>

                </div>

                <div style={{ backgroundColor: "#121212", width: "500px", height: "350px", borderRadius: "20px", display: "flex", flexDirection: "column", alignItems: "center", justifyContent: "center", marginLeft: "50px", marginRight: "50px"}}>

                <FaMapLocationDot  style={{ width: "150px", height: "150px" }} size="sm"/>

                <p style={{color: 'gray'}}>Address</p>

                <p style={{ fontWeight: 700, color: "#ffcf49", fontSize: 24, marginTop: "10px" }}>Riga</p>

            </div>

            </div>

        </div>

    );

};

export default Contacts;

import React, { useContext, useEffect, useState } from 'react';

import { useParams } from "react-router-dom";

import './GamesPage.scss';

import Categories from "../../component/Categories/Categories";

import Games from "../../component/Games/Games";

import { AuthContext } from '../../App';

import axios from 'axios';

const GamesPage = () => {

    const params = useParams();

    const { type } = params;

    const [selectedCategory, setSelectedCategory] = useState({ name: 'all' });

    const { fetchNewUserData } = useContext(AuthContext);

    useEffect(() => {

        fetchNewUserData();

    }, []);

    const changeCategory = (category) => {

        setSelectedCategory(category);

    }

    const [img, setImg] = useState(null);

    const uploadImage = (e) => {

        const file = e.target.files[0];

        setImg(file);

    }

    const handleSaveImage = async () => {

        const formData = new FormData();

        formData.append('image', img);

        try {

            const response = await axios.post('http://localhost:8080/upload', formData, {

                headers: {

                    'Content-Type': 'multipart/form-data',

                },

            });

            console.log(response.data);

        } catch (error) {

            console.error('Error:', error);

        }

    }

    return (

        <div className="GamesPage">

            <div className="GamesPage-Categories">

                <Categories selectedCategory={selectedCategory} changeCategory={changeCategory} />

            </div>

            <div className="GamesPage-Games">

                <Games type={type} category={selectedCategory} />

            </div>

        </div>

    );

};

export default GamesPage;

import React, {useContext, useEffect, useState} from 'react';

import axios from "axios";

import { AuthContext } from '../../App';

const Ban = ({close}) => {

    const { authData, updateAuthData } = useContext(AuthContext);

    const [error, setError] = useState('');

    const handleSubmit = () => {

        // axios.post('http://localhost:8080/users/banmyself', {

        //     headers: {

        //         'Authorization': `Bearer ${authData.jwt}`,

        //     },

        // }).then(r => {

        //     updateAuthData(null);

        //     localStorage.removeItem('user');

        //     console.log(r.data);

        //     setError('');

        //     close();

        // }).catch(err => {

        //     console.log(err)

        //     if (err && err.response) {

        //         setError(err.response.data.error);

        //     }

        // });

        axios.post('http://localhost:8080/users/banmyself', {

        },

        {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

            localStorage.removeItem('user');

            updateAuthData({});

            console.log(r.data);

            setError('');

            close();

        }).catch(err => {

            console.log(err)

            if (err && err.response) {

                setError(err.response.data.error);

            }

        });

    }

    return (

        <div>

            <div className="Modal-Error">

                {error}

            </div>

            <div className="Modal-input">

                <p>Are you sure you want to ban? Ban will be forever</p>

            </div>

            <button

                className={`Button Modal-button`}

                onClick={handleSubmit}

            >

                Confirm

            </button>

        </div>

    );

};

export default Ban;

import React, { useContext, useEffect, useState } from 'react';

import './Profile.scss';

import { AuthContext } from '../../App';

import axios from 'axios';

import ProfileInfo from './ProfileInfo';

import Deposit from '../../component/Deposit/Deposit';

import Transactions from '../../component/Transactions/Transactions';

import Withdraw from '../../component/Withdraw/Withdraw';

const Tabs = {

    INFO: {

        name: 'info',

        content: <ProfileInfo />

    },

    DEPOSIT: {

        name: 'deposit',

        content: <Deposit />

    },

    WITHDRAW: {

        name: 'withdraw',

        content: <Withdraw />

    },

    TRANSACTIONS: {

        name: 'transactions',

        content: <Transactions />

    },

};

const Profile = () => {

    const [selectedTab, setSelectedTab] = useState(Tabs.INFO);

    const queryParams = new URLSearchParams(window.location.search);

    const isDeposit = queryParams.get('deposit') ? true : false;

    useEffect(() => {

        if (isDeposit) {

            setSelectedTab(Tabs.DEPOSIT);

        }

    }, [isDeposit])

    const renderMenu = () => {

        return (

            <div className="Profile-Menu">

                <div className={`Profile-Menu-Item ${selectedTab === Tabs.INFO ? 'Profile-Menu-Item\_selected' : ''}`} onClick={() => setSelectedTab(Tabs.INFO)}>Info</div>

                <div className={`Profile-Menu-Item ${selectedTab === Tabs.DEPOSIT ? 'Profile-Menu-Item\_selected' : ''}`} onClick={() => setSelectedTab(Tabs.DEPOSIT)}>Deposit</div>

                <div className={`Profile-Menu-Item ${selectedTab === Tabs.WITHDRAW ? 'Profile-Menu-Item\_selected' : ''}`} onClick={() => setSelectedTab(Tabs.WITHDRAW)}>Withdraw</div>

                <div className={`Profile-Menu-Item ${selectedTab === Tabs.TRANSACTIONS ? 'Profile-Menu-Item\_selected' : ''}`} onClick={() => setSelectedTab(Tabs.TRANSACTIONS)}>Transactions</div>

            </div>

        );

    }

    return (

        <div className="Profile">

            { renderMenu() }

            <div className="Profile-Content">

                {selectedTab.content}

            </div>

        </div>

    );

};

export default Profile;

import React, { useContext, useEffect, useState } from 'react';

import './Profile.scss';

import { AuthContext } from '../../App';

import axios from 'axios';

import Modal from '../../component/Modal/Modal';

import ChangePassword from '../../component/ChangePassword/ChangePassword';

import Ban from './Ban';

const ProfileInfo = () => {

    const { authData } = useContext(AuthContext);

    const [isChangePasswordModalOpen, setIsChangePasswordModalOpen] = useState(false);

    const [isBanModalOpen, setIsBanModalOpen] = useState(false);

    const [user, setUser] = useState(null);

    const openModal = () => {

        setIsChangePasswordModalOpen(true);

    };

    const closeModal = () => {

        setIsChangePasswordModalOpen(false);

    }

    const openBanModal = () => {

        setIsBanModalOpen(true);

    };

    const closeBanModal = () => {

        setIsBanModalOpen(false);

    }

    useEffect(() => {

        axios.get('http://localhost:8080/users/info?id=' + authData.id, {

            headers: {

                'Authorization': `Bearer ${authData.jwt}`,

            },

        }).then(r => {

            setUser(r.data);

        }).catch(e => {

            console.log(e);

        })

    }, []);

    return (

        <div className="ProfileInfo">

            <Modal content={<ChangePassword close={closeModal} />} isOpen={isChangePasswordModalOpen} close={closeModal} />

            <Modal content={<Ban close={closeBanModal} />} isOpen={isBanModalOpen} close={closeBanModal} />

            {user && (

                <div className='ProfileInfo-Info'>

                    <p>Email: {user.email}</p>

                    <p>Name: {user.name}</p>

                    <p>Role: {user.role.name}</p>

                    <p>Balance: {user.balance}€</p>

                    <button className="Button\_secondary" onClick={openModal}>Change password</button>

                    {user.role.name !== 'admin' && (

                    <div style={{marginTop: "20px"}}>

                        <button className="Button" onClick={openBanModal} style={{backgroundColor: "rgb(230,20,20)", color: "white", padding:"20px"}}>Ban Myself</button>

                    </div>

                    )}

                </div>

            )}

        </div>

    );

};

export default ProfileInfo;

import axios from 'axios';

export const handleSaveImage = async (img) => {

    const formData = new FormData();

    formData.append('image', img);

    return axios.post('http://localhost:8080/upload', formData, {

        headers: {

            'Content-Type': 'multipart/form-data',

        },

    }).then(r => {

        console.log(r.data);

        return r.data;

    }).catch(err => {

        console.log(err);

        return null;

    });

}

export const handleDeleteImage = async (path) => {

    return axios.post('http://localhost:8080/deleteImg', {

        path: path,

    }).then(r => {

        console.log(r.data);

        return r.data;

    }).catch(err => {

        console.log(err);

        return null;

    });

}

export const formatGameDate = (dateString) => {

    const date = new Date(dateString);

    const day = date.getDate();

    const month = date.getMonth() + 1;

    const hours = date.getHours();

    const minutes = date.getMinutes();

    const formattedDay = day < 10 ? `0${day}` : day;

    const formattedMonth = month < 10 ? `0${month}` : month;

    return `${formattedDay}/${formattedMonth} ${hours}:${minutes}`;

};

export const format2Num = (num) => {

    if (num) {

        const n = parseFloat(num);

        if (!isNaN(n)) {

            return parseFloat(n.toFixed(2));

        }

    }

    return null;

}

export const convertDate = (dateString) => {

    const date = new Date(dateString);

    const year = date.getFullYear();

    const month = String(date.getMonth() + 1).padStart(2, '0');

    const day = String(date.getDate()).padStart(2, '0');

    const hours = String(date.getHours()).padStart(2, '0');

    const minutes = String(date.getMinutes()).padStart(2, '0');

    return `${year}-${month}-${day}T${hours}:${minutes}`;

}

// This is your Prisma schema file,

// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?

// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "mysql"

url = "mysql://root:root@localhost:3306/bet\_db"

}

model User {

id Int @id @default(autoincrement())

name String

email String @unique

password String

role Role @relation(fields: [role\_id], references: [id])

role\_id Int

balance Float @default(0.00) @map("DECIMAL(10,2)")

is\_locked Boolean @default(false)

bets Bet[]

DepositHistory DepositHistory[]

WithdrawHistory WithdrawHistory[]

}

model Role {

id Int @id @default(autoincrement())

name String

User User[]

}

model Category {

id Int @id @default(autoincrement())

name String @unique

games Game[]

img String?

channel\_stream String?

}

model Team {

id Int @id @default(autoincrement())

name String @unique

teamGames TeamGame[]

img String?

}

model TeamGame {

id Int @id @default(autoincrement())

team Team @relation(fields: [team\_id], references: [id], onDelete: Cascade)

team\_id Int

coefficient Float @map("DECIMAL(10,2)") @default(0.00)

totalBid Float @default(0.00)

game1 Game? @relation(name: "TeamGame1")

game2 Game? @relation(name: "TeamGame2")

}

model Game {

id Int @id @default(autoincrement())

category Category @relation(fields: [category\_id], references: [id], onDelete: Cascade)

category\_id Int

teamGame1 TeamGame @relation(name: "TeamGame1", fields: [teamGame1\_id], references: [id], onDelete: Cascade)

teamGame1\_id Int @unique

teamGame2 TeamGame @relation(name: "TeamGame2", fields: [teamGame2\_id], references: [id], onDelete: Cascade)

teamGame2\_id Int @unique

type GameType

bets Bet[]

is\_active Boolean @default(true)

is\_bets\_open Boolean @default(true)

game\_start\_date DateTime

game\_finish\_date DateTime

channel\_stream String?

team\_won String?

}

enum TeamSelection {

team1

team2

}

enum GameType {

manual

auto

}

model Bet {

id Int @id @default(autoincrement())

amount Float @map("DECIMAL(10,2)") @default(0.00)

game Game @relation(fields: [game\_id], references: [id], onDelete: Cascade)

game\_id Int

selectedTeam TeamSelection

user User @relation(fields: [user\_id], references: [id], onDelete: Cascade)

user\_id Int

is\_returned Boolean @default(false)

}

model DepositHistory {

id Int @id @default(autoincrement())

amount Float @map("DECIMAL(10,2)") @default(0.00)

payment\_method String

date DateTime

user User @relation(fields: [user\_id], references: [id], onDelete: Cascade)

user\_id Int

}

model WithdrawHistory {

id Int @id @default(autoincrement())

amount Float @map("DECIMAL(10,2)") @default(0.00)

receiver String

date DateTime

user User @relation(fields: [user\_id], references: [id], onDelete: Cascade)

user\_id Int

}

import betService from '../service/betService.js';

import gameService from '../service/gameService.js';

import userService from '../service/userService.js';

const setUserBalanceById = async (req, res) => {

    const { user\_id, new\_balance } = req.body;

    userService.setUserBalanceById(user\_id, new\_balance)

        .then(user => {

            res.json(user);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const changeUserNameById = async (req, res) => {

    const { user\_id, new\_name } = req.body;

    userService.setUserNameById(user\_id, new\_name)

        .then(user => {

            res.json(user);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const changeUserEmailById = async (req, res) => {

    const { user\_id, new\_email } = req.body;

    userService.setUserEmailById(user\_id, new\_email)

        .then(user => {

            res.json(user);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const changeGameCoefficient = async (req, res) => {

    const { game\_id, selected\_team, new\_coefficient } = req.body;

    gameService.changeGameCoefficient(game\_id, selected\_team, new\_coefficient)

        .then(r => {

            res.json(r);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const changeUserLockedStatusById = async (req, res) => {

    const { user\_id, is\_locked } = req.body;

    userService.changeIsLockedByUserId(user\_id, is\_locked)

        .then(user => {

            res.json(user);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

export default {

    setUserBalanceById,

    changeUserNameById,

    changeUserEmailById,

    changeGameCoefficient,

    changeUserLockedStatusById,

};

import authService from '../service/authService.js';

const register = async (req, res) => {

    const { name, email, password } = req.body;

    authService.register(name, email, password)

        .then(user => {

            res.json(user);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const login = async (req, res) => {

    const { email, password } = req.body;

    authService.login(email, password)

        .then((response) => {

            res.status(200).json({ message: 'Success', ...response });

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const authenticateToken = async (req, res) => {

    const { token } = req.body;

    if (!token) {

        res.status(403);

        return;

    }

    const r = authService.authenticateToken(token)

        .then(r => {

            if(r) {

                res.status(200).json(r);

            }

            res.status(403);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

export default {

    register,

    login,

    authenticateToken,

};

import authService from '../service/authService.js';

import betService from '../service/betService.js';

const create = async (req, res) => {

    const authHeader = req.headers['authorization'];

    const token = authHeader && authHeader.split(' ')[1];

    const email = authService.getUserEmailFromToken(token);

    const { game\_id, selected\_team, amount } = req.body;

    betService.create(email, game\_id, selected\_team, amount)

        .then((bet) => {

            res.json(bet);

        }).catch((error) => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const getUserActiveBetsByEmail = async (req, res) => {

    const authHeader = req.headers['authorization'];

    const token = authHeader && authHeader.split(' ')[1];

    const { email = authService.getUserEmailFromToken(token), sort, order, active } = req.query;

    let formattedSort = {};

    const is\_active = active === '0' ? false : true;

    switch(sort) {

        case 'date':

            const date = is\_active ? 'game\_start\_date' : 'game\_finish\_date';

            formattedSort = {

                game: {

                    [date]: order

                }

            };

            break;

        case 'amount':

            formattedSort = {

                amount: order

            };

            break;

        case 'id':

            formattedSort = {

                id: order

            };

            break;

        default:

            formattedSort = {

                id: 'desc'

            };

            break

    };

    // betService.getActiveBetsByUserEmail(email)

    betService.getActiveBetsByUserEmail(email, formattedSort, is\_active)

        .then((bets) => {

            res.json(bets);

        }).catch((error) => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const getUserArchivedBetsByEmail = async (req, res) => {

    const authHeader = req.headers['authorization'];

    const token = authHeader && authHeader.split(' ')[1];

    // const email = authService.getUserEmailFromToken(token);

    const { email = authService.getUserEmailFromToken(token) } = req.query;

    betService.getArchivedBetsByUserEmail(email)

        .then((bets) => {

            res.json(bets);

        }).catch((error) => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const returnBet = async (req, res) => {

    const { id } = req.body;

    betService.returnBet(id)

        .then((bet) => {

            res.json(bet);

        }).catch((error) => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const returnBetsByGameId  = async (req, res) => {

    const { id } = req.body;

    betService.returnBetsByGameId(id)

        .then(bets => {

            res.json(bets);

        }).catch((error) => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

export default {

    create,

    getUserActiveBetsByEmail,

    getUserArchivedBetsByEmail,

    returnBet,

    returnBetsByGameId,

};

import categoryService from '../service/categoryService.js';

const getAll = async (req, res) => {

    try {

        const categories = await categoryService.getAll();

        res.json(categories);

    } catch (error) {

        console.log(error);

        const { statusCode = 500, message } = error;

        res.status(statusCode).json({ error: message });

    }

};

const edit = async (req, res) => {

    const { id, name, channel\_stream, image } = req.body;

    categoryService.edit(id, name, channel\_stream, image)

        .then(team => {

            res.json(team);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const create = async (req, res) => {

    const { name, channel\_stream, image } = req.body;

    categoryService.create(name, channel\_stream, image)

        .then(category => {

            res.json(category);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const deleteById = async (req, res) => {

    const { id } = req.body;

    categoryService.deleteById(id)

        .then(r => {

            res.json(r);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

export default {

    getAll,

    create,

    deleteById,

    edit,

};

import gameService from '../service/gameService.js';

const create = async (req, res) => {

    const { team\_name\_1, coefficient\_1, team\_name\_2, coefficient\_2, game\_type, game\_start\_date, game\_finish\_date, category\_name } = req.body;

    gameService.create(team\_name\_1, coefficient\_1, team\_name\_2, coefficient\_2, game\_type, game\_start\_date, game\_finish\_date, category\_name)

        .then(game => {

            res.json(game);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const getActiveGames = async (req, res) => {

    const { category, type, is\_active, sort, order } = req.query;

    let formattedSort = {};

    switch(sort) {

        case 'startDate':

            formattedSort = {

                game\_start\_date: order

            };

            break;

        case 'finishDate':

            formattedSort = {

                game\_finish\_date: order

            };

            break;

        case 'id':

            formattedSort = {

                id: order

            };

            break;

        case 'team1':

            formattedSort = {

                teamGame1: {

                    coefficient: order

                }

            };

            break;

        case 'team2':

            formattedSort = {

                teamGame2: {

                    coefficient: order

                }

            };

            break;

        default:

            formattedSort = {

                id: 'desc'

            };

            break

    };

    const active = is\_active === "1" ? true : false;

    gameService.getActiveGames(category, type, active, formattedSort)

        .then(games => {

            res.json(games);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const getArchivedGames = async (req, res) => {

    const { category, type, is\_active } = req.query;

    gameService.getArchivedGames(category, type, is\_active)

        .then(games => {

            res.json(games);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const findById = async (req, res) => {

    const { id } = req.query;

    gameService.findById(parseInt(id))

        .then(r => {

            res.json(r);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const changeBetsStatus = async (req, res) => {

    const { id, is\_bets\_open } = req.body;

    gameService.changeBetsStatus(parseInt(id), is\_bets\_open)

        .then(r => {

            res.json(r);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const updateGameInfo = async (req, res) => {

    const { id, game\_start\_date, game\_finish\_date, coefficient\_1, coefficient\_2 } = req.body;

    gameService.updateInfo(id, game\_start\_date, game\_finish\_date, coefficient\_1, coefficient\_2)

    .then(r => {

        res.json(r);

    }).catch(error => {

        console.log(error);

        const { statusCode, message } = error;

        res.status(statusCode).json({ error: message });

    });

}

const getAwaitingResultGames = async (req, res) => {

    gameService.getAwaitingResultGames()

        .then(r => {

            res.json(r);

        }).catch(error => {

            console.log(error);

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const setTeamWin = async (req, res) => {

    const { id, teamWon } = req.body;

    gameService.setTeamWin(id, teamWon)

    .then(r => {

        res.json(r);

    }).catch(error => {

        console.log(error);

        const { statusCode, message } = error;

        res.status(statusCode).json({ error: message });

    });

}

export default {

    create,

    getActiveGames,

    getArchivedGames,

    findById,

    changeBetsStatus,

    updateGameInfo,

    getAwaitingResultGames,

    setTeamWin,

};

import teamService from '../service/teamService.js';

const create = async (req, res) => {

    const { name, image } = req.body;

    console.log(image)

    teamService.create(name, image)

        .then(team => {

            res.json(team);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const edit = async (req, res) => {

    const { id, name, image } = req.body;

    teamService.edit(id, name, image)

        .then(team => {

            res.json(team);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const deleteById = async (req, res) => {

    const { id } = req.body;

    teamService.deleteById(id)

        .then(r => {

            res.json(r);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const findByName = async (req, res) => {

    const { name } = req.query;

    if(!name) {

        teamService.findAll()

        .then(teams => {

            res.json(teams);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

    } else {

        teamService.findByName(name)

            .then(team => {

                res.json(team);

            }).catch(error => {

                const { statusCode, message } = error;

                res.status(statusCode).json({ error: message });

            });

    }

};

export default {

    create,

    findByName,

    deleteById,

    edit,

};

import userService from '../service/userService.js';

import authService from '../service/authService.js';

import PDFDocument from 'pdfkit';

import blobStream from 'blob-stream';

const getAllUsers = async (req, res) => {

    userService.getAllUsers()

        .then(users => {

            res.json(users);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const getUserById = async (req, res) => {

    const { id } = req.query;

    userService.getUserById(id)

        .then(user => {

            res.json(user);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const banSelfUser = async (req, res) => {

    const authHeader = req.headers['authorization'];

    const token = authHeader && authHeader.split(' ')[1];

    const email = authService.getUserEmailFromToken(token);

    const user = await userService.getUserByEmail(email);

    userService.changeIsLockedByUserId(user.id, true)

        .then(users => {

            res.json(users);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const banUserById = async (req, res) => {

    const { id } = req.body;

    userService.changeIsLockedByUserId(id, true)

    .then(r => {

        res.json(r);

    }).catch(error => {

        const { statusCode, message } = error;

        res.status(statusCode).json({ error: message });

    });

};

const unBanUserById = async (req, res) => {

    const { id } = req.body;

    userService.changeIsLockedByUserId(id, false)

    .then(r => {

        res.json(r);

    }).catch(error => {

        const { statusCode, message } = error;

        res.status(statusCode).json({ error: message });

    });

};

const update = async (req, res) => {

    const { id, name, balance } = req.body;

    userService.update(id, name, balance)

        .then(user => {

            res.json(user);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

};

const changePassword = async (req, res) => {

    const { old\_password, new\_password } = req.body;

    const authHeader = req.headers['authorization'];

    const token = authHeader && authHeader.split(' ')[1];

    const email = authService.getUserEmailFromToken(token);

    const user = await userService.getUserByEmail(email);

    userService.updateUserPassword(user.id, old\_password, new\_password)

        .then(r => {

            res.json(r);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const deposit = async (req, res) => {

    const { id, amount, payment\_method } = req.body;

    userService.deposit(id, amount, payment\_method)

        .then(r => {

            const doc = new PDFDocument();

            let buffers = [];

            doc.on('data', buffers.push.bind(buffers));

            doc.on('end', () => {

                let pdfData = Buffer.concat(buffers);

                res.setHeader('Content-Disposition', 'attachment; filename="file.pdf"');

                res.setHeader('Content-Type', 'application/pdf');

                res.setHeader('Content-Length', pdfData.length);

                res.send(pdfData);

            });

            // add your content to the document here, as usual

            doc

            .fontSize(18)

            .text(`Check number: ${r.depositHistory\_id}`)

            .text(`Deposit amount: ${parseFloat(amount).toFixed(2)}€`)

            .text(`Payment method: ${payment\_method}`)

            .text(`Customer email: ${r.email}`)

            .text(`Customer name: ${r.name}`)

            .text(`New balance: ${r.balance.toFixed(2)}€`)

            .text(`Date: ${new Date()}`)

            .moveDown()

            .text('Company name: eWager', { align: 'right' })

            .text('Company bank account number: NJ1238891KS8952', { align: 'right' })

            .text('Company registration number: LV123912323214', { align: 'right' });

            // finalize the PDF and end the stream

            doc.end();

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode || 500).json({ error: message });

        });

}

const withdraw = async (req, res) => {

    const { id, amount, receiver } = req.body;

    userService.withdraw(id, amount, receiver)

        .then(r => {

            res.json(r);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

const getTransactionsHistory = async (req, res) => {

    const { id } = req.query;

    userService.getTransactionsHistory(id)

        .then(r => {

            res.json(r);

        }).catch(error => {

            const { statusCode, message } = error;

            res.status(statusCode).json({ error: message });

        });

}

export default {

    getAllUsers,

    banSelfUser,

    getUserById,

    update,

    changePassword,

    deposit,

    withdraw,

    getTransactionsHistory,

    banUserById,

    unBanUserById,

};

import jwt from 'jsonwebtoken';

import userService from './../service/userService.js';

const jwtSecret = process.env.JWT\_SECRET;

function \_getTokenFromHeaders(headers) {

  const authHeader = headers['authorization'];

  const token = authHeader && authHeader.split(' ')[1];

  return token;

};

export function authenticateToken(req, res, next) {

  const token = \_getTokenFromHeaders(req.headers);

  if (token == null) return res.sendStatus(401)

  jwt.verify(token, jwtSecret, async (error, data) => {

    // console.log(err)

    if (error) {

      console.log(error);

      return res.sendStatus(403)

    }

    const user = await userService.getUserByEmail(data.email);

    if(user.is\_locked) {

      return res.status(403).json({ error: 'User is locked' });

    }

    req.user = user;

    next()

  });

};

export function hasRole(requiredRoles) {

  return function(req, res, next) {

    const token = \_getTokenFromHeaders(req.headers);

    const { email } = jwt.decode(token);

    userService.getUserByEmail(email)

      .then((user) => {

        if(!user) {

          return res.sendStatus(403);

        }

        const { role } = user;

        const { name: roleName } = role;

        if (!requiredRoles.includes(roleName)) {

          return res.status(403).json({ error: 'User has no access' });

        }

        next();

      }).catch((error) => {

        console.log(error);

        const { statusCode, message } = error;

        return res.sendStatus(statusCode);

      });

  }

};

export default {

  authenticateToken,

  hasRole,

};

import bcrypt from 'bcrypt';

import { Prisma, PrismaClient } from '@prisma/client';

import { SALT\_ROUNDS } from '../config/authConfig.js';

import 'dotenv/config';

import jwt from 'jsonwebtoken';

import userService from './userService.js';

const prisma = new PrismaClient();

const jwtSecret = process.env.JWT\_SECRET;

const errors = {

    'unknown': { statusCode: 500, message: 'Internal Server Error' },

    'userAlreadyExists': { statusCode: 409, message: 'User with this email already exists' },

    'userNotFound': { statusCode: 404, message: 'User not found' },

    'userBanned': { statusCode: 403, message: 'User banned' },

    'incorrectPassword': { statusCode: 400, message: 'Incorrect password' },

};

const register = async (name, email, password) => {

    try {

        const hashedPassword = await bcrypt.hash(password, SALT\_ROUNDS);

        const userRoleName = 'user';

        const userRole = await prisma.role.findFirst({

            where: {

                name: userRoleName

            }

        });

        if (!userRole) {

            throw errors.unknown;

        }

        const user = await prisma.user.create({

            data: {

                name,

                email,

                password: hashedPassword,

                role\_id: userRole.id

            }

        });

        return user;

    } catch (error) {

        if (

            error instanceof Prisma.PrismaClientKnownRequestError && error.code === 'P2002'

        ) {

            throw errors.userAlreadyExists;

        } else {

            throw errors.unknown;

        }

    }

};

const login = async (email, password) => {

    const user = await prisma.user.findUnique({

        where: {

            email

        },

        include: {

            role: true,

        }

    });

    if (!user) {

        throw errors.userNotFound;

    }

    if (user.is\_locked) {

        throw errors.userBanned;

    }

    const isPasswordMatch = await bcrypt.compare(password, user.password);

    if (!isPasswordMatch) {

        throw errors.incorrectPassword;

    }

    const jwtToken = \_generateJwtToken(email);

    return {

        balance: user.balance,

        token: jwtToken,

        id: user.id,

        role: user.role.name,

    };

};

const authenticateToken = (token) => {

    try {

        // jwt.verify(token, jwtSecret);

        return jwt.verify(token, jwtSecret, async (error, data) => {

            if (error) {

              console.log(error);

              return null;

            }

            const user = await userService.getUserByEmail(data.email);

            if(user.is\_locked) {

                return null;

            }

            return user;

          });

    } catch (error) {

        return null;

    }

}

const getUserEmailFromToken = (token) => {

    const { email } = jwt.decode(token);

    return email;

}

const \_generateJwtToken = (email) => {

    return jwt.sign({ email }, jwtSecret, { expiresIn: "30d" });

};

export default {

    register,

    login,

    getUserEmailFromToken,

    authenticateToken,

};

import { Prisma, PrismaClient, TeamSelection } from '@prisma/client';

import userService from './userService.js';

import gameService from './gameService.js';

const prisma = new PrismaClient();

const errors = {

    'unknown': { statusCode: 500, message: 'Internal Server Error' },

    'noBalance': { statusCode: 402, message: 'Insufficient Funds' },

    'gameNotFound': { statusCode: 404, message: 'Game Not Found' },

    'betsClosed': { statusCode: 400, message: 'Bets Is Closed' },

    'userNotFound': { statusCode: 404, message: 'User Not Found' },

};

const create = async (userEmail, gameId, selectedTeam,  amount) => {

    try {

        await gameService.updateGamesStatusByDate();

        const game = await gameService.findById(gameId);

        if (!game.is\_bets\_open) {

            throw errors.betsClosed;

        }

        const user = await userService.getUserByEmail(userEmail);

        if (amount > user.balance) {

            throw errors.noBalance;

        }

        const teamSelection = selectedTeam === TeamSelection.team1 ? TeamSelection.team1 : TeamSelection.team2;

        const bet = await prisma.bet.create({

            data: {

                amount,

                selectedTeam: teamSelection,

                game: {

                    connect: { id: gameId }

                },

                user: {

                    connect: { id: user.id }

                },

            },

            include: {

                game: {

                    include: {

                        teamGame1: true,

                        teamGame2: true,

                    },

                },

            },

        });

        const updatedUser = await userService.reduceBalanceById(user.id, amount);

        const teamGameId1 = bet.game.teamGame1\_id;

        const teamGameId2 = bet.game.teamGame2\_id;

        const coefficients = \_adjustOddsDynamically(

            bet.game.teamGame1.coefficient,

            bet.game.teamGame2.coefficient,

            bet.game.teamGame1.totalBid + (teamSelection === TeamSelection.team1 ? amount : 0),

            bet.game.teamGame2.totalBid + (teamSelection === TeamSelection.team2 ? amount : 0)

        );

        if(teamSelection === TeamSelection.team1) {

            await prisma.teamGame.update({

                where: { id: teamGameId1 },

                data: {

                    totalBid: {

                        increment: amount,

                    },

                    coefficient: parseFloat(coefficients.coefficient1),

                },

            });

            await prisma.teamGame.update({

                where: { id: teamGameId2 },

                data: {

                   coefficient: parseFloat(coefficients.coefficient2),

                },

            });

        } else {

            await prisma.teamGame.update({

                where: { id: teamGameId2 },

                data: {

                    totalBid: {

                        increment: amount,

                    },

                    coefficient: parseFloat(coefficients.coefficient2),

                },

            });

            await prisma.teamGame.update({

                where: { id: teamGameId1 },

                data: {

                    coefficient: parseFloat(coefficients.coefficient1),

                },

            });

        }

        const newBet = await prisma.bet.findFirst({

            where: {

                id: bet.id

            },

            include: {

                game: {

                    include: {

                        teamGame1: true,

                        teamGame2: true,

                    },

                },

                user: true,

            },

        })

        return {

            ...newBet,

            new\_balance: updatedUser.balance,

        };

    } catch (error) {

        console.log(error);

        if(error.statusCode) {

            throw error;

        }

        throw errors.unknown;

    }

};

function \_adjustOddsDynamically(coefTeam1, coefTeam2, totalBidTeam1, totalBidTeam2) {

    const totalBid = totalBidTeam1 + totalBidTeam2;

    const shareTeam1 = totalBidTeam1 / totalBid;

    const shareTeam2 = totalBidTeam2 / totalBid;

    const baseChange = Math.abs(shareTeam1 - shareTeam2);

    let newCoefTeam1 = coefTeam1 + baseChange \* (shareTeam2 > shareTeam1 ? 1 : -1);

    let newCoefTeam2 = coefTeam2 + baseChange \* (shareTeam1 > shareTeam2 ? 1 : -1);

    const minCoef = 1.01;

    const maxCoef = 100;

    newCoefTeam1 = Math.max(minCoef, Math.min(maxCoef, newCoefTeam1));

    newCoefTeam2 = Math.max(minCoef, Math.min(maxCoef, newCoefTeam2));

    return {

        coefficient1: newCoefTeam1.toFixed(3),

        coefficient2: newCoefTeam2.toFixed(3)

    };

}

const getActiveBetsByUserEmail = async (email, sort, is\_active = true) => {

    const user = await prisma.user.findUnique({

        where: {

            email: email

        }

    });

    const activeBets = await prisma.bet.findMany({

        where: {

            user\_id: user.id,

            game: {

                is\_active,

            },

        },

        include: {

            game: {

            include: {

                teamGame1: {

                    include: {

                        team: true,

                    }

                },

                teamGame2: {

                    include: {

                        team: true,

                    }

                },

            }

            }

        },

        orderBy: sort

    });

    return activeBets;

};

const getArchivedBetsByUserEmail = async (email) => {

    await gameService.updateGamesStatusByDate();

    const user = await prisma.user.findUnique({

        where: { email },

        include: {

          bets: {

            where: {

              game: {

                is\_active: false,

              },

            },

            include: {

              game: {

                include: {

                    teamGame1: {

                        include: {

                            team: true,

                        }

                    },

                    teamGame2: {

                        include: {

                            team: true,

                        }

                    },

                }

              }

            },

          },

        },

      });

      if (!user) {

        throw errors.userNotFound;

      }

      const activeBets = user.bets.filter((bet) => bet.game !== null);

      return activeBets;

}

const returnBet = async (id) => {

    const bet = await prisma.bet.findFirst({

        where: {

            id,

        },

        include: {

            user: true,

        },

    });

    await prisma.user.update({

        where: {

            id: bet.user\_id,

        },

        data: {

            balance: {

                increment: bet.amount,

            },

        },

    });

    return await prisma.bet.update({

        where: {

            id,

        },

        data: {

            is\_returned: true,

        },

    });

}

const returnBetsByGameId = async (id) => {

    const game = await prisma.game.findFirst({

        where: {

            id,

        },

        include: {

            bets: {

                include: {

                    user: true,

                },

            },

        },

    });

    const returnedBetsPromises = game.bets.map(async (bet) => {

        await prisma.user.update({

            where: {

                id: bet.user.id,

            },

            data: {

                balance: {

                    increment: bet.amount,

                },

            },

        });

        return prisma.bet.update({

            where: {

                id: bet.id,

            },

            data: {

                is\_returned: true,

            },

        });

    });

    return Promise.all(returnedBetsPromises);

}

export default {

    create,

    getActiveBetsByUserEmail,

    getArchivedBetsByUserEmail,

    returnBet,

    returnBetsByGameId,

};

import { Prisma, PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

const errors = {

    'unknown': { statusCode: 500, message: 'Internal Server Error' },

    'alreadyExists': { statusCode: 400, message: 'Category Already Exists' },

};

const create = async (name, channel\_stream, image) => {

    try {

        const category = await prisma.category.create({

            data: {

                name,

                channel\_stream,

                img: image,

            }

        });

        return category;

    } catch (error) {

        console.log(error)

        if (error.statusCode) {

            throw error;

        }

        if (error instanceof Prisma.PrismaClientKnownRequestError && error.code === 'P2002') {

            throw errors.alreadyExists;

        } else {

            throw errors.unknown;

        }

    }

};

const deleteById = async (id) => {

    try {

        await prisma.category.delete({

            where: {

                id

            },

        });

        return true;

    } catch (error) {

        throw errors.unknown;

    }

}

const getAll = async () => {

    try {

        const categories = await prisma.category.findMany();

        return categories;

    } catch (error) {

        console.log(error);

        throw errors.unknown;

    }

}

const findByName = async (name) => {

    try {

        const category = await prisma.category.findFirst({

            where: {

                name,

            },

        });

        return category;

    } catch (error) {

        console.log(error);

        throw errors.unknown;

    }

}

const edit = async (id, name, channel\_stream, image) => {

    try {

        const data = {

            name,

            channel\_stream,

            ...(image !== null && { img: image })

        };

        const team = await prisma.category.update({

            where: {

                id

            },

            data

        });

        return team;

    } catch (error) {

        console.log(error)

        throw errors.unknown;

    }

}

export default {

    create,

    getAll,

    findByName,

    deleteById,

    edit,

};

import { Prisma, PrismaClient } from '@prisma/client';

import teamService from './teamService.js';

import categoryService from './categoryService.js';

const prisma = new PrismaClient();

const errors = {

    'unknown': { statusCode: 500, message: 'Internal Server Error' },

    'alreadyExists': { statusCode: 400, message: 'Game Already Exists' },

    'team1NotFound': { statusCode: 404, message: 'Team1 Not Found' },

    'team2NotFound': { statusCode: 404, message: 'Team2 Not Found' },

    'categoryNotFound': { statusCode: 404, message: 'Category Not Found' },

};

const create = async (teamName1, coefficient1, teamName2, coefficient2, gameType, gameStartDate, gameFinishDate, categoryName) => {

    try {

        const category = await categoryService.findByName(categoryName);

        if (!category) {

            throw errors.categoryNotFound;

        }

        const team1 = await teamService.findByName(teamName1);

        if (!team1) {

            throw errors.team1NotFound;

        }

        const team2 = await teamService.findByName(teamName2);

        if(!team2) {

            throw errors.team2NotFound;

        }

        const teamGame1 = await \_createTeamGame(team1.id, coefficient1);

        const teamGame2 = await \_createTeamGame(team2.id, coefficient2);

        const game = await prisma.game.create({

            data: {

                teamGame1\_id: teamGame1.id,

                teamGame2\_id: teamGame2.id,

                type: gameType,

                game\_start\_date: new Date(gameStartDate),

                game\_finish\_date: new Date(gameFinishDate),

                category\_id: category.id,

            },

            include: {

                category: true,

                teamGame1: {

                    include: {

                        team: true,

                    },

                },

                teamGame2: {

                    include: {

                        team: true,

                    },

                },

            },

        });

        const formattedGame = \_formatGameCreateResponse(game);

        return formattedGame;

    } catch (error) {

        console.log(error);

        if (error.statusCode) {

            throw error;

        }

        if (error instanceof Prisma.PrismaClientKnownRequestError && error.code === 'P2002') {

            throw errors.alreadyExists;

        } else {

            throw errors.unknown;

        }

    }

};

const findById = async (id) => {

    const game = await prisma.game.findFirst({

        where: {

            id

        },

        include: {

            category: true,

            teamGame1: {

                include: {

                    team: true,

                },

            },

            teamGame2: {

                include: {

                    team: true,

                },

            },

            bets: {

                include: {

                    user: true,

                },

            },

        },

    });

    const betsTeam1 = game.bets.filter(bet => bet.selectedTeam === 'team1');

    const betsTeam2 = game.bets.filter(bet => bet.selectedTeam === 'team2');

    const totalTeam1Amount = betsTeam1.reduce((acc, bet) => acc + bet.amount, 0);

    const totalTeam2Amount = betsTeam2.reduce((acc, bet) => acc + bet.amount, 0);

    game.betsTeam1 = betsTeam1;

    game.betsTeam2 = betsTeam2;

    game.totalTeam1Amount = totalTeam1Amount;

    game.totalTeam2Amount = totalTeam2Amount;

    game.totalAmount = totalTeam1Amount + totalTeam2Amount;

    // const totalAmount = game.bets.reduce((acc, bet) => acc + bet.amount, 0);

    // const totalTeam1Amount = game.bets

    //     .filter(bet => bet.selectedTeam === 'team1')

    //     .reduce((acc, bet) => acc + bet.amount, 0);

    // const totalTeam2Amount = game.bets

    //     .filter(bet => bet.selectedTeam === 'team2')

    //     .reduce((acc, bet) => acc + bet.amount, 0);

    // game.totalAmount = totalAmount;

    // game.totalTeam1Amount = totalTeam1Amount;

    // game.totalTeam2Amount = totalTeam2Amount;

    return game;

};

const \_formatGameCreateResponse = (game) => {

    const { teamGame1, teamGame2 } = game;

    const formattedGame = {

        id: game.id,

        type: game.type,

        teamGame1: {

            id: teamGame1.id,

            coefficient: teamGame1.coefficient,

            team: teamGame1.team,

            totalBid: teamGame1.totalBid,

        },

        teamGame2: {

            id: teamGame2.id,

            coefficient: teamGame2.coefficient,

            team: teamGame2.team,

            totalBid: teamGame2.totalBid,

        },

    };

    return formattedGame;

};

const \_formatDate = (date) => {

    return new Date(date).toISOString;

}

const changeGameCoefficient = async (gameId, selectedTeam, newCoefficient) => {

    const game = await prisma.game.findFirst({

        where: {

            id: gameId,

        },

    });

    if (!game) {

        throw errors.gameNotFound;

    }

    let teamGameId = {};

    if (selectedTeam === TeamSelection.team1) {

        const teamGame = await prisma.teamGame.findFirst({

            where: {

                id: game.teamGame1\_id,

            },

        });

        teamGameId = teamGame.id;

    } else {

        const teamGame = await prisma.teamGame.findFirst({

            where: {

                id: game.teamGame2\_id,

            },

        });

        teamGameId = teamGame.id;

    }

    const updatedTeamGame = await prisma.teamGame.update({

        where: {

            id: teamGameId,

        },

        data: {

            coefficient: newCoefficient,

        }

    });

    return updatedTeamGame;

};

const updateGamesStatusByDate = async () => {

    const currentDate = new Date();

    const games = await prisma.game.findMany({

        where: {

            game\_finish\_date: {

                lt: currentDate,

            },

            is\_active: true,

        },

        include: {

            teamGame1: true,

            teamGame2: true,

        }

    });

    await prisma.game.updateMany({

        where: {

            game\_finish\_date: {

                lt: currentDate,

            },

        },

        data: {

            is\_active: false

        }

    });

    return await \_closeBetsOnGameStartDate();

};

const \_closeBetsOnGameStartDate = async () => {

    const currentDate = new Date();

    const games = await prisma.game.updateMany({

        where: {

            game\_start\_date: {

                lt: currentDate,

            },

            is\_bets\_open: true,

        },

        data: {

            is\_bets\_open: false

        }

    });

    return games;

};

const getActiveGames = async (name, type, is\_active = true, sort) => {

    updateGamesStatusByDate();

    if(name === 'all') {

        return await prisma.game.findMany({

            where: {

                is\_active,

                type,

            },

            include: {

                category: true,

                teamGame1: {

                    include: {

                        team: true,

                    },

                },

                teamGame2: {

                    include: {

                        team: true,

                    },

                },

            },

            orderBy: sort,

        });

    }

    return await prisma.game.findMany({

        where: {

            is\_active,

            category: {

                name: name,

            },

            type,

        },

        include: {

            category: true,

            teamGame1: {

                include: {

                    team: true,

                },

            },

            teamGame2: {

                include: {

                    team: true,

                },

            },

        },

        orderBy: sort,

    });

};

const getArchivedGames = async (name, type) => {

    updateGamesStatusByDate();

    const games = await prisma.game.findMany({

        where: {

            is\_active: false,

        },

        include: {

            category: true,

            teamGame1: {

                include: {

                    team: true,

                },

            },

            teamGame2: {

                include: {

                    team: true,

                },

            },

        },

    });

    return games;

}

const \_createTeamGame = async (teamId, coefficient) => {

    const teamGame = await prisma.teamGame.create({

        data: {

            team\_id: teamId,

            coefficient: coefficient,

        }

    });

    return teamGame;

};

const changeBetsStatus = async (id, is\_bets\_open) => {

    const game = await prisma.game.update({

        where: {

            id,

        },

        data: {

            is\_bets\_open: is\_bets\_open,

        },

    });

    return game;

};

const updateInfo = async(id, gameStartDate, gameFinishDate, coefficient1, coefficient2) => {

    const game = await prisma.game.findFirst({

        where: {

            id,

        },

        include: {

            teamGame1: true,

            teamGame2: true,

        }

    });

    await prisma.teamGame.update({

        where: {

            id: game.teamGame1.id,

        },

        data: {

            coefficient: coefficient1,

        },

    });

    await prisma.teamGame.update({

        where: {

            id: game.teamGame2.id,

        },

        data: {

            coefficient: coefficient2,

        },

    });

    const updatedGame = await prisma.game.update({

        where: {

            id,

        },

        data: {

            game\_start\_date: new Date(gameStartDate),

            game\_finish\_date: new Date(gameFinishDate),

        },

    });

    return updatedGame;

}

const getAwaitingResultGames = async () => {

    const games = await prisma.game.findMany({

        where: {

            is\_active: false,

            team\_won: null,

        },

        include: {

            category: true,

            teamGame1: {

                include: {

                    team: true,

                }

            },

            teamGame2: {

                include: {

                    team: true,

                }

            },

        }

    });

    return games;

}

const setTeamWin = async (id, teamWon) => {

    const game = await prisma.game.update({

        where: {

            id: id,

        },

        data: {

            team\_won: teamWon,

        },

    });

    const bets = await prisma.bet.findMany({

        where: {

            game\_id: game.id

        },

        include: {

            game: {

                include: {

                    teamGame1: true,

                    teamGame2: true,

                },

            },

        },

    });

    bets.map(async bet => {

        if (bet.selectedTeam === bet.game.team\_won) {

            const coef = bet.selectedTeam === 'team1' ? bet.game.teamGame1.coefficient : bet.game.teamGame2.coefficient;

            const win = bet.amount \* coef;

            await prisma.user.update({

                where: {

                    id: bet.user\_id,

                },

                data: {

                    balance: {

                        increment: win,

                    },

                },

            });

        }

    });

}

export default {

    create,

    findById,

    changeGameCoefficient,

    getActiveGames,

    updateGamesStatusByDate, // NO ROUTE

    getArchivedGames,

    changeBetsStatus,

    updateInfo,

    getAwaitingResultGames,

    setTeamWin,

};

import { Prisma, PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

const errors = {

    'unknown': { statusCode: 500, message: 'Internal Server Error' },

    'alreadyExists': { statusCode: 400, message: 'Team Already Exists' },

};

const create = async (name, image) => {

    try {

        const team = await prisma.team.create({

            data: {

                name,

                img: image

            }

        });

        return team;

    } catch (error) {

        console.log(error)

        if (error.statusCode) {

            throw error;

        }

        if (error instanceof Prisma.PrismaClientKnownRequestError && error.code === 'P2002') {

            throw errors.alreadyExists;

        } else {

            throw errors.unknown;

        }

    }

};

const edit = async (id, name, image) => {

    try {

        const data = {

            name,

            ...(image !== null && { img: image })

        };

        const team = await prisma.team.update({

            where: {

                id

            },

            data

        });

        return team;

    } catch (error) {

        console.log(error)

        throw errors.unknown;

    }

};

const deleteById = async (id) => {

    try {

        await prisma.team.delete({

            where: {

                id

            },

        });

        return true;

    } catch (error) {

        console.log(error)

        throw errors.unknown;

    }

}

const findByName = async (name) => {

    try {

        const team = await prisma.team.findFirst({

            where: {

                name

            }

        });

        return team;

    } catch (error) {

        throw errors.unknown;

    }

};

const findAll = async () => {

    try {

        const team = await prisma.team.findMany();

        return team;

    } catch (error) {

        throw errors.unknown;

    }

};

export default {

    create,

    findByName,

    findAll,

    deleteById,

    edit,

};

import { Prisma, PrismaClient } from '@prisma/client';

import bcrypt from 'bcrypt';

import { SALT\_ROUNDS } from '../config/authConfig.js';

const prisma = new PrismaClient();

const errors = {

    'unknown': { statusCode: 500, message: 'Internal Server Error' },

    'userNotFound': { statusCode: 404, message: 'User Not Found' },

    'incorrectPassword': { statusCode: 401, message: 'Incorrect password' },

    'insufficientBalance': { statusCode: 400, message: 'Insufficient balance' },

};

const getAllUsers = async () => {

    try {

        const users = await prisma.user.findMany({

            include: {

                role: true,

            },

        });

        return users;

    } catch (error) {

        console.log(error);

        if(error.statusCode) {

            throw error;

        }

        throw errors.unknown;

    }

};

const getUserByEmail = async (email) => {

    try {

        const user = await prisma.user.findFirst({

            where: {

                email,

            },

            include: {

                role: true,

            },

        });

        return user;

    } catch (error) {

        throw errors.unknown;

    }

};

const getUserById = async (id) => {

    try {

        const user = await prisma.user.findFirst({

            where: {

                id: parseInt(id),

            },

            include: {

                role: true,

            },

        });

        return user;

    } catch (error) {

        console.log(error);

        throw errors.unknown;

    }

};

const reduceBalanceById = async (id, amount) => {

    try {

        const user = await prisma.user.update({

            where: {

                id,

            },

            data: {

                balance: {

                    decrement: amount,

                },

            },

        });

        return user;

    } catch (error) {

        throw errors.unknown;

    }

};

const setUserBalanceById = async (id, newBalance) => {

    try {

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                balance: newBalance,

            },

        });

        return updatedUser;

    } catch (error) {

        throw errors.userNotFound;

    }

};

const setUserNameById = async (id, newName) => {

    try {

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                name: newName,

            },

        });

        return updatedUser;

    } catch (error) {

        console.log(error);

        throw errors.userNotFound;

    }

};

const setUserEmailById = async (id, newEmail) => {

    try {

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                email: newEmail,

            },

        });

        return updatedUser;

    } catch (error) {

        console.log(error);

        throw errors.userNotFound;

    }

};

const changeIsLockedByUserId = async (id, status) => {

    try {

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                is\_locked: status,

            },

        });

        return updatedUser;

    } catch (error) {

        console.log(error);

        throw errors.userNotFound;

    }

}

const update = async (id, name, balance) => {

    try {

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                name,

                balance,

            },

        });

        return updatedUser;

    } catch (error) {

        console.log(error);

        throw errors.userNotFound;

    }

}

const updateUserPassword = async (userId, oldPassword, newPassword) => {

    try {

        const user = await prisma.user.findUnique({

            where: {

                id: userId

            }

        });

        if (!user) {

            throw errors.unknown;

        }

        const isPasswordMatch = await bcrypt.compare(oldPassword, user.password);

        console.log(isPasswordMatch)

        if (!isPasswordMatch) {

            throw errors.incorrectPassword;

        }

        const hashedNewPassword = await bcrypt.hash(newPassword, SALT\_ROUNDS);

        await prisma.user.update({

            where: {

                id: userId

            },

            data: {

                password: hashedNewPassword

            }

        });

        return true;

    } catch (error) {

        throw errors.incorrectPassword;

    }

};

const deposit = async(id, amount, paymentMethod) => {

    try {

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                balance: {

                    increment: parseFloat(amount)

                }

            },

        });

        const depositHistory = await prisma.depositHistory.create({

            data: {

                user\_id: updatedUser.id,

                amount: parseFloat(amount),

                payment\_method: paymentMethod,

                date: new Date(),

            },

        })

        return {

            ...updatedUser,

            depositHistory\_id: depositHistory.id,

        };

    } catch (error) {

        console.log(error);

        throw errors.unknown;

    }

}

const withdraw = async(id, amount, receiver) => {

    try {

        const user = await prisma.user.findUnique({

            where: {

                id: id,

            },

        });

        if (!user) {

            throw errors.userNotFound;

        }

        if (user.balance < amount) {

            throw errors.insufficientBalance;

        }

        const updatedUser = await prisma.user.update({

            where: {

                id,

            },

            data: {

                balance: {

                    decrement: parseFloat(amount)

                }

            },

        });

        await prisma.withdrawHistory.create({

            data: {

                user\_id: updatedUser.id,

                amount: parseFloat(amount),

                receiver: receiver,

                date: new Date(),

            },

        });

        return updatedUser;

    } catch (error) {

        console.log(error);

        throw error;

    }

}

async function getTransactionsHistory(userId) {

    // Assuming you have a PrismaClient instance named prisma

    try {

        const depositHistory = await prisma.depositHistory.findMany({

            where: {

                user\_id: parseInt(userId)

            },

            orderBy: {

                date: 'desc'

            }

        });

        const withdrawHistory = await prisma.withdrawHistory.findMany({

            where: {

                user\_id: parseInt(userId)

            },

            orderBy: {

                date: 'desc'

            }

        });

        // Merge and sort deposit and withdraw history

        const transactionsHistory = [...depositHistory, ...withdrawHistory];

        transactionsHistory.sort((a, b) => b.date - a.date);

        return transactionsHistory;

    } catch (err) {

        console.error(err);

        return [];

    }

}

export default {

    getAllUsers,

    getUserByEmail,

    reduceBalanceById,

    getUserById,

    setUserBalanceById,

    setUserNameById,

    setUserEmailById,

    changeIsLockedByUserId,

    update,

    updateUserPassword,

    deposit,

    withdraw,

    getTransactionsHistory,

};