

Fejlesztői dokumentáció

Gameknowledge

Tartalomjegyzék

# Bevezetés

A Gameknowledge című weboldal annak céljából készült, hogy a felhasználók könnyedén tájékozódhassanak a videójáték ipar fejlődéseiről, változásairól, mindezek mellett bárki egyszerűen találhasson információkat kedvenc, de akár régebbi játékairól egy helyen, összeszedett és átlátható formában.

A Gameknowledge fő célja, hogy a rengeteg információt, amely kering az interneten a játékokról egyhelyre gyűjtve tiszta, hiteles valamint gördülékenyen visszakereshető formában tárolja és jelenítse meg a felhasználóknak.

Az oldal bizonyos szolgáltatásait bárki igénybe veheti minden gond nélkül, azonban bizonyos funkciókat a felhasználók csak az oldalra való sikeres regisztráció után használhatnak. Ezen funkciók megtalálhatóak a Használati útmutató részben.

A fejlesztés során igénybe lett véve néhány eszköz amelyek a munkát segítették. Ezek a következők:

* Discord, amely a megbeszélések, és egyeztetések végrehajtásához vált elengedhetetlenné.
* Trello, ahol tárolni tudjuk a kitöltött drótváz mint minták oldalakat, a tervezés és fejlesztés során keletkezett idő eltolódásokat valamilyen probléma illetve akadály miatt, a frissítéseket, dokumentációkat, fejlesztési és tervezési folyamatokat és a felhasznált forrásokat. Továbbá lehetővé teszi a visszakövethetőséget.
* GitHub, ahová maguk a kódok, és azok minden egyes frissítései amelyek bekövetkeztek a fejlesztés során kerültek feltöltésre, valamint itt láthatóak a verziókban bekövetkezett változások és azok a pontok ahol elértünk egy kitűzött mérföldkövet. A visszakövethetőséget ez teszi ki legnagyobb többségben.

Verziószám: 1.0.0

Kapcsolattartók:

Földházi Zsolt,

[foldhazi.zsolt.2019@osztalyterem.katolikuskeri.hu](mailto:foldhazi.zsolt.2019@osztalyterem.katolikuskeri.hu)

Vicickó Vencel,

[vicicko.vencel.2019@osztalyterem.katolikuskeri.hu](mailto:vicicko.vencel.2019@osztalyterem.katolikuskeri.hu)

# Rendszerkövetelmények

Bármely webböngészőt futtatni képes operációs rendszeren képes működni a weboldal.

Egy legalább Intel Pentium 4-es processzor vagy későbbi, amely képes az SSE3-ra elegendő hardware a működéshez, azonban ajánlott egy Intel(R) Core(TM) i3-1005G1 1.20GHz processzor, 4 GB RAM valamint egy Windows 11-es operációs rendszer.

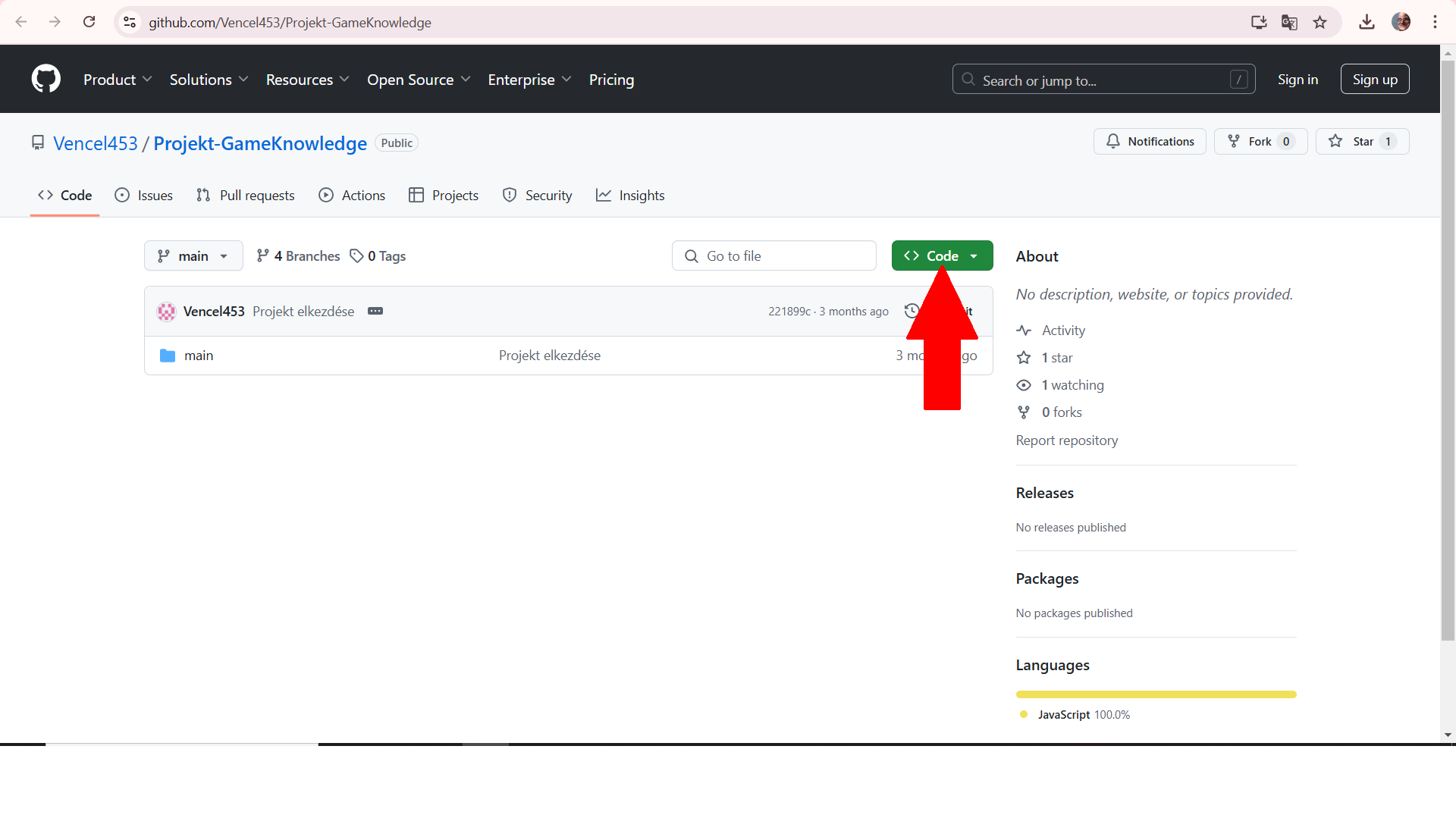
Mindezeken felül, az oldal működéséhez elengedhetetlen a game\_knowledge adatbázis, az Angular keretrendszer 18-as verziója és a NodeJS szoftverrendszer.

# Telepítési útmutató

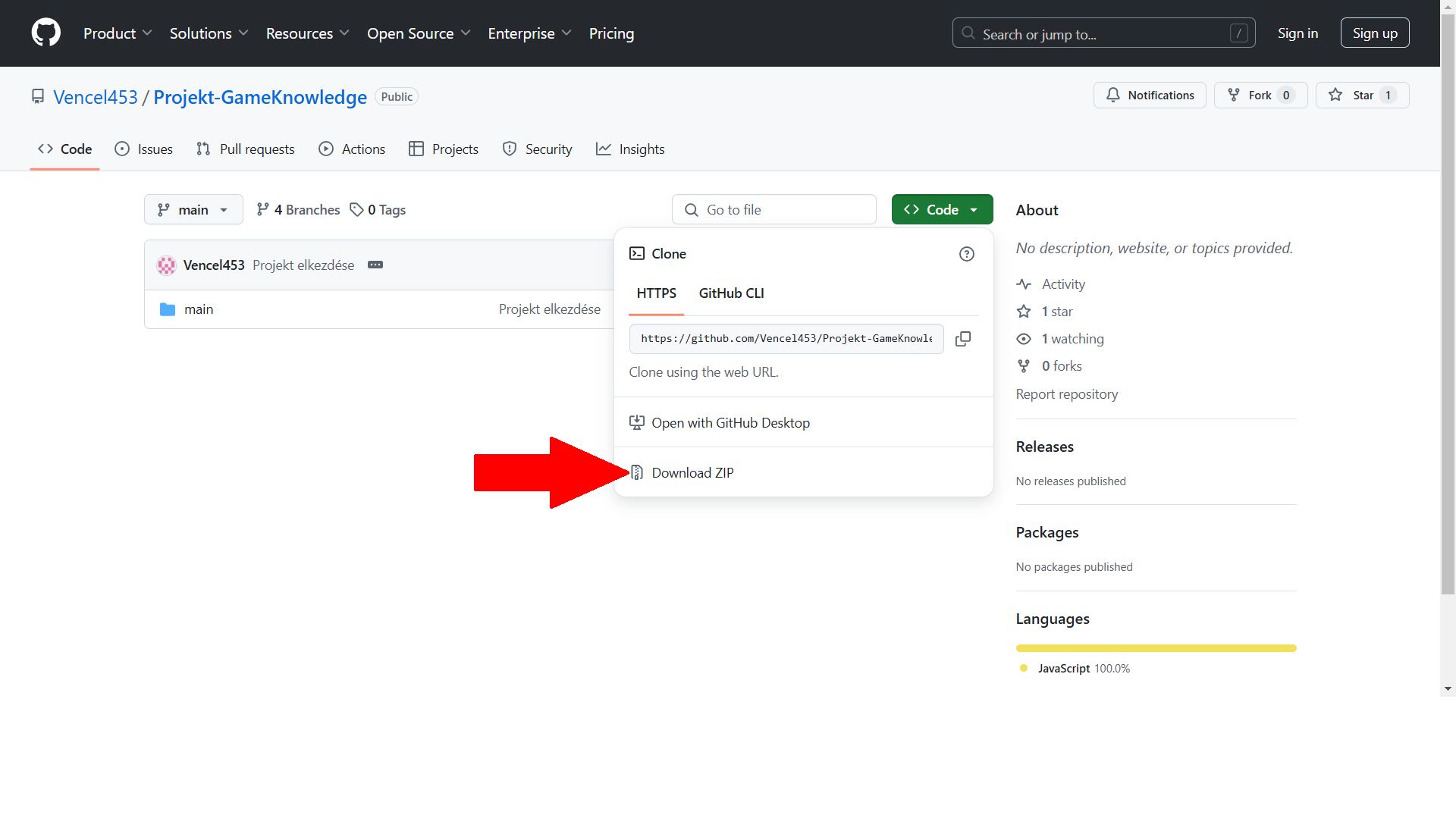
1. Szükség van a forrás kódokra, amelyeket a Github weblapunkról egyszerűen letölthetőek. Navigáljon az oldalunkra az alábbi link segítségével és kövesse a további lépéseket!

<https://github.com/Vencel453/Projekt-GameKnowledge.git>

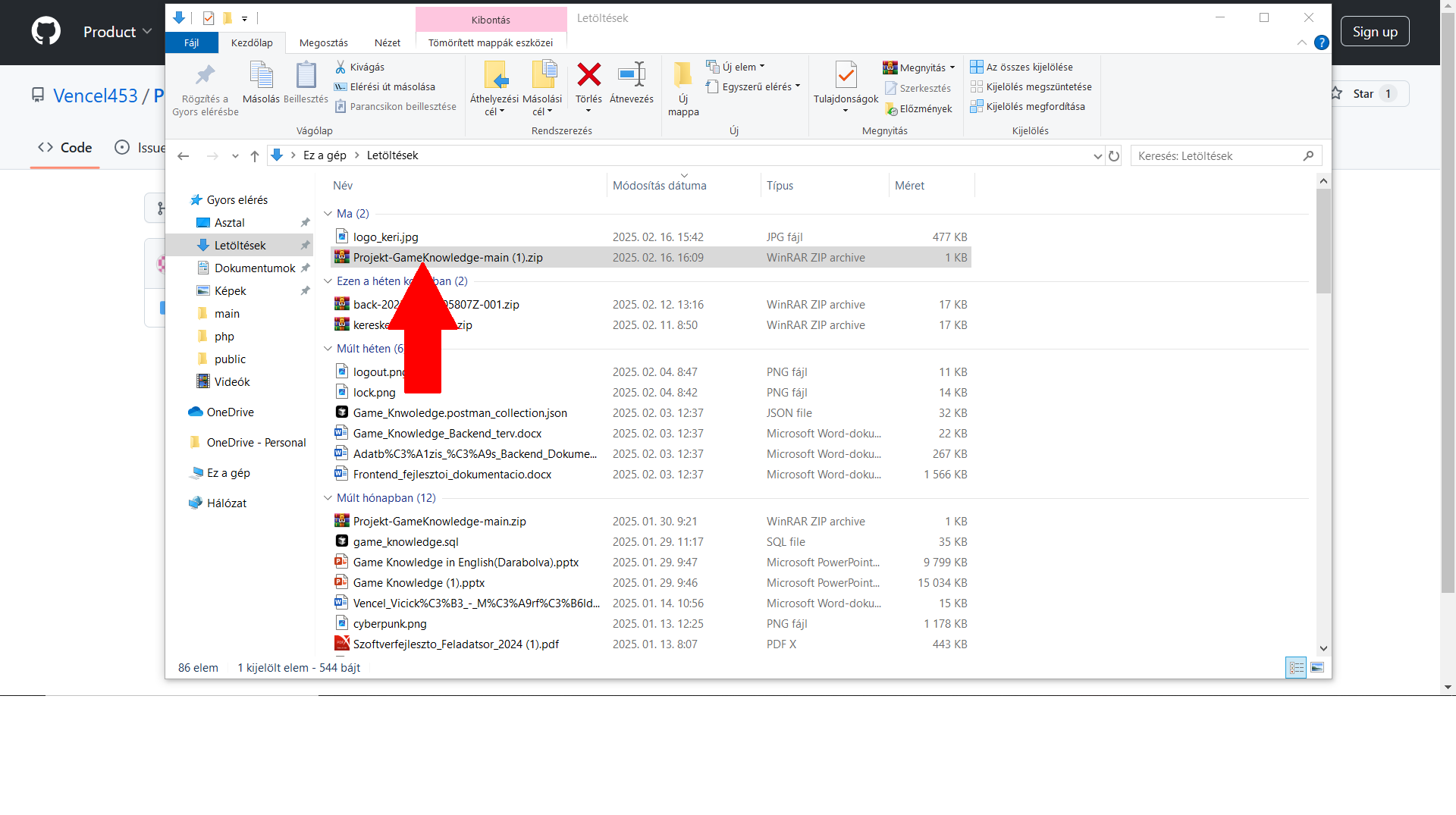
1. Az alábbi kép alapján válassza a < >Code nevű lenyíló menü-t!



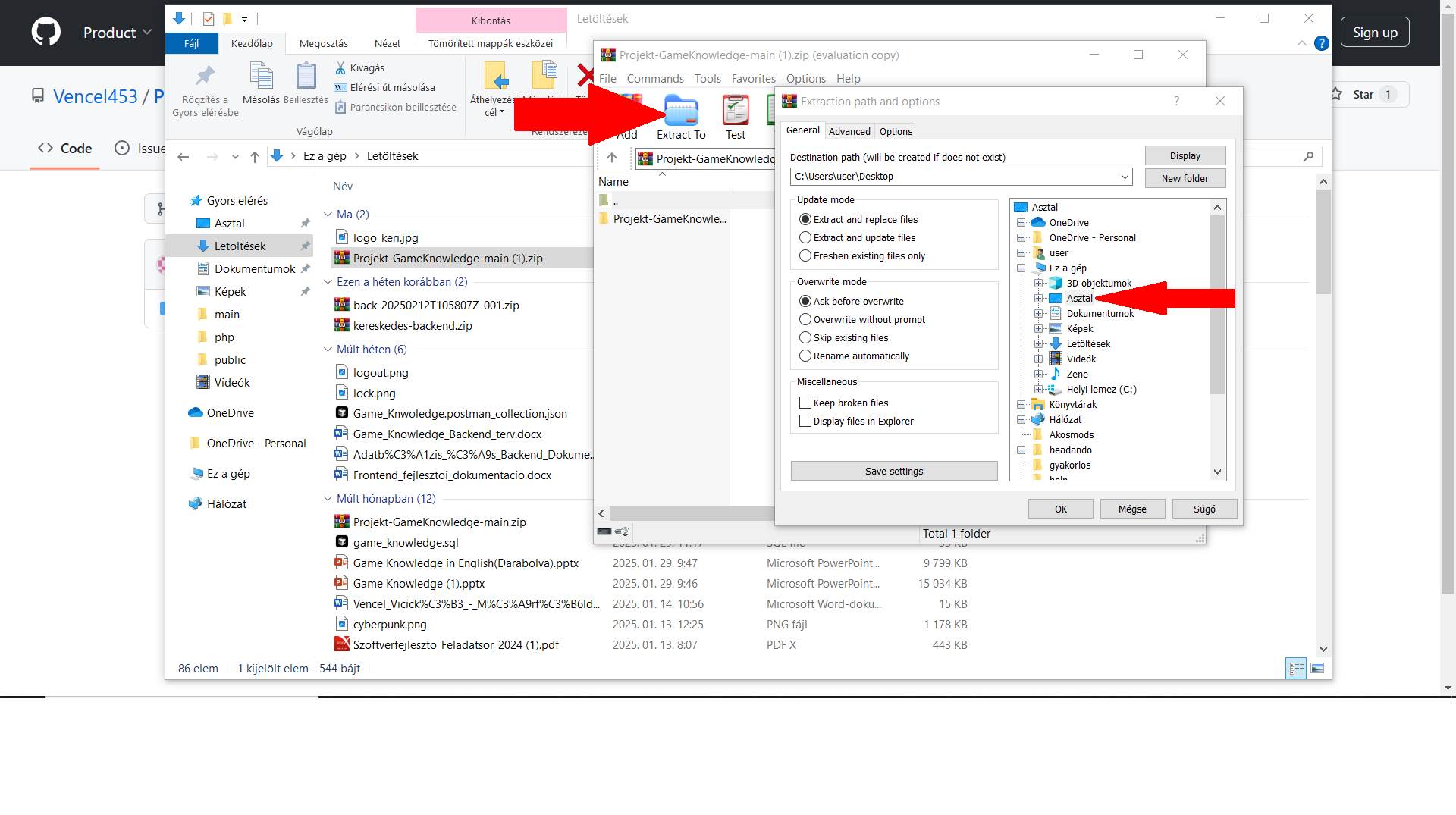
1. *kép*
2. Az alábbi kép alapján válassza a Download ZIP opciót és töltse le a tömörített állományt!



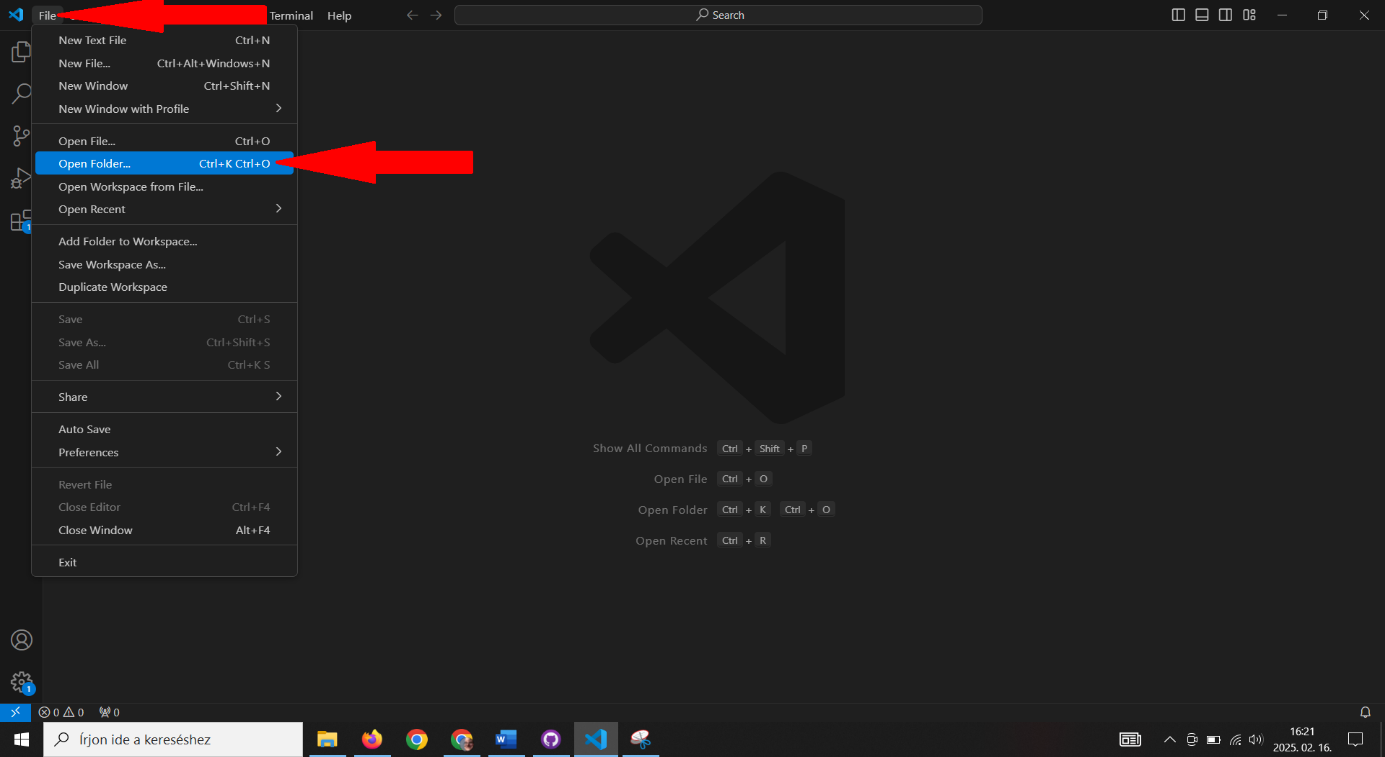
1. *kép*
2. Keresd meg a letöltött ZIP állományt a saját gépeden (valószínűleg a letöltésekben lesz megtalálható), és nyisd meg valamelyik tömörítő programmal, majd az általad kiválasztott helyre bontsd ki!



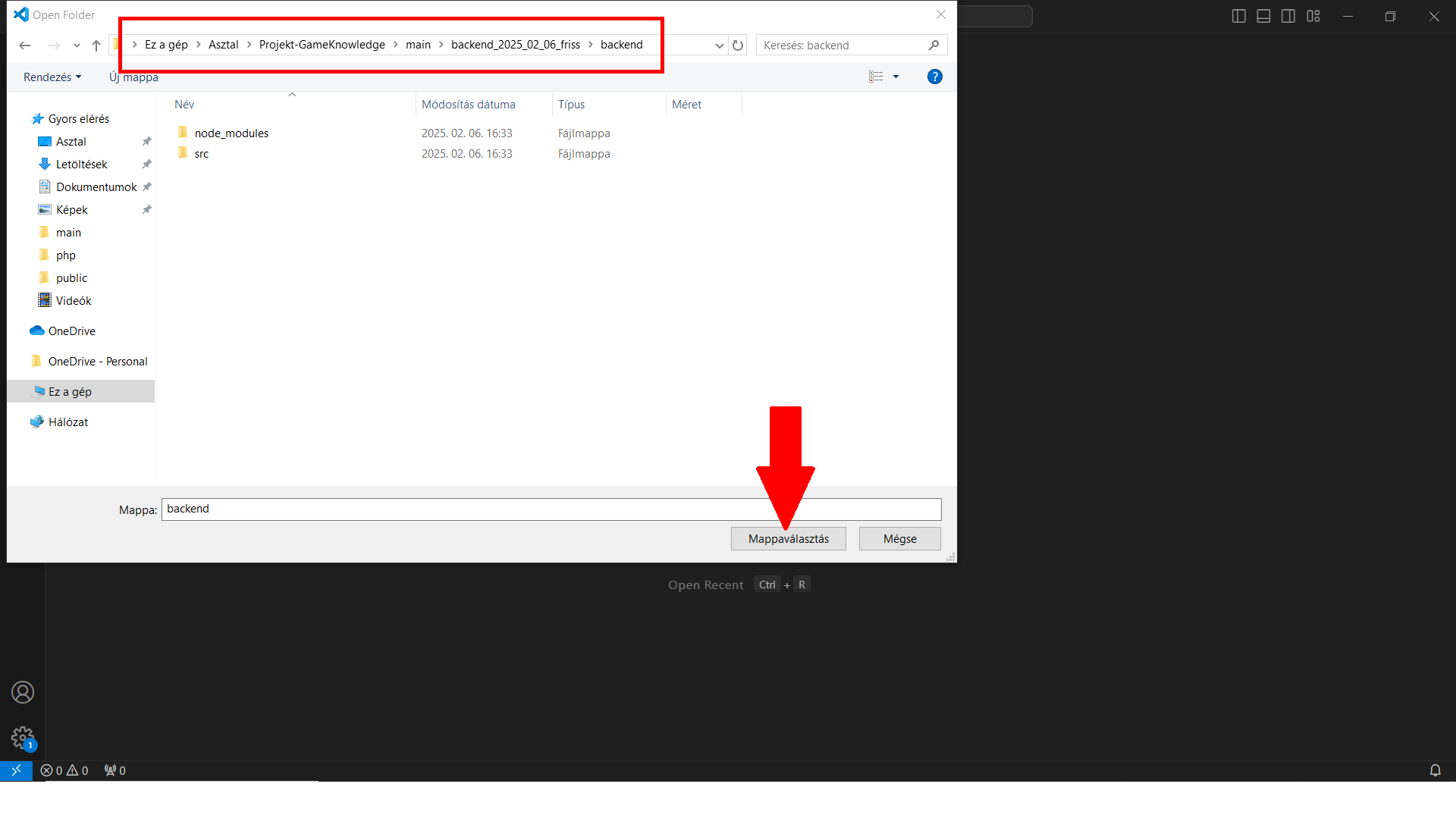
1. *kép*
2. A felugró ablak jobb oldalán válaszd ki, hogy hova szeretné kicsomagolni a fájlt (az alábbi kép jelenleg az Asztalt válassza), majd kattintson az OK gombra!



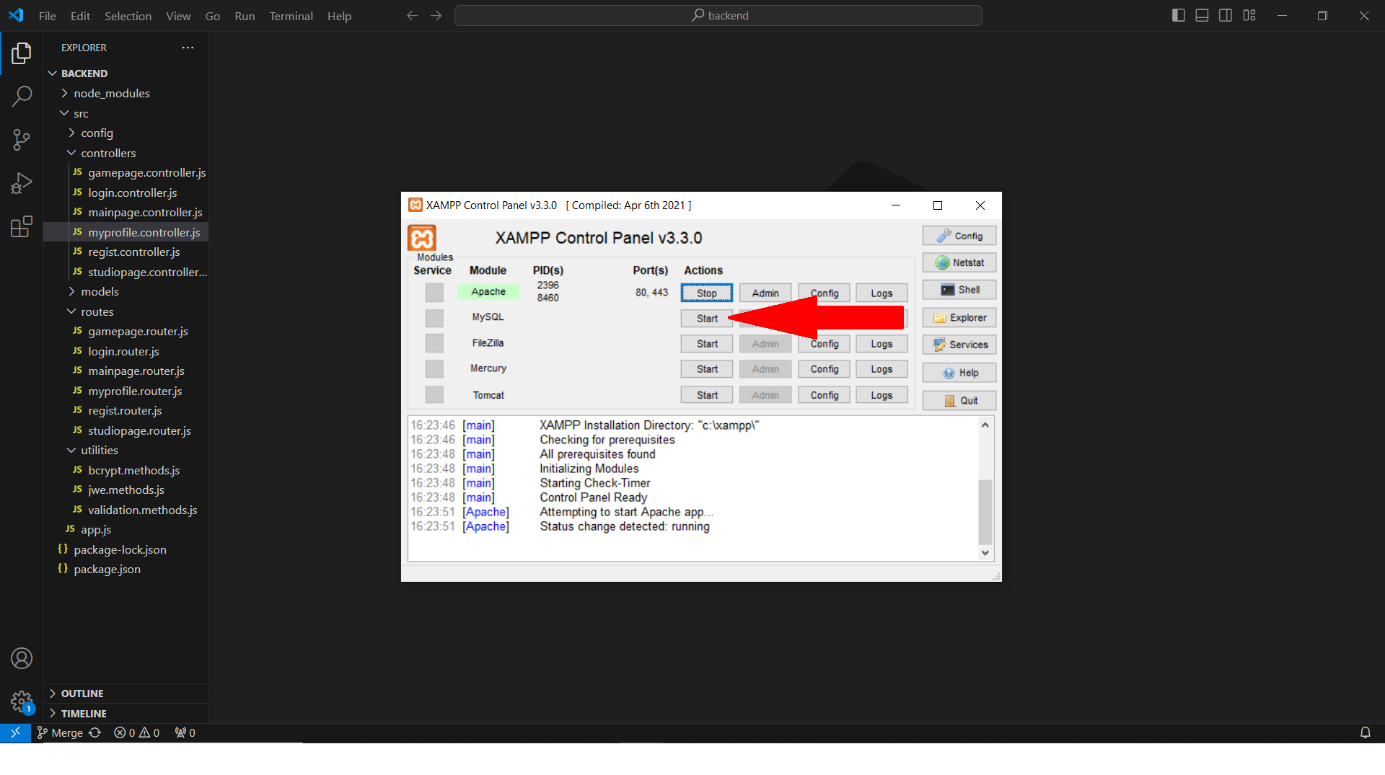
1. *kép*
2. Nyissa meg a Visual Studio Code programot és válassza ki az alábbi opciót az alábbi kép alapján!



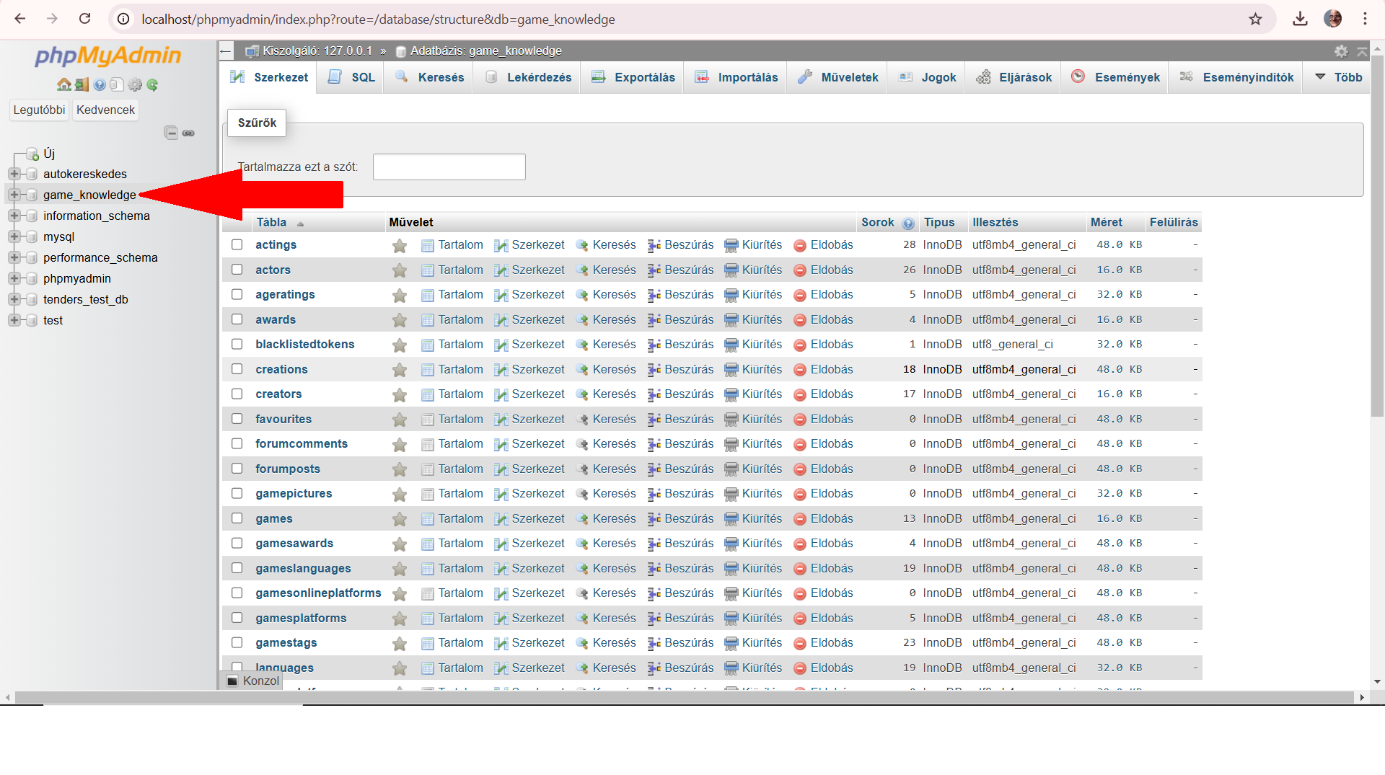
1. *kép*
2. Válassza ki a kicsomagolt mappát, majd azon belül is a backend mappát és nyissa meg a Visual Studio Code-ban!



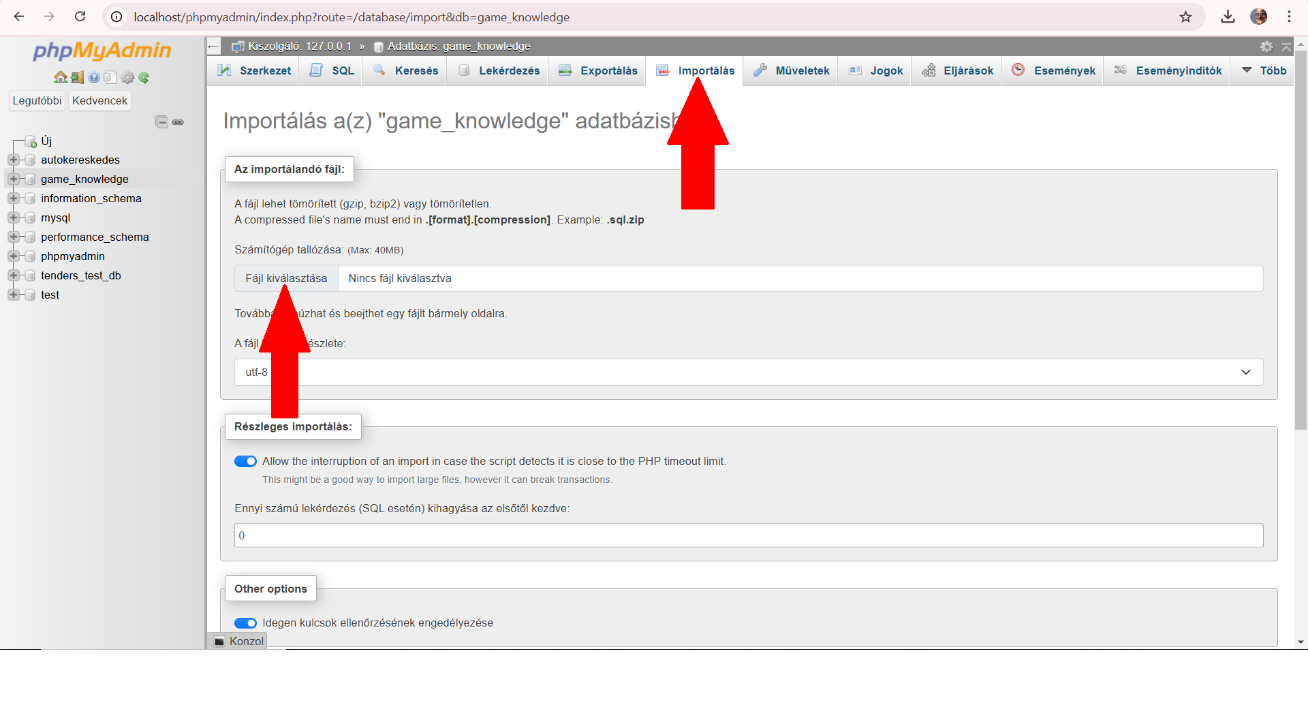
1. *kép*
2. Nyissa meg a XAMPP nevű programot, és indítsa el az Apache és MySQL modulokat az alábbi kép alapján!



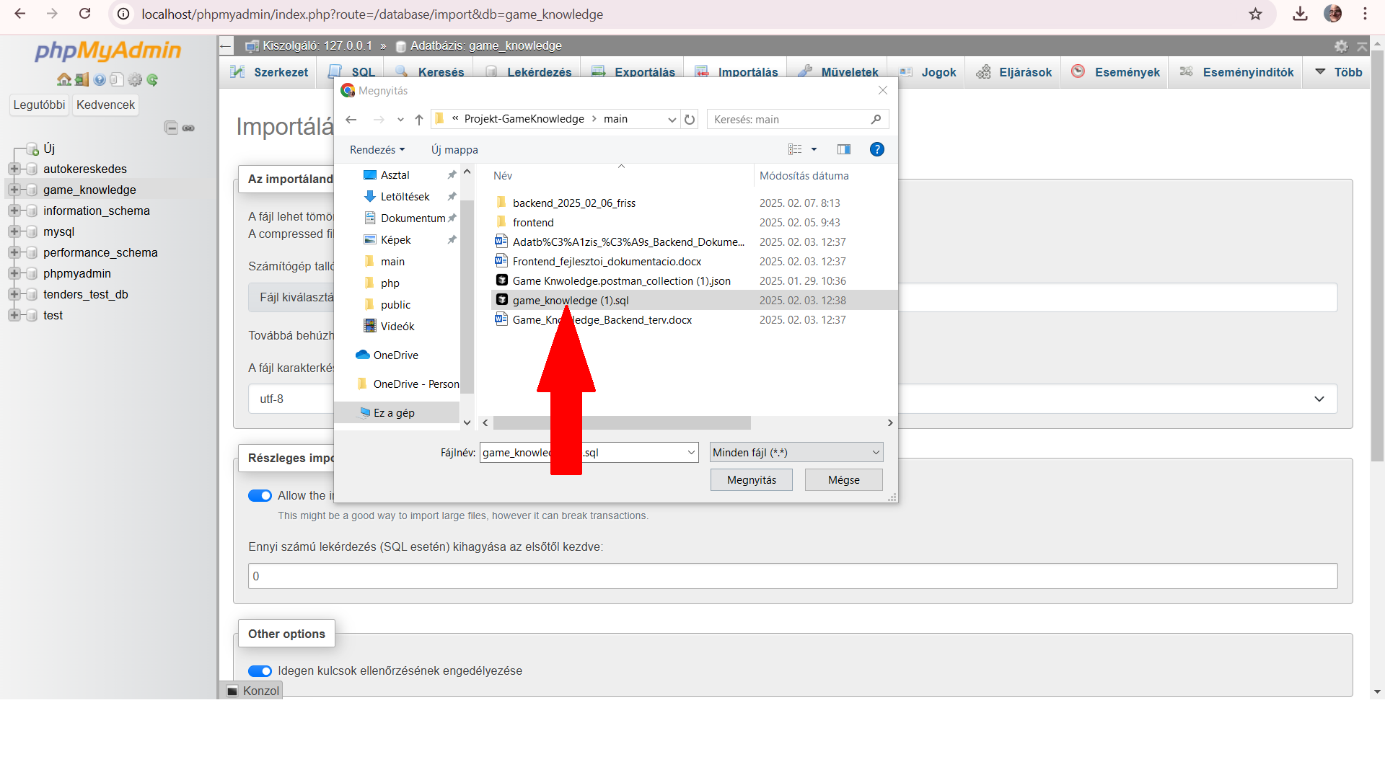
1. *kép*
2. Nyisson meg egy böngészőt, és illessze be a keresőbe az alábbi linket:  
   http://localhost/phpmyadmin/  
   Ezzel megnyitja a saját helyi localhost-át amire szüksége lesz.
3. A bal oldalt léteznie kellene már egy üres adatbázisnak game\_knowledge névvel, nyissa meg. Ha nem létezik akkor hozza létre az Új opcióval és nyissa meg.



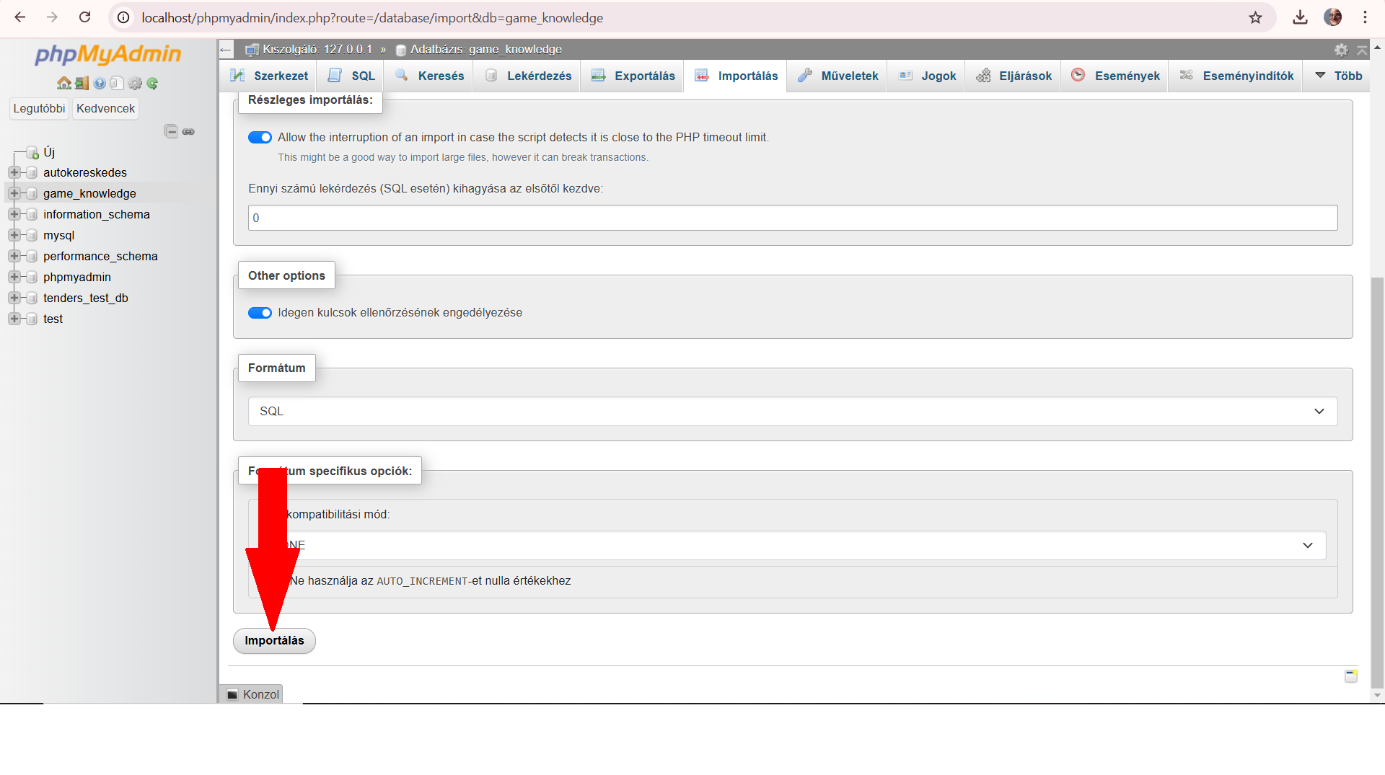
1. *kép*
2. A fenti opciók közül válassza ki az Importálás opciót!



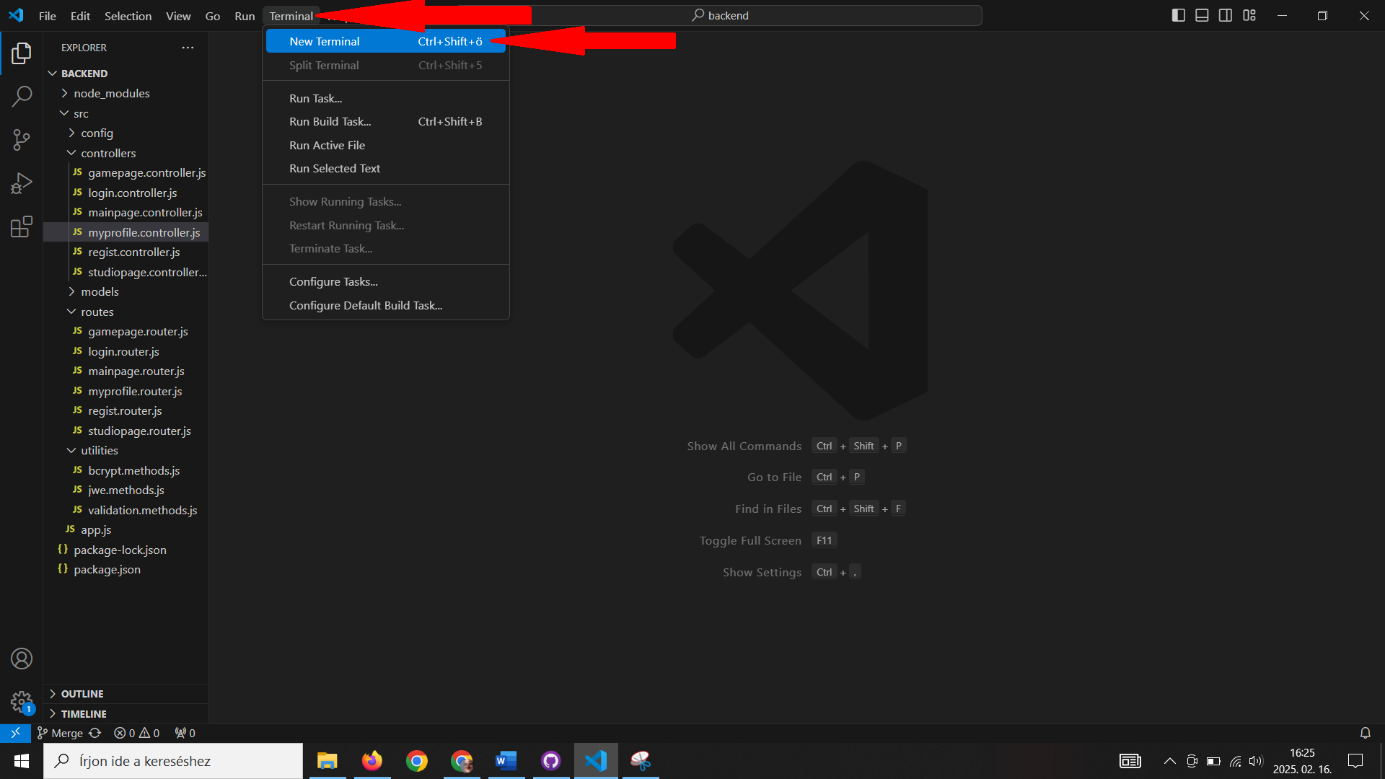
1. *kép*
2. Keresse meg a forrásfájlok közt a game\_knowledge.sql fájlt majd válassza ki!



1. *kép*
2. Görgessen le és importálja a kiválasztott fájl tartalmát az üres adatbázisba az Importálás opcióval!

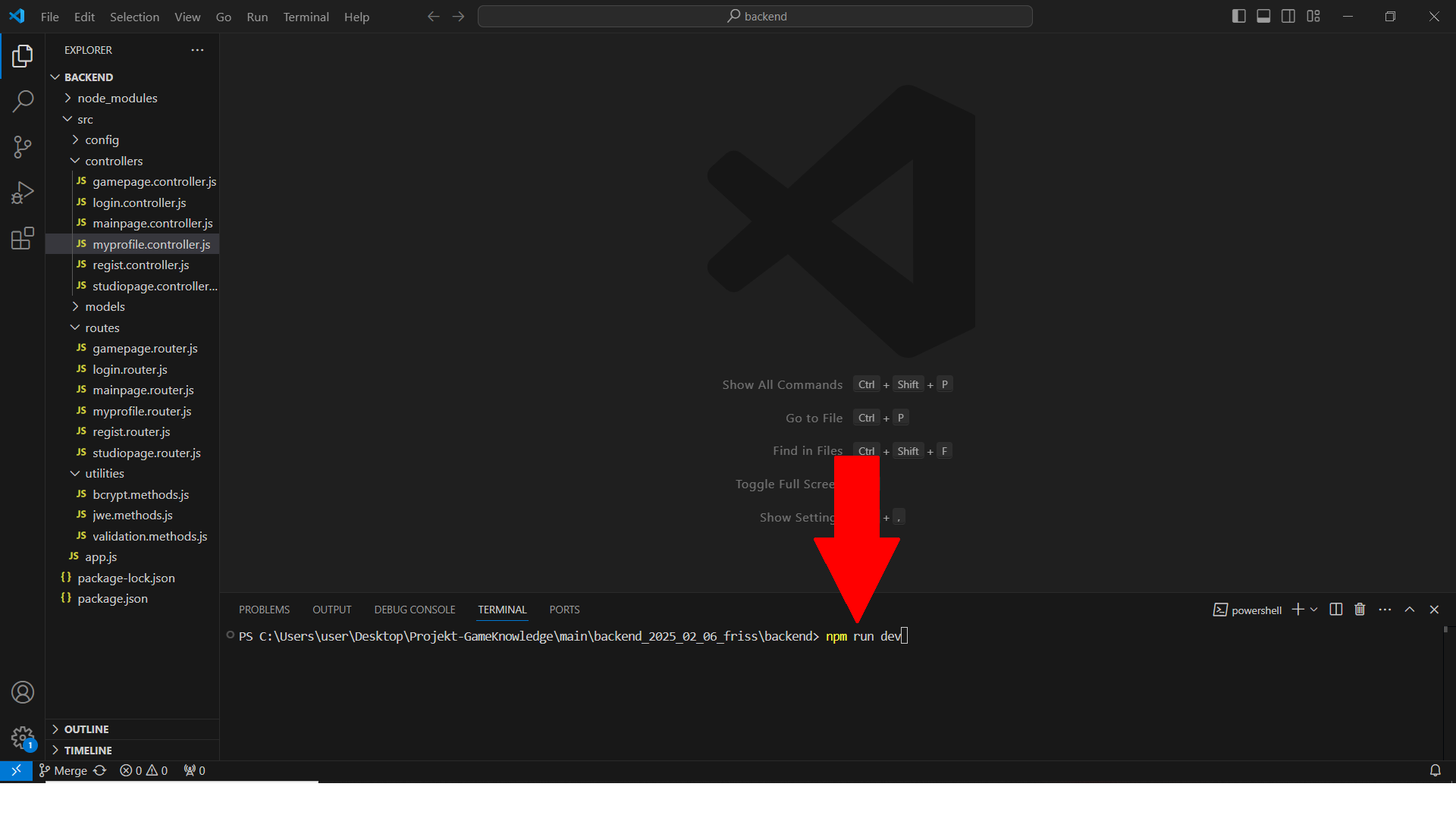


1. *kép*
2. Térjen vissza a Visual Studio Cobe-ba (ahol megnyitotta a backend-et) és válassza ki a Terminál opciót az alábbi kép alapján, majd ezen belül is a New Terminal opciót!



1. *kép*
2. Az alul megjelenő terminálba írja be a következő parancsot: npm run dev

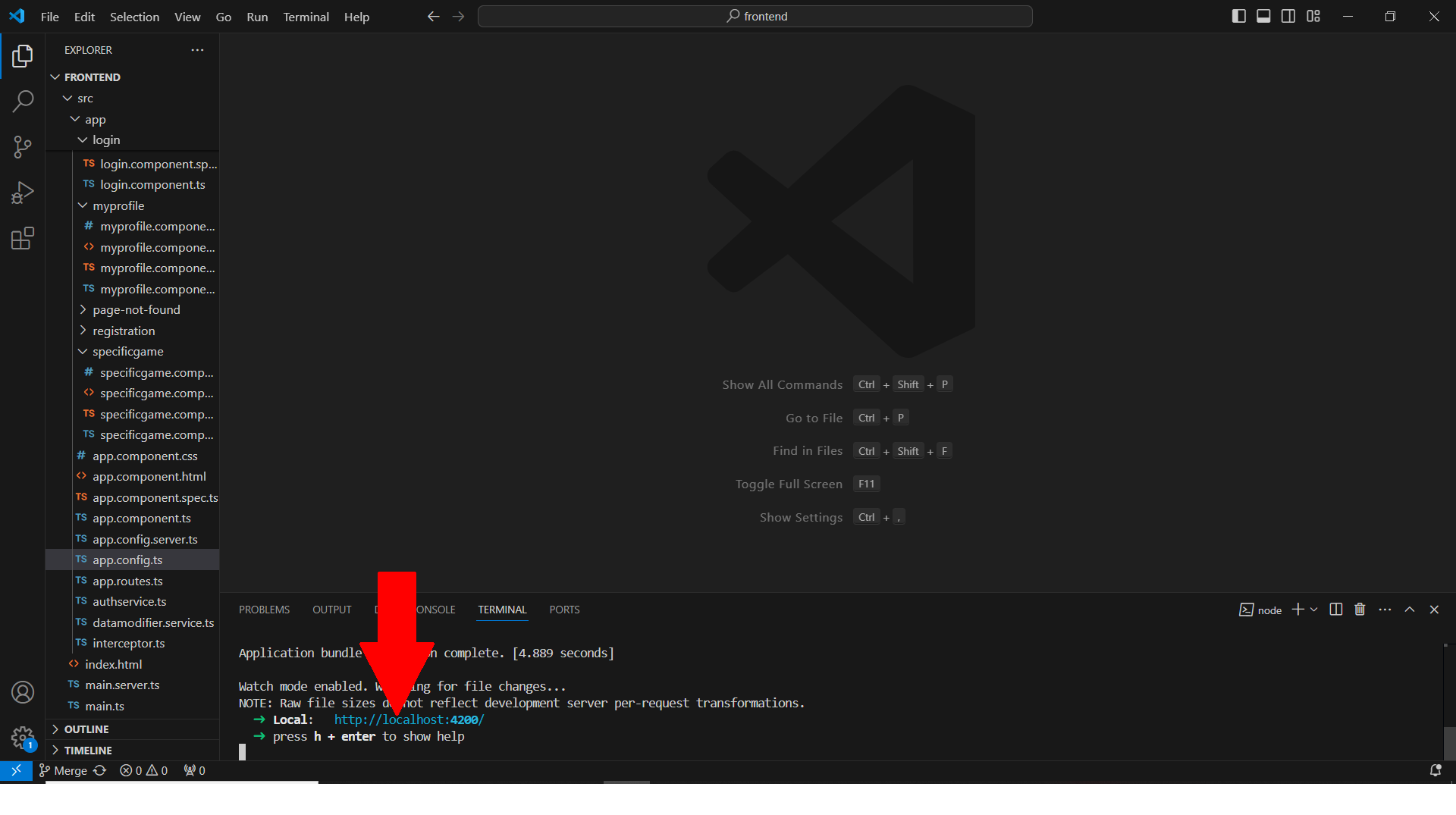
Ezzel elindítja a backend szervert, amely kapcsolatban áll az adatbázissal, és a weblap műveleteit fogja végezni.



1. *kép*
2. A backend szerver leállítása a CTRL + C billentyű kombináció leütésével állítható le.
3. Az előzőekben letöltött és kicsomagolt állományból (lásd a *3. képen*), válassza ki és nyissa meg Visual Studio Code-ban a frontend mappában található frontend állományt (lásd a *6. képen*)!
4. Válassza ki ismét a Terminal, azon belül is a New Terminal opciót (lásd a *12. képen*)!
5. Az alul megjelent terminálban (lásd a *13. képet*) gépelje be a következő parancsot: ng serve

Ezáltal elindul localhost-on az angular szervere.

1. Kattintson az alábbi képen ábrázolt localhost-os linkre, amely megjelent a terminálban, és a weboldal megjelenik az ön által alapértelmezetten használt böngészőjében.



1. *kép*

Megjegyzés: Az Angular terminálban előfordulhatnak ismeretlen eredetű hibaüzenetek. Ezek a működést nem gátolják meg!

# Frontend fejlesztői dokumentáció

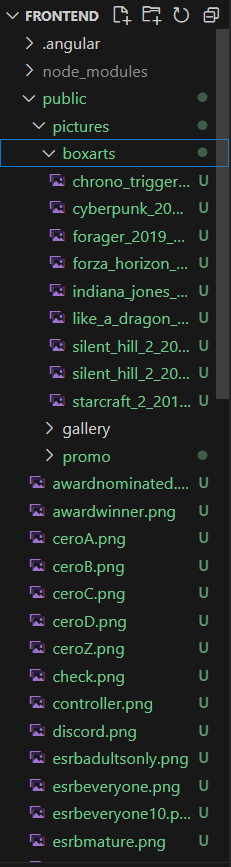
## Általános információk

A weboldal frontend része az Angular 18-as verziójában készült, amely keretrendszer azért lett kiválasztva mert biztosítani tudja számunkra számos funkciókat (pl.: a weboldal localhost-on való futtatását vagy a felmerülő kritikus hibák esetén a nagyobb visszalépések lehetőségét). A keretrendszerben megalkotott frontend tartalmaz bootstrap elemeket és osztályazonosítókat valamint statikus képi és ikon fájlokat. Mindezek mellett a weboldal frontend része úgynevezett standalone komponensekből épül fel, amelyek összessége alkotja az oldal(ak) felépítéseit, tartalmait, funkcióit és stílusait.

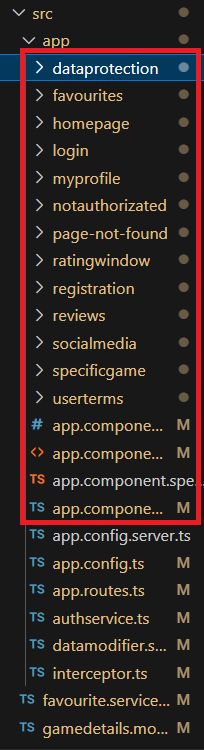
## Mappastruktúra

 Baloldalt a frontend mappa struktúrája látható. Fentről lefelé az első mappa tartalmazza azon szükséges elemeket amely az angular keretrendszer megfelelő működéséhez szükségesek, valamint a cache.

Következik a node\_modules mappa, amely szükséges a localhost-on való működéshez, ugyanis az ehhez tartozó mappák és fájlok itt vannak tárolva.



A public nevű mappa tartalmazza a statikus képeket amelyeket használ a frontend egyes helyeken (pl.: navigációs sáv kereső nagyító ikonja). Továbbá tartalmazza a pictures nevű mappát, ezen belül is pedig 3 különböző mappát, a boxarts-okat, a gallery-t és a promo-t. A boxarts mappa tartalmazza a játékok borítóképeit. A gallery-ben találhatóak azok a képek amelyek az adott játékhoz tartozó jeleneteket, játékmenetet valamint esetleges mechanikákat mutatják. A promo mappa tartalmazza azon játékok promóciós képeit amelyek a kezdőlapon a slideshow-ban jelennek meg.

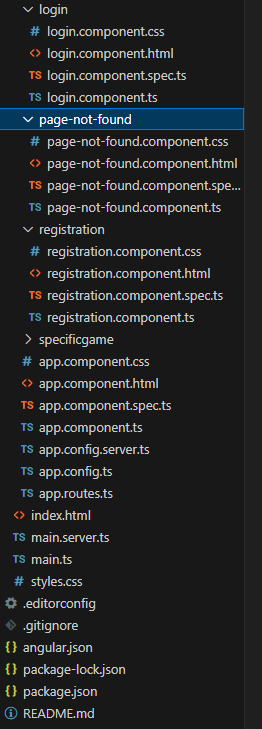


Ezt követően az src nevű mappa, amely tartalmazza a főbb fájlokat. Ezek többek közt a standalone komponensek (keretezve), mint a homepage, a login vagy a registration. Ezek tartalmazzák maga az oldalnak(oldalaknak) a tartalmukat, stílusait és funkcióikat.

Ezen felül itt található az úgynevezett app.routes.ts fájl amely a routing konfigurációját tartalmazza a frontend-nek.

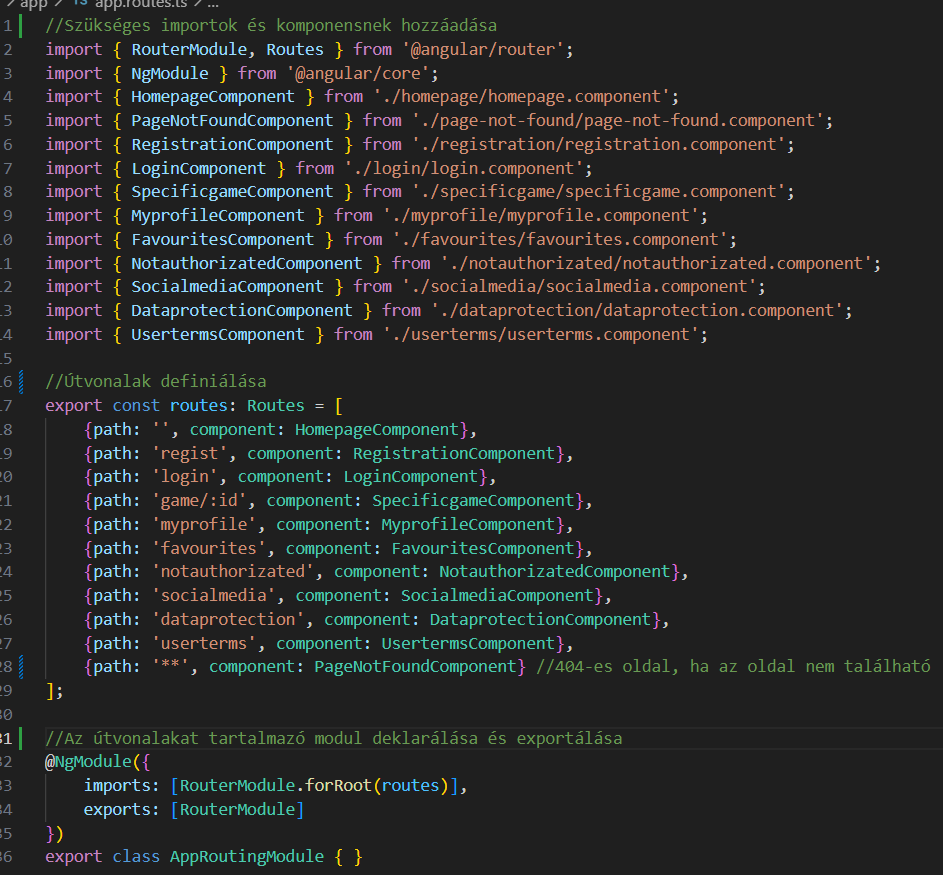
Majd következnek az app. fájlok mint az app.component.html, az app.component.ts, vagy az app.component.css. Ezekben a fájlokban találhatóak azok a weboldal elemek, stílusok és funkciók amelyek az összes más különálló (standalone) komponensekre érvényesek kell hogy legyenek. Példának okáért ilyen a navigációs sáv, annak tartalmai és funkciói amelyeknek meg kell jelenniük és működniük kell az összes oldalon. Az ezeket követő fájlok (angular.json, package.lock.json stb.) már az angular megfelelő működéséért felelnek. Ezek óvatlan módosításai, vagy eltávolításuk kritikus hibákat eredményezhetnek!

## Standalone komponens alapú felépülés



Ebben a részben a weblap standalone, avagy különálló komponenseiről olvashat. A fenti képen láthat egy részletet a mappastruktúrából. A komponensek a következők: login, és a hozzá tartozó többi fájl, a page-not-found, a registration, az app. fájlok az app.routes.ts kivételével, valamint a specificgame. Ezen komponensnek tartalmazzák a weblap oldalainak megfelelő tartalmát, stílusát és funkcióját. A login komponens a bejelentkező felületet valósítja meg, a registration a regisztrációs felületért felel, a page-not-found amikor valamilyen oknál kifolyólag egy adott oldal nem elérhető így alapértelmezetten átirányításra kerül erre az oldalra, és a specificgame komponens amely mindig, a felhasználó által kiválasztott specifikus játék oldalát valósítja meg. Ezen komponensek függetlenek egymástól, mégis képesek együtt dolgozni. Egy átláthatóbb megoldás hosszútávon, azonban ha a későbbiekben szükséges módosítások megkövetelik valamilyen modul, más komponens vagy csomag importálását, akkor tudni kell, hogy mely más komponenseknek is szüksége van e az újonnan importált dolgok valamelyikére, és ha igen akkor azt minden egyes komponensben importálni szükséges. A komponensek 4db fájlt tartamaznak: HTML, CSS és 2 Typescript fájlt. Esetleges módosítások esetén önnek csak a .spec.ts-re végződő fájl kivételével bármelyiket módosíthatja, ugyanis a 4 fájl közül csak 3 az, amelyik a komponens tartalmának manipulálására szolgál, a .spec.ts-re végződő fájl egy alapvető konfigurációt tartalmaz melyet az angular hoz létre a komponens létrehozásakor annak céljából, hogy a komponens megfelelően működjön.

## Routing konfiguráció

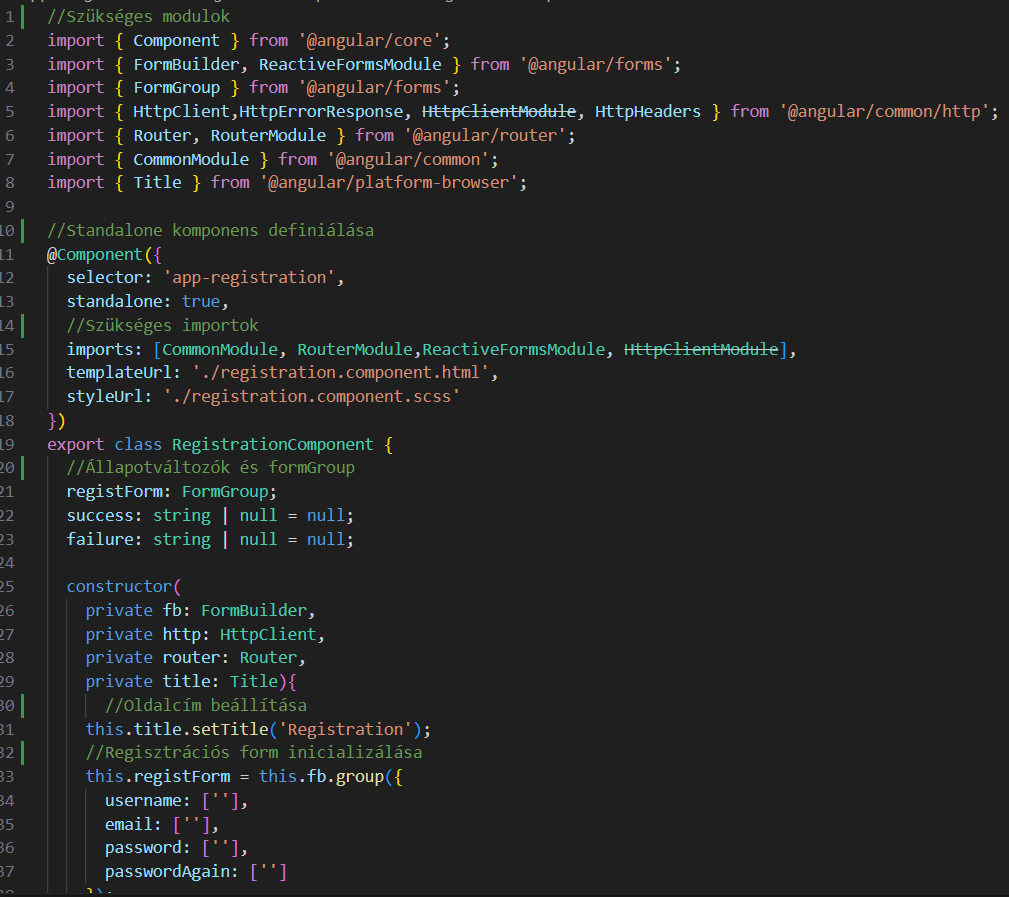


A képen a routing konfigurációs fájl tartalmát látja. Felülről lefelé haladva olvashatja az információkat a működéséről. A fájl az importokkal kezd. Ezek az importok szükségesek ahhoz hogy az útvonalak működhessenek. Az első kettő import ágyazza be azokat a szükséges Angular által nyújtott csomagokat amelyek szükségesek a működéshez. Az összes további import a standalone komponenseket importálják amelyek az útvonalak definiálásához szükségesek hogy melyik útvonal melyik komponensre vezessen. Ezután következnek az útvonal definiálások. Létrehozunk egy Routes tömböt amely tárolja az útvonalakat. A tömbön belül pedig létrehozzuk ezeket. A „path” szóval adhatjuk meg a képen látható idézőjelek között az általunk elnevezett útvonalat. Ezt adjuk meg a komponens HTML részében létrehozott linknek az úgynevezett „routerLink” attribútummal amiről a továbbiakban láthat képet. Ezután a „component” szóval definiáljuk hogy az általunk megadott útvonal melyik komponensünkre mutasson. Ez így megy addig amíg el nem érünk az utolsó komponensünkhöz. Ezután a következő útvonalnak beállítjuk a „\*\*” értéket amellyel jelezzük hogy ez az útvonal felelős azért, hogyha az oldalunk nem találja a megfelelő irányt, akkor alapértelmezetten ezt az irányt kövesse. Látja is a képen hogy jelen esetben az oldal ebben a helyzetben a „PageNotFoundComponent”-et fogja elénk tárni. Ezzel lekezeljük könnyedén azt az eshetőséget amikor valamilyen oknál kifolyólag nem találjuk a keresett oldalt. Fontos azonban, hogy ez az útvonal beállítás mindig kerüljön a legvégére az útvonalak elemeinek sorában! Különben szintaktikai hiba miatt nemfog működni. Ezután következnek az utolsó lépések. Ezen lépések következtében a képen látható kóddal a fájl tartalmát modullá deklaráljuk, és exportáljuk. Ezzel válik az egész fájl teljesen müködőképessé. A következő példa képen pedig láthatja a helyes alkalmazását az útvonalaknak a komponensnek HTML kódrészeiben:

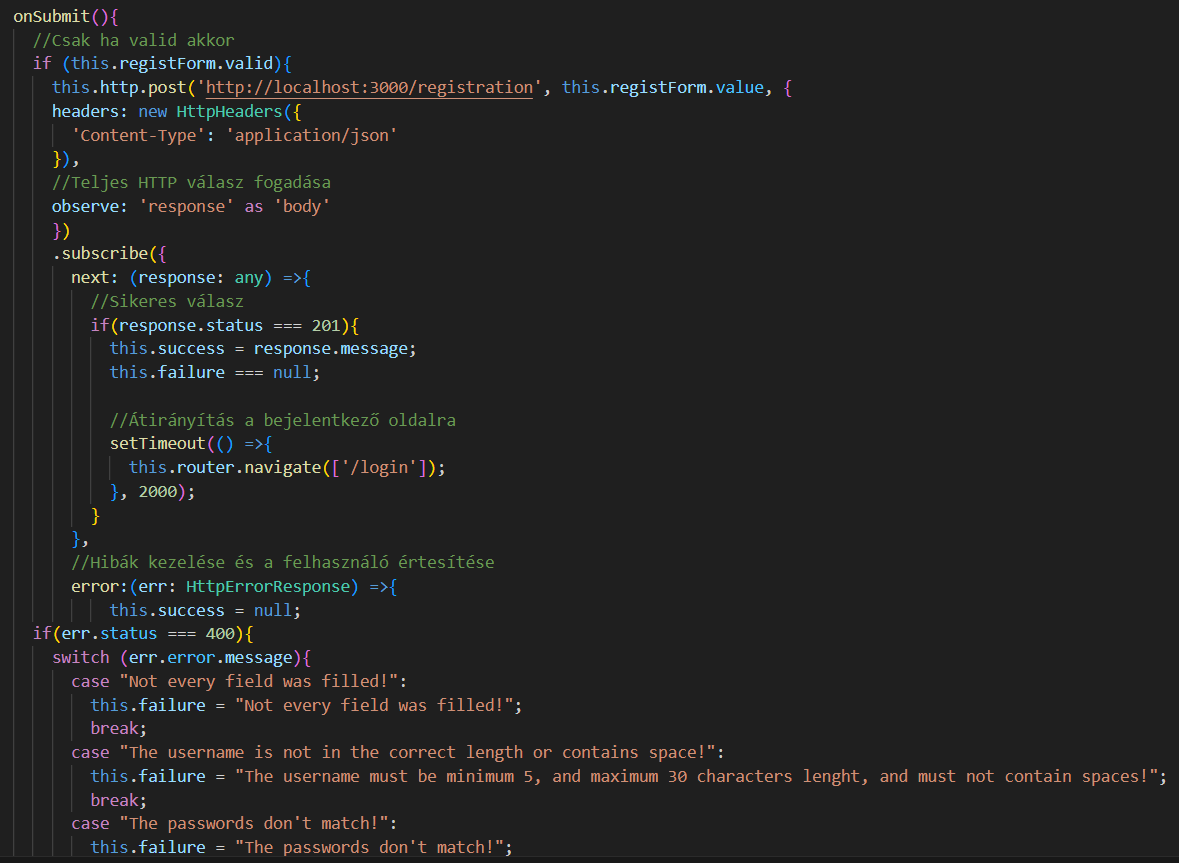


## Regisztráció

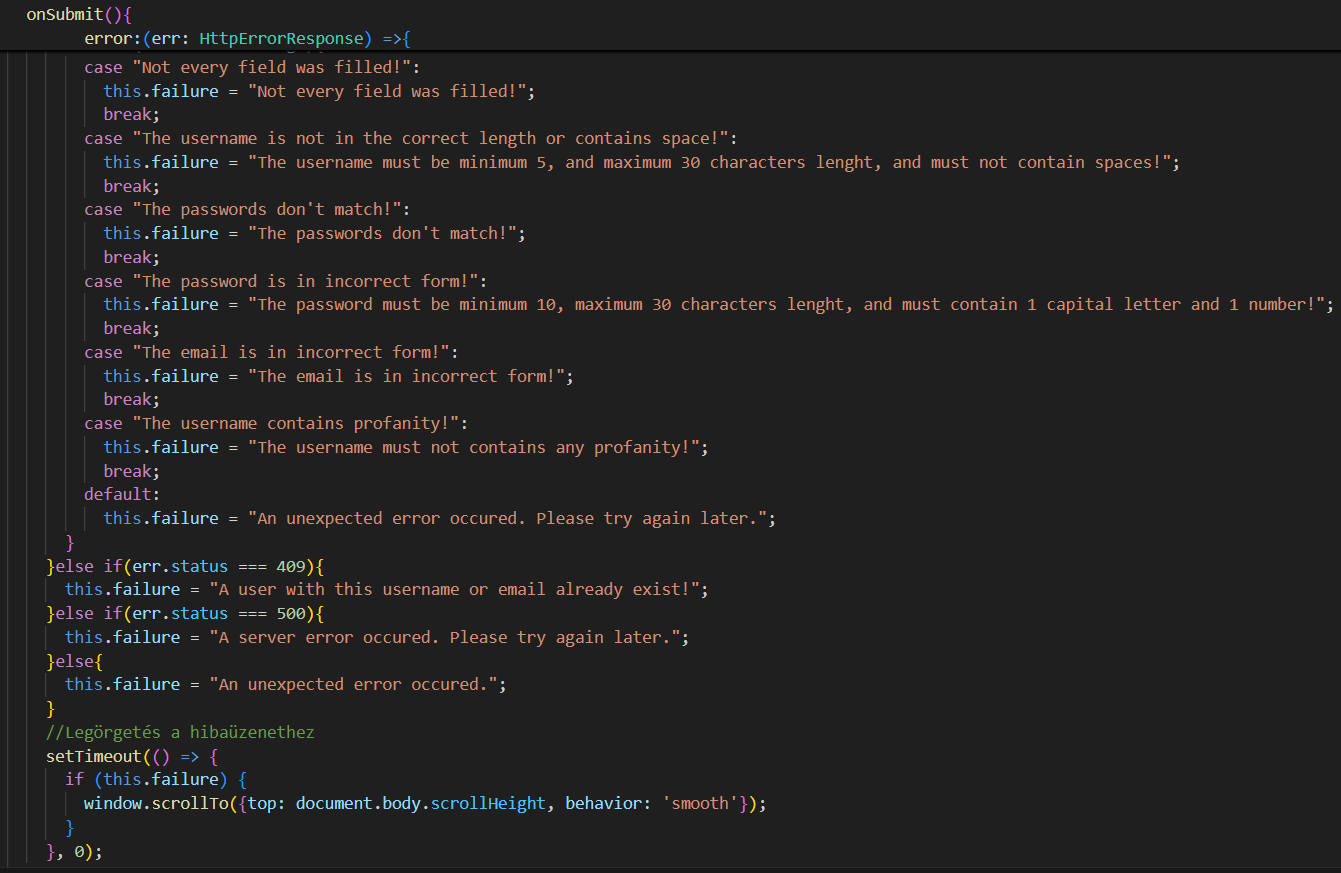
A következő részben a regisztráció működéséről olvashat. A regisztrációt a „registration” standalone komponens végzi, pontosabban a komponens typescript fájla.



A képen a „registration” komponens typescript fájlának egy részét láthatja. Fentről lefelé haladva, először megtörténnek a szükséges csomagok és modulok importálása. Majd tovább haladva az Angular automatikusan előállított standalone komponens definícióját láthatja. Fontos hogy az ebben a részben szereplő „imports” részben kötelező a képen látható csomagok és modulok még egyszeri importálása a megfelelő működés érdekében. Ugyanis ezek a modulok és csomagok teszik lehetővé a reaktív űrlapok használatát valamint az adatkommunikációt. Tovább haladva lefelé, létrehozzuk a FormGroup-ot és az állapot változókat. A „registForm” tárolja a felhasználó által bevitt adatokat, majd a „success” és a „failure” változók tárolják az üzeneteket amelyek értesítik a felhasználót a regisztráció menetéről. A kontruktor injektálja a szükséges szolgáltatásokat, beállítja az oldal megfelelő címét és létrehozza a regisztrációs űrlapot az üres mezőkkel.

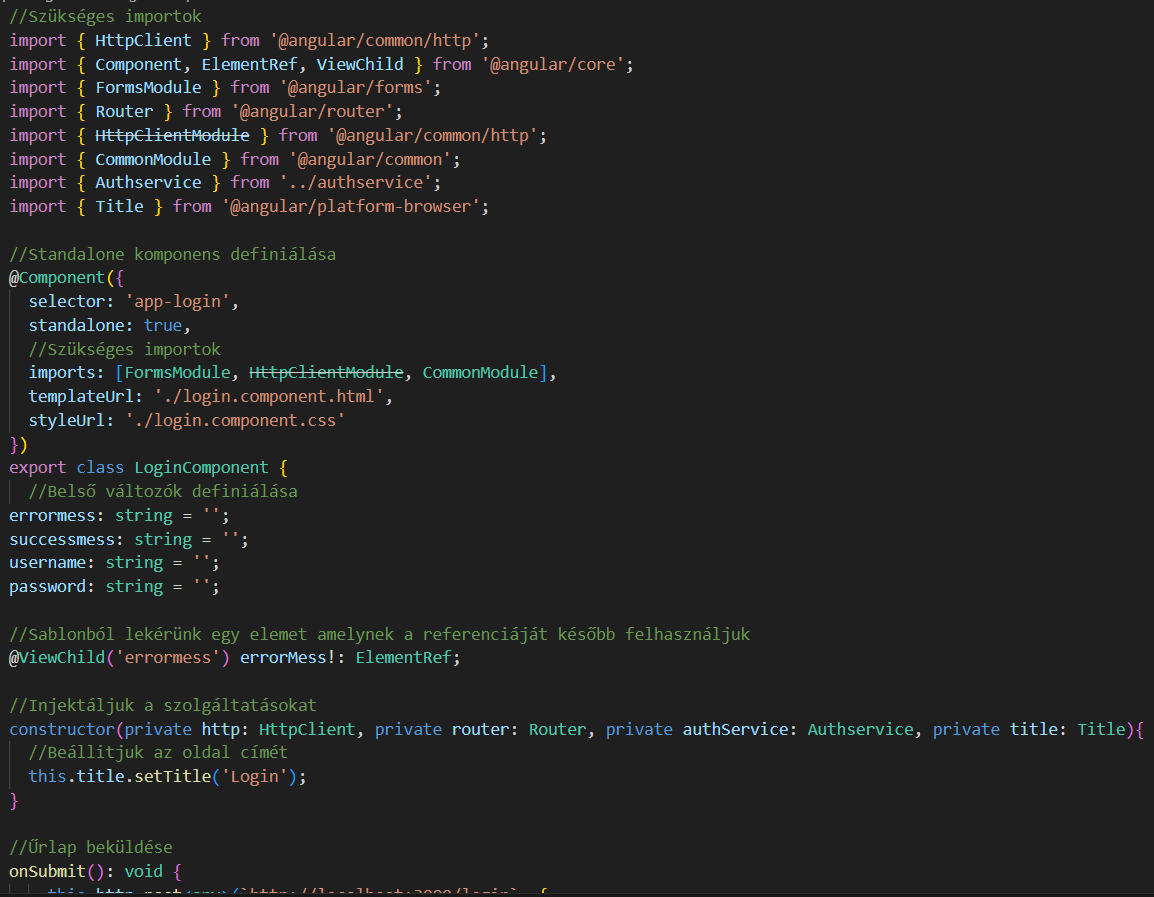


A következő képen láthatja a további részét a regisztrációs typescript fájlnak. Itt, az „onSubmit” metódus csak akkor küldi el a POST kérést a képen látható URL-nek hogyha a felhasználó által megadott adatok megfelelnek a szabályoknak. Ha nem felelnek meg akkor értesítjük a problémáról amelyet láthat ezen a képen is, és fog látni a további képen. A „HttpClient” segítségével elküldjük az adatokat a végpontra, a kérés header-ében megadjuk hogy JSON formátumú adatot küldünk a backend logikájának megfelelően és az „observe: response” opcióval pedig a teljes http választ figyeljük, így elérhetjük a státuszkódot és a kód alapján könnyedén kezelhetjük a hibákat. Ha a válasz státusza a 201-es akkor a „success” változóba tároljuk az üzenetet és visszaadjuk azt a felhasználónak, majd átnavigáljuk a bejelentkezés oldalra. Hiba esetén pedig a státuszkódoknak megfelelően értesítjük a felhasználót a hibáról.



Itt láthatja is a teljes hibakezelést amely egy switch case feltételt használ valamint a görgető parancsot amely hiba esetén a hiba kiírásához görgeti az oldalt.

## Bejelentkezés



Csak úgy mint előzőleg, itt is végrehajtásra kerülnek a szükséges csomagok és modulok importálása. A http kérésekért felelősek a „HttpClient” és „HttpClientModule” csomagok, az „ElementRef” és „ViewChild” segítségével lehetőségünk van a HTML-ben található elemek referenciájának lekérésre a közvetlen manipuláció érdekében, a FromsModule felel az űrlapkezelésekért, a Router a navigációt teszi lehetővé, az Authservice ez egyedi szolgáltatás amely a hitelesítéseket végzi és a Title amely az oldal címének beállításához szükséges. Következik ismét a standalone komponens definíciója, majd ismét néhány belső változók definiálása. A „errormess” és „successmess” a sikeres bejelentkezés valamint a sikertelen bejelentkezés üzeneteit tárolják, a „username” a felhasználónevet és a „password” pedig a jelszót. A „ViewChild” segítségével lekérünk egy HTML elemet amelyhez a „#errormess” lokális változó van rendelve és így később ezt az elemet manipulálhatjuk. Következik a konstruktor amelyben injektáljuk a szolgáltatásokat és beállítjuk az oldal megfelelő címét.



Következik az „onSubmit” metódus amely akkor hívódik meg amikor a felhasználó beküldi az űrlapot. POST metódussal elküldjük a „username”-t és „password”-t. Ha a kérés sikeres akkor lefut a next, és megvizsgáljuk hogy van e hiba. Ha nincs akkor a „success” változóba beállítjuk az üzenetet és meghívjuk az Authservice login metódusát és bejelentkeztetjük a felhasználót, majd átirányítjuk a kezdőlapra. Probléma esetén érkezik az error, megvizsgáljuk milyen státuszkód érkezett és az alapján értesítjük a felhasználót és legörget az oldal az üzenethez.

## Authservice(Kijelentkezés)

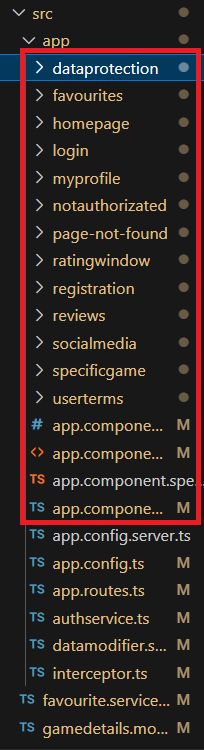
A következőkben az authservice.ts fájlról és a kijelentkezés működéséről olvashat. Ez egy egyedileg létrehozott szolgáltatás amely számtalan funkciót valósít meg. Felhasználói bejelentkezési állapot, felhasználónév és admin státusz tárolása és frissítése BehaviorSubject-ek segítségével. A token és felhasználói adatok kezelésének biztosítása, azaz adatok mentése és törlése a localstorage-ból valamint annak betöltése indításkor. A felhasználói fiók törlésére szolgáló HTTP DELETE metódus is itt található valamint ellenőrzi a kód futási környezetét hogy elkerüljük a szerveroldali hibákat.



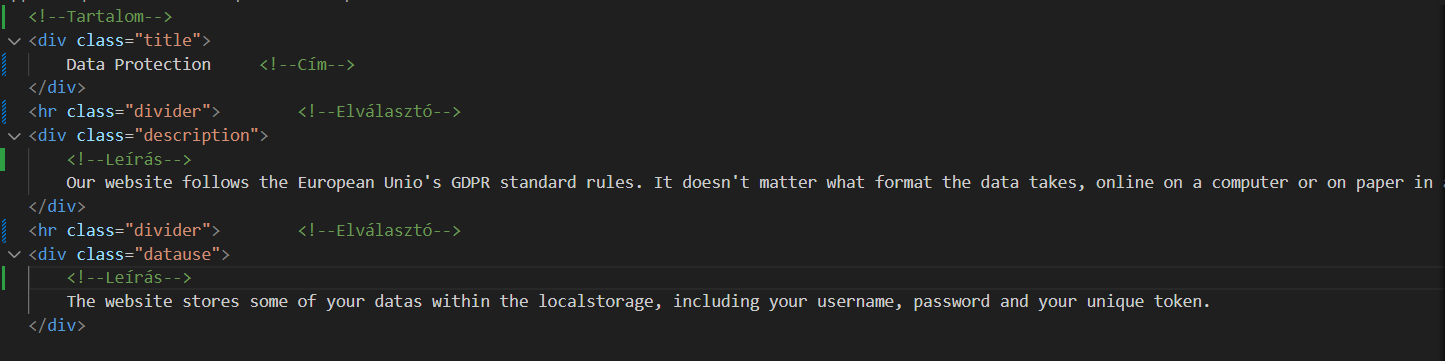
A képen a kijelentkezés működése látható. A logout metódus meghívásakor először ellenőrzi hogy a localstorage elérhető e. Ha igen akkor a korábban megkapott adatokat a localstorage-ból törli és alapértelmezettre visszaállítja.

## Standalone komponensek

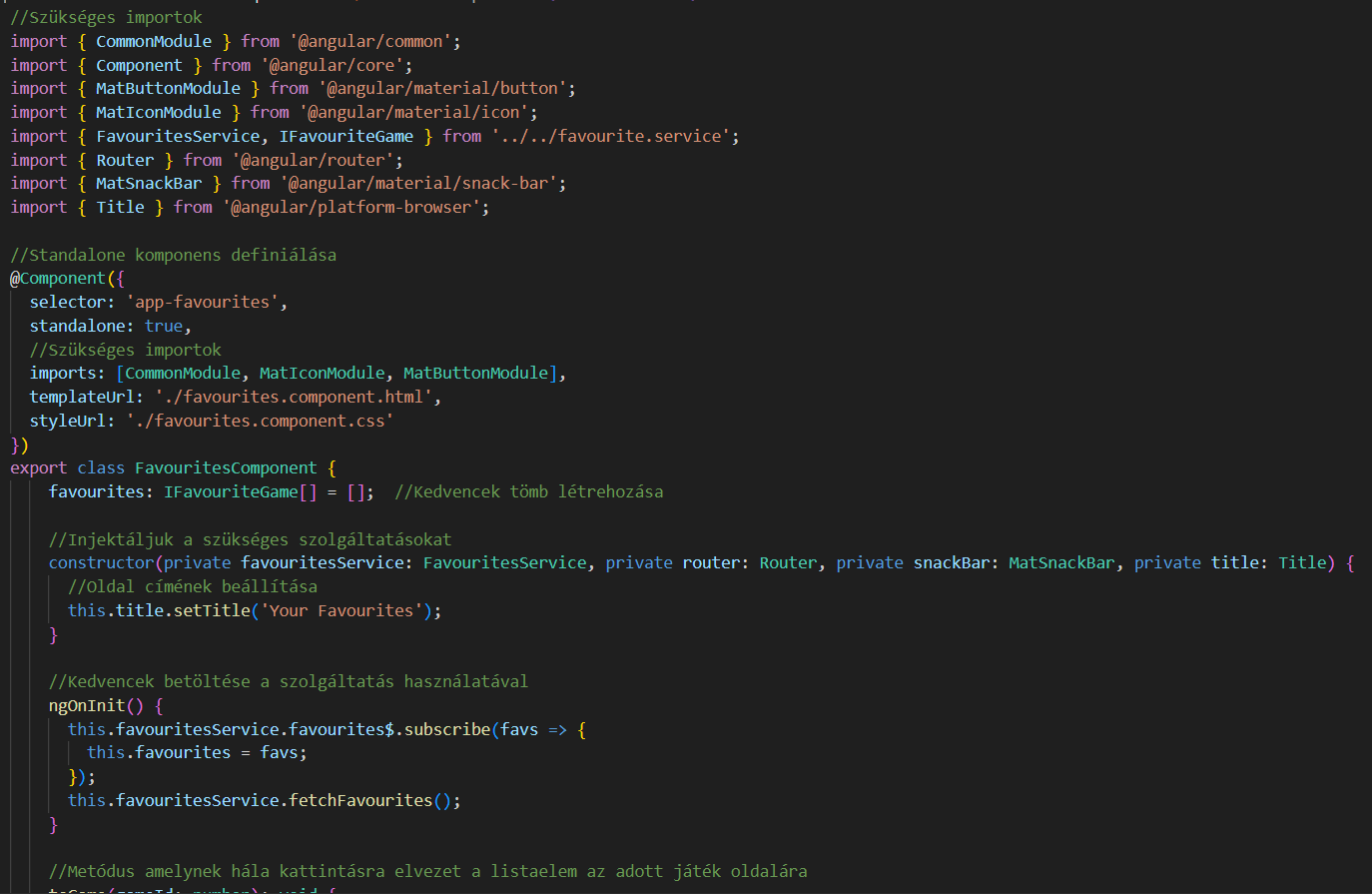
A következő részben a komponensekről olvashat.

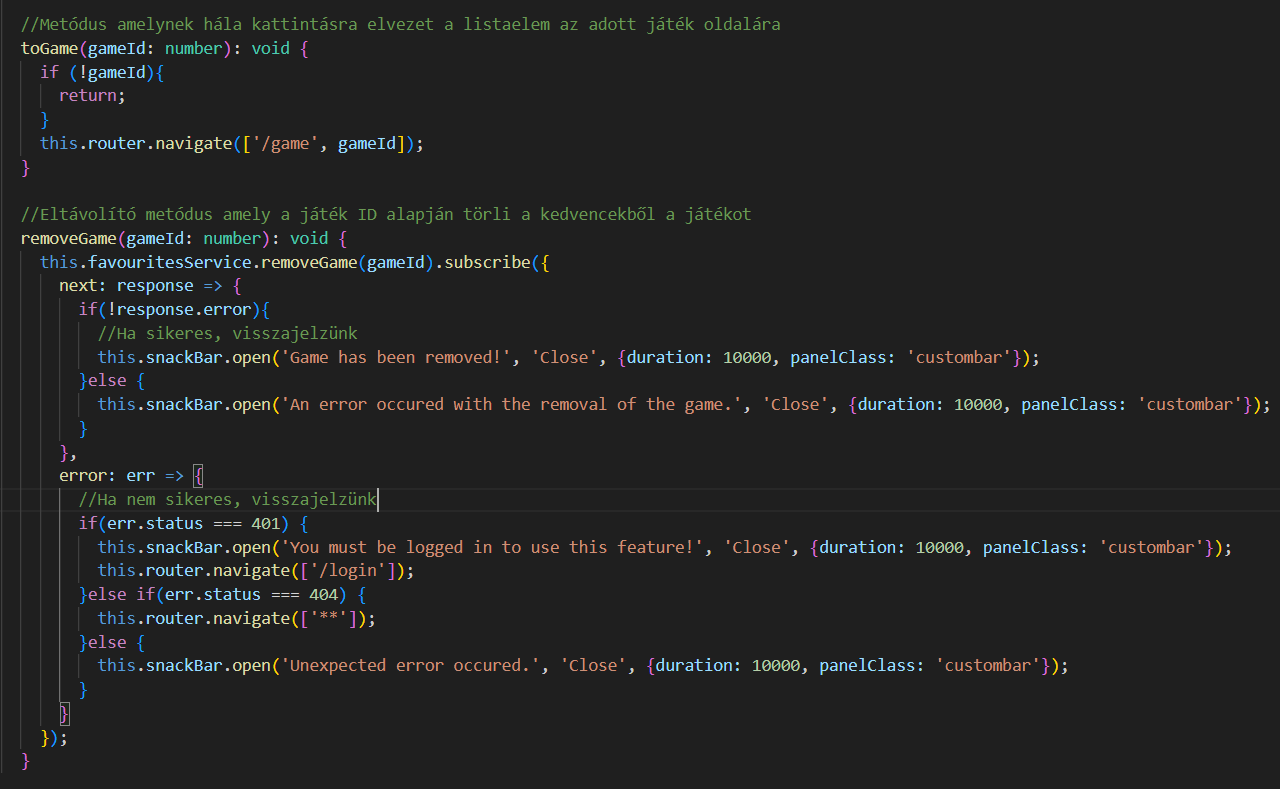


Felülről lefelé haladva tájékozódhat az egyes komponensek működéseiről. Az első komponens tehát a dataprotection komponens amely az adatbiztonsági leírásokat tartalmazza.

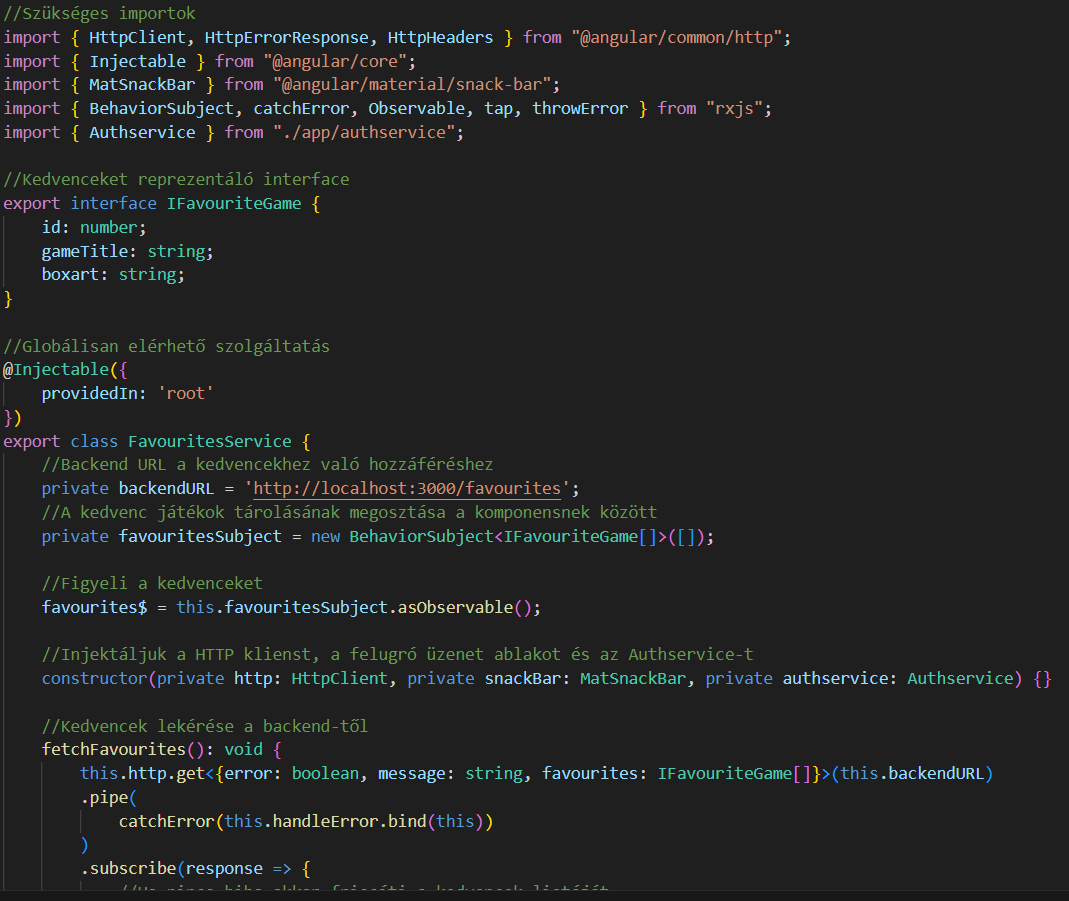


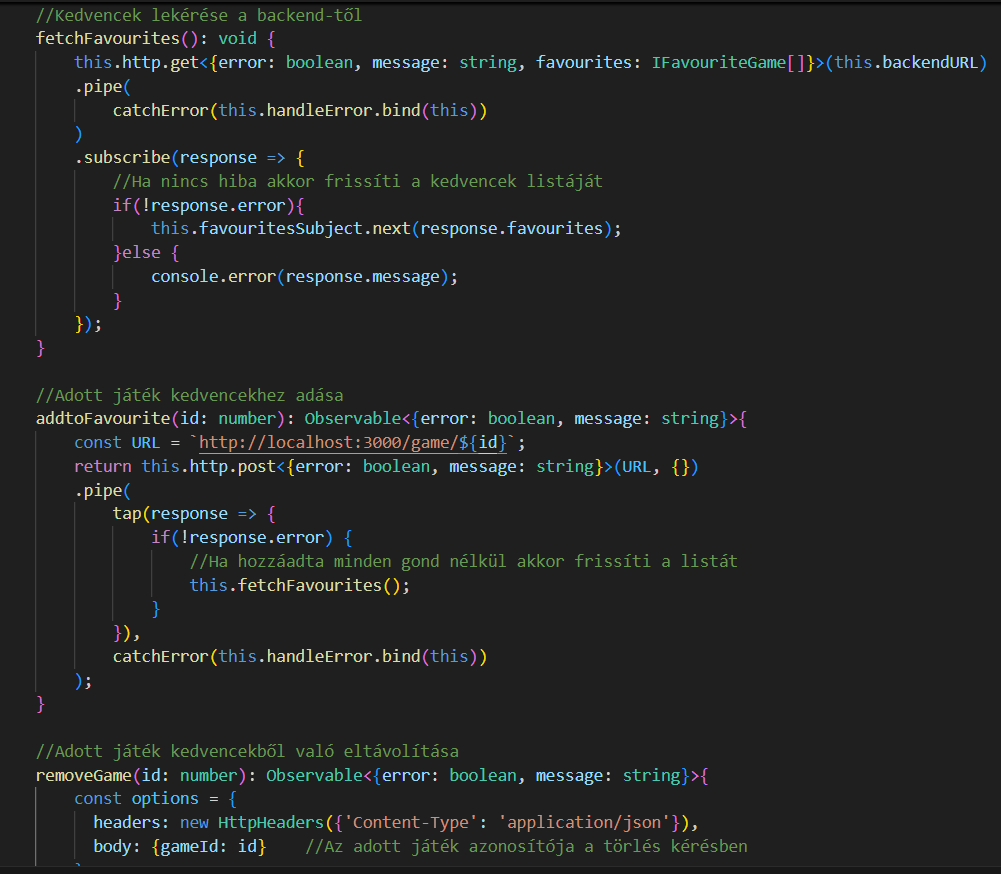
A typescript fájla csupán egyetlen beállítást rejt magában, az oldal címének megfelelő beállítását amelynek logikáját megtalálhatja az előző képek valamelyikében. A második komponens a favourites, avagy a kedvencek komponense. Ez a komponens felel a kedvencek lista helyes, reszponzív megjelenítéséért és működéséért.

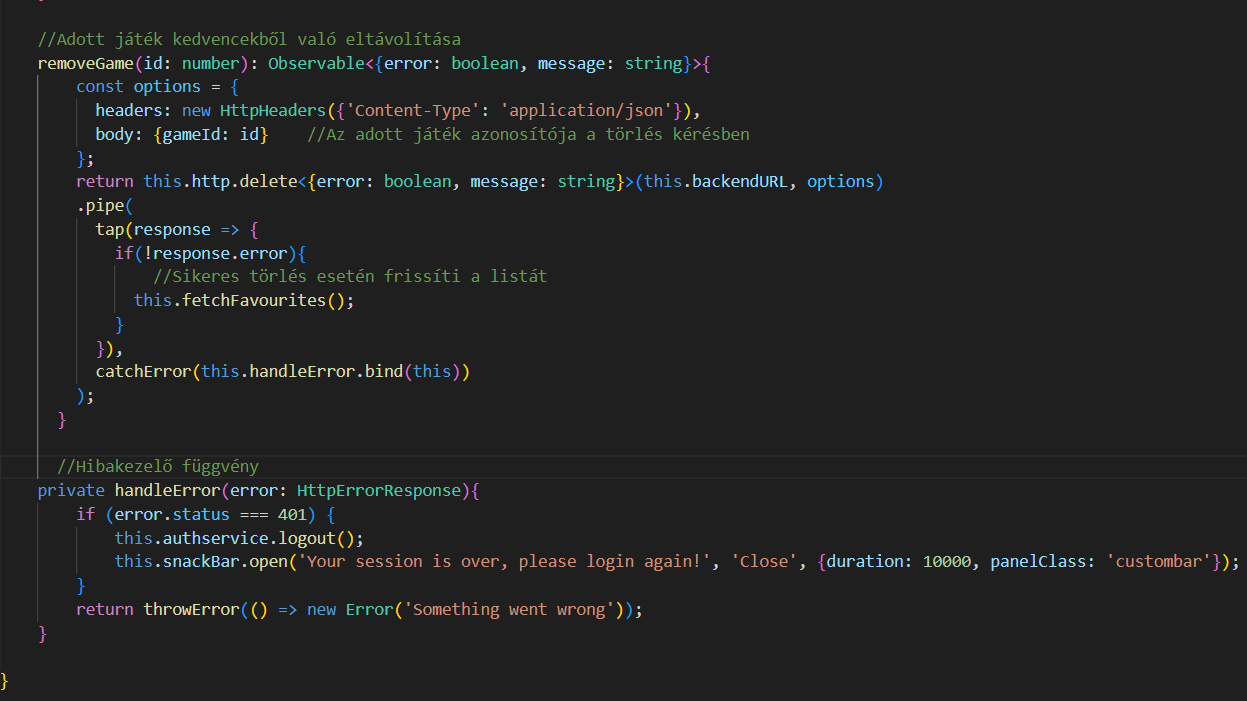




A felépítése követi a standalone komponens logikát, csak úgy mint az előzőekben először megtörténnek a szükséges csomag és modulok importálása, ezt követi a standalone definíció majd a működés logika. A logika úgy kezdődik hogy először létrehozunk egy üres kedvencek tömböt, majd egy kontruktorba injektáljuk a szükséges csomagokat és modulokat valamint megadjuk az oldal megfelelő címét. Ezt követően igénybe vesszük az egyedi kedvencek szolgáltatást amelyre azonnal kitérünk, és feliratkozunk rá, utána pedig frissítjük a tartalmat. A következő képeken látható a kedvencek szolgáltatás:







Amint láthatja, hasonlóan működik mint az authservice szolgáltatás. Itt is szükséges néhány csomag importálása hogy jól működjön a szolgáltatás. Továbbá itt található a backend végpontja amelyre küldésre kerülnek a kérések, megosztásra kerül az imént említett kedvencek tömb, majd ezután dinamikusan figyeli a kedvenceket. Szintén következik egy konstruktor a megfelelő csomagok injektálásával és utána itt került megírásra a kedvencek lekérdezése metódus amely frissíti a kedvencek listát hacsak nincsen valamilyen hiba jelen. Következik a kedvencekhez adás metódus amely a backend logikának megfelelően összeállítja az URL-t beleépítve az adott játék egyedi azonosítóját ez által tudva pontosan hogy melyik játékot akarja a kedvencekhez adni a felhasználó és elküldi a POST kérést. Ha nincs probléma akkor hozzáadja a tömbhöz és frissíti is. A kedvencekből való törlés eltérőbb mint a hozzáadás ugyanis itt a body-ban küldjük el a backend-nek az adott játék azonosítóját ezzel megadva pontosan hogy melyik játékot akarjuk törölni a listából a HTTP DELETE metódussal. Utolsóként pedig egy hibakezelő függvény gondoskodik arról hogy a felhasználó csak akkor tudhassa elérni a kedvencek oldalt ha bevan jelentkezve egy fiókkal. Visszatérve a kedvencek komponens második képéhez, láthatja hogy a MatSnackBar felugró üzenet ablakát használjuk a felhasználó értesítésére valamint meghívjuk és használjuk a kedvencek szolgáltatást hogy lehetővé tegyük a játékok eltávolítását. Az utolsó metódus amely ehhez a részhez tartozik a toGame metódus amely az adott játék azonosítója alapján rákattintás esetén elvezet az adott játék oldalára. A következő komponens a homepage avagy a kezdőlap komponens.