### **1. hét**

A projekt ötletének kidolgozása volt a fő fókusz. Egy dolgozói beléptetőrendszer alapjait fektettük le, amely képes RFID-alapú azonosításra, időnaplózásra és havi riportálásra. Meghatároztuk a célcsoportot (pl. gyártó cégek, irodák), valamint a rendszer fő funkcióit. Előkészítettük a fejlesztési környezetet Visual Studio-ban és Visual Studio Code-ban. Átgondoltuk a lehetséges fejlesztési ütemtervet, valamint azt, hogy a projekt hogyan bővíthető majd mobilos platformra is MAUI segítségével. Ez volt a tervezés és koncepcióalkotás fázisa.

### **2. hét**

Kutatást végeztünk más hasonló beléptetőrendszerek technikai megoldásairól. Különösen a mobil- és webes platformok közötti adatkommunikációra koncentráltunk. Részletesen áttanulmányoztam a .NET MAUI dokumentációt, hogy megtudjam, milyen REST API kapcsolatokat lehet kiépíteni vele. Emellett először próbáltam ki MYSQL adatbázist ASP.NET környezetben, amit egyszerűen lehet fejleszteni és karbantartani. Az ötletet tovább finomítottuk a tapasztalatok alapján, és elkészítettünk egy MVP (minimum viable product) tervet is.

### **3. hét**

Tudtam hogy az Klebelsberg Kuno könyvtárban is használnak egy ilyern beléptető rendszert, úgyhogy egy ismerősömet aki ott dolgozik, megkértem hogy mutassa meg hogy milyen használat közben. Ezek után elkezdtük az első vázlatokat. A cél egy olyan softver, ahol a dolgozók digitálisan kezelhetik munkaidejüket, és a vezetők valós időben ellenőrizhetik az adatokat. Fontos szempont volt a felhasználóbarát kialakítás és a mobilos elérés biztosítása. Az ötletet egy prezentáció formájában is rögzítettük, amelyben bemutattuk a lehetséges felhasználási területeket. Gyakran konzultáltáltunk Ottiliával a technikai kivitelezhetőségről. A Klebelsberg könyvtárban használton kivül más hasonló rendszerek működését és felépítését is megnéztük.

### **4. hét**

Áttekintettük a projekt technológiai hátterét és a szükséges eszközöket. ASP.NET Core Web API-t választottuk backendnek, MySQL adatbázissal, MAUI-t a mobilalkalmazáshoz, és Svelte-et a frontendhez. Ellenőriztem, hogy ezek a technológiák mennyire kompatibilisek egymással, és hogyan lehet őket REST API-n keresztül összekötni. Kipróbáltam egy egyszerű Svelte fetch példát, amely sikeresen lekérte az adatokat a backendtől. Ugyanígy, MAUI-ból is létrehoztam egy alap projektet, amely sikeresen hívta az API egyik végpontját. Szerencsére óráinkon is sok példát néztünk erre a felhasználásra, úgyhogy viszonylag könnyen boldogultunk.

### **5. hét**

Rögzítettük a projekt főbb funkcióit és a megvalósítás sorrendjét. A rendszer célja, hogy valós időben naplózza a dolgozók be- és kilépéseit, és a későbbiekben ezek alapján havi riportot készítsen. A bejelentkezés, admin kezelés, jogosultságok szétválasztása és az időnaplózás mind külön modulként lesz kezelve. Megrajzoltuk a backend-frontend kapcsolatokat egy egyszerű architektúra diagrammal. A fejlesztési fázisokat heti bontásban megterveztük, figyelembe véve a MAUI mobilalkalmazás beépítését is. Ezzel kialakult a fejlesztési ütemterv..

### **6. hét**

Elkezdtük az ASP.NET Core backend projekt létrehozását. Beállítottuk a projektet Employee\_Attendance\_Api néven, és konfiguráltuk az SQLite kapcsolatot az appsettings.json segítségével. Elindítottuk a Swagger integrációt is, hogy könnyebben tudjuk tesztelni a fejlesztés alatt álló végpontokat. Az alapértelmezett middleware-k beállítása után elkezdtük hozzáadni a szükséges kontroller osztályokat. A CORS szabályozást is bevezettük, hogy majd a Svelte frontend kommunikálhasson az API-val. Az autentikációs és jogosultsági szolgáltatásokat előkészítettük, de még nem aktiváltuk.

### **7. hét**

Elkészítettük az adatbázis teljes szerkezetét. A Dolgozo, Munkaora és HaviMunka táblákat úgy terveztük meg, hogy azok lefedjék a dolgozói beléptetés minden fontos aspektusát. A Dolgozo tábla tárolja a felhasználók alapadatait, a Munkaora a napi belépési és kilépési időket, a HaviMunka pedig az összesített havi teljesítményt. Létrehoztuk az első EF Core migrációt, majd MySQL-el teszteltük a kapcsolatokat. Az adatbázis szerkezete jól skálázható és bővíthető maradt. Beállítottuk a kapcsolatok típusait, és ellenőriztük a külső kulcsok helyes működését.

### **8. hét**

Kialakítottuk a három fő adatmodellt: Dolgozo, Munkaora, és HaviMunka. Ezeket regisztráltam az AppDbContext-ben, majd létrehoztam az első migrációt. Az SQLite adatbázisba feltöltöttünk néhány tesztadatot, és teszteltük az első lekérdezéseket. Az adatkapcsolatok jól működnek. Emellett készítettünk egy adatbázis diagramot is a jobb áttekinthetőség érdekében.

### **9. hét**

Az API szerkezetét tovább finomítottuk a közben felmerült újabb igényekhez igazodva,: bevezettük a DTO-kat, így a modellek nem kerülnek közvetlenül a kliensekhez. Átdolgoztuk a RegisterRequest és LoginRequest adatstruktúrákat. Swagger dokumentációban beállítottuk az endpointokat és azok bemeneti adatait. Teszteltük az admin jogosultságokat, és kiépítettük a megfelelő hibakezelési logikát. Ezután minden hiba HTTP státusszal és egyértelmű üzenettel tér vissza. A rendszer megfelel a RESTful API irányelveinek.

### **10. hét**

A hitelesítés logikáját alakítottuk ki, amely során JWT token alapú autentikációt vezettük be.  
 A Login végpont ellenőrzi a felhasználónevet és a jelszót, és ha helyesek, tokennel tér vissza. A jelszavakat BCrypt-tel titkosítva tároljuk az adatbázisban. A rendszer külön kezeli az admin és a dolgozói jogosultságokat. Teszteljuk a bejelentkezési folyamatot különféle szerepkörökkel, és működik a szerepkör alapú hozzáférés is. A tokeneket egyórás lejárattal generáljuik, így növelve a biztonságot.

### **11. hét**

Implementáltuk a dolgozói bejelentkezés és kijelentkezés funkciókat az AttendanceController segítségével. A dolgozók mostantól egy gombnyomással indíthatják vagy zárhatják le a műszakjukat. A rendszer ellenőrzi, hogy van-e nyitott műszak, mielőtt új belépést rögzít. A kilépés ideje alapján kiszámolható a ledolgozott idő, amely bekerül a munkaidő naplóba. A beléptető API-kat teszteltük Swaggerben és Postmanben is. A válaszidők gyorsak, és a rendszer hibatűrő lett.

### **12. hét**

Kialakítottuk a havi munkaidő-összesítő logikát a HaviMunka entitás segítségével. Az API automatikusan összegzi a dolgozók napokra bontott munkaidejét, és elmenti az adatokat. A rendszer most már képes teljes hónapokat elemezni és riportálni admin felületre. A backend logikát optimalizáltam, hogy ne legyen szükség manuális futtatásra – a riport frissítése automatikus. Ehhez LINQ lekérdezéseket és egy kis időzítési logikát is használtam. A riportok JSON-ben jelennek meg, de később exportálhatók lesznek Excelbe i

### **13. hét**

Az adminfelület továbbfejlesztésére fókuszáltunk. Mostantól lehetőség van dolgozók törlésére és adatainak frissítésére is. Kialakítottunk egy egyszerű keresőmezőt, amellyel könnyen szűrhetők a dolgozók. Emellett részletesebb visszajelzések kerültek az API válaszaiba, például sikeres törlés vagy sikertelen módosítás esetén. Swagger-ben aktiváltuk az Authorize gombot, így már ott is tesztelhetőek a jogosultsághoz kötött végpontok. Kisebb UI-javításokat is végeztem, főleg a táblázatok elrendezésén.

### **14. hét**

Elkezdtem a mobilalkalmazás fejlesztését .NET MAUI keretrendszerrel. A projekt célja, hogy a dolgozók mobilon is egyszerűen kezelhessék a munkaidő-nyilvántartást. Az első lépés a környezet beállítása volt, majd létrehoztam az első képernyőket. Beállítottam az API kapcsolódást Android platformon keresztül. Teszteltem az első adatlekérdezéseket, amelyek a dolgozói token érvényességét ellenőrzik. A mobilalkalmazás jelenleg fejlesztői módban működik, de a legfontosabb alappillérek már elérhetők. Az UI-t XAML-ben építettem, és figyeltem a responsivitásra. Sikerült összekötnöm egy androidos okostelefonnal is a Visual Stuido-t így ténylegesen valós időben láthattuk hogy hogyan működik az App. Sikerült a telefont fejlesztői módba kapcsolni valamint engedéjeztük rajta a hibakeresést, így tudodott hozzá csatlakozni a Visual Studio. Az App valós időben tesztelhető volt így de lecsatlakoztatás után is működött már az App, mivel mindern egyes ujra indításkor lényegében felfrissítette azt a Visual Studio.

### **15. hét**

Befejeztem a .NET MAUI alkalmazás fejlesztését. Elkészült a felhasználói felület, beleértve a bejelentkezési oldalt, valamint a „Munka kezdése” és „Munka befejezése” gombokat. Ezek a gombok API-hívásokat indítanak, amelyek naplózzák a belépési és kilépési időket. Az alkalmazás már token alapú autentikációt használ, így a dolgozók csak érvényes bejelentkezés után használhatják a funkciókat. Androidos eszközön a tesztelés zökkenőmentesen zajlott, a válaszidők megfelelőek. A hibakezelést is beépítettem az appba, például ha nincs internetkapcsolat vagy rossz token van megadva. Emellett kialakítottam a kapcsolatot az adatbázis felé REST-en keresztül.

### **16. hét**

A projekt záró hetében véglegesítettük a teljes rendszert. Az adminfelületen keresztül már stabilan kezelhetők a dolgozók, valamint az adatok megjelenítése is hibamentes. Teszteltük az alkalmazást többféle eszközön, beleértve mobiltelefonokat is. Az MAUI alkalmazás is sikeresen csatlakozik az ASP.NET Core API-hoz. A tokenes hitelesítés megfelelően működik mobilon is, így a biztonság garantált. Elkészült a részletes felhasználói dokumentáció a rendszer telepítéséről és működtetéséről. A dokumentációban szerepelnek képernyőképek, hibaelhárítási javaslatok, valamint lépésről lépésre leírt útmutatók. Az admin jogosultságok kezelése szintén átesett egy kis átszervezésen.. A végleges verziót elmentettük GitHubra.