**Fîță Robert**

**Tăbăcaru Andrei**

[**https://github.com/FitaRoberttt/hangman-game.git**](https://github.com/FitaRoberttt/hangman-game.git)

**Hangman**

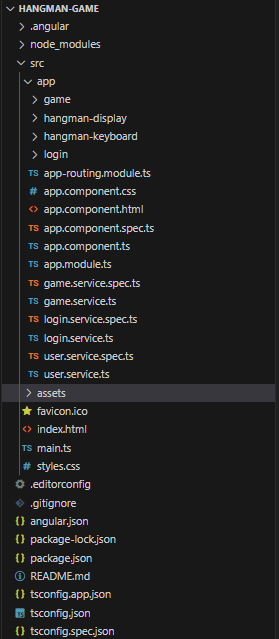
Jocul "Hangman" implementat pe platforma web, cu un Front-End construit în Angular și un Back-End în Java, oferă o experiență captivantă și interactivă pentru utilizatori. În esență, jocul implică ghicirea unui cuvânt, literă cu literă, într-un număr limitat de încercări, cu fiecare răspuns incorect contribuind la desenarea unui om atârnat (hanged man) pe ecran.

**Caracteristici principale:**

* **Interfață Utilizator Prietenoasă:**
  + Front-End-ul Angular oferă o interfață modernă și prietenoasă pentru utilizatori.
  + Design responsiv pentru o experiență de utilizare plăcută pe diverse dispozitive.
* **Generare Cuvinte Aleatoare:**
  + Back-End-ul Java generează cuvinte aleatoare pentru ghicit, având diferite nivele de dificultate.
* **Joc Interactiv:**
  + Utilizatorii pot începe un joc nou selectând nivelul de dificultate și începând să ghicească litere.
  + Fiecare literă corect ghicită este afișată, în timp ce literele incorecte contribuie la desenarea unui om atârnat.
* **Feedback și Stare Joc:**
  + Utilizatorii primesc feedback instant pentru fiecare ghicire sau greșeală.
  + Starea jocului (cuvântul ghicit, încercările rămase) este actualizată în timp real.
* **Final de Joc:**
  + Jocul se încheie atunci când utilizatorul ghicește cuvântul sau când numărul de încercări este epuizat.
  + Se afișează un rezumat al rezultatului, indicând dacă utilizatorul a câștigat sau pierdut.
* **Persistența Datelor:**
  + Back-End-ul gestionează persistența datelor jocului, permițând reluarea sau încheierea jocurilor începute.

**Prezentarea Proiectului: Front-end**

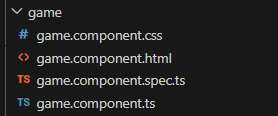
1. Structură:



Clientul este împărțit în componente, fiecare având 4 file-uri specifice

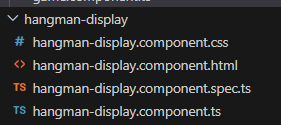
(CSS, HTML, SPEC.TS, TS)

-> Componenta “Game”

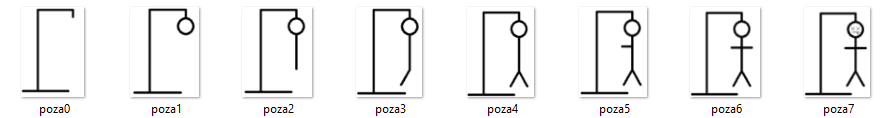


Este componenta principala care uneste restul componentelor, cu ajutorul ei ruleaza aplicatia in modul dorit.

-> Componenta “Hangman-display”



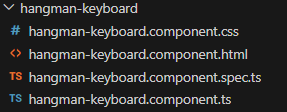
Reprezintă etapele personajului de-a lungul ghicirii cuvântului



Poza0=starea inițială de la începutul jocului

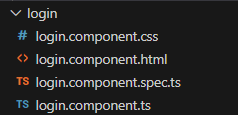
Poza7=starea finală in cazul pierderii

-> Componenta “Hangman-keyboard”



Reprezintă tastatura prin intermediul căreia utilizatorul poate ghici literele din cuvant.

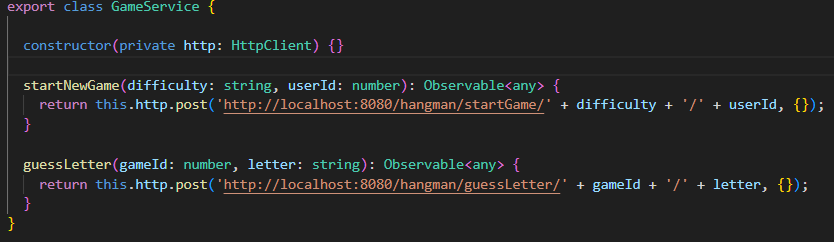
-> Componenta “Login”



Reprezintă o etapă obligatorie, utilizatorul trebuie mai intai să se logheze pentru a putea folosi jocul.

Alte file importante sunt: “game.service.ts”, “login.service.ts”, în acestea are loc comunicarea client-server prin request-uri:

-> “game.service.ts”

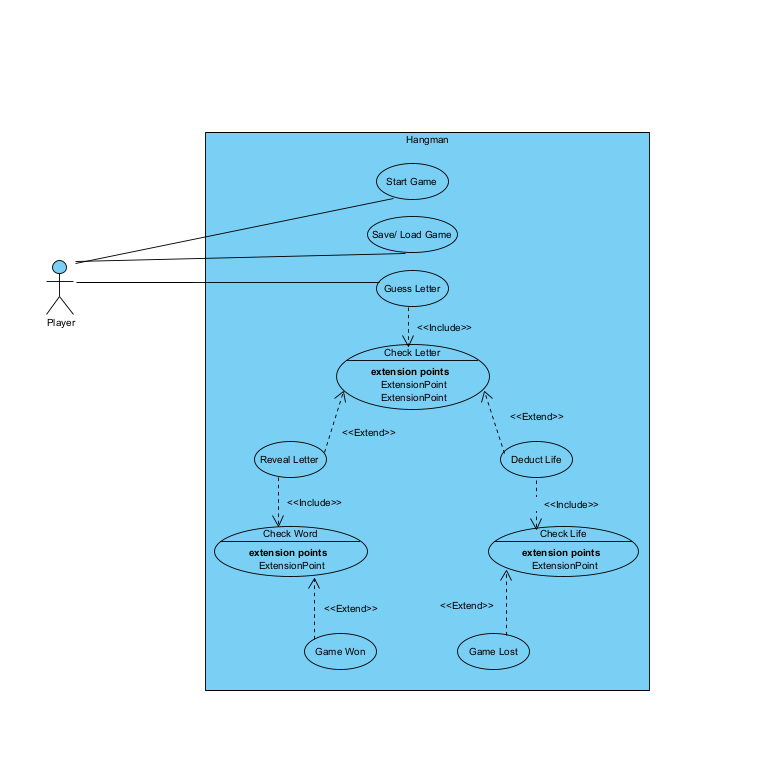


-> “login.service.ts”



**Diagrame:**

1. Use-case diagram

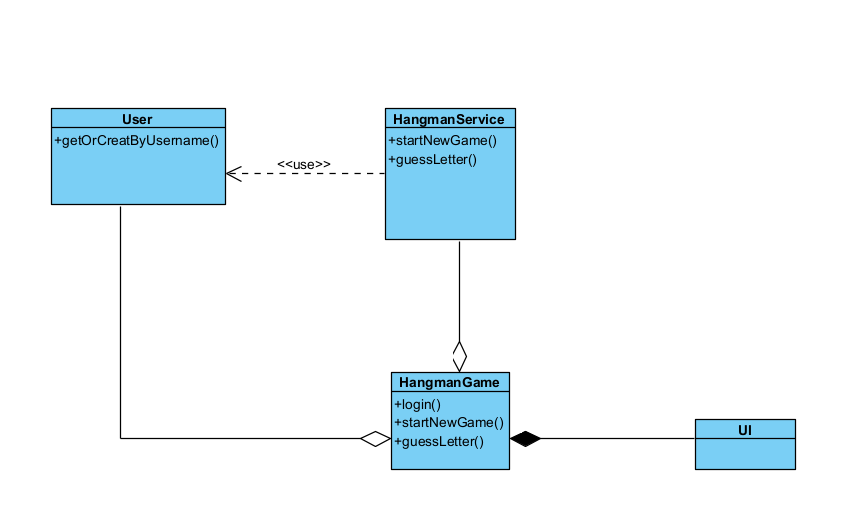


* Utilizatorul are optiunea de a incepe un nou joc sau de a continua un joc deja existent. Ambele cazuri includ aceeasi logica: ghicitul literelor. In functie de litera oferita de jucator urmează următoarele scenarii:

-> scenariul bun in care litera este corecta si se verfica daca cuvantul este ghicit complet lucru care inseamna ca a castigat

-> scenariul rau in care litera este gresita si se verifica daca mai sunt vieti disponibile, daca nu mai sunt, inseamna ca a pierdut.

1. Class diagram



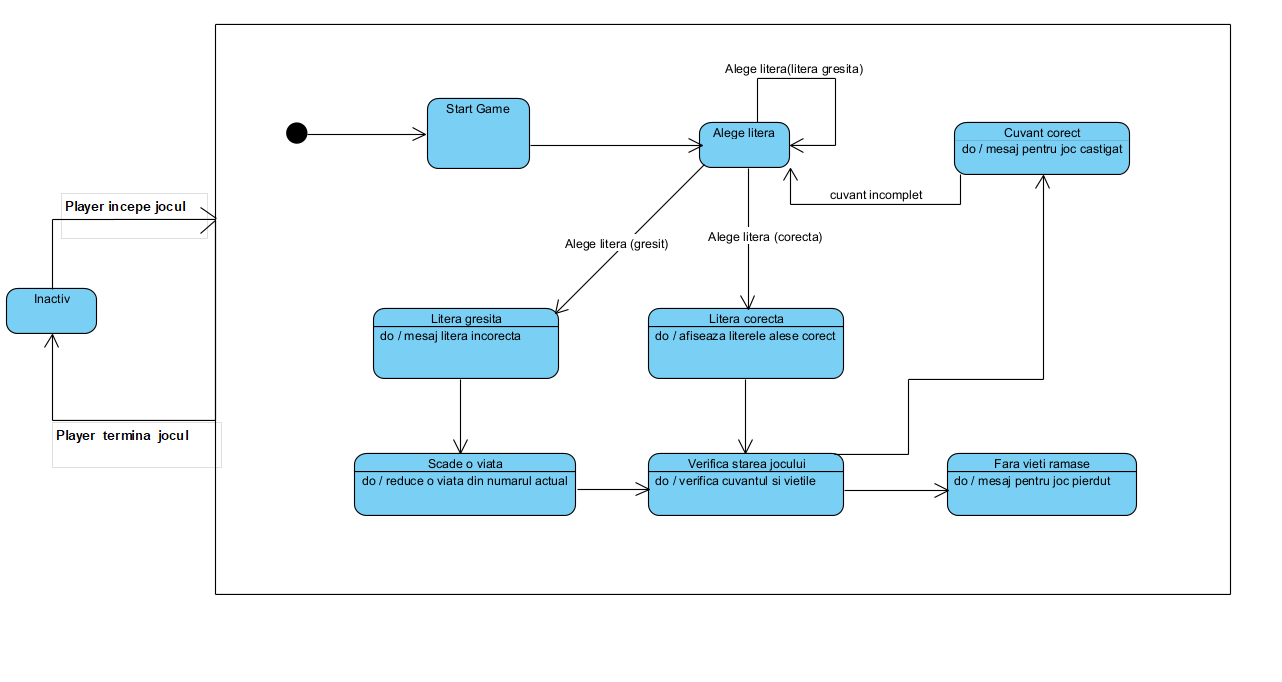
* În diagrama de clase se prezintă legăturile dintre utilizator si aplicație. Se pot observa relatiile:

-> de agregare între user si joc (jocul poate avea mai mulți useri) de asemenea între funcționalități și joc(jocul are mai multe functionalitati)

-> de compozitie intre UI(User Interface) si joc( e nevoie de UI pentru ca aplicatia sa poata fi folosita)

-> relație de dependență între functionalitați și utilizator

1. Statechart diagram



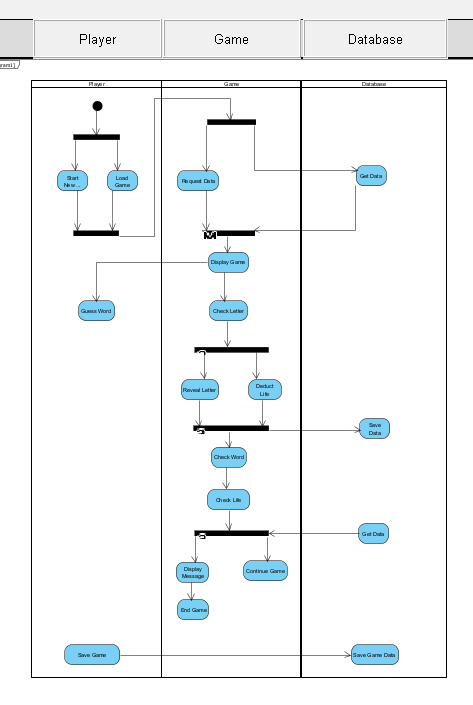
* Prin diagrama de stare se prezintă comportamentul aplicatiei. Player-ul incepe jocul, in functie de ce literă alege (gresită/corectă) va avea un parcurs diferit.

Astfel:

-> dacă este corectă, se afisează litera in cuvant si se verifica starea cuvantului. In cazul in care s-au ghicit toate literele se afiseaza un mesaj pentru joc castigat. Altfel se reintoarce la ghicit litere.

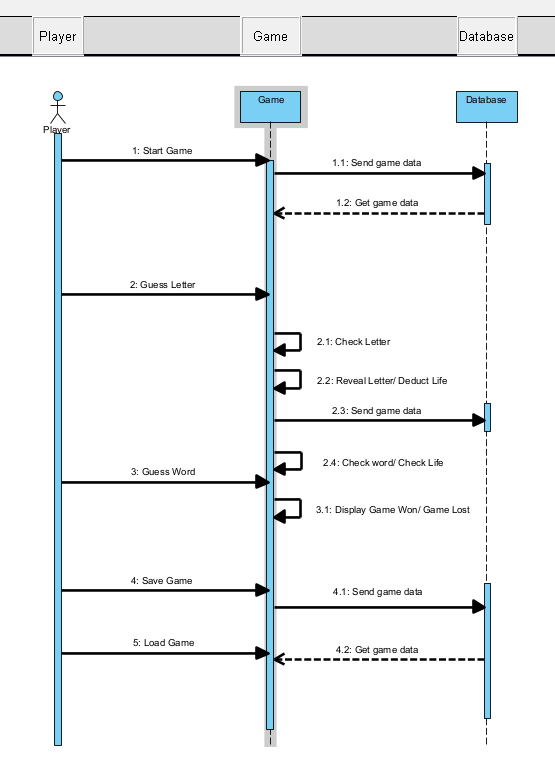
-> daca este gresita, utilizatorul este anuntat ca litera este gresita adaugandu-se din partile lipsa la personaj si scazandu-se din vietile posibile. In cazul in care s-au scurs toate vietile se afiseaza un mesaj de joc pierdut.

1. Activity diagram



* Diagrama de activitati arată etapele parcurgerii jocului și legăturile dintre jucator, joc și informatiile stocate în baza de date. Pe toată durata sesiunii este o comunicare continuă între aplicatie si baza de date, lucru total necesar pentru o interacțiune cât mai bună a utilizatorului cu aplicatia.

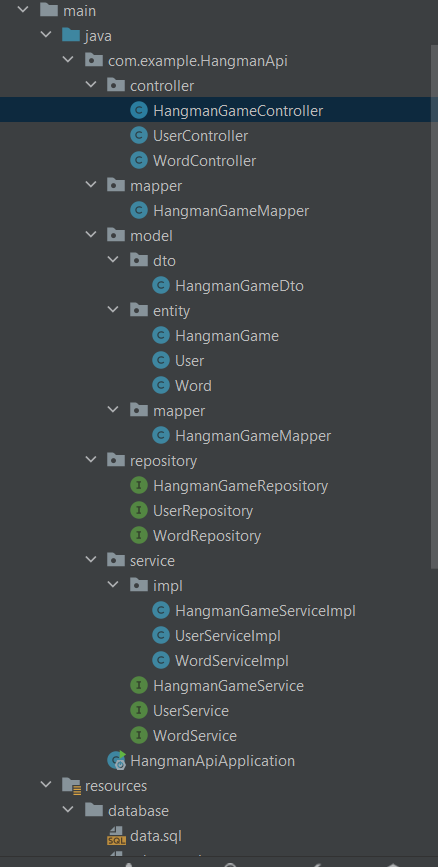
1. Sequence diagram



* Diagrama de secvențe arată în mod explicit ce se întâmplă în spatele jocului după fiecare acțiune a utilizatorului si cum comunică în mod activ jocul cu baza de date, pentru nu a exista pierdere de informatii. Astfel jucatorul poate duce sesiunea la bun sfarsit fara a aparea probleme.

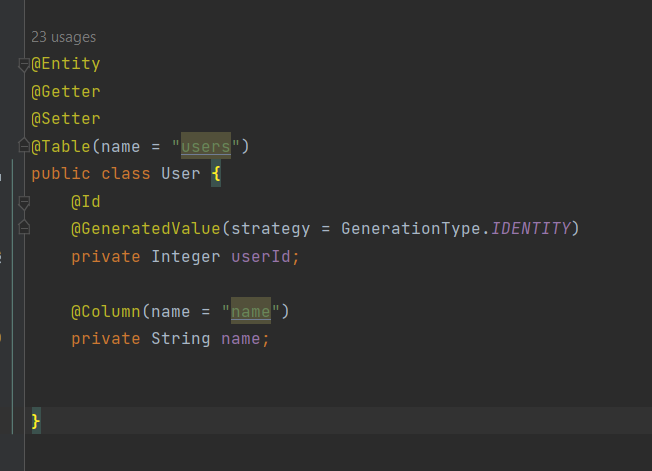
Back-end: Reprezintă partea aplicației în care este implementată logica jocului.

Structura:

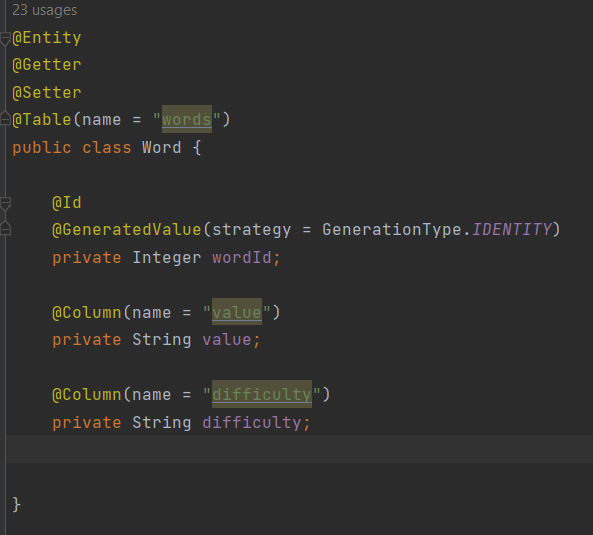


Partea de back-end este alcatuită din cele 3 entități principale ale aplicației

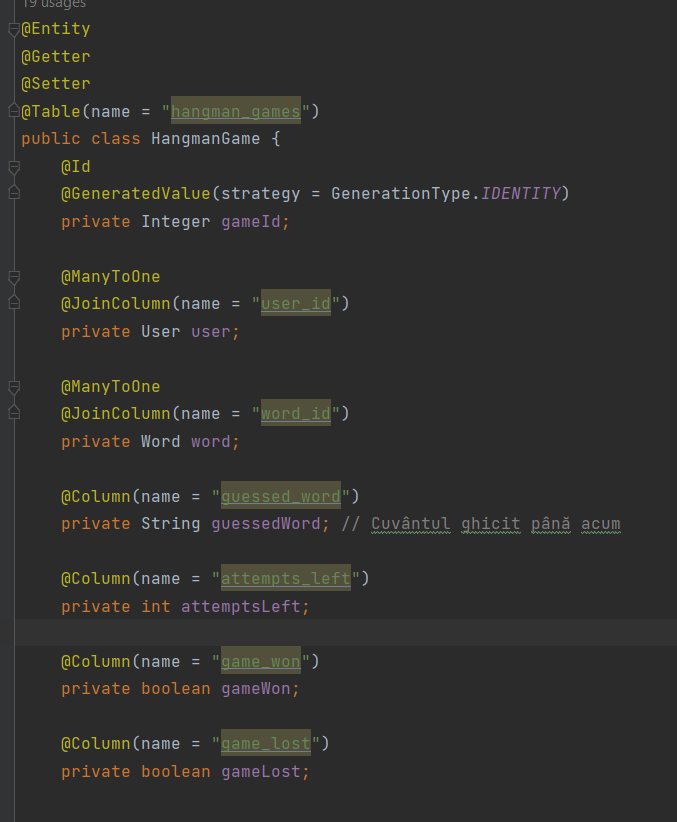
-User:



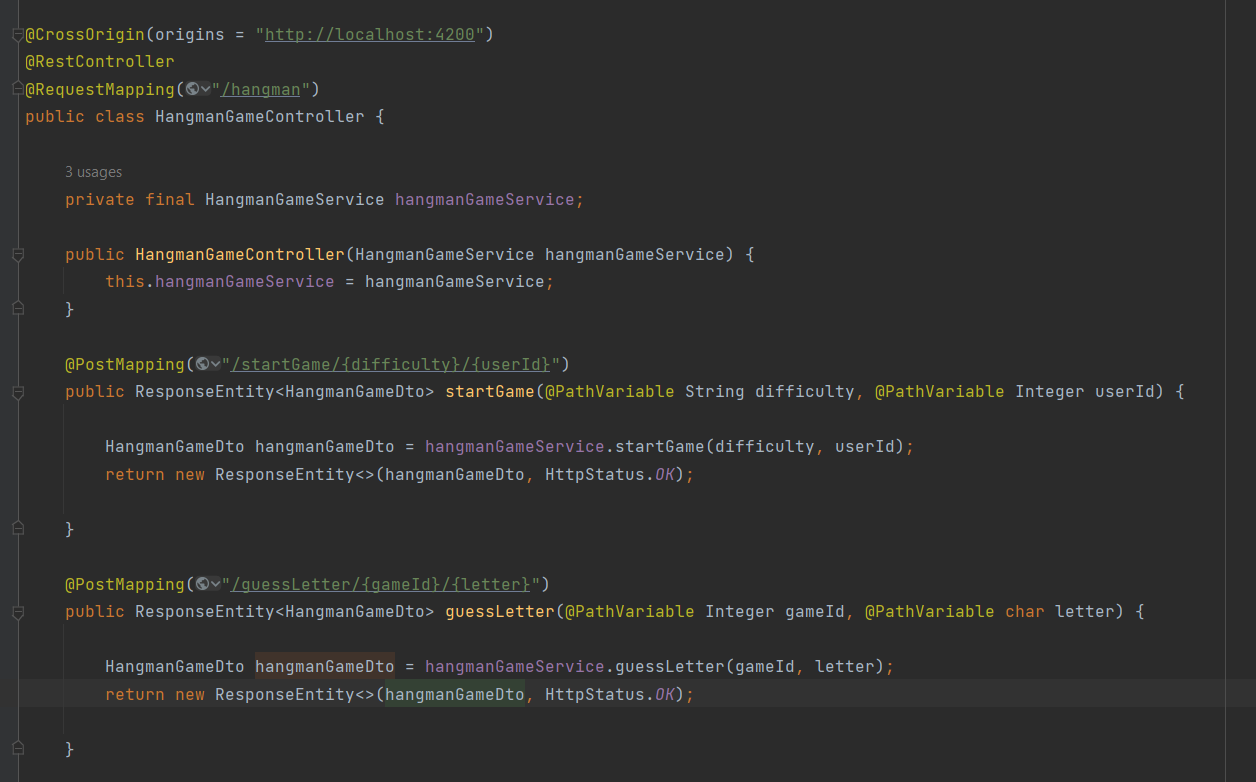
-Word:



-HangmanGame:

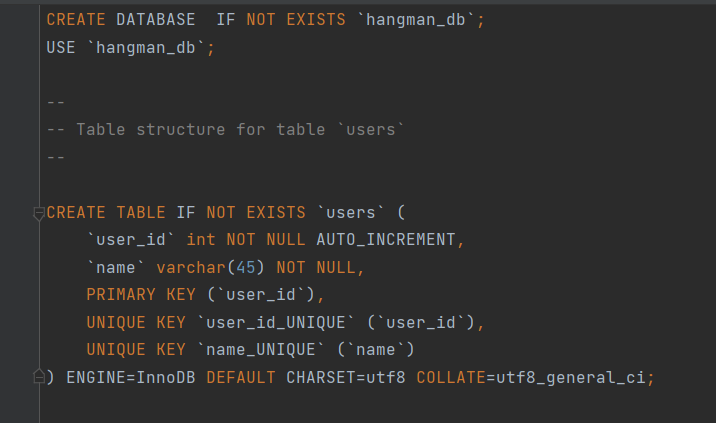


În folderul controller se realizează comunicare cu front-end ul aplicatiei, dar in același timp și cu service-ul care transmite informațiile mai departe către repository și baza de date.

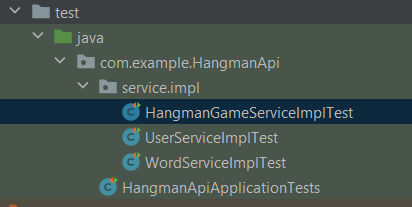




În folderul resources se gestionează conexiunea cu baza de date, dar si scripturile necesare pentru inițializarea ei în momentul pornirii aplicației.



În folderul test sunt implementate unit testele care au rolul de a verifica comportamentul metodelor implementate în clasele UserService, HangmanGameService și WordServiceImplTest.



În clasa de teste (**HangmanGameServiceImplTest**) se verifică comportamentul metodei **startGame** din clasa **HangmanGameServiceImpl**. Rolurile principale ale acestor teste sunt:

**1) Aranjarea (Setup):**

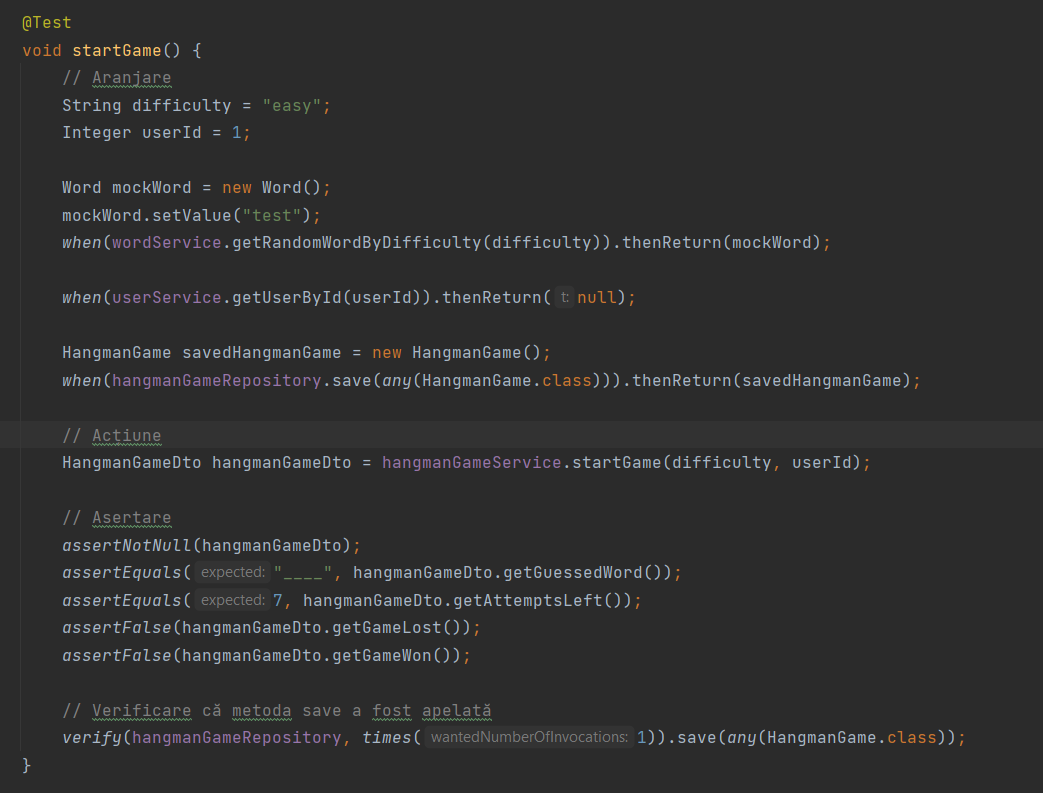
* Se folosesc anotările **@Mock** și **@InjectMocks** pentru a crea obiecte mock și a injecta acele mock-uri în clasa pe care dorim să o testăm (**HangmanGameServiceImpl**).
* În metoda **setUp**, se deschid mock-urile utilizând **MockitoAnnotations.openMocks(this)**. Acest lucru pregătește mediul pentru rularea testelor.

2) **Acțiune (Action):**

* Se specifică comportamentul mock-urilor folosind instrucțiuni **when**. De exemplu, se spune că atunci când se apelează **wordService.getRandomWordByDifficulty(difficulty)**, trebuie să returneze un cuvânt mock.
* Se specifică că atunci când se apelează **userService.getUserById(userId)**, trebuie să returneze **null**.
* Se specifică că atunci când se apelează **hangmanGameRepository.save(any(HangmanGame.class))**, trebuie să returneze un joc mock salvat.

3) **Asertare (Assertion):**

* Se apelează metoda **hangmanGameService.startGame(difficulty, userId)** pentru a obține un obiect **HangmanGameDto**.
* Se asigură că obiectul rezultat are anumite proprietăți așteptate, cum ar fi **guessedWord**, **attemptsLeft**, **gameLost**, și **gameWon**.
* Se verifică că metoda **save** a fost apelată o singură dată pe **hangmanGameRepository** pentru a salva noul joc.



Astfel, cele prezentate urmăresc modul de funcționare al jocului.