A picture containing logo

Description automatically generated**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA**

**INFORMATIEI**

**Documentatie Proiect**

**Inginerie Software**

**Studenti: Iamnitchi Bogdan, Anghel Radu, Varvari Alexandru**

**Data: 29.12.2023**

**Grupa: 30235**

1. **Despre Aplicatie**

Ca idee generala vrem sa creeam o versiune de Kahoot (vezi [aici](https://kahoot.com/)) dar orientata pe fotbal.

Mai exact o sa existe mai multe camere identificate de un cod unic in care se pot aduna mai multi utilzatori, unde se vor derula mai multe intrebari din fotbal. Care raspunde cel mai rapid dar si corect la toate intrebarile o sa fie castigatorul acelei runde.

Vrem sa avem o pagina de home pe care fiecare poate intra (ca si pe Kahoot) iar acolo utilizatorul sa gaseasca o scurta prezentare a jocului. Daca utilizatorul doreste sa joace, dar nu are un cont va fi redirectionat catre pagina de sign in respectiv sign up.

Odata logat va fi redirectionat catre o noua pagina in care acesta va putea selecta doua mari optiuni, mai precis pentru a **creea o camera** de joc sau a da pur si simplu **join la o camera**. Cand se alege crearea unei camere utilizatorul devine **host** ceea ce îi va pemite sa aiba mai multe setari precum ar fi numarul maxim de jucatori timpul pentru o intrebare si multe altele. Daca se alege optiunea de join atunci utilizatorului i se va cere un cod unic care identifica aceasta camera. Host-ul este singurul care poate vedea acest cod si da share mai departe celorlalti participanti. Desigur acest cod va fi disponibil si in URL-ul camerei.

Host-ul camerei poate da drumul jocului si poate juca si el inclusiv, fiecare va avea un timp in care sa raspunda la intrebari, si va putea vedea la final cate puncte a acumulat.

Host-ul camerei mai poate si creea proprile intrebari care vor fi vizibile dupa ce se da start si celorlalti participanti.

**Ce am reusit sa facem?**

Orice aplicatie necesita un sistem de login, si am facut multi ani login-uri diferite dar care erau foarte simple si nu se puteau nicidecum lansa in productie. Asa ca ne am gandit sa vedem cum e sa implementam un sistem de login la nivel de productie, chiar cum gasim la numeroase aplicati precum Facbook, Instagram TikTok in care te poti loga si folosind alta aplicatii social.

Am implementat si o sectiune de chat pt utilizatori aplicatiei, inainte ca sa intri in chat ca un safety feature am setata sa se mai ceara parola inca odata utilizatorului. Dupa aceea userul isi poate vedea conversatiile pe care le a facut cat si istoricul mesajelor. Desigur se pot creea si grupuri de mai multe persoane asfel putand fi foarte usor distribuit codul camerei in care se va juca. Se pot da chiar si sugesti de intrebari astfel incat host-ul sa le adauge in camera prin acest intermediu.

Pentru a implementa acest chat system am folosit Chat Engine (vezi [aici](https://chatengine.io/))

Am reusit sa implementez in mare parte tot ce mi-am propus si ce am descris mai sus la nivelul de management al camerelor. Asfel incat ficare poate creea sau da join la o camera. Odata ce ai creat o camera esti hostul ei si ai mai multe optiuni printre care sa faci update la setarile initiale ale camerei precum si a creea intrebarile care vor fi afisate odata ce se da start la camera. Toate acestea sunt insotite de pop-uri care iti vor spune daca ce ai facut este bine sau nu.

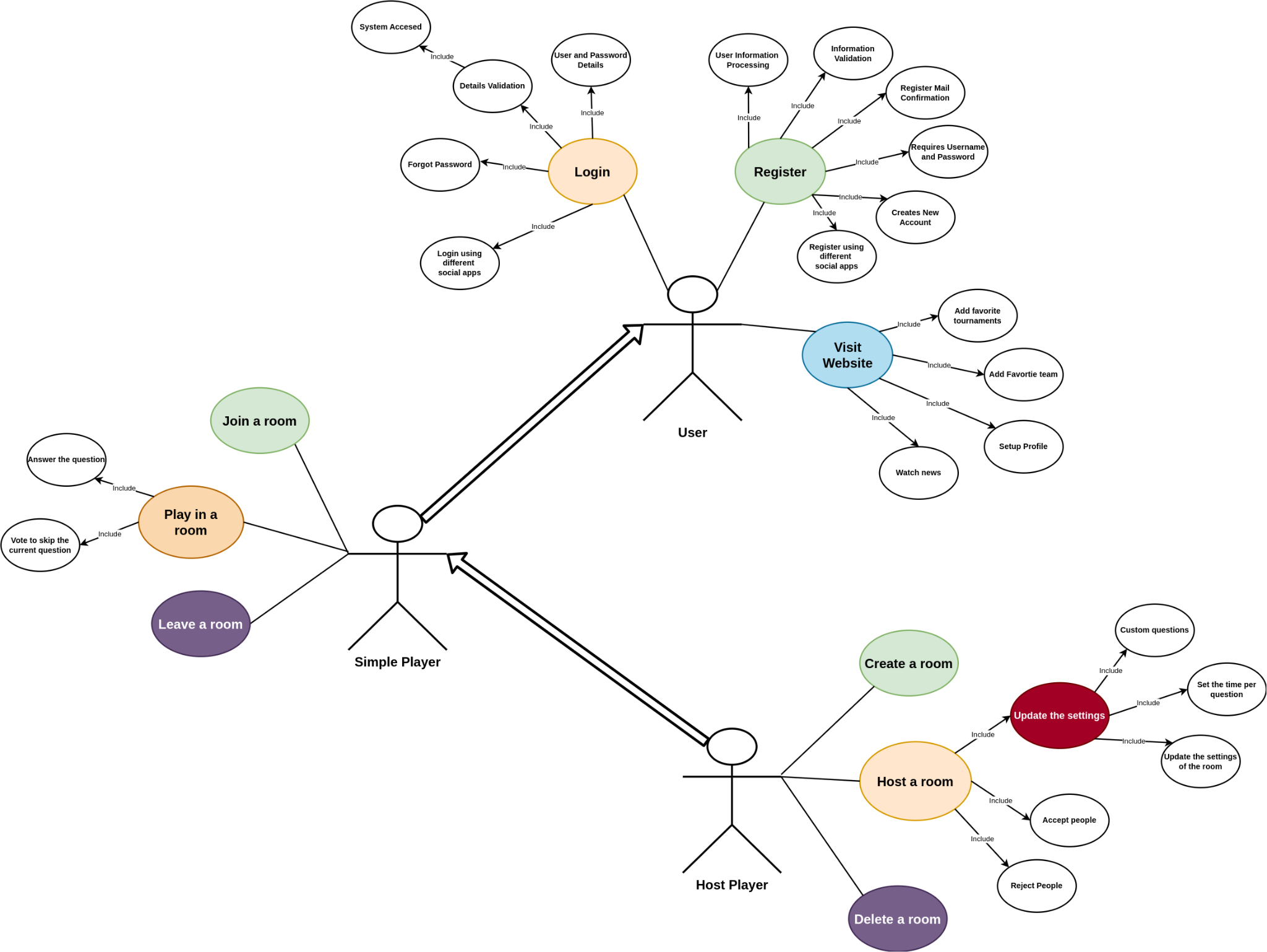
Odata ce camera (quizz-ul) a inceput backend-ul va furniza intreabri pentru fiecare din camera, fiecare utilizator atat cat si host-ul poate raspunde la intrebari si la final-ul suitei de intrebari va aparea o pagina in care vei afla punctajul final si ce poti face mai departe.

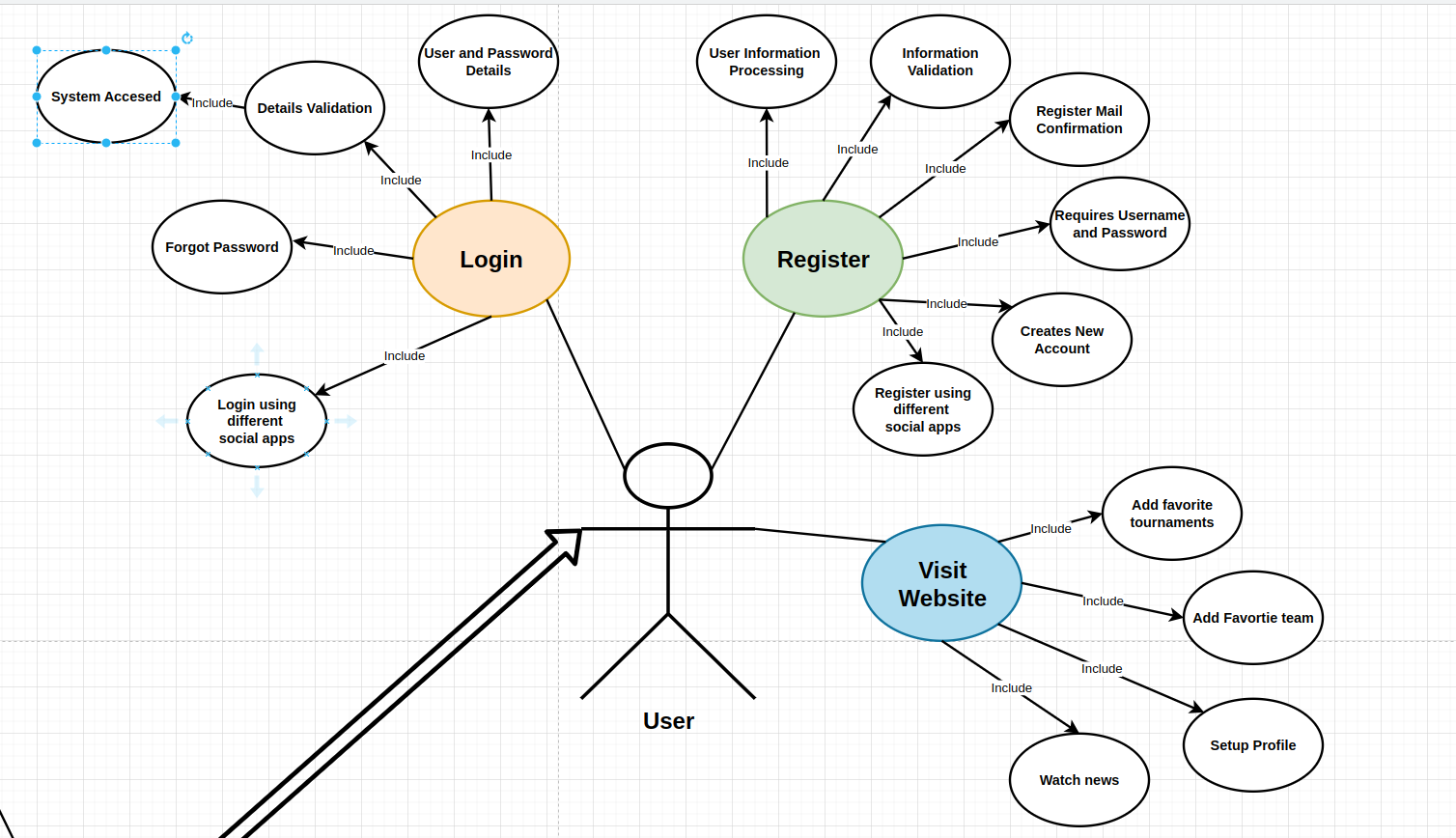
Utilizatori simpli nu pot face mare lucru insa la finalizarea quizz-ului host-ul poate da restart si aceel quizz se va mai juca inca odata de fiecare utilizator.

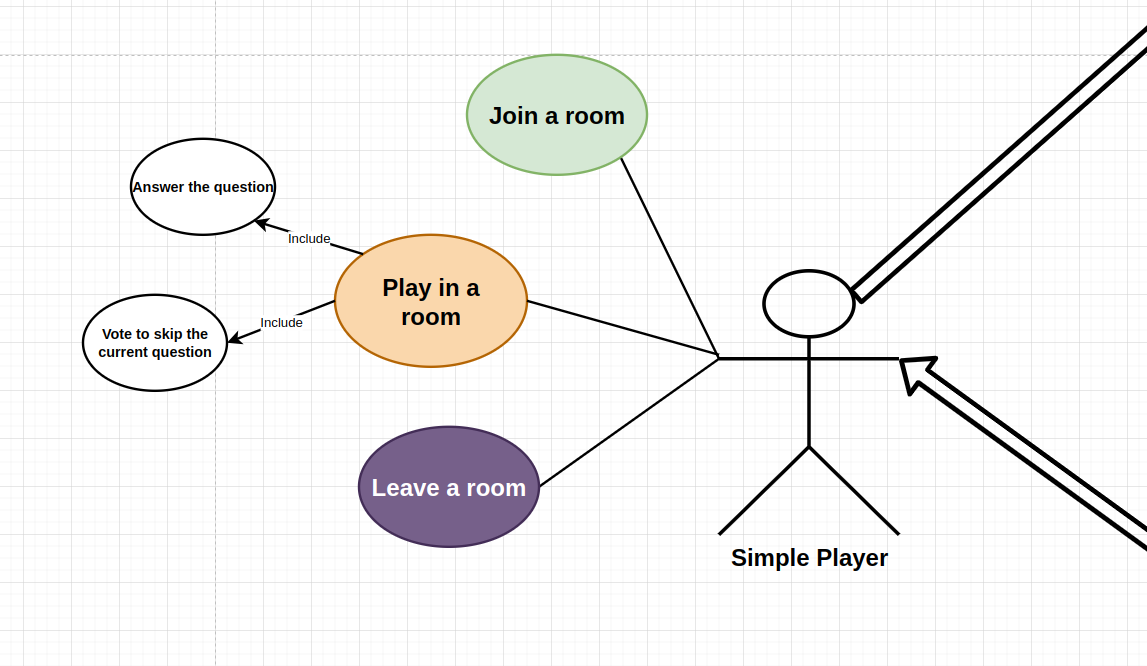
Punctajele partiale sunt salvate in baza de date si se actualizeaza automat. Odata ce host-ul iese din camera ficare utilizator va fi dat afara si va primi un mesaj in care va afla ca camera nu mai exista.

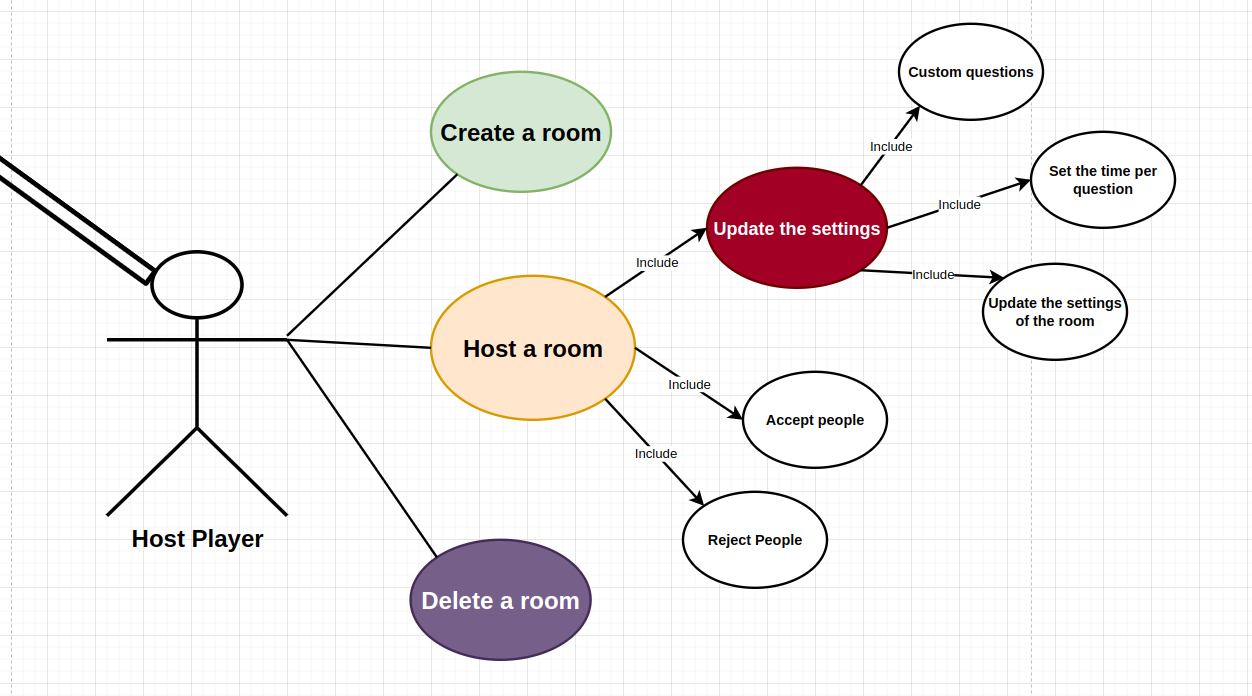
Bogdan: La nivel functional am implememntat foarte multe si am scris enorm de mult cod, m am dedicat foarte mult acestui proiect si cred ca am facut o treaba buna. Are nevoie de mai mult Frontend Design dar baietii au fost cam absenti deci am fost eu full stack si limbajele au fost complet noi deci a trebuit sa invat tot de la 0 a fost o munca solicitanta dar cu un rezultat pot zice eu bine meritat.

1. **Diagrama USE CASE**

****

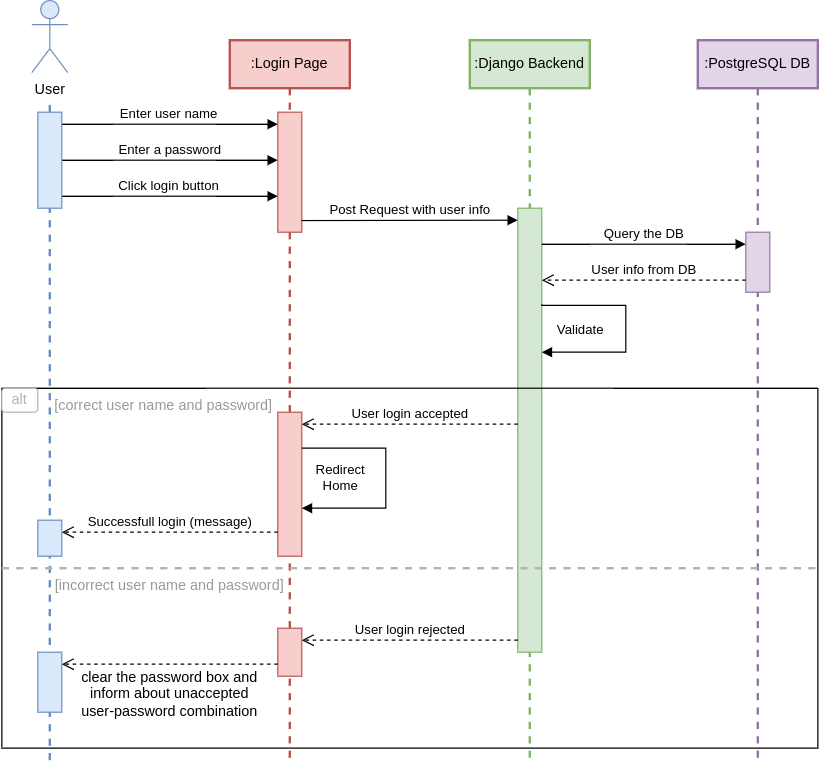
****

****

****

1. **Alte Diagrame**

**Iamnitchi Bogdan - Sequence Diagram for Login**

****

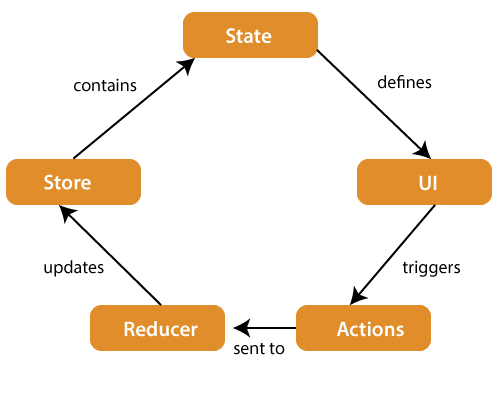
1. **Design Pattern-uri**

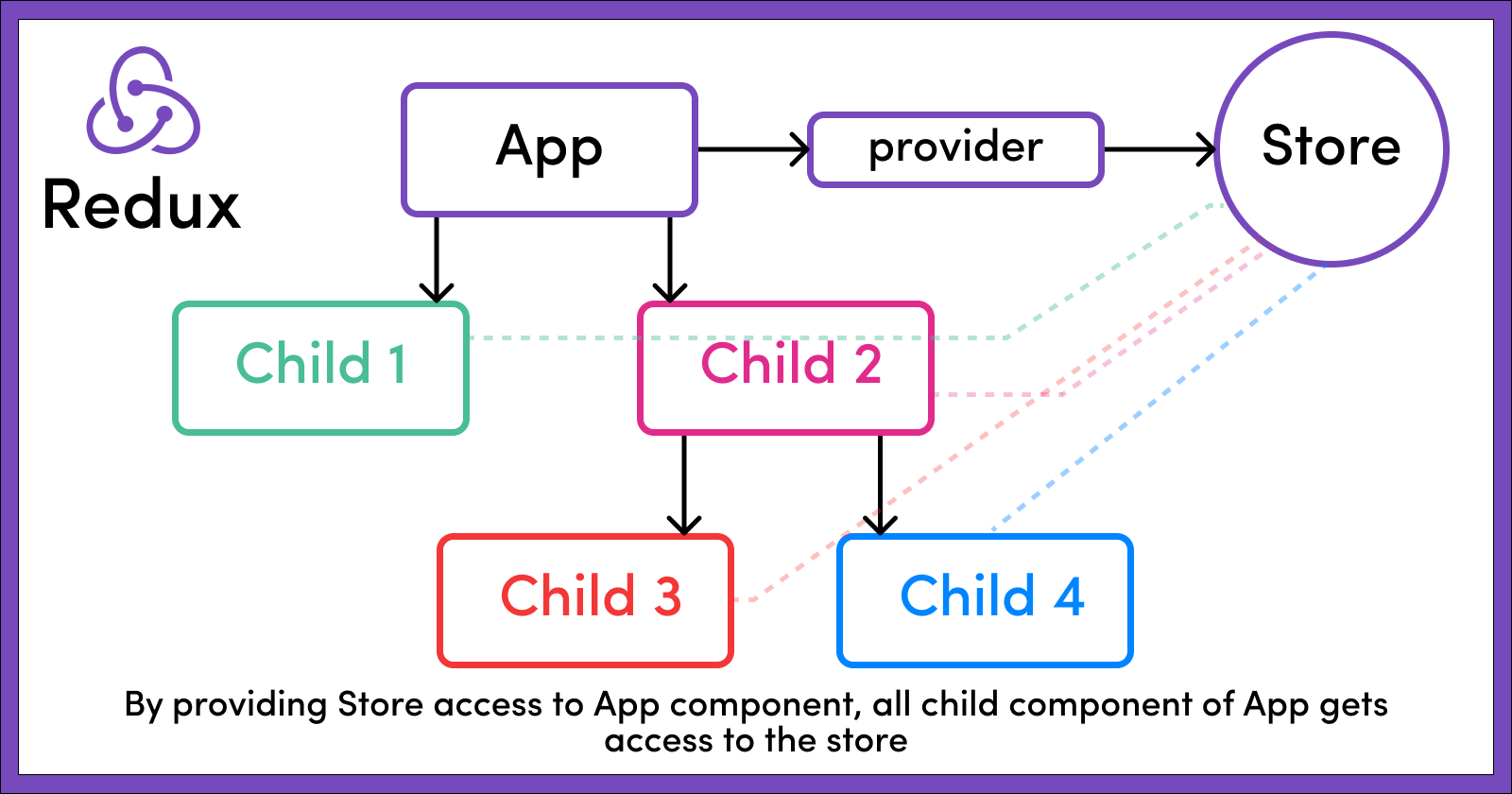
**Iamnitchi Bogan - Observer Design Pattern**

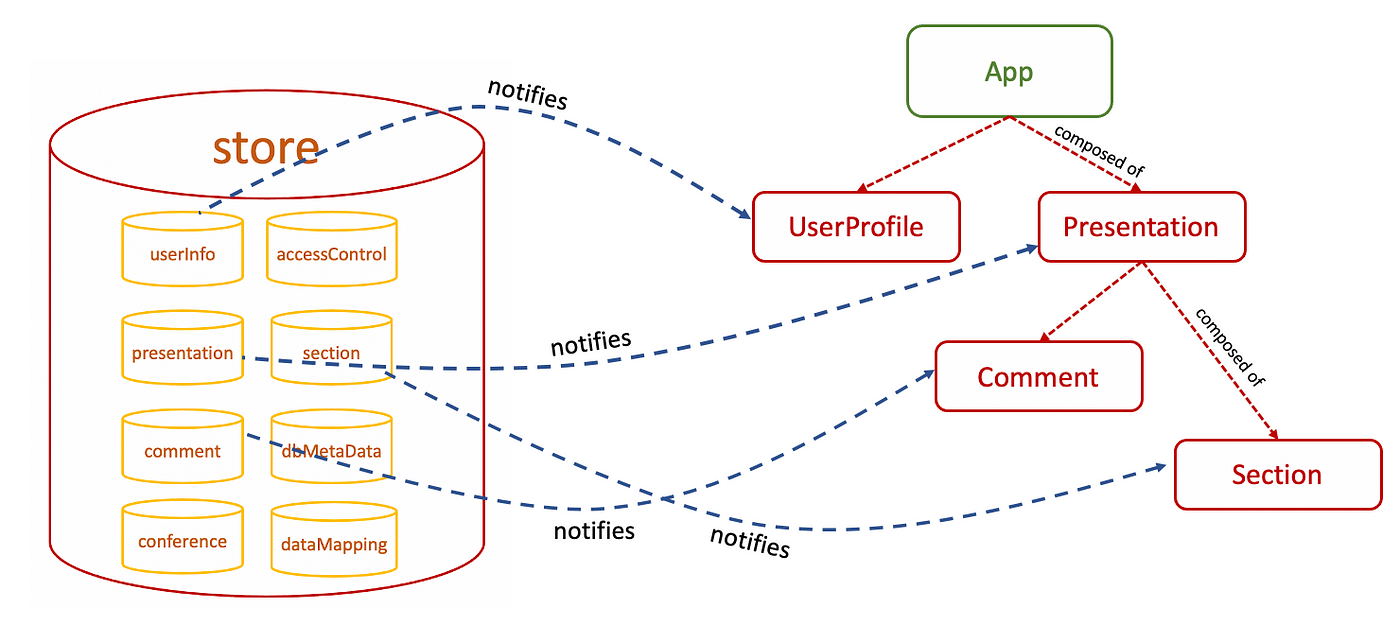
Problema a aparut atunci cand am implementat sistemul de login in React, deoarece fiecare componenta avea nevoie sa stie daca user-ul este logat sau nu. De exemplu NavBardul trebuie sa stie daca user-ul este logat deoarce daca nu este arata butonul de Login / SignUp iar daca este trebuie sa afiseze butonul de Logout.

O solutie simpla ar fi sa pasam ca un argument la instantierea unei componente o variabila care ne zice daca user-ul este logat sau nu, dar asta ar insemna ca si componentele care nu au nevoie neaparat de aceasta variabila o sa o primeasca doar pentru a o da mai departe catre o componenta care poate va avea nevoie.

Astfel am cautat o solutie mai avantajoasa pe internet ca tot omul, si am dat de asa numit-ul Redux, care este o dependinta pe care o instalam si dupa o includem si ne permite sa avem un store (un loc static in care pastram starea aplicatiei noastre) la care au access fiecare componenta care creaza pagina noastra.



Implementand aceasta componenta am realizat ca defapt se supune unui design pattern si mai precis Observer. Deoarece odata ce o actiune a avut loc nu face altceva decat sa notifice celelalte compinente “hei uite care e noua stare”.



**Anghel Mihai Radu - Singleton Design Pattern**

Dupa research-ul facut am ajuns la concluzia ca acest design ar fi cel mai bun pentru a seta conexiunea la baza de date deoarece numele bazei de date va fi acelasi pentru toti utilizatorii aplicatiei.

**Varvari Alexandru - Composite Design Pattern**

Una dintre probleme a fost navigarea intre pagini, iar rezolvarea a fost adaptarea acestuia pentru a crea un tamplate pentru toate paginile, acestea avand componente comune si similaritati (navbar-ul fiind comun in toate paginile).

Acesta ajuta la ulterioara creare a fiecarei pagini. Dupa ce este incarcat template-ul se incarca particularitatile fiecarei pagini.

1. **Limbaje si Framework-uri utilizate**

Inainte de enumerarea limbajelor vreau sa spun ca niciunul din noi trei nu am stiu in avans nici un limbaj, a fost pur decizia noastra sa ne aruncam la ceva mai nou si care se foloseste la nivel mai inalt.

**Astfel am decis sa implementam aplicatia astfel:**

* **Backend-ul in Python folosind Framework-ul Django**
* **Frontend-ul in JavaScript utilizand Framework-ul React**

Insa odata implementand acest proiect am decis sa mai schimbam unele limbaje pentru a ne usura munca. De la inceput am zis ca daca nu reusim sa finalizam proiectul macar sa punem o baza solida si sa ne documentam bine pentru alte proiecte viitoare poate chiar ce vom face la locul de munca.

Asfel am vrut ca aplicatia noasta sa si arate bine, asfel pentru toate stilizarile am folosit un framework aditional **shadcn/ui** (vezi [aici](https://ui.shadcn.com/))care are diverse componente gata de utilizat. Insa acest framework merge doar cu **TypeScript** si am zis ca de ce nu deoarece e foarte similar cu JavaScript si oricum nu era mare schimbarea pt ca nu stiam nici unul din limbaje. Desigur acest framework iti ofera posibilitatea si de a schimba orice componenta deoarce genereaza codul sursa al componentelor care poate fi modificat. Asfel am mai avut nevoie de un framework pentru acest lucru si acesta este **Tailwind Css** (vezi [aici](https://tailwindcss.com/)). Toata aplicatia de frontend este rulata de un enviorement special si acela **Vite** (vezi [aici](https://vitejs.dev/)),

Pe partea de backend a fost destul de simplu deoarece am folosit doar Python (v3), partea dificila a fost ca trebuia sa inveti cum sa folosesti frameworkul Django, de exemplu ce fisiere trebuie modificate ce fisiere aditionale poti creea. Ce am mai folosit pe partea de backend a fost Djoser care este Implementare REST a sistemului de autentificare Django. Biblioteca **Djoser** furnizează un set de vederi Django Rest Framework pentru a gestiona acțiuni de bază, cum ar fi înregistrare, autentificare, deautentificare, resetare parolă și activare cont.

1. **Baza de date**

Baza de date utilizata a fost una pe model relational si mai precis PosgreSQL. Nu am avut de creeat tabele noi sau ceva din intervata bazei de date deoarece Django se ocupa de tot acest lucru. Se creeaza niste modele (clase in python) pe care Django stie sa le traduca in cod SQL.

Pentru a face acest lucru trebuie doar sa dam comanda

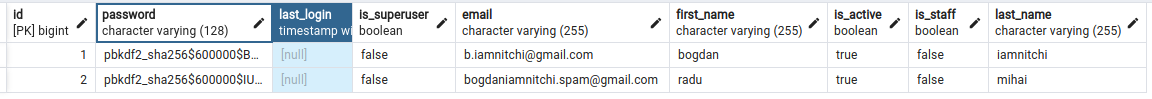
* python3 manage.py makemigration

Dupa care putem vedea ca sa creat un db.sql care e codul sql care va urma sa fie migrat spre serverul de baze de date. Folosim deci:

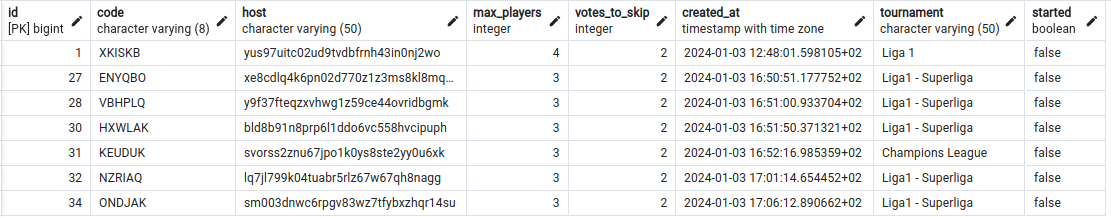
* python3 manage.py migrate

Acum toate modele / clasele facute in python se vor traduce si migra in baza de date sub forma de tabele, cu tot cu relati si ce mai trebuie. Daca deschidem PgAdmin o sa vedem ca toate modificarile au avut loc.

Desigur au fost introduse si numeroase tabele care tin strict de Django dar le putem vedea si pe acelea custom realizate de noi.

**AUTHENTICATION:**

**ROOMS:**

 **QUESTIONS:**

1. **README - how to**

**Link repo:** [**https://github.com/BRA-Team/kick-off-picks**](https://github.com/BRA-Team/kick-off-picks)

Partea aceasta va fi putin mai dificila dar nu imposibila pentru ca trebuie instalate o groaza de dependinte atat pe partea de frontend cat si pe partea de backend.

1. Primul pas este sa faci clone la repo-ul nostru

**git clone git@github.com:BRA-Team/kick-off-picks.git**

2. Al doilea pas este sa intri in folderul creat si sa il deschizi de preferat cu VsCode ca asa am lucrat si noi. Acolo o sa vezi 3 foldere: about unde tinem documentatia si alte scheme si explicatii, backend si frontend. Desigur mai gasesti si acolo un README.md unde sunt acelasi explicatii.

Ai nevoie de doua servere unul de frontend care randeaza tot ce tine de frontend, iar unul de backend care se ocupa cu reqeust-urile. (Se poate rula in productie doar pe serverul de backend insa e sensibil la erori nu recomand)

**SERVER FRONTEND – terminal separat**

**Pentru a rula serverul de frontend:**

1. Trebuie sa intri in folderul de frontend

* cd frontend-tsx

1. Dupa care sa rulezi:

* npm install

\*daca acest pas genereaza erori poti rula

* npm install --legacy-peer-deps

1. Dupa care trebuie sa rulezi:

- npm run build (altfel serverul de backend nu porneste)

1. Dupa care rulezi:

* npm run dev (si il lasi asa deschis)

**\* de precizat ca ai nevoie sa instalezi node.js si npm (vezi** [**aici**](https://nodejs.org/en/download)**)**

**FOARTE INPORTANT – sa creezei in frontend-tsx un fisier numit .env in care sa copiezi fisierul continut-fisier-env.txt din folderul about**

**SERVER BACKEND – terminal separat**

1. Trebuie sa ai neaparat Python instalat pe sistem altfel nu o sa poti rula proiectul nostru (deci ar fi bine sa vizitezi <https://www.python.org/downloads/> daca nu il ai deja) Pentru a putea sa verifici daca ai python instalat poti sa rulezi:

* **python --version**

1. Trebuie desigur sa ai si PIP (Preferred Installer Program) care este un manager de pachete / dependinte pentru python. De obicei este instalat odata cu python.

Pentru a putea sa verifici daca ai pip instalat poti sa rulezi:

* **pip --version**

1. Urmatorul pas este sa deschizi un terminal in VsCode sau orice folosesti si te duci in folderul backend folosind comanda:

* **cd backend**

1. Acum urmeaza sa instalam niste de dependinte. Pentru a nu iti incarca calculatorul cu dependinte de care nu ai nevoie, o sa cream un asa numit virtual environment. E ca si cum o sa ai instalate toate dependitele si versiunea corecta de python doar pentru proiectul nostru.
2. Asfel hai sa instalam dependinta care ne ofera aceasta posibilitate:

* **pip install virtualenv**

1. Pentru a creea un virtual environment trebuie sa rulezi:

* **python -m venv <virtual-environment-name>**

1. Acum daca totul a mers cu succes trebuie doar sa il activam:

* **<venv\_name>\Scripts\activate.bat**

1. Pentru a verifica ca virtual env este activa te poti uita in terminal la linia de comanda si o sa apara ceva in fata cu niste paranteze de exemplu **(venv)-**bogdan@vivobook sau:

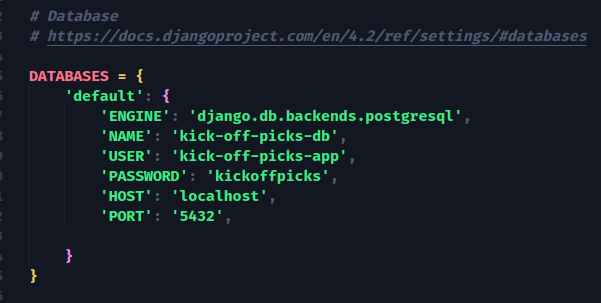
* **pip list**

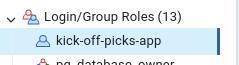
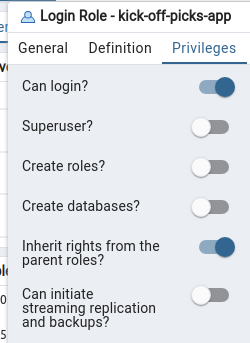
Care iti va arata pachetele instalate, ar trebui sa ai numa doua deoarece abia ai activat environment-ul

1. Acum odata ce **virtual enviormentul este activat** trebuie sa instalam toate dependintele folosite in acest proiect, ca sa fie mai simplu le-am pus toate in requirements.txt, ce trebuie sa faci este sa rulezi:

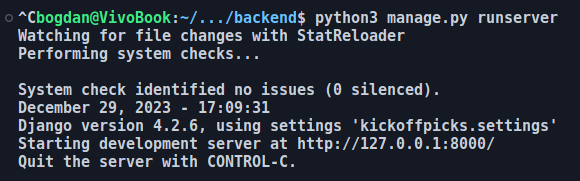
* **pip install -r requirements.txt**

1. Mai trebuie insa facute cateva modificari deoarece server-ul nu poate rula fara o baza de date activa la care sa se conecteze. Astfel incat trebuie sa deschizi fisierul settings.py gasit in **backend/kickoffpicks/settings.py** si sa dai scroll pana nu vezi:

aici trebuie sa completezi la engine cu baza de date pe care o folosesti, la name numele bazei de date care trebuie sa o creezi, user vezi mai jos, parola host si port.

1. In baza ta de date trebuie sa ai definit un nou user / rol care trebuie sa aibe permisiunea de login, si sa completezi acel user in settings, la sectiunea user, vezi pasul de mai sus.
2. Acum daca toate au mers bine ar trebui sa poti rula server-ul

* **python3 manage.py runserver**

****

1. Acum trebuie doar sa apesi pe link-ul de acolo care te redirectioneaza la site-ul nostru!

De precizat daca esti pe linux la fiecare comanda in loc de **python** va fi **python3** iar la activarea **virtual env** se va folosi comanda:

* **source .venv/bin/activate**