

# **TDK Dolgozat**

Készítette:  
Dávid Szabolcs

**Szegedi Tudományegyetem**

**Informatikai Intézet**

**Műszakbeosztás optimalizálása AI-vezérelt  
interakcióval**

TDK Dolgozat

*Készítette:*

**Dávid Szabolcs**

programtervező informatikus

MSC hallgató

2. évfolyam

*Témavezető:*

**Czégel András**

Optimization expert

Lufthansa Systems Hungária

*Belső konzulens:*

**Dr. Gazdag-Tóth Boglárka**

Tudományos főmunkatárs

Szeged

2024

# Tartalomjegyzék

Absztrakt . . . . .	5
<b>1. Bevezetés</b>	<b>6</b>
<b>2. Megoldásmód</b>	<b>8</b>
<b>3. Az alkalmazásról</b>	<b>9</b>
3.1. Felhasználói felület . . . . .	9
3.1.1. Felhasználói nézet . . . . .	9
3.1.2. Adminisztrátori nézet . . . . .	13
3.1.3. Chat ablak . . . . .	14
3.2. Tech stack . . . . .	14
3.3. Frontend . . . . .	15
3.4. Backend . . . . .	15
3.4.1. Adatmodellek . . . . .	15
<b>4. Chat-asszisztens</b>	<b>19</b>
4.1. ChatGPT . . . . .	19
4.2. OpenAI és LangChain . . . . .	20
4.2.1. Promptok . . . . .	20
4.2.2. Eszközök . . . . .	21
4.2.3. AgentExecutor . . . . .	22
4.2.4. Memória . . . . .	23
4.3. Kérések az asszisztens felé . . . . .	23
4.3.1. Felhasználói folyamatok . . . . .	24
4.3.2. Adminisztrátori folyamatok . . . . .	29
<b>5. Tervező motor</b>	<b>31</b>
5.1. Problémaleírás az új műszakok generálásához . . . . .	31
5.1.1. Matematikai modell . . . . .	31
5.1.2. Célfüggvény és korlátozó feltételek . . . . .	33
5.2. Problémaleírás újra tervezés esetén betegség követően . . . . .	36
5.2.1. Matematikai modell . . . . .	37
5.2.2. Célfüggvény és korlátozó feltételek . . . . .	38
5.3. Megoldó . . . . .	41

5.3.1. OR-Tools . . . . .	41
5.3.2. SCIP . . . . .	42
5.3.3. Modell implementálása . . . . .	42
5.3.4. Futásidő és eredmények . . . . .	43
<b>6. Alkalmazás használata</b>	<b>46</b>
<b>7. Kitekintés</b>	<b>47</b>

# Absztrakt

A műszakok tervezése és kiosztása az alkalmazottak részére mindennapos probléma a cégek számára a mai világban. Kis létszám esetén kézzel egyszerűen megoldható, de nagyobb mennyiségű munkavállaló esetén már nehéz feladat elé állnak azok, akik a műszakok megfelelő elkészítéséért felelnek. Dolgozatomban egy alkalmazás implementálása volt a cél, ahol a felhasználók igényeket adhatnak le a hét napjainak különböző műszakjaira, melyet egy matematikai modell különböző, előre meghatározott szabályok alapján megold, majd hozzárendeli a felhasználókat a műszakokhoz ügyelve arra, hogy minél több legyen ezek közül előzetes igény alapján kiosztva. Ezeket az igényeket az alkalmazottak egy chat asszisztens segítségével tudják leadni, melynek másik oldalán a matematikai modell található, a kettő között pedig egy LLM helyezkedik el, így nincs másra szükség a folyamatban. Ez fogadja a bejövő kéréseket, azokat csoportosítja és feldolgozza, majd a megfelelő műveleteket végzi el a tökéletes végrehajtás érdekében. Eredményül emberi nyelven válaszol a felhasználónak a művelet sikerességéről, esetleges hibáiról, vagy csak tájékoztatást ad, a matematikai modellt pedig felparaméterezi, mely később ezek igényekből kinyert adatok alapján létrejött paraméterek mentén optimalizálja a beosztást. Az LLM minden kérést fogad, de kizárólag csak a kontextushoz tartozó kérdésre válaszol. A felhasználók minden héten, a két hétre esedékes műszakokra tudnak jelentkezni, vagy szabadságigényt leadni. A matematikai modell ezek alapján kiosztja a munka, szabad és tartalékos napokat és a szabadságokat – amennyiben még van felhasználható napjuk. A szabadnapokra, különböző műszakokra szabályok sokasága van definiálva a gördülékeny és szabályos munkavégzés érdekében. Betegség esetén azonnal cselekedni kell, a modell először a tartalékos kollégákat próbálja beosztani, ha pedig erre nincs lehetőség, úgy újra generálja a hét azon szakaszát a beosztásnak, figyelve a szabályok meg nem sértésére, próbálva minimális módon eltérni az eddigi beosztásoktól.

# 1. fejezet

## Bevezetés

Az optimalizálás az emberek hétköznapijainak szerves része, pedig sok esetben tudomásunk sincs róla. Csak bele kell gondolni, hogy egy átlagos szombat délelőtti várás során is optimalizálás történik, hiszen arra törekszünk, hogy a lehető legjobb minőségű és mennyiségű terméket tegyük a kosarunkba, majd a lehető legkevesebbet fizessünk a kasszánál. Hasonló módon, amikor elindulunk otthonról a bevásárlóközpontba, szeretnénk a legrövidebb utat megtenni addig, mivel a hosszabb út több időt, energiát és üzemanyagot igényel, mely hamarabbi tankoláshoz vezet, ami pénzbe kerül. Ezzel megint csak, tudva-tudatlanul, de optimalizálás veszi helyét. Példák sokaságát lehetne felhozni és észrevehető, hogy nem csak a hétköznapi emberek próbálják megtalálni a legjobb, vagy legolcsóbb megoldást, de az állatvilágban is megjelenik ez a viselkedés. A költöző madarak például a legrövidebb utat igyekeznek megtalálni két pont között, ezzel is energiát takarítva meg.

Nyilvánvaló, hogy átlagemberként nem gondoljuk túl, csináljuk a dolgokat minden esetben tudatosan, hiszen nincs akkora hasznunk egy-egy tevékenység hatékonyabb végrehajtásában. Természetesen - visszatérve a vásárlás példára - , ha jobban odafigyelünk, hogy mit teszünk a kosarunkba, milyen kiszerezésben, akció keretein belül, több pénz marad a zsebünkben, mint ha csak véletlenszerűen választanánk azonos termékek különböző márkái között.

Míg egy bevásárlásnál pár száz, ezer forintot tudunk megspórolni, addig mások, a gazdaság nagyobb résztvevői jóval nagyobb haszonra tudnak szert tenni, ha optimálisan végzik tevékenységüket. Tegyük fel, hogy létrehoztunk egy telekommunikációs szolgáltatót, mely egy képzeletbeli városban nyújt televízió, telefon és internet szolgáltatást a város lakóinak. A társaságunk bevétele több részből tevődik össze. Egyik a havidíj, melyet a felhasználók fizetnek a szolgáltatás igénybevételéért, a jelentősebb kiadások pedig az infrastruktúra fenntartása, és a dolgozók bére. Jelen esetben foglalkozunk csak utóbbival, azon belül a központ munkatársaival, ahova az ügyfelek panaszokkal, kérésekkel fordulnak telefonos úton. Mivel fontos számunkra az elégedettség, azt szeretnénk, hogy ha valaki betelefonál, a lehető legkevesebbet kelljen várnia, amíg nem kapcsolják egy munkatársunkhoz és azonnal tudja intézni a dolgát. Ehhez arra van szükség, hogy mindig legyen szabad operátor. Viszont ha túl sok alkalmazott dolgozik, az a cég költségeire negatívan fog hatni, mivel mindenkit ki kell fizetni és lehetnek időszakok, amikor üres

járat van. Ennek elkerülése érdekében szeretnénk valahogy azt megoldani, hogy mindig pontosan (vagy legalább megközelítőleg) annyian dolgozzanak, hogy az ügyfelek elégedettsége elérjen egy képzeletbeli szintet a várakozási időre való tekintettel, azaz minden hívást a lehető leggyorsabban meg tudjanak válaszolni. Viszont azt sem szeretnénk, hogy ez a létszám sokkal magasabb legyen, mivel abban az esetben több alkalmazottat kell foglalkoztatni egy időpillanatban, mint amennyire valójában szükség lenne, ezzel pedig megnő a kiadásunk.

Nagyszerű, meg is van az optimalizálandó probléma. A műszakok kezdetét és hosszát, illetve az azokat ellátó munkatársak számát múltbéli tapasztalatok alapján határozzuk meg. Látjuk, hogy egy ügyfélbázis méretéhez függően mely napszakban körülbelül hány hívásra számíthatunk és tudjuk, hogy az ügyfelek átlagosan mennyi időt töltenek a vonalban. Ha ezekkel mind megvagyunk, nincs más dolgunk, mint elkészíteni a beosztást. Ha nagyon pontosan foglalkozunk a dologgal, akkor órára lebontva határozzuk meg az alkalmazottak számát, példánkban viszont a napot csak három műszakra osztjuk. Ez a gyakorlatban is gyakrabban fordul elő. Reggeli, délutáni és esti. A reggeli 8-16, a délelőtti 10-18, az esti pedig 14-22 óra között fogja foglalkoztatni az alkalmazottjainkat. Ennek elkészítésére több módszer is lehet. Megtehetjük akár véletlenszerűen is, minden műszakra beosztunk valakit és később foglalkozunk az esetleges következményekkel. Ez természetesen gondokat okozhat, hiszen lehet, hogy valaki a hét összes napján dolgozna, és lehet, hogy valaki egy napot sem kap. Probléma adódhat abból is, ha valaki egy esti műszak után másnap reggelre kap munkát. Minden ország meghatároz magának egy munkatörvényt, melyet a munkáltatónak be kell tartani, aminek a megszegése komoly büntetéseket vonhat maga után. A törvénykönyvben szereplő törvények számtalan szabályt fogalmaznak meg, legyen ez napi, vagy heti munkaidőre vonatkoztatva, szünetek beiktatására, vagy túlórára való behívásra. Az alkalmazásban szereplő szabályok nem lesznek mélyebben tárgyalva és implementálva, egy általánosabb szabályhalmaz lesz használva.

Tegyük fel, hogy sikerült definiálni a beosztásnak a felépítését és szabályait, az viszont még hiányzik, hogy ezek a műszakok hogyan lesznek kiosztva. A már említett véletlenszerű beosztás működhet, viszont a vezetőség szeretné betartani a törvényeket és megfelelni a szerződéseknak. Ehhez az informatikus kollégánk elkészített egy alkalmazást, melyben a dolgozók minden hétre leadhatják igényüket, mely napok mely műszakjaiban szeretnének dolgozni és az igények alapján szeretnénk ezt optimalizálni - a maximális elégedettség és a törvények betartása mellett. Erre egy matematikai modellt használunk, a felhasználók pedig egy LLM [9], azaz Large Language Model (Nagy Nyelvi Model) segítségével kommunikálnak, "beszélgetnek" a műszakjaikat tervező alkalmazással, hasonlóan a ChatGPT-hez [16] [13]. Az adminisztrátor - nevezzük így, aki a felelős az alkalmazottak munkájáért és beosztásáért - pedig hasonló módon, a felületen az LLM-el kommunikálva képes a matematikai modell segítségével optimalizálni a beosztást, hogy az minél jobban kielégítse a felhasználók igényeit, mégis a lehető legkevesebb embert használva műszakonként, a profitmaximalizálás érdekében.

TDK dolgozatom célja egy ilyen alkalmazás elkészítése, mely segíti a munkáltatókat, hogy megteremtsék a legkedvezőbb beosztást, ami számukra és a dolgozók számára is megfelelő, a lehető legkényelmesebb és leggyorsabb módon.

## 2. fejezet

# Megoldásmód

Az alkalmazás két fő komponensből épül fel.

A *frontend*, amit a felhasználók látnak és használnak, azaz a *user interface*, amin megtekinthetik a jelenlegi, múltbéli beosztásaikat heti lebontásban, valamint a következő hétre vonatkozó beosztásigényüket. Mellette megtalálható egy chatablak, ahol kéréseket tudnak megfogalmazni egy GPT-alapú [30] modell felé. Ez a modell fogja a fogadott kéréseket - legyenek ezek igények műszakokra, szabadságtervezés - elemezni, majd a megfelelő módon lekezelné.

A másik része az alkalmazásnak a *backend*, ahol egyidőben több dolog is történik. Az adatbázis műveleteket megelőzi egy köztes réteg, ahol a GPT modell helyezkedik el. Minden egyes üzenet, mely a *UI*-ről jön, legyen az egy alkalmazott, vagy adminisztrátori szerepben lévő felhasználó által megfogalmazott kérdés, ezen a feketedobozon megy keresztül. Azt, hogy mi lesz a kimenete, amit a modell válaszol az üzenet küldőjének, és milyen hatása lesz, az nagyjában függ a megfogalmazott kéréstől. A modell képes megkülönböztetni különböző eszközök segítségével, hogy milyen szerepkörben lévő felhasználótól kapta az utasítást, ez az utasítás mire vonatkozik, milyen lépéseket kell tennie ahhoz, hogy a lehető legkisebb hibalehetőséggel teljesítse a kapott kérést, és pontos választ adjon vissza.

Mihelyt minden felhasználó kialakította saját beosztásigényét a következő hétre, úgy az adminisztrátornak lehetősége van beleszólni a végső műszakok kiosztásába. Ezt a háttérben egy matematikai modell teszi meg, előre meghatározott szabályok és feltételek mentén. Mivel számos tényező miatt lehet hogy a modell nem megoldható, az adminisztrátoroknak lehetőségük van átalakítani az igényeket. Ilyenkor szabadságokra leadott kéréseket tudnak kitörölni, vagy áthelyezni ha szükséges, de a modell korlátozó feltételein is képesek változtatni. Ezt a folyamatot lehet, hogy ismételni kell mindaddig, amíg nem sikerült a szabályoknak megfelelő beosztást létrehozni. Amint a felhasználók igénye elér egy olyan formát, melyet a modell meg tud oldani, úgy a kiadott utasítást követően létrejön a következő beosztás, mely a jelenlegi héttől két hét múlva esedékes. Ezt a műveletet minden héten szombat-vasárnap kell elvégezniük az adminisztrátoroknak, annak érdekében hogy időben rá tudjanak készülni a dolgozók.



## 3. fejezet

# Az alkalmazásról

### 3.1. Felhasználói felület

Az alkalmazás nyitólapján egy egyszerű belépőrendszerrel találkozik a felhasználó. Amikor új munkatárs érkezik, úgy őt az adminisztrátorok képesek regisztrálni. Ekkor egy felhasználónevet kap, mely a vezeték és keresztnévének első három betűje összefűzve, és inicializálja az adatbázisban a megfelelő értékeket a későbbi használatra.

Belépést követően két csoportot különböztetünk meg a felhasználói jogosultságok szerint. Alkalmazottak és adminisztrátorok. A telefonközpontos példában az alkalmazottak lesznek az operátorok, akikhez a hívások beérkeznek, és a beosztásukat szeretnék a lehető legjobb módon kialakítani. Az adminisztrátorok az alkalmazottak felettesei, akik értük felelnek. Ők fogják a végső szót rámondani a leadott beosztásigényekre, melyből később a kész, műszakokra lebontott beosztás fog születni. Az adminisztrátorok egy *Supervisor* nevű csoporttal vannak megkülönböztetve az alkalmazáson belül, erről később lesz még szó.

#### 3.1.1. Felhasználói nézet

A felhasználó felület hasonlóan néz ki mind a két csoport tagjainak számára. A baloldalon egy panel látható.

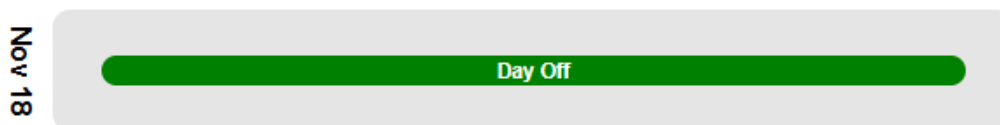
Ez, alkalmazottak esetén ketté van bontva, és két heti *roster* helyezkedik el benne. *Roster*-nek hívjuk egy felhasználó egy heti beosztását, hétfőtől vasárnapig, melyen a műszakok (shift) szerepelnek. A baloldali *roster* mindig az adminisztrátorok által gondosan ellenőrzött, majd a matematikai modell által megoldott, adott felhasználóra vonatkozó beosztás. A jobboldalon a leadott igények láthatóak. A két *roster* felett egy-egy legördülő listán a felhasználó ki tudja választani, hogy mely hetet szeretné megtekinteni, mind a múltbéli és jelenlegi heti beosztáshoz, mind pedig a múltbéli, jelen heti, jövőheti és azt követő heti beosztásigény megtekintéséhez.

Adott időpontban mindig a két hét múlva esedékes hétre tudják leadni az igényüket, ezzel biztosítva, hogy az alkalmazottaknak a legenerált és kiosztott műszakok után legyen idejük alkalmazkodni és tervezni. Ezt egy paraméter segítségével az adminisztrátorok be tudják állítani, hogy hány héttel előre lehessen műszakigényt leadni.

A heti *roster* napi felbontásban van megjelenítve. Egyes napokon a következő jelzések láthatóak.

### Szabadnap

*Day Off*, amennyiben szabadnapja van a felhasználónak, nincs beosztva.



3.1. ábra. Szabadnap.

### Tartalékos nap

*Reserve*, azaz tartalékos abban az esetben, amikor nincs előre beosztva, viszont arra az időszakra bármikor behívhatják, így készenlétben kell lennie.

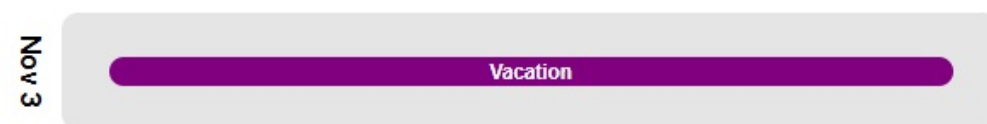


3.2. ábra. Tartalékos nap.

Ez a valóságban is jelenlévő dolog, például a légitársaságokban. Ilyenkor a pilótáknak a városban kell tartózkodniuk, hogy adott időn belül ki tudjanak érni a reptérre, bármilyen nemkívánt esemény következtében.

### Szabadság

*Vacation* felirat látható, ha szabadság igény, vagy már elfogadott pihenőidőre lett beosztva valaki.



3.3. ábra. Szabadság.

### Betegszabadság

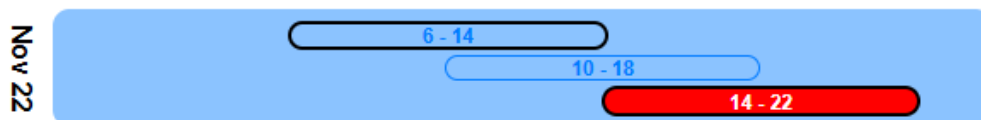
*Sickness* szöveg jelenik meg, ha a felhasználó betegszabadságon tartózkodik.



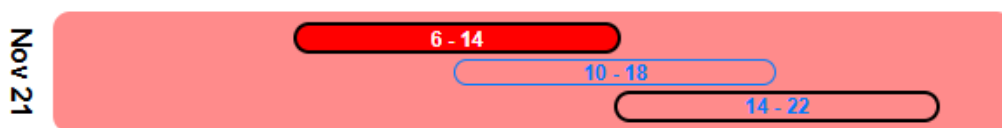
3.4. ábra. Betegszabadság.

### Tartalékból való behívás

Amikor egy felhasználó lebetegszik, úgy az ő műszakját pótolni kell valakivel. Ilyenkor először a tartalékos munkatársak közül választ valakit a modell, ha pedig nincs szabad tartalékos már aznapra, úgy szabadnapból fog behívni. A tartalékos műszakról való munkavégzést halvány kék, szabadnapból pedig halvány piros háttér jelzi.



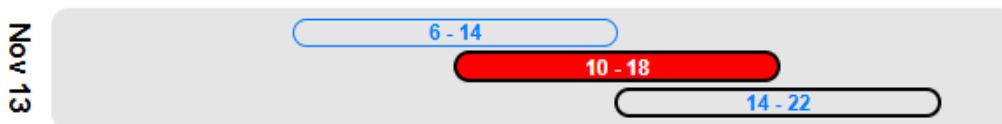
3.5. ábra. Tartalékos műszakból történő behívás.



3.6. ábra. Szabadnapból történő behívás.

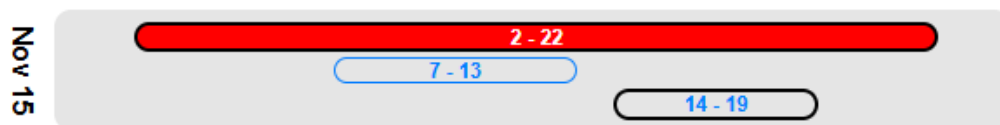
### Munkanap

Ellenkező esetben pedig beosztott állapotban van a felhasználó. Minden nap három különböző műszak található.



3.7. ábra. Munkanap.

Azt, hogy egy adott napon hány műszak van, a műszakok mettől meddig tartanak, az adminisztrátorok paraméterek beállításával tudják meghatározni. Jelen esetben adott napon három: reggeli, délutáni és esti műszak van megkülönböztetve, melyek 8-16, 10-18 és 14-22 óra közötti munkavégzést jelentenek. A műszakokat jelző "bucik" időarányosan jelennek meg. A külső komponens mely tartalmazza azokat reprezentálja a 24 órát, mint teljes napot, az elhelyezkedésük pedig arányos a kezdeti, végidőpontjukkal és az időintervallum hosszával. Egy 2-22, 7-13, 14-19-es konfiguráció így néz ki:



3.8. ábra. Különböző intervallumok.

A felhasználói felületen mindig az a műszak lesz piros színnel kitöltve, melyhez az alkalmazott be lett osztva a három közül adott napon, vagy igényt adott le, ha az igényeket tekinti meg. A többi üresen fog látszódni. A kész beosztásokban azok a műszakok erősebb fekete körvonallal jelennek meg, melyekre igény volt leadva a felhasználó által, azaz ezt megelőzőleg szeretne volna, hogy abba az időpontba osszák be. Mivel egy napon több műszakra is adhatnak le igényt, így mind a három műszak megjelenhet fekete körvonallal, de kizárólag egy lehet pirossal kitöltve - egy napon maximum egy műszakban lehet dolgozni. A felhasználói nézeten ezek mellett kék körvonallal van megjelenítve a jelenlegi napot jelző panel.

The screenshot displays the user roster panel with the following components:

- Calendar View:** Shows a schedule for Nov 11 - Nov 17 and an application for Nov 25 - Dec 1. Shifts are represented by colored bars (red, blue, black) indicating different shift types.
- Chat Window:** Contains messages from the user and the system.
  - User message (20:01): "I want to apply for every shift except for monday night and thursday all day"
  - System message (20:02): "You have applied for: Monday morning and afternoon, Tuesday morning, afternoon, and night, Wednesday morning, afternoon, and night, Friday morning, afternoon, and night, Saturday morning, afternoon, and night, Sunday morning, afternoon, and night, and cancelled: Monday night. With that, your ongoing applications are Monday morning and afternoon, Tuesday morning, afternoon, and night, Wednesday morning, afternoon, and night, Friday morning, afternoon, and night, Saturday morning, afternoon, and night, Sunday morning, afternoon, and night, and cancellations are Monday night, Thursday morning, afternoon, and night. Would you like to modify it any further, or save them?"
  - User response (20:02): "let's save them"
  - System message (20:02): "There were no ongoing modifications to save. If you need to make any changes or have any questions, feel free to ask!"
- Buttons:** "Submit request" and "Logout" are visible at the bottom of the interface.

3.9. ábra. Felhasználói roster panel.

### 3.1.2. Adminisztrátori nézet

Adminisztrátori jogosultsággal rendelkező felhasználók számára a kinézet hasonló, ám ők nem a saját, hanem az összes beosztottjuk *roster*-ét látják heti lebontásban. Hasonlóan a felhasználókhöz, ők is, az egyik legördülő listából tudják kiválasztani, mely hétre szeretnék megjeleníteni a műszakokat. A másiktól pedig azt, hogy a jelentkezéseket, vagy már a generált műszakokat szeretnék látni.

Ebben a nézetben egymás alatt vannak a felhasználók megjelenítve, nekik pedig vízszintesen helyezkedik el a beosztásuk.

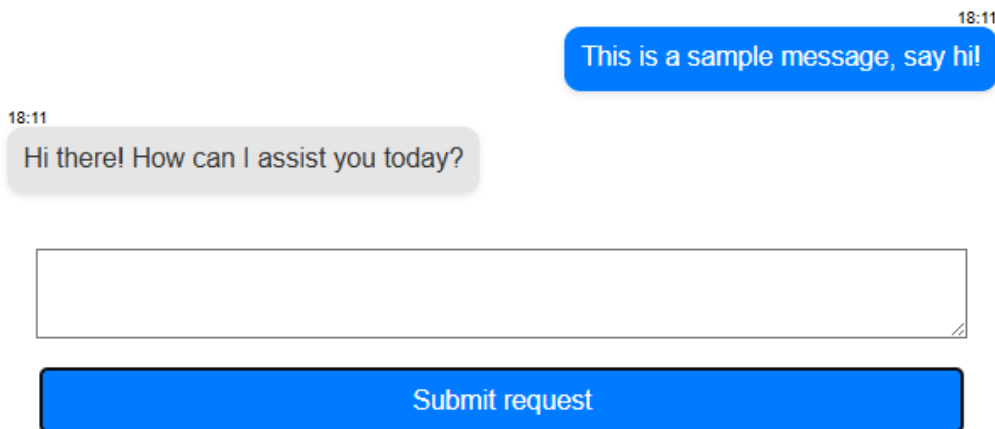
A tervező nézetben a leadott igények láthatóak, a generált beosztásnál pedig már a pihenőnapok, tartalékos műszakok és kivett szabadságok, betegségek. Szintén fekete körvonallal vannak feltüntetve azok a műszakok, melyekre a felhasználók igényt adtak le. Ezzel könnyedén figyelemmel tudják követni, hogy ki az, aki a számára kedvező műszakot kapta és ki az aki nem.

Username	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
jamsmi	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	6-14 10-18 14-22
johjoh	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	Reserve	Day Off
robwil	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22
micbro	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	Reserve
davmil	6-14 10-18 14-22	Reserve	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	Day Off
ricmar	Reserve	Day Off	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22
marand	Day Off	Sickness	Sickness	Sickness	Sickness	Sickness	Day Off
pattay	Day Off	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22
lintho	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	Day Off	6-14 10-18 14-22
elimo	6-14 10-18 14-22	6-14 10-18 14-22	6-14 10-18 14-22	Day Off	6-14 10-18 14-22	Day Off	6-14 10-18 14-22

3.10. ábra. Egy teljes, generált beosztás.

### 3.1.3. Chat ablak

Mindkét csoport számára a felület jobboldalán foglal helyet a chat komponens, mely egy szövegdobozból és egy szövegmezőből áll, ahol az eddigi üzenetek jelennek meg dátummal ellátva. Jobb oldalra igazítva a saját, felhasználó által küldött, bal oldalra pedig a chat asszisztens által válaszként érkezett válaszok jelennek meg.



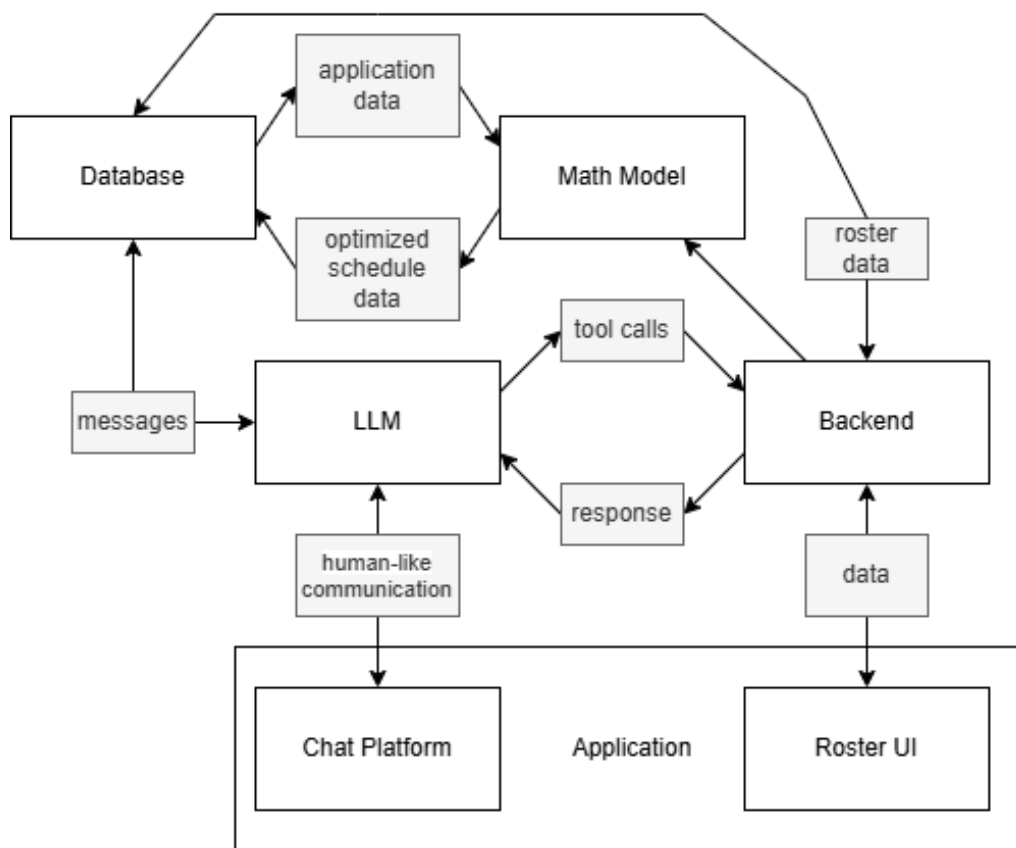
3.11. ábra. Chat ablak.

## 3.2. Tech stack

Az alkalmazás frontend része a React nevű keretrendszer segítségével készült el, mely korunk egyik legnépszerűbb JavaScript könyvtára felhasználói felületek építésére a Facebook által. Egyik nagy előnye a komponens-alapú architektúra, mely segítségével a rendszert apró, újrahasznosítható komponensekre bontva, majd azokat felhasználva van lehetőség az alkalmazást felépíteni. A backend Djangoval lett megalkotva, Python nyelven. A választás a Django egyszerűsége miatt történt, beépített adatbázisával és REST műveletivel gyors és könnyed módon lehet az alkalmazást felépíteni köré, majd hozzákapcsolni a frontendhez.

A két réteg között helyezkedik el a GPT réteg, mely a felhasználói felületről érkező emberi szöveget alakítja át a backend számára olyan formává, melyet az ott lévő függvények és eszközök kezelni tudnak. Ezt követően azok eredményét visszaküldve, ismét a GPT rétegen keresztül emberi nyelvi formára visszaalakítva kerül vissza a felhasználóhoz. Ezek mellett jelen van egy matematikai modell, mely teljesen független része az alkalmazásnak, ami a beosztásokat automatikusan generálja le. Ezt az adminisztrátor felhasználók tudják "elindítani". A modell az adatbázisból kinyert adatok alapján dolgozik, és kimenetelet, ismét a GPT rétegen keresztül közli, ad információt annak futásának sikerességéről vagy sikertelenségéről, majd sikeres futás esetén módosítja az adatbázis megfelelő elemeit.

Az alkalmazás különböző részeinek kapcsolatát a 3.12 ábra szemlélteti.



3.12. ábra. Az alkalmazás komponensei.

### 3.3. Frontend

A felhasználó felület számos komponensből épül fel, melyeket a React segítségével egyszerűen lehet használni, akár többször is. Ezek fő elemei a *roster* panel, display komponensek, ahol az adatbázisból, REST API-n keresztül lekért adatok kerülnek megjelenítésre.

### 3.4. Backend

Amíg a felhasználó csak a "jéghegy" csúcsát látja, minden művelet a háttérben, a backenden történik. Minden adat, logika itt helyezkedik el, mely az alkalmazás esetében láthatatlan a felhasználók számára.

#### 3.4.1. Adatmodellek

Az egyik legfontosabb dolog a megfelelő adattípusok definiálása. Ezek lesznek azok az objektumok majd, melyek az információt hordozzák. Az alkalmazás két különböző elemet különböztet meg, mellyel műveleteket végez, és megjeleníti a felhasználói felületen.

*Roster*-ek és üzenetek. A felhasználók kezelése a Django beépített *User* modelljével van megvalósítva. A két modell a következőképpen épül fel.

### Roster

A *Roster* objektumban vannak eltárolva egy adott hétre vonatkozó adatok egy felhasználóhoz rendelve.

```
Roster: {
    week_number: int
    year: int
    application: str
    schedule: str
    modification: str
    work_days: str
    off_days: str
    reserve_days: str
    reserve_call_in_days: str
    day_off_call_in_days: str
    vacation: str
    sickness: str
    published: bool
    reserve_call_in: bool
    day_off_call_in: bool
    owner: User
}
```

3.13. ábra. Roster objektum adattagjai.

A *week\_number* és *year* mezők magukért beszélnek, azt jelzik, mely év mely hetére vonatkozik a *roster*. Az utolsó, *owner* mező segítségével vannak a felhasználóhoz hozzákapcsolva 1-1 kapcsolatban, azaz minden felhasználóhoz pontosan egy darab *roster* tartozik adott év adott hetén.

Az *application*, *schedule*, *modification* mezők 21-21, a *work\_days*, *off\_days*, *reserve\_days*, *vacation* és *sickness* pedig 7-7 karakterből álló sztringek. Ezek minden esetben kizárólag 0 és 1 értékekkel vannak feltöltve. A napokra vonatkozó mezők a hét napja in jelzik a munkavégzést, szabadnapot, tartalékos műszakot, vakációt és a betegséget. Amennyiben adott nap indexén 1-es érték szerepel, úgy azon a napon a felhasználónak dolgoznia kell, szabadságon van, stb. Így, egy heti *roster* vakáció nélkül a következőképpen épülhet fel.

Ebben az esetben hétfőtől csütörtökig be van osztva műszakokra, pénteken tartalékosként, a hétvégén pedig szabad a felhasználó. Fontos megjegyezni, hogy adott *roster*-en a négy mező azonos indexein pontosan egyszer szerepelhet 1-es karakter. Azaz, ha a sztringek karaktereit indexenként összeadjuk, minden összegnek pontosan 1-nek kell lennie. A napokra vonatkozó különféle szabályok miatt változhat, hogy egy héten hány napot



```
{
  work_days: 1111000,
  off_days: 0000011,
  reserve_days: 0000100,
  vacation: 0000000,
  sickness: 0000000,
}
```

3.14. ábra. Napokra vonatkozó változók.

kell dolgozni, lehet szabad vagy tartalékos, ez nagyban a szabadságok számától függ, de befolyásolhatják esetleges nem várt események, vészhelyzetek.

Két mező még, mely hasonlóan a fentiekhez 7 hosszú sztringeket tárolnak, a *reserve\_call\_in\_days* és *day\_off\_call\_in\_days*. Ezek reprezentálják a tartalékos műszakból, rosszabb esetben a szabadnapból való behívást, mely más felhasználók betegsége esetén történhet meg. A hozzájuk kapcsolódó *reserve\_call\_in* és *day\_off\_call\_in* logikai mezők jelzik, ha adott héten történt behívás.

Az *application* mező minden hétre a leadott igényeket tárolja, a *schedule* az éles beosztást. Mivel minden nap három műszakra tudnak jelentkezni, így 7x3 hosszú lesz ez a bináris sztring. Hasonló módon, itt is 1-es karakter jelzi adott indexen az adott műszakra vonatkozó igényt, vagy beosztást. Hármásával egy napot tesznek ki, ezért amíg az *application* esetében lehet több 1-es is a hármásokban, úgy a *schedule* mezőben már nem, mivel egy napon kizárólag egy műszakban dolgozhatnak a felhasználók. Az előbbi példát folytatva, amennyiben az *application* és *schedule* mezők így néznek ki:

```
{
  application: 101110010101111000000,
  schedule: 1001000100010000000000,
}
```

3.15. ábra. Műszakokra vonatkozó változók.

Ez azt jelenti, hogy a felhasználó a következő műszakokra jelentkezett:

- hétfő reggeli és esti,
- kedd reggeli és délelőtti,
- szerda délelőtti,
- csütörtök reggeli és esti,
- pénteken az összes,
- hétvégén pedig egyik műszakra sem

A beosztása pedig a következő lett:

- hétfő reggel,
- kedd reggel,

- szerda délelőtt,
- csütörtök este

A többi nap mind nullákkal van feltöltve, mivel pénteken tartalékos, szombaton és vasárnap pedig szabadnapos.

Az *published* mező minden esetben hamis értéket tárol, amíg nem lett legenerálva adott hétre a műszak, ezt követően minden *roster*-hez igaz értéket ad neki az alkalmazás, ennek a frontenden való megjelenítésben van szerepe.

Az utolsó, *modification* mező a felhasználók módosításait menti el, majd használja fel, ha végeztek azzal és szeretnék véglegesíteni a változtatásokat. Ez, ahogy majd látjuk, folyamatosan tárolja a különféle kéréseket, de nem írja felül az *application* változót mindaddig, amíg nem szeretné végleg menteni a felhasználó azt. Ezzel lehetősége van, hogy akár napokon át "gondolkodhasson" és módosíthassa azt, hogyan képzei el a következő heti munkáját - bár nincs különbség aközött, ha minden egyes művelet után elmenti.

### Message

A másik, kisebb modell az üzenetek tárolására szolgál. Felépítése a következő:

```
Message: {
  text: Char,
  date: Date,
  sent_by_user: Boolean,
  owner: User
}
```

3.16. ábra. Message objektum adattagjai.

A *text* mezőben van eltárolva az üzenet tartalma, a *date* mező az elküldés időpontját jelzi, mely megjelenítésre kerül a chat ablakban, a logikai változó pedig azt indikálja, hogy felhasználótól, vagy az asszisztentől érkezett az üzenet. Ez a nyelvi modellhez szükséges, hiszen a későbbiekben szeretnénk megkülönböztetni emberi és gépi üzenetet, a modell tanulása érdekében.

## 4. fejezet

# Chat-asszisztens

Számtalan cég a mai napig kézzel készíti a beosztást, mely időigényes, és több munkaerőre van szükség hozzá. Kevés alkalmazott esetén könnyű dolog elé néznie, de a létszám növekedésével ez a feladat igen nagyra nőheti ki magát, és ekkor még nem biztos, hogy a megoldás optimális lesz. Vannak helyek, ahol egy hatalmas fali órarendbe tudják beírni magukat a dolgozók, és adok-kapok módon az fogja elvinni a műszakot, aki hamarabb jelentkezik rá. Ezzel persze el lehet kerülni a beosztás tervezését. Van, ahol a hasonló módon leadott igényekhez még ezt követően kell kialakítani a végleges műszak-dolgozó párosokat. Sok helyen online felületeket használnak, ahol a preferenciák kis szövegdobozokba vannak leadva, majd tovább küldve, és ebből kell kibogozni a pontos igényt, majd ezek alapján megépíteni a következő hetet. Ilyenkor bonyolultságot okozhat, mikor több tíz, akár száz kollégának kell feldolgoznia a kérését, melyek gyakran *"h-k-sz-p-sz 8-16 bármikor"* formában is bekerülhetnek a rendszerbe. Az okosabb rendszerek esetén már külön műszakokra lehet kattintgatni, akár időintervallumot megadni, amit már könnyebb használni, és később valamilyen módszer segítségével kész beosztást generálni belőle.

### 4.1. ChatGPT

Az elmúlt évek egyik leggyorsabban fejlődő AI asszisztense kétség nélkül a ChatGPT [16] [13] volt. A 2022-ben elstartoló alkalmazás rettentően gyorsan kinőtte magát, és az emberek hétköznapijainak a része lett, sok esetben kiváltva a már évtizedek óta jól működő keresőmotorokat, mint például a Google. Fejlődésének hatalmaságát a különböző verziói, és azok folyamatosan növekvő tudása is jól mutatja. Az első modellt, a GPT-1-et [6] [2] 2018-ban mutatták be, mely az első volt a GPT (generative pre-trained transformer) [30] szériából, 117 millió paraméterrel. Számtalan könyvvel és szöveggel tanították be felügyelet nélküli tanítás segítségével, mellyel szövegfelismerő feladatokra használták. Viszonyításképp, egy egyszerű képfelismerő neuronháléhoz körülbelül ezredannyi paraméterre van szükség, és ez még csak az első változata volt a GPT-nek. Alig egy évvel később, a GPT-2 [1] [2] [24] már 1,5 milliárd paramétert tartalmazott, mely képes volt egyszerre több fejezetnyi szöveget is generálni. A Covid első évében, 2020-ban a GPT-3 [26] [2] [29] 175 milliárd paraméterrel került a piacra. Ez volt az a modell, ami először

tette lehetővé, hogy emaileket, verseket és már programnyelvi kódokat is tudjon írni. A nyelvek közötti fordítás sem okozott nehézséget neki. Ez az a GPT, melyet a legtöbben használtak, használnak a mai napig mindenféle feladatra. A legújabb, GPT-4 [3] [2] [17] pedig még több újtással tört be az emberek életébe 2023-ban. Ez nem csak szövegeket, hanem képeket, videókat és hangfelvételeket is fel tud dolgozni, hasonló válaszdíóvel, mint az ember.

Motivációm volt ezt a technológiát beépíteni az alkalmazásba, hogy a felhasználók számára megkönnyítse a különböző, esetenként bonyolult igények leadását, azoknak átalakítását olyan formára, melyet a különböző függvények és eszközök fel tudnak dolgozni. A használatához ne legyen szükség senkire a 'másik oldalon', mégis tudjon bármilyen kérdésre válaszolni.

## 4.2. OpenAI és LangChain

Az OpenAI és LangChain [14] [11] eszközeire esett a választás, hogy LLM-t integráljak az alkalmazásba. Segítségével a felhasználók könnyedén, emberi módon tudnak kommunikálni, mint ember az emberrel. A LangChain lehetőséget biztosít a bemenetek és a memória kezelésére úgynevezett *agent*-eken (ügynökökön) keresztül, az OpenAI segítségével pedig API-n keresztül el lehet érni a legújabb GPT modelleket [21] [19] [20] [18].

Használatához előzetes lépésekre van szükség, melyek a *prompt*, *chain* és *memory* menedzsment. A *prompt* segítségével fogalmazzuk meg a feladatot, a bemenetek formáját és értelmezését, azok használatát, és a kimenetek típusát, és formátumát. A *prompt engineering*-re [12] [7] egy egész szakma épül, fontos alapköve az LLM pontos és hatékony működéséhez. A *chain management* határozza meg a különböző folyamatok közötti eredmények megfelelő összekapcsolását. A modelleknek *tool*-okat (eszközöket) adhatunk meg feladatok elvégzésére, melyeknek különböző bemenetekre van szüksége, és ezek segítségével tudjuk a *tool*-ok az eredményeit egymás felé irányítani. Végül a memória-kezelés a kontextus megtartásához szükséges. Két üzenet között nem alapvető, hogy az LLM emlékezzen, így szükség lehet rá, ha több üzenetből álló beszélgetést szeretnénk létrehozni.

### 4.2.1. Promptok

Az alkalmazásban két fő *agent* [22] szerepel az adminisztrátorok és a felhasználók számára. Erre azért van szükség, hogy a különböző folyamatokat pontosan el tudják végezni a saját *tool*-jaik használatával, mivel azok nem mindig egyeznek meg. Hozzájuk a következő *prompt* [10] [7] tartozik (4.1), melyben definiálva vannak az *agent*-ek számára használatos *tool*-ok, és egy folyamat, amely mentén az inputot értelmeznie kell, kiválasztania egy eszközt, melyet használni fog.

Az alkalmazásban a *gpt-4o-2024-08-06* [19] nevű modellt van használva. Számos modell közül lehet választani, azonban erősebb modelleknél drágábbak a tokenek, melyek a használatához szükségesek. A tokenek olyan szövegrészek, amelyek segítségével a modell feldarabolja a bemenetet, majd választ generál. Ezek befolyásolják a felhasználási

```
system

Respond to the human as helpfully and accurately as possible. You have access to the
  following tools:
{tools}

Use a json blob to specify a tool by providing an action key (tool name) and an
  action_input key (tool input).
Valid "action" values: "Final Answer" or {tool_names}
Provide only ONE action per $JSON_BLOB, as shown:
```
{{
  "action": $TOOL_NAME,
  "action_input": $INPUT
}}
```

Follow this format:
Question: input question to answer
Thought: consider previous and subsequent steps
Action:
```
$JSON_BLOB
```
Observation: action result
... (repeat Thought/Action/Observation N times)
Thought: I know what to respond
Action:
```
{{
  "action": "Final Answer",
  "action_input": "Final response to human"
}}
```

Begin! Reminder to ALWAYS respond with a valid json blob of a single action. Use tools
  if necessary. Respond directly if appropriate. Format is Action:```${JSON_BLOB}``then
  Observation

human
{question}
{agent_scratchpad}
```

4.1. ábra. Prompt.

korlátokat és a költségeket is, mivel a szöveg feldolgozása a tokenek számán alapul. Általános szabályként egy token körülbelül 4 angol karakternek felel meg, de ez eltérhet más nyelvek esetében, különösen a magyarhoz hasonló, hosszabb szavakat tartalmazó nyelveknél. Ezeket figyelembe véve egy köztes erősségű, áru modellre esett a választás. Paraméterként a *temperature* (hőmérséklet) lett beállítva, mely a modell véletlenszerűségét határozza meg. 0-ás hőmérséklettel az *output* determinisztikusabb lesz, azaz hasonló kérdések esetén ugyan azt a választ fogja generálni.

### 4.2.2. Eszközök

A két *agent* között lévő különbség a *tools* lista, mely azok az implementált függvények, amiket az *agent* használni tud. Egy ilyen *tool* lehet egy egyszerű dátum lekérdező függvény (4.2).

```
from langchain.tools import StructuredTool

def get_current_date():
    from datetime import datetime

    now = datetime.now()
    return now

tools = [
    StructuredTool.from_function(
        func=get_current_date,
        name="Current_date/time_function",
        description="This_function_returns_the_current_date_and_time",
    )
]
```

4.2. ábra. Tool dátum lekérdezéséhez.

A megfelelő *prompt* és *description* segítségével, egy egyszerű kérdés a felhasználótól az *agent* felé a megfelelő függvény használatát fogja eredményezni ahelyett, hogy az *agent* magától találja ki, mit kell pontosan csinálnia és több iteráción át gondolkodjon. Erősebb nyelvi modellek képesek rá, hogy különféle segédfüggvények nélkül válaszoljanak, de minél több támogatást kapnak, annál pontosabbak lesznek a bemenetre érkező válaszok.

### 4.2.3. AgentExecutor

Az *agent*-ek és a *tool*-ok a definiálásukat követően az *AgentExecutor* osztályban található *from\_agent\_and\_tools* függvénnyel vannak létrehozva azokat az objektumokat, melyek az *invoke* metódus segítségével vannak meghívva a bemenetre. Ennek eredménye lesz az a válasz, melyet a modell a kérdés alapján fogalmaz meg és kerül vissza a felhasználóhoz.

```
from langchain.agents import AgentExecutor

agent_executor = AgentExecutor.from_agent_and_tools(
    agent=agent,
    tools=tools,
    verbose=True,
    memory=memory,
    handle_parsing_errors=True,
    max_iterations=5,
)
```

4.3. ábra. AgentExecutor példányosítás.

A *verbose* paraméter az előforduló hibák kezelésére ad információt a folyamat során. A *memory* a kontextus megtartásához, a *handle\_parsing\_errors* pedig arra szolgál, hogy az *agent\_executor* le tudja kezelni a fellépő hibákat és ne álljon le futás közben. A *max\_iterations* limitálja a végrehajtandó iterációk számát.

A *prompt* és a *tool* hatására a konzolban megjelenik a "gondolkodásmód".

```
> Entering new AgentExecutor chain...
{
  "action": "Current date/time function",
  "action_input": {}
}
2024-10-22 21:26:23.154997{
  "action": "Final Answer",
  "action_input": "Today's date is October 22, 2024."
}

> Finished chain.
Bot: Today's date is October 22, 2024.
```

4.4. ábra. A hívás különböző szakaszai.

### 4.2.4. Memória

A kontextus megtartásához szükség van memóriára. Ehhez a már említett *Message* adatmodell van használva. Minden elküldött üzenet, majd az arra kapott válasz mentésre kerül az adatbázisba, illetve az *AgentExecutor* memóriájába. A beszélgetés minden esetben egy "kezdeti" üzenettel kezdődik (4.5). Felhasználók esetében ez az alábbi üzenettel van inicializálva az *agent*:

```
You are an AI assistant that can provide helpful answers using available tools.
You will have to explain the user's upcoming week's application and modify it
if needed. If you are unable to answer, you can use the following tools: [...]. You
don't have to use them, if it's not mandatory
```

A memória létrehozása után betöltésre kerül a kezdeti üzenet.

```
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)
memory.chat_memory.add_message(SystemMessage(content=initial_message))
```

4.5. ábra. Inicializáló üzenet betöltése.

Ezt követően az adatbázisból lekért, adott felhasználóhoz tartozó üzenetek következnek. Ezeket az objektumokat a *sent\_by\_user* paraméterük alapján választjuk szét, hogy azok típusa emberi vagy gépi üzenet és megfelelő módon kerüljenek be a memóriába.

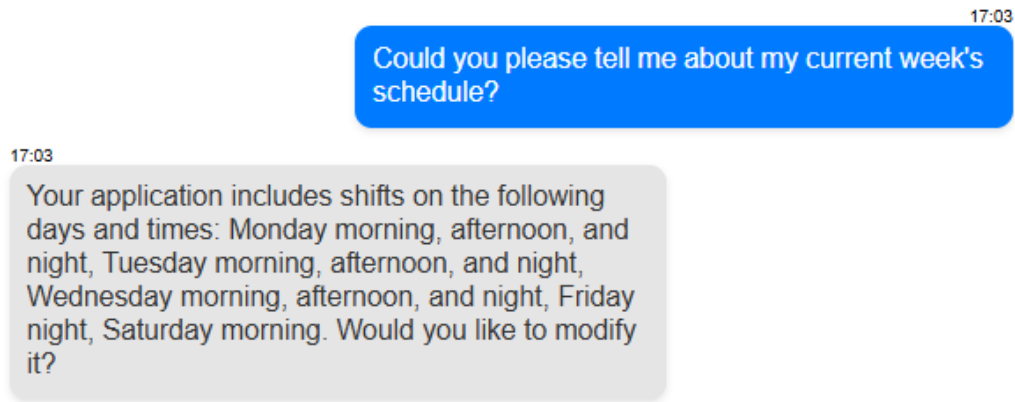
## 4.3. Kérések az asszisztens felé

Az asszisztens felé bármilyen kéréssel fordulhatnak a felhasználók, de hasonló módon a ChatGPT-hez, bántó vagy rosszindulatú kérdésre nem fog választ adni. A *prompt* és *tool*-ok megfelelő definiálásával meg lehet határozni a kontextust, ezt minden esetben érdemes precízen megtenni a hatékonyság érdekében. Az alkalmazásban a kérések fajtája a felhasználók csoportjától függ. A beosztottak, akik nem rendelkeznek a *Supervisor* csoport jogaival, a következő fő témákban tudnak kérdezni, információt kérni az asszistenstől.

### 4.3.1. Felhasználói folyamatok

#### Meglévő igények információinak lekérése

Ez, a felhasználó aktív, nyitott heti beosztásigényéről készít egy összegzést (4.6).



4.6. ábra. Jelentkezések listázására vonatkozó kérés.

A modell futása során látható (4.7), hogy melyik eszközt használja, (action) majd annak kimenetét hogyan, milyen formában küldi vissza a felhasználó számára.

```
> Entering new AgentExecutor chain...
```json
{
  "action": "Application summarization",
  "action_input": {}
}
```agent_output='Your application includes shifts on the following days and times:
Monday morning, afternoon, and night, Tuesday morning, afternoon, and night,
Wednesday morning, afternoon, and night, Friday night, Saturday morning. Would you
like to modify it?'```json
{
  "action": "Final Answer",
  "action_input": "Your application includes shifts on the following days and times:
Monday morning, afternoon, and night, Tuesday morning, afternoon, and night,
Wednesday morning, afternoon, and night, Friday night, Saturday morning. Would you
like to modify it?"
}
```
> Finished chain.
```

4.7. ábra. Igények lekérése a háttérben.

#### Igény leadása műszakokra

A felhasználók a nyitott hétre tudnak igényt leadni bármely műszak(ok)ra. Ilyenkor meg kell mondaniuk, hogy melyik napon melyik *shift*-et szeretnék elvinni. Az, hogy a kérést milyen formában teszik meg teljesen rájuk van bízva, az asszisztens képes átalakítani az általuk küldött emberi szöveget a megfelelő, előre meghatározott JSON formátumra, majd



azzal tovább dolgozni. Előnye, hogy nem kell, hogy a kapott input nyelvtanilag helyes legyen. Akár szavakból, szórészetekből is képes a modell megérteni a kérést. Legyen a következő a felhasználótól érkező igény:

I want to apply for all day Monday, Tuesday, Wednesday, and for the night of Friday, morning of Saturday.

Az erre történő reakciót a LLM felől a 4.8 ábra mutatja be.

```
> Entering new AgentExecutor chain...
'''
{
  "action": "Modification",
  "action_input": "Apply for all day Monday, Tuesday, Wednesday, and for the night of
    Friday, morning of Saturday"
}
'''
> Entering new AgentExecutor chain...
Thought: The user wants to apply for all shifts on
Monday, Tuesday, and Wednesday, as well as the night shift on Friday and the morning
  shift on Saturday. I will create a JSON object to represent this request.

Action:
```json
{
  "monday": { "morning": true, "afternoon": true, "night": true },
  "tuesday": { "morning": true, "afternoon": true, "night": true },
  "wednesday": { "morning": true, "afternoon": true, "night": true },
  "friday": { "night": true },
  "saturday": { "morning": true }
}
'''
Action:
```json
{
  "action": "Final Answer",
  "action_input": {
    "monday": { "morning": true, "afternoon": true, "night": true },
    "tuesday": { "morning": true, "afternoon": true, "night": true },
    "wednesday": { "morning": true, "afternoon": true, "night": true },
    "friday": { "night": true },
    "saturday": { "morning": true }
  }
}
'''
```

4.8. ábra. Egyszerű igény feldolgozása a modell által.

Szépen látszik, ahogy a modell átgondolva kialakítja, a *prompt*-ban meghatározott formátumú JSON-t, melyet a megfelelő függvények fel tudnak dolgozni.

Minden egyes igény után, legyen az új műszak, vagy törlés a tervezettek közül, nem történik azok elmentése azonnal az adatbázisban. A folyamat akkor ér véget, amikor a felhasználó kijelenti, hogy szeretné elmenteni a módosításait (lead egy kérést mentésre). Addig egy ideiglenes mezőben vannak tárolva az adat, hogy melyik műszakkal milyen módosítást szeretne a felhasználó végezni. Alapértelmezetten ez, a *modification* mező egy 21 hosszú,  $x$  karakterekből álló sztring, nem úgy mint a többi hasonló mező, melyek nullával vannak létrehozva és később 0, 1 karakterek szerepelnek bennük. Ez azért

szükséges, mivel így ha egy műszakra új igényt adnak le, úgy adott indexen *1*-es karakter, törlés esetén *0*-ás karakter jelenik meg, melyet a backend a megfelelő módon le tud kezelni.

Az előző kéréshez tartozó bináris: `111111111xxxxx11xxxxx`

Napokra szétbontva:

- hétfő: 111
- kedd: 111
- szerda: 111
- csütörtök: xxx
- péntek: xx1
- szombat: 1xx
- vasárnap: xxx

Jól tükrözi a kérést, összevetve a JSON-el.

A felhasználók több iteráción keresztül tudják módosítani az igényeiket kérésekkel.

Az előző példához visszatérve, egy folyamatban lévő módosítás után a felhasználó változtatni szeretné az igényeit. A határidő előtt egy nappal meggondolja magát, mégsem szeretne hétfő reggel és péntek este dolgozni, helyette a csütörtök délutáni és szombat esti műszakot szeretné megkapni:

Let's delete the monday morning and friday night application, and apply for thursday afternoon and saturday night

A kérésben szándékosan szerepelnek helyesírási hibák. A gondolkodási folyamat a 4.9 ábrán látható.

A bináris eddigi alakja: `111111111xxxxx11xxxxx` volt, utána pedig a megváltozott bináris: `011111111x1xxx01x1xxx` lett.

- hétfő: **0**11
- kedd: 111
- szerda: 111
- csütörtök: x**1**x
- péntek: xx**0**
- szombat: 1x**1**
- vasárnap: xxx

Látszik, hogy a hétfő reggeli és csütörtöki műszakra való törlés hogyan írja felül a folyamatban lévő igényt, és változik a hétfő délutáni és szombat esti műszak.

Minden üzenet végén az asszisztens rákérdez, szeretné-e a felhasználó elmenteni a módosításait. Mentés esetén egy egyszerű adatbázis művelet történik, az átmeneti változtatások felülírják az igényeket, vagyis a *modification* mezőben tárolt adat kerül az *application* mező helyére, de kizárólag csak azon indexen lévő karakterek, melyek változtatva lettek, azaz *0* vagy *1*-es érték található ott.

```
> Entering new AgentExecutor chain...
Action:
'''
{
  "action": "Modification",
  "action_input": "Delete Monday morning and Friday night application, and apply for
    Thursday afternoon and Saturday night."
}
'''

> Entering new AgentExecutor chain...
Thought: I need to modify the application by canceling the shifts on Monday morning and
  Friday night, and applying for Thursday afternoon and Saturday night.
Action:
'''
{
  "monday": { "morning": false },
  "friday": { "night": false },
  "thursday": { "afternoon": true },
  "saturday": { "night": true }
}
'''
```json
{
  "action": "Final Answer",
  "action_input": {
    "monday": { "morning": false },
    "friday": { "night": false },
    "thursday": { "afternoon": true },
    "saturday": { "night": true }
  }
}
'''
```

4.9. ábra. A legadott igény módosítása.

### Szabadságok kezelése

Két fontos művelet tudnak még elvégezni a felhasználók, szabadságok igénylését vagy azok törlését. A módszer hasonló, azzal a különbséggel hogy itt nem műszakokra, hanem napokra, időszakokra jelentkeznek. Ilyenkor meg kell adniuk, mettől meddig szeretnék a szabadságot betervezni.

```
I want to have vacation from november 5 to november 8.
```

Az asszisztens itt is egy JSON formátumra konvertálja a kérést egy plusz információval, amiben az jelenik meg hogy az igény új vakációra vagy törlésre vonatkozik (4.10). Ilyenkor nem kell véglegesíteniük a felhasználóknak azt, egyből látszódik a változás a panelen. Szabadságigényt csak a nyitott hét hétfőjétől tudnak leadni, viszont a vége lehet később is, nem kell, hogy azon a héten legyen vége. Amennyiben egy új igény érkezik szabadságra, úgy azokon a napokon lévő, már esetleg megjelölt műszakigények elvesznek. Fontos viszont, hogy előre meghatározott számú szabadnapja van mindenkinek. Ha egy felhasználó hamarabb szeretné elkezdni a szabadságot, mint a nyitott hét hétfője, vagy a leadott igényével együtt több szabadnapja lenne mint a megengedett, úgy az asszisztens

```
> Entering new AgentExecutor chain...
{
  "action": "Application for vacation",
  "action_input": "I want to have vacation from November 5 to November 8"
}

> Entering new AgentExecutor chain...
Action:
```
{
  "start": "05-11", "end": "08-11", "save": true
}
```
{
  "action": "Final Answer",
  "action_input": { "start": "05-11", "end": "08-11", "save": true }
}
```

> Finished chain.
agent_output='You have applied vacation from 05 Nov to 08 Nov'```json
{
  "action": "Final Answer",
  "action_input": "You have successfully applied for vacation from November 5 to
  November 8."
}
```
```

4.10. ábra. Szabadság igény leadása.

egy hibüzenet formájában jelzi azt felé. Utóbbi esetben tájékoztatást is kap, a még hátralévő szabadnapok számáról. Véglegesített szabadság visszavonására nincsen lehetőség, azaz ha egy beosztásigény módosítási határideje lezárult (péntek éjféllal), úgy szombaton és vasárnap az adminisztrátor elkezd megalkotni a végleges beosztást a matematikai modell segítségével, úgy már azon a beosztásigényen változtatni nem lehet.

### Betegség

Sajnos gyakori dolog, hogy reggel felkelve rosszul érzi magát az ember. Ilyenkor ha tud, egyből szól a felettesének, akinek azonnal meg kell oldani a helyettesítést. Az alkalmazásban erre hasonlóan a szabadságigényléshez van mód. Betegszabadságot azonban kizárólag a jelenlegi naptól kezdve adhat le a felhasználó az asszisztensnek, azt viszont bármeddig. Itt feltételezzük, hogy tudja, hány napot lesz távol. A helyettesítést a modell azonnal megteszi. Először, ha van még szabad tartalékos műszakban lévő felhasználó, úgy azok közül fog választani, próbálva olyat behívni, aki eddig az adott napon lévő tartalékosok közül a legkevesebb alkalommal lett behívva - ezzel elkerülve, hogy valaki kevesebbszer és mások többször legyenek mozgósítva. Ezt minden betegséggel jelzett napra megteszi. Amennyiben elér egy olyan naphoz, ahol erre nincs lehetőség, úgy arra a napra, szabadnapban lévő felhasználót fog behívni - a későbbiekben erősen kompenzálva ezért a dolgozót - majd a hét hátralévő részét újra optimalizálja. Erre azért van szükség, hogy a meglévő szabályokat a lehető legjobban be tudja tartani az újra számolt beosztás minden felhasználóra. Ilyenkor csak a következő naptól kezdődő napok műszakjai változnak.

### 4.3.2. Adminisztrátori folyamatok

A másik csoportnak, a *Supervisor* jogokkal rendelkező adminisztrátoroknak egy dolguk van: megalkotni a végleges beosztást. Erre nagyon egyszerűen meg tudják kérni az asszisztent. A leadott műszakigényekbe nem tudnak beleszólni, viszont a szabadnapokra beérkezett kérésekbe igen. Ezt tehetik valamilyen szakmai vagy személyes indokkal, hogy nem fogadják el a kérést, és törlik azokat. Ilyenkor előre egyeztetnek a dolgozókkal. Esetenként annak érdekében, hogy a matematikai modell meg tudja oldani a feladatot. Mivel feltételek sokasága határozza meg azt, így egy nagy gyenge pontja a vakációk kezelése, hiszen már heti egy nap is nagyban átírja a korlátozó feltételeket.

#### Szabadságigény elutasítása



4.11. ábra. Szabadságot tartalmazó roster.

A felhasználónak kedd és péntek között van egy 4 napos szabadságigénye. Az adminisztrátor úgy gondolja, nem engedi el pihenni mind a négy napra, így kiadja a kérést az asszisztens felé. A kérés háttérben történő futása a 4.12 ábrán látható.

Please remove the vacation request for davmil from november 5 to november 6.

#### Műszakok generálása

Amikor az adminisztrátorok le akarják generálni a végleges beosztást, úgy előfordulhat, hogy a modell nem ad megoldást, nem lesz fizibilis. Ebben az esetben korrigálniuk kell, szabadságigényeket törölni. Az alkalmazásban az adminisztrátorok látják, hogy azon a héten kinek mikor van betervezett vakációja és ezek közül kell egy vagy több napot elutasítaniuk. Meg kell jegyezni ugyan, hogy ekkor, ha valakinek visszatörlésre kerül egy, vagy több szabadnapja, úgy azokra a napokra nem lesz műszakokra vonatkozó leadott igénye, és teljesen véletlenszerűen dől el, hogy dolgozni fog-e, vagy sem, bár a modell próbál szabadnapokat (*Off Days*) vagy tartalékos napokat (*Reserve Days*) berakni ezekre az időszakokra.

```
> Entering new AgentExecutor chain...
Action:
'''
{
  "action": "Reject vacation",
  "action_input": { "user_request": "davmil from november 5 to november 6" }
}
'''

> Entering new AgentExecutor chain...
Action:
'''
{
  "user": "davmil",
  "start": "05-11",
  "end": "06-11"
}
'''
'''
{
  "action": "Final Answer",
  "action_input": { "user": "davmil", "start": "05-11", "end": "06-11" }
}
'''

> Finished chain.
agent_output='Vacation claim is successfully rejected for davmil from November 5 to
November 6'
I know what to respond.

Action:
'''
{
  "action": "Final Answer",
  "action_input": "Vacation claim is successfully rejected for davmil from November 5 to
November 6."
}
'''

> Finished chain.
```

4.12. ábra. Felhasználó szabadságának törlése adminisztrátor által.

## 5. fejezet

# Tervező motor

A chat-asszisztens mellett az alkalmazás másik nagy komponense a tervező motor. Ez a műszakok optimalizálását végzi egy lineáris programozási modell megoldásával. A különböző műszakok, illetve feltételek 1-1 egyenlettel vagy egyenlőtlenséggel vannak kifejezve, majd a modell ezekre egy optimális megoldást ad (feltéve, hogy létezik), értéket adva az ismeretlen változóknak, melyekből a beosztás épül fel.

A modellhez szükségesek változók, feltételek és egy célfüggvény, amit minimalizálni vagy maximalizálni szeretnénk. Ebben az esetben maximalizálás lesz, mivel a felhasználók által beérkezett igényeket akarjuk minél jobban kielégíteni, hogy a legtöbb műszakra vonatkozó igényből beosztott műszak legyen.

### 5.1. Problémaleírás az új műszakok generálásához

Jelen esetben a hét napjaira, minden nap pedig 3 műszakra kell beosztani az alkalmazottakat. Egy napon csak egy műszakban dolgozhatnak a felhasználók. Amennyiben szabadnapjuk van, tartalékos beosztásban vagy szabadságon vannak, úgy azon napokon nem lehet műszak. Ezen felül kikötés, hogy éjszakai műszakot nem követhet reggeli, vagy délutáni műszak. Minden egyes műszakon legalább egy, előre meghatározott mennyiségű alkalmazottra van szükség, és megköveteljük, hogy minden nap legalább 2 (paraméterezhető) felhasználó legyen tartalékos, bármilyen nem várt esemény esetére. A célfüggvény az igények maximalizálása lesz.

#### 5.1.1. Matematikai modell

##### Jelölések

A modell felépítéséhez számos változóra és paraméterre van szükség. Fontos, hogy ezek meg legyenek különböztetve, külön pontba szedve.

Paraméterek:

- $I$ : A felhasználók halmaza, ahol  $i \in I$  a felhasználókat jelöli,  $i = 1, \dots, n$  indexeléssel.

- $J$ : A napok halmaza, ahol  $j \in J$  a napokat jelöli,  $j = 1, \dots, 14$  indexeléssel.
- $S$ : A műszakok halmaza, ahol  $s \in S$  a műszakokat jelöli,  $s = 1, \dots, 42$  indexeléssel.
- $d_s$ : A  $s$  műszakhoz rendelt minimális felhasználószám, amely a műszak időpontjának megfelelően van meghatározva, azaz a  $\lfloor \frac{s-1}{3} \rfloor + 1$  napon.
- $q$ : Az egy héten belüli munkanapok száma, ahol minden munkanap legalább egy felhasználót igényel.
- $r$ : Az egy héten belüli szabadnapok száma, amelyek során a felhasználók nem rendelkezhetők műszakhoz.
- $t$ : Az egy héten belüli tartalékos napok száma, amikor a felhasználók tartalékként állnak rendelkezésre.
- $u_{vac}$ : Az egy héten belüli szabadságnapok száma, amikor a felhasználók nem állnak rendelkezésre.
- $u_{sick}$ : Az egy héten belüli betegszabadságnapok száma, amikor a felhasználók nem állnak rendelkezésre.
- $v$ : A napokhoz rendelt minimális tartalékos felhasználószám.

### Változók:

- $x_{i,s}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó be van-e osztva a  $s$  műszakhoz;  $x_{i,s} = 1$ , ha be van osztva, és  $x_{i,s} = 0$ , különben.
- $y_{i,s}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó igényt nyújtott-e be a  $s$  műszakra;  $y_{i,s} = 1$ , ha igen, és  $y_{i,s} = 0$ , különben.
- $z_{i,s}$ : Bináris döntési változó, amely az  $x_{i,s}$  és  $y_{i,s}$  szorzatát reprezentálja.
- $l_{i,j}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó dolgozik-e a  $j$  napon;  $l_{i,j} = 1$ , ha dolgozik, és  $l_{i,j} = 0$ , különben.
- $m_{i,j}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználónak szabadnapja van-e a  $j$  napon;  $m_{i,j} = 1$ , ha szabadnapos, és  $m_{i,j} = 0$ , különben.
- $n_{i,j}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó tartalékos-e a  $j$  napon;  $n_{i,j} = 1$ , ha tartalékos, és  $n_{i,j} = 0$ , különben.
- $o_{i,j}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó szabadságon van-e a  $j$  napon;  $o_{i,j} = 1$ , ha szabadságon van, és  $o_{i,j} = 0$ , különben.
- $w_{i,j}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó betegszabadságon van-e a  $j$  napon;  $w_{i,j} = 1$ , ha betegszabadságon van, és  $w_{i,j} = 0$ , különben.



## 5.1.2. Célfüggvény és korlátozó feltételek

### Célfüggvény

A változók definiálásával meg kell adni egy célfüggvényt, mely során a modell optimalizálni fogja az egészértékű lineáris modellt. A cél, hogy a kiadott igények és a végleges beosztás a lehető legjobban passzoljon, azaz minél több igényből legyen tényleges műszak.

$$\text{Maximize } \sum_{i \in I} \sum_{s \in \{22, 42\}} z_{i,s} \quad (5.1)$$

### Korlátozó feltételek

A korlátozó feltételek lesznek azok, melyek a hétköznapi nyelvből vett szabályokból egyenlet formáját veszik fel a modellben. Ahogy már említve volt, ezek a kikötések lehetnek a munkatörvényből származóak, a cég által felépített, vagy akár a dolgozói szövetkezet lévén létrehozott szabályozások. A modellben az alábbiak szerepelnek:

**0. Változók fixálása:** ♦ A szabályok betartásához szükség van az előző heti beosztás figyelembevételére. Például, vasárnap esti műszak után a modell ne generálhasson hétfő reggelit. Ennek érdekében lettek a változók nem 7, hanem 14 napra definiálva, ugyancsak a műszakra vonatkozó halmaz is pontosan kétszer akkora, mint ahány műszakból áll egy hét. Fontos megjegyezni, hogy az éppen megoldandó feladathoz tartozó hét mindig a második, azaz a napok indexei  $\{8, 9, \dots, 14\}$ , a műszakoké pedig  $\{22, 23, \dots, 42\}$ . Az első héthez tartozó változók le vannak fixálva, melyek már egy generált beosztást definiálnak. Erre azért van szükség, mert a szabályokat hétről hétre folytonosan akarjuk alkalmazni, ezzel pedig a felmerülő hibákat lehet elkerülni. Ez azt jelenti, hogy az első hétre vonatkozóan értéket adunk a változóknak, és megadjuk a modellnek, hogy ezekkel az értékekkel számoljon, azokat ne változtassa a futás során. Ekkora az  $x_{i,s}$  (műszakok),  $l_{i,j}$  (munkanapok),  $m_{i,j}$  (szabadnapok),  $n_{i,j}$  (tartalékos napok),  $o_{i,j}$  (szabadságok) és  $w_{i,j}$  (betegszabadságok) változóit, a műszakoknál a 1..21, napoknál pedig a 1..7 indexeken egyenlővé tesszük az előző heti beosztás alapján 0 vagy 1-es értékkel.

- $x_{i,s}$ : 1, ha  $i$  felhasználó dolgozott az  $s$  műszakban, különben 0  $\forall i \in I, \forall s \in \{1, 2, \dots, 21\}$
- $l_{i,j}$ : 1, ha  $i$  felhasználó dolgozott a  $j$  napon, különben 0  $\forall i \in I, \forall j \in \{1, 2, \dots, 7\}$
- $m_{i,j}$ : 1, ha  $i$  felhasználó szabadnapon volt a  $j$  napon, különben 0  $\forall i \in I, \forall j \in \{1, 2, \dots, 7\}$

- $n_{i,j}$ : 1, ha  $i$  felhasználó tartalékos volt a  $j$  napon, különben 0  $\forall i \in I, \forall j \in \{1, 2, \dots, 7\}$
- $o_{i,j}$ : 1, ha  $i$  felhasználó szabadságon volt a  $j$  napon, különben 0  $\forall i \in I, \forall j \in \{1, 2, \dots, 7\}$
- $w_{i,j}$ : 1, ha  $i$  felhasználó beteg volt a  $j$  napon, különben 0  $\forall i \in I, \forall j \in \{1, 2, \dots, 7\}$

**1. Segédváltozókra vonatkozó feltételek:** ♦ A legegyszerűbb mód vizsgálni, hogy mikor egyezik egy igény ( $y_{is}$ ) és egy műszak ( $x_{is}$ ), ha a két változó össze van szorozva, eredményül pedig 1-et adnak. Ezt viszont a célfüggvénybe nem lehet beépíteni, mivel a kifejezés nem lineáris, a feladat pedig igen. Ehhez be kell vezetni egy segédváltozót, legyen  $z_{is}$ . Az  $x_{is}$  és  $y_{is}$  szorzata csak akkor lehet 1, ha mindkét tag nem nulla, azaz  $x_{is} = 0$  és/vagy  $y_{is} = 0$  esetén  $z_{is}$  is nulla lesz [8].

$$z_{i,s} \leq x_{i,s} \quad \forall i \in I, \forall s \in \{22, 23, \dots, 42\} \quad (5.2)$$

$$z_{i,s} \leq y_{i,s} \quad \forall i \in I, \forall s \in \{22, 23, \dots, 42\} \quad (5.3)$$

**2. Segédváltozókra vonatkozó feltételek:** ♦ Ezt követően meg kell adni, hogy  $z_{is}$  akkor és csak akkor legyen 1, ha mind  $x_{is}$  és  $y_{is}$  is egyenlő 1-gyel:

$$z_{i,s} \geq x_{i,s} + y_{i,s} - 1 \quad \forall i \in I, \forall s \in \{22, 23, \dots, 42\} \quad (5.4)$$

**3. Műszak lefedettség:** ♦ Minden műszakban legalább egy előre meghatározott számú operátornak kell dolgoznia. Ez függ a hét napjain és a különböző napszakokon:

$$\sum_{i \in I} s_{i,s} \geq d_s \quad \forall s \in \{22, 23, \dots, 42\} \quad (5.5)$$

**4. Munkanapok száma:** ♦ Minden felhasználó pontosan  $q$  napra kell, hogy műszakot kapjon:

$$\sum_{j \in \{8,9,\dots,14\}} l_{i,j} = q \quad \forall i \in I \quad (5.6)$$

**5. Szabadnapok száma:** ♦ Minden felhasználó pontosan  $r$  szabadnapot kell, hogy kapjon:

$$\sum_{j \in \{8,9,\dots,14\}} m_{i,j} = r \quad \forall i \in I \quad (5.7)$$

**6. Tartalékos napok száma:** ♦ Minden alkalmazottnak pontosan  $t$  tartalékos napot kell, hogy kapjon:

$$\sum_{j \in \{8,9,\dots,14\}} n_{i,j} = t \quad \forall i \in I \quad (5.8)$$

**7. Napokra vonatkozó feltétel 1:** ♦ Minden nap kizárólag vagy munkanap, vagy szabadnap, vagy tartalékos nap, vagy szabadság, vagy pedig betegség miatti távollét. Nem lehet egyszerre kettő, vagy több:

$$l_{i,j} + m_{i,j} + n_{i,j} + o_{i,j} + w_{i,j} = 1 \quad \forall i \in I, \forall j \in \{8,9,\dots,14\} \quad (5.9)$$

**8. Napokra vonatkozó feltétel 2:** ♦ Minden nap kizárólag egy műszakban dolgozhat az alkalmazott:

$$\sum_{k=1}^3 s_{i,(j-1) \cdot 3+k} \leq 1 \quad \forall i \in I, \forall j \in \{8,9,\dots,14\} \quad (5.10)$$

**9. Napokra vonatkozó feltétel 3:** ♦ A felhasználók kizárólag csak kiosztott munkanapra kaphatnak műszakokat:

$$\sum_{k=1}^3 s_{i,(j-1) \cdot 3+k} = l_{i,j} \quad \forall i \in I, \forall j \in \{8,9,\dots,14\} \quad (5.11)$$

**10. Napokra vonatkozó feltétel 4:** ♦ Tartalékos napot szabadnapnak kell követnie:

$$m_{i,j+1} \geq n_{i,j} \quad \forall i \in I, \forall j \in \{1,2,\dots,13\} \quad (5.12)$$

**11. Napokra vonatkozó feltétel 5:** ♦ Tartalékos nap előtt nem lehet szabadnap:

$$m_{i,j-1} \leq 1 - n_{i,j} \quad \forall i \in I, \forall j \in \{2,3,\dots,14\} \quad (5.13)$$

**12. Napokra vonatkozó feltétel 6:** ♦ A felhasználóknak maximum 2 esti műszakjuk lehet a héten:

$$\sum_{j \in \{8,9,\dots,14\}} s_{i,3*(j-1)+3} \leq 2 \quad \forall i \in I \quad (5.14)$$

**13. Tartalékos napok:** ♦ Minden nap legalább  $v$  tartalékos műszak kell, hogy kiosztva legyen:

$$\sum_{i \in I} n_{i,j} \geq v \quad \forall s_j \in S, j \in 14 \quad (5.15)$$

A változók közül a  $q$ ,  $r$  és  $t$  a  $u_{vac}$  és  $u_{sick}$  függvényében van megadva az alábbi egyenletek segítségével:

$$q = \min(4, 7 - (u_{vac} + u_{sick}))$$

$$r = \max(0, 2 - (u_{vac} + u_{sick}))$$

$$t = \begin{cases} 1 & \text{if } (u_{vac} + u_{sick}) < 3 \\ 0 & \text{otherwise} \end{cases}$$

Ahol  $q$  a munkanapok,  $r$  a szabadnapok,  $t$  pedig a tartalékos napok száma,  $u$ , azaz a heti szabadságok számától függően. A 11. és 12. feltételnél megköveteljük, hogy azon a héten a felhasználónak legyen legalább egy szabadnapja, hogy szabadság vagy betegség esetén ez ne okozzon gondot.

## 5.2. Problémaleírás újra tervezés esetén betegséget követően

A hirtelen betegségek komolyan fel tudják borítani a már alaposan megtervezett beosztásokat. Ennek kikerülése érdekében lettek bevezetve a tartalékos műszakok, hogy 1-1 nem várt eseményt le tudjon kezelni az alkalmazás, és ne legyen kiesés a műszakokban. A tartalékos felhasználók száma viszont véges minden nap, és bár ritka, de nem kizárt egy olyan helyzet, mikor nincs már szabad tartalékos. Például, ha minden nap kettő *reserve* műszakot generálunk, és adott napon már két felhasználó is beteget jelentett, egy harmadik betegszabadság esetén valamit tenni kell. Ebben a szituációban a modell először egy szabadnapos felhasználó behívását fogja kezdeményezni, majd a hét hátralévő részét újra számolja, ügyelve a minimális változásokra, mégis, betartva a különböző napokra és műszakokra vonatkozó szabályokat. Természetesen ezt az opciót szeretnénk elkerülni, így érdemes a felhasználók és napi tartalékos műszakok számát úgy kialakítani, hogy ilyen eset meg se történhesen, de fel kell rá készülni. Ez a fejezet ezt fogja tárgyalni és mutatja be, hogy változik a modell ebben az esetben. Mivel a betegséget többségében a jelenlegi héten jelzik a felhasználók, és az, illetve a következő hét, melyekre már generált beosztás van, így minden esetben ezt a két hetet kell figyelembe venni. A változók fixálása függ attól, hogy mely napon nem lehet több tartalékost behívni, így lehet, hogy csak pár napra kell fixálni és a maradék, akár 10-12 napra újra tervezni, de az is lehet, hogy valaki előre tudja, hogy a következő héten lesz egy kisebb műtétje. Abban az esetben csak pár napról lesz szó.

## 5.2.1. Matematikai modell

### Jelölések

Ebben a modellben a változók és paraméterek többségében megegyeznek, de vannak helyek ahol új változók bevezetésére van szükség. Mivel az a cél, hogy az újra generált beosztás minimálisan térjen el a jelenlegitől, az új beosztást szeretnénk a régihez mérten illeszteni a lehető legpontosabban.

Paraméterek:

- Az előző modellből átvett paraméterek:  $I, J, S, d_s, q, r, t, u, v$
- $e$ : a maximum limit felül kiosztott munkanapokat számát szabályozó súly, mely a célfüggvényben jelenik meg
- $f$ : a maximum limit mellett ki nem osztott szabadnapok számát szabályozó súly, mely a célfüggvényben jelenik meg
- $g$ : a maximum limit mellett ki nem osztott tartalékos napok számát szabályozó súly, mely a célfüggvényben jelenik meg
- $h$ : A szabadságigény első napját követő nap indexe. Erre azért lesz szükség, mivel a műszakokat és napokat ettől az indextől kezdve kell újra generálni a változók halmazában.

Változók:

- $x_{i,s}^{\text{prev}}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó be volt-e osztva a  $s$  műszakhoz;  $x_{i,s}^{\text{prev}} = 1$ , ha be van osztva, és  $x_{i,s}^{\text{prev}} = 0$ , különben.
- $x_{i,s}$ : nem változik.
- $l_{i,j}^{\text{prev}}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó dolgozott-e a  $j$  napon;  $l_{i,j}^{\text{prev}} = 1$ , ha dolgozott, és  $l_{i,j}^{\text{prev}} = 0$ , különben.
- $l_{i,j}$ : nem változik.
- $m_{i,j}^{\text{prev}}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználónak szabadnapja volt-e a  $j$  napon;  $m_{i,j}^{\text{prev}} = 1$ , ha szabadnapos volt, és  $m_{i,j}^{\text{prev}} = 0$ , különben.
- $m_{i,j}$ : nem változik.
- $n_{i,j}^{\text{prev}}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó tartalékos-e a  $j$  napon;  $n_{i,j}^{\text{prev}} = 1$ , ha tartalékos, és  $n_{i,j}^{\text{prev}} = 0$ , különben.
- $n_{i,j}$ : nem változik.
- $o_{i,j}^{\text{prev}}$ : Bináris döntési változó, amely azt jelöli, hogy a  $i$  felhasználó szabadságon volt-e a  $j$  napon;  $o_{i,j}^{\text{prev}} = 1$ , ha szabadságon volt, és  $o_{i,j}^{\text{prev}} = 0$ , különben.

- $o_{i,j}$ : nem változik.
- $w_{i,j}$ : nem változik.
- $z_{i,s}^x$ : Bináris döntési változó, amely az  $x_{i,s}^{\text{prev}}$  és  $x_{i,s}$  szorzatát reprezentálja.
- $z_{i,j}^l$ : Bináris döntési változó, amely az  $l_{i,j}^{\text{prev}}$  és  $l_{i,j}$  szorzatát reprezentálja.
- $z_{i,j}^m$ : Bináris döntési változó, amely az  $m_{i,j}^{\text{prev}}$  és  $m_{i,j}$  szorzatát reprezentálja.
- $z_{i,j}^n$ : Bináris döntési változó, amely az  $n_{i,j}^{\text{prev}}$  és  $n_{i,j}$  szorzatát reprezentálja.
- $p_{i,j}^l$ : Büntetés, ha  $i$  felhasználó plusz munkanapot kap  $j$  napon.
- $p_{i,j}^m$ : Büntetés, ha  $i$  felhasználó nem kap szabadnapot  $j$  napon.
- $p_{i,j}^n$ : Büntetés, ha  $i$  felhasználó nem kap tartalékos napot  $j$  napon.

## 5.2.2. Célfüggvény és korlátozó feltételek

### Célfüggvény

Ebben az esetben nem az igények és a legenerált műszakok maximalizálása, hanem a régi és új beosztás részei, a műszakok, szabadnapok és tartalékos napok megtartása a cél. A szabadságok és leadott betegségigények fixek.

$$\begin{aligned}
 \text{Maximize} \quad & \sum_{i \in I} \sum_{s \in \{1,2,\dots,42\}} z_{i,s}^x \\
 & + \sum_{i \in I} \sum_{j \in \{1,2,\dots,14\}} (z_{i,j}^l + z_{i,j}^m + z_{i,j}^n) \\
 & - \sum_{i \in I} \sum_{j \in \{1,2,\dots,14\}} (e \cdot p_{i,j}^l + f \cdot p_{i,j}^m + g \cdot p_{i,j}^n)
 \end{aligned} \tag{5.16}$$

### Korlátozó feltételek

**0. Változók fixálása:** ♦ Egészen a betegség napjáig, ahova már nem lehet tartalékos felhasználót beosztani, hasonló módon az előző modellhez, "fixálni" kell minden változót, mivel a múltbéli és jelenlegi napokat nem kell újra számolni. A változás az előző modellhez képest kizárólag az index lesz, ameddig a változókhoz értéket rendelünk a beosztás alapján. Ezt a  $h$  paraméter szabályozza, napok esetén a  $h$ . napig, műszakok esetén pedig a  $(h-1) \cdot 3 + 1$ . műszakig kell az előzetes beosztás alapján értékeket adni, attól függetlenül, hogy a jelenlegi vagy a következő héten van ez a nap.  $h$  az a nap, ahova már nem tudnak helyettesítést intézni, és attól a naptól kezdve kell újragenerálni a beosztást.

**1. Segédváltozókra vonatkozó feltételek:** ♦ Hasonlóan az előző modellhez, segédváltozók bevezetésére van szükség. Mivel azt szeretnénk, hogy a lehető legkisebb mértékben változzon a beosztás, nem az igények és műszakok szorzatát maximalizáljuk, hanem a régi és új műszakok, munka-, szabad-, tartalékos napok, valamint szabadság és betegszabadságokét. Ehhez minden változó után új segédváltozóra van szükség.

$$z_{i,s}^x \leq x_{i,s} \quad \forall i \in I, \forall s \in \{1, 2, \dots, 42\} \quad (5.17)$$

$$z_{i,s}^x \leq x_{i,s}^{\text{prev}} \quad \forall i \in I, \forall s \in \{1, 2, \dots, 42\} \quad (5.18)$$

$$z_{i,j}^1 \leq l_{i,s} \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.19)$$

$$z_{i,j}^1 \leq l_{i,s}^{\text{prev}} \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.20)$$

$$z_{i,j}^m \leq m_{i,s} \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.21)$$

$$z_{i,j}^m \leq m_{i,s}^{\text{prev}} \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.22)$$

$$z_{i,j}^n \leq n_{i,s} \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.23)$$

$$z_{i,j}^n \leq n_{i,s}^{\text{prev}} \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.24)$$

**2. Segédváltozókra vonatkozó feltételek:** ♦ Minden  $z_{i,s}$  'típusú' változó akkor és csak akkor legyen 1, ha a linearizáláskor hozzá tartozó, minkettő bináris változó értéke 1:

$$z_{i,s}^x \geq x_{i,s} + x_{i,s}^{\text{prev}} - 1 \quad \forall i \in I, \forall s \in \{1, 2, \dots, 42\} \quad (5.25)$$

$$z_{i,j}^1 \geq l_{i,j} + l_{i,j}^{\text{prev}} - 1 \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.26)$$

$$z_{i,j}^m \geq m_{i,j} + m_{i,j}^{\text{prev}} - 1 \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.27)$$

$$z_{i,j}^n \geq n_{i,j} + n_{i,j}^{\text{prev}} - 1 \quad \forall i \in I, \forall j \in \{1, 2, \dots, 14\} \quad (5.28)$$

**3. Műszak lefedettség:** ♦ Nem változik.

**4. Munkanapok, 5. Szabadnapok, 6. Tartalékos napok száma:** ♦ Az előző modell szerint minden felhasználó pontosan  $q$  napra kell, hogy műszakot kapjon,  $r$  napon szükséges, hogy szabadnap legyen és  $t$  darab tartalékos nap kell a *roster*-en. Mivel ezek a feltételek nem teljesülnek, ha valakit akár tartalékos, akár szabadnapról hívnak be, úgy minden egyes egyenlethez bevezetésre kerül egy büntetőváltozó, mely megengedi, hogy az eredetinel kevesebb szabad, tartalékos, vagy több szabadnapja legyen, ezt viszont büntetni fogja a célfüggvény. Így, a napokra vonatkozó feltételek:

Ha  $h > 7$ :

$$\sum_{j \in \{8,9,\dots,14\}} l_{i,j} - p_{i,j}^1 = q, \quad \forall i \in I, \quad (5.29)$$

$$\sum_{j \in \{8,9,\dots,14\}} m_{i,j} + p_{i,j}^m = r, \quad \forall i \in I, \quad (5.30)$$

$$\sum_{j \in \{8,9,\dots,14\}} n_{i,j} + p_{i,j}^n = t, \quad \forall i \in I. \quad (5.31)$$

Különben:

$$\sum_{j \in \{1,2,\dots,7\}} l_{i,j} - p_{i,j}^1 = q, \quad \forall i \in I, \quad (5.32)$$

$$\sum_{j \in \{8,9,\dots,14\}} l_{i,j} = q, \quad \forall i \in I, \quad (5.33)$$

$$\sum_{j \in \{1,2,\dots,7\}} m_{i,j} + p_{i,j}^m = r, \quad \forall i \in I, \quad (5.34)$$

$$\sum_{j \in \{8,9,\dots,14\}} m_{i,j} = r, \quad \forall i \in I, \quad (5.35)$$

$$\sum_{j \in \{1,2,\dots,7\}} n_{i,j} + p_{i,j}^n = t, \quad \forall i \in I, \quad (5.36)$$

$$\sum_{j \in \{8,9,\dots,14\}} n_{i,j} = t, \quad \forall i \in I. \quad (5.37)$$

$$p_{i,j}^1 \geq 0, \quad \forall i \in I \quad \forall j \in J. \quad (5.38)$$

$$p_{i,j}^m \geq 0, \quad \forall i \in I \quad \forall j \in J. \quad (5.39)$$

$$p_{i,j}^n \geq 0, \quad \forall i \in I \quad \forall j \in J. \quad (5.40)$$

Ezekkel a feltételekkel a modell megengedi, hogy 1-1 felhasználó akár 5-6 napot is dolgozzon egy héten, vagy kevesebb szabadnapja legyen mint a megengedett, ezt viszont a célfüggvényben büntetni fogja. Ezeknek a büntetéseknek a súlyozását az  $e$ ,  $f$  és  $g$ , paraméterek szabályozzák.



**7., 8., 9. Napokra vonatkozó feltétel:** ♦ Nem változik.

**10., 11., 12. Napokra vonatkozó feltétel:** ♦ Tartalékos napot szabadnapnak kell követnie, előtte pedig nem lehet szabadnap, illetve a felhasználóknak maximum 2 esti műszakjuk lehet a héten:

$$m_{i,j+1} \geq n_{i,j}, \quad \forall i \in I, \forall j \in \{1, 2, \dots, 13\} \text{ such that: } \begin{cases} \text{reserve\_call\_in}(i) = \text{True}, \\ \text{day\_off\_call\_in}(i) = \text{True}. \end{cases} \quad (5.41)$$

$$m_{i,j-1} \leq 1 - n_{i,j}, \quad \forall i \in I, \forall j \in \{2, 3, \dots, 14\} \text{ such that: } \begin{cases} \text{reserve\_call\_in}(i) = \text{True}, \\ \text{day\_off\_call\_in}(i) = \text{True}. \end{cases} \quad (5.42)$$

$$\sum_{j \in \{8, 9, \dots, 14\}} s_{i, 3*(j-1)+3} \leq 2 \quad \forall i \in I \text{ such that: } \begin{cases} \text{reserve\_call\_in}(i) = \text{True}, \\ \text{day\_off\_call\_in}(i) = \text{True}. \end{cases} \quad (5.43)$$

A tartalékos napra vonatkozó feltételek és az maximális éjszakai műszakok számát korlátozó feltételek kizárólag azoknál a felhasználóknál marad meg, akit azon a héten nem hívtak be tartalékos, vagy szabadnapról.

**13. Tartalékos napok:** ♦ Megegyezik az első modellel.

A  $q$ ,  $r$  és  $t$  változók ebben az esetben az eredeti *roster*-ek alapján vannak megadva, azaz az új beosztásban mind a munka-, szabad- és tartalékos napok számának meg kell egyeznie a régi beosztásban szereplővel. Mivel ez nem biztos, hogy teljesülne, ezt a rájuk vonatkozó büntető változók ( $p_{i,j}^l, p_{i,j}^m, p_{i,j}^n$ ) használata teszi lehetővé.

## 5.3. Megoldó

A felírt lineáris modell megoldásához már csak egy megoldóra van szükség. A dolgozatban ehhez több módszer és solver kipróbálása után az OR-Tools könyvtár segítségével az SCIP-re [28] [27] esett a választás.

### 5.3.1. OR-Tools

Az OR-Tools [4] egy nyílt forráskódú optimalizáló könyvtár, amelyet a Google fejlesztett ki, és célja, hogy segítsen a matematikai optimalizálási problémák megoldásában. A

könyvtár támogatja az egészértékű lineáris programozást, így széleskörű alkalmazási lehetőségeket kínál különböző területeken, mint például logisztika, ütemezés és erőforrás-allokáció. Az OR-Tools Python, C++ és Java nyelveken érhető el, lehetővé téve a fejlesztők számára, hogy rugalmasan integrálják a megoldásokat a saját alkalmazásaikba. Az eszköz egyszerű, mégis hatékony megközelítése segít a bonyolult problémák gyors és hatékony kezelésében.

### 5.3.2. SCIP

A SCIP (Solving Constraint Integer Programs) egy nyílt forráskódú matematikai optimalizáló szoftver, amelyet főleg egészértékű programozási problémák megoldására fejlesztettek ki. A SCIP célja, hogy hatékony megoldásokat kínáljon a kombinatorikus optimalizálási feladatokra, mint például a gráfok és a hálózatok optimalizálása. A szoftver integrálható más optimalizáló keretrendszerekkel, például a Google OR-Tools-szal, így lehetőséget ad a felhasználóknak arra, hogy különböző programozási nyelveken és platformokon használják.

### 5.3.3. Modell implementálása

Az alkalmazásban kevés paraméter, viszont annál több változó van definiálva. A változók a matematikai modelltől látott műszakok, munkanapok, szabadnapok, tartalékos napok és szabadságok. Az előző heti beosztás szerint az első hétre vonatkozó változók, majd a szabadságok függvényében az adott heti napok fixálását követően meghatározzuk a korlátozó feltételeket és a célfüggvényt. Minden egyes felhasználóhoz 42 feltétel tartozik (melyek a meglévő heti betervezett szabadság függvényében lehet, hogy kevesebb), ezek mellett pedig 28, felhasználótól független korlátozó egyenlet van a modellben. Ez összesen

$$\text{feltételek száma} = 28 + \text{felhasználók száma} \times 42$$

darab feltételt eredményez, melyet a model *verbose* módban való futása is igazol. Ebből az olvasható ki, hogy a futtatás során 658 feltétellel kezd, melyek közül 105-öt egyből el is dob, a változók száma pedig 630 lesz, bár a modellben lényegesen több van. Ez visszafejthető az előlegesen kitörölt 840 változóból, melyből kiderül hogy összesen 1470 változó szerepel a modellben.

### 5.3.4. Futásidő és eredmények

Az egyik eszköz, mellyel mérni lehet a modellt a futásidő. Ezt jelen esetben a változók és a feltételek függvényében érdemes számolni. A futásidő teszteléséhez összesen 100 felhasználóra lett futtatva a modell, tízesével növelve a felhasználók számát, amely a korlátozó feltételeket az előző részben található képlet alapján növeli. Ezek mellett vizsgálva van a hatékonysága a modellnek, azaz hogyan tudja kielégíteni az igényeket napokra lebontva. Egy igény pontosan akkor van kielégítve, ha a felhasználó:

- nem adott le igényt a napra és nem is kapott egy műszakot sem, vagy
- a megkapott műszak előzetes igény alapján lett kiosztva

Amennyiben valamelyik a kettő közül teljesül, úgy azt kielégített, ellenkező esetben kielégítetlen igénynek vesszük.

Egy ilyen kielégítetlen igény lehet a következő.



5.1. ábra. Egy "szellős" jelentkezés.

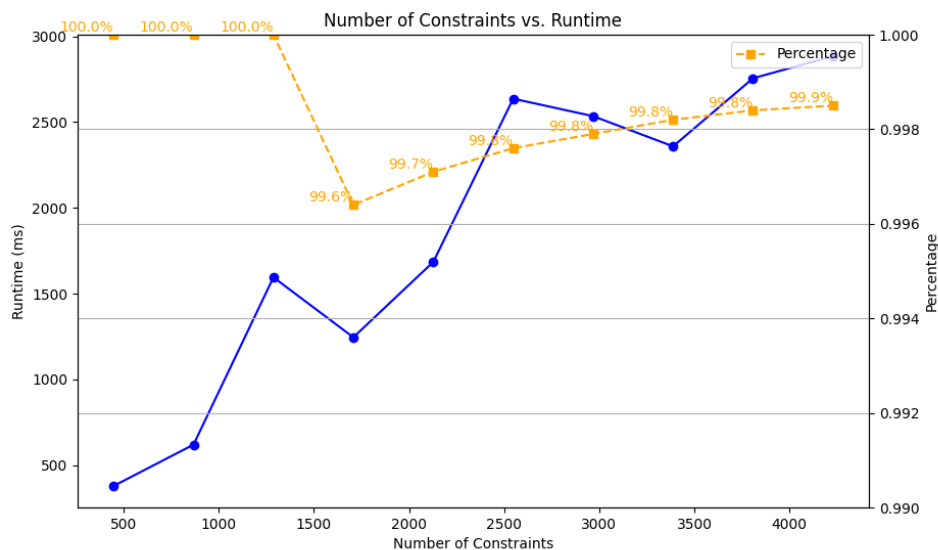


5.2. ábra. És a hozzá tartozó műszakok.

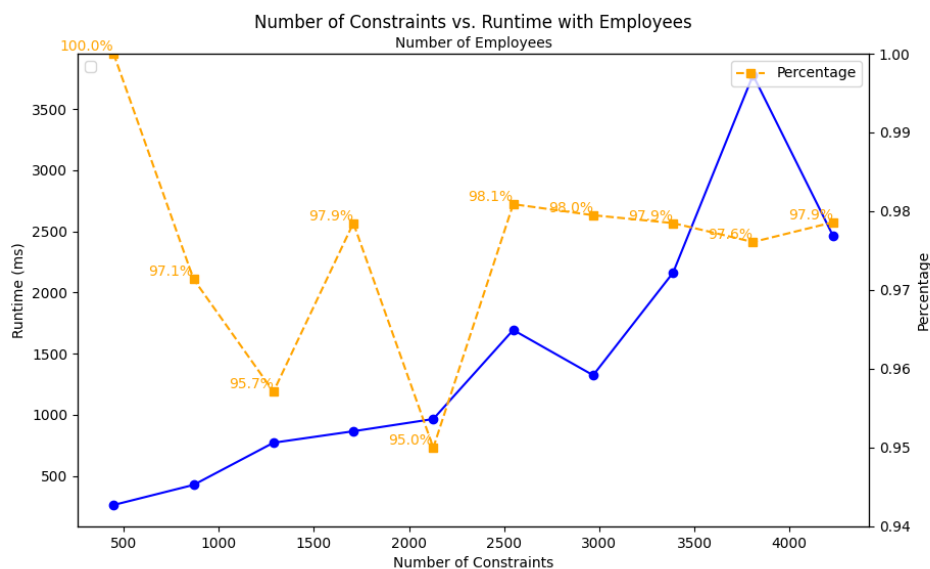
Látható, hogy a felhasználó kedd, szombat és vasárnapi napra pontosan azokat a műszakokat kapta, de péntekre is be lett osztva, pedig nem adott le igényt. Persze, ha kevesebb napra szeretne műszakot, mint a minimum (4), biztos, hogy lesz olyan nap, ahol be lesz osztva leadott igény nélkül.

Így, a különböző futások eredményét diagramon lehet szemléltetni (5.3 ábra). A futásidő többnyire lineárisan nő, ez nagyobb mennyiségű felhasználónál lenne jobban látható. A hiba az elején nulla, azaz mindenkit sikerült megfelelően beosztani, majd ahogy nő a minta, előjön 1-1 rosszul kiosztott műszak.

Meg kell jegyezni, hogy ebben a példában az igények teljesen véletlenszerűen lettek generálva, azaz a felhasználók *roster*-jein az *application* mező véletlenszerűen lett feltöltve, 0 és 1-es karakterek. Amennyiben ezek generálása úgy történik, hogy minden felhasználó pontosan 4 darab napra teszi le a jelentkezését, szabadságok nélkül (viszont a napokon véletlen, hogy egy vagy több műszakra), akkor a hiba változik (5.4 ábra). Természetes, így már jobban előjön, hogy a modell már nem tudja minden esetben teljesen kielégíteni a műszakokat, de így is jó eredményt ad.



5.3. ábra. Futásidőre és hibára vonatkozó eredmények véletlenszerű igények esetén.



5.4. ábra. Futásidőre és hibára vonatkozó eredmények pontosan 4 napra leadott igények esetén.

Betegség esetén, beosztás újragenerálást követően a táblázatban látható eredmények alapján jól látható, hogy többszöri újraszámolást követően is nagy pontossággal képes a modell megtartani az eredeti beosztást, mind a műszakokra, mind pedig a munka-, szabad- és tartalékos napokra. A futásidő hasonló az eredeti modell futásidejéhez. Érdekes még megnézni, hogy a modell hány felesleges feltételt dob el a számolás előtt. Ezekben az ese-

tekben véletlenszerű napokra történik a betegség igény leadása, olyan formában viszont, hogy az fedve legyen másik igényekkel, így biztos, hogy újra számolásra lesz szükség, tartalékos műszak hiányában.

	<b>1st Recalculate</b>	<b>2nd Recalculate</b>	<b>3rd Recalculate</b>
<b>Runtime</b>	395ms	422ms	449ms
<b>Constraints</b>	2049	2051	2036
<b>Presolved Constraints</b>	447	418	390
<b>Presolved Variables</b>	604	568	535
<b>Shift Match</b>	630/630	622/630	622/630
<b>Shift Match Percentage</b>	100%	98.7%	98.7%
<b>Workday Match</b>	206/210	207/210	203/210
<b>Workday Match Percentage</b>	98%	98.5%	96.6%
<b>Offday Match</b>	210/210	208/210	206/210
<b>Offday Match Percentage</b>	100%	99%	98%
<b>Reserve Day Match</b>	210/210	210/210	208/210
<b>Reserve Day Match Percentage</b>	100%	100%	99%

5.1. táblázat. Úrjaszámolás eredmények

## 6. fejezet

# Alkalmazás használata

Az alkalmazás parancssorból indítható el. Használatához egy legalább 3.11.5-ös Python [23] és 18.16-os verziójú node.js [15] szükséges. A Django-nál 5.0.4-es [5], React-nál pedig 18.2.0-ás verzió [25] van használatban.

Letöltés és kicsomagolás után egy virtuális környezetben a requirements.txt fájlban megtalálható könyvtárakat a `pip install -r requirements.txt` paranccsal kell telepíteni, majd elindítani a backendet és a frontendet. Ezek a `python $PATH/manage.py runserver`, illetve az `npm run dev` parancsok kiadásával tehető meg.

## 7. fejezet

### Kitekintés

TDK dolgozatom célja egy alkalmazás felépítése volt, mely képes segíteni beosztások tervezésében egy matematikai modell segítségével, a felhasználók interakcióit pedig egy LLM kezeli.

Az alkalmazás két modell köré lett felépítve. Egy LLM, azaz nagy nyelvi modell és egy matematikai modell köré lett felhúzva egy felhasználói felület, melyet egy backend üzemeltet, szolgáltatja az információt. A felhasználók képesek igényeket leadni a jövőbeli beosztásuk kialakítása érdekében a nap három műszakjára, ezt egy emberi nyelven elküldött üzenet segítségével. Ha változtatni szeretnének, megtehetik azt minden hét péntek éjfélig. Amennyiben ki szeretnék használni az éves szabadságukat, úgy azokra is tehetnek igényt. Ezeket a beérkező kéréseket az OpenAI és a LangChain dolgozza fel és alakítja át többségében JSON formátumra, amiken az implementált függvények további műveleteket végeznek. Ezeknek kimenetelét a nyelvi modell feldolgozza, majd emberi szöveggé konvertálva visszaküldi a felhasználó felé, tájékoztatva őt az elkezdett folyamat sikerességéről, majd megjelenítve a UI-on. A felhasználók felettesei, az adminisztrátorok ellenőrzik a leadott igényeket, módosítják, ha szükség van rá. Amennyiben rendben látják azokat, úgy a matematikai modellt elindítva, az megoldja az adatbázisból kinyert adatok és előre megtervezett korlátozó feltételek alapján létrejött lineáris programozási modellt. Betegség esetén az alkalmazás először algoritmikusan próbál helyettesítést találni a tartalékos műszakok közül. Amennyiben erre már nincs lehetőség, mert egy napon minden tartalékos be lett hívva, úgy a modell először szabadnapos felhasználót hív be, majd a szabályok betartása érdekében újra generálja a hét hátralévő részéhez tartozó műszakokat. Ezt próbálja a lehető legkisebb változtatással megtenni. A matematikai modellt további feltételekkel lehet kiegészíteni, a chat asszisztentst pedig erősebb nyelvi modellel hatékonyabbá, robosztusabbá lehet tenni.

# Irodalomjegyzék

- [1] Jay Alammar. Gpt-2 detailed model architecture, 2019. Accessed: 2024-11-18.
- [2] Fawad Ali. Gpt-1 to gpt-4: Each of openai's gpt models explained and compared, 2023. Accessed: 2024-11-12.
- [3] Elena Canorea. Gpt-4: A complete guide to understanding its functionalities, 2024. Accessed: 2024-11-18.
- [4] Google Developers. Google optimization tools documentation, 2024. Accessed: 2024-11-12.
- [5] Django Software Foundation. Django 5.0.4 release notes, 2024. Accessed: 2024-11-18.
- [6] Vyacheslav Efimov. Large language models, gpt-1 — generative pre-trained transformer, 2024. Accessed: 2024-11-18.
- [7] Louie Giray. Prompt engineering with chatgpt: A guide for academic writers. *Annals of Biomedical Engineering*, 2023.
- [8] Fred Glover and Eugene Woolsey. Further reduction of zero-one polynomial programming problems to zero-one linear programming problems. *Operations Research*, 21(1):156–161, February 1973.
- [9] Shi Qiuc Humza Naveeda, Asad Ullah Khanb. A comprehensive overview of large language models. *The Mathematical Gazette*, 2024.
- [10] hwchase17. Simple chatbot agent prompt on langchain hub, 2024. Accessed: 2024-11-12.
- [11] IBM. Langchain: A topic overview, 2024. Accessed: 2024-11-17.
- [12] IBM. Prompt engineering, 2024. Accessed: 2024-11-12.
- [13] Dinesh Kalla. Study and analysis of chat gpt and its impact on different fields of study. *International Journal of Innovative Science and Research Technology*, 2023.
- [14] LangChain. Langchain documentation, 2024. Accessed: 2024-11-12.



- [15] Node.js Project. Node.js v18.16.0 released, 2023. Accessed: 2024-11-17.
- [16] OpenAI. Chatgpt, 2024. Accessed: 2024-11-12.
- [17] OpenAI. Gpt-4 technical report. 2024.
- [18] OpenAI. Openai model documentation, gpt-4-turbo, 2024. Accessed: 2024-11-12.
- [19] OpenAI. Openai model documentation, gpt-4o, 2024. Accessed: 2024-11-12.
- [20] OpenAI. Openai model documentation, gpt-4o-mini, 2024. Accessed: 2024-11-12.
- [21] OpenAI. Openai model documentation, modells, 2024. Accessed: 2024-11-12.
- [22] PromptEngineering.org. Exploring agentic and wagentic workflows: The power of ai agent collaboration, 2024. Accessed: 2024-11-12.
- [23] Python Software Foundation. Python documentation, version 3.11.5, 2023. Accessed: 2024-11-17.
- [24] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [25] React Team. React 18.2.0 - june 14, 2022, 2022. Accessed: 2024-11-18.
- [26] Sciforce. What is gpt-3, how does it work, and what does it actually do?, 2021. Accessed: 2024-11-18.
- [27] et al. Suresh Bolusani, Mathieu Besancon. The scip optimization suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin, February 2024.
- [28] Mathieu Besancon et al. Suresh Bolusani. The scip optimization suite 9.0. Technical report, Optimization Online, February 2024.
- [29] Benjamin Mann et al Tom B. Brown. Language models are few-shot learners. 2020.
- [30] Yanan Zheng Xiao Liu. Gpt understands, too. *KeAi*, 2023.

## **Mesterséges intelligencia használata**

A dolgozat 3.2-es pontjának első bekezdése, valamint az implementált kódban a függvénydokumentációk a ChatGPT segítségével készültek.