

*Miskolci Egyetem  
Gépészmérnöki és Informatika Kar  
Általános Informatikai Intézeti Tanszék*



**MISKOLCI**  
E G Y E T E M  
UNIVERSITY OF MISKOLC

# **Egy magánlinkia webalkalmazása**

## **SZAKDOLGOZAT**

***Készítette:***

Baumel Márton Benedek

O09CTQ

Mérnökinformatikus BSc hallgató  
Korszerű WEB technológiák szakirány

***Témavezető:***

Agárdi Anita

Tanársegéd

# Tartalomjegyzék:

1. Megvalósíthatósági Tanulmány
2. Bevezetés
3. Irodalom feldolgozása, háttér információk
4. A szoftverfejlesztési lépéseinek megfelelően, a fejlesztési munka leírása
  - 4.1 Előkészületek
  - 4.2 Programspecifikáció
    - 4.2.1. Bemenő és Kimenő adatok
    - 4.2.2. A rendszer funkciói és Képernyőtervek
    - 4.2.3. Hardver és szoftver követelmény
  - 4.3. A felhasználható fejlesztőeszközök kiválasztása
  - 4.4 A terv ismertetése

# 1. Megvalósíthatósági Tanulmány

Ezzel a webalkalmazással képes lesz egy magánklinika doktora menedzselni a beteg kórtörténetét, képes lesz szabad időpontokat kiadni, amit a beteg meg tud igényleni vizsgálatra és ezeket az igényeket fogja tudni kezelni, azaz képes lesz elfogadni vagy visszautasítani. A beteg szemszögéből képesek leszünk látni a saját kórtörténetünket, képesek leszünk kiválasztani az adott doktornál a doktor által nyitott időpontok egyikét és képesek leszünk az időpontjainkat követni az alkalmazás segítségével.

Az alkalmazást orvosok és páciensek lesznek képesek használni, mindkettő felhasználó körnek saját felhasználó felülete lesz.

Orvosok szempontjából számítani fog a specializációjuk, de ezen felül mindegyikük ugyan olyan lehetőségekkel fog rendelkezni az alkalmazás használatakor.

Páciensek különböző végzettségű, beosztású hozzáértésű emberek körét takarja ezért az alkalmazás megvalósításakor törekedni kell az átlátható oldal kinézetre és megteremteni a minél kényelmesebb felhasználását az oldalnak, hogy ezek a tényezők ne játszanak szerepet,

A program írásakor nincsenek megmeglévő bemenő dokumentumok és a program nem állít elő dokumentumot.

A program funkciói:

- Orvos szempontból:
  - Bejelentkezés
  - Időpont kiírás
  - Időpont igény elfogadása/vissza utasítása
  - Email-en való jelzés
  - Saját időpontok követése
  - Páciensek megtekintése

- Páciens kórtörténet megtekintése
- Páciens kórtörténetéhez új bejegyzés

- Páciens szempontjából:

- Regisztráció
- Bejelentkezés
- Időpont kiválasztása és megigénylése
- Saját időpontok követése
- Email jelzés
- Kórtörténet megtekintése

A felület megalkotásakor a legfontosabb számomra az átláthatóság és a minél könnyebb felhasználói élmény megalkotása elérése volt.

A programot webalkalmazásként képzeltem el ezért egy böngésző futtatására alkalmas hardveren, ami csatlakozik az internethez többre nem lesz szükség az alkalmazás használatához.

A programhoz nem tartozik megrendelő ezért egyedül alkottam meg a szoftver paramétereit.

Feladat	Megvalósíthatóság
Mire való a szoftver, milyen feladatokat lehet vele elvégezni?	Érthető és a jelenlegi tudásanyagommal megvalósítható.
Kik fogják kezelni, használni a szoftvert?	Érthető és megvizsgáltam kik fogják használni.
Végzettség, beosztás, kor, hozzáértés, tapasztalat?	Érthető és megvizsgáltam kik fogják használni.
Milyen bemenő- és kimenő adatokat használ?	Érthető és nem lesznek bemenő és kimenő adatok.
Milyen funkciói legyenek a programnak?	Érthető és a jelenlegi tudásanyagommal megvalósítható.

Milyen kinézetű, felületű legyen a program?	Érthető és a jelenlegi tudásanyagommal megvalósítható.
Milyen (hardver, szoftver) környezetben működik majd a program?	Érthető és lefektettem a hardveres követelményeket.

## 2. Bevezetés:

Az informatika technológiai újdonságai drasztikusan meghatározzák életünk minden egyes szegmensét, gondolhatunk a legegyszerűbb feladatoktól, mint például az otthoni fűtés beállítása, ma már könnyedén betudjuk állítani a telefonunkról vagy éppenséggel egy weboldalon keresztül hogy milyen formába szeretnénk a lakás hőmérsékletét tudni vagy a legösszetettebb dolgokig, mint egy óriási vegyi rendszer működtetése, egyszerűen a jelenkor vívmányai nélkül ezek a folyamatok nehezen működnének vagy éppenséggel egyáltalán nem ilyen formában valósulnának meg hanem megnehezítenék a mai életünket.

Ezen folyamat alól az orvostudomány és a beteg ellátás sem kivétel. A legtöbb műszer ma már valamilyen számítógépre küldi el a méréseinek az adatait és a számítógépen történik meg a kiértékelés folyamata is már. Ezzel rengeteg előnyhöz tudunk jutni mivel ezeknek a vizsgálatoknak a kiértékelési folyamat ideje lecsökkent, emellett az emberi hiba esélye is kevesebb mert egy gép végzi el ezeket a kritikus számításokat, ahol nagy a hibázásra a lehetőség.

Emellett az újdonságok úgy is képesek segíteni az orvosok munkáját és a betegek életét, hogy csak közvetett módon avatkozik bele. A beteg regisztráció ma már nem papíron történik, hanem akkor, amikor is megérkezik a beteg, a regisztrációs pultnál kért okmányait felmutatva egy rendszer végzi el a kért okmány alapján a beteg azonosítási folyamatot, amivel biztonságosabb és gyorsabb ez a folyamat.

Emellett az orvosnak is átláthatóbb és egyszerűsödik ezzel a feladata mivel tudni fogja, hogy éppen hány beteg várakozik kint a rendelőjében, képes lesz a beteg kórtörténetét gyorsan áttekinteni annak érdekében, hogy jobb ellátást tudjon biztosítani. Ma már ha a beteg valamilyen gyógyszerre szorul képes az EESZT rendszerén felírnia a receptet és a páciens közvetlenül kitudja majd egy patikába váltania azt.

Számomra mindig is probléma volt, hogy ha egy orvosnál voltam valamilyen vizsgálaton akkor én betegként soha nem tudtam visszanézni a kórtörténetemet vagy éppenséggel az orvos által adott

tanácsokat, amik nem kerültek fel a leletemre. Ha erre volt is valamilyen lehetőség akkor azt általánosan papír formájába tudtam csak megtekinteni, amin csak egy adott lelet volt leírva és nem lehet azonnal visszatekinteni a korábbi problémákat.

Számomra még nagy probléma, hogy nem tudom saját magam számára kiválasztani a megfelelő időpontot, hanem a legjobb esetben kapok egy napot amikor eltudok menni egy rendelésre és valamikor sorra tudok kerülni. Ez nagyon sokszor frusztráló, mivel kiszámíthatatlanná teszi azt a napot, amikor ellátáshoz tudok jutni, mert nem tudhatom előre mikor fogok sorra kerülni.

Ezért a szakdolgozatom témájának egy olyan rendszert készítek, amivel erre a probléma körre megoldást nyújtok egy könnyen átlátható, reszponzív webalkalmazás segítségével egy magánklinika számára.

### 3. Irodalom feldolgozása, háttér információk

Az én alkalmazásom nem egy valós cég megbízásából készül el ezért minden aspektusában, mint a megjelenés és az alkalmazáshoz használt eszközök és technológiákban én magam döntöttem el, hogy hogyan is fogom megvalósítani. Emellett semmilyen meglévő nyilvántartási rendszer vagy bármilyen adat nem áll a rendelkezésemre ezért ezeket az adatokat is saját magam fogom kitalálni.

Mielőtt még az alkalmazás fejlesztéséhez hozzá kezdek, megvizsgáltam a Magyarországon elérhető hasonló témában készült alkalmazásokat, hogy milyen funkciókkal rendelkeznek, milyen technológiákkal lettek lefejlesztve ezenkívül próbáltam őket összehasonlítani és kitalálni, hogy milyen funkciókkal lehetne jobbá tenni ezeket a meglévő megvalósításokat és ezt a tudást felhasználva próbálom megtervezni a saját alkalmazásomat.

Az egyik legnépszerűbb oldal ahol időpontokat lehet foglalni egy szakorvoshoz a Dokio[1] ahol is a páciensnek lehetősége van időpontot foglalni a saját kedve szerint, úgy, hogy kiválasztja a szakorvost akihez szeretne menni, ezek után az orvos által biztosított időpontok közül választ egyet és ezzel történik meg a időpont egyeztetés. Ez az oldal nem egy klinikának biztosítja a szolgáltatásait, azaz egyszer lehet nála foglalni különböző klinikára különböző orvosokhoz.

Az oldal biztosít online tünetellenőrzést, amivel a beteg, ha nem is tudja eldönteni, hogy konkrétan milyen orvost keressen meg a problémájával ezzel az eszközzel képes a válaszai alapján a rendszer ajánlani neki egy szakorvost, akihez fordulhat a problémáival. Ezzel a segítséggel a beteg gyorsabban és pontosabban találhat szakorvost, ami óriási segítség lehet, hogy ha sürgősen kell segítséget találnia.

Ezen kívül még az oldal biztosít egy Tudástár nevezetű funkciót, ahol is hasznos információkat lehet kapni az adott problémáról. Az oldal leírása szerint ezeket az információkat az adott szakterület egyik elismert szakértője írta vagy lektorálta, amivel a páciens pontos adatokhoz juthat, ami szintén nagy segítség lehet a betegnek mivel is nem kell saját magának felkutatnia és kideríteni



az adott betegséghez tartozó információkat, hanem kézhez kapja a hiteles és pontos leírását az adott problémának.

Az oldal felépítés és kinézet megfelel a mai weboldal szabványoknak, amivel képes jó felhasználói élményt biztosítani, ami ezért fontos, mivel több tanulmány is igazolja, hogy a felhasználó hamarabb veszi igénybe az oldal szolgáltatásait, ha az oldal felépítése megfelel a mai szabványoknak [2].

Több fajta foglalási oldal létezik a Dokio-on kívül, de mivel az én alkalmazásom nem specifikusan csak egy időpontfoglaló oldal ezért próbáltam keresni konkrét magánklinika oldalt, hogy információkat gyűjtsek róla. Mivel országszerte rengeteg ilyen fajta oldal létezik ezért a saját környékemre szűkítettem le a keresést és kiválasztottam kettő ilyen magánklinikát ami Miskolcon működöik és rendelkezik webes megoldásokkal.

A Macroklinika [3] rendelkezik egy online oldala, ahol is megtalálhatók a szakterületek és a hozzájuk tartozó orvosok. Emellett az oldal tartalmazza a különböző kezelésekhöz tartozó ár listákat és megtalálható egy hírek felület, ahol is az aktuális információk vannak feltüntetve, mint például milyen új kezelések érhetőek el, vagy éppenséggel milyen akciók vannak jelen pillanatban.

Ezekén felül az oldal biztosít online időpontfoglalást, ahol is képesek vagyunk kiválasztani az adott szakkezelést és a kezeléshez tartozó orvost, ezek után az orvos által biztosított időpontok közül választva tudjuk lefoglalni a saját időpontunkat.

Az oldal kinézet, nem felel meg a ma elvárt design követelményeknek, mint például a Jakob's Law [4] azaz az oldal megjelenése nem hasonlít a ma elvárt szabványokhoz ezért több felhasználónak is kényelmetlen lehet a használat, emellett a Goal-Gradient Effect[5] sem teljesül, azaz a szín átmenet helyett feltűnő színeket használ annak érdekében, hogy a felhasználó figyelmét felkeltse az adott funkcióra ezzel pedig megtöri az oldal szín sémáját, ami egyrészt a felhasználói élményt ronthatja emellett lehetséges, hogy a keresett információt emiatt nem találhatja meg az oldalon, ami miatt potenciális ügyfelektől eshet el.

A másik választott magánklinika oldala az a Erzsébet Fürdő Gyógyászati és Szűrőközpont[6] oldala. Ennek az oldalnak a főoldala képezi a híreket ezzel pedig képes a felhasználónak rögtön információt adni az aktuális hírekről, azaz például, hogy egy megjelent egy új fajta kezelés vagy éppenséggel egy adott kezelés jelenleg akciós.

Emellett az oldalon megtalálható egy részletes lista a kezelésekről és a hozzájuk tartozó orvosokról. Ha kiválasztunk egy kezelést akkor megjelenik az időpontfoglalás lehetősége is, ahol a szakterülethez tartozó orvosok közül választhatunk és az orvos által adott időpontok közül tudunk választani saját magunk számára a többi oldalon megszokott módon.

Miután megvizsgáltam ezeket az oldalakat megpróbáltam össze gyűjteni azokat a lényeges funkciókat és technikai követelményeket, amiket a saját megoldásomban szeretnék használni.

Az alkalmazásom adatbázisának a leszűrt tapasztalatok alapján nem lenne jó döntés relációs tábla használata, mivel is rengeteg fajta adatot tartalmaznak ezek az oldalak, mint például: híreket, ár listákat, orvosi rendeléseket, orvosok adatai, időpont foglalásokat és még számos adatot és ezért annak érdekében, hogy minél rugalmasabban tudjam megalkotni az alkalmazásomat és későbbiekben minél hatékonyabban tudjam a változásokat kezelni ezért nem alkalmazhatok relációs táblát.

Számomra legkézenfekvőbb ilyen adatbázis rendszer a MongoDB[7], mivel is a keresett funkciókat képes vagyok vele megvalósítani és korábbi tapasztalattal is rendelkezek a használatával.

Mivel webalkalmazást szeretnék csinálni ezért célszerű szétbontanom az alkalmazásomat backend (ez a része lesz az alkalmazásnak, ami a felhasználó kéréseit és az adatbázis elérését és használatát fogja biztosítani) illetve frontend részre (ez a része fogja az alkalmazás megjelenítését és a felhasználóval való kommunikációs felületet biztosítani).

A backend részt korábbi tapasztalatimból kiindulva JavaScripttel fogom kifejleszteni Express webalkalmazás keretrendszer használatával [8]. A frontend részt pedig Angular keretrendszerrel

fogom lefejleszteni, amivel képes leszek egy egysége kinézetet adni az alkalmazásomnak amellet, hogy az alkalmazás funkciót komponensekbe tudom rendezni, amivel a kód átláthatóságát tudom növelni [9].

A technikai szükségletek leszűrése után az alkalmazásom funkcióinak a meghatározása a következő feladatom a minták tanulmányozása alapján, amivel pontosabb képet sikerül kapnom arról, hogy milyen funkciók is szükségesek az alkalmazásomban és ezeket a funkciókat hogyan is lehetne megvalósítani úgy, hogy a hasonló alkalmazások hiányosságait kitudjam javítani.

Az elsődleges és legfontosabb funkció, ami minden ilyen alkalmazásban szerepelt az időpontfoglalás lehetősége, úgy, hogy az orvos képes megadni azokat az időszavakat amik közül a beteg tud válogatni és a beteg kitudja választani a számára megfelelő időpontot. Ezt a funkciót annyiba egészíteném még ki, hogy egy vissza igazoló email-t szeretnék küldeni a foglalásról és az időpont előtti nap szeretném majd a páciensnek jelezni, hogy foglalása van a holnapi napra.

Szeretném én is a főoldalon megjeleníteni az aktuális híreket, amivel a páciens rögtön tájékozódhat az új kezelésekről vagy éppenséggel, hogy ha valami rendkívüli történik azt már a főoldalon láthatja. Emellet szeretném a kezeléseket egy aloldalon megjeleníteni, ahol a páciens képes lenne az adott szakorvosról és a kezelésről információt kapni és ezek után egyből letudja foglalni az időpontját a kiválasztott kezelésnél a kiválasztott orvosnál.

Emellet a legnagyobb eltérés a vizsgált oldalakhoz képest az az, hogy az orvos és a páciens képes lenne az oldalon egymással közvetlenül kommunikálni, megbeszélni az adott problémát ezzel is gyorsítva a betegnek az ellátást. Ezeken felül a páciens képes lenne a saját kórtörténetét visszatekinteni, ami sok segítség lehet akkor, ha elfelejtette volna, hogy milyen tanácsokkal látta el az orvosa.

Ezeket a funkciókat egy korszerű és a mai weboldalak megjelenítés szabályainak megfelelően szeretném biztosítani annak érdekében, hogy a felhasználói élmény minél zökkenőmentesebb lehessen.

## **4. A szoftverfejlesztési lépéseinek megfelelően, a fejlesztési munka leírása**

A további fejezetekben fogom kifejteni a Magánklinika Webalkalmazásom implementálásához szükséges előkészületeket, programspecifikációkat. Ezután a program fejlesztéshez felhasználható fejlesztőeszközöket mutatom be, majd a programterv mellett az érdemi megvalósítást, tesztelést és végül a felhasználó leírást fogom bemutatni.

### **4.1 Előkészületek**

Az elkészítendő feladatom egy magánklinika webalkalmazása, ami azt a célt szolgálja, hogy egy kitalált klinikának biztosítson minden olyan szolgáltatást, ami ahhoz szükséges, hogy minél egyszerűbb legyen az orvos munkája és minél jobb ellátást kapjon a páciens.

Az informatika és az egészségügy fejlődése az elmúlt évtizedekben egymást segítve drasztikusan fejlődött és megváltozott. Egy 2004-ben készült cikk alapján, amit egy belgyógyász osztályvezető főorvos adott „Az informatika a betegellátás eszközévé vált, szervesen beépült a napi munkafolyamatokba. A betegről összegyűjthető, rendszerezhető és nyilvántartható információk integrációját segíti elő, és egyben - a gyógyászatban nélkülözhetetlen - döntéstámogatást is biztosítja a meglévő tudásbázis elérhetőségével. A sokrétű betegellátó tevékenység más elemeihez hasonlóan az informatika is jelentős költséget képvisel, amit be kell építeni az egészségbiztosítás költségtérítési rendszerébe. A gyógyításban a naponta észlelhető előnyökön túl a gazdálkodási feladatok, hatósági jelentési kötelezettségek és vezető információs területen is kíváncsok vizsgálni az informatika költséghatékonyságban megnyilvánuló szerepét.”[10] ebből is látható, hogy az informatikai fejlesztések elengedhetetlenné váltak mára az orvostudomány számára és a betegellátásban is nélkülözhetetlen szerepet tölt be.

Az alkalmazásom ötlete onnan ered, hogy amikor is bármilyen orvosnál jártam, nagy sok zavaró tényezővel és rengeteg olyan problémával találkoztam, amit véleményem szerint könnyedén kilehet küszöbölni egy webalkalmazás segítségével annak érdekében, hogy a minél jobb és korszerűbb szolgáltatást kapjon a beteg.

Emellett ugyan ezen az oldalon szeretnék egy olyan felület létrehozni, amivel is orvosként is betudnék lépni annak érdekében, hogy az elérhető rendelési időpontjaimat saját magam tudjam testre szabni és képes legyek a beteggel egy dedikált helyen kommunikálni annak érdekében, hogy minél gyorsabban és kényelmesebben tudjon segítséget kapni.

Mivel az alkalmazásomban kettő szerepkör létezik ezért a felhasználókat páciens és orvos szerepkörre fogom felosztani annak érdekében, hogy minél pontosabb leírást tudjak adni arról, hogyan is képzeltem az alkalmazásom használatát.

Bármilyen szerepkörű felhasználó az oldal megnyitásakor látni fogja az oldal leírását és az aktuális híreket.

A doktor szerepkörű felhasználók nem tudnak az oldalon regisztrálni, hanem külön kérésre kerülnek rögzítve az adatbázisba, de bejelentkezni ugyan úgy tudnak, mint a páciens felhasználók. A sikeres bejelentkezés után őket egy olyan főoldal fogadja majd, ahol is látják, hogy az adott nap milyen eseményei is lesznek. Az oldalon ezen kívül képesek lesznek megnézni milyen későbbi eseményei lesznek és képes lesz a felkérésekre reagálni. Megfogja tudni nézni a páciensekkel folytatott beszélgetéseit itt tud küldeni új üzenetet és visszanézni a korábbi üzeneteit. Ezen kívül képes lesz az adott naptári hetet kiválasztva új időpontokat kiírni a rendeléseihez és a profiljához tartozó adatokat is képes lesz módosítani, kivéve a nevét és azonosító számát.

A páciens szerepkörű felhasználó a regisztrációt követően rendelkezni fog egy fiókkal amivel később befog tudni jelentkezni. A főoldalon fogja látni az adott napra számára vonatkozó eseményeket. A páciens képes lesz megnézni a későbbi eseményeit és képes lesz az adott szakrendelésekhez tartozó rendelésre jelentkezni, amit az orvos felhasználó jóváhagyása után lesz hivatalosan is elfogadva. Képes lesz a doktorral folytatott beszélgetést megtekinteni, ahol a szakrendelések utáni hasznos információkat vissza tudja követni és ha bármilyen kérdése lenne azt az orvosnak le is tudja írni. Ezenfelül a páciens is képes lenne változtatni a fiókjához köthető adatokat kedve szerint, kivéve a nevét és TAJ számát.

Ezek a feladatok megvalósításán kívül arra fogok törekedni, hogy minél átláthatóbb, felhasználó barát oldalt tudjak létrehozni annak érdekében, hogy a felhasználóknak bármilyen háttérrel is rendelkezzenek, a felhasználói élmény minél jobb legyen számukra.

## **4.2 Programspecifikáció**

### **4.2.1. Bemenő és Kimenő adatok**

Mivel az alkalmazásom ötletét saját magam találtam ki ezért már létező adatforrásom nincsen ezért ilyen fajta bemenő adatokkal nem rendelkezek. Emellett bemenő adataim csak a felhasználók által lesznek generálva.

Doktor szerepkör által generált adatok:

-Profil adatok

- Teljes név
- Email
- Jelszó ("hashed" formában lesz eltárolva)
- Telefonszám
- Doktor azonosító (a rendszer által generált 4 karakterből álló egyedi azonosító)
- Profil kép (png vagy jpg formátumban)

-Üzenet adatok: szöveg adatok

-Esemény adatok

- szabad időpontok
- lefoglalt időpontok
- korábbi időpontok

Páciens által generált adatok:

-Profil adatok

- Teljes név
- Email
- Jelszó ("hashed" formában lesz eltárolva)
- Telefonszám
- TAJ szám
- Profil kép (png vagy jpg formátuban)

-Üzenet adatok: szöveg adatok

-Esemény adatok:

- visszaigazolt időpontok
- visszaigazolatlan időpontok
- korábbi időpontok

A felhasználók által generált adatok egy adatbázisban lesznek elmentve és később felhasználva.

Az alkalmazásomban nagy szerepet tölt be a rugalmasság és a bővíthetőség, ezért a fejlesztést nem relációs adatbázissal terveztem implementálni mivel is ez olyan kööttséget jelentene számomra, ami a későbbi fejlesztési feladatokat nagyban megnehezítené.

Ezt a MongoDB-vel terveztem megvalósítani mivel is ezzel a nem relációs adatbázissal van a legtöbb tapasztalatom. Emellett a fejlesztés alatt képes biztosítani ingyenes lokális megoldást, viszont a későbbiekben, ha szeretném van rá mód, hogy az összes adattomat egy felhő szolgáltatás által tudjam elérni.

## 4.2.2. A rendszer funkciói és Képernyőtervek

Mivel szeretném, hogy az alkalmazásom megjelenése minél kiforrottabb legyen még a fejlesztés elkezdése előtt, ezért megterveztem az összes felhasználó felületem vázlatát a Figma tervező program segítségével, ezzel rengeteg későbbi munkát tudok megspórolni mivel is képes vagyok az alkalmazásomnak felületei vázlat szinten látni és ha bármilyen probléma felmerülne még ebben a változatában képes vagyok azt kijavítani. Emellett láthatóm azt is, hogy milyen felhasználók láthatják az adott oldalt ezzel pedig tisztázni tudom az adott oldal láthatóságát a felhasználók szerepköréhez.

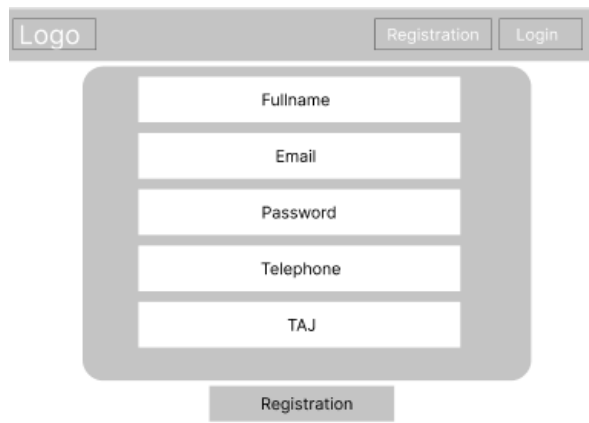
**Minden felhasználó számára elérhető oldalak és funkciói:**

Főoldal megjelenése:





Regisztráció megjelenése:



The registration form UI mockup features a header bar with a 'Logo' button on the left and 'Registration' and 'Login' buttons on the right. The main content area is a light gray rounded rectangle containing five white input fields stacked vertically, labeled 'Fullname', 'Email', 'Password', 'Telephone', and 'TAJ'. Below this container is a 'Registration' button.

Bejelentkezés megjelenése:

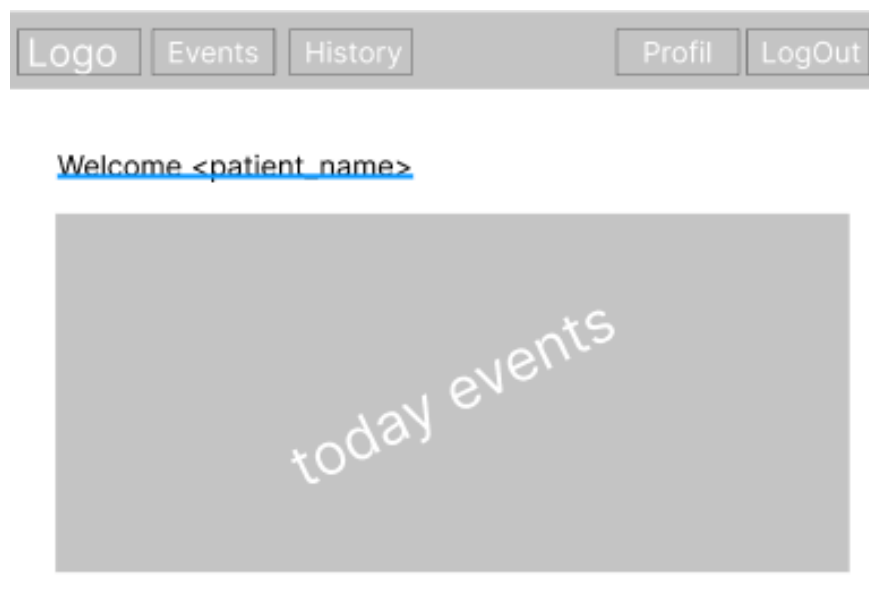


The login form UI mockup features a header bar with a 'Logo' button on the left and 'Registration' and 'Login' buttons on the right. The main content area is a light gray rounded rectangle containing two white input fields stacked vertically, labeled 'Email' and 'Password'. Below this container is a 'Login' button.

**A pácienshez tartozó oldalak megjelenése és funkciói:**

Főoldal:

Ezen a felületen láthatja a felhasználó milyen eseményei a mai napon



The patient home page UI mockup features a header bar with a 'Logo' button on the left, 'Events' and 'History' buttons in the center, and 'Profil' and 'LogOut' buttons on the right. The main content area displays a blue underlined text 'Welcome <patient\_name>' followed by a large gray rectangular box containing the text 'today events' in a light gray, italicized font.

Események kezelése:

Ezen a felületen tudjuk megnézni milyen már biztosan elfogadott rendelései lesznek a páciensnek a közeljövőben:



Rendelésre való jelentkezés:

Először is szükségünk lesz kiválasztani az adott rendelést és rendeléshez tartozó orvost, ezek utána az ő által megadott időpontok közül választva tudunk választani saját magunknak időpontot amit az orvosnak elkell majd fogadnia:

The image illustrates a three-step process for booking an appointment in a web application.

**Step 1: Search for treatment**

The interface shows a navigation bar with "Logo", "Events", "History", "Profil", and "LogOut". Below the navigation bar, there is a search bar labeled "Search for treatment". A grid of buttons is displayed, each containing "Treatment Name" and "Doctor Name". The first button in the top-left corner is highlighted with a blue border.

**Step 2: Patient\_Events4\_AfterLogin**

An arrow points down to the second screen. The navigation bar remains the same. Below it, there is a section titled "Fix events" and "Open events for you". A large box on the left is labeled "Doctor Description". To the right, there is a table with three rows of event data:

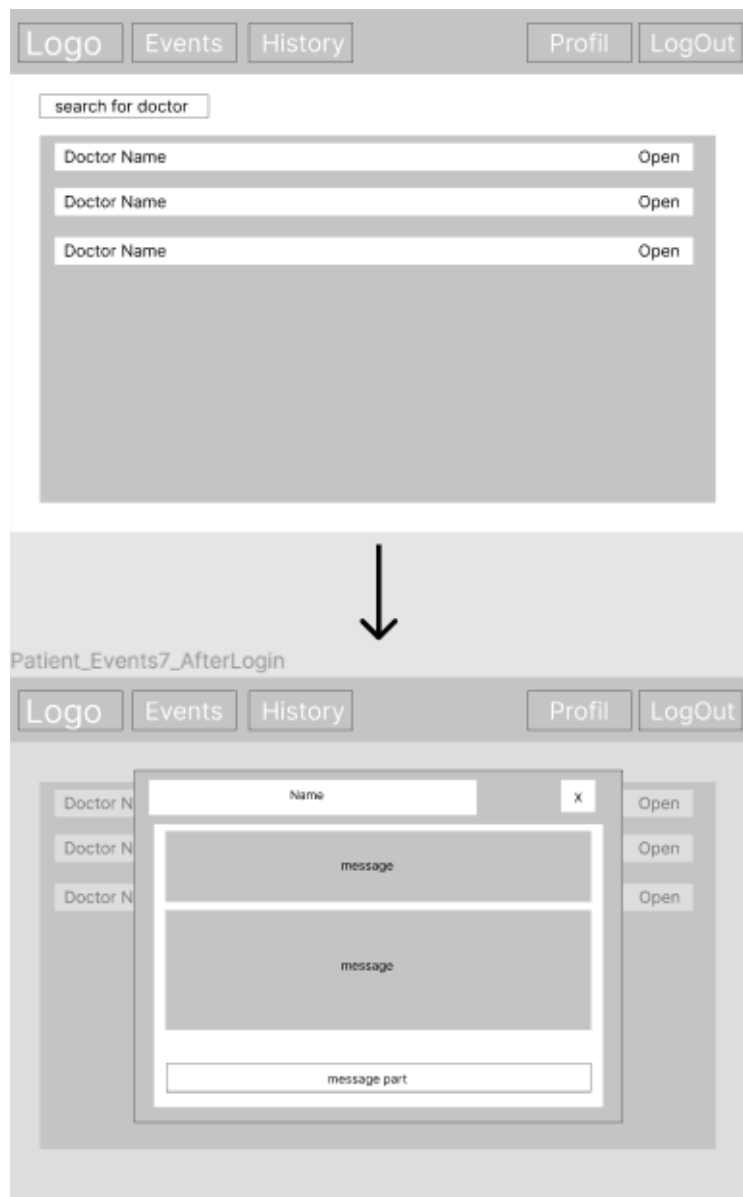
Date	Time	Action
05.10.	10:00-11:00	Request
05.10.	10:00-11:00	Request
05.10.	10:00-11:00	Request

**Step 3: Patient\_Events5\_AfterLogin**

An arrow points down to the third screen. The navigation bar is the same. A modal window is open, titled "Event Description With Doctor Name". The modal contains three input fields: "Event Description With Doctor Name", "Problem", and "Time". Below these fields are two buttons: "Request" and "Back".

History funkció:

Ebben a funkcióban leszünk képesek az orvosunkkal beszélgetést folytatni és a korábbi tanácsokat is itt vissza fogjuk tudni követni:



Profil és profilbeállítások:

A többi elemnek a cseréje is hasonló módon történik meg, mint a jelszóé, azaz a változtatás előtt kérni fogja az aktuális jelszót, hogy validálva legyen a felhasználó és csak utána fogja engedni az adatok változtatását a rendszer:

The diagram illustrates the process of changing a password. It starts with a 'Profil' (Profile) page containing fields for Name, Phone Number, Email, and Password, along with a 'Change It' button. An arrow points down to the 'Patient\_Events9\_AfterLogin' page, which shows the 'Change Password' form. This form includes fields for 'Old password', 'New Password \*1', and 'New Password \*2', with a 'Change it' button at the bottom.

**Az orvoshoz tartozó oldalak megjelenés és funkciói:**

Főoldal:

A bejelentkezés után, ezen az oldalon fogja látni a mai napra szóló eseményeit.

The screenshot shows the doctor's main page after login. The navigation bar includes 'Logo', 'Events', 'Patient', 'Profil', and 'LogOut'. The main content area displays a welcome message 'Welcome <doctor\_name>' and a large grey box with the text 'today events' overlaid diagonally.

Lefoglalat események:

Ezen a felületen tudjuk megnézni milyen már biztosan elfogadott rendelései lesznek a doktornak a jövőben:

The screenshot shows a web interface for managing appointments. At the top is a navigation bar with buttons for 'Logo', 'Events', 'Patient', 'Profil', and 'LogOut'. Below the 'Events' button is a sub-menu with 'Fix events' and 'Open events for you'. The main content area is divided into three sections: 'Today', 'Tomorrow', and 'Later'. Each section contains a table of appointments. The 'Today' section has one row with the time '10:00-11:00', the text 'Client Name', and the status 'Open'. The 'Tomorrow' section has two rows, each with the same time, text, and status. The 'Later' section is currently empty.

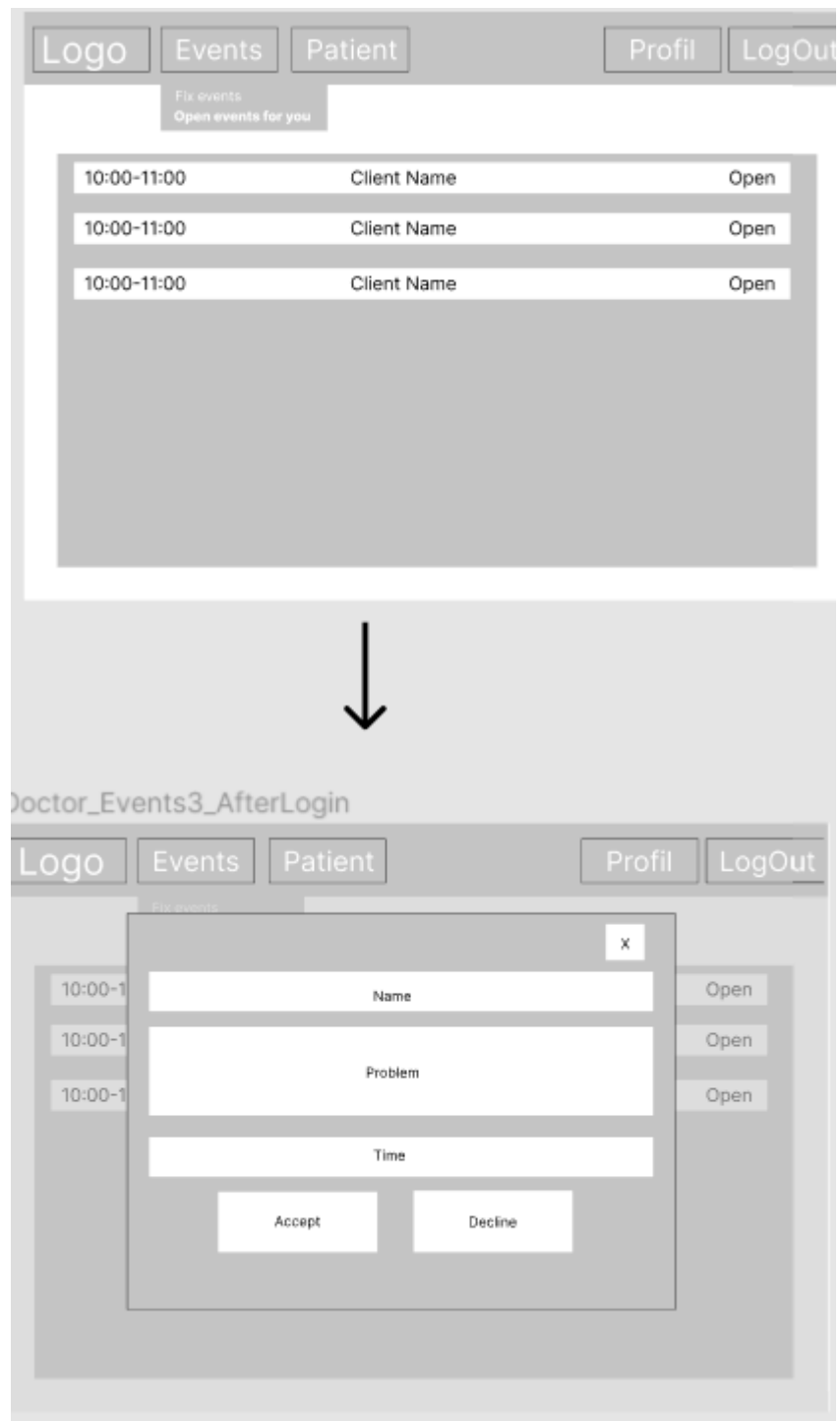
Today		
10:00-11:00	Client Name	Open

Tomorrow		
10:00-11:00	Client Name	Open
10:00-11:00	Client Name	Open

Later		
-------	--	--

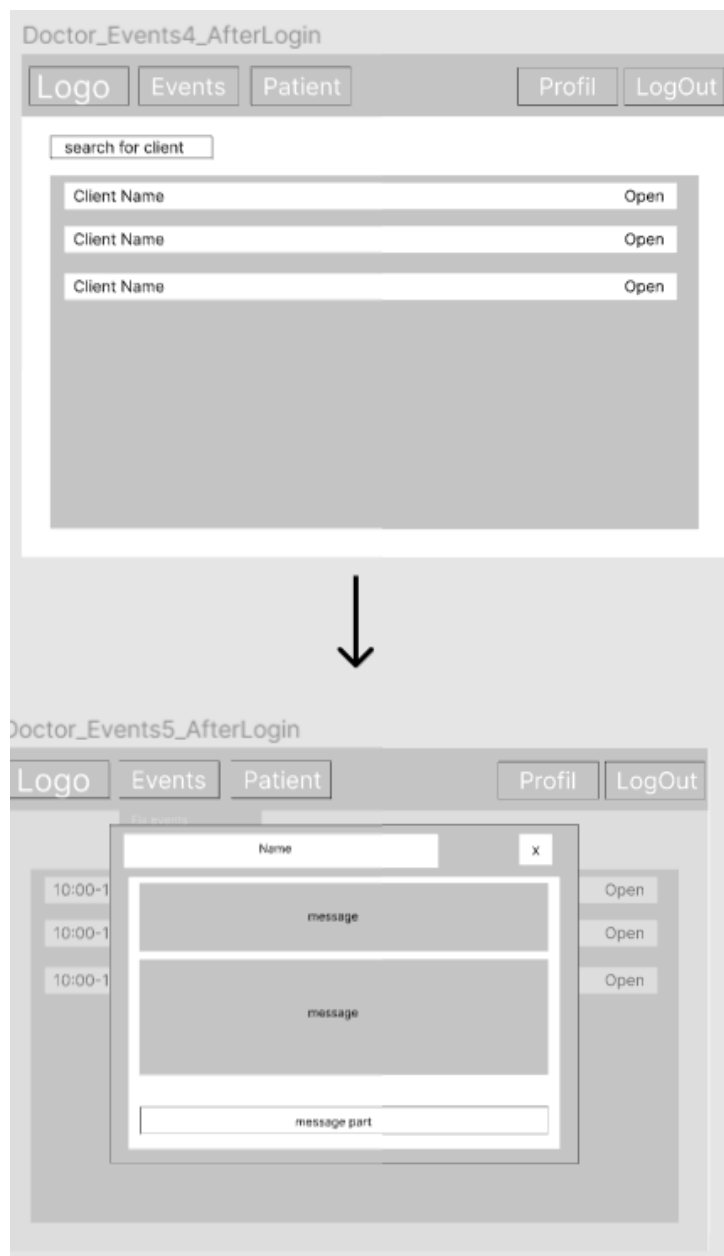
Az elfogadásra váró események:

Ezen a felületen tudjuk megnézni, hogy milyen kérések érkeztek a meghirdetett rendelési időpontra és itt tudjuk elfogadni vagy éppenséggel elutasítani a kérést:



Patient funkció:

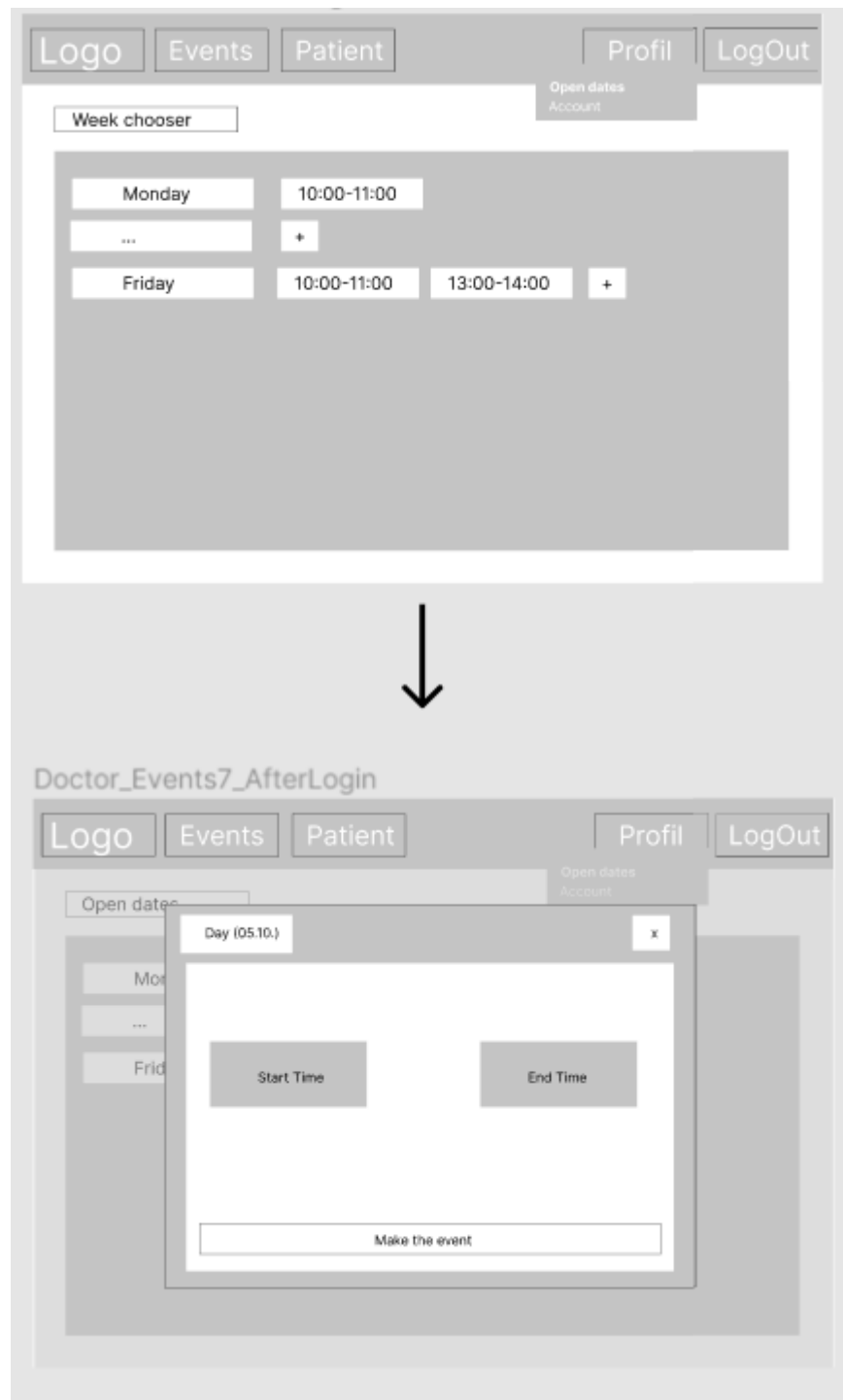
Ezen a felületen leszünk képesek az páciensünkkel beszélgetést folytatni és a rendelés utáni tanácsokat leírni neki.





Időpont megadás:

Ezen a felületen lesz képes a doktor egy adott naptári hetet kiválasztva új időpontot kiadni, úgy, hogy a napot kiválasztva, egy új ablakban megfogja tudni adni a kezdő és befejező időpontot:



## Profil és profilbeállítások:

A többi elemnek a cseréje is hasonló módon történik meg, mint a jelszóé, azaz a változtatás előtt kérni fogja az aktuális jelszót, hogy validálva legyen a felhasználó és csak utána fogja engedni az adatok változtatását a rendszer:

The diagram illustrates the process of changing a user profile. It is divided into two main sections by a downward-pointing arrow.

**Top Section (Profile Page):**

- Navigation bar: Logo, Events, Patient, Profil, LogOut.
- Sub-navigation: Open dates, Account.
- Profile form fields:
  - Profil Image (circular placeholder)
  - Name (text input)
  - Phone Number (text input)
  - Email (text input)
  - Specialization (text input)
  - Password (text input) with a "Change It" button next to it.

**Bottom Section (Password Change Modal):**

Modal title: Doctor\_Events9\_AfterLogin

Navigation bar: Logo, Events, Patient, Profil, LogOut.

Sub-navigation: Open dates, Account.

Modal form fields:

- Old password (text input)
- New Password \*1 (text input)
- New Password \*2 (text input)
- Change it (button)

### 4.2.3. Hardver és szoftver követelmény

Az alkalmazást bármilyen böngészőből elindítva, a felhasználó képes használni és mivel az oldal megjelenése reszponzív, azaz alkalmazkodik az adott kijelző méretéhez ezért tableten vagy telefonon is egy böngészőben megnyitva képes lesz az alkalmazást gond nélkül használni.

Ténylegesen ezért csak annyi követelmény van a felhasználó elé támasztva, hogy rendelkezzen egy böngészővel és aktív internet kapcsolattal az alkalmazás használatához.

## 4.3. A felhasználható fejlesztőeszközök kiválasztása

A webalkalmazásomat Windows 11 (Build szám: 22000.652) rendszer alatt kezdtem el fejleszteni. A választásom azért erre az operációs rendszerre esett mivel is ezzel a rendszerrel van a legtöbb tapasztalatom és ezen a rendszeren rendelkezek az összes fejlesztő eszközzel.

Mivel rengeteg különböző fajta adattal fogok dolgozni és szeretnék képes lenni alkalmazkodni az új igényekhez minél gyorsabban és gördülékenyebben ezért nem használhatok fel relációs táblát. A relációs tábla egyik megkötése az, hogy az adatok és a táblák között kapcsolat jön létre és a kapcsolatban részt vevő adatokat nehezen vagy egyáltalán nem lehet dinamikusán módosítani az új igényekhez.

A nem relációs adatbázisban az adatok nem táblákban vannak rendezve, hanem valamilyen fajta dokumentumban ezzel adva lehetőséget arra, hogy a későbbiekben dinamikusán testre lehessen szabni azt. Számos ilyen rendszer létezik ma már, de én a MongoDB-t választottam azért mert ezzel a rendszerrel van a legtöbb tapasztalatom, emellett a fejlesztés alatt képes biztosítani ingyenes lokális megoldást, viszont a későbbiekben, ha szeretném van rá mód, hogy az összes adattomat egy felhő szolgáltatás által tudjam elérni.

A MongoDB 2007-ben jött létre[11] Dwight Merriman, Eliot Horowitz és Kevin Ryan által akik a DoubleClick nevű internetes reklám cégben dolgoztak. Mivel 400000 reklámot kellett másodperceként szolgáltatniuk ezért a régi adatbázis megoldások nem bírták el a terhelés és nem

tudtak alkalmazkodni a változásokhoz. Ennek a megoldására jött létre a MongoDB, ami egy nem relációs adatbázis, amire szoktak hivatkozni NoSQL adatbázisként is. Az adatokat táblák helyett JSON dokumentumokba vannak tárolva annak érdekében, hogy minél könnyebben lehessen az adattal dolgozni

Magát az adatbázis kezelését nem parancs sorból fogom végezni, hanem a MongoDB Compass nevezetű programmal, mivel ez biztosít egy grafikus felületet, amivel is egyszerűbb dolgozni. A fejlesztés elején az adatbázisomat csak lokális formában fogom tárolni.

A backend a programnak azon része, ami a későbbiekben össze fogja kötni az adatbázist és az adatbázishoz tartozó műveletek elfogja végezni. A nyelv kiválasztásánál számra a legfontosabb szempontok azok voltak:

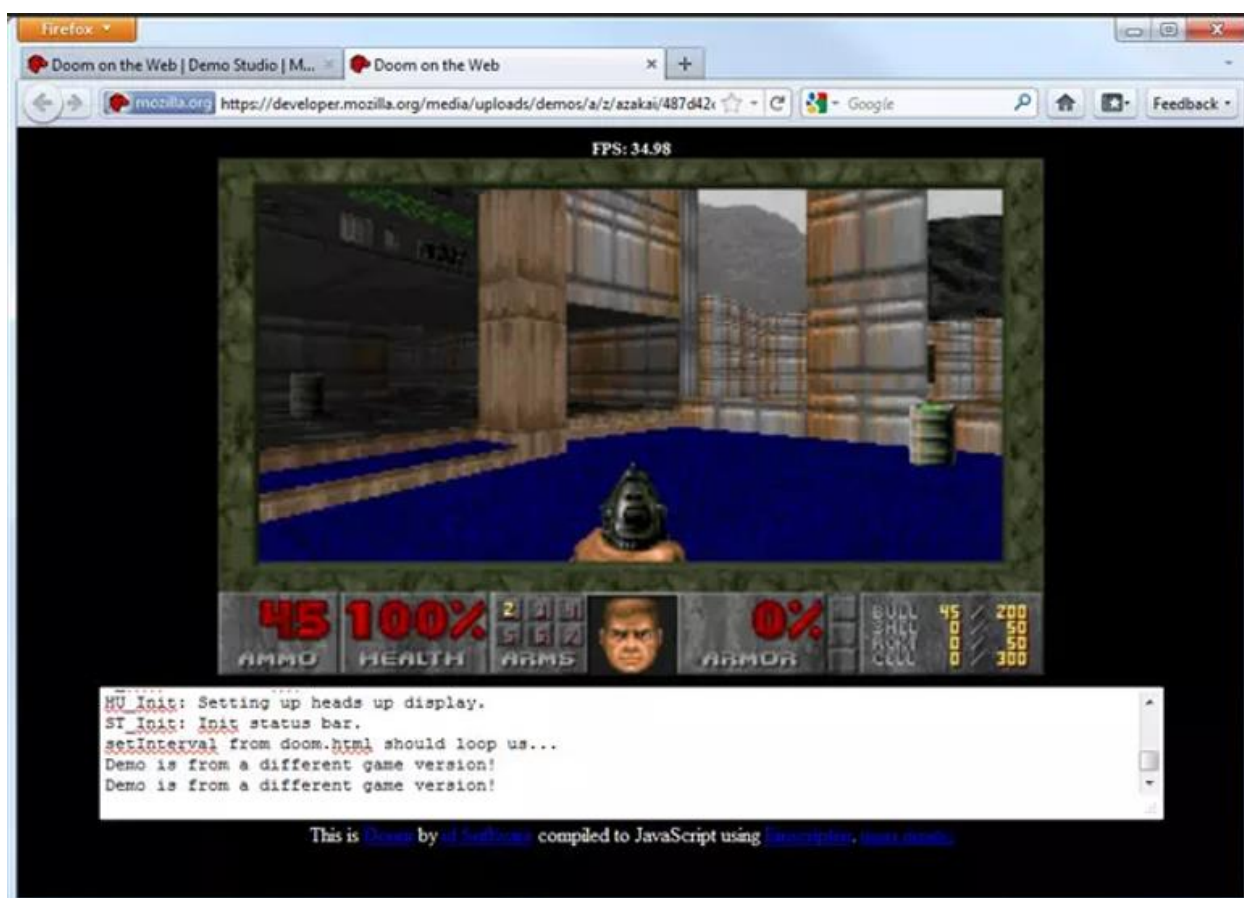
- bármilyen számítógépen tudjam fejleszteni a programot és tudjam futtatni rajta
- a használat egyszerű legyen
- a fejlesztő eszközök és környezte legyen ingyenes
- elkerüljem az, hogy az alkalmazásom robosztus legyen
- rendelkezzek az adott nyelven fejlesztési tapasztalattal
- képes legyek rajta REST API-t létrehozni

A választható nyelveket leszűkítettem háromra: Java, C# és JavaScript. Ezek a nyelvek az összes feltételnek megfeleltek, amit a backend nyelv felé támasztattam, de végül a JavaScript mellett döntöttem.

Azért döntöttem így mivel a számomra a legegyszerűbb megoldást ezen a nyelven tapasztaltam mivel is fejlesztéséhez elegendő a Node.js telepítése és egy szöveg szerkesztő és máris eltudom kezdeni a munkát. Emellett ezzel a döntésemmel egy olyan nyelvet sikerült választanom, ami nagyon rugalmas és sokkal átláthatóbb megoldást képes nyújtani, mint a másik kettő nyelv.

A JavaScriptet Brandon Eich fejlesztett 1995-ben [12]. A JavaScript annak érdekében jött létre, hogy képesek legyünk az oldalak DOM modelljét manipulálni, de a megjelenése óta mindenre is felhasználták már ami elképzelhető.

2007-ben egy amerikai programozó, Jeff Atwood még viccesnek szánta az a kijelentését, miszerint „Bármilyen alkalmazást amit meglehet írni JavaScriptben, az meglesz írva JavaScriptben”[13] ma már ez a kijelentése nem tűnik viccesnek sőt, a nevével létrejött az Atwood törvénye ami az utóbbi kijelentést foglalja magába.



Annak érdekében, hogy a böngészőn kívül is tudjuk használni a JavaScript-et, szükségünk van egy futtató környezethez, ami nem más, mint a Node.js[14] amivel képesek leszünk szerver oldali programot írni.

A Node.js 2009-ben jött létre Ryan Dahl által. Ez egy open-source, cross-platform (azaz platform függetlenül működik) futtató környezet. Ehhez tartozik egy csomag menedzser, a node package manager (az npm[15]), amire különböző felhasználók tölthetnek fel különböző kód implementálásokat amivel megkönnyíthetik ezrek de sok esetben milliók munkáját.

Az utolsó fejlesztéshez szükséges technológia pedig az Express. Az Express az egyik egy Node.js-hez használható web keretrendszer, ami web kérések fogadását és küldését könnyíti meg és add egy egységes kinézetet a programnak.

Leghasznosabb funkciói, ami miatt a használat mellett döntöttem:

- HTTP kérések kezelése
- Általános beállítások egyszerűen használhatóak (port beállítás, kód szétbontás, CORS)
- Képes "middleware" réteget létrehozni
- Adat sablonok használat
- Sok hasznos npm csomag elérhető rá
- Bármilyen probléma esetén tudok segítséget találni mivel is ez az egyik legnépszerűbb webes keretrendszer a Node.js-hez

A fejlesztés implementálásához használt eszözeim:

- A Microsoft által fejlesztett Visual Studio Code szöveg/kód szerkesztőt fogom használni a forrás állományom szerkesztéséhez. Ez a program ingyenes és rengeteg testre szabási lehetőséggel rendelkezik ezért rendes fejlesztő környezetet (IDE-t) nem fogok alkalmazni a fejlesztés alatt
- Parancssort fogom még használni, hogy el tudjam indítani az alkalmazást és ahhoz fogom még alkalmazni, hogy tudjam használni a Node.js-t és az npm-et.

Verziók:

Technológia Neve	Verzió
JavaScript	ECMAScript 2017

Node.js	18-as verzió
Express	4.18.1.
Visual Studio Code	1.67.

A backend technológiák ismertetése után az alkalmazás másik részét fogom ismertetni, a frontend-et. A frontend részre azért van szükségünk, mivel míg a backend képes kezelni az adtabázissal a kapcsolatot és az adtabázissal történő lekérdezéseket és az adtabázissal adataival történő módosításokat képes végre hajtani, a frontend része az alkalmazásnak azért szükséges, hogy a felhasználóval történő interakciókat tudjuk kezelni.

Mivel webalkalmazást csinálok ezért ehhez HTML, CSS és JavaScript technológiákat használok elsősorban, de mivel ezzel a megközelítéssel az alkalmazás rövid idő alatt menedzselhetlenné válna a sok ismétlődő kód miatt ezért mást megközelítést kell alkalmaznom.

Annak érdekében, hogy a kódom minél egységesebb és állathatóbb maradjon a jövőben ezért egy frontend keretrendszert fogok használni. Sok fajta ilyen keretrendszer létezik már, de a backendhez hasonlóan itt is leszűkítettem a keresést előfeltételekkel:

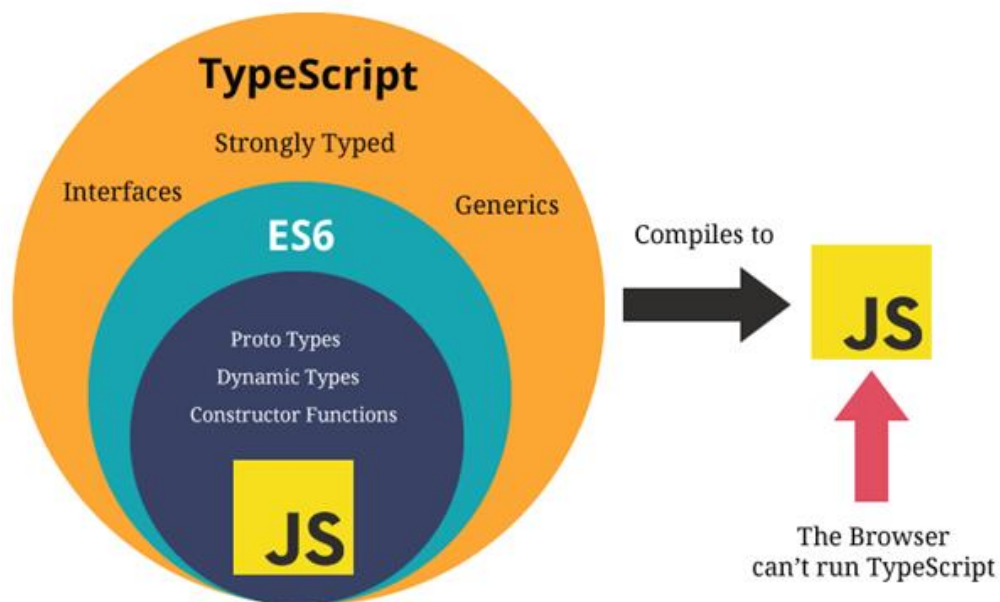
- legyen egy kész könyvtára a felhasználható megjelenítési eszközökből
- megbízható dokumentációval rendelkezzen
- tesztelhető legyen
- komponens alapú legyen a megközelítése
- képes legyen HTTP alapú kéréseket kezelni
- rendelkezzen tapasztalattal a használatával
- TypeScript lehetősége

Az összes feltételnek kettő ilyen keretrendszer felel meg: az Angular és a React. A végső döntésem végül az Angular lett mivel is egyrészt a React hivatalosan nem keretrendszer de mivel minden olyan funkcióval rendelkezik, ami ahhoz kell, hogy keretrendszerként tudjuk használni ezért szokás rá keretrendszerként is hivatkozni de hivatalosan csak egy frontend könyvtárnak

számít ami megnehezítheti a fejlesztést mivel is minden eszközt külön be kell húznunk, másrészt az Angular TypeScript alapú amivel a fejlesztés alatt rengeteg hibát tudok küszöbölni és később a tesztelés fázisánál egyszerűbb módon fogom tudni letesztelni az alkalmazásom frontend részét.

Az AngularJS 2010-ben jelent meg, az első hivatalos frontend keretrendszer között és felforgatta az akkori trendeket a webalkalmazások fejlesztésében. A fejlesztést Google végezte és az első megjelenése óta rengeteg változtatáson esett át a rendszer annak érdekében, hogy minél jobbá váljon. Később 2014-ben a Angular Version 2 megjelenésével az AngularJS nevet megváltoztatták Angular-ra. Jelen pillanatban a 13. verziójú rendszer elérhető, amit én is használni fogok az alkalmazásom fejlesztéséhez.

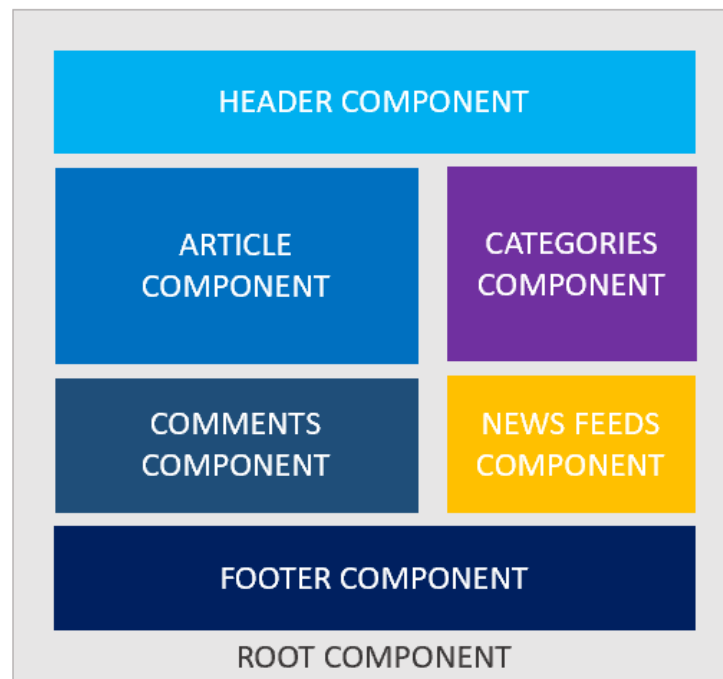
Az Angular TypeScript alapú, azaz a fejlesztés nem sima JavaScript nyelven fog megtörténni, hanem egy a JavaScript felett álló nyelv által. Ez azért hasznos mivel mielőtt a kód átalakulna JavaScript-é azelőtt átesik rengeteg ellenőrzésen ami azért hasznos mivel a JavaScript-ben nem kell típusokat megadni ezért rengeteg programozó által elkövetett hiba fordulhat elő viszont ezzel, hogy a programozó rá van kényszerítve arra, hogy bizonyos szabályokat betartson ami a tsconfig.json-ben vannak el tárolva ezért a hibák száma is drasztikusan csökkenhet.





Ezenfelül a TypeScript-ben lehet használni interfészeket, amik egyrészt a kód értelmezésében is nagy segítség tud lenni másrészt a tervezési fázisban átláthatóbb tervet tudunk készíteni a kódunkhoz de a későbbi tesztelési munkákat is ezzel megtudjuk könnyíteni úgy, hogy az interface-nek egy mock-ját használjuk majd fel a teszt kódjában.

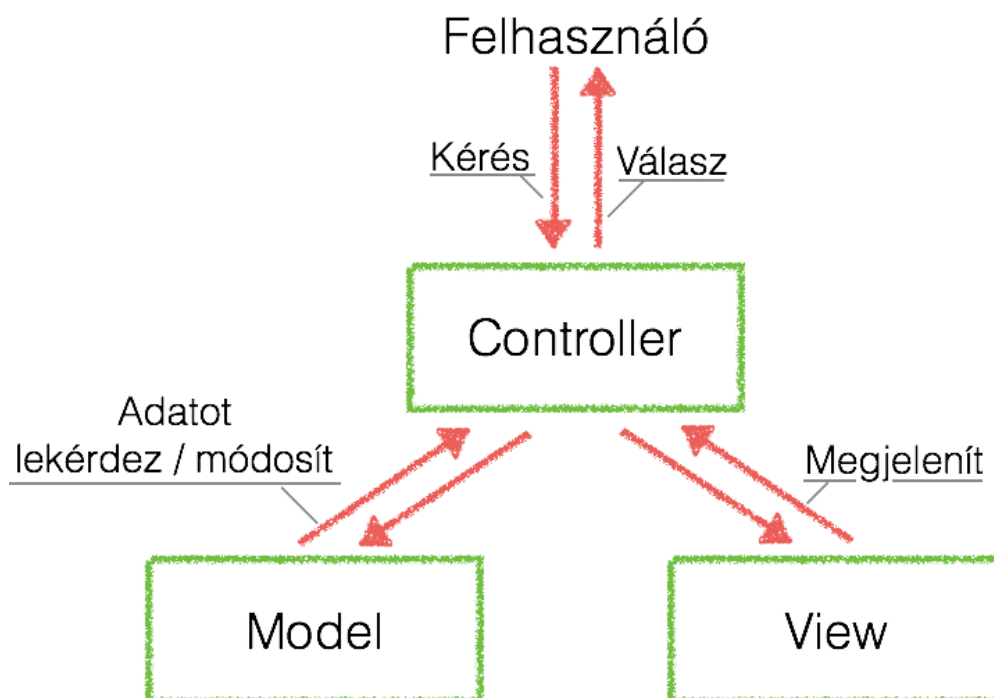
Az Angular lehetőséget add arra, hogy a különböző felületek, komponensekre lehessen szétbontani, ami azért hasznos, hogy minél kisebb részeivel tudjunk foglalkozni az adott nagy komponensnek. Ezzel a komponenseket is képesek leszünk alrendszerekként kezelni ami az alkalmazásunk működést olyan formán segíti, hogy egy adott komponenst többször is feltudjuk használni vagy egy komponenst úgy tudunk megírni, hogy a paraméterei függvényében reagáljon az adott eseményre.



A fejlesztéshez ugyan úgy a Microsoft által fejlesztett Visual Studio Code szöveg/kód szerkesztőt fogom használni a forrás állományom szerkesztéséhez, ugyan azzal a verzióval amit a backend fejlesztésénél használok. Emellett pedig itt is a parancssort fogom használni az alkalmazás indításához és a különböző csomagok letöltéséhez és telepítéséhez.

A technológiákon kívül követni fogom az MVC (Model-View-Controller) programtervezési mintát. Ez minta három részből áll és azt a célt szolgálja, hogy a frontend és backend elkülönítésen kívül a kódban is legyenek jól meghatározható réteg annak érdekében, hogy a kód logikai rész is felbontható legyen alegységekre.

- Model: Adatokat és adatszerkezetet leírást tartalmazhat. Logika nem szerepelhet benne.
- View: Az a kód rész, ami tartalmazza a felhasználó által interaktálható felület és a felületnek a viselkedésének a kódját.
- Controller (Vezérlő): A program logikai része itt szerepel, ez a része köti össze a Modell és a View részt. A View rész által generált kéréseket a Controller rész szolgálja ki a Model réteg által kapott adatokkal.



A programom lefejlesztéséhez egy verziókövető rendszert fogok alkalmazni, ami azt, a célt szolgálja, hogy a feladatom bármelyik mentési pontjára vissza tudjak térni, hogy ha bármilyen hiba történne a fejlesztés alatt.

Rengeteg fajta verzió követő rendszer létezik, de számomra a legmegfelelőbb a git. Egyrészt rengeteg ingyenes tároló oldal létezik hozzá, ami nagy segítség a fejlesztésben. Én a GitHub-ot fogom használni, mivel is ezt az oldalt használtam az eddigi munkáim során a legtöbbet.

Annak érdekében, hogy képes legyek a fejlesztési és dokumentációs folyamatok külön választani, hogy ne legyen bennük átfedés, a git-et használva branch-ekre fogom bontani a projektemet

- main branch: A programom és a dokumentáció ezen a helyen található meg együttesen, ahol is az a verzió lesz elérhető, ami bármilyen fajta bemutatásra alkalmas
- dev branch: A alkalmazás fejlesztés alatt lévő verziója itt lesz megtalálható
- documentation branch: A dokumentáció fejlesztés alatt lévő rész itt lesz megtalálható

Ha valamilyen új funkciót implementálok annak is fogok nyitni egy új branch-et amit ha befejeztem akkor össze merge-elem a logikailag felette álló branch-el ezzel pedig a programom vissza követhetőséget szeretném növelni.

Ezekon felül a GitHub által biztosított ingyenes GitHub Desktop programot fogom használni a git által elérhető funkciók használatához. Ez azért hasznos, mivel ez a program rendelkezik egy grafikus felülettel, ahol is képes vagyok pár kattintással elvégezni azokat a feladatokat elvégezni, amit parancssorból rengeteg gépeléssel tudnák csak megoldani.

## **4.4 A terv ismertetése**

- funkciók jellemzése
- szerpkör – uml - funkció
- adatbázis megjelenése

## **4.5 A megvalósítás**

## Források:

- 1: <https://dokio.hu/>
- 2: <https://www.forbes.com/sites/forbesagencycouncil/2017/03/23/the-bottom-line-why-good-ux-design-means-better-business/?sh=40a67cab2396>, <https://www.theedigital.com/blog/ux-design-website>
- 3: <https://www.macroklinika.hu/>
- 4: <https://lawsofux.com/jakobs-law/> (Book: Laws of Ux)
- 5: <https://lawsofux.com/goal-gradient-effect/> (Book: Laws of Ux)
- 6: <https://www.erzsebetfurdo.hu/>
- 7: <https://www.mongodb.com/databases/non-relational>
- 8: <https://hu.ilusionity.com/977-express-explained-with-examples-installation-routing-middleware-and-more>, <https://www.makeuseof.com/what-is-express/>
- 9: <https://angular.io/start>
- 10: <https://www.vg.hu/kozelet/2004/11/az-informatika-jelentosege-a-korhazakban-2>
- 11: <https://www.mongodb.com/company#:~:text=MongoDB%20was%20founded%20in%202007,the%20shortcomings%20of%20existing%20databases>.
- 12: [https://www.w3schools.com/js/js\\_history.asp#:~:text=JavaScript%20was%20invented%20by%20Brendan,Mozilla's%20latest%20version%20was%201.8](https://www.w3schools.com/js/js_history.asp#:~:text=JavaScript%20was%20invented%20by%20Brendan,Mozilla's%20latest%20version%20was%201.8).
- 13: <https://blog.codinghorror.com/about-me/>
- 14: [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction)
- 15: <https://www.npmjs.com/>

Doom kép - <https://ipon.hu/magazin/cikk/doom-atirat-html5-es-javascript-alapokon>  
ts-js kép - <https://www.linkedin.com/pulse/typescript-superset-javascript-sunil-sharma/>  
angular components kép - <https://www.dotnettricks.com/learn/angular/components>  
mvc modell kép - <https://www.letscode.hu/2016/05/02/tiszta-kod-6-resz-beszelnunk-kell-az-mvc-rol/>