

Shopventure

Készítette:

Agócs Armand, Csizmadia Bence, Gellén László Dávid

Bevezetés _____	3
A Weboldal kezdetei _____	4
Az oldal felépítése _____	5
Az oldal kinézete _____	5
Reszponzív elemek _____	8
Az oldal funkcionalitása _____	10
Biztonság _____	11
Shopventure _____	11
A játék technológiai háttere _____	13
A játék felépítése (“node hierarchia”) _____	14
Felső szintű node-ok _____	14
A “shop.tscn” főbb node-jai: _____	16
Adatbázis-kommunikáció _____	20

Bevezetés

Projektünk középpontjában egy **boltszimulátor játék** áll, amelyet kifejezetten úgy terveztünk, hogy szórakoztató és interaktív élményt nyújtson a felhasználóknak. A játék lényege, hogy a játékosok saját boltjukat menedzselik, eladnak különböző termékeket, fejlesztenek polcokat, és interakcióba lépnek vásárlókkal. E köré a játék köré építettük fel a teljes weboldalt, amely kiegészítő funkciókat nyújt a játékosok számára.

A weboldal egy modern, letisztult felület, amely lehetővé teszi a játékosok számára, hogy **véleményt** is alkothatnak. Öszehasonlítsák eredményeiket másokkal, beállításokat kezeljenek.

A technológiai alapokat a **Next.js** biztosítja, amely gyors, megbízható és keresőoptimalizált működést tesz lehetővé. A dizájnhoz a **Chakra UI** keretrendszert választottuk, amivel responszív és modern felhasználói élményt tudunk létrehozni. Az oldal minden elemét úgy alakítottuk ki, hogy mobilon és asztali gépen egyaránt könnyen használható legyen.

Az adatbázis- és felhasználókezelés terén a **Supabase** szolgálja a projekt alapját. Ezzel biztosítjuk a **biztonságos regisztrációt és bejelentkezést**, valamint a valós idejű adatlekérdezést, amely mind a játékban, mind a weboldalon folyamatos adatfrissítést tesz lehetővé. Így a statisztikák és eredmények automatikusan naprakészek.

A játékfejlesztéshez a **Godot 4.3** játékmotort használtuk, amely lehetővé teszi, hogy a játék közvetlenül kommunikáljon a Supabase adatbázissal.

A Weboldal kezdetei

A weboldal elkészítéséhez olyan megoldásra volt szükség amivel egyszerűen tudjuk kezelni az adatbázist és a közte lévő kapcsolatot, így emiatt a választás nem más lett, mint a **Next.js** használata. A **Next.js** egy nyílt forráskódú webfejlesztési keretrendszer, amelyet a **Vercel** magáncég hozott létre, amely **React**-alapú webes alkalmazásokat kínál szerveroldali megjelenítéssel és statikus megjelenítéssel. Emellett hogy biztosítsuk a felhasználók élményét úgy döntöttünk, hogy **Chakra UI**-t fogunk használni UI rendszerként. A **Chakra UI**-t Nigirából hozta egy csapat, amivel reszponzív és modern designt tudunk létrehozni.

Most, hogy megvannak az alapok mint a karakterrendszer és a design utána jött az, hogy milyen adatbázist használjunk. Az adatbázisnak úgy gondoltuk, hogy modern felhőalapú adatbázist szeretnénk, hogy bárholnan bármikor elérjük így jött a választás a **Supabase** adatbázisára mivel azzal könnyen össze tudjuk kötni a weboldalt és a játékot. "A Supabase egy nyílt forráskódú Firebase alternatíva. Indítsa el projektjét Postgres-adatbázissal, hitelesítéssel, azonnali API-kkal, Edge-funkciókkal, valós idejű előfizetésekkel, tárolással és vektoros beágyazásokkal." szerepel a weboldalon.

Az oldal elkészítéséhez a következő csomagokat használtuk, így azokat telepítenünk kell.

```
"@chakra-ui/react": "^2.8.2",
"@supabase/supabase-js": "^2.46.2",
"@tinymce/tinymce-react": "^5.1.1",
"browser-image-compression": "^2.0.2",
"framer-motion": "^10.12.16",
"next": "^15.1.2",
"react": "^18.2.0",
```

Importok

Ezeket a csomagokat az npm install parancsával telepítem, majd felhasználom őket a weboldal felépítése közben. Ezek a csomagok elengedhetetlenek az oldalunk működéséhez.

Az oldal felépítése

Következő lépés az npm csomagok telepítés után nem más, mint elkezdni a weboldal felépítését..

A weboldal elkezdéséhez tisztítani kell az index.js-t ami a Next.js generál előre, majd azt átírni egy működő főoldallá. A főoldalnak egyszerűnek kell lennie rendes navigációval és CTA-val (Call to action) így ez alapján készítettem el a dizájnt.

Az oldalnak szüksége lesz szövegre, képre, gombra és különböző elemekre amiket a Chakra UI-ból kell beimportálni.



```
index.js
import {
  Box, Button, Container, Heading, HStack, Stack, Text, Image, Center, Highlight, Tabs, TabList,
  TabPanels, Tab, TabPanel, useColorModeValue, SimpleGrid, Link, Grid, Avatar, VStack, Progress, Card,
  CardHeader, CardBody, Flex
} from "@chakra-ui/react";
```

Chakra UI importok

Az oldalsban ezeket az importokat használjuk mindenhol, mivel ezek a @chakra-ui/react csomagból származó Componentsek.

A következő lépés minden mást beimportálni.

Az főoldalba hogy látszódjanak a vélemények a játékról létre kell hozunk változókat és magát a rendszert, ami az adatbázisból lekérdezi az adatokat majd feltölti a létrehozott elemeket.

A LandingPage komponensben található, és a következő funkciókat látja el:

Színek és háttér beállítása: A useColorModeValue hook segítségével különböző színeket és háttérátmeneteket állít be a világos és sötét módhoz.

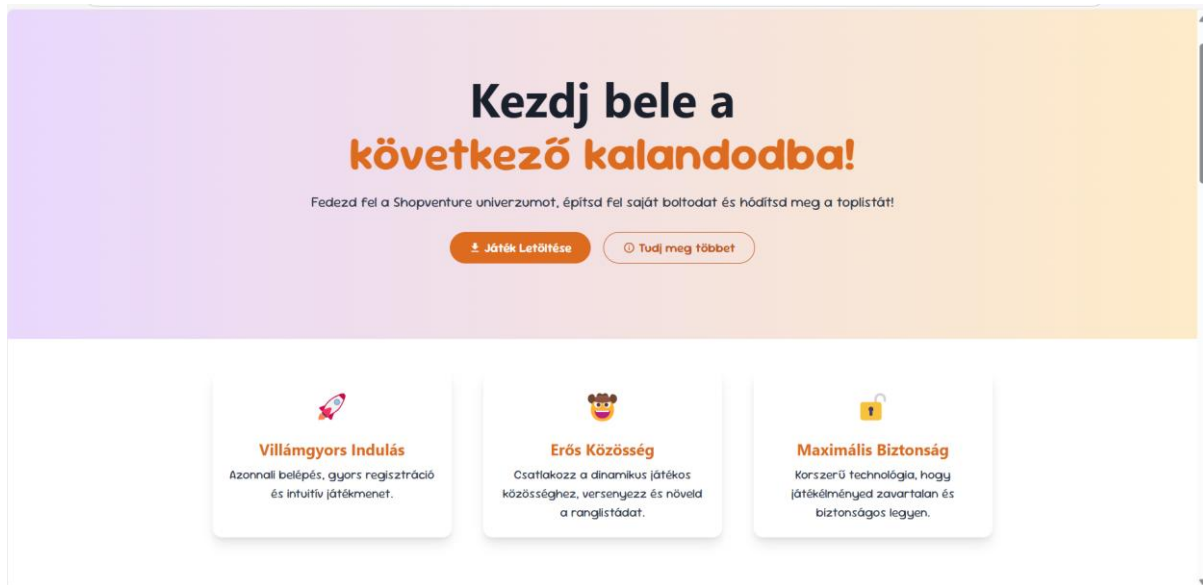
Értékelések kezelése: A useState hookkal egy reviews állapotot hoz létre, amely az értékeléseket tárolja. Az useEffect hook egy aszinkron függvényt futtat, amely a Supabase adatbázisból lekéri az értékeléseket, és beállítja azokat az állapotba.

Átlagos értékelés számítása: Az értékelések átlagát kiszámítja, és kerekíti.

Gördítés funkció: Egy függvény, amely a "Tudj meg többet" gomb megnyomásakor gördíti az oldalt egy adott szekcióhoz.

Az oldal kinézete

Majd ezek után jön maga a front end megoldás



Shopventure főoldal

Az oldal egy szép, letisztult színes megoldást kapott, hogy a felhasználók ne csak egy fekete-szürke oldalt kapjanak, hanem egy boldog megoldást, megfelelő terekkel.

Uralkodj a boltodban!

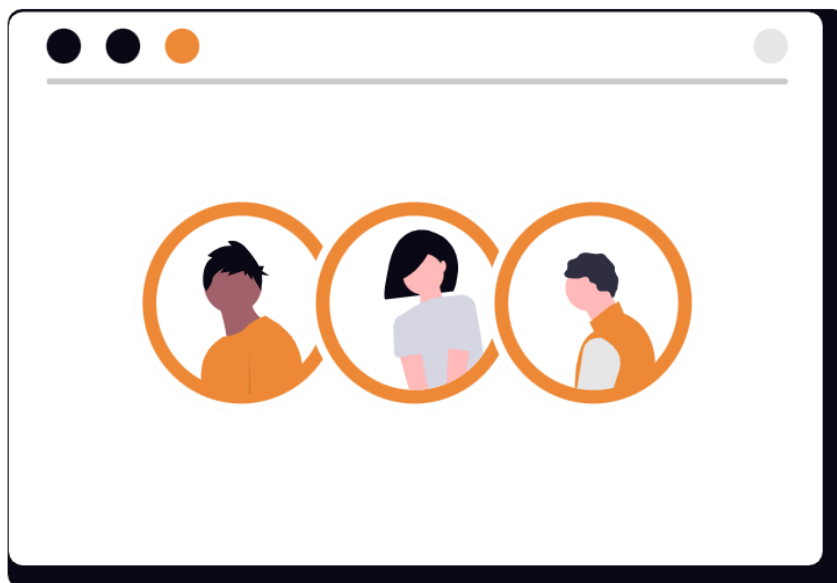
Állítsd be az árakat, optimalizáld a stratégiádat és figyeld, ahogy a vásárlók reagálnak.



Shopventure felhívás

Az emberek imádnak

Nézd meg, hogy játékosaink hogyan vélekedtek játékunkról



Értékelés: 9.7



Értékelj minket

Értékelések oldalunkon

Majd végül a leírások a játékhoz.

További Információk

Shopinfo

Stratégiák

Jutalmak

Üdvözlünk a Shopventure-ben! 🎉

Fedezd fel az üzleti lehetőségek végtelen tárházát, ahol minden döntésed új utakat nyit meg előtted. Itt nem csupán egy élménydús játékvilágot találsz, hanem egy kalandot, amely a kreativitás és az innováció erejét ötvözi. 🚀

A Shopventure-ben minden apró részlet számít: az üzleted stílusos dekorációjától az árak finomhangolásáig minden elem hozzájárul a sikerhez. Minden döntés közelebb visz a céljaid megvalósításához. 🏆

Merülj el a dinamikus üzleti világban, ahol minden lépésed értékes, és az apró változtatások is nagy hatással vannak a fejlődésre. 📈

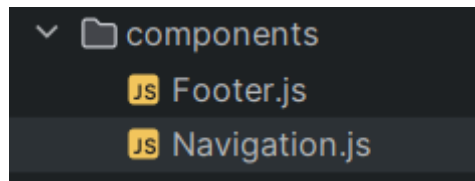
Indulj el ezen az inspiráló úton, és építsd fel saját sikeres történetedet a Shopventure-ben! ☀️

Leírások az oldalban

A weboldalhoz funkcionalitásához elengedhetetlen egy megfelelő navigáció és egy footer, hogy egy könnyen kezelhető és érthető weblapot kapjunk.

Reszponzív elemek


A navigáció és footer egy – egy komponens amit a components mappában tárolok.



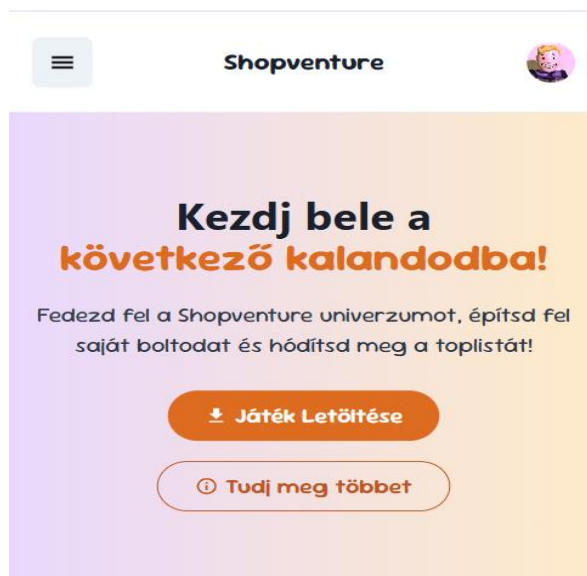
Komponensek a Webstormban

Navbár gépen:

Shopventure

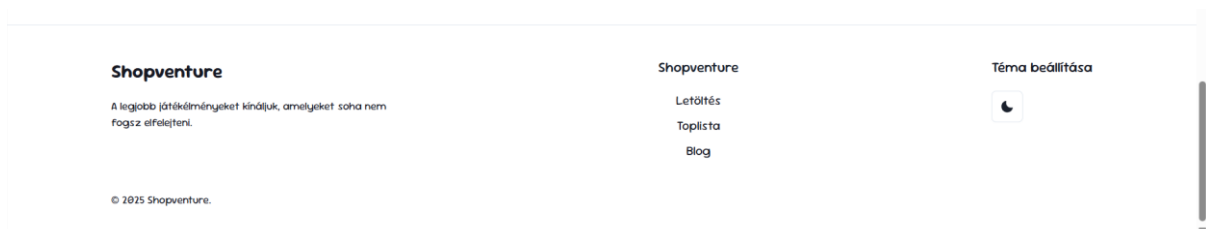
Letöltés Top Blog 

Navbár telefonon a rezszponzív design elérése után.



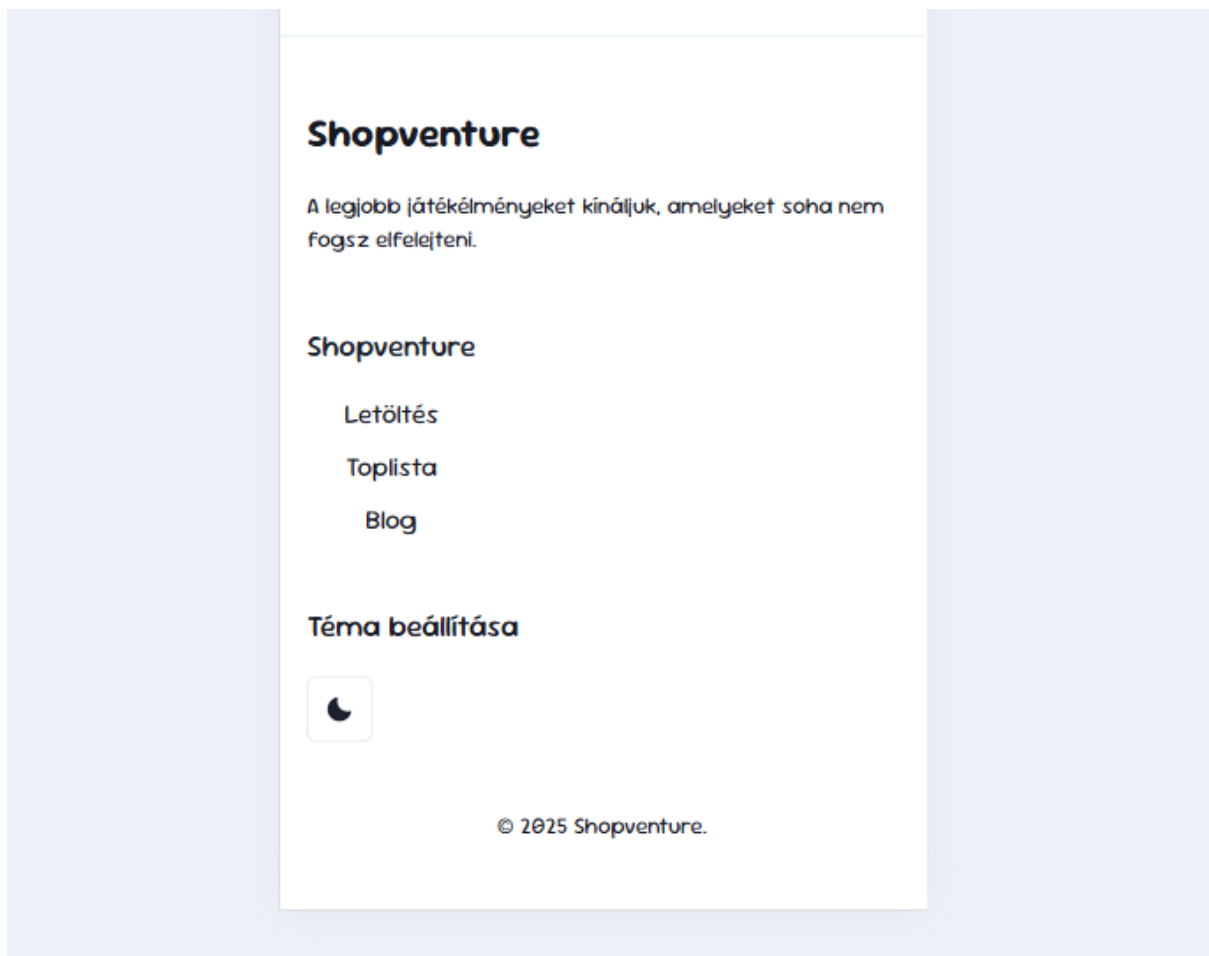
Telefonos kinézet

Footer gépen:



Számítógépes kinézet

Footer telefonon



Footer mobiltelefonon

A főoldal után a további aloldalakat kell létrehozni:

JS _app.js	Az applikáció és dokumentáció, ezek alapok.	60
JS _document.js		
JS download.js	letöltés	61
JS index.js	főoldal	62
JS login.js	bejelentkezés	63
JS profile.js	profil	64
JS register.js	regisztráció	65
JS reviews.js	értékelések	66
JS settings.js	beállítások	67
JS top.js	toplista	68

Webstorm file elrendezés magyarázattal

Az oldal funkcionalitása

A regisztráció (register.js) és a bejelentkezés (login.js) a Supabase beépített auth funkcióira támaszkodik, amelyek lehetővé teszik a biztonságos és egyszerű felhasználókezelést. Ezek a funkciók segítenek abban, hogy az e-mail és jelszó alapú hitelesítés gyorsan megvalósítható legyen, miközben a háttérben a Supabase gondoskodik a biztonságos adatkezelésről és tokenekről. Így a fejlesztés során nem szükséges külön hitelesítési rendszert építeni, hanem megbízható, jól tesztelt megoldásra építhetünk, amely gördülékenyen kezeli a regisztrációt és bejelentkezést.

Bejelentkezés

```
login.js
const { data: {user}, error } = await supabase.auth.signInWithPassword({
  email,
  password,
});
```

Bejelentkezési mechanika

Ez a kódrészlet a Supabase hitelesítési rendszerét használja jelszavas bejelentkezéshez. Az email és password változókat átadva a signInWithPassword metódus elküldi a belépési adatokat a Supabase szerver felé. A válaszból a data tartalmazza a sikeres bejelentkezés esetén kapott információkat (például felhasználói adatokat és tokeneket), míg az error változó hibát jelez, ha a bejelentkezés sikertelen.

Regisztráció

```
register.js
const { data: signUpData, error: signUpError } = await supabase.auth.signUp({
  email,
  password,
});
```

Regisztrációs mechanika

Ez a kódrészlet a Supabase regisztrációs folyamatát valósítja meg. A signUp metódus az email és password értékek alapján létrehoz egy új felhasználói fiókot. A válaszból a signUpData tartalmazza a regisztráció eredményét, például a felhasználói adatokat és státuszt, míg a signUpError változóban jelenik meg az esetleges hiba, ha a regisztráció sikertelen.

Biztonság

A felhasználók adatainak védelme nagy prioritásunk volt, mivel tudjuk, hogy az emberek szeretnék megőrizni az adataikat. A Supabase nagyban támogatja a biztonságot például azzal, hogy a felhasználók privátabb információit encrypteli, emiatt például a jelszavakat sem láthatja senki.

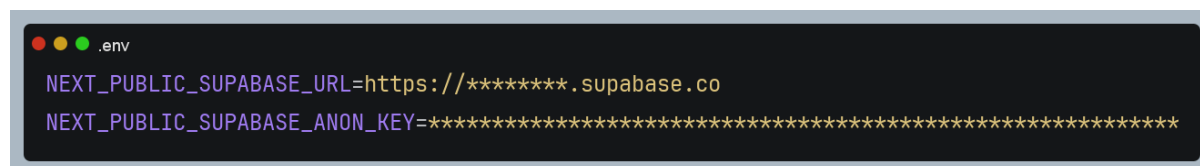
Példa egy jelszóra:

\$2a\$10\$6aWe3FbgJcMvjTwITDRRCuFQtSCbt1K.IXP6qCvZlct/Xu7tDERN6

Encrypton = A legalapvetőbb szinten a titkosítás az információ vagy az adatok védelmét jelenti, amely során matematikai modellek segítségével olyan módon titkosítjuk azokat, hogy csak azok a felek férhetnek hozzá, akik rendelkeznek a titkosítás feloldásához szükséges kulccsal.

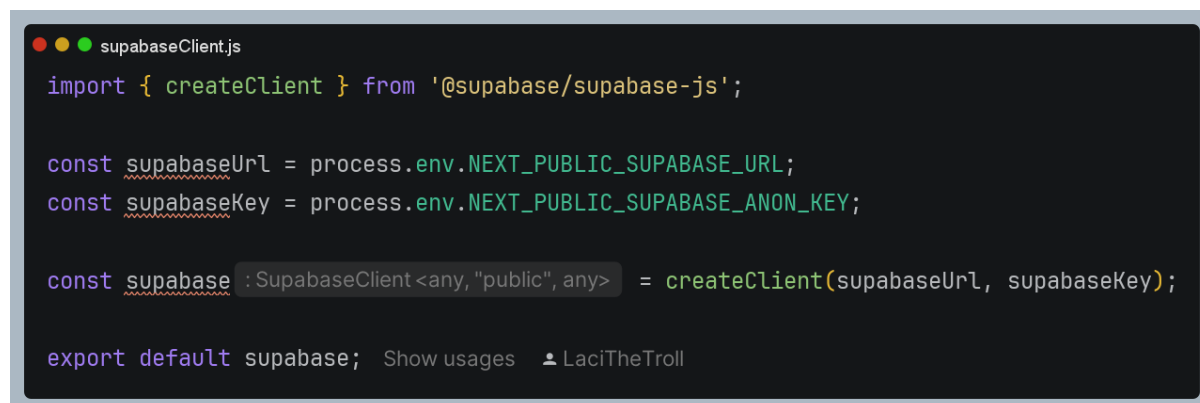
A supabase mellett az oldalunk is titkosítást használ a supabase kulcsok miatt. Erre pontosan a .env ad lehetőséget.

Az .env fájl tartalmazza az egyéni felhasználói környezeti változókat, amelyek felülírják az /etc/environment fájlban beállított változókat.



Biztonsági env file

Amit így használunk fel az utils/supabaseClient fileban:



A supabase Utilja

Shopventure

A **Shopventure** egy egyedülálló bolt szimulátor játék, amelyben te, mint bolt tulajdonos, irányítod a napi működést. A játék alapja, hogy polcokat töltesz fel különböző termékekkel,

dobozokkal és egyéb árucikkkel, miközben NPC-k (nem játszható karakterek) érkeznek és vásárolnak tőled. A hangulatot a színes grafika és az egyszerű, de szórakoztató játékmenet adja.

A játék világában minden nap új kihívások várnak. Mivel a boltod folyamatosan fejlődik, új polcok, dobozok és termékek jelennek meg. Ahogy növekszik a boltod népszerűsége, úgy egyre bonyolultabbá válik a menedzsment, hiszen több polc kell.

A **dobozok** egy különleges szerepet kapnak a Shopventure világában, mivel azok a termékek szállítását és tárolását segítik elő.

A **Shopventure** egy igazi pörgős, stresszes, de szórakoztató bolt menedzsment játék, ahol a siker kulcsa a gyors döntésekben és a taktikai gondolkodásban rejlik. Ha szereted a szimulátorokat és a boltok világát, akkor ez a játék mindenképpen neked való. A jó stratégia és a megfelelő termékkínálat mellett a vásárlók is hűségesek lesznek, és a boltod hamarosan a környék legfelkapottabb helyévé válhat.

A játék technológiai háttere

A játékhoz egy olyan modern, kis méretű és gyors megoldásra volt szükség, amely lehetővé teszi a már meglévő weboldallal és adatbázissal való egyszerű és hatékony kommunikációt. Az optimalizált teljesítmény és a könnyen integrálható rendszer érdekében az elkészítéshez a Godot játékmotor 4.3-as verzióját választottuk. A Godot egy rendkívül sokoldalú, platformfüggetlen, ingyenes és nyílt forráskódú játékmotor, amely az MIT licenz alatt érhető el. Ezáltal nemcsak a költségek csökkentése válik lehetővé, hanem a közösségi fejlesztés és támogatás is garantált. A motor fejlesztői környezete minden főbb platformon, így Windows, Linux és macOS rendszerek alatt is elérhető, és rendkívül könnyen használható. A Godot emellett képes platformfüggetlen futtatható állományok létrehozására, így a fejlesztők számára lehetővé válik, hogy PC-re, mobiltelefonokra, webes alkalmazásokhoz, sőt akár virtuális valóság (VR) platformokra is fejlesszenek játékokat. Ez a széleskörű támogatás biztosítja, hogy a projekt minden igényét kielégíthessük, függetlenül attól, hogy melyik eszközön vagy platformon szeretnénk futtatni a játékot.

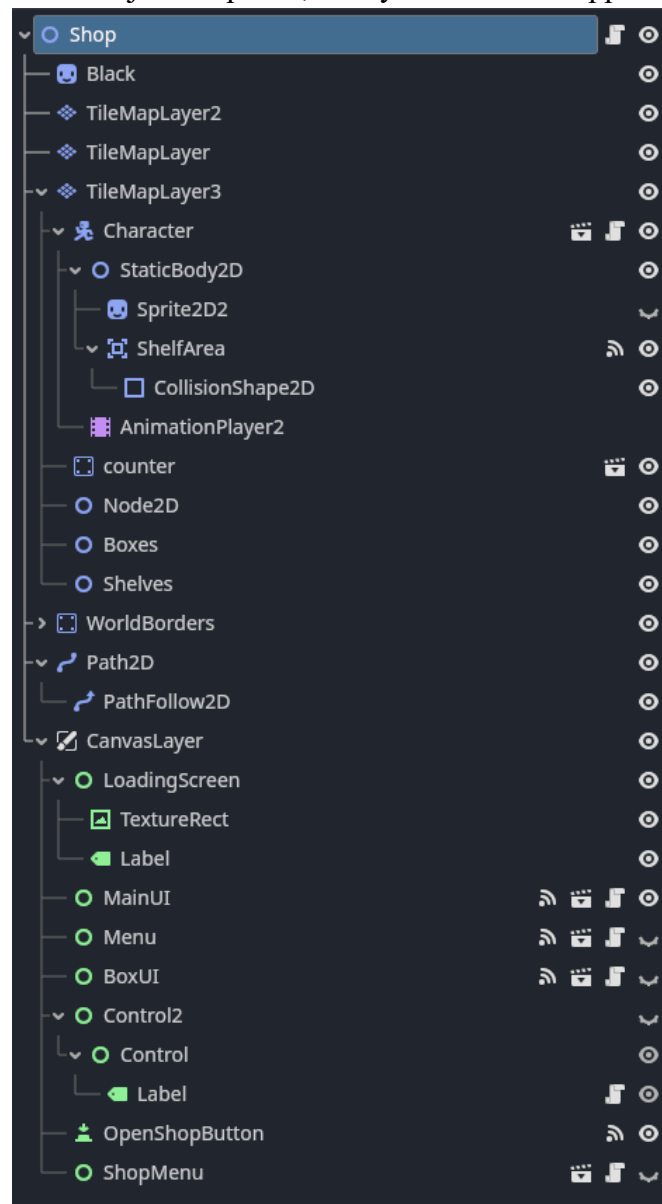
A Godot játékmotor egy rendkívül rugalmas fejlesztési környezetet biztosít, amely lehetővé teszi a játékok programozását C++, C# vagy a kifejezetten a motorhoz kifejlesztett, Python-alapú egyedi nyelv, a GDScript segítségével. Választásunk a GDScriptre esett, mivel ez a nyelv a Python programozási nyelvre épít, amelyet már jól ismertünk, és amely gyors fejlesztést, tiszta kódot és könnyen tanulható szintaxist biztosít. A GDScript rugalmassága és a Godot motor szoros integrációja lehetővé teszi számunkra, hogy gyorsan és hatékonyan készítsünk dinamikus játékmenetet anélkül, hogy a teljesítmény rovására menne.

A Godot motor egyik kiemelkedő tulajdonsága a node-ok (csomópontok) hierarchikus rendszere, amely lehetővé teszi a speciális származtatást és öröklődést, így könnyedén kezelhetjük a komplex játékelemeket és azok kölcsönhatásait. Minden node egy adott funkciót vagy objektumot képvisel a játékban, és ezeket a node-okat scene-ekbe (jelenetek) rendezhetjük, amelyek újrafelhasználható és instanciálható egységekként működnek. Ezzel a struktúrával nemcsak a játékfejlesztés válik hatékonyabbá, hanem a későbbi karbantartás és bővítés is könnyedén megoldható. A node-ok között signal-ok (jelek) segítségével kommunikálhatunk, amelyek lehetővé teszik az adatok továbbítását és az események kezelését. Ez a megoldás rendkívül jól illeszkedik a modern fejlesztési gyakorlatokhoz, ahol az adatok dinamikus és gyors áramlása kulcsfontosságú.

A játék összes erőforrása – beleértve a scripteket, grafikus elemeket, hangokat és egyéb fájlokat – közvetlenül a számítógép fájlrendszerébe kerülnek mentésre. Ez az egyszerű, de hatékony tárolási megoldás nemcsak a játék működését gyorsítja, hanem a verziókezelés során is nagy előnyt jelent. A fájlok egyszerű kezelése és a projekt könnyen nyomon követhető verzióinak tárolása megkönnyíti a csapatmunka folyamatát és a szoftververzió-kezelő rendszerek (pl. Git) használatát, így biztosítva a fejlesztési folyamat zökkenőmentességét.

A játék felépítése (“node hierarchia”)

A játék fő node-ja a shop.tscn, amely a következőképpen épül fel:



A játék hierarchiája

Felső szintű node-ok

- **TileMapLayer3:** Tárolja a játék dinamikus, azaz mozgatható, interaktálható elemeit. Ide tartozik a karakter (“Character”), az eladási terület (“counter”), a tárgyak vásárlására szolgáló dobozok (“Boxes”), és a polcok (“Shelves”).
- **WorldBorders:** A játéktérület limitálására szolgáló CollisionShape-eket tárolja és kezeli.
- **Path2D:** Az egyedi NPC-megoldáshoz szükséges útvonaladatokat tárolja vektorformátumban.
- **CanvasLayer:** A felhasználói kezelőfelület olyan részeit tárolja, mint a betöltőképnyő (“LoadingScreen”), a fő kezelőfelület (“MainUI”), az escape-menü

("Menu"), az interaktálható elemek tárolóinak kezelésére használt felület ("BoxUI"), a boltkezelőgomb ("OpenShopButton") és a boltkezelő-felület ("ShopMenu").

A “shop.tscn” főbb node-jai:

- **TileMapLayer3/Character:** A játékos karakterét tárolja, és kezeli a vele kapcsolatos billentyű bevételeket. A fő algoritmus a Godot által előre létrehozott, fizikai folyamatok kezelésére szolgáló metódusban (“_physics_process”) lett elhelyezve.

```
func _physics_process(delta: float) -> void:
    >| if velocity.x < 0:
    >| >| $Sprite2D.flip_h = 1
    >| elif velocity.x > 0:
    >| >| $Sprite2D.flip_h = 0
    >| >|
    >| if Globals.can_move:
    >| >| var direction := Input.get_axis("ui_left", "ui_right")
    >| >| var directionUp := Input.get_axis("ui_up", "ui_down")
    >| >|
    >| >| if direction:
    >| >| >| velocity.x = direction * SPEED
    >| >| else:
    >| >| >| velocity.x = move_toward(velocity.x, 0, SPEED)
    >| >| >|
    >| >| if directionUp:
    >| >| >| velocity.y = directionUp * SPEED
    >| >| else:
    >| >| >| velocity.y = move_toward(velocity.y, 0, SPEED)
    >| else:
    >| >| velocity.x = 0
    >| >| velocity.y = 0
    >|
    >| if velocity.x != 0 || velocity.y != 0:
    >| >| $AnimationPlayer.play("walk")
    >| >| if !$AudioStreamPlayer2D.playing:
    >| >| >| $AudioStreamPlayer2D.play(0.5)
    >| else:
    >| >| $AnimationPlayer.play("idle")
    >| >| $AudioStreamPlayer2D.stop()
    >| >|
    >| move_and_slide()
```

Script, amely a karaktert kezeli

Az algoritmus a játékos sebessége alapján kezeli a karakter kép (“Sprite2D”) horizontális irányát. A mozgathatóság egy globális változóban lett eltárolva (“Globals”), amely amennyiben IGAZ, annyiban a billentyűzetten W-A-S-D vagy nyílombok által megadott irány figyelembevételével állítja a játékos sebességét. Amennyiben az hamis, a karakter sebessége nullázódik.

Az algoritmus emellett a játékos sebessége alapján kezeli az animációkat (“AnimationPlayer”) és a mozgáshangot (“AudioStreamPlayer2D”), majd a Godot beépített move_and_slide() függvénye meghívásával kezeli az esetleges ütközéseket.

- **TileMapLayer3/Boxes:** Ez a node tárolja az összes, a játékon belüli tárgyak vásárlására szolgáló dobozt. A dobozok dinamikusan kerülnek létrehozásra, egy külön node-ban, amely a box.tscn névre hallgat. Mivel a dobozra fizika nem hat, ezért statikus elemként kezeljük, így itt nincs physics process. Ehelyett a sima folyamat metódusban (“_process”) kerül eltárolásra a fő algoritmus. A továbbiakban a beépített billentyűlenyomás érzékelő metódust (_input) használjuk a kezelésre.


```

>| if Input.is_key_pressed(KEY_F) and can_interact:
>| >| $"/../../../../CanvasLayer/BoxUI".visible = true
>| >| $"/../../../../CanvasLayer/BoxUI".emit_signal("items_changed", items, "Doboz")
>| >| $"/../../../../CanvasLayer/BoxUI".set_meta("open_node", get_path())
>| >| Globals.can_move = false
>| >| await get_tree().create_timer(0.5).timeout

```

Gombnyomások kezelése

A fentebbi script az F gomb lenyomására láthatóvá teszi a BoxUI-t, majd küld annak egy signal-t a doboz tartalmával (“items változó”). Emellett a BoxUI metaadatai open_node változójának értékét a doboz elérési útvonalára módosítja, későbbi használatra. Végül a globális változókon keresztül letiltja a mozgást, és egy 0.5 másodperces időzítőt indít a spamelés elkerülésére miatt.

- **TileMapLayer3/Shelves:** Ez a node tárolja az összes játékon belül létrehozott polcot. A polcok dinamikusan, a játékos kérésére kerülnek létrehozásra. A játékban több polc szint elérhető, ezért ezen node-ok elnevezése shelv_lvl1_, shelv_lvl2_, shelv_lvl3_ és shelv_lvl4_. A polcok fizikai elemek, mivel van ütközésük.

```

func _input(ev):
>| if Input.is_key_pressed(KEY_F) and can_interact:
>| >| $"/../../../../CanvasLayer/BoxUI".visible = true
>| >| $"/../../../../CanvasLayer/BoxUI".emit_signal("items_changed", items, "Polc (Szint 1)")
>| >| $"/../../../../CanvasLayer/BoxUI".set_meta("open_node", get_path())
>| >| Globals.can_move = false>| >| >|
>| >| await get_tree().create_timer(0.5).timeout
>| >|

func _on_area_2d_body_entered(body: Node2D) -> void:
>| if body.name == "Character":
>| >| $Control/UseLabel.visible = true
>| >| can_interact = true
>| >| $"/../../../../Character".set_meta("current_interactable", get_path())
>| elif body.name.contains("NPC"):
>| >| print("xd")
>| pass # Replace with function body.

func _on_area_2d_body_exited(body: Node2D) -> void:
>| if body.name == "Character":
>| >| $Control/UseLabel.visible = false
>| >| can_interact = false
>| >| $"/../../../../Character".set_meta("current_interactable", NodePath())
>| pass # Replace with function body.

```

A polcok kódjának egy fő része

A fentebbi script kezeli a polcokkal való interakciót. Először is az _input metódusban hasonlóképpen (mivel az interaktálható elemek kezelőfelülete megegyező) kezeljük a billentyűlenyomást, mint a dobozok esetében. Másodszor a játékos ütközését az _on_area_2d_body_entered és _on_area_2d_body_exited függvények meghívásával kezeljük. Az entered függvény az ütköző test nevét ellenőrzi, amely ha megegyezik a karakterrel, láthatóvá teszi a polcon az UseLabel kétdimenziós szöveget. Ez jelzi a játékos számára, hogy a polc közelében van, és azzal kapcsolatba tud lépni. Emellett a can_interact változót igazra állítja, majd a karakter metaadataiban a

`current_interactable` változót a polc elérési útvonalára módosítja, későbbi elérés céljából. Az `exited` függvény kezeli azt, amikor a játékos már nem ütközik a polccal, így ez csupán az `entered` függvény változásait érvényteleníti.

- **TileMapLayer3/Node2D:** Ez a node tárolja a játékon belüli, nem játékos-irányított karaktereket (“NPC-k”). Az NPC-k, mint az előbb már ahogy azt kifejtettük, a `Path2D` vektorútvonalát követik. A játékban hat darab NPC típus található, ezek neve a következőképp alakul: `npc1.tscn`, `npc2.tscn`, `npc3.tscn`, `npc4.tscn`, `npc5.tscn` és `npc6.tscn`. Univerzális scriptet használnak, azaz az összes NPC viselkedése ugyanazon az elven működik. Mivel minden egyes NPC fizikai elem a játékban, így az arra céltartott függvényt használjuk.

```
▼ func _physics_process(delta):>|
>| var p = $"../../../../Path2D".curve.get_point_position(point)
▼>| if mode == 0:>|>|
>|>| var coll = await move_to_point(p, delta)
▼>| elif mode == 1:
>|>| velocity.x = 0
>|>| velocity.y = 0
>|>| move_and_slide()
>|>|
>|>| $AnimationPlayer.play("idle")
▼>| elif mode == 2:
>|>| var coll = await move_to_point(counter_position, delta)
>|>| print(coll)
>|>| pass
```

Az NPC-ket kezelő kód

A függvény először is lekéri a `Path2D`-ből a vektorútvonal egy pontját. Majd egy `if` elágazásban lekezeli az NPC jelenlegi módját. Három mód van: 0 - véletlenszerű mozgás, 1 – statikus, azaz nem mozgó, 2 - mozgás a fizetési felület felé.

A nullás mód esetében kezeljük a mozgást a `move_to_point` egyedi függvény segítségével.

```

func move_to_point(p, delta):
    >| >| if hit: return
    >| >| var collision = move_and_collide(velocity * delta)
    >| >| if collision and mode == 0:
    >| >| >| $AnimationPlayer.play("idle")
    >| >| >| velocity.x = 0
    >| >| >| velocity.y = 0
    >| >| >| hit = true
    >| >| >| var collider = collision.get_collider()
    >| >| >|
    >| >| >| if collider.name.contains("shelv"):
    >| >| >| >| if collider.items.size() != 0:
    >| >| >| >| >| var r = rnd.randi_range(0, collider.items.size() - 1)
    >| >| >| >| >| npc_items_value += collider.items[r].price
    >| >| >| >| >|
    >| >| >| >| >| collider.items.pop_at(r)
    >| >| >| >| >|
    >| >| >| >| >| await get_tree().create_timer(rnd.randf_range(1,4)).timeout
    >| >| >| elif collider.name == "counter":
    >| >| >| >| if npc_items_value != 0:
    >| >| >| >| >| await get_tree().create_timer(rnd.randf_range(5,15)).timeout
    >| >| >| >| >| Globals.add_money(npc_items_value*Globals.game_values.sell_multiplier)
    >| >| >| >| >| npc_items_value = 0
    >| >| >| >| >|
    >| >| >| >| >| queue_free()
    >| >| >| >| >| hit = false
    >| >| >| >| >| point = rnd.randi_range(0, $"../../../../Path2D".curve.point_count-1)
    >| >| >| >| >| return
    >| >| >| if !y_pos_got:
    >| >| >| >| if round(position.x) > round(p.x):
    >| >| >| >| >| velocity.x = -max_speed
    >| >| >| >| >| $Sprite2D.flip_h = true
    >| >| >| >| else:
    >| >| >| >| >| velocity.x = max_speed
    >| >| >| >| >| $Sprite2D.flip_h = false
    >| >| >| >| >| else: velocity.x = 0
    >| >| >| >| >|
    >| >| >| if !x_pos_got:
    >| >| >| >| if round(position.y) > round(p.y):
    >| >| >| >| >| velocity.y = -max_speed
    >| >| >| >| else:
    >| >| >| >| >| velocity.y = max_speed
    >| >| >| >| >| else: velocity.y = 0
    >| >| >| >| >|
    >| >| >| >| >| if round(position.y) == round(p.y): x_pos_got = true
    >| >| >| >| >| if round(position.x) == round(p.x): y_pos_got = true
    >| >| >| >| >| if x_pos_got and y_pos_got:
    >| >| >| >| >| >| if mode == 0: point = rnd.randi_range(0, $"../../../../Path2D".curve.point_count-1)
    >| >| >| >| >| >| x_pos_got = false
    >| >| >| >| >| >| y_pos_got = false
    >| >| >| >| >| >|
    >| >| >| >| >| if velocity.x != 0 || velocity.y != 0:
    >| >| >| >| >| >| $AnimationPlayer.play("walk")
    >| >| >| >| >| >| if !$AudioStreamPlayer2D.playing:
    >| >| >| >| >| >| $AudioStreamPlayer2D.play(0.5)
    >| >| >| >| >| else:
    >| >| >| >| >| >| $AnimationPlayer.play("idle")
    >| >| >| >| >| >| $AudioStreamPlayer2D.stop()
    >| >| >| >| >| >|
    >| >| >| >| >| return collision

```

Az NPC-k mozgása

A `move_to_point` függvény a `move_and_collide` függvény felhasználásával ellenőrzi, hogy van-e jelenleg ütközés. Amennyiben van, és az NPC mode-ja megegyezik a 0-val, lejátssza a statikus animációt, és lekéri az ütköző elemet. Amennyiben az ütköző elem egy polc, annyiban az NPC ki fog venni egy véletlenszerű tárgyat a polc tárolójából, majd annak értékét eltárolja. Ha a polc tárolója üres, egy véletlenszerű pontra folytatja a mozgást. Amennyiben az ütköző elem a fizetési terület, ellenőrzi, hogy vett-e már ki tárgyat. Amennyiben ez nem igaz, úgy folytatja a mozgást. Amennyiben vett már ki tárgyat, úgy az NPC a továbbiakban statikus módra vált, és egy 5-15 időintervallumon belül fizetni fog. A fizetett érték a játékos pénztárcájába kerül, és az NPC törlődik. Amennyiben nincs ütközés, néhány alap matematikai számítás segítségével addig mozgatja az NPC-t, ameddig az el nem éri a kívánt pontot. Amint azt elérte, az NPC kap egy új pontot, és a továbbiakban afelé fog mozogni.

Adatbázis-kommunikáció

Az adatbázis kommunikációt a supabase.gd globális script hivatott kezelni. A kommunikációt eleinte egy külső modullal szeretnénk volna megvalósítani, azonban ez a Godot 4.3-as verziójával nem volt kompatibilis, így saját megoldást kellett készítenünk. A globális scriptünk a Supabase REST API-jával kommunikál, az elérési útvonalat a SUPABASE_URL változóba tároltuk, az API biztonsági kulcsát pedig a SUPABASE_API_KEY változóban.

Az alábbi függvények biztosítják az elérést:

```
▼ func _ready():  
    >| http_request = HTTPRequest.new()  
    >| add_child(http_request)  
    >| http_request.connect("request_completed", request_completed)
```

_ready függvény

A **_ready** függvény létrehoz egy HTTP lekérés node-ot, majd hozzáadja azt a fő scene-hez. Ezután csatlakoztat egy signal-t, amely az elvégzett lekéréseket kezeli.

```
▼ func login_user(email: String, password: String) -> void:  
    >| method = "login"  
    >| var url = "%s/auth/v1/token?grant_type=password" % SUPABASE_URL  
    ▼ >| var data = {  
        >| >| "email": email,  
        >| >| "password": password  
        >| }  
    >| var json_data = json.stringify(data)  
    ▼ >| var headers = [  
        >| >| "apikey: " + SUPABASE_API_KEY,  
        >| >| "Content-Type: application/json"  
        >| ]  
  
    >| await http_request.request(url, headers, HTTPClient.METHOD_POST, json_data)
```

login_user függvény

A **login_user** függvény két paramétert kér be, email és jelszó. A **/auth/v1/token/** URL-re továbbít egy HTTP lekérést, amelynek átadja a bejelentkezési adatokat JSON formátumban.

```
▼ func logout() -> void:  
    >| method = "logout"  
    >| var url = "%s/auth/v1/logout" % SUPABASE_URL  
    ▼ >| var headers = [  
        >| >| "apikey: " + SUPABASE_API_KEY,  
        >| >| "Authorization: Bearer " + Globals.api_access_token  
        >| ]  
  
    >| await http_request.request(url, headers, HTTPClient.METHOD_POST)
```

logout függvény

A **logout** függvény kezeli a játékos kijelentkezését, amit továbbít a **/auth/v1/logout/** URL-re.

```
>|  
▼ func push_data(table, data):  
  >| method = "push"  
  >| var url = "%s/rest/v1/%s" % [SUPABASE_URL, table]  
  >| print(url)  
  ▼ >| var headers = [  
    >| >| "apikey: " + SUPABASE_API_KEY,  
    >| >| "Content-Type: application/json",  
    >| >| "Authorization: Bearer " + Globals.api_access_token  
    >| ]  
  >|  
  >| await http_request.request(url, headers, HTTPClient.METHOD_POST, data)
```

push_data függvény

A **push_data** függvény hivatott az adatbázisba új adatot beszúrni. Paraméterei az adatbázis egy táblája, és a beszúrandó adat. Ezeket a **/rest/v1/tábla_neve** URL-re továbbítja a már bejelentkezett felhasználó azonosító tokenjét használva.

```
2 >|  
3 ▼ func delete(table, query, callback: Callable = func v(body) -> void: return):  
4   >| method = "delete"  
5   >| last_callback = callback  
6   >| var url = "%s/rest/v1/%s?%s" % [SUPABASE_URL, table, query]  
7   >| print(url)  
8   >|  
9   ▼ >| var headers = [  
10    >| >| "apikey: " + SUPABASE_API_KEY,  
11    >| >| "Authorization: Bearer " + Globals.api_access_token  
12    >| ]  
13   >|  
14   >| await http_request.request(url, headers, HTTPClient.METHOD_DELETE)
```

delete függvény

A **delete** függvény az adatbázisból töröl egy adatot a megadott paraméterek alapján. Bekér egy táblát, paramétereket és egy visszatérési függvénytt. Ezeket továbbítja a **/rest/v1/delete/** URL-re.

```

56  func query(table, query, callback: Callable = func v(body) -> void: return):
57      >| method = "query"
58      >| last_callback = callback
59      >| var url = "%s/rest/v1/%s?%s" % [SUPABASE_URL, table, query]
60      >| print(url)
61      >|
62  >| var headers = [
63      >| >| "apikey: " + SUPABASE_API_KEY,
64      >| >| "Authorization: Bearer " + Globals.api_access_token
65      >| ]
66      >|
67      >| await http_request.request(url, headers, HTTPClient.METHOD_GET)
68  >|

```

query függvény

A **query** függvény felel meg a lekérdező függvénynek. Bekér egy táblát, egy lekérdező paramétert és egy visszatérési függvényt. Ezeket továbbítja a **/rest/v1/select/** URL-re.

```

func request_completed(result: int, response_code: int, headers: PackedStringArray, body: PackedByteArray):
    if method == "login":
        var bodyUtf = body.get_string_from_utf8()
        json.parse(bodyUtf)
        var data = json.data;
        if response_code == 400:
            print("xdxd")
            supabase_last_error = "Hibás adatok"
            $"/root/MainMenu/LoginScreen/Label4".text = supabase_last_error
            $"/root/MainMenu/LoginScreen/Label4".visible = true
            await get_tree().create_timer(supabase_last_error.length() * 0.25).timeout
            $"/root/MainMenu/LoginScreen/Label4".visible = false
            supabase_last_error = ""
            return
        elif response_code == 200:
            Globals.api_access_token = data.access_token
            Globals.user_uuid = data.user.id
            await query("profiles", "select=username&id=eq.%s" % Globals.user_uuid, getUsername)
        else:
            supabase_last_error = "Ismeretlen hiba"
            return
    elif method == "query":
        last_callback.call(body.get_string_from_utf8())
    elif method == "logout":
        Globals.api_access_token = ""
        Globals.api_refresh_token = ""
        Globals.user_uuid = ""
        Globals.username = ""
    elif method == "push":
        print("push")
        print(result)
        print(response_code)
        print("body")
        print(body.get_string_from_utf8())
        return
    elif method == "delete":
        print("del")
        print(result)
        print(response_code)
        print(body.get_string_from_utf8())
        last_callback.call()
        last_callback = free_callback
        return
    else: return
    pass

```

request_completed függvény

A **request_completed** függvény fut le akkor, amikor egy adott lekérés lefut. Ezeknek megfelelően dönti el, hogy mi történjen a lekérés után. Kijelentkezés esetében például nullázza a mentett felhasználói adatokat, míg delete esetében meghívja a visszatérési függvényt.