** **

**ZÁRÓDOLGOZAT**

Készítették:

Ambrus Kristóf-Oláh Márk-Szesztai Péter

Konzulens:

Kerényi Róbert Nándor, Kiss Roland, Németh Bence, Négyesi Gábor, Farkas Zoltán

Miskolc

2024.

Miskolci SZC Kandó Kálmán Informatikai Technikum

Miskolci Szakképzési Centrum

SZOFTVERFEJLESZTŐ- ÉS TESZTELŐ SZAK

KaKöNyír

Kandó Könyvtárnyilvántartó rendszer

Készítették:

Ambrus Kristóf-Oláh Márk-Szesztai Péter

Miskolc

2023-2024

Tartalomjegyzék

[1. Bevezetés 1](#_Toc164757758)

[2. Fejlesztői eszközök 2](#_Toc164757759)

[2.1. Adatbázis: 2](#_Toc164757760)

[2.1.1. WampServer: 2](#_Toc164757761)

[2.1.1.1. MySQL: 3](#_Toc164757762)

[2.1.1.2. phpMyAdmin: 3](#_Toc164757763)

[2.2. BackEnd: 3](#_Toc164757764)

[2.2.1. ASP.NET: 3](#_Toc164757765)

[2.2.2. Web Api: 5](#_Toc164757766)

[2.2.3. Entity Framework: 5](#_Toc164757767)

[2.2.4. CORS Policy 6](#_Toc164757768)

[2.2.5. A C# 7](#_Toc164757769)

[2.2.6. JWT Token 8](#_Toc164757770)

[2.2.7. CRUD (Create, Read, Update, Delete) 8](#_Toc164757771)

[2.2.8. Az ORM 9](#_Toc164757772)

[2.2.9. Postman: 10](#_Toc164757773)

[2.3. Frontend: 11](#_Toc164757774)

[2.3.1. React: 14](#_Toc164757775)

[2.3.1.1. Előnyei: 15](#_Toc164757776)

[2.3.1.2. Hátrányai/Hátrányosságai: 15](#_Toc164757777)

[2.3.1.3. React felület telepítése: 17](#_Toc164757778)

[2.3.2. Node.js: 17](#_Toc164757779)

[2.3.2.1. A Node.js története: 18](#_Toc164757780)

[2.3.2.2. Node.js architektúra: 18](#_Toc164757781)

[2.3.2.3. Node.js modul-jai: 19](#_Toc164757782)

[2.3.2.4. Node csomagkezelő: 19](#_Toc164757783)

[2.3.3. Visual Studio Code: 19](#_Toc164757784)

[2.3.3.1. Visual Studio vs Visual Studio Code: 20](#_Toc164757785)

[2.3.3.2. Különbségek: 20](#_Toc164757786)

[2.4. A WPF: 21](#_Toc164757787)

# Bevezetés

A könyvtárak az adat, továbbá a szakértelem dön͏t͏ő forrásai, a digitalizáció korában p͏e͏dig rendkívül fon͏tossá vált͏ ͏a haték͏ony, valamint ko͏rsze͏rű könyvtári nyilvántartási rendszerek ͏kialakítása͏.͏ Ez a dolgozat a mi általunk ki͏alakított könyvt͏á͏ri nyilvántart͏ási rends͏zerre ͏össz͏po͏ntosít, a napi kö͏nyvtári működés nehézségeinek ͏és követelményeinek megértése alapján.

Ebben a dolgozatban részletesen ismertetjük a szoftver létrehoz͏ásának processzus͏át, jellemzőit, továbbá előnyeit. Azt is megv͏itatjuk, hogy ez a pr͏ogra͏m hogyan javítja a könyvtári szolgáltatások hatékonyságát é͏s felhasználói͏ élményét.

A dolgozatunk célja, hogy átfogó képet mutassunk arról, hogyan optimalizálhatók a könyvtárak a mai technológia segítségével, mik az előnyei, hátrányai és választ adunk arra is, hogy milyen kihívásokkal kellett szembesülnünk készítése közben.

Programunk az iskolánknak szeretne segíteni, hisz észrevettük, hogy iskolai könyvtárunknak még nincsen digitalizálva az adattárolása, egyben szeretne segíteni tanulótársainknak a könnyebb könyv választástásban is.

A project során tárolásra és verziókezelésre használtuk a [GitHub](https://github.com/ElitCovenant/osszes/tree/main)-ot, megbeszélésre a Discord-ot és Teamspeak-et, a feladatok számontartására pedig a [Trello](https://trello.com/b/cNCCjp6B/librarymanager)-t használtuk.

# Fejlesztői eszközök

# Adatbázis:

Adatbázisunk 8 relációból(táblából) áll. Látható a (1. Adatbázis\_szerkezet) képen. Egyik fő táblánk a **book** reláció, amely 15 mező-t tartalmaz, a másik fő táblánk az a **user**, aminek 8 mezője van. Ennek a két táblának vannak al-táblái, amelyek név szerint a: **rules**, **account\_img**, **loan\_history**, **publisher**, **series** és az **author**. Az utolsó három reláció az a book táblának az al-táblái.Az author tárolja az adott könyv íróját, a series tárolja a könyv szériáját, a publisher pedig a kiadókat tároljuk. A user-nek a rules al-táblájában tároljuk a felhasználók rangjait, az account\_img-be pedig az oldalunkon választható profil képeket tároljuk. A book és user táblának közös al-táblája pedig, a loan\_history. Itt tároljuk a felhasználók által kikölcsönzött könyveket.

1. Adatbázis szerkezet

# WampServer:

A WampServer egy Windows webfejlesztő platform dinamikus webes alkalmazásokhoz, amelyek Apache2 szervert, PHP szkriptnyelvet, MySQL adatbázist és MariaDB-t használnak. A PhpMyAdmin segítségével könnyebben kezelhetjük az adatbázisokat, valamint a MySQL Workbench GUI-t a MySQL és MariaDB szervereket.

Az évek során kipróbált localhost szerverkörnyezetek közül a Wampserver a legjobb. Könnyen használható, több lehetőség, könnyen konfigurálható. A legjobban kifort és a leghatékonyabb helyi webszerver szerintünk.

**Romain Bourdon** 2005-ben adta ki először a nagyvilágnak a programját. Azóta már nyílt forráskódú lett, és több fejlesztő is segítkezik a kódbázis fejlesztésében és karbantartásában. A WampServer a közösség által eléggé támogatott projekt, amely folyamatosan frissül és új funkciókkal bővűl.

Amit jó tudni, a WAMP-ról, hogy a LAMP-ból származik (az L a Linuxot jelenti). A kettő között az egyetlen különbség az, hogy a WAMP-ot Windowshoz, míg a LAMP-ot Linux alapú operációs rendszerekhez használják. A “WAMP” jelentése:

* „**W**” a Windows rövidítése, van még LAMP (Linux)-hoz és MAMP (Mac)-hez.
* „**A**” az Apache rövidítése. Az Apache a weblapok kiszolgálásáért felelős szerverszoftver. Amikor jön egy kérés felőled, hogy láss egy oldalt, az Apache HTTP-n keresztül teljesíti a kérést, és megmutatja a webhelyet.
* „**M**” a MySQL rövidítése. A MySQL feladata, hogy a szerver adatbázis-kezelő rendszere legyen. Tárolja az összes releváns információt, például a webhely tartalmát, felhasználói profiljait stb.
* „**P**” a PHP rövidítése. Ez az a programozási nyelv, amelyet a WordPress írásához használtak. Úgy működik, mint az egész szoftvercsomag ragasztója. A PHP az Apache-val együtt fut, és a MySQL-el kommunikál.

# MySQL:

A MySQL az Oracle által kifejlesztett relációs adatbázis-kezelő rendszer (**RDBMS**), amely strukturált lekérdezési nyelven (SQL) alapul. Az adatbázis az adatok strukturált gyűjteménye. A MySQL-t egy svéd cég, a MySQL AB hozta létre, amelyet a svédek David Axmark, Allan Larsson és a finn Michael "Monty" Widenius alapítottak. A MySQL első verziója 1995. május 23-án jelent meg. Ezt a Sun Microsystems 2008-ban vásárolta meg, és aztán az Oracle 2010-ben megvásárolta a Sun-t.

# phpMyAdmin:

A **phpMyAdmin** egy PHP nyelven írt ingyenes szoftvereszköz, amely a MySQL webes adminisztrációjának kezelésére szolgál. A phpMyAdmin a **MySQL** és a **MariaDB** műveletek széles skáláját támogatja. A gyakran használt műveletek (adatbázisok, táblák, oszlopok, relációk, indexek, felhasználók, engedélyek stb. kezelése) végrehajthatók a felhasználói felületen keresztül, miközben továbbra is lehetősége van bármilyen SQL utasítás közvetlen végrehajtására.

A phpMyAdmin projekt a Software Freedom Conservancy tagja. Az SFC egy non-profit szervezet, amely segít népszerűsíteni, javítani, fejleszteni és megvédeni a szabad, ingyenes és nyílt forráskódú szoftvereket (**FLOSS**) tartalmazó projekteket.

# BackEnd:

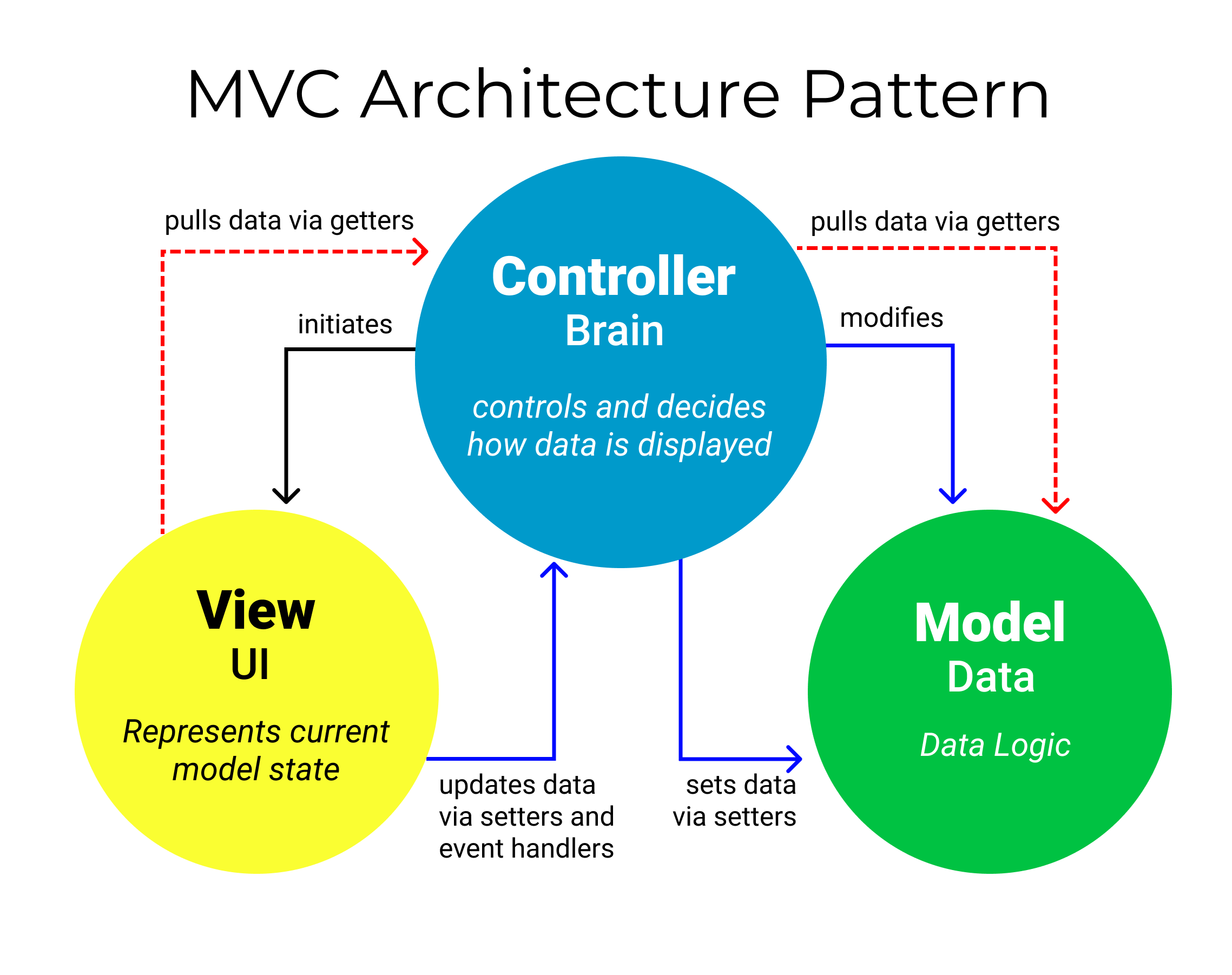
# ASP.NET:

Az **ASP.NET** egy nyílt forráskódú webes keretrendszer, amelyet a Microsoft hozott létre modern webalkalmazások és -szolgáltatások a .NET segítségével történő létrehozására, az 1.0-s verzió pedig 2002-ben jelent meg, hogy a fejlesztők dinamikus webalkalmazásokat, szolgáltatásokat és webhelyeket készíthessenek. A keretrendszer a szabványos HTTP protokollal működik, amely az összes webalkalmazásban használt szabványos protokoll.

A .NET pedig egy olyan fejlesztői platform, amely széles eszköz kínálattal, sok programozási nyelv támogatással rendelkezik, és számos különböző típusú alkalmazás létrehozására alkalmas. Az alapplatform olyan komponenseket tartalmaz, amelyek több különböző típusú alkalmazás létrehozására alkalmasak. A további keretrendszerek, például az ASP.NET, kiterjesztik a .NET-et bizonyos típusú alkalmazások létrehozásához szükséges összetevőkkel.

Az ASP.NET az ASP (**Active Server Pages**) technológia utódja, és jelentős fejlesztést jelentett a rugalmasság és a teljesítmény tekintetében, kiterjeszti a .NET-et, így használható a csomagok és könyvtárak nagy ökoszisztémája, amely minden .NET-fejlesztő számára elérhető. Saját könyvtárak is létrehozhatóak, amelyek megoszthatók a .NET platformon írt alkalmazások között. A keretrendszer tartalmazza a .NET keretrendszer alapkönyvtárait, de a gyakori webes minták könyvtárait is.

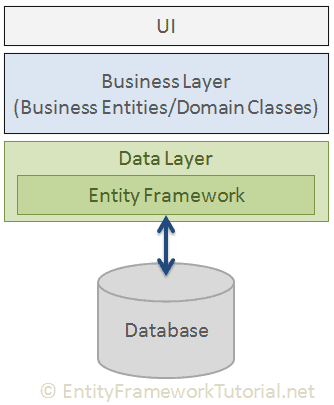
Az egyik ilyen könyvtár a **Model View Controller** (MVC), amely lehetővé teszi az MVC tervezési minta használatát webalkalmazások és webhelyek tervezéséhez, amelyet a felhasználói felület (nézet, vagy angolul view), az adatok (modell) és az alkalmazáslogika (vezérlő, vagy angolul controller) szétválasztására használnak. A webhelyekhez használt MVC-mintát használva a kérések egy vezérlőhöz kerülnek továbbításra, amely felelős a Modellel való együttműködésért a műveletek végrehajtása és/vagy az adatok lekéréséért.

A Vezérlő kiválasztja a megjeleníteni kívánt nézetet, és ellátja a modellel. A Nézet a végső oldalt jeleníti meg a Modell adatai alapján. Megkönnyíti a “tiszta” modell osztályok létrehozását, és leegyszerűsíti a kapcsolódást az adatbázissal. Deklaratív módon határozza meg az érvényesítési szabályokat a C# attribútumok használatával, amelyeket a kliensen és a kiszolgálón alkalmaznak.  


2. MVC\_Modell

Az ASP.NET továbbra is támogatott és frissített, ezért a legújabb fejlesztési eszközökkel lehet dolgozni benne, ami az egyik oka volt, hogy ezt a keretrendszert használjuk, a másik pedig, hogy az asztali alkalmazás programozási nyelve a C# volt, és mivel a keretrendszer támogatja azt, egyszerűbb megoldás volt egy programnyelvet használni a két projektrészhez, mint megtanulni egy teljesen újat.

# Web Api:

Az **API** (application programming interface) vagy alkalmazásprogramozási felület olyan szabályok vagy protokollok összessége, amelyek lehetővé teszik a szoftveralkalmazások számára, hogy kommunikáljanak egymással az adatok, szolgáltatások és funkciók cseréje érdekében. Az API-k leegyszerűsítik és felgyorsítják az alkalmazás- és szoftverfejlesztést azáltal, hogy lehetővé teszik a fejlesztők számára, hogy más alkalmazások adatait, szolgáltatásait integrálják, ahelyett, hogy a semmiből fejlesztenék azokat.

3. Entity Framework

Az API-ok az alkalmazástulajdonosok számára is egyszerű és biztonságos módot kínálnak arra, hogy alkalmazásadataikat és funkcióikat elérhetővé tegyék szervezetük részlegei számára. Az alkalmazástulajdonosok adatokat és funkciókat is megoszthatnak vagy forgalmazhatnak üzleti partnerekkel vagy harmadik felekkel.

Az API-k csak a szükséges információk megosztását teszik lehetővé, más belső rendszerrészleteket rejtve, amik segítik a rendszer biztonságában tartását. A kiszolgálóknak vagy eszközöknek nem kell teljes mértékben felfedniük az adatokat – az API-k lehetővé teszik kis adatcsomagok megosztását az adott kérés szempontjából.

Manapság a legtöbb API már WEB API. A webes API-ok a Remote API egy fajtája (ami azt jelenti, hogy az API protokollokat használ a külső erőforrások manipulálására), amelyek az alkalmazás adatait és funkcióit teszik közzé az interneten keresztül.

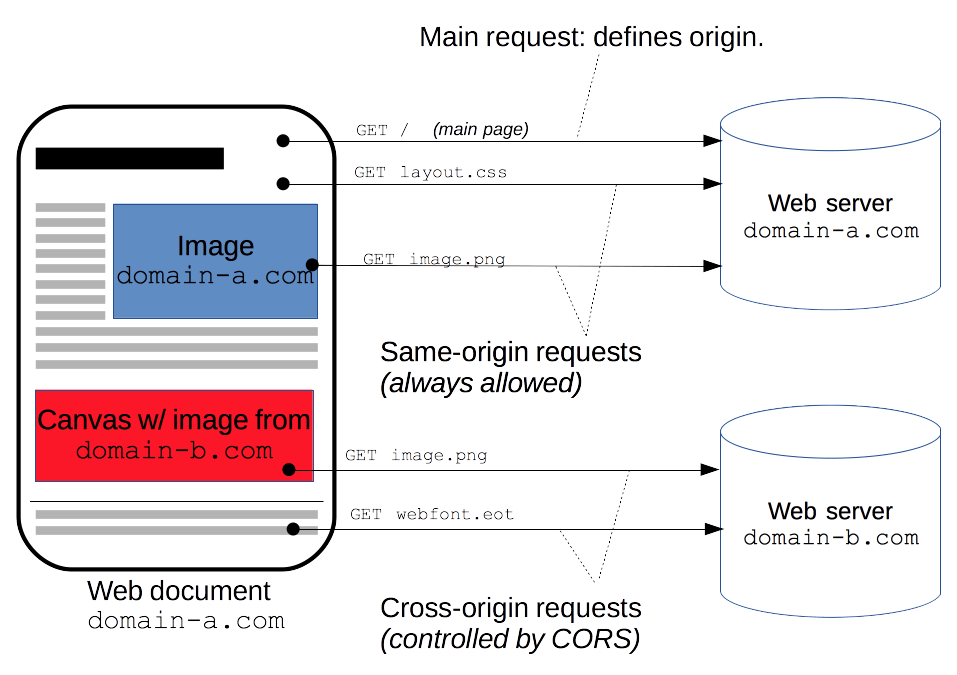
# Entity Framework:

Az Entity Framework egy nyílt forráskódú ORM-keretrendszer a Microsoft által támogatott .NET-alkalmazásokhoz. Lehetővé teszi a fejlesztők számára, hogy tartományspecifikus osztályok objektumai segítségével dolgozzanak az adatokkal anélkül, hogy a mögöttes adatbázistáblákra és oszlopokra kellene összpontosítaniuk, ahol ezeket az adatokat tárolják. Az Entity Framework segítségével a fejlesztők magasabb absztrakciós szinten dolgozhatnak, amikor adatokkal foglalkoznak, és adatorientált alkalmazásokat hozhatnak létre és karbantarthatnak kevesebb kóddal a hagyományos alkalmazásokhoz képest.

Az Entity Framework API (EF6 és EF Core) magában foglalja a tartomány (entitás) osztályok adatbázissémához való leképezését, a LINQ-lekérdezések SQL-be ​​való lefordítását és végrehajtását, az entitásokon az élettartamuk során bekövetkezett változások nyomon követését és a változtatások mentését az adatbázisban.

# CORS Policy

Cross-Origin Resource Sharing (CORS) egy HTTP-fejlécen alapuló mechanizmus, amely lehetővé teszi a kiszolgáló számára, hogy jelezze a sajátjától eltérő eredetet (tartományt, sémát vagy portot), amelyből a böngészőnek engedélyeznie kell az erőforrások betöltését. A CORS egy olyan mechanizmusra is támaszkodik, amellyel a böngészők "elővizsgálati" kérést küldenek az eredetközi erőforrást tároló kiszolgálónak annak ellenőrzése érdekében, hogy a kiszolgáló engedélyezi-e a tényleges kérést. Ebben az elővizsgálatban a böngésző fejléceket küld, amelyek jelzik a HTTP-metódust és a tényleges kérésben használt fejléceket.



4. CORS Policy

Biztonsági okokból a böngészők korlátozzák a parancsfájlokból kezdeményezett eltérő eredetű HTTP-kérelmeket. A fetch() és az XMLHttpRequest például az azonos eredetre vonatkozó szabályzatot követi. Ez azt jelenti, hogy az API-kat használó webalkalmazások csak ugyanabból a forrásból kérhetnek erőforrásokat, ahonnan az alkalmazás be lett töltve, kivéve, ha a más forrásokból érkező válasz tartalmazza a megfelelő CORS-fejléceket.

A CORS mechanizmus támogatja a biztonságos eredetközi kéréseket, valamint a böngészők és szerverek közötti adatátvitelt. A böngészők CORS-t használnak olyan API-kban, mint a fetch() vagy az XMLHttpRequest az eredetközi HTTP-kérések kockázatának csökkentésére.

Az eredetközi erőforrás-megosztási szabvány új HTTP-fejlécek hozzáadásával működik, amelyek lehetővé teszik a kiszolgálók számára, hogy leírják, mely eredetek olvashatják be ezeket az információkat egy webböngészőből. Ezenkívül olyan HTTP-kérési módszerek esetén, amelyek mellékhatásokat okozhatnak a kiszolgáló adataira (különösen a GET-től vagy bizonyos MIME-típusoknál a POST-tól eltérő HTTP-metódusok), a specifikáció előírja, hogy a böngészőknek "ellenőrizniük" kell a kérést, támogatott metódusokat kell kérniük a kiszolgálótól a HTTP OPTIONS kérési módszerrel, majd a kiszolgáló "jóváhagyása" után el kell küldeniük a tényleges kérést. A kiszolgálók arról is tájékoztathatják az ügyfeleket, hogy a "hitelesítő adatokat" (például a cookie-kat és a HTTP-hitelesítést) el kell-e küldeni kérésekkel.

A CORS-hibák hibákat eredményeznek, de biztonsági okokból a JavaScript nem fér hozzá a hibával kapcsolatos információkhoz. A kód csak annyit tud, hogy hiba történt. Az egyetlen módja annak, hogy meghatározzuk, mi történt konkrétan, ha megnézi a böngésző konzolját a részletekért. A következő szakaszok ismertetik a forgatókönyveket, valamint részletezik a használt HTTP-fejléceket.

# A C#

A C# nyelv a legnépszerűbb nyelv a .NET platformon, egy ingyenes, többplatformos, nyílt forráskódú fejlesztői környezetben. A C#-programok számos különböző eszközön futtathatók, az eszközök internetes hálózata (IoT) eszközöktől a felhőig és bárhol a kettő között. Írhat alkalmazásokat telefonra, asztali számítógépre és laptopra, valamint kiszolgálókra. A C# egy platformfüggetlen általános célú nyelv, amely a fejlesztőket produktívvá teszi, miközben nagy teljesítményű kódot ír. Több millió fejlesztővel a C# a legnépszerűbb .NET nyelv.

A C# széles körű támogatást nyújt az ökoszisztémában és az összes .NET számítási feladatban. Az objektumorientált elvek alapján más paradigmák számos jellemzőjét magában foglalja, nem utolsósorban a funkcionális programozást. Az alacsony szintű funkciók támogatják a nagy hatékonyságú forgatókönyveket nem biztonságos kód írása nélkül. A legtöbb .NET futtatókörnyezet és kódtár C# nyelven íródott, és a C# fejlesztései gyakran minden .NET fejlesztő számára előnyösek.

A C#-alkalmazások kihasználják a .NET Runtime automatikus memóriakezelését. A C#-alkalmazások a .NET SDK által biztosított kiterjedt futásidejű kódtárakat is használják. Egyes összetevők platformfüggetlenek, például a fájlrendszerkönyvtárak, az adatgyűjtések és a matematikai könyvtárak. Mások egyetlen számítási feladatra vonatkoznak, például a ASP.NET Core webkódtárak vagy a .NET MAUI UI kódtár. A NuGet gazdag nyílt forráskódú ökoszisztémája kiegészíti a futtatókörnyezet részét képező kódtárakat. Ezek a könyvtárak még több használható összetevőt biztosítanak.

A C# egy erősen típusos nyelv. Minden deklarált változónak van egy, a fordításkor ismert típusa. A fordító vagy a szerkesztőeszközök jelzik, ha helytelenül használja az adott típust. Ezeket a hibákat a program futtatása előtt kijavíthatja. Az alapvető adattípusok be vannak építve a nyelvbe és a futásidőbe: olyan értéktípusok, mint az int, double, char, referenciatípusok, például sztring, tömbök és egyéb gyűjtemények. A programok írása közben saját típusokat hoz létre.

Ezek a típusok lehetnek értékek strukturáltsági típusai vagy objektumorientált viselkedést definiáló osztálytípusok. A rekordmódosítót hozzáadhatja a struct vagy az osztály típusokhoz, így a fordító szintetizálja a kódot az egyenlőségi összehasonlításokhoz. Létrehozhat felületdefiníciókat is, amelyek meghatározzák azt a szerződést vagy tagok készletét, amelyet az interfészt megvalósító típusnak biztosítania kell. Általános típusokat és módszereket is definiálhat. Az általános szabályok típusparaméterekkel biztosítanak helyőrzőt egy tényleges típushoz, amikor használják. A kód írásakor függvényeket, más néven metódusokat definiálhat a struct és az osztály típusok tagjaiként.

Ezek a metódusok határozzák meg a típusok viselkedését. A módszerek túlterhelhetők, különböző számú vagy típusú paraméterekkel. A metódusok igény szerint értéket is visszaadhatnak. A metódusok mellett a C#-típusok tulajdonságokkal is rendelkezhetnek, amelyek accessoroknak nevezett függvények által támogatott adatelemek.

A C#-típusok eseményeket határozhatnak meg, amelyek lehetővé teszik, hogy egy típus értesítse az előfizetőket a fontos műveletekről. A C# támogatja az objektumorientált technikákat, például az öröklődést és a polimorfizmust az osztálytípusokhoz.

# JWT Token

A JSON webes jogkivonat (JWT) JSON-objektum, amely az információk biztonságos átvitelére szolgál a weben keresztül (két fél között). Használható hitelesítési rendszerhez, és információcserére is használható. A token főleg fejlécből, hasznos teherből és aláírásból áll. Ezt a három részt pontok (.) választják el egymástól.

A JWT meghatározza az egyik féltől a másiknak küldött információk szerkezetét, és két formában jelenik meg - szerializált, deszerializált. A szerializált megközelítést elsősorban arra használják, hogy az adatokat a hálózaton keresztül továbbítsák minden egyes kéréssel és válasszal. Míg a deszerializált megközelítés az adatok webes jogkivonatba való olvasására és írására szolgál.

**Deserialized** : A JWT deszerializált formában csak a fejlécet és a hasznos terhet tartalmazza. Mindkettő egyszerű JSON-objektum.

**Header** : A JWT-ben lévő fejlécet leginkább a JWT-re alkalmazott titkosítási műveletek, például a rajta használt aláírási/visszafejtési technika leírására használják. Tartalmazhatja az általunk küldött információk média/tartalom típusára vonatkozó adatokat is. Ez az információ JSON-objektumként van jelen, majd ez a JSON-objektum BASE64URL kódolva lesz. A fejlécben található titkosítási műveletek határozzák meg, hogy a JWT aláírt/aláíratlan vagy titkosított-e, és így milyen algoritmustechnikákat kell használni.

**Payload** : A payload a JWT azon része, ahol az összes felhasználói adat ténylegesen hozzá van adva. Ezeket az adatokat a JWT "állításainak" is nevezik. Ezeket az információkat bárki elolvashatja, ezért mindig tanácsos bizalmas információkat ide tenni. Ez a rész általában felhasználói adatokat tartalmaz. Ez az információ JSON-objektumként van jelen, majd ez a JSON-objektum BASE64URL kódolva lesz. Annyi jogcímet helyezhetünk el egy hasznos teherben, amennyit csak akarunk, bár a fejléccel ellentétben a hasznos adatokban nem kötelező jogcím.

**Signiture** : Ez a JWT harmadik része, és a jogkivonat hitelességének ellenőrzésére szolgál. BASE64URL kódolt fejléc és hasznos adat össze van kapcsolva a dot(.) függvényrel, majd kivonatolva van a fejlécben definiált kivonatoló algoritmussal egy titkos kulccsal. Ezt az aláírást ezután hozzáfűzi a fejléchez és a hasznos adatokhoz a dot(.) használatával.

# CRUD (Create, Read, Update, Delete)

Az API-k létrehozásakor azt szeretnénk, hogy modelljeink négy alapvető funkciótípust biztosítsanak. A modellnek képesnek kell lennie erőforrások létrehozására, olvasására, frissítésére és törlésére. Az informatikusok gyakran hivatkoznak ezekre a funkciókra a CRUD rövidítéssel. Ahhoz, hogy egy modell teljes legyen, képesnek kell lennie legfeljebb e négy funkció végrehajtására. Ha egy művelet nem írható le e négy művelet egyikével, akkor potenciálisan saját modellnek kell lennie.

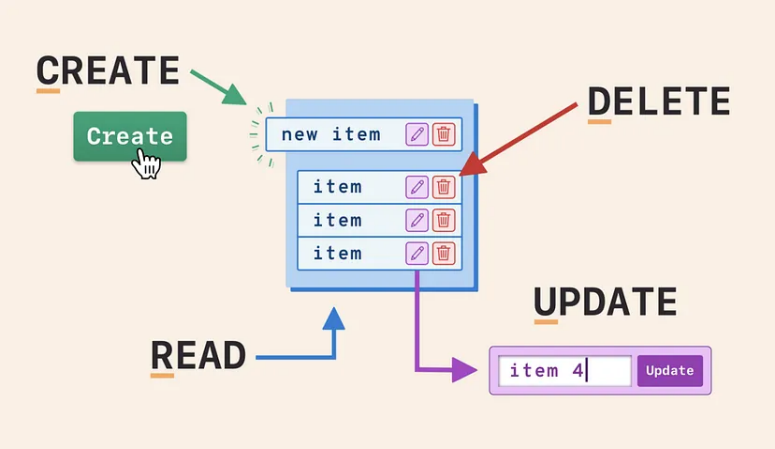
A CRUD paradigma gyakori a webalkalmazások létrehozásában, mert emlékezetes keretet biztosít a fejlesztők emlékeztetésére a teljes, használható modellek felépítésére. Képzeljünk el például egy rendszert a könyvtári könyvek nyomon követésére. Ebben a hipotetikus könyvtári adatbázisban el tudjuk képzelni, hogy lenne egy könyvforrás, amely könyvobjektumokat tárolna.

**Létrehozás** (Create) — Ez egy olyan függvényből állna, amelyet akkor hívnánk meg, amikor egy új könyvtári könyvet adunk hozzá a katalógushoz. A függvényt hívó program megadja a "title", "author" és "isbn" értékeket. A függvény meghívása után egy új bejegyzésnek kell lennie a könyvek erőforrásában, amely megfelel ennek az új könyvnek. Emellett az új bejegyzéshez egyedi azonosító van rendelve, amellyel később hozzáférhet ehhez az erőforráshoz.

**Olvasás** (Read) — Ez egy olyan függvényből állna, amely a katalógusban jelenleg szereplő összes könyv megtekintésére szolgálna. Ez a függvényhívás nem változtatja meg a katalógusban lévő könyveket - egyszerűen lekéri az erőforrást, és megjeleníti az eredményeket. Funkciónk lenne egyetlen könyv visszakeresésére is, amelyhez megadhatnánk a címet, a szerzőt vagy az ISBN-t. Ismétlem, ezt a könyvet nem módosítanák, csak visszakeresnék.

**Frissítés** (Update) — Lennie kell egy függvénynek, amelyet meg kell hívni, ha egy könyvvel kapcsolatos információt meg kell változtatni. A függvényt hívó program megadja a "title", "author" és "isbn" új értékeit. A függvényhívás után a könyvek erőforrás megfelelő bejegyzése tartalmazza a megadott új mezőket.

**Törlés** (Delete) — Lennie kell egy függvénynek, amellyel el lehet távolítani egy könyvtári könyvet a katalógusból. A függvényt hívó program egy vagy több értéket ("title", "author" és/vagy "isbn") ad meg a könyv azonosításához, majd ez a könyv törlődik a könyvek erőforrásból. A függvény meghívása után a books erőforrásnak tartalmaznia kell az összes korábbi könyvet, kivéve az imént töröltet.



5. CRUD modell

# Az ORM

Az ORM vagy az Object Relational Mapper egy olyan szoftver, amelyet az adatbázisok által használt adatreprezentációk és az objektumorientált programozásban használt adatábrázolások közötti fordításra terveztek. Alapvetően az adatokkal való munka e két módja természetesen nem illeszkedik egymáshoz, ezért az ORM megpróbálja áthidalni a két rendszer adattervei közötti szakadékot.

A fejlesztő szempontjából az ORM lehetővé teszi, hogy adatbázis-alapú adatokkal dolgozzon ugyanazokkal az objektumorientált struktúrákkal és mechanizmusokkal, amelyeket bármilyen típusú belső adathoz használna. Az ORM-ek ígérete az, hogy nem kell speciális technikákra támaszkodnia, vagy feltétlenül meg kell tanulnia egy új lekérdezési nyelvet, például az SQL-t ahhoz, hogy produktív legyen az adataival.

Általában az ORM-ek absztrakciós rétegként szolgálnak az alkalmazás és az adatbázis között. Megpróbálják növelni a fejlesztői termelékenységet azáltal, hogy kiküszöbölik a sablonkód szükségességét, és elkerülik a kínos technikák használatát, amelyek megtörhetik a választott nyelvtől elvárt idiómákat és ergonómiát.

Bár az ORM-ek hasznosak lehetnek, fontos, hogy eszközként tekintsünk rájuk. Nem minden esetben lesznek hasznosak, és előfordulhat, hogy kompromisszumokat kell figyelembe vennie. Általánosságban elmondható, hogy az ORM jó választás lehet, ha a nyelv számos objektumorientált funkcióját használja sok állapot kezelésére.

Az összetett öröklési kapcsolatokkal rendelkező objektumokba ágyazott állapot kezelésének következményeit például nehéz lehet manuálisan figyelembe venni. Emellett segíthetnek a projekt könnyebb elindításában, és olyan funkciókkal kezelhetik az adatstruktúra változásait, mint a sémaáttelepítés.

Bár az ORM-ek gyakran hasznosak, nem tökéletesek. Néha az ORM által bevezetett absztrakciós szint megnehezítheti a hibakeresést. Előfordulhat, hogy az ORM által az adatbázis és az alkalmazás közötti fordításhoz használt ábrázolás nem teljesen pontos, vagy kiszivárogtathatja a belső megvalósítás részleteit. Ezek bizonyos esetekben problémákat okozhatnak.

Fontos megérteni, hogy mik a projekt követelményei, és hogyan szeretné elkölteni az erőforrásokat a szoftver létrehozásakor. Az ORM egy olyan eszköz, amely segíthet az adatbázis-alapú alkalmazások könnyebb felépítésében, de magának kell eldöntenie, hogy hozzáadott értéket ad-e a projekthez.

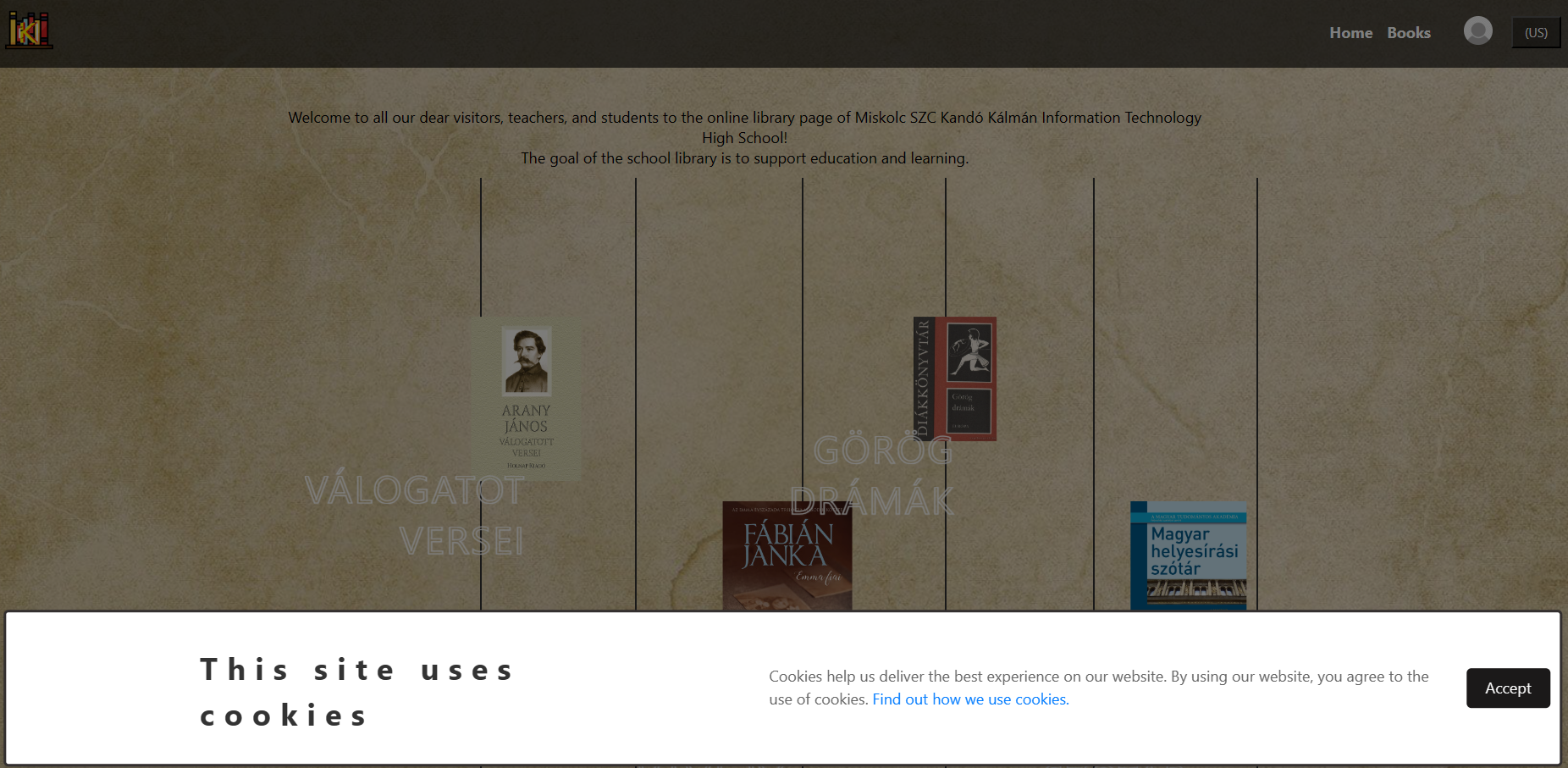
Általában jó ötlet korán megkérdezni magától, hogy egy ORM segítené-e a projektet. A kompromisszumok értékelése jó gyakorlat lehet, amely segít megérteni, hogyan szeretné összpontosítani erőfeszítéseit.

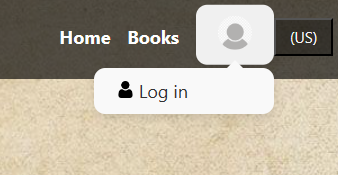
# Postman:

A Postman egy olyan szoftveralkalmazás, amely lehetővé teszi a fejlesztők számára API-k (alkalmazásprogramozási felületek) tesztelését, dokumentálását és megosztását. A fejlesztők széles körben használják az API-k tesztelésének egyszerűsítésére azáltal, hogy felhasználóbarát felületet biztosítanak a kérések végrehajtásához, a válaszok megtekintéséhez és a hibakereséshez. Ettől eltekintve, ha ki szeretné venni a Mocking API-t és a felhasználóbarát API-dokumentációt, ez a cikk ideális lehetőséget kínál az Ön számára. Az Apidog egy hatékony API-fejlesztő eszköz, amely összeköti az API-fejlesztési folyamatban részt vevő összes résztvevőt. All-in-One munkaterület API-tervezéshez, dokumentációhoz, hibakereséshez, utánzáshoz és teszteléshez. Emellett számos tesztelési eszközt is biztosít, beleértve az egységteszteket, az integrációs teszteket és a terheléses teszteket, hogy az API különböző körülmények között a várt módon teljesítsen. Ezekkel az eszközökkel biztosíthatja, hogy az API megbízható, biztonságos és méretezhető legyen.

# Frontend:

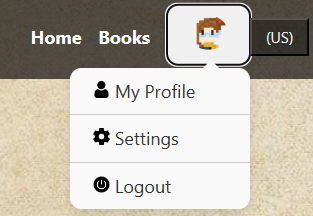
Mi frontendünk elkészítéséhez **React**, **Node.js** és **Visual Studio Code**-ot használtunk. Weblapunkat próbáltuk minél jobban felhasználóbarátabbá tenni. Megynitásakor egy Süti elfogadására késztető Modal ugrik fel, amelyekkel a felhasználó hozzájárúl oldalunk sütieihez, majd jobb felül a felhasználó a bejelentkezetlen ikonra nyomva be tud lépni kkszki-s email címmel.Például:”*gyakorlas@kkszki.hu*” (Lásd 7. kép)



6. Frontend kezdőlap

7. Bejelentkezési fül

Amiután sikeresen bejelentkezett a felhasználó, azután ez a menüpont megváltozik és már 3 menüt láthatunk rajta. “*My Profil*”, “*Settings*”, “*Logout*”.

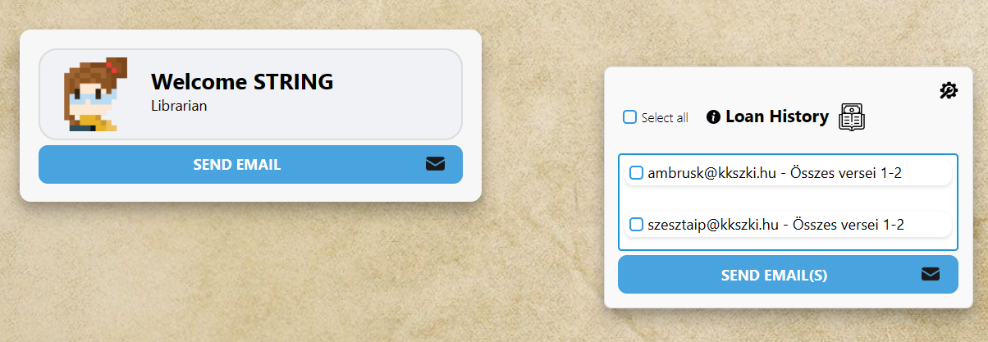


8. Bejelentkezett menüpontok

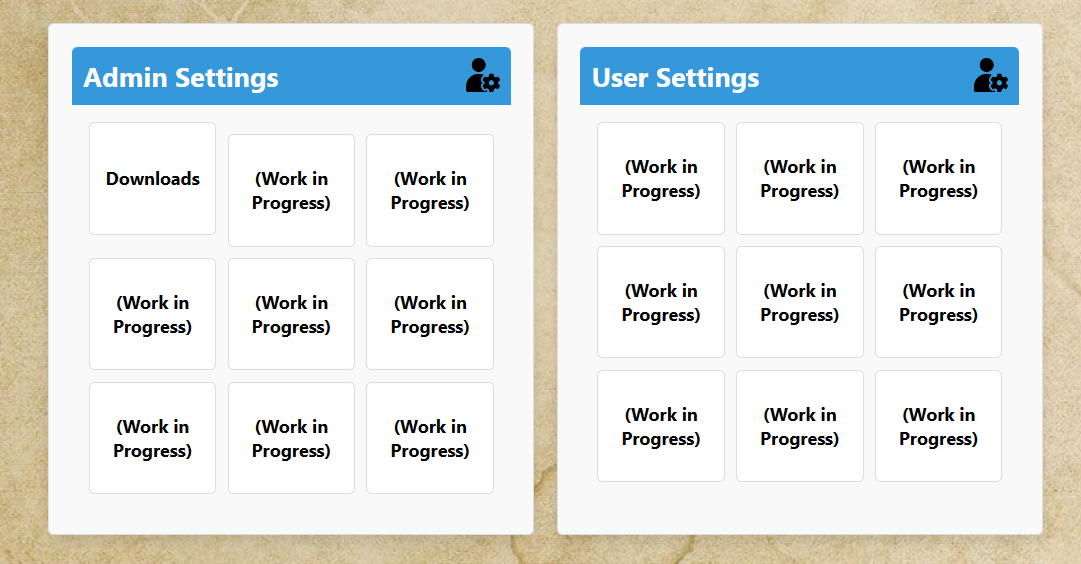
Kétféleképpen különböztetjük meg felhasználóinkat. “*Admin*” és “*User*”.

Mindkét felhasználó más dolgokat lát a weblapon. Más dolgokhoz van hozzáférése.

**Admin felhasználó:**

* Minden olyan dolgott tud, amit a *User* felhasználó, kivévő ő nem tud könyvet kölcsönözni.
* Tud email-t küldeni olyan felhasználóknak, akik kölcsönöztek könyvet.
* Tud email-t küldeni egyesével bármilyen felhasználónak.
* 2-vel több profilkép közül tud választani, mint az alap felhasználó.
* A “Settings” menüpontban látszódik neki egy másik settings komponens, ahol letudja tölteni a “*WPF*”-et.

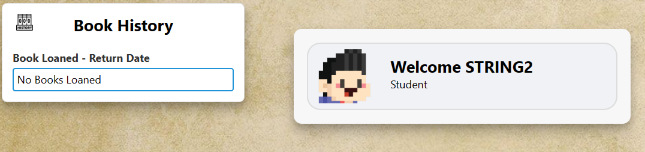
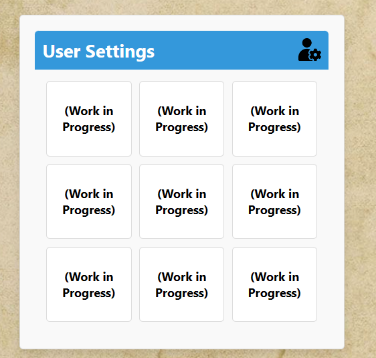
9. Admin profil

****

10. Admin settings

**User felhasználó:**

* Látja az adatbázisban eltárolt könyveket a Navbar “*Books*” menüpontjában.
* A könyvek menüpontban tudja használni a könyv kereső komponenst.
* A “*My Profil*”-ban látja a saját könyv kölcsönzéseit.
* Csak 2 profilkép közül tud választani.
* A “*Settings*”-ben pedig látja a “*User Settings*” komponenst.



12. User settings

11. User profil

A Book menüpontban lehet megnézni az összes olyan könyvet, amie l van tárolva az adatbázisban.

13. Books menüpont kinézete

Amint legörgetünk ennek az oldalnak az aljára, akkor észrevehetjük, hogy a lap alján látható a következő oldalra lépés és a job oldalt egy felfele mutató nyíl jelenik meg.

14. Books weboldal alsó része

A bal oldali keretezésnél láthatóak a: “*First*”,”*Prev*”,”*Next*”,”*Last*”.

Ezek használata:

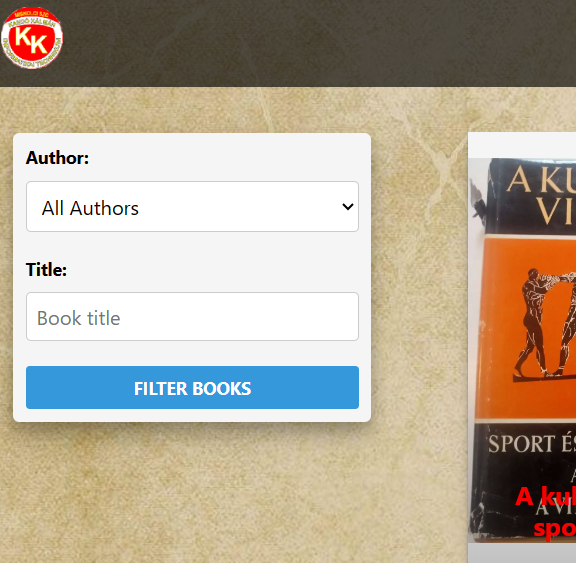
**First:** A legelső oldalra vissza.

**Prev:** Az előző oldalra vissza visz.

**Next:** A következő oldalra visz.

**Last:** Az utolsó oldalra visz.

A jobb oldalon látható piros keretben lévő gomb funkciója, hogyha a felhasználó nem szeretne felgörgetni, akkor arra nyomva felvisz minket az oldal tetejére.

Utoljára a könyveknél, pedig lehet szűrni a könyv írójára és a könyv nevére. A weblap bal fenti oldalán.

15. Könyv kereső ablak

A fenti egy lenyilló menüpont, ahol a könyv szerzőjére tudunk szűrni, alatta pedig a könyv nevére való szűrés lehetőség látható.

# React:

A **React** egy olyan keretrendszer, amely a Webpack segítségével automatikusan lefordítja a React, a JSX és az ES6 kódot a CSS-fájl előtagjainak kezelése közben. A React egy JavaScript-alapú **UI fejlesztői könyvtár**. Bár a React inkább könyvtár, mint nyelv, széles körben használják a webfejlesztésben. A könyvtár először 2013 májusában jelent meg, és mára az egyik leggyakrabban használt frontend könyvtár a webfejlesztéshez. A React különféle bővítményeket kínál a teljes alkalmazás-architektúra támogatásához, mint például a **Flux** és a **React Native**, a puszta felhasználói felületen túl.

A React népszerűsége manapság felülmúlta az összes többi front-end fejlesztési keretet. Íme, miért:

* **Dinamikus alkalmazások egyszerű létrehozása:** A React megkönnyíti a dinamikus webalkalmazások létrehozását, mivel kevesebb kódolást igényel, és több funkcionalitást kínál, szemben a JavaScripttel, ahol a kódolás gyakran nagyon gyorsan bonyolulttá válik.
* **Jobb teljesítmény:** A React Virtuális DOM-ot használ, ezáltal gyorsabban hoz létre webalkalmazásokat. A Virtual DOM összehasonlítja az összetevők korábbi állapotait, és csak a Real DOM megváltoztatott elemeit frissíti, ahelyett, hogy újra frissítené az összes összetevőt, ahogyan azt a hagyományos webalkalmazások teszik.
* **Újrafelhasználható összetevők:** Az összetevők bármely React alkalmazás építőkövei, és egyetlen alkalmazás általában több összetevőből áll. Ezeknek az összetevőknek megvan a saját logikájuk és vezérlőelemeik, és az alkalmazás során újra felhasználhatók, ami viszont drámaian csökkenti az alkalmazás fejlesztési idejét.
* **Egyirányú adatfolyam:** A React egy egyirányú adatfolyamot követ. Ez azt jelenti, hogy a React-alkalmazások tervezésekor a fejlesztők gyakran a szülőkomponensekbe ágyazzák be az alárendelt összetevőket. Mivel az adatok egyetlen irányban áramlanak, könnyebbé válik a hibák hibakeresése.
* **Kis tanulási görbe:** A React könnyen megtanulható, mivel többnyire az alapvető HTML és JavaScript fogalmakat egyesíti néhány előnyös kiegészítéssel. Ennek ellenére, ahogy az más eszközök és keretrendszerek esetében is, el kell szánnia egy kis időt a React könyvtárának megfelelő megértésére.
* **Használható webes és mobilalkalmazások fejlesztésére is:** Azt már tudjuk, hogy a Reactot webes alkalmazások fejlesztésére használják, de ez nem minden. Létezik egy React Native nevű keretrendszer, amely magából a React-ból származik, és amely rendkívül népszerű, és gyönyörű mobilalkalmazások létrehozására használják. Tehát a valóságban a React webes és mobilalkalmazások készítésére is használható.
* **Dedikált eszközök az egyszerű hibakereséshez:** a Facebook kiadott egy Chrome-bővítményt, amellyel a React alkalmazások hibakeresésére használhatók. Ez gyorsabbá és egyszerűbbé teszi a React webalkalmazások hibakeresési folyamatát.

# Előnyei:

* A React.js testreszabott virtuális DOM-ot épít fel. Mivel a JavaScript virtuális DOM gyorsabb, mint a hagyományos DOM, ez javítja az alkalmazások teljesítményét.
* A ReactJS csodálatos felhasználói felületet tesz lehetővé. .
* A React különféle architektúrákat integrál.
* A React egyszerűbbé teszi a teljes szkriptelési környezet folyamatát.
* Megkönnyíti a fejlett karbantartást és növeli a teljesítményt.
* Garantálja a gyorsabb megjelenítést
* A mobilalkalmazások fejlesztésére szolgáló szkript elérhetősége a React legjobb tulajdonsága.
* A ReactJS-t egy nagy közösség támogatja.

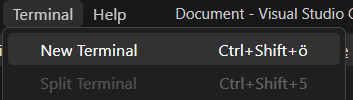
# Hátrányai/Hátrányosságai:

* Csak az alkalmazás fő részével foglalkozik, ennek eredményeként további script-eket kell kiválasztani, ha a fejlesztői eszközök teljes gyűjteményére vágyunk.
* Beépített szkriptet és JSX-et alkalmaz, amit egyes programozók kényelmetlennek találhatnak.
* Számos eszköz és könyvtár létezik, például a Redux és a Reflux, amelyek növelhetik a React teljesítményét. Még magát a Reactot is rendszeresen frissítik. Sajnos ennek van egy árnyoldala is. Egyes fejlesztők úgy gondolják, hogy a React technológiák olyan gyorsan frissülnek és felgyorsulnak, hogy nincs idő dokumentálni vagy megfelelő utasításokat írni. A fejlesztőknek csak ritka szöveges útmutatók maradnak, amelyek nem fedik le a részleteket.
* Míg egyesek azzal érvelnek, hogy a Reactot folyamatosan fejlesztik, és megkönnyítik a munkájukat, mégis előfordulhat, hogy ezt negatívumként fogják fel. Ennek az az oka, hogy minden folyamatosan fejlődik, ami azt eredményezi, hogy a fejlesztők nem elégedettek a folyamatok vagy az új mechanikák folyamatos újratanulásának szükségességével, és előfordulhat, hogy egyes fejlesztők nem érzik jól magukat, ha lépést akarnak tartani ezzel a tempóval.
* A JSX-et a legtöbb ember előnynek tekintené, nem pedig hátránynak, és mégis, botránynak is tekinthető. Ez zavart okozhat azoknak, akik nem nagyon ismerik ezt. A JSX egyszerű szavakkal egy JavaScript-bővítmény, amely olvashatóbbá és tisztábbá teheti a kódot. A HTML és a JavaScript keveréke a JSX-ben sokkal bonyolultabbá teszi a React JS elsajátítását, és ez nem lehet vonzó a törekvő fejlesztők számára. A fejlesztők és a tervezők panaszkodnak a JSX tanulásának bonyodalmaira és az ebből következő nehéz tanulási görbére.

# React felület telepítése:

Első lépésként telepítenünk kell a [Visual Studio Code](https://code.visualstudio.com/)-ot.

Ezután hozzunk létre egy mappát, ahol szeretnénk tárolni az aplikációnkat, majd indítsuk el a Visual Studio Code-ot és keressük ki azt a mappát, amit létrehoztunk neki. (Lásd: 16.1. kép)



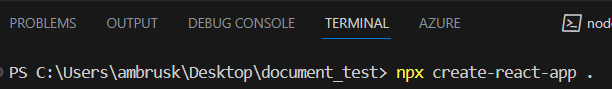
16.3. Visual Studio Code Terminal

16.2. Visual Studio Code megnyitott mappa

16.1. Visual Studio Code mappa kikeresése

Aztán, ha megnyitottuk, akkor ezt láthatjuk (Lásd 16.2. kép). Amint ez megtörtént nyomjunk a “*Terminal*” fülre, majd a “*New Terminal*”-ra (16.3. kép).

Azután a terminálba, amit megnyitottuk írjuk be “*npm create-react-app .*”



17. React telepítése

A terminal, ha kiírja, hogy “*Happy Hacking*”, akkor tudhatjuk, hogy sikeresen letöltöttük a React-ot.

Ezután a terminálba írjuk be, hogy “*npm start*” és az alap weboldalunk el is indult.

# Node.js:

A Node.js egy nyílt forráskódú, többplatformos futási környezet és könyvtár, amelyet webalkalmazások futtatására használnak a felasználó böngészőjén kívül.

Szerveroldali programozásra használják, és elsősorban nem blokkoló, eseményvezérelt kiszolgálókhoz, például hagyományos webhelyekhez és háttér API-szolgáltatásokhoz használják, de eredetileg a valós idejű, push-alapú architektúrákat szem előtt tartva tervezték. Minden böngészőnek megvan a saját JS-motor verziója, a node.js pedig a Google Chrome V8 JavaScript motorjára épül.

Leegyszerűsítve ez azt jelenti, hogy teljes webhelyek futtathatók egységes “**verem**” segítségével, ami gyorssá és egyszerűvé teszi a fejlesztést és a karbantartást, lehetővé téve, hogy a projekt üzleti céljainak elérésére összpontosítson.

Az a tény, hogy a Node.js nyílt forráskódú, azt jelenti, hogy ingyenesen használható, és a fejlesztők globális közössége folyamatosan módosítja és fejleszti. Fontos megérteni a Node.js-t, hogy valójában nem keretrendszer vagy könyvtár - mint a hagyományos alkalmazásszoftvereknél -, hanem a JavaScript egy futási környezet. A futtatókörnyezet (néha RTE-re rövidítve) tartalmaz olyan webes API-kat, amelyekhez a fejlesztő hozzáférhet kód létrehozásához, és egy JavaScript-motort, amely elemzi a kódot. Ez könnyűvé, rugalmassá és könnyen telepíthetővé teszi, minden olyan funkció, amely segít optimalizálni és felgyorsítani az alkalmazásprojektet.

# A Node.js története:

A Node.js-t 2009-ben hozta létre Ryan Dahl. Korábban Dahl bírálta a meglévő népszerű webszerverek és a közös kódolási módszerek által kínált korlátozott lehetőségeket.

Abban az időben a szerverek nehezen kezelték a nagy mennyiségű párhuzamos kapcsolatokat, és a kódok vagy blokkolták a teljes folyamatot, vagy azt sugallták, hogy több veremre volt szükség. Ezek mind olyan problémák voltak, amelyek hátráltatták a vállalkozásokat abban, hogy olyan sokoldalú termékeket hozzanak létre, amelyek megfelelnek a nagy mennyiségű felhasználói igényeknek.

A Node.js kezdeti kiadása csak Linux és Mac OS X operációs rendszereket támogatott. Fejlesztését és karbantartását eleinte a Dahl vezette, később pedig a Joyent, egy szoftver- és szolgáltató cég szponzorálta.

# Node.js architektúra:

A Node.js mechanikája hozzájárul a fejlesztők körében való népszerűségéhez. Míg a legtöbb alternatív futási környezet többszálú feldolgozási modelleket használ, a Node.js mindezt egyetlen szálon belül végzi el.

A többszálú feldolgozási beállításokban minden kiszolgálónak korlátozott szálkészlete van, amelyhez hozzáférhet. Tehát minden alkalommal, amikor egy szerver kérést kap, kihúz egy szálat a készletből, és hozzárendeli a kéréshez, hogy gondoskodjon a feldolgozásról. Ebben az esetben a feldolgozás szinkron és szekvenciális, ami azt jelenti, hogy egyszerre egy műveletet hajtanak végre.

A többszálas feldolgozás során a rendszer minden kéréskor kiválaszt egy szálat, amíg az összes korlátozott szálat fel nem használják. Amikor ez megtörténik, a kiszolgálónak meg kell várnia, amíg egy foglalt szál újra felszabadul. Ez lassú és nem hatékony alkalmazásokat eredményezhet, ami az ügyfélélménytől a potenciális ügyfelek konverziójáig mindenre átütő hatásokhoz vezethet. Ez különösen akkor jelenthet problémát, ha az alkalmazásnak nagyszámú egyidejű ügyfélkéréssel kell foglalkoznia.

A Node.js azonban egyszálú feldolgozást használ. A kettő közötti különbség az, amilyennek képzeli: az egyszálú architektúrák minden kérést egyetlen főszálon dolgoznak fel, eseményhurkokat használva a blokkoló bemeneti/kimeneti műveletek nem blokkoló módon történő futtatására.

Egy egyszálú architektúra elméletileg sokkal gyorsabban és hatékonyabban tud működni és skálázható, mint a többszálas beállítások. Ryan Dahl erre gondolt, amikor először megírta a Node.js-t, és ez nagyban hozzájárult ahhoz, hogy miért olyan népszerű a webalkalmazás-fejlesztők körében.

# Node.js modul-jai:

A Node.js több „modult” tartalmaz, amelyeket egyedi kontextusokban tartanak, így nem zavarják a többi modult, és nem szennyezik a node.js globális hatókörét. Ez kulcsfontosságú a nyílt forráskódú szoftvereknél.

A Node.js modulja egy egyszerű vagy összetett funkció, amely JavaScript-fájlokba van rendezve, és újrafelhasználható a Node.js alkalmazásban. A Node.js-ben háromféle modul létezik: alapmodulok, helyi modulok és harmadik féltől származó modulok.

* Az **alapmodulok** a Node.js alapvető, csupasz funkcióit tartalmazzák. Ezek automatikusan betöltődnek, amikor egy csomóponti folyamat elindul, és a Node.js bináris terjesztésének részét képezik.
* A **helyi modulok** a Node.js alkalmazásban létrehozott modulok. Különböző és további funkciókat tartalmaznak külön fájlokban és mappákban az alapvető funkciócsomaghoz. A helyi modulok csomagolhatók és terjeszthetők a szélesebb Node.js közösség számára is.
* A **harmadik féltől származó modul** egy meglévő kód, amelyet egy harmadik fél írt, és importálható a Node.js alkalmazásba különböző szolgáltatások és funkciók bővítéséhez vagy hozzáadásához.

# Node csomagkezelő:

A Node csomagkezelő (vagy röviden 'npm') csinál néhány dolgot; Először is, online tárhelyként működik a nyílt forráskódú Node.js projektek közzétételéhez. Másodszor, parancssori lehetőségként használják az adott tárhellyel való interakcióhoz, segítve a csomagtelepítést, a verziókezelést és a függőségek kezelését.

Leggyakrabban programok közzétételére, megtalálására, telepítésére és fejlesztésére használják. Lényegében egy hasznos felülettel segíti a fejlesztőket a Node.js eszközök és csomagok legjobb kihasználásában.

# Visual Studio Code:

A Visual Studio Code vagy a VS Code egy szövegszerkesztő, amely több testreszabható funkciót kínál beépülő modulok formájában a fejlesztők számára, hogy egy meglehetősen szelektív fejlesztői környezetet tudjon kialakítani. A VS Code könnyű, és könnyen telepíthető bármilyen platformra.

Miért a **Visual Studio Code**?:

* A VS Code egy **többplatformos kódszerkesztő**, amely könnyen futtatható macOS, Windows és Linux rendszeren. A VS Codes összes webtechnológiája végül Electron Framework-et használ, ami azt jelenti, hogy a VS Code-on keresztül épített alkalmazások karcsúak és zökkenőmentesek frissítésükkor.
* A VS Code viszonylag sokkal gyorsabb, mint a Visual Studio. Az IDE-hez képest gyorsabban tud elindulni.
* A Visual Studio Code **hihetetlenül rugalmas**. Nagyjából bármire képes, amit egy fejlesztő akar. Annak ellenére, hogy kódszerkesztőnek tervezték, megismételheti, amit egy integrált fejlesztői környezet csinál ugyanazon.
* A VS Code a Visual Studióval összehasonlítva meglehetősen **egyszerű** és **zökkenőmentes** a fejlesztési tapasztalatok szempontjából. A VS Code meglehetősen áramvonalas és egyszerű, így a fejlesztők nem keverednek bele semmilyen bonyolultságba.
* A Visual Studio Code a legjobb választás webfejlesztéshez. Hihetetlen támogatást kínál ezernyi eszközével és bővítményével.

# Visual Studio vs Visual Studio Code:

A Visual Studio segítséget nyújt számítógépes programok, webhelyek, webes alkalmazások, mobilalkalmazások és webszolgáltatások fejlesztésében. A Visual Studio vagy az IDE a Microsoft szoftverfejlesztési platformját, azaz a Windows API-t, a Windows Presentation Foundationt, a Windows Forms-t, a Microsoft Silverlightot és a Windows Store-t veszi igénybe a natív kód előállításához és kezeléséhez.

Míg a Visual Studio Code használható a kód írására, szerkesztésére és hibakeresésére, mindezt egy helyen. A VS Code számos programozási nyelvet támogat, amelyekhez a fejlesztőnek nincs szüksége webes támogatásra. Minden megtalálható a beépített többnyelvű támogatásban.

A fejlesztők mindenféle fejlesztéshez támaszkodhatnak a Visual Studio Code-ban, amennyiben azt a megfelelő eszközökkel párosítják. A VS Code beépített JavaScript, Node.js és TypeScript támogatással érkezik. Ha ez nem elég, egyszerűen hozzáadhatja a szükséges támogatást olyan nyelvekhez, mint a Python, C#, PHP, Java és még sok más a bővítmények telepítésével.

Az elsődleges ok, amiért a VS Code jobbnak tűnik a használat szempontjából a Visual Studio vs. Visual Studio Code vitájában, az az, hogy hihetetlen támogatást kínál, minden platformon működik, valamint könnyű és robusztus. Mindenre képes egy helyen.

# Különbségek:

|  |  |
| --- | --- |
| Visual Studio | Visual Studio Code |
| A Visual Studio egy integrált fejlesztői környezet, más néven IDE | A Visual Studio Code egy kódszerkesztő. A fejlesztő könnyen szerkesztheti a kódját. |
| A VS lassabb, ha a különböző platformokon való teljesítményről van szó. A feldolgozási sebesség lassabb. | A VS Code összehasonlítva gyorsabb |
| Visual Studio-nak van egy ingyenes szerkesztője fejlesztőknek, de van egy jobb és fizetett IDE verziója is. | VS Code teljesen ingyenes és nyílt forráskódú. |
| A VS a legjobb és legfejlettebb IntelliSense-t használja. | VS Code-ban nincs IntelliSense |
| A teljes letöltési méret elég nagy | A Visual Studiohoz képest a Visual Studio Code meglehetősen kevés helyet foglal. Nem igényel sok tárhelyet. |
| A VS több helyet igényel a jobb és gördülékenyebb működéshez. | A VS Code viszonylag nem igényel sok helyet a futtatáshoz. Könnyen fut 300 MB rammal. |
| A Visual Studio csak macOS és Windows rendszeren fut. | A VS Code futhat macOS, Windows és Linux alatt is. |
| Nem sok professzionális fejlesztésű bővítmény érhető el a Visual Studio számára. | A VS Code professzionálisan összeállított bővítmények és bővítmények széles skáláját tartalmazza, hogy megfeleljen mindenféle szerkesztési és fordítási igénynek. |

# A WPF:

Windows megjelenítési alaprendszer (WPF), egy felbontásfüggetlen felhasználói felületi keretrendszer, amely vektoralapú renderelő motort használ, és a modern grafikus hardverek előnyeinek kihasználására készült. A GUI keretrendszer lehetővé teszi, hogy olyan alkalmazást hozzon létre, amely a GUI elemek széles skáláját tartalmazza, például címkéket, szövegdobozokat és más jól ismert elemeket.

GUI keretrendszer nélkül ezeket az elemeket manuálisan kellene megrajzolnia, és kezelnie kellene az összes felhasználói interakciós forgatókönyvet, például a szöveg- és egérbevitelt. Ez sok munka, így ehelyett a legtöbb fejlesztő egy GUI keretrendszert fog használni, amely elvégzi az összes alapvető munkát, és lehetővé teszi a fejlesztők számára, hogy nagyszerű alkalmazások készítésére összpontosítsanak.

A WPF átfogó alkalmazásfejlesztési szolgáltatásokat kínál, beleértve az XAML (Extensible Application Markup Language) nyelvet, vezérlőket, adatkötést, elrendezést, 2D és 3D grafikákat, animációkat, stílusokat, sablonokat, dokumentumokat, médiát, szöveget és tipográfiát. A WPF a .NET része, így olyan alkalmazásokat hozhat létre, amelyek a .NET API más elemeit is tartalmazzák.

Tárhelyek: