Project Hephaistos

Készítette
Biliczki Bence
Fodor Valentin Sándor
Szegedi Dávid
Szoftverfejlesztő és -tesztelő szak

Témavezető
Kerényi Róbert Nándor
oktató

Miskolc, 2024

Tartalomjegyzék

Bevezetés

A Project Hephaistos egy innovatív, egyetemisták számára készült órarend-tervező alkalmazás, amely célja, hogy hatékonyan támogassa a hallgatókat az órák, tanórán kívüli tevékenységek és egyéb kötelezettségek rendszerezésében. Az alkalmazás CS nyelven készült, és egy gépészmérnökhallgató igényei alapján lett fejlesztve, különös figyelmet szentelve a funkcionalitásnak és a felhasználóbarát felületnek. Az intelligens ütemezési megoldások lehetővé teszik az egyéni preferenciák, az egyetemi órarend és az időbeli ütközések figyelembevételét.

0.1. Használt Technológiák

A Hephaistos projekt különböző, modern technológiákat alkalmaz a backend, frontend fejlesztéséhez, valamint az adatbázis kezeléséhez. A backend fejlesztés során erőteljes és skálázható megoldásokat biztosítanak, például Node.js használatával, amelyek lehetővé teszik a gyors, aszinkron feldolgozást és az API-k kialakítását. A backend kezelheti a felhasználói adatokat, autentikációt, valamint biztosítja az alkalmazás logikai rétegeit, és kezeli a kommunikációt az adatbázissal.

A frontend fejlesztésére a projekt React környezetben dolgozik, hogy dinamikus, interaktív felhasználói felületeket kínáljon, amelyek gyorsan reagálnak a felhasználói interakciókra. Ezek a könyvtárak lehetővé teszik a komponens-alapú struktúrák kialakítását, így a fejlesztés hatékonyabb és karbantarthatóbb lesz. A CSS preprocesorok és UI frameworkök (pl. Material-UI, Bootstrap) tovább javítják a felhasználói élményt és az alkalmazás megjelenését.

Az adatbázis-kezeléshez a projekt fejlesztése során MySQL relációs adatbázist használunk, amely lehetővé teszi az adatok hatékony tárolását és gyors lekérdezését. A backend és az adatbázis közötti kommunikációt közvetlen SQL lekérdezések segítségével végezzük, amelyek egyszerűsítik az adatok kezelését és lekérdezését az alkalmazás logikai rétegében. A projekt publikálása után az adatbázis az Azure felhőszolgáltatáson keresztül kerül elérhetővé, biztosítva a skálázhatóságot, megbízhatóságot és biztonságot a termelési környezetben. A Hephaistos projekt által használt technológiai stack elősegíti a rugalmas, skálázható és könnyen karbantartható alkalmazások fejlesztését, biztosítva a zökkenőmentes integrációt a backend, frontend és az adatbázis között.

0.1.1. Backend

A projekt backend fejlesztéséhez a .NET Core keretrendszert használjuk, amely egy nyílt forráskódú, platformfüggetlen fejlesztői környezet. A .NET Core lehetővé teszi a fejlesztők számára, hogy modern, felhőalapú alkalmazásokat építsenek, amelyek

nagy teljesítményűek és könnyen skálázhatók. A keretrendszer előnye, hogy több operációs rendszeren is működik, beleértve a Windows, Linux és macOS platformokat, így rugalmas fejlesztési lehetőségeket kínál A .NET Core különösen jól támogatja a mikroszolgáltatás-alapú architektúrákat, így lehetőség van az alkalmazás kisebb, jól elkülöníthető szolgáltatásokra bontására. Ez nemcsak a karbantarthatóságot növeli, hanem biztosítja a magas rendelkezésre állást és skálázhatóságot is. Az integrációs lehetőségek, például a Microsoft Azure-val való zökkenőmentes együttműködés, lehetővé teszik a felhőszolgáltatások egyszerű használatát. A fejlesztés során az Entity Framework Core biztosítja az adatok kezelését, egyszerűsítve az adatbázisokkal való munkát, miközben a web API-k gyors létrehozására is lehetőséget ad, támogatva a RESTful szolgáltatásokat. Ezzel biztosítva van a könnyű adatkommunikáció a frontend és backend között.

ASP.NET Core

Az ASP.NET Core a .NET Core keretrendszer része, amelyet webalkalmazások és API-k fejlesztésére használnak. Az ASP.NET Core előnyei közé tartozik a magas teljesítmény, a platformfüggetlenség és a moduláris felépítés. Az ASP.NET Core támogatja a modern webfejlesztési szabványokat és eszközöket, mint például a dependency injection, a middleware-ek és a RESTful API-k.

Az ASP.NET Core alkalmazások könnyen telepíthetők és skálázhatók, így ideálisak a felhőalapú környezetekben történő futtatásra. Az ASP.NET Core lehetővé teszi a fejlesztők számára, hogy gyorsan és hatékonyan hozzanak létre biztonságos és megbízható webalkalmazásokat.

0.1.2. Adatbázis

Az adatok tárolására MySQL adatbázist használunk. A MySQL egy népszerű, nyílt forráskódú relációs adatbázis-kezelő rendszer, amely nagy teljesítményt és megbízhatóságot kínál. A PHPMyAdmin eszközt használjuk az adatbázis kezelésére, amely egy webalapú felületet biztosít az adatbázisok kezeléséhez.

0.1.3. Frontend

A React egy népszerű JavaScript-könyvtár, amelyet dinamikus és interaktív felhasználói felületek (UI) fejlesztésére használnak. A React lehetővé teszi a fejlesztők számára, hogy komponens-alapú módon építsenek fel alkalmazásokat, ahol minden komponens saját állapotot és viselkedést tartalmaz. A virtuális DOM használata gyorsabb renderelést biztosít, mivel csökkenti a tényleges DOM-műveletek számát. A React alkalmazásokat könnyen skálázhatjuk és karbantarthatjuk, és a könyvtár számos kiegészítő

eszközt kínál, mint például Redux az állapotkezeléshez, vagy React Router a navigációhoz. Emellett a React lehetőséget ad mobilalkalmazások készítésére is React Native segítségével.

React

A React egy komponens-alapú JavaScript könyvtár, amelyet a Meta fejlesztett ki és 2013-ban adtak ki. A React fő célja a felhasználói felületek egyszerűbb és hatékonyabb fejlesztése. A React lehetővé teszi a fejlesztők számára, hogy újrafelhasználható komponenseket hozzanak létre, amelyek könnyen karbantarthatók és bővíthetők.

A React egyik legfontosabb jellemzője a virtuális DOM (Document Object Model), amely javítja az alkalmazás teljesítményét azáltal, hogy minimalizálja a valódi DOM manipulációkat. A React támogatja a JSX (JavaScript XML) szintaxist, amely lehetővé teszi a HTML-szerű kód írását a JavaScript-ben, növelve a kód olvashatóságát és karbantarthatóságát.

A React ökoszisztémája számos kiegészítő könyvtárat és eszközt tartalmaz, mint például a React Router a kliensoldali útvonalkezeléshez és a Redux az állapotkezeléshez. Ezek az eszközök segítenek a fejlesztőknek hatékonyabb és skálázhatóbb alkalmazásokat létrehozni.

0.1.4. Fejlesztői Eszközök

A fejlesztés során különböző eszközöket használunk a hatékonyság növelése és a hibakeresés megkönnyítése érdekében:

- **Visual Studio Code (VSCode)**: Egy népszerű kódszerkesztő, amely számos bővítménnyel rendelkezik, például az `ESLint` és `Prettier` segítségével.
- **React Developer Tools**: Egy böngészőbővítmény, amely lehetővé teszi a React komponensek és állapotok ellenőrzését a böngészőben.
- **PHPMyAdmin**: Egy webalapú eszköz a MySQL adatbázisok kezelésére.

0.2. Projekt Felépítése

A Hephaistos projekt három fő komponensből áll: backend, frontend és adatbázis. Az alábbiakban bemutatjuk ezeknek a komponenseknek a felépítését és funkcióit.

0.3. Órarend Generátor Funkció

A Project Hephaistos backend tartalmaz egy órarend generátor funkciót, amely segíti a felhasználókat az optimális órarend összeállításában. Ez a funkció figyelembe veszi a

felhasználó által teljesített tantárgyakat, a tantárgyak előfeltételeit, valamint az időbeli ütközéseket.

0.3.1. TimetableGenerator Osztály

A `TimetableGenerator` osztály felelős az órarend generálásáért. Az osztály a következő fő metódust tartalmazza:

- **GenerateClosestTimetable**: Ez a metódus a megadott kreditszám alapján generál egy optimális órarendet. A metódus figyelembe veszi az elérhető tantárgyakat és az ütemezési konfliktusokat.

Az alábbi kód egy részletet mutat be a `GenerateClosestTimetable` metódusból:

```
public (IEnumerable<SubjectSchedule> Timetable, IEnumerable<Subject> OmittedSubjects) GenerateClosestTimetable(
    int creditValue,
    List<Subject> _availableSubjects,
    List<SubjectSchedule> _existingSchedules
) {
    var selectedSubjects = new List<Subject>();
    var omittedSubjects = new List<Subject>();
    var selectedSchedules = new List<SubjectSchedule>();
    int totalCredits = 0;

    foreach (var subject in _availableSubjects) {
        if (totalCredits + (subject.CreditValue ?? 0) > creditValue) {
            omittedSubjects.Add(subject);
            continue;
        }

        var subjectSchedules = _existingSchedules.Where(s => s.SubjectId == subject.Id).ToList();

        if (!subjectSchedules.Any(schedule => IsOverlapping(schedule, selectedSchedules))) {
            selectedSubjects.Add(subject);
            selectedSchedules.AddRange(subjectSchedules);
            totalCredits += subject.CreditValue ?? 0;
        } else {
            omittedSubjects.Add(subject);
        }
```

```
    }

    return (selectedSchedules, omittedSubjects);
}
```

0.3.2. TimetableGeneratorController Végpont

A `TimetableGeneratorController` egy API végpontot biztosít az órarend generálásához. A végpont a következőképpen működik:

- **POST /api/timetablegenerator/generate**: Ez a végpont fogadja a felhasználó által megadott kreditszámot, és visszaadja az optimális órarendet, valamint azokat a tantárgyakat, amelyek kimaradtak az ütemezésből.

Az alábbi kód egy részletet mutat be a végpont implementációjából:

```
[HttpPost("generate")]
[Authorize]
public async Task<IActionResult> GenerateTimetable([FromBody]
   int creditValue, [FromHeader(Name = "Authorization")] string
   Authorization) {
    var userId = _jwtHelper.ExtractUserIdFromToken(
       Authorization);
    var availableSubjects = GetAvailableSubjects(userId.Value);
    var existingSchedules = _context.Subjectschedules
        .Where(schedule => availableSubjects.Any(subject =>
           subject.Id == schedule.SubjectId))
        .ToList();

    var (timetable, omittedSubjects) = _timetableGenerator.
       GenerateClosestTimetable(creditValue, availableSubjects,
       existingSchedules);

    return Ok(new {
        Timetable = timetable.Select(schedule => new {
            SubjectName = schedule.Subject?.Name,
            DayOfWeek = schedule.DayOfWeek,
            StartTime = schedule.StartTime,
            EndTime = schedule.EndTime
        }),
        OmittedSubjects = omittedSubjects.Select(subject => new
            {
```

```
                SubjectName = subject.Name
            })
    });
}
```

0.3.3. Felhasználói Élmény

A funkció biztosítja, hogy a felhasználók egy könnyen értelmezhető és áttekinthető órarendet kapjanak, amely tartalmazza az órák nevét, időpontját és napját, így a tanulók gyorsan ráláthatnak napi és heti beosztásukra. Az órarend minden órát megfelelő időpontban és napra rendezve jelenít meg, figyelembe véve a tantárgyak közötti ütközéseket és azok optimális elosztását. A felhasználói élmény javítása érdekében az órarend könnyen navigálható, és biztosítja, hogy a tanulók minden szükséges információt megtaláljanak. Amennyiben egy tantárgy valamilyen okból nem került be az órarendbe, a rendszer egy külön listában jeleníti meg azokat az órákat, amelyek kimaradtak, lehetővé téve a felhasználók számára, hogy tisztában legyenek azzal, mely tantárgyak nem szerepelnek az ütemezésben. Ez segít elkerülni az esetleges hiányosságokat, és biztosítja, hogy a tanulók tisztában legyenek azokkal a tantárgyakkal, amelyeket még nem tudtak beilleszteni. Továbbá, a rendszer lehetőséget ad arra, hogy a felhasználók az egyes órákra kattintva részletes információkat is elérhessenek, például az oktató nevét, az órák helyszínét és esetleges házi feladatokat vagy egyéb fontos tudnivalókat. A kimaradó tantárgyak esetében a felhasználók szűrőfunkciókat is alkalmazhatnak, így könnyen kereshetnek a listában, és információkat kaphatnak arról, hogy miért nem szerepel az adott tantárgy az órarendjükben, vagy milyen alternatív megoldások léteznek annak beillesztésére.

1. fejezet

Relációs adatkezelő rendszerek

1.1. Relációs adatbázis

Relációs adatbázisnak nevezzük azokat az adatbázisokat, amelyek a relációs adatmodell elvein alapulnak. Ezekben az adatok táblázatos formában, ún. relációkban (táblákban) vannak tárolva, ahol a sorok egyedi rekordokat, az oszlopok pedig mezőket (attribútumokat) jelentenek. A relációs adatbázis több ilyen táblát tartalmaz, amelyeket egymással kapcsolatokon (kulcsokon) keresztül lehet összekötni. Az ilyen adatbázisokat relációs adatbázis-kezelő rendszerek (pl. MySQL, PostgreSQL, SQL Server) segítségével lehet létrehozni, módosítani és kezelni. Az adatok lekérdezésére és módosítására jellemzően az SQL nyelvet használjuk. A relációs adatbázisok erőssége a strukturált adattárolás, a konzisztencia és az adatintegritás biztosítása.

1.1.1. A relációs adatbáziskezelő rendszerek

A relációs adatbáziskezelő rendszerek (RDBMS, azaz Relational Database Management System) olyan szoftverek, amelyek az adatokat táblák formájában kezelik. Minden tábla oszlopokból és sorokból áll, ahol az oszlopok különböző attribútumokat jelölnek, míg a sorok egyedi rekordokat reprezentálnak. Az RDBMS-ekben a táblák között relációk, azaz kapcsolatok hozhatók létre, így biztosítva az adatok egymáshoz kapcsolódó kezelését.

Az RDBMS-ek egyik legfontosabb jellemzője a strukturált lekérdezőnyelv, azaz az SQL (Structured Query Language), amely lehetővé teszi az adatok lekérdezését, módosítását és kezelését. Az adatokat kulcsok segítségével kapcsoljuk össze, mint például az elsődleges kulcs (primary key) és az idegen kulcs (foreign key). Az adatok normalizálásával csökkenthetők az ismétlődések, és fenntartható az adatintegritás. A relációs adatbáziskezelő rendszerek széles körben elterjedtek különböző üzleti és technológiai területeken, mivel lehetővé teszik az adatok hatékony tárolását és kezelését, valamint a nagy mennyiségű adat gyors lekérdezését.

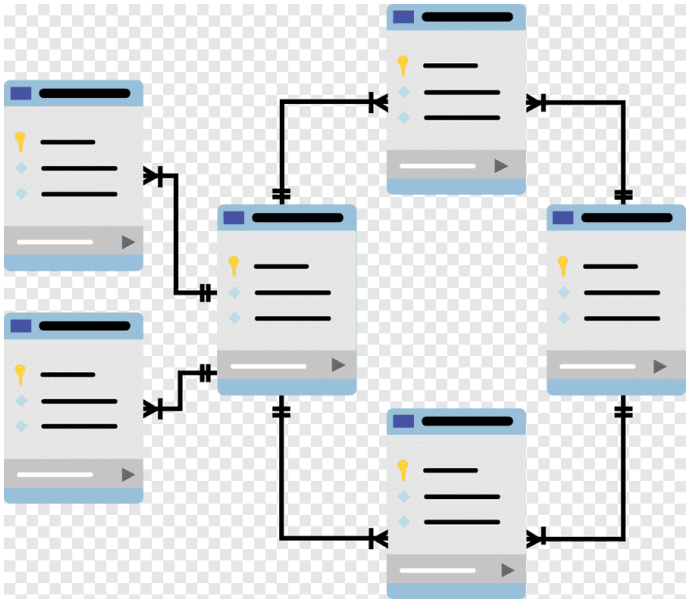
1.1.2. MySQL és MariaDB, valamint PHPMyAdmin

A MySQL és a MariaDB két népszerű, nyílt forráskódú relációs adatbáziskezelő rendszer. Mindkettő hasonló struktúrát és működési elveket követ, mivel a MariaDB a MySQL-ből származik. A MySQL-t eredetileg a Sun Microsystems fejlesztette ki, majd a Sun felvásárlása után az Oracle tulajdonába került. A MariaDB-t a MySQL eredeti fejlesztői hozták létre az Oracle irányításától való függetlenség megőrzése érdekében, és azóta külön fejlesztési irányt követ.

Mindkét rendszer könnyen kezelhető, nagy teljesítményt és skálázhatóságot kínál, így gyakran használják webalkalmazások és más dinamikus rendszerek adatbázisaként. A MySQL és a MariaDB támogatják a különféle táblastruktúrákat (például InnoDB és MyISAM), a tranzakciókezelést, valamint a különféle biztonsági mechanizmusokat, mint például a felhasználói jogosultságok finomhangolása.

A PHPMyAdmin egy népszerű, nyílt forráskódú webes eszköz, amelyet kifejezetten MySQL és MariaDB adatbázisok kezelésére fejlesztettek ki. Lehetővé teszi a felhasználók számára, hogy grafikus felületen keresztül, SQL parancsok írása nélkül menedzseljék az adatbázisokat. Ezzel az eszközzel könnyen létrehozhatók és módosíthatók táblák, kezelhetők a felhasználói jogosultságok, és futtathatók különböző SQL lekérdezések.

A PHPMyAdmin előnye, hogy böngésző alapú, így bármilyen eszközről könnyen elérhető, nem szükséges hozzá parancssori ismeret. A felhasználók könnyen átláthatják az adatbázis struktúráját, létrehozhatnak exportálási és importálási műveleteket, valamint automatikusan generált diagramok segítségével áttekinthetik a táblák közötti kapcsolatokat. Ezen kívül a PHPMyAdmin lehetőséget biztosít biztonsági mentések készítésére, így a felhasználók megőrizhetik és helyreállíthatják az adatokat.



1.1. ábra. Relációs adatbázis

2. fejezet

React Fejlesztői Környezet

2.1. Bevezetés

A React egy népszerű JavaScript-könyvtár, amelyet a felhasználói felületek egyszerűbb fejlesztésére használnak. Ezt a Meta fejlesztette, és 2013-ban adták ki. Azóta széles körben alkalmazzák a webes fejlesztésben, különösen az egyoldalas alkalmazások (SPA) készítéséhez. Ez a szakdolgozat a React fejlesztői környezetek felépítését, a szükséges eszközöket és a hatékony fejlesztéshez szükséges alapelveket vizsgálja.

2.2. React Fejlesztői Környezet

A React fejlesztői környezet beállítása az első lépés a sikeres alkalmazásfejlesztéshez. Ehhez szükséges a Node.js és az npm telepítése, amelyek biztosítják a JavaScript futtatókörnyezetet és a csomagkezelést. A React alkalmazások gyors indításához a create-react-app eszköz használata javasolt, amely előre konfigurált fejlesztési környezetet biztosít. A React alapja a komponens-alapú felépítés, amely lehetővé teszi az alkalmazások moduláris felépítését és könnyebb karbantartását. A JSX szintaxis segítségével a JavaScript és HTML kód egyszerűen kombinálható, növelve a kód olvashatóságát. A state és props kezelésével pedig dinamikusan irányíthatjuk az alkalmazás adatfolyamát és a felhasználói interakciókat. A megfelelő környezet beállításával és az alapok elsajátításával gyorsan elérhetjük a React alkalmazás fejlesztésének sikeres elindítását.

2.2.1. React Telepítése és Projektindítás

A React telepítéséhez először szükségünk van a Node.js és npm (Node Package Manager) telepítésére, amelyek segítségével kezelhetjük a JavaScript könyvtárakat és függőségeket. Miután a Node.js és az npm telepítve van, létrehozhatunk egy új React projektet az `create-react-app` eszközzel.

Az alábbi parancsokkal létrehozható egy új React projekt:

```
npx create-react-app my-app
cd my-app
npm start
```

A `create-react-app` gyorsan felállít egy alap React projektstruktúrát, amely a következő fájlokat tartalmazza:

- `src/` mappa: a projekt alapvető forráskódját tartalmazza, és ebben található minden olyan fájl, amely az alkalmazás működéséhez szükséges. A mappa tartalmazza a komponensek fájljait, amelyek a felhasználói felület különböző részeit alkotják, valamint az alkalmazás logikáját, amely kezeli a különböző funkcionalitásokat, adatkezelést és interakciókat. A komponensek hierarchikus felépítésében a fejlesztők könnyen megtalálják a különböző funkciókhoz tartozó részeket, ezáltal segítve a karbantartást és a bővítést.

- `public/` mappa: a projekt statikus fájljait tartalmazza, amelyek közvetlenül elérhetők a böngészőből, és nem igényelnek semmilyen feldolgozást a build folyamat során. Ide tartoznak a különböző statikus fájlok, mint például képek, ikonszettek, betűtípusok és más, az alkalmazásban használt fix fájlok, amelyek nem változnak az alkalmazás működése közben.

- `node_modules/`: tartalmazza az alkalmazás számára szükséges összes külső függőséget és csomagot, amelyeket a projekt telepítése során a npm (Node Package Manager) segítségével töltünk le. Ebbe a mappába kerülnek az összes könyvtár és csomag, amelyeket a projekt kódja futtatásához szükséges, például React, utility könyvtárak, tesztelési keretrendszerek, és egyéb fejlesztői eszközök.

2.2.2. Fejlesztői Eszközök

A React fejlesztéséhez számos fejlesztői eszköz áll rendelkezésünkre, amelyek növelhetik a termelékenységet és a hibakeresés hatékonyságát.

React Developer Tools

A React Developer Tools egy hivatalos böngészőbővítmény, amely lehetővé teszi a React alapú webalkalmazások mélyreható vizsgálatát és hibakeresését közvetlenül a böngésző fejlesztői eszközein belül. A bővítmény segítségével a fejlesztők átlátható módon megtekinthetik az alkalmazás teljes komponensfáját, beleértve a komponensek közötti hierarchiát és kapcsolatokat. Különösen hasznos funkciója, hogy lehetővé teszi az egyes komponensekhez tartozó props és state értékek valós idejű megfigyelését, illetve ezek változásainak nyomon követését a felhasználói interakciók során. A modern React fejlesztés során gyakran használt Hook-ok – mint például a useState, useEffect, vagy useContext – szintén jól követhetők a bővítményen keresztül, így a fejlesztők részletes információt kaphatnak az alkalmazás belső állapotairól. Ezen kívül a React Developer

Tools egy beépített profilozó eszközt is tartalmaz, amely lehetővé teszi a komponens-renderelések teljesítményének elemzését, megmutatva, hogy mely komponensek renderelődnek újra, milyen gyakorisággal, és mennyi idő alatt. Mindezek együttesen nagyban hozzájárulnak a hatékonyabb hibakereséshez, az optimalizáláshoz, és az alkalmazás működésének jobb megértéséhez, ezáltal elengedhetetlenné téve a bővítményt minden komolyabb React fejlesztési projekt során.

2.3. React Alapjai

A React egy összetevő-alapú megközelítést alkalmaz, amely lehetővé teszi, hogy az alkalmazás kisebb, újrafelhasználható komponensekből épüljön fel. Mivel minden komponens saját állapotot és viselkedést kezelhet, az alkalmazás könnyen kezelhető és bővíthető. Ez a moduláris felépítés elősegíti a kód karbantartását, mivel a különböző komponensek függetlenül fejleszthetők, tesztelhetők és frissíthetők anélkül, hogy az alkalmazás többi része károsodna. A komponens-alapú struktúra lehetővé teszi a kód logikai szétválasztását és az újrafelhasználhatóságot. Egy-egy komponens akár több helyen is alkalmazható, így a fejlesztők elkerülhetik az ismétlődő kódok írását, ami növeli a fejlesztés hatékonyságát. Emellett a komponensek izoláltsága segíti a hibakeresést, mivel könnyen beazonosítható, hogy melyik rész felelős egy adott viselkedésért vagy hibáért. Ez a megközelítés nemcsak a fejlesztés gyorsításában, hanem a csapatmunkában is előnyt jelent, mivel a különböző fejlesztők párhuzamosan dolgozhatnak a különböző komponenseken. A React ezáltal biztosítja, hogy az alkalmazás nagy méretű és összetett funkciók mellett is skálázható és rugalmas maradjon, miközben a kód alapja átlátható és jól struktúrált.

Ez a változat hangsúlyozza a moduláris felépítést, az újrafelhasználhatóságot és a karbantarthatóságot, és azt, hogy ezek hogyan segítik a gyorsabb fejlesztést és a jobb csapatmunkát. Ha bármit szeretnél finomítani, csak szólj!

2.3.1. Komponensek és JSX

A React komponensek JavaScript függvények vagy osztályok, amelyek HTML-szerű kódot, úgynevezett JSX-et használnak. A JSX egy szintaxisbővítés, amely lehetővé teszi, hogy a JavaScript kódban HTML elemeket írjunk, miközben megőrizzük a JavaScript teljes funkcionalitását. A JSX segít abban, hogy a kód olvashatóbbá váljon, mivel a logikai struktúrák és a megjelenítés közvetlenül egymás mellett találhatóak. A JSX-ben írt HTML szerű kód valójában nem közvetlenül HTML, hanem JavaScript kifejezésekké fordul le, amit a React futtatáskor képes renderelni. Ez lehetővé teszi a dinamikus tartalom generálását, mivel a komponens kódjában szereplő adatokat vagy logikai feltételeket is könnyen integrálhatjuk a megjelenítésbe. A JSX alapvetően javítja a kód

karbantartását és bővíthetőségét, mivel az alkalmazás minden része (adatok, logika és megjelenítés) egy helyen, tiszta és érthető módon van strukturálva. A JSX nemcsak a statikus HTML elemeket támogatja, hanem lehetőséget ad arra is, hogy JavaScript kifejezéseket ágyazzunk be, mint például változókat, függvényhívásokat vagy ciklusokat. Mivel végül a JSX JavaScript-re fordítódik, így a böngésző számára is érthető és végrehajtható kódot eredményez, miközben megőrzi a dinamikus viselkedést és a felhasználói interakciók kezelését. A React alkalmazásokban a JSX alapú komponensek lehetővé teszik a újrafelhasználható kódok létrehozását, így a fejlesztők gyorsan és hatékonyan tudják kezelni az alkalmazás felhasználói felületét. A JSX és a React együtt biztosítják, hogy az alkalmazás rugalmasan alkalmazkodjon a különböző felhasználói igényekhez, miközben könnyen bővíthető és karbantartható marad.

Microsoft Azure Elérhetőség

A Project Hephaistos alkalmazásunkat a Microsoft Azure felhőszolgáltatás segítségével publikáltuk, amely lehetővé teszi a skálázható és megbízható működést. Az Azure biztosítja az alkalmazás számára a szükséges infrastruktúrát, beleértve a webalkalmazás hosztolását, az adatbázis-kezelést és a biztonságos hozzáférést.

Az alkalmazás elérhető az alábbi linken: https://hephaistos-backend-c6c5ewhraedvgzex.germanywestcentral-01.azurewebsites.net/index.html

Azure Szolgáltatások Használata

A projekt során az alábbi Azure szolgáltatásokat használtuk:

- **Azure App Service**: Az alkalmazás hosztolására és futtatására használtuk. Ez egy teljesen menedzselt platform, amely lehetővé teszi a gyors telepítést és a folyamatos integrációt.
- **Azure SQL Database**: Az alkalmazás adatainak tárolására és kezelésére szolgál. Ez egy skálázható, biztonságos és megbízható relációs adatbázis-kezelő rendszer.
- **Azure Monitor**: Az alkalmazás teljesítményének és állapotának nyomon követésére használtuk. Ez segít az esetleges problémák gyors azonosításában és megoldásában.
- **Azure Active Directory (AAD)**: A felhasználók hitelesítésére és az alkalmazás biztonságának növelésére használtuk.

Előnyök

A Microsoft Azure használata számos előnyt biztosít a projekt számára:

- **Skálázhatóság**: Az alkalmazás könnyen skálázható a felhasználói igények növekedésével.
- **Megerősített Biztonság**: Az Azure beépített biztonsági funkciói, például a titkosítás és a hozzáférés-szabályozás, garantálják az adatok védelmét.
- **Globális Elérhetőség**: Az Azure globális adatközpont-hálózata lehetővé teszi az alkalmazás gyors és megbízható elérését a világ bármely pontjáról.
- **Költséghatékonyság**: Az Azure rugalmas árazási modellje lehetővé teszi, hogy csak a ténylegesen használt erőforrásokért fizessünk.

A Microsoft Azure segítségével biztosítjuk, hogy a Project Hephaistos alkalmazás megbízhatóan és hatékonyan szolgálja ki a felhasználók igényeit.

```
import React from 'react';


function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}

export default Welcome;
```

2.3.2. Állapotkezelés a React-ben

A React komponensek állapotkezelése alapvető fontosságú a dinamikus, interaktív felhasználói felületek létrehozásához. Az állapot (state) lehetővé teszi, hogy a komponens megőrizze az aktuális adatállapotát, és reagáljon a felhasználói interakciókra, például kattintásokra vagy űrlapváltozásokra. Az useState Hook segítségével egyszerűen létrehozhatunk és módosíthatunk állapotokat, amelyek hatására a komponens újrarenderelődik, és a felület frissül. Egyszerűbb alkalmazásoknál elegendő a helyi állapotkezelés, míg komplexebb esetekben érdemes globális megoldásokat is alkalmazni, például a Context API-t vagy Reduxot. Az állapotkezelés tehát kulcsfontosságú eszköz a React-komponensek viselkedésének irányításához és a felhasználói élmény kialakításához.

useState Hook

A `useState` egy hook, amely lehetővé teszi az állapotkezelést a funkcionális komponensekben. Az alábbi példa egy egyszerű számlálót mutat be, amely a `useState` használatával kezeli az állapotot:

```
import React, { useState } from 'react';

function Counter() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>You clicked {count} times</p>
            <button onClick={() => setCount(count + 1)}>
                Click me
            </button>
        </div>
    );
}
```

```
15
16 export default Counter;
```

2.3.3. Props

A React komponensek közötti kommunikációt az úgynevezett `props` (properties) segítségével valósítják meg. A `props` adatokat továbbítanak az egyik komponensből a másikba.

```
1 function Greeting(props) {
2     return <h1>Hello, {props.name}</h1>;
3 }
4
5 function App() {
6     return <Greeting name="React Developer" />;
7 }
8
9 export default App;
```

2.3.4. Komponensek Típusai

A React komponensek két fő típusa:

- **Funkcionális komponensek**: egyszerű JavaScript függvények, amelyek a `props`-ot paraméterként veszik át és JSX-et adnak vissza.
- **Osztály alapú komponensek**: osztályok, amelyek bonyolultabb állapotkezelést és életciklus-módszereket tesznek lehetővé.

2.4. Jelszó Módosítása Funkció

A React frontend alkalmazás tartalmaz egy jelszó módosítási funkciót, amely lehetővé teszi a felhasználók számára, hogy biztonságosan megváltoztassák a jelszavukat. A funkció célja, hogy a felhasználók könnyen és gyorsan frissíthessék fiókjuk biztonsági beállításait, miközben minden szükséges lépést biztosít a személyes adataik védelme érdekében. A jelszó módosítása során a rendszer a legjobb biztonsági gyakorlatokat alkalmazza, mint például a jelszó erősségének ellenőrzését, hogy biztosítsa a felhasználók adatainak védelmét. Ez a funkció a következő komponensekből áll:

2.4.1. ChangePasswordForm Komponens

A `ChangePasswordForm` egy React komponens, amely biztosítja a jelszó módosításához szükséges űrlapot. Az űrlap tartalmaz mezőket a régi jelszó, az új jelszó és az új jelszó megerősítésének megadásához. Az alábbi kód egy részletet mutat be a komponensből:

```
1   import React, { useState } from 'react';
2
3   function ChangePasswordForm() {
4       const [oldPassword, setOldPassword] = useState('');
5       const [newPassword, setNewPassword] = useState('');
6       const [confirmPassword, setConfirmPassword] = useState(
            '');
7
8       const handleChangePassword = () => {
9           if (newPassword !== confirmPassword) {
10              alert('Az új jelszavak nem egyeznek!');
11              return;
12          }
13          // Jelszó módosítási logika
14      };
15
16      return (
17      <form>
18      <input type="password" placeholder="Régi jelszó"
            onChange={(e) => setOldPassword(e.target.value)} />
19      <input type="password" placeholder="Új jelszó" onChange
            ={(e) => setNewPassword(e.target.value)} />
20      <input type="password" placeholder="Új jelszó megerősít
            ése" onChange={(e) => setConfirmPassword(e.target.
            value)} />
21      <button type="button" onClick={handleChangePassword}>
```

```
            Jelszó módosítása</button>
        </form>
        );
    }
```

2.4.2. Backend Integráció

A jelszó módosítási funkció a backend API-val kommunikál, hogy ellenőrizze a régi jelszót és frissítse az új jelszót az adatbázisban. Az API végpontja a következő:

```
POST https://localhost:5001/change-password
Headers: {
    Authorization: Bearer <token>
}
Body: {
    "oldPassword": "current_password",
    "newPassword": "new_password"
}
```

2.4.3. Felhasználói Élmény

A funkció biztosítja, hogy a felhasználók értesítést kapjanak a sikeres vagy sikertelen jelszó módosításról. A hibák, például a nem egyező jelszavak vagy a helytelen régi jelszó, azonnal megjelennek az űrlapon.