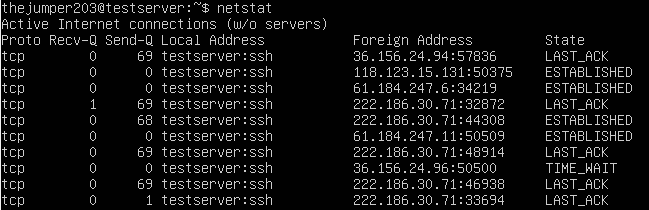
Szerveroldal

A szerveroldal feladata a kliensek felől érkező kérések kiszolgálása és az adatok központi tárolása, kezelése. Az alkalmazás webes és mobil verzióját is ugyanaz a rendszer szolgálja ki. Ez a rendszer áll egy webszerverből és egy adatbázis szerverből.

Webszerver

A kliens oldalról érkező HTTP kéréseket egy Apache[[1]](#footnote-1) webszerver szolgálja ki, amelyre PHP értelmező is van telepítve. Ezzel a szerver képes szerveroldali PHP kódok futtatására. Hálózati operációs rendszerként az Ubuntu 18.04 szerverekre szánt verziója szolgál, amelyre még különböző kiegészítő szolgáltatások lettek telepítve, mint pl. SSH a távoli eléréshez, SFTP a távoli biztonságos fájlátvitelhez. Maga a szerver virtualizált környezetben fut egy Windows 10 alapú asztali számítógépen, amely a VirtualBox nevű virtualizációs alkalmazás segítségével lett létrehozva. Ahhoz, hogy ez a szerver az internet felől is elérhető legyen, a forgalomirányítón meg kell nyitni a megfelelő portokat (port forwarding), és szükség van egy dinamikus DNS[[2]](#footnote-2) szolgáltatásra is. Ez azért szükséges, mert az internetszolgáltatótól kapott IP cím folyamatosan változik, valamint egy választott DNS nevet könnyebb megjegyezni és használni, mint egy IP címet.

A projekt készítése során a szervert több és különböző jellegű támadás is érte az internet felől. Az egyik támadási forma célja az lehetett, hogy root(rendszergazda) hozzáférést szerezzenek a szerverhez SSH protokollon keresztül. Ennek során egyszerre több -jellemzően kínai- IP címről próbáltak root-ként kapcsolódni és bejelentkezni, szerencsére sikertelenül. A szerver üzembe helyezésekor az SSH kiszolgáló konfigurálása során már letiltásra került a root bejelentkezés, amellyel pontosan az ilyen próbálkozások védhetők ki, mivel, ha a külső behatoló ki is találja a jelszót, a rendszer akkor sem engedi belépni. Root-ként bejelentkezve gyakorlatilag teljes hozzáférést kaptak volna a szerver bármely szolgáltatásához, valamint az adatokhoz. A próbálkozásokat valószínűleg valamilyen fajta automatizált folyamatok hajtották végre, amellyel az SSH alapértelmezett 22-es portján próbáltak kapcsolódni. Védelmi intézkedésként az SSH alapértelmezett portja át lett állítva egy teljesen véletlenszerű portra, így a kiszolgálóhoz csak annak ismeretében lehet kapcsolódni. Ezután további hasonló próbálkozásoknak nem volt nyoma.



1. kép - Néhány TCP kapcsolat a támadó IP címekkel a netstat parancs kimenetén

A másik támadási próbálkozás a webszerver ellen irányult, amely során, a naplófájlok bejegyzései alapján, a támadók (szintén leginkább kínai és orosz IP címekről) több érvénytelen HTTP kérés mellett, a WordPress elnevezésű tartalomkezelő és blog-rendszer admin felületét szerették volna elérni. Azonban mivel a szerveren nem található meg a WordPress rendszer, így ezek a próbálkozások is természetesen szintén sikertelenek voltak.

Ezen kísérletek hatására szükség volt néhány biztonsági intézkedésre, amelyek megnehezíthetik a támadók dolgát. A szerveren szigorítva lett a hozzáférés szabályozás, jogosultági rendszer, valamint szigorúbb tűzfal szabályok léptek érvénybe, így csak azok a szolgáltatások érhetők el az internet felől, amelyek feltétlenül szükségesek, az adatbázis szerver pedig egyáltalán nem érhető el kívülről. Ezen kívül a támadó IP címek nagyrésze tiltólistára került, így az onnan érkező bejövő kapcsolatok automatikusan vissza lesznek utasítva. A kliensek és a szerver közötti biztonságos kommunikációért a HTTPS protokoll felel, amely egy hitelesített titkosított csatornán keresztül továbbítja az adatokat, megakadályozva az adatokhoz való illetéktelen hozzáférést.

A fejlesztés során előkerültek különböző problémák a dinamikus DNS használatával is. Mivel ezeket a DDNS címeket bárki létrehozhatja, így nem lehet tudni, hogy valójában ki áll egy-egy ilyen cím mögött, ezért nagyon sok rendszer az ilyen DDNS domaineket automatikusan biztonsági fenyegetésnek tekinti, és megtiltja a kapcsolódást ezekhez. A projekt esetében a szerver címét az Egyetemi kollégium webszűrője blokkolta, így a kollégista csoporttagok nem tudták elérni a szervert. Ennek kiküszöböléséhez szükség volt egy másik szerverre, így a 000webhost nevű ingyenes webtárhely szolgáltatás lett alkalmazva, amely már bárhonnan gond nélkül elérhető és korlátozásokkal ugyan, de biztosítja azokat a szolgáltatásokat, amelyek a rendszer működéséhez szükségesek, gyakorlatilag 0-24 órás rendelkezésre állással, szemben a saját szerverrel.

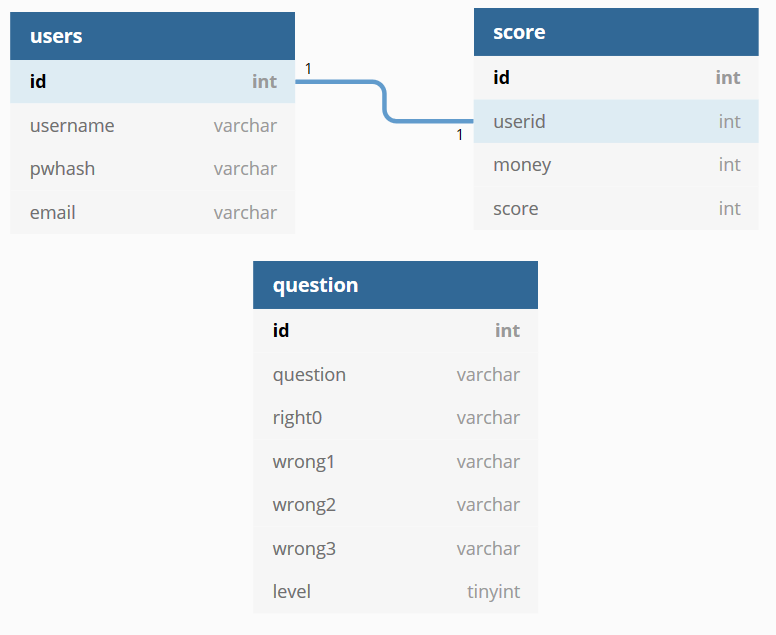
Adatbázis

Az adatbázisok kezelése a MYSQL relációs adatbáziskezelő-rendszer segítségével történik. Az adatok egy milliomos nevű adatbázisban vannak, amelyben a következő táblák találhatóak:

|  |  |  |
| --- | --- | --- |
| users | | |
| id | int | Felhasználó azonosítója |
| username | varchar(150) | Felhasználónév |
| pwhash | varchar(150) | A jelszó lenyomata |
| email | varchar(150) | Email cím |

|  |  |  |
| --- | --- | --- |
| question | | |
| id | int | Kérdés azonosítója |
| question | varchar(150) | Kérdés szövege |
| right0 | varchar(150) | Helyes válasz |
| wrong1 | varchar(150) | Helytelen válaszok |
| wrong2 | varchar(150) |
| wrong3 | varchar(150) |
| level | tinyint | Kérdés szintje |

|  |  |  |
| --- | --- | --- |
| score | | |
| id | int | Azonosító |
| userid | int | Felhasználó azonosító |
| money | int | Legmagasabb megnyert összeg |
| score | int | Pontszám |



1. ábra - Az adatbázis felépítése

Látható, hogy a users id mezője és a score userid mezője között 1:1 kapcsolat található. Ez azt jelenti, hogy minden felhasználóhoz egyetlen rekord tartozhat a score táblában és minden egyes pontszám csak egyetlen felhasználóhoz tartozhat. Új felhasználó hozzáadásakor a score táblában is létrejön egy új rekord a megfelelő felhasználói azonosítóval. A score táblát az alábbi SQL kód hozza létre:

CREATE TABLE score (

id INT NOT NULL AUTO\_INCREMENT,

userid INT NOT NULL FOREIGN KEY REFERENCES user(id) ON DELETE CASCADE,

money INT DEFAULT 0,

score INT DEFAULT 0,

PRIMARY KEY (id));

A userid idegen kulcsként szolgál, amely a users tábla id mezőjére hivatkozik. Az ON DELETE CASCADE paraméter pedig arra szolgál, hogy ha törlődik egy felhasználó a users táblából, akkor a hozzá tartozó pontszám is törlődjön a score táblából, megakadályozva azt, hogy olyan rekordok maradjanak a táblában, amelyek már senkihez sem tartoznak.

PHP

sqlcredits.php

Ebben a fájlban találhatók meg azok az adatok, amelyek az adatbázishoz történő kapcsolódáshoz szükségesek, pl.: adatbázis címe, felhasználónév, jelszó, adatbázis neve. Célszerű ezeket az adatokat külön tárolni, és minden használatnál erre a fájlra hivatkozni, mivel így egy esetleges módosításnál elég, csak ebben az állományban módosítani, valamint biztonsági szempontból is előnyösebb, mivel ezt a fájl kívülre tehetjük azon a könyvtáron, amely az internet felől elérhető.

signup.php

Ez a fájl felel a felhasználók regisztrációjáért. A kliensen lévő regisztrációs űrlapból ide érkeznek be az adatok egy HTTP POST kéréssel. Ebből a POST kérésből a kód kinyeri a paramétereket (felhasználónév, jelszó, email), majd egy trim() függvény segítségével levágja a végéről az esetleges whitespace karaktereket (pl szóköz, tabulátor). Az SQL injection nevű támadási módszer ellen véd a real\_escape\_string() függvény, amely megakadályozza, hogy a SQL kódot injektáljanak a formon keresztül:

$username=$sqlconn->real\_escape\_string(trim($\_POST["username"]));

A fenti kódsoron látható a real\_escape\_string() függvény használata, amely úgy küszöböli ki az SQL injection támadás lehetőségét, hogy escape-eli (feloldja) az olyan potenciálisan veszélyes karaktereket, mint például a ’ vagy a ” mégpedig úgy, hogy beszúrja eléjük az escape karaktert, amely a \ (backslash). Így azokat karaktereket máshogy fogja értelmezni a kód, és egy rosszindulatú felhasználó már nem tud kártékony kódot beilleszteni a szövegdobozon keresztül.

Ezután megvizsgálja, hogy minden paraméter kitöltésre került-e, amennyiben valamelyik paraméter üres, abban az esetben egy „empty” üzenet megy vissza a kliensnek, ahol az ennek megfelelő hibaüzenetet kapja. Ezután ellenőrzi, hogy van-e már regisztrált felhasználó ezzel a névvel és/vagy emaillel. Ilyenkor a program egy „exist” üzenetet küld a kliensnek, ahol a felhasználó szintén egy üzenetet kap, hogy az adott email és/vagy username már használatban van. Amennyiben nincs, úgy az adott felhasználó regisztrálásra kerül az adatbázisba és bejelentkezhet.

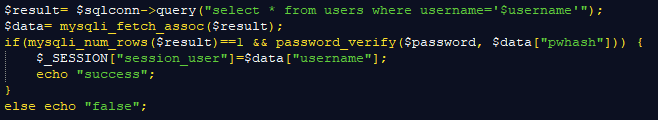
Biztonsági okokból a jelszó nem kerül tárolásra az adatbázisban, hanem csak annak a lenyomata, így még az sem tudhatja meg a jelszót, aki hozzáfér az adatbázishoz. A ténylegesen eltárolt karakterláncot az alábbi kódsor hozza létre:

$pwhash=password\_hash($password,PASSWORD\_DEFAULT);

A password\_hash() egy beépített PHP függvény, amely egy karaktersorozat lenyomatát készíti el.

login.php

A bejelentkezésnél a felhasználónevet és jelszót a regisztrációnál használt módszerhez hasonlóan kapja meg és ellenőrzi.

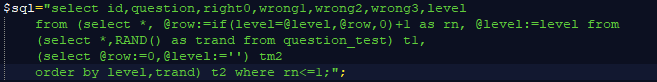


2. kép - A megadott felhasználónevet és jelszót ellenőrző kódrészlet

A felhasználó ellenőrzése úgy történik, hogy először a megadott felhasználónév összes adatát lekérdezi az adatbázisból, amelyet utána egy asszociatív tömbbé alakít. Ezután a program megvizsgálja, hogy pontosan egy eredményt kapott-e vissza az adatbázisból, valamint lefuttatja a password\_verify() függvényt, amely egy hash-t készít a megadott jelszóból és összehasonlítja az adatbázisban tárolt hash-sel. Amennyiben egyezik a kettő, úgy logikai igaz értékkel tér vissza. A bejelentkezés tehát akkor sikeres, ha az megadott felhasználónév létezik, és a password\_verfy() függvény is igaz értékkel tér vissza. Ilyenkor a program egy „success” üzenetet küld vissza a kliens fel, ellenkező esetben pedig egy „false” üzenetet, amelyre a kliens hibaüzenetet dob a felhasználónak. Ugyanez történik akkor is ha valamelyik mező üresen maradt.

getQuestions.php

A játék megkezdésekor a kliens küld egy kérést, amellyel a játék kérdéseit kérdezi le. Paraméterként egy számot kap, hogy hány kérdést adjon vissza.



3. kép - A kérdéseket lekérdező SQL utasítás

A fenti kód az adatbázisból szintenként kérdezi le a kérdéseket véletlenszerűen, és minden egyes kérdés egy-egy asszociatív tömbbe kerül. Ezek az asszociatív tömbök szintén egy tömbbe kerülnek, amelyet a következő kódsor kap meg paraméterként:

echo json\_encode($rows);

Ez a sor annyit tesz, hogy az adatbázisból kapott kérdéseket JSON[[3]](#footnote-3) formátumba kódolja és visszaküldi a kliensnek, ami ezt feldolgozza és tárolja.

getScores.php

Lekéri a tíz legnagyobb pontszámmal rendelkező játékos nevét és pontszámát, majd JSON formátumban visszaküldi a kliensnek.

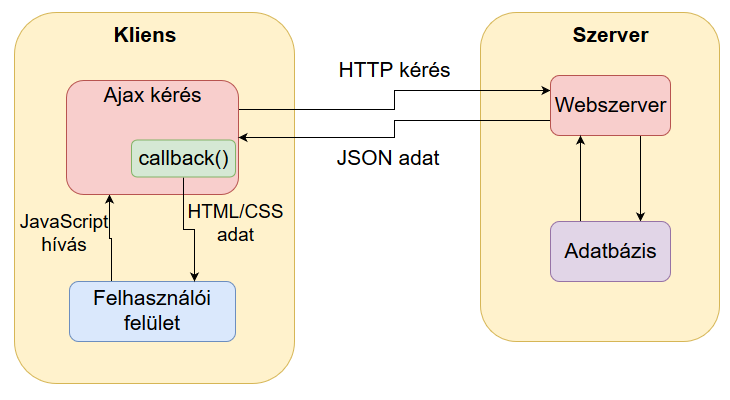
highscore.php

Minden játék végén a kliens küld egy POST kérést ennek a kódnak, amely paraméterként tartalmazza a felhasználónevet és az elért nyereményt. Ezután összehasonlítja az adatbázisból lekéri a felhasználó pontszámait és összehasonlítja az éppen elért nyereménnyel. Ha utóbbi magasabb, akkor az új nyeremény íródik az adatbázisba.

Kliensoldal

Ajax[[4]](#footnote-4)

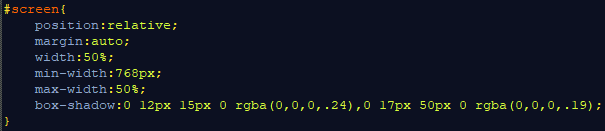
Az alkalmazás fejlesztése során, mind a mobil alkalmazás, mind a webes verzió esetén, az AJAX használata kulcsfontosságú. Az AJAX egy olyan webes technológia, amely a szerver és a kliens közötti adatcserét valósítja meg anélkül, hogy a webes felületet újra kellene tölteni, mindezt aszinkron módon. Ez azt jelenti, hogy amikor a kliens küld egy kérést a szervernek (például lekéri a kérdéseket), akkor a folyamat egy külön szálon indul el, így a háttérben futhat és várakozhat addig ameddig nem érkezik válasz. Ezalatt a webes felület reszponzív marad, és továbbra reagál a felhasználó eseményeire. Természetesen ez különböző problémákat is okozhat, hiszen, ha egy program előbb dolgozna a lekért adatokkal, mint ahogy azok beérkeztek, akkor az hibákhoz vezet. Ennek a kiküszöbölésére használják az úgynevezett callback (visszahívás) függvényeket, amely akkor hívódik meg ha a kérésre valamilyen válasz érkezett. Jelen esetben ez kétféle lehet, sikeres vagy sikertelen. Sikeres kérés esetén a kapott adatokkal már dolgozhat a program, például a beérkeztek a kérdések, akkor indulhat a játék és az első kérdés betölthető.



2. ábra - Az AJAX működésének egyszerű vázlata

Webes verzió

A projekt tervezése és fejlesztése során fontos szempont volt, hogy az egyes platformokon egységesen nézzen ki és ugyanúgy működjön az alkalmazás. Az elsődleges platform a mobilalkalmazás, amely ezáltal mindig előrébb tartott a fejlesztés szempontjából és a webes verzió mindig utólag frissült. Mivel mind az alkalmazás, mind a webes felület ugyanúgy webes elemekből épül fel, így a kód egy-egy platformspecifikus dolgot leszámítva nagyjából azonos minden platformon. A webes felületen az alkalmazás egy container (tároló) divben található, amely rögzített képaránnyal és nagyjából fix mérettel középre igazítva jelenik meg a böngészőben.



4. kép - A container div CSS kódja

Böngésző esetén számolni kellett azzal, hogy az ablak átméretezhető, így az alkalmazás minimum szélességét meg kellett határozni. Ez 768 pixel lett, mivel ennél a szélességnél még nem esik szét a játék felülete. A maximális szélesség, pedig a képernyő 50%-a lehet. A mobilalkalmazás esetében a kijelzőméret adott, így ott nem kellett számolni az átméretezéssel és lehetett a teljes mérethez viszonyítani.

1. Apache HTTP szerver: egy nyílt forráskódú webkiszolgáló alkalmazás [↑](#footnote-ref-1)
2. DNS: Domain Name System, magyarul tartománynévrendszer [↑](#footnote-ref-2)
3. JSON: JavaScript Object Notation: egy szöveg alapú szabvány, amely egyszerű adatstruktúrák, asszociatív tömbök reprezentálására szolgál [↑](#footnote-ref-3)
4. AJAX: Asynchronous JavaScript and XML, Aszinkron JavaScript és XML [↑](#footnote-ref-4)