*„Szoftverfejlesztő és tesztelő technikus szakma"  
Komplex programozói munka*

*Zárófeladat leírás*

*Kun György*

*2/14.EC*

*2022/23 tanév*



*Debreceni SZC*

*Mechwart András Gépipari és Informatikai Technikum*

Tartalomjegyzék

[1. Bevezetés 6](#_Toc134469267)

[A. A raktárkezelő megoldás célja és funkciói 6](#_Toc134469268)

[B. A programozási nyelvek és keretrendszerek választása 6](#_Toc134469269)

[a) Visual Studio Code 6](#_Toc134469270)

[i. Előnyök 7](#_Toc134469271)

[ii. Hátrányok 7](#_Toc134469272)

[b) Django webes keretrendszer 8](#_Toc134469273)

[iii. Előnyök 8](#_Toc134469274)

[iv. Hátrányok 8](#_Toc134469275)

[c) React 9](#_Toc134469276)

[v. Előnyök 9](#_Toc134469277)

[vi. Hátrányok 9](#_Toc134469278)

[d) Git és Github 10](#_Toc134469279)

[vii. Előnyök 10](#_Toc134469280)

[viii. Hátrányok 10](#_Toc134469281)

[C. A dokumentum célja és felépítése 10](#_Toc134469282)

[2. Követelmények 11](#_Toc134469283)

[D. A rendszerrel szemben támasztott követelmények 11](#_Toc134469284)

[E. A felhasználói igények elemzése 11](#_Toc134469285)

[F. Funkcionális és nem-funkcionális követelmények 11](#_Toc134469286)

[e) Funkcionális követelmények: 11](#_Toc134469287)

[f) Nem funkcionális követelmények: 12](#_Toc134469288)

[3. Rendszerterv 12](#_Toc134469289)

[G. A rendszer architektúrája és komponensei 12](#_Toc134469290)

[g) Rendszer Architektúra: 12](#_Toc134469291)

[ix. Frontend réteg 12](#_Toc134469292)

[x. Backend réteg 12](#_Toc134469293)

[xi. Adatbázis réteg 12](#_Toc134469294)

[xii. API réteg: 12](#_Toc134469295)

[xiii. Egyéb komponensek 13](#_Toc134469296)

[H. Rendszerkomponensek 13](#_Toc134469297)

[h) Adatbázis réteg 13](#_Toc134469298)

[i) Backend réteg 13](#_Toc134469299)

[j) API réteg 13](#_Toc134469300)

[k) Frontend réteg 13](#_Toc134469301)

[I. Az adatbázis és adatmodell tervezése 13](#_Toc134469302)

[J. Az API protokoll tervezése és implementációja 14](#_Toc134469303)

[4. Implementáció 15](#_Toc134469304)

[K. A backend és frontend fejlesztése 15](#_Toc134469305)

[l) A backend fejlesztése: 15](#_Toc134469306)

[xiv. 1. Előkészületek: 15](#_Toc134469307)

[xv. 2. Adatmodell tervezése: 15](#_Toc134469308)

[xvi. 3. Nézetek (views) és URL-ek: 15](#_Toc134469309)

[xvii. 4. Serializerek: 15](#_Toc134469310)

[xviii. 5. API végpontok: 15](#_Toc134469311)

[m) A frontend fejlesztése: 16](#_Toc134469312)

[xix. 1. Projekt beállítása: 16](#_Toc134469313)

[xx. 2. Komponensek kialakítása: 16](#_Toc134469314)

[xxi. 3. API kérések: 16](#_Toc134469315)

[L. Az adatbázis és adatmodell implementációja 16](#_Toc134469316)

[n) Adatbázis tervezése: 16](#_Toc134469317)

[o) Adatmodellek létrehozása: 16](#_Toc134469318)

[p) Adatbázis migrációk: 16](#_Toc134469319)

[q) Adatbázis inicializálása: 16](#_Toc134469320)

[r) Adatbázis műveletek: 16](#_Toc134469321)

[M. Az API protokoll implementációja és tesztelése 17](#_Toc134469322)

[s) Az API végpontok kialakítása: 17](#_Toc134469323)

[t) Az API válaszok formázása: 17](#_Toc134469324)

[u) Az API tesztelése: 17](#_Toc134469325)

[5. Tesztelés 17](#_Toc134469326)

[N. Az egységtesztek és integrációs tesztek tervezése és végrehajtása 17](#_Toc134469327)

[v) Az egységtesztek tervezése: 17](#_Toc134469328)

[w) Az egységtesztek végrehajtása: 17](#_Toc134469329)

[x) Az integrációs tesztek tervezése: 17](#_Toc134469330)

[y) Az integrációs tesztek végrehajtása: 18](#_Toc134469331)

[O. A rendszer teljesítménytesztelése 18](#_Toc134469332)

[P. Hiba és hibajavítási folyamat 18](#_Toc134469333)

[z) Hibák azonosítása: 18](#_Toc134469334)

[aa) Hibák javítása: 18](#_Toc134469335)

[6. Felhasználói dokumentáció 18](#_Toc134469336)

[Q. Az alkalmazás használatának bemutatása 18](#_Toc134469337)

[bb) 1. Django adminisztrációs felület: 19](#_Toc134469338)

[cc) 2. React felhasználói felület: 19](#_Toc134469339)

[R. A funkciók leírása 19](#_Toc134469340)

[S. A felhasználói felület használatának bemutatása 19](#_Toc134469341)

[7. Összegzés 19](#_Toc134469342)

[T. Az elvégzett munka értékelése 19](#_Toc134469343)

[U. Az elért eredmények összefoglalása 19](#_Toc134469344)

[V. Továbbfejlesztési lehetőségek és javaslatok 20](#_Toc134469345)

[8. Hivatkozások 20](#_Toc134469346)

[W. A felhasznált források és dokumentációk listája 20](#_Toc134469347)

[X. A használt programozási nyelvek és keretrendszerek hivatkozásai 20](#_Toc134469348)

# Bevezetés

Jelen anyag a "Szoftverfejlesztő és tesztelő technikus szakma", „Komplex programozói munka” kisérő dokumentuma, bemutatja az adott programozói munkát és annak környezetét.

A programozói munka célja egy, a lehető legegyszerűbb, a jelenleg rendelkezésre álló legmodernebb, de emellett a lehető leggazdaságosabban elérhető – optimális esetben díjmentesen igénybevehető - rendszerelemekből kialakítani egy alap, minimalista raktárnyilvántartó és lekérdező megoldást.

Természetesen mindennek a "Képzési és Kimeneti Követelmények", "Szoftverfejlesztő és tesztelő technikus szakma", 5-0613-12-03 dokumentum 18. oldal. 8.4.2. A pontjának megfelelően történik, mely szerint:

- életszerű, valódi problémára megoldást nyújtson.

- adattárolási és kezelési funkciók is

- RESTful architecturának megfelelő szerver/kliens komponensek

- kliens: desktop és mobil környezet is

(Asztalinál webes megoldás is szükséges)

## A raktárkezelő megoldás célja és funkciói

Az adott rendszer célja egy viszonylag kis raktárkészlettel rendelkező vállalkozás segítése az anyagok nyilvántartásában. Indító kérdésként csupán az alapvető nyilvántartási adatokat – azonosító, név, leírás, ár, kép – kezelő megoldás kialakítása körvonalazódott.

Elvárás volt, hogy az áruk/berendezések – továbbiakban: elemek – tényleges kezelése elsősorban a Backend részen történjen. A Frontend rész egy erősen korlátozott, jelenleg csak lekérdezésre igénybevehető megoldást nyújtson.

## A programozási nyelvek és keretrendszerek választása

Alapvetően a programozási nyelv(ek) és a keretrendszer(ek) kiválasztása a józan észen, és természetesen azon alapult, hogy a képzés folyamán milyen megoldások kerültek bemutatásra. Az semmiképpen nem lett volna kivitelezhető (sem időt, sem energiát tekintve), hogy olyan megoldásokat használjunk melyek teljesen eltérnek az általunk megismertektől, azaz ennek megfelelően teljesen az alapoktól kellett volna azok használatát megismerni.

Ennek megfelelően fejlesztő rendszerként a Microsoft Visual Studio Code került kiválasztásra. A keretrendszerek kapcsán pedig a Django és a React került a dobogó tetejére. A programozási nyelveket ez természetesen meghatározta: JavaScript és Python.

A döntés megalapozottságát az adott elemek pozitív és negatív tulajdonságaival kívánom alátámasztani:

### Visual Studio Code

Amikor a programozási nyelvek és keretrendszerek kiválasztásáról van szó, számos indok szól a Visual Studio Code (VS Code) használata mellett. Az VS Code egy népszerű és erőteljes fejlesztői környezet (IDE), amely számos előnyt kínál a programozók számára. Itt van néhány indok a VS Code használata mellett:

#### Előnyök

Lightweight (könnyűsúlyú) és gyors: Az VS Code egy könnyűsúlyú IDE, ami gyorsan indul és kevés erőforrást használ. Ez lehetővé teszi a hatékony munkavégzést, különösen olyan helyzetekben, ahol a rendelkezésre álló erőforrások korlátozottak vagy amikor gyorsan szeretnél dolgozni egy reakcióképes fejlesztői környezetben. Az iskolai környezetben ez kiemelten jelentkezett, mivel az ottani számítógép hardver háttere egy-egy komolyabb IDE elindításakor nem percekben, hanem ötpercekben mérhető elindulási időt produkált.

Kiterjeszthetőség: Az VS Code rendelkezik egy gazdag kiterjesztési ökoszisztémával, amely lehetővé teszi a fejlesztők számára, hogy testreszabják az IDE-t, azaz hozzáadjanak különböző funkciókat és eszközöket a munkafolyamathoz. Az elérhető kiterjesztések, bővítmények segítségével könnyen integrálhatóak további programozási nyelvek támogatása, plusz fejlesztői eszközök és további funkciók. Nem utolsó sorban a könnyű testreszabhatóság is komoly érv mellette.

Az VS Code támogat számos programozási nyelvet és keretrendszert. Ilyen például a jelenleg is kiemelten fontos JavaScript és Python. Emellett a Java és C# is a sorba tartozik sok más mellett. Ez lehetővé teszi, hogy egyetlen IDE-ben dolgozhassunk különböző projekteken, anélkül hogy át kellene váltanunk különböző fejlesztői környezetek között – magyarán könnyebben kezelhető, mivel a menürendszer, a billentyűzet kombináció és a környezet is azonos marad.

Az VS Code számos kódszerkesztési és hibakeresési funkcióval rendelkezik. Ilyenek például az automatikus kódkiegészítés, szintaktikai kiemelés, hibakeresési pontok beállítása és integrált hibakereső eszközök. Ezek a funkciók segítenek a hatékonyabb kód írásban, a könnyebb hibakeresésben.

Az VS Code elérhető Windows, macOS és Linux rendszereken is. Ez lehetővé teszi a fejlesztők számára, hogy azonos környezetet használjanak a különböző operációs rendszereknél.

#### Hátrányok

Természetesen a „másik oldalt” is meg kell vizsgálnunk, azaz milyen indokok szolhatnak a VS Code használata ellen?

Mivel az VS Code egy könnyűsúlyú fejlesztői környezet (IDE), néhány haladó vagy specifikus funkció hiányozhat belőle, amelyeket más nagyobb és szakosodottabb IDE-k könnyebben biztosítanak. Például, ha egy specifikus nyelvet vagy keretrendszert használunk, amelyhez szükségesek olyan funkciók vagy eszközök, amelyek nem állnak rendelkezésre az alapértelmezett VS Code-ban, akkor más IDE-t is érdemes lehet fontolóra venni. (Jelen

Az VS Code nem rendelkezik teljes projektmenedzsment funkciókkal, mint például az integrált verziókezelés, automatikus építés és tesztelés, valamint a folyamatkezelés és a munkatársmegosztás. Ha olyan projekteken dolgozunk, amelyek nagyobb méretűek vagy összetettebbek szükség lehet teljes projektmenedzsmentre. Ebben az esetben más IDE vagy fejlesztői környezet lehet a jobb választás.

Bár az VS Code rendkívül népszerű és széles körben elterjedt, más IDE-k és fejlesztői környezetek mögött gyakran nagyobb és aktívabb támogató közösség áll. Ez azt jelenti, hogy a kevésbé elterjedt vagy specifikus problémákra talán nehezebb megtalálni a megoldásokat vagy segítséget.

Noha a VS Code nagyon rugalmas és kiterjeszthető, vannak olyan esetek, amikor az IDE nem tud teljes mértékben alkalmazkodni egyes projektek vagy környezetek specifikus igényeihez. Ez lehet például olyan esetekben, amikor speciális debuggolási funkciók, hardvereszközök vagy integrációk szükségesek.

### Django webes keretrendszer

A Django egy nyílt forráskódú webes keretrendszer, amelyet a Python programozási nyelvhez készítettek.

#### Előnyök

Számos beépített eszközzel rendelkezik, amelyek lehetővé teszik a fejlesztők számára, hogy gyorsan és hatékonyan dolgozzanak. A keretrendszer beépített admin felülete, form validáció és URL routing funkciói jelentősen csökkentik a fejlesztési időt.

Beépített biztonsági funkciói közé tartozik a cross-site scripting (XSS), cross-site request forgery (CSRF) és SQL injection támadások elleni védelem. A keretrendszernek a biztonságra való fókuszálása lehetővé teszi a fejlesztők számára, hogy biztonságos és megbízható webalkalmazásokat hozzanak létre.

Olyan funkciókat biztosít, mint például az automatikus input validáció, az adatvédelmi rétegek és a biztonságos cookie-kezelés, hogy növelje az alkalmazás biztonságát.

Lehetővé teszi a fejlesztők számára, hogy nagyobb, összetett alkalmazásokat hozzanak létre. A keretrendszer moduláris felépítése lehetővé teszi a fejlesztők számára, hogy könnyen bővítsék az alkalmazást és skálázzák azt a növekvő igényekhez. Lehetővé teszi a fejlesztőknek, hogy könnyen kezeljék és karbantartsák a kódot, valamint a rendszer különböző részeinek függetlenül történő fejlesztését és tesztelését.

Hatalmas internetes közösséggel rendelkezik, amely tartalmazza a fejlesztőket, felhasználókat, dokumentációs szakértőket és a keretrendszer továbbfejlesztésére szakosodott szervezeteket. A közösség folyamatosan fejleszti a keretrendszert, frissíti és javítja a dokumentációt, így a fejlesztők mindig a legújabb információkat és eszközöket használhatják.

Nyílt forráskódú, ingyenesen elérhető keretrendszer, így nem kell licencdíjat fizetni érte. A nyílt forráskódú szoftverek előnye, hogy lehetőséget adnak a fejlesztőknek, hogy a saját igényeikre szabják az alkalmazást, és segítsenek a fejlesztésben.

Beépített ORM (Objektum-relációs leképezés) réteggel rendelkezik, amely lehetővé teszi az egyszerű és hatékony adatbázis-kezelést. A Django ORM segítségével könnyedén definiálható adatmodell, létrehozható és frissíthető az adatbázis, és könnyedén lekérdezhetőek az adatok. Ez a funkció nagyon hasznos és időtakarékos, különösen a webalkalmazások fejlesztése során.

Tartalmaz egy beépített adminisztrációs felületet, amely lehetővé teszi az egyszerű és gyors adminisztráció különböző tartalmakhoz és adatokhoz. Az adminisztrációs felület automatikusan generálja az adatmodellekhez kapcsolódó felületeket – adatok kezelése: létrehozás, frissítés vagy törlés az adatbázis rekordokban.

#### Hátrányok

Ha kisebb vagy egyszerűbb webalkalmazást vagy webhelyet kívánunk létrehozni, akkor a Django túl nagy és bonyolultnak tekinthető. A Django erőteljes és moduláris, de ez némi komplexitással jár, amely felesleges lehet az egyszerű projektekben.

Magasabb tanulási görbével rendelkező keretrendszer lehet, különösen, ha kevés tapasztalattal rendelkezünk a Pythonban vagy a webfejlesztésben. A Django használatához meg kell ismernünk a keretrendszer alapelveit, mint például az MVC (Model-View-Controller) tervezési minta és az ORM (Objektum-relációs leképezés) használata. (Ezt saját magam is közvetlenül tapasztaltam.)

Bár a sok előre definiált funkcióval és eszközzel rendelkezik, amelyek megkönnyítik a fejlesztést, ezek néha korlátozhatják a rugalmasságot. Ha egyedi és nagyon specifikus igényekkel rendelkező projektet fejlesztünk, akkor egy alacsonyabb szintű keretrendszer vagy egyedi fejlesztés lehet jobb választás.

Mivel egy magasabb szintű keretrendszerről van szó, ezért némi teljesítményveszteséggel járhat a kisebb és gyorsabb megoldásokhoz képest. Ha a projekt célja a nagy forgalmú vagy nagyon gyors webalkalmazás, akkor más keretrendszerek vagy technológiák hatékonyabbak lehetnek.

### React

A React egy népszerű JavaScript könyvtár és keretrendszer, amelyet a felhasználói felületek (UI) fejlesztésére használhatunk.

#### Előnyök

Komponensekre épül, amelyek önálló, újra felhasználható és könnyen karbantartható egységeket alkotnak. Ez lehetővé teszi a fejlesztők számára, hogy a komponenseket függetlenül fejlesszék és teszteljék, majd összeállítsák őket egy nagyobb alkalmazássá. Ez a megközelítés segíti a kód rendezett és könnyen skálázható maradását.

A megoldás egy virtuális DOM-ot (Virtual Document Object Model) használ, amely egy absztrakt reprezentációja a valós DOM-nak. A virtuális DOM hatékonyan kezeli az alkalmazás állapotának változásait és optimalizálja a DOM manipulációt. Ez javítja az alkalmazás teljesítményét és reaktívabb felhasználói élményt nyújt.

Lehetővé teszi az adatkezelés következetességét és egyszerűségét a komponensek állapotán keresztül. A komponensek állapotának frissítése és az ezekből eredő felhasználói felületi változások automatikusan történnek, ami megkönnyíti az alkalmazás reaktivitásának fenntartását.

Rendelkezik egyéb hasznos eszközökkel és könyvtárakkal, mint például a Redux és a React Router, amelyek segítenek az alkalmazás állapotkezelésében és a navigáció kezelésében. Ez a gyors és hatékony fejlesztést teszi lehetővé, valamint megkönnyíti az alkalmazás funkcióinak bővítését.

Rendkívül népszerű és aktív közösséggel rendelkezik. Ez azt jelenti, hogy rengeteg dokumentáció, tutorial, könyvtár és hasznos erőforrás áll rendelkezésre a fejlesztők számára. A közösség folyamatosan fejleszti és javítja a Reactet, így a frissítések és az új funkciók rendszeresen elérhetők.

#### Hátrányok

A React egy könyvtár, amely csak a felhasználói felület (UI) réteget kezeli. Ha egy teljeskörű webalkalmazást fejlesztünk, akkor más könyvtárakat vagy keretrendszereket is szükséges lehet használni, mint például a Redux az állapotkezeléshez vagy a React Router a navigációhoz. Ez több tanulási görbével és komplexitással járhat.

Ha korábban nem használtunk Reactet vagy nincs tapasztalatunk a JavaScript-ben, akkor tanulási időre lesz szükséged a keretrendszer megértéséhez és hatékony használatához. A React követ egyedi megközelítéseket és tervezési mintákat, mint például a JSX és a komponens alapú fejlesztés, amelyeket meg kell érteni.

Ha a fejlesztési csapatnak nincs tapasztalata a Reacttel vagy a JavaScripttel, akkor a tanulási idő és az átképzés költségei befolyásolhatják a választást. Fontos megfontolni a rendelkezésre álló erőforrásokat és a fejlesztői készségeket a keretrendszer kiválasztásakor.

A rendszer hatékonyan kezeli a felhasználói felület változásait a virtuális DOM segítségével. Azonban nagyobb méretű és bonyolultabb alkalmazások esetén a React teljesítménye csökkenhet, és a DOM manipuláció költségessé válhat. Ezért más, könnyebb keretrendszerek lehetnek hatékonyabbak ilyen esetekben.

A React a felhasználói felületre koncentrál, és nem nyújt beépített megoldásokat más típusú alkalmazásokhoz, mint például a háttérrendszer vagy a szerveroldali fejlesztés. Ha olyan alkalmazást fejlesztesz, amelyhez szükség van ezekre a funkciókra, akkor más keretrendszerek vagy nyelvek lehetnek alkalmasabbak.

### Git és Github

A Git és a GitHub a verziókezelés és a forráskód megosztására szolgáló, az egyik legismertebb megoldás.

#### Előnyök

A Git lehetővé teszi a fejlesztők számára a forráskód változásainak nyomon követését és kezelését. Ez lehetővé teszi a könnyű visszatérést korábbi verziókhoz, a változtatások összehasonlítását és a könnyű ágkezelést a párhuzamos fejlesztéshez.

Lehetővé teszi a csapatmunkát és az együttműködést a fejlesztők között. A közös repositoryban történő munka során könnyen összeolvashatók a különböző ágak, megoldhatók az ütközések, és könnyen visszakereshetők a változtatások.

Segítségükkel nyomon követhető a fejlesztési folyamat és dokumentálhatók a változtatások. A commit üzenetek és az issue tracking rendszer segítségével dokumentálhatók a változtatások okai és a problémák megoldása.

Ezekkel könnyen létrehozhatók biztonsági mentések a forráskódról. A lokális Git repositoryk lehetővé teszik a helyi mentések létrehozását, míg a GitHub a távoli tárolás biztonságát nyújtja, így ha a helyi példányok elvesznek vagy megsérülnek, a forráskódot vissza lehet állítani a távoli tárolóból.

A platformja lehetővé teszi a forráskód nyilvános vagy privát megosztását. Ez lehetővé teszi a projektek nyilvánossá tételét vagy más fejlesztők meghívását a közös munkához. Emellett a GitHub széles körű közösségi funkciókat kínál, amelyek segítségével más fejlesztők hozzájárulhatnak a projekthez, javíthatják és adhatnak visszajelzést.

#### Hátrányok

Ha egy egyszerűbb vagy kisebb projektet fejlesztünk, amelynek nincs sok fejlesztője, és nincs szükség a verziókezelés és a kollaboráció ilyen szintű bonyolultságára, akkor a Git és a GitHub használata túlzott lehet. Ebben az esetben egyszerűbb megoldások, például helyi verziókezelő rendszerek vagy más tárolók is elegendőek lehetnek.

Ha a projektnek szigorú biztonsági előírásoknak kell megfelelnie, vagy adatvédelmi szempontból különleges követelmények vannak, akkor a nyilvános tárolók használata, mint például a GitHub, problémákat okozhat. Ebben az esetben más, privát tárolók vagy önálló verziókezelő rendszerek lehetnek jobb választások.

Ha a GitHub privát repository-t szeretnénk használni, akkor bizonyos költségek merülhetnek fel, különösen ha nagyobb csapat vagy vállalat számára történik a fejlesztés. Ha a költségek jelentenek problémát, akkor más ingyenes vagy olcsóbb verziókezelő megoldások használatát érdemes fontolóra venni.

Noha a Git és a GitHub népszerűek, nem mindenki számára jelentik az ideális verziókezelő és tárolási megoldást. Egyes fejlesztők előnyben részesítik más verziókezelő rendszerek vagy tárolók használatát, amelyek jobban illeszkednek a fejlesztési folyamatukhoz vagy személyes preferenciáikhoz.

## A dokumentum célja és felépítése

Ezen dokumentummal törekszünk az adott lehetőségekkel élve bemutatni a szoftverfejlesztéshez használt elemeket, a fejlesztés lépéseit, annak különféle problémakörét és az opciókat.

Nem csak speciálisan az adott megoldásra, hanem általánosabb nézetben is bemutatásra kerülnek a megfelelő részek. Ezáltal szélesebb képet mutatva a szoftverfejlesztés adott szegmenséről.

# Követelmények

## A rendszerrel szemben támasztott követelmények

1. Adminisztrátori hozzáférés kezelése. Lehetővé kell tenni az adminisztrátor számára, hogy a Backend részhez további felhasználókat regisztráljon. Frontend oldali felhasználó kezelés nem tartozott a követelmények közzé. Jövőbeli fejlesztés esetén legyen lehetséges a felhasználók számára, hogy regisztráljanak egy fiókot az alkalmazásba, és bejelentkezzenek a későbbi használat során.
2. Csoportok kezelése: A rendszernek lehetővé kell tennie az adminisztrátor számára, hogy létrehozzanak, szerkesszenek és töröljenek áru csoportokat. Minden csoporthoz tartoznia kell egy névnek és leírásnak.
3. Termékek kezelése: Az adminsztrátoroknak legyen lehetőségük termékek hozzáadására, szerkesztésére és törlésére. A termékeknek tartalmazniuk kell egyedi azonosítót, nevet, leírást és képet.
4. Felhasználói élmény: Az alkalmazásnak intuitív és felhasználóbarát felhasználói felületet kell biztosítania, amely egyszerűen kezelhető és könnyen érthető a felhasználók számára.
5. Skálázhatóság és rugalmasság: Az alkalmazásnak skálázhatónak és rugalmasnak kell lennie, hogy könnyen alkalmazkodhasson a későbbi, esetleges valós környezetben történő működésre.

## A felhasználói igények elemzése

Az előbbi felhasználó igények a kiválasztott rendszerkörnyezet és fejlesztői megoldások segítségével kielégíthetőek. Mivel a rendszer rugalmasan továbbfejleszthető a nélkül, hogy jelentős strukturális módosítást kellene végrehajtani ezért az is kijelenthető, hogy reálisan vállalható utóműveletekkel valós mikróvállalati környezetre is implementálható.

## Funkcionális és nem-funkcionális követelmények

A korábbi felsorolásnak megfelelően a következő csoportosítás is megvalósítható a követelmények kapcsán. Itt a jövőbeli – nem ezen projekt keretében megvalósítandó – követelményeket is feltüntetjük a teljesség érdekében:

### Funkcionális követelmények:

1. Felhasználói regisztráció és bejelentkezés lehetősége.
   1. Jelen projektben csupán az adminisztrátori hozzáférés kezelése van fókuszban
2. Raktárak kezelése (létrehozás, szerkesztés, törlés).
   1. Jelen projektben csupán az adminisztrátor általi kezelés van fókuszban
3. Termékek kezelése (hozzáadás, szerkesztés, törlés).
4. Készletnyilvántartás (termékmozgások rögzítése, készletszint követése).
5. Keresés és szűrés lehetősége termékek és raktárak között.
6. Különböző riportok és kimutatások generálása (pl. készletszint, termékmozgások stb.).
7. Árukészlet bevételezése és kimenetele az alkalmazásból.
8. Raktárkészletek megtekintése és frissítése.

### Nem funkcionális követelmények:

1. Felhasználóbarát és intuitív felhasználói felület.

2. Biztonságos hozzáférés és adatvédelem.

3. Megfelelő teljesítmény és gyors válaszidő.

4. Skálázhatóság és rugalmasság a későbbi bővítésekhez és növekedéshez.

5. Kompatibilitás különböző böngészőkkel és eszközökkel.

6. Könnyű karbantarthatóság és dokumentáció a fejlesztés és támogatás során.

Ezek a funkcionális és nem funkcionális követelmények segítenek meghatározni az alkalmazás alapvető funkcionalitását és elvárásait, valamint az általános minőségi és működési követelményeket a sikeres záróprojekt elkészítése és a későbbi mikrovállalkozások számára való utóhasznosítás céljából.

# Rendszerterv

## A rendszer architektúrája és komponensei

A Django és a React keretrendszerek alkalmazása esetén a rendszer architektúráját és komponenseit általában a következőképpen lehet leírni:

### Rendszer Architektúra:

A rendszer architektúrája több rétegből és komponensből áll, amelyek együttműködnek a teljes rendszer működtetésében és a felhasználók interakcióinak kezelésében.

#### Frontend réteg

Ez a React keretrendszerrel készült felhasználói felület, amely felelős a felhasználói élményért és a megjelenítésért. Ez tartalmazza az oldalak, komponensek, űrlapok, navigációs elemek és más interaktív elemeket, amelyeket a felhasználóval való kommunikációhoz használnak.

#### Backend réteg

Ez a Django keretrendszerrel készült üzleti logika és adatkezelési réteg. Ez a rész felelős a kérések fogadásáért, az adatbázis műveletek végrehajtásáért, az adatok feldolgozásáért és a válaszok generálásáért. Itt történik az adatok feldolgozása, az autentikáció, az autorizáció, a műveletek validációja és az üzleti logika végrehajtása.

#### Adatbázis réteg

Ez a réteg tartalmazza az adatbázis vagy adattároló rendszert, amelyben az alkalmazás adatait tárolja. A Django általában relációs adatbázisokkal (például PostgreSQL, MySQL) vagy NoSQL adatbázisokkal (például MongoDB) integrálható.

#### API réteg:

Az API réteg lehetőséget biztosít a kommunikációra a frontend és a backend között. A Django REST Framework (DRF) használatával lehetőség van RESTful API-k létrehozására, amelyekkel a frontend kommunikál és adatokat kérhet vagy módosíthat.

#### Egyéb komponensek

További fejlesztési lehetőségként jelentkezhet: autentikációs és autorizációs szolgáltatásokat, külső API-k integrációja, további adatfeldolgozás vagy ütemezést végző háttérkomponenseket létrehozása más rendszerekkel.

## Rendszerkomponensek

A Django és React keretrendszerek együtt történő alkalmazásával az ún. "MERN stack" (MongoDB, Express, React, Node.js) használatára van lehetőség. A stack architektúra különböző rétegekből áll, amelyek lehetővé teszik a fejlesztők számára, hogy elkülönítsék az alkalmazás különböző részeit.

### Adatbázis réteg

… felelős az adatok tárolásáért és kezeléséért. A Django esetében az adatbázis réteg általában a Django ORM (Object Relational Mapping) és a hozzá kapcsolódó adatbázis kezelő rendszer, például PostgreSQL. A React alkalmazásokban általában nem használnak adatbázis réteget, hanem a backend réteg biztosítja az adatokat az API-n keresztül.

### Backend réteg

… felelős az adatok feldolgozásáért és az üzleti logika végrehajtásáért. A Django alkalmazások esetében a backend réteg általában a Django framework, ami Python nyelven íródott, míg a React alkalmazásokban az Express.js backend framework általában Node.js-en keresztül fut.

### API réteg

… biztosítja az alkalmazások közötti kommunikációt. Az API-t gyakran RESTful API-ként (Representational State Transfer) valósítják meg, amelyek lehetővé teszik az adatok küldését és fogadását az alkalmazások között. A Django alkalmazások esetében az API réteg általában a Django Rest Framework (DRF), míg a React alkalmazásokban az API réteg az Express.js keretrendszeren keresztül valósul meg.

### Frontend réteg

…felelős a felhasználói felület (UI) megjelenítéséért és a felhasználói interakciók kezeléséért. A React keretrendszer egy JavaScript alapú keretrendszer, amely lehetővé teszi a fejlesztők számára, hogy könnyedén építsenek felhasználói felületet, amelynek állapota dinamikusan változhat. A frontend réteg összetevői React komponensek, amelyek különböző módon kapcsolódnak az API-hoz, és kezelik az adatok megjelenítését és frissítését a felhas

## Az adatbázis és adatmodell tervezése

Jelen esetben, mivel meglehetősen egyszerű adatszerkezetről van szó, komoly tervezést nem kívánt a rendszer.

Az adatmodell tervezésekor a models.py fájlban definiáltak az adatbázisbeli táblák megfelelői. Az állományban három különböző modell van definiálva: Part, Device, Product és ProductGroup.

Ez a gyakorlatban egy eszköz részegységét, magát az eszközt, egy terméket és a termékcsoportot jelenti.

## Az API protokoll tervezése és implementációja

Az API (Application Programming Interface) protokoll tervezése és implementációja a Django és React környezetben egy egységes interfészt biztosít a Django backend és a React frontend között. Az API protokoll meghatározza, hogy hogyan történik az adatcserélés és a kommunikáció a két rendszer között.

A Django és React közötti API protokoll tervezése és implementációja magában foglalja a következő fontos lépéseket:

* 1. Adatmodell definiálása
     1. Az adatok meghatározása ezen dokumentáció más részén leírásra került. Röviden:
        1. Eszközök adatai
        2. Termékek adatai
        3. Konkrét termék adatai
        4. Csoportok adatai
        5. Konkrét csoportelem

Általánosan:

* + - * 1. Az API protokoll tervezésekor először meg kell határozni az adatmodellt, azaz az adatok struktúráját és típusait, amelyeket az API-n keresztül lehet lekérdezni vagy módosítani. Ez magában foglalja a Django backendben a modellek, mezők és kapcsolatok definiálását.
  1. Végpontok (endpoints) meghatározása
     1. 'devices/'  
        'products/'  
        'products/<int:ean>/'  
        'groups/'  
        'groups/<int:id>/'
     2. Általánosan:
        + 1. Az API protokoll tervezése során meghatározásra kerülnek az ún. végpontok (endpoints), amelyek az URL-eket jelentik, amelyekhez a React frontend hozzáférhet az adatok kezeléséhez. Ezek az URL-ek meghatározzák, hogy melyik funkcióhoz vagy adathoz tartozó kérések érkezhetnek be a szerverre, és hogy milyen műveletek hajthatók végre (pl. GET, POST, PUT, DELETE).
  2. HTTP metódusok kezelése
     1. Jelen esetben kizárólag a GET metódusokat használjuk
        1. Általánosan:
           1. Az API protokoll tervezése és implementációja során figyelembe kell venni az HTTP metódusokat, amelyekkel a kérések és válaszok közlekednek a kliens és a szerver között. Például a GET metódus a lekérdezéseket jelenti, a POST metódus az adatok létrehozását, az UPDATE és DELETE metódusok pedig a módosítást és törlést végzik.
  3. Adatátviteli formátum kiválasztása
     1. Jelen esetben a JSON formátumu adatátviteli formátumot választottuk. Általánosan:
        1. Az adatokat a kliens és a szerver között általában valamilyen adatátviteli formátumban kell átvinni, például JSON (JavaScript Object Notation) vagy XML (eXtensible Markup Language) formátumban. Az API protokoll tervezése során el kell dönteni, hogy melyik adatátviteli formátumot használják az adatok küldésére és fogadására.
  4. Hitelesítés és jogosultságkezelés
     1. Jelen esetben kizárólag a Backenden történt alapvető jogosultságkezelés. A Frontend oldalon nem. Ennek egyrészt az az oka, hogy egy oktatási rendszerről van szó, másrészt, hogy egy zárt, saját hálózaton történik a kommunikáció és semmilyen érzékeny adatot nem mozgat.
        1. Az API protokoll tervezésekor figyelembe kell venni a hitelesítési és jogosultságkezelési mechanizmusokat, amelyek biztosítják az adatok védelmét és az autorizált hozzáférést. Ez magában foglalhatja például token alapú autentikációt, JWT (JSON Web Token

# Implementáció

## A backend és frontend fejlesztése

A raktárkezelő programunk implementálásához Django és React keretrendszereket használtunk. Az alábbiakban bemutatjuk a backend és frontend fejlesztését.

### A backend fejlesztése:

#### 1. Előkészületek:

* Telepítettük és beállítottuk a Django keretrendszereket a fejlesztői környezetben (pl. Visual Studio Code).
* Létrehoztuk a Django projekteket és az ahhoz tartozó alkalmazásokat.

#### 2. Adatmodell tervezése:

* Definiáltuk az adatmodelleket a raktárkezelő alkalmazás számára, beleértve az árukat és csoportokat.
* Használtuk a Django modell osztályokat és mezőket az adatok tárolásához és a kapcsolatok létrehozásához.

#### 3. Nézetek (views) és URL-ek:

* Létrehoztunk nézeteket a különböző funkciókhoz, például az áruk és csoportok kezeléséhez.
* Definiáltuk az URL-eket a nézetekhez, hogy hozzáférhessünk azokhoz a frontend oldalról.

#### 4. Serializerek:

* Létrehoztuk a serializereket az adatok átalakításához JSON formátumba.
* Ez lehetővé tette az adatok küldését és fogadását a frontend és backend között.

#### 5. API végpontok:

* Definiáltuk az API végpontokat, amelyek lehetővé tették az adatok kérését és módosítását a frontend részéről.
* A Django REST Framework-öt használtuk az API végpontok kezeléséhez és az adatok megfelelő formázásához.

### A frontend fejlesztése:

#### 1. Projekt beállítása:

* Létrehoztuk a React projekteket a fejlesztői környezetben (pl. Visual Studio Code).
* Telepítettük a szükséges függőségeket, például a React Router-t és az HTTP kliens könyvtárat.

#### 2. Komponensek kialakítása:

* Létrehoztuk a React komponenseket a felhasználói felület elemek megjelenítéséhez, például az áruk és csoportok listázásához.

#### 3. API kérések:

* Küldtünk kéréseket az API végpontokhoz, hogy lekérdezzük az adatokat a backend részéről.

## Az adatbázis és adatmodell implementációja

### Adatbázis tervezése:

* + Először definiáltuk az adatbázis sémáját, az áruk és csoportok struktúráját.
  + Az adatok tárolásához és kezeléséhez a Django ORM (Object-Relational Mapping) keretrendszerét használtuk.

### Adatmodellek létrehozása:

* + Létrehoztuk az áruk és csoportok modelleket a Django modell osztályok segítségével.
  + Definiáltuk a szükséges mezőket és kapcsolatokat az adatok tárolásához és egymáshoz való viszonyulásukhoz.

### Adatbázis migrációk:

* + Elvégeztük a Django migrációkat a modellek alapján.
  + Ez létrehozta és frissítette az adatbázist a megfelelő táblákkal és mezőkkel.

### Adatbázis inicializálása:

* + Készítettünk inicializáló scriptet vagy adminisztrációs felületet a kezdeti adatok betöltésére az áruk és csoportok tábláiba.
  + Ez lehetővé tette az adatok kezdeti feltöltését a raktárkezelő programhoz.

### Adatbázis műveletek:

* + Implementáltuk az adatbázishoz való hozzáférést és műveleteket a Django ORM segítségével.
  + Lehetővé tettük az áruk és csoportok lekérdezését, listázását és keresését a raktárkezelő programban.

Az adatbázis és adatmodell implementációja révén sikerült megalapozni a raktárkezelő programunk alapstruktúráját és adatkezelését. A további funkciók fejlesztésekor ezekre a modellre és adatbázisra támaszkodhattunk.

## Az API protokoll implementációja és tesztelése

### Az API végpontok kialakítása:

* + Implementáltuk a szükséges Django REST framework nézeteket és URL-mintákat az áruk és csoportok kezeléséhez.
  + Az API végpontokat úgy alakítottuk ki, hogy lehetővé tegyék az áruk és csoportok lekérdezését.

### Az API válaszok formázása:

* + Megfelelő formátumban alakítottuk ki az API válaszokat JSON formátumban.
  + Biztosítottuk, hogy az áruk és csoportok adatai könnyen értelmezhetőek és feldolgozhatóak legyenek a frontend oldalon.

### Az API tesztelése:

* + Alaposan teszteltük az API végpontokat, hogy megbizonyosodjunk a helyes működésükről.

Az API protokoll implementációja és tesztelése során sikerült biztosítani a szükséges funkcionalitást a raktárkezelő program számára. Az áruk és csoportok lekérdezése az API segítségével történik, és a megfelelő hitelesítési és hozzáférési korlátozásokat is beépítettük.

# Tesztelés

## Az egységtesztek és integrációs tesztek tervezése és végrehajtása

### Az egységtesztek tervezése:

* + A frontend és backend egységeinek elkülönített tesztelése érdekében terveztünk egységteszteket.
  + Definiáltuk a teszteseteket az áruk és csoportok lekérdezésére, valamint az adminisztrációs oldali kezelőfelület funkcióinak ellenőrzésére.

### Az egységtesztek végrehajtása:

* + Implementáltuk az egységteszteket a fejlesztői környezetben.
  + Futtattuk a teszteket, és ellenőriztük az egyes funkciók helyes működését.
  + Korrigáltuk az esetleges hibákat és hibakezelést biztosítottunk a rendszerben.

### Az integrációs tesztek tervezése:

* + Az alkalmazás komponenseinek összekapcsolását és együttműködését teszteltük integrációs tesztekkel.
  + Terveztünk teszteseteket, amelyek ellenőrzik a frontend és backend közötti adatátvitelt és kommunikációt.

### Az integrációs tesztek végrehajtása:

* + Implementáltuk az integrációs teszteket, és futtattuk őket a rendszerben.
  + Ellenőriztük a komponensek közötti adatátvitelt, a visszajelzéseket és a rendszer általános stabilitását.

Az egységtesztek és integrációs tesztek tervezése és végrehajtása révén alaposan teszteltük a raktárkezelő program funkcióit. Biztosítottuk, hogy az áruk és csoportok lekérdezése, valamint az adminisztrációs oldali felület helyesen működjön. A tesztek segítségével megbizonyosodtunk a rendszer stabilitásáról és megbízhatóságáról.

## A rendszer teljesítménytesztelése

Tényleges teljesítménytesztelésre az adott megoldás kapcsán nem volt mód, mivel ahhoz olyan mennyiségű adatbevitelre és felhasználói input biztosítására lett volna szükség, amely az adott környezetben nem életszerű.

Azonban az a tény, hogy az iskolai - korántsem a csúcskategóriát képviselő - számítógépes környezetben is könnyedén és gyorsan használható volt, bizakodóvá tesz. Vélelmezem, hogy nagyobb, de még reális terhelés esetén is megfelelően működőképes marad a megoldás.

## Hiba és hibajavítási folyamat

### Hibák azonosítása:

* + A tesztelés során azonosítottunk hibákat, amelyek befolyásolhatják a rendszer stabilitását vagy a funkcionalitást.
  + Rögzítettük a hibákat és dokumentáltuk azokat, beleértve a hiba leírását, az okokat és a reprodukálási lépéseket.

### Hibák javítása:

* + Megkezdtük a hibák javítását a frontend és backend kódokban.
  + Elvégeztük a szükséges módosításokat, optimalizációkat és hibajavításokat.
  + Futtattuk újra a teszteket, hogy megbizonyosodjunk a hibák sikeres javításáról.

A tesztelés során azonosított hibákat gyorsan és hatékonyan javítottuk, hogy biztosítsuk a rendszer stabil működését. A dokumentált hibákat felhasználtuk a kód optimalizálásához és a fejlesztési folyamat további javításához. A tesztelés és a hibajavítások révén a raktárkezelő program adminisztrációs oldali áru és csoport kezelése, valamint az áruk és csoportok lekérdezése megbízhatóan működik.

# Felhasználói dokumentáció

## Az alkalmazás használatának bemutatása

Az éles webtárhelyre, illetve helyi szerverre történő telepítés jelen a dokumentáció kereteit meghaladja. A következő lépésekkel a fejlesztő környezetbe – Visual Studio Code – történő github forrás letöltése és beállítása történik. A rendszer a megfelelő funkcionalitással fog ebben az esetben is működni.v

### 1. Django adminisztrációs felület:

* + A Django adminisztrációs felületéhez való hozzáféréshez nyisd meg a böngészőt és látogass el a webalkalmazásod URL-címéhez.
  + Jelentkezz be az adminisztrációs felületre a megfelelő felhasználónévvel és jelszóval.
  + Az adminisztrációs felületen látni fogod a raktárkezeléshez kapcsolódó modulokat, például az "Áruk" és "Csoportok" menüpontokat.
  + Kattints az "Áruk" menüpontra, hogy megtekinthesd az elérhető áruk listáját.
  + Kattints egy adott áru nevére, hogy részletes információkat kapj róla.
  + A "Csoportok" menüpont alatt pedig megtekintheted a rendelkezésre álló csoportok listáját.

### 2. React felhasználói felület:

* + Nyisd meg a webalkalmazást a böngészőben.
  + A React felhasználói felületen a főoldalon az áruk és csoportok lekérdezése automatikusan megtörténik.
  + A főoldalon látni fogod az elérhető áruk és csoportok listáját.
  + Navigálhatsz az áruk és csoportok között a megjelenített linkek segítségével.
  + Az áruk részletes adatait az árukhoz tartozó linkre kattintva tekintheted meg.

Kérjük, vedd figyelembe, hogy a frontend oldalon jelenleg nincs lehetőség az adatok módosítására. Az alkalmazás célja kizárólag az áruk és csoportok lekérdezése és megjelenítése a felhasználó számára. Az adminisztrációs felületen keresztül történik az áruk és csoportok kezelése.

## A funkciók leírása

## A felhasználói felület használatának bemutatása

# Összegzés

## Az elvégzett munka értékelése

Az elvégzett munka során sikeresen létrehoztunk egy minimalista raktárkezelő programot, amely megfelel az előzetesen meghatározott követelményeknek. A programban kiemelt figyelmet fordítottunk a felhasználóbarát felület kialakítására, valamint a hatékony adatkezelésre.

A backend és frontend fejlesztése során összehangolt csapatmunka folyt.

A tesztelés során alaposan ellenőriztük a rendszer működését és azonosítottunk, majd javítottunk a hibákat. Ezáltal sikerült biztosítani a program stabilitását és megbízhatóságát.

## Az elért eredmények összefoglalása

Az elvégzett munka eredményeként egy jól működő és felhasználóbarát raktárkezelő programot hoztunk létre, amely megkönnyíti az adminisztrációs feladatokat az áruk és csoportok kezelése terén.

## Továbbfejlesztési lehetőségek és javaslatok

A jövőbeli fejlesztési javaslatokat a következőkben részletezzük:

1. Vonalkód beolvasási lehetőség
2. Felhasználói regisztráció és bejelentkezés lehetősége.
3. Raktárak közvetlen felhasználói kezelése (létrehozás, szerkesztés, törlés).
4. Termékek közvetlen felhasználói kezelése (hozzáadás, szerkesztés, törlés).
5. Felhasználó általi készletnyilvántartás (termékmozgások rögzítése, készletszint követése).
6. Keresés és szűrés lehetősége termékek és raktárak között.
7. Különböző riportok és kimutatások generálása (pl. készletszint, termékmozgások stb.).
8. Árukészlet bevételezése és kiadása az alkalmazással.
9. Raktárkészletek megtekintése és frissítése.

# Hivatkozások

## A felhasznált források és dokumentációk listája

* + Django hivatalos dokumentáció: https://docs.djangoproject.com/
  + React hivatalos dokumentáció: https://reactjs.org/docs/
  + JSON hivatalos dokumentáció: https://www.json.org/json-en.html
  + API tervezés és REST alapok: https://restfulapi.net/
  + Visual Studio Code hivatalos dokumentáció: https://code.visualstudio.com/docs
  + JavaScript hivatalos dokumentáció: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
  + Bootstrap hivatalos dokumentáció: https://getbootstrap.com/docs/5.1/getting-started/introduction/
  + Git hivatalos dokumentáció: https://git-scm.com/doc
  + GitHub hivatalos dokumentáció: https://docs.github.com/
  + DB Browser for SQLite (DBlight) hivatalos dokumentáció: https://sqlitebrowser.org/

Ezen források és dokumentációk segítettek minket a fejlesztés során, és mélyebb megértést nyújtottak a Webalkalmazások, Frontend, Backend, Django, React, JSON, API, REST API, Visual Studio Code, JavaScript, Bootstrap, Git, GitHub, DBlight témákban.

## A használt programozási nyelvek és keretrendszerek hivatkozásai

Meglátásom szerint a leghatékonyabb, ha magán a fejlesztési szükségleteken és folyamat lépésein keresztül mutatjuk be a használt eszközöket és hivatkozásokat.

* Operációs rendszer
  + Az aktuális alap környezetet a Microsoft Windows 10 operációs rendszere biztosítja. Ez meghatározza a többi „alrész” típusát, mivel jellemzően nem forrásállományokat hanem már lefordított telepítőket lehet elérni a rendszereknél.
  + Mivel böngészőn keresztül érhetőek el a fejlesztendő rendszer elemei ezért a Microsoft Edge, Google Chrome, Firefox böngésző – vagy annak alternatívái is szükségesek.
* Fejlesztői környezet: Microsoft Visual Studio Code (VS Code)
  + A VS Code egy fejlesztői környezet (IDE), amely segíti a kódszerkesztést, hibakeresést és a projektkezelést. Használata a kódszerkesztési és projektmenedzsment folyamatokat egyszerűsíti és támogatja.
  + Hasznos eszköz a kódszerkesztéshez, hibakereséshez és a projektkezeléshez, és könnyen integrálható sok egyéb fejlesztést segítő elemhez.
    - Például telepíthetők kiegészítők és bővítmények több programozási nyelvhez, fájl formátumhoz. E mellett a beépített terminál használható a parancssoros műveletek végrehajtására.
* Backend és Frontend rendszer kialakításához használatos környezetek
  + Ahhoz, hogy egy Backend (Django) és Frontend (React) rendszert egy lokális Windows operációs rendszerű gépen futtassunk, szükségünk van néhány szoftveres műszaki előfeltételre.
    - 1. Python: A Django egy Python alapú keretrendszer, ezért a Python programozási nyelv telepítve kell legyen a fejelsztéshez használatosszámítógépen.
      * A hivatalos Python weboldal: https://www.python.org. Innen letölthető a legfrissebb stabil verzió.
    - 2. Node.js és npm: A React egy JavaScript könyvtár, ezért szükség van a Node.js-re és a hozzá tartozó csomagkezelőre, az npm-re.
      * A Node.js hivatalos weboldala: https://nodejs.org. Innen letölthető a legfrissebb stabil Node.js verziót, amihez az npm is tartozik.
    - 3. Django: Szükséges telepíteni a Django keretrendszert a Python környezetbe. Miután a Python telepítése megtörtént a parancssorban (Command Prompt vagy PowerShell) kiadható a a következő parancs a
      * Django telepítéséhez:
        + pip install Django
      * Django-projekt létrehozása (Szintén parancssoros művelet, melyet a kívánt munkakönyvtárban kell futtatni. Ez létrehoz egy új Django projektet a megadott "myproject" nevű mappában.):
        + django-admin startproject myproject
      * Virtuális környezet kialakítása erősen ajánlott. Ezzel elkülöníthető a rendszer függősége más Python projektektől. A következő parancs a „mappanév” alkönyvtárban hoz létre egy egy új virtuális környezetet.
        + python -m venv **mappanév**
      * Django függőségek telepítéséhez a aktiválni kell a virtuális környezetet és telepíteni a szükséges modulokat. ( A reguirements.txt egyedileg előállított állomány):
        + .\env\Scripts\activate
        + pip install -r requirements.txt
      * Az adott Django projekt-ben használt modulok a következők – abc sorrendben
        + Az "asgiref" modul egy referencia-implementációt nyújt az ASGI (Asynchronous Server Gateway Interface) szabványhoz. Az ASGI egy interfész, amely lehetővé teszi aszinkron webalkalmazások fejlesztését a Django keretrendszerben.
        + A "Django" modul maga a Django keretrendszer, amely egy nagy szintű webalkalmazás-fejlesztési keretrendszer. A Django rendkívül hatékonyan segíti a webalkalmazások fejlesztését, ahol gyakran használt funkciókat és komponenseket tartalmaz, például URL kezelést, adatbázis modellezést, felhasználókezelést stb.
        + A "django-cors-headers" modul lehetővé teszi a Cross-Origin Resource Sharing (CORS) beállítását a Django alkalmazásban. A CORS segítségével a frontend alkalmazások kéréseket küldhetnek az alkalmazásodhoz, még akkor is, ha a kérés forrása más domainről vagy portról érkezik.
        + A "djangorestframework" modul egy kiterjesztett és rugalmas keretrendszert kínál a RESTful API-k (Representational State Transfer) fejlesztéséhez Django-ban. A modul tartalmazza a népszerű "Model-View-Controller" (MVC) tervezési mintát, amely lehetővé teszi az egyszerű és hatékony API-k készítését.
        + A "Pillow" modul egy képfeldolgozási könyvtár, amely lehetővé teszi a képek manipulálását a Django alkalmazásban. A modul számos funkciót kínál, például kép méretezést, forgatást, szűrők alkalmazását stb.
        + A "pytz" modul egy időzónakezelő könyvtár, amely segít a helyes időzónák kezelésében a Django alkalmazásban. Ez fontos lehet, ha az alkalmazásod különböző időzónákban működik vagy időzóna-specifikus időpontokat kell kezelned.
        + A "sqlparse" modul egy SQL szövegelemző és formázó, amely segít a Django alkalmazásban használt SQL lekérdezések olvashatóságának javításában. A modul lehetővé teszi, hogy szépen formázzuk és jobban megértsük azon SQL lekérdezéseket, amiket a Django generál.