



Rendszerterv

Teljesen elosztott rendszer elkészítése nagy méretű
felvonórendszer központi monitorozására és irányítására

Szoftverarchitektúrák házi feladat

Készítették:

Hajnal Árpád Erik - XXXXXXX

Jankó András - XXXXXXX

Szénási Krisztián - XXXXXXX

Tarcza Lídia - XXXXXXX

Tartalom

Tartalom.....	2
A rendszer célja, funkciói és környezete	4
Feladatkiírás.....	4
A rendszer által biztosítandó funkciók	4
A program környezete.....	5
Megvalósítás.....	7
Architektúra	7
Fő komponens.....	7
Adatbázis.....	7
Back end/ DAL és BBL	8
Front end/ Grafikus felhasználói felület.....	9
Sí liftek.....	11
Sí liftek cli felülete.....	14
Kommunikáció komponensek közt.....	20
Síliftek által küldött üzenetek.....	20
Síliftek által fogadott üzenetek.....	25
Konténerizáció és orkesztráció	26
Docker Compose alapú fejlesztői környezet.....	26
Frontend szolgáltatás	26
Backend szolgáltatás.....	26
Sífelvonó.....	27
RabbitMQ.....	27
Keycloak.....	27
Kubernetes/Minikube környezet	27
ConfigMap.....	28
Perzisztens tárolók (PVC-k)	28
Frontend szolgáltatás	28
Backend szolgáltatás.....	29
RabbitMQ szolgáltatás	29

Keycloak szolgáltatás	30
PostgreSQL adatbázisok	31
Sífelvonó szolgáltatások	31
Telepítési leírás	32
Rendszerkövetelmények	32
Telepítési útmutató	32
Fontos megjegyzések	33
Támogatott platformok	33
A program készítése során felhasznált eszközök	34
Összefoglalás	35
Továbbfejlesztési lehetőségek	36
Hivatkozások	37

A rendszer célja, funkciói és környezete

Feladatkiírás

A feladat egy teljesen elosztott rendszer elkészítése, mely lehetővé teszi egy nagy méretű felvonórendszer központi monitorozását és irányítását.

- Egy központi adminisztrációs felületről lehessen követni a felvonók állapotát, terheltségét, várható sorbanállási időt, szélerősséget stb.
- A felvonóknak jelezniük kell az általuk észlelt hibákat, vészhelyzeteket a központi komponens felé.
- A központból lehessen utasításokat küldeni a felvonók operátorai felé (lift indítás, leállítás stb.) A központ folyamatosan monitorozza a felvonókat működtető komponenseket és hiba esetén jelezze ezt az operátorok felé.
- Vészhelyzet esetén lehessen megbízhatóan leállítani egy felvonót a központi felületről.
- Egy külön komponens jelenítse meg a nagyközönség számára a felvonók egyszerűsített állapotát térképen.
- A rendszer komponensei elosztott módon kerüljenek megvalósításra.
- A rendszer kritikus részei redundánsan legyenek megvalósítva.
- A komponensek kommunikációja legyen megbízható módon megvalósítva.
- A liftek terheltségének változása egy egyszerű sorbanállásos modellel legyen közelítve.

A rendszer által biztosítandó funkciók

Az alkalmazás képes egy síparadicsomban (vagy akár bármely más helyen, ahol használnak felvonókat) működő felvonók monitorozására, és irányítására. Az alkalmazás elosztott módon lett implementálva, azaz az egyes komponensek külön alkalmazásként is működnek, így lehetőség nyílik az alkalmazás több gépen való futtatására és skálázására. A rendszer központi komponensének állapotát egy bizonyos intervallumonként lementjük, és bármilyen okból az meghibásodna, akkor azt a Kubernetes klaszter azonnal újraindítja, és a legutolsó mentéstől indul újra a működés. Az alkalmazás központi komponense mellett minden felvonóhoz tartoznak Worker komponensek, amelyek az egyes felvonók monitorozását, és irányítását is végzik. Az egyes síliftek jelenlegi állapotát minden állapotváltozásnál lementjük, legyen az a központ által kért állapotváltozás vagy a Worker operátor által kiadott parancs, ezzel biztosítva azt, hogy ne történjen inkonzisztencia a központi elem leállása esetén.

A felvonók jelzik az általuk észlelt hibákat, vészhelyzeteket a központi komponens felé. A központ is folyamatosan monitorozza a felvonókat működtető komponenseket és hiba esetén jelzi ezt az operátorok felé. (Például kiugró vagy hibás értékek)

A síliftek a következő adatokat küldik magukról:

- Jelenlegi állapot (FULL STEAM, HALF STEAM vagy STOP)
- Terheltség [db ember]
- Várható sorbanállási idő
- Szél erősség (kezdőpont és végpont)
- Hőmérséklet (kezdőpont és végpont)

A központból javaslatokat/figyelmeztetéseket lehet küldeni a felvonók felé (FULL STEAM, HALF STEAM vagy STOP). Ekkor a felvonónál lévő operátor dönt, hogy jogos-e a központ által adott parancs, és ha nem ért egyet felülríthatja. (2 szintű döntéshozatal/Failsafe) Vészhelyzet esetén a központ közvetlenül is küldhet egy emergency stop parancsot a felvonónak, amit a sílift operátora nem tud felülbírálni. Az emergency stop parancs kiadásához egy külön gomb áll rendelkezésre a központnak, aminek megnyomása után, meg is kell erősíteni a megállítási szándékot.

A Master és Worker komponensekbe egyaránt csak autentikáció után lehet belépni. Lehetőség van a központi komponens hétköznapi felhasználó szintű elérésére is, amelynek során a felhasználó egy olyan nézetet lát, amin keresztül tájékozódhat a liftek elhelyezkedéséről, és terheltségéről. Itt látja majd a sítérp térképét, az egyes felvonókat jól láthatóan kiemelve rajta, és a várható sorbanállási időt minden egyes felvonónál, mindezt egy felhasználóbarát felületen. Ebben a nézetben természetesen nincs lehetőség parancsok kiadására a liftek felé.

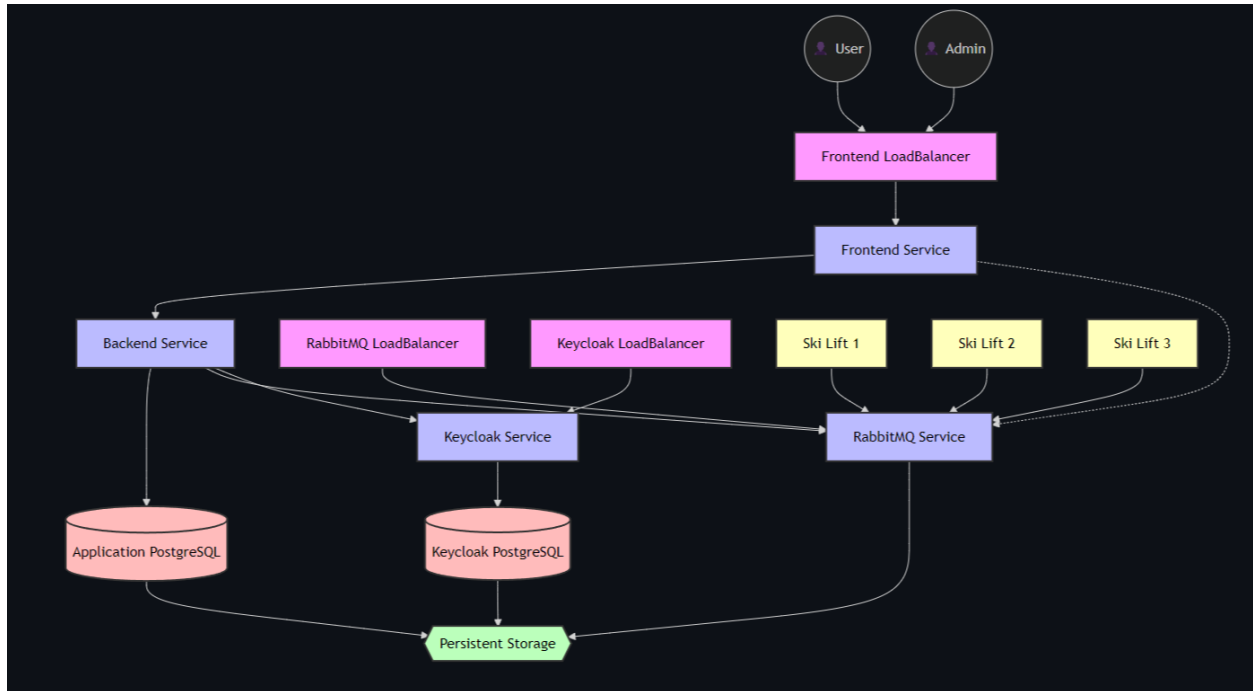
A komponensek UID segítségével azonosítják egymást, és a Master csak beregisztrált komponensektől fogad el üzeneteket. A komponensek közötti kommunikációt AMQP protokoll segítségével valósítjuk meg, ami garantálja a FIFO sorrendiséget az egyes liftenként küldött üzeneteken belül, illetve minden üzenetet titkosítva küldünk át. A liftek terheltségének változása egy M/M/c Queue (Erlang-C) modellel van közelítve. Mivel a jelenlegi megoldással a RabbitMQ queue könnyen válhat "single point of failure"-é, különös prioritást kapott a redundanciában. A megbízhatóság és hibatűrés érdekében kulcsfontosságú a RabbitMQ megfelelő klaszterkonfigurációja, a magas rendelkezésre állás biztosítása, és az üzenetek perzisztenciája.

A program környezete

Az alkalmazást elosztott módon készítettük el, ezáltal az egyes komponensek külön alkalmazásként is működnek, így lehetőség nyílik az alkalmazás több gépen való futtatására és skálázására. A program fő komponensének adatbázisához Postgres adatbázist használtunk, hozzá a backend rétegnek SpringBootot, a UI pedig Angularral készült. A szükséges felhasználói autentikációra Keycloak-ot használtunk.

A síliftek rendszere és a hozzá tartozó CLI interfész pythonban íródott. A komponensek közötti kommunikációhoz AMQP-t használtunk, amiért a RabbitMQ szoftver felelt.

A rendszer modellezése érdekében, a síparadicsom egy Kubernetes klaszterben készült el, ezen belül az egyes komponensek külön-külön Docker konténerek formájában helyezkedtek el, amihez Minikube-ot használtunk.



0.1. ábra Rendszer architektúrája

Megvalósítás

Az alkalmazást a feladatkiírásnak megfelelően egy több komponensből álló alkalmazásként készítettük el. A főkomponens egy vékonykliens alkalmazás valósít meg, ez kezeli a publikus nézetet, valamint az fő operátor felületet. Emellett készültek mellék komponensek is, amelyek a különböző síliftek irányításáért felelősek.

Az Alkalmazásunk a SnowPeak nevet kapta, mivel síliftek monitorozására és irányítására szolgál. Az ezen lifteket használó síparadicsomok pedig a havas hegyeken, sokszor azok csúcsa környékén helyezkednek el.

A fejezetben áttekintést adunk a program architektúrájáról, bemutatjuk az egyes komponensek feladatait és felelősségeit, továbbá részletesen ismertetjük a használt adatmodellt és a grafikus felhasználói felület felépítését

Architektúra

A SnowPeak architektúrája két nagyobb komponensre osztható: Fő komponens, és síliftek. A fő komponens pedig további rétegekre osztható:

- Fő komponens
 - Adatbázis (Database Layer, DB)
 - Back end (Data Access Layer és Business Logic Layer egyben)
 - Felhasználói felület (Graphical User Interface, GUI)
- Sí liftek

Fő komponens

Adatbázis

Célja: perzisztens adattárolás megvalósítása a rendszerbe konfigurált liftek szenzor és log üzeneteinek naplózásához, illetve a frontend komponens kiszolgálása a liftek törzsadatairól. PostgreSQL adatbáziskezelő rendszert használtunk.

Az adatbázis táblák és paraméterek bemutatása:

1. LIFT tábla

- Elsődleges célja a sífelvonók alapadatainak tárolása
- Főbb mezők:
 - lift_id: Egyedi azonosító (UUID)
 - lift_start_* és lift_end_*: A felvonó kezdő- és végpontjának koordinátái (magasság, szélesség, hosszúság)
 - lift_num_seats: Ülések száma
 - lift_seat_capacity: Ülések kapacitása
 - lift_status: A felvonó állapota (alapértelmezett érték: 'PUBLIC')

- lift_master_operator_id: A felvonó fő üzemeltetőjének azonosítója
- 2. LOG tábla
 - A felvonókkal kapcsolatos események naplózására szolgál
 - Mezők:
 - log_id: A napló bejegyzés egyedi azonosítója (UUID)
 - log_payload: Az esemény leírása/tartalma (max 255 karakter)
 - log_time: Az esemény időpontja
 - log_lift_id: Külső kulcs a LIFT táblára - megmutatja melyik felvonóhoz tartozik az esemény
- 3. WORKER tábla
 - A felvonókon dolgozó személyzet nyilvántartása
 - Mezők:
 - worker_id: A dolgozó azonosítója (UUID)
 - worker_lift_id: Külső kulcs a LIFT táblára - megmutatja melyik felvonón dolgozik

Kapcsolatok:

- A LOG tábla a log_lift_id mezőn keresztül kapcsolódik a LIFT táblához (many-to-one)
- A WORKER tábla a worker_lift_id mezőn keresztül kapcsolódik a LIFT táblához (many-to-one)

Ez az adatbázis struktúra lehetővé teszi:

- A sífelvonók helyadatainak és kapacitásának nyilvántartását
- Az események/tevékenységek naplózását felvonónként
- A dolgozók és felvonók összerendelésének követését

Back end/ DAL és BBL

Célja: frontend komponens kiszolgálása az adatbázisból kiolvasott felvonó törzsadatokkal. A környezeti fájlok beolvasása, transzformálása és mentése az adatbázisba, az adatbázis LIFT táblájának karbantartása. RabbitMQ-ból érkező felvonó üzenetek fogadása és mentése az adatbázisba.

A megvalósításhoz Spring Boot keretrendszert használtunk. RESTful API-n keresztül kommunikálunk a klienssel és az API-ról részletes, olvasható dokumentációt biztosítunk Swagger segítségével, valamint az OpenAPI segítségével biztosítjuk a frontendnek az API generálásának lehetőségét. A Flyway nyújtotta szolgáltatásokkal adatbázis-migrációt valósítunk meg.

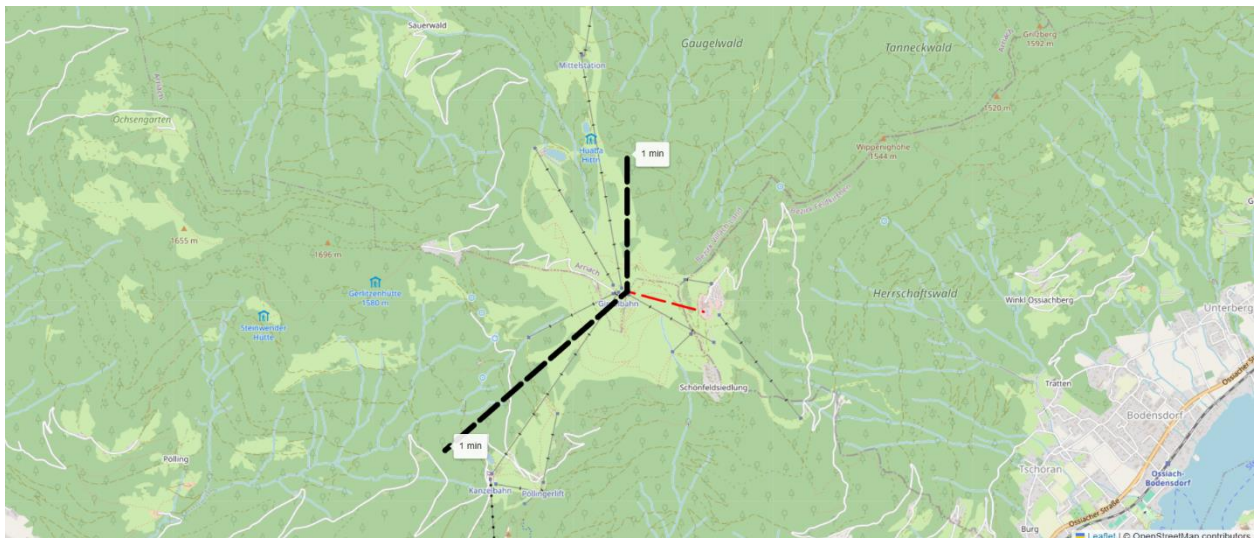
A back end felel a síliftektől érkezett logok eltárolásáért, illetve az időjárás adatokat is validálja. Ha hibás adat érkezik, ezt üzenetben jelzi az adott síliftnek.

Front end/ Grafikus felhasználói felület

Célja: grafikus felület biztosítása a nagyközönség számára a liftek elérhetőségéről, várakozási idejéről, valamint a master operátor adminisztrációs felületének implementálása.

A cél eléréséhez Angular keretrendszert használtunk. Az esztétikus és felhasználóbarát design elkészítésében az Angular Material volt a segítségünk. A Keycloak szolgáltatásaival biztosítottuk a megfelelő jogosultságkezelést. Websocket kapcsolatot alakítottunk ki a RabbitMQ-val a valós idejű kommunikációs megoldásáért. Az OpenStreetMap és a Leaflet javascript könyvtárak segítségével hoztuk létre a publikus térkép nézetet.

Képernyőfotók a frontend komponensről működés közben:



Publikus nézet

Belépés

Jelentkezzen be a felvonók kezeléséhez!

Operátor adminisztrációs felület bejelentkezés nélkül

Kilépés

Elérhető liftek:

Lift 1

Lift 2

✓ Lift 3

Hőmérséklet szenzor adatok:

Idő	Hely	Érték
2024-11-24T13:32:33.147957	peak	-19.96265192627068
2024-11-24T13:32:38.151378	base	-9.346761078125656
2024-11-24T13:32:38.152230	peak	-20.1138300302225


Szél szenzor adatok:

Idő	Hely	Érték
2024-11-24T13:32:33.148527	peak	21.092968552695012
2024-11-24T13:32:38.152733	base	12.603785670223195
2024-11-24T13:32:38.153078	peak	21.356332161124755

Parancsok:

Idő	Típus	Operátor	Eredmény	Egyéb
24T13:32:15.141834	remove_card	4013-93e5-55fe75528630	SUCCESSFUL	
2024-11-24T13:32:26.143661	insert_card		SUCCESSFUL	{ "cardInserted": "09397786-7bd6-4615-93e5-55fe75528630" }
2024-11-24T13:32:32.943008	change_state	09397786-7bd6-4615-93e5-55fe75528630	SUCCESSFUL	{ "newState": "FULL_STEAM" }

Publikus nézet:



Ütemezés

Teljes gőz

Üzenet

Küldés

VÉSZLEÁLLÁS

Operátor adminisztrációs felület bejelentkezett állapotban

Sí liftek

Célja: a felvonó komponens egy olyan elem a rendszerben, amely egy fizikai felvonót reprezentál. Kétirányú kommunikációt tesz lehetővé. Egyrészt fogadja az operátor által küldött vezérlő utasításokat, más részt visszajelzést ad a felvonó aktuális állapotáról és működéséről. Képes automatikusan feldolgozni és végrehajtani a távolról érkező parancsokat.

A komponens fontosabb részei a motor (*Engine*), utasítások (*Command*), vezérlő (*Controller*), hitelesítő (*Auth*), megfigyelő (*Monitor*), távoli (*Remote*), nézet (*View*), szenzor (*Sensor*) és matematika (*Math*) komponensek.

Motor (*Engine*)

A motor felelős a sílift aktuális állapotáért. Három féle állapotot képes tárolni. Ezek a *leállított (STOPPED)*, *fél gőz (HALF STEAM)* és *teljes gőz (FULL STEAM)* állapotok.

Utasítások (*Command*)

A síliften számos utasítást lehet végre hajtani. Minden egyes utasításhoz tartozik egy utasítás leíró (*CommandDescriptor*) és egy utasítás eredmény (*CommandResult*). A leíróból többek között kiderül maga az utasítás típusa, az utasítást kiadó felhasználó azonosítója és annak időpontja. Az eredményekből többek között kiderül milyen kimenete (*outcome*) volt az adott utasításnak. A kimenet lehet sikeres (*SUCCESSFUL*), sikertelen (*FAILED*) és késleltetett (*DELAYED*). Abban az esetben, ha az utasítás sikertelen volt, egy kivétel (*exception*) objektum is elérhető, amely tartalmazza sikertelenség okát. A késleltetett utasítások nevükből fakadóan később kerülnek végrehajtásra. Ilyenkor jelzés értékként egy késleltetett eredménye lesz az utasításnak, ami egyből kiolvasható.

Vezérlő (*Controller*)

A motor egy vezérlőn keresztül irányítható. Alapvetően utasítás leírókat képes feldolgozni. A sikeres utasítások végrehajtnak a motoron míg a sikertelenek nem változtatják meg a motor állapotát. A késleltetett utasítások a nevükből fakadóan a leíróban definiált idő elteltével fognak végrehajtásra kerülni. Egy vezérlő tartalmaz hitelesítőket, valamint monitorokat is. Ezen komponensek később kerülnek bemutatásra.

Hitelesítő (*Auth*)

A sílift komponens tartalmaz egy hitelesítőt (*Authenticator*) valamint egy felhatalmazót (*Authorizer*). Bizonyos utasítások csak mágneskártya behelyezésével hajthatók végre. Erről a hitelesítő gondoskodni. Kártyát igénylő utasítások esetén az vezérlő az utasítást először elküldi a hitelesítőnek. Abban az esetben, ha a behelyezett kártya helyes, a folyamat tovább csordul és az utasítás végrehajtnak a motoron, ellenkező esetben a folyamat megszakad és egy sikertelen eredmény kerül legyártásra.

Az felhatalmazó segítségével különböző jogokat lehet megkövetelni bizonyos utasítások végrehajtásához. Minden egyes utasítás külön-külön feldolgozható és eldönthető, hogy a végrehajtás engedélyezett-e vagy sem. Jelen esetben semmilyen extra jogkör nem létezik, de nem elrugaszkodott gondolat, hogy erre a jövőben szükség lehet, így ezzel a megvalósítással ilyesmi logika nagyon könnyedén bevezethető az alkalmazásba.

Összefoglalva a hitelesítő csak egy helyes mágneskártya meglétét ellenőrzi, míg a felhatalmazó a parancsokat külön-külön képes feldolgozni és eldönteni, hogy az adott utasítás engedélyezett-e vagy sem.

Megfigyelő (*Monitor*)

A megfigyelők olyan komponensek, melyek feliratkozhatnak egy vezérlőhöz és értesítést kaphatnak a kiadott utasításokról, valamint azok eredményeiről. Ilyen speciális monitor a loggoló (*BaseCommandLogger*). Ebből két félé konkrét változat létezik: a *fájl loggoló* (*FileCommandLogger*) és a *rabbit mq loggoló* (*RabbitMQLogger*). Az előbbi minden egyes utasítást egy fájlba ír ki lokálisan, míg az utóbbi pedig hálózaton keresztül *rabbit mq* segítségével küldi el a feliratkozott szereplőknek az utasításokat. Ezen komponensek segítségével egy esetleges szerencsétlenség esetén visszaolvasható a kiadott utasítások története, és eldönthető, hogy az adott operátor helyesen cselekedett-e.

Távoli (*Remote*)

A távoli komponens egy összefoglaló név, amely minden olyan komponenst tartalmaz, amelyek távoli kommunikációt valósítanak meg a központi irányító szobával. Ide tartoznak a *pika kliensek*. Két félé létezik: üzenet küldő (*PikaProducer*) és üzenet fogadó (*PikaConsumer*). Ezek a kliensek egyedi kapcsolatot valósítanak meg a *rabbit mq* brókerrel, amelyek esetleges kapcsolódási hibák esetén megkísérlik az újra kapcsolódást. Magukhoz a kliensek feliratkozhatnak kapcsolódási esemény megfigyelők (*ConnectionEventObserver*). Ezeknek a megfigyelőknek a kliensek értesítést küldenek különböző kapcsolódási eseményekről, így a felhasználó könnyedén tájékoztatható ezekről. Esetleges kapcsolódási hiba esetén az üzenet küldő ideiglenesen eltárolja az üzeneteket (*pending messages*) memóriában és újra kapcsolódás esetén elküldi őket így az üzenetek nem vesznek el még akkor sem, ha a *rabbit mq* bróker ideiglenesen meghibásodik.

Az általános log üzeneteken túlmenően, amik az előző szakaszban lettek ismertetve, lehetőség van igény szerinti (*on demand*) üzenetek küldésére is. Ezek egy távoli kommunikáló (*RemoteCommunicator*) segítségével lehetségesek, melynek egy konkrét megvalósítása az üzenet küldő *pika klienst* használja. Ilyen üzenet például a jelentés (*MessageReport*), melynek segítségével különböző eseményekről, vészhelyzetekről tudja figyelmeztetni az operátor a központot.

Szenzor (*Sensor*)

A szenzor komponensek felelősek az egyes szenzoradatok küldéséért, amit mi generálunk és egy általunk kreált eloszlást követ. Hőmérsékleti, és szélesség adatokat küldünk a felvonó aljáról (base), és tetejéről (peak). Factory és observer design patterneket használtunk ebben a komponensben, aminek segítségével akár többféle szenzorfajta hozzáadható a jövőben.

Matematika (*Math*)

Az ErlangCModel egy olyan osztály, amelyet a sílift-sorok dinamikájának szimulálására terveztünk az Erlang C sorbanállási modell segítségével. A kulcsfontosságú teljesítménymutatók kiszámításához figyelembe veszi a sílift különféle fizikai paramétereit (például a vonalsebességet, a hordozókapacitást és a lejtő geometriáját), valamint a működési tényezőket (például érkezési arányt és rakodási hatékonyságot). Az osztály módszereket biztosít a várakozási idők, a várólisták hosszának és a rendszer kihasználtságának becslésére, miközben figyelembe veszi a valós hatékonysági tényezőket és a minimális betöltési időt is. A `get_performance_metrics()` metóduson keresztül tudjuk meghívni és betekintést nyújt a felvonó működési teljesítményébe, beleértve az átlagos várakozási időt, a sorhelyigényt, valamint az elméleti és a tényleges kapacitást.

Nézet (*View*)

Egy nézet képes felhasználó inputokat fogadni, amiket a vezérlő felé továbbít. Ezenfelül visszajelzéseket is ad, így a felhasználó meggyőződhet a kiadott utasítások kimeneteléről. Ez úgy került megvalósításra, hogy a nézetek egyben eredmény monitorok is, így a vezérlő a többi monitorral együtt a nézetet is értesíti az utasítások eredményéről.

A sílift jelenleg egy parancs soros nézeten (*CommandLineInterfaceView*) keresztül vezérelhető. Itt szöveges utasítások hatására a megfelelő utasítás leírók kerülnek legyártásra, amelyeket a vezérlő fog feldolgozni.

A nézet egy kapcsolódási esemény megfigyelő is egyben, így a különböző kapcsolódási eseményekről értesítéseket kap, melyeket a felhasználónak vizuálisan megjeleníti.


```
> help

Available commands:

insert_card <card_id>
- Inserts a card into the system, using the specified <card_id>.

remove_card
- Removes the currently inserted card from the system.

change_state <state>
- Changes the engine state to the specified <state>.
- Valid states include MAX_STEAM, FULL_STEAM, HALF_STEAM, and STOPPED.

display_status
- Displays the current status of the lift system.

emergency_stop
- Stop the ski lift in a case of an emergency.

abort <command_id>
- Aborts a delayed command with the specified <command_id>.

report <severity> <message>
- Send a report to the central room.
- Possible severities are INFO, WARNING and DANGER.

suggestion_level <severity>
- Suggestions with severities equal to or greater than the selected level will be displayed.
- Order is INFO < WARNING < DANGER.
- If you set it to NONE, no suggestions will be displayed.

help
- Displays this help message listing all available commands.

> █
```

Kártya behelyezés

Kártyát az *insert_card* parancs segítségével lehet behelyezni. A rendszer jelzi a felhasználó számára, hogy a kártyát elfogadta-e vagy sem.

```
> insert_card wrong_card
Error: Card with number "wrong_card" was rejected!

> █
```

```
> insert_card secret
CARD ACCEPTED

> █
```

Kártya eltávolítása

Kártyát a *remove_card* parancs segítségével lehet eltávolítani.

```
> remove_card  
CARD REMOVED  
  
> █
```

Lift állapotának megváltoztatása

A lift állapotát a *change_state* parancs segítségével lehet megváltoztatni. A parancs neve után meg kell adni a kívánt állapot nevét. A lehetséges állapot nevek a következők: STOP (leállított), HALF_STEAM (fél gőz) és FULL_STEAM (teljes gőz). A parancs használata mágneskártyához kötött.

```
> change_state stop  
STATE CHANGED  
  
> change_state half_steam  
STATE CHANGED  
  
> change_state full_steam  
STATE CHANGED  
  
> █
```

A lift aktuális állapotának lekérdezése

A lift aktuális állapota a *display_status* parancs segítségével jeleníthető meg. A parancs használata mágneskártyához kötött.

```
> display_status  
EngineState.FULL_STEAM  
  
> █
```


Késleltetett parancs megszakítása

A lift képes parancsokat késleltetve feldolgozni. Abban az esetben, ha egy ilyen parancs a bekerül a rendszerbe a felhasználó figyelmeztetve lesz róla, valamint tájékoztatásként a szükséges utasítás is megjelenik, amivel a parancs megszakítható. A parancs használata mágneskártyához kötött.

```
> change_state full_steam 15
Type "abort 18" to abort it.

>
> abort 18
COMMAND 18 ABORTED

> █
```

Jelentés küldése

Az operátornak lehetőségük van jelentéseket küldeni a központi szobába. Magát a jelentést három kategóriába sorolhatják súlyosság szempontjából: INFO, WARNING és DANGER. A jelentést szabad szavasan fogalmazhatják meg. Mindezt a *report* parancs segítségével tehetik meg. Egy ilyen kiadott parancsnak két eredménye lehet. Sikeres elküldés esetén a MESSAGE SENT felirat jelenik meg, viszont, ha jelenleg kapcsolódási problémák állnak fent a központtal bármilyen okból kifolyólag, akkor a MESSAGE PENDING felirat fog megjelenni. Ezek az üzenetek abban az esetben fognak elküldésre kerülni, ha a kapcsolat helyre állt. A parancs használata mágneskártyához kötött.

```
> report WARNING The weather is getting worse.
REPORT SENT

> █
```

```
> report INFO Wind is better.
REPORT PENDING

> █
```

Javaslatok szűrése

A javaslatok, amelyek megjelennek a képernyőn három kategóriába sorolhatók: INFO, WARNING és DANGER. Az operátor szűrhet ezkere a javaslatokra a *suggestion_level* paranccsal. Az operátornak meg kell adnia a szintet, amire kíváncsi. A szintek nevei megegyeznek a javaslatok kategóriáinak neveivel. Abban az esetben, ha a választott szint INFO, minden javaslat meg fog

jelenni, WARNING esetén az INFO kivételével bármi és DANGER esetén pedig csak a DANGER üzenetek. Alapból a szint INFO-ra van állítva.

```
> suggestion_level WARNING
```

```
> █
```

Hibás parancsok

Nem létező parancs esetén a rendszer hibát dob és javaslatot tesz a felhasználónak, hogy használja a *help* parancsot.

```
> non_existent_command
Did not recognize "non_existent_command" command. Type "help" to display the available commands.
```

```
> █
```

Abban az esetben, ha létező parancsot próbál használni az operátor rosszul, a következő üzenet jelenik meg:

```
> report WRONG_CATEGORY This is the message.
Incorrect arguments for command "report". Type "help" to display correct usage.
```

```
> █
```

Kapcsolódási események

A távoli kommunikációhoz kapcsolatra van szükség a központtal. Két féle kommunikációs csatornát használ a rendszer. Az egyik a távoli javaslatok csatorna, a másik pedig a távoli vészleállítás csatorna. Ezekhez a csatornákhöz tartozó kapcsolódási eseményekről a felhasználó értesítéseket kap.

Sikeres kapcsolódás esetén a következő üzenetek jelennek meg:

```
[2024-11-23 20:40:32] [INFO] Connected to "direct_suggestions" channel.
[2024-11-23 20:40:32] [INFO] Connected to "direct_emergency_stop" channel.
```

```
>
```

```
>
```

Esetleges szétkapcsolás esetén a következő figyelmeztetések jelennek meg:

```
[2024-11-23 20:41:49] [WARNING] Connection to "direct_emergency_stop" was closed.
[2024-11-23 20:41:49] [WARNING] Connection to "direct_suggestions" was closed.
```

```
>
```

```
>
```

Maga a rendszer ilyenkor a háttérben periodikusan újra kapcsolódásokat kísérel meg. Abban az esetben, ha egy ilyen kapcsolódási próbálkozás sikertelen a felhasználó szintén

figyelmeztetve lesz. Fontos azonban, hogy ez a figyelmeztetés csak egyszer fog megtörténni, ellenkező esetben a felhasználót zavarhatja a sok üzenet és akadályozhatná a munkája elvégzésében.

```
[2024-11-23 20:41:49] [WARNING] Could not connect "direct_emergency_stop" channel.  
[2024-11-23 20:41:49] [WARNING] Could not connect "direct_suggestions" channel.  
>  
>
```

Távoli javaslatok

Az operátorok kaphatnak távoli javaslatok a központból. Ezek szintén három kategóriába sorolhatók: INFO, WARNING és DANGER. Maguk az üzenetek különböző színekben jelennek meg a kategóriától függően.

```
[2024-11-23 22:00:53] [INFO] This is a message.  
>  
[2024-11-23 22:00:58] [WARNING] This is a message.  
>  
[2024-11-23 22:01:03] [DANGER] This is a message.  
> █
```

Távoli vészleállítás

A központi operátoroknak lehetőségük van távoli vész leállítás parancsok kiadására. Egy vész leállítás tartalmaz egy magyarázatot, valamint egy késleltetést, aminek lejártá előtt liftnél tartózkodó operátor megszakíthatja a parancsot a már korábban ismertetett *abort* parancs segítségével. A felületen egy ilyen üzenet a következő módon jelenik meg:

```
>  
[2024-11-23 22:04:29] [DANGER] This is an emergency stop.  
Emergency stop in 15 seconds.  
Type "abort 1" to abort it.
```

Kommunikáció komponensek közt

A kommunikációt a komponensek között a RabbitMQ komponens segítségével oldjuk meg, amelyben exchangekkel, és topicokkal oldunk meg. Ezeket most részletesen ismertetjük:

Sílifttek által küldött üzenetek

exchange name: topic_skilift

exchange_type: topic

A kimenő üzeneteket a sífelvonók küldik. Minden útvonal általában a skilift.<lift_id> mintát követi, megkönnyítve az üzenetek adott sífelvonók szerinti szűrését. Minden üzenettípusnak megvan a maga egyéni elérési mintája., lehetővé téve az egyes típusok szerinti szűrést is. Ezenkívül az üzenet törzse a messageKind kulcs, amely segít gyorsan azonosítani az üzenettípusokat, különösen akkor, ha több egyidejűleg kerül feldolgozásra.

Ha az összes üzenetet megszeretnénk kapni, a skilift.# mintát használhatjuk. A location mindenhol mező vagy base vagy peak. Mindegyik üzenetben megtalálható a következő fejléc:

```
{  
  "lift_id": "liftabc123"  
}
```

Sensor Data

A sífelvonó rendszeres időközönként különféle típusú szenzoradatokat küld.

Wind, routing_key: skilift..logs.sensor.wind, payload (body):

```
{  
  "messageKind": "sensor"  
  "type": "wind"  
  "value": 12.3,  
  "timestamp": "2024-11-10T19:26:05.199209",  
  "location": "base"  
}  
Temperature  
routing_key: skilift..logs.sensor.temperature  
payload (body):  
{  
  "messageKind": "sensor"  
  "type": "temperature"  
  "value": -2.3,  
  "timestamp": "2024-11-10T19:26:05.199209",  
  "location": "base"  
}
```

Command Log

routing_key: skilift.logs.command

A sífelvonó CLI paneljén a kezelő által végrehajtott parancsok naplózásra és elemzésre kerülnek, amelyek hasznosak lehetnek az események, például a balesetek áttekintésénél.

7 különféle parancs létezik:

- insert_card
- remove_card
- change_state
- display_status
- abort_command
- emergency_stop
- message_report

Ezen parancsok mindegyike általában hasonló mintát követ az üzenetekben:

```
{
  "messageKind": "command",
  "type": <name of the command: str>,
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": <SUCCESSFUL | FAILED | DELAYED>,
  "exception": <exception name: str | null>,
  ["args": {...}]
}
```

Ha a parancs sikertelen, a kivétel tartalmazza a hibát; ellenkező esetben null lesz.

Az args szakasz csak az argumentumokat igénylő parancsokhoz használható.

A sikertelen parancsok értékes információkkal szolgálhatnak a mester operátorok számára. A sikeres és a sikertelen parancsok is szűrhetők a következőkkel: skilift.<skilift_id>.logs.command.successful vagy skilift.<skilift_id>.logs.command.failed.

Insert Card

Ez a parancs azt szimulálja mikor az operátor egy kártyát dug a terminálba.

Sikeres kimenet példa:

```
{
  "messageKind": "command",
  "type": "insert_card",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": null,
  "outcome": "SUCCESSFUL",
  "exception": null,
  "args": {
    "card_inserted": "abc123"
  }
}
```

A hibás kimenetnél értelemszerűen az outcome FAILED, és kapunk egy exceptiont is.

Remove Card

Ez a parancs azt szimulálja mikor az operátor kiveszi a kártyát a terminálból.

```
{
  "messageKind": "command",
  "type": "remove_card",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": "SUCCESSFUL",
  "exception": null
}
```

Change State

A `change_state` parancs a sífelvonó állapotának módosítására szolgál. A parancs sikeres végrehajtásához egy elfogadott kártyát kell behelyezni a panelbe.

A lehetséges állapotok: `FULL_STEAM` | `HALF_STEAM` | `STOPPED`

```
{
  "messageKind": "command",
  "type": "change_state",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": "SUCCESSFUL",
  "exception": null,
  "args": {
    "new_state": "FULL_STEAM"
  }
}
```

Display Status

Kiírja a sílift jelenlegi állapotát.

```
{
  "messageKind": "command",
  "type": "display_status",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": "SUCCESSFUL",
  "exception": null
}
```

Abort Command

Az `abort_command` az időablak lejárta előtt távolról elküldött vészleállítási parancsok törlésére szolgál. A parancshoz szükség van egy `commandtoabort` nevű argumentumra, amely egy egyszerű egész szám, amely a törölni kívánt parancs azonosítására szolgál. Fontos megjegyezni, hogy ez az azonosító kizárólag a sílift-komponensen belül használatos, és csak az adott sífelvonó egyediségét garantálja az aktuális munkamenet során.

```
{
  "messageKind": "command",
  "type": "abort_command",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": "SUCCESSFUL",
  "exception": null,
  "args": {
    "command_to_abort": 1
  }
}
```

Emergency Stop

Az `emergency_stop` parancs a sífelvonó vészhelyzetben történő leállítására szolgál. Ez a parancs nem igényel hitelesítést, így behelyezett kártya nélkül is végrehajtható.

```
{
  "messageKind": "command",
  "type": "emergency_stop",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": "SUCCESSFUL",
  "exception": null,
}
```

Message Report

A `message_report` paranccsal üzeneteket küldhetünk a központi műtőbe. A parancs eredménye egy távoli üzenet, amelyet a következő részben ismertetünk. A parancsnak két argumentuma van: `severity`, amely az üzenet súlyosságát jelzi és maga az üzenet.

A severity állapotai: INFO | DANGER | WARNING

```
{
  "messageKind": "command",
  "type": "message_report",
  "timestamp": "2024-11-10T19:26:05.199209",
  "user": "abc123",
  "outcome": "SUCCESSFUL",
  "exception": null,
  "args": {
    "severity": "DANGER",
    "message": "a kid fell off:("
  }
}
```


Síliftek által fogadott üzenetek

Ha üzeneteket szeretnénk küldeni egy adott sífelvonónak, minden bejövő üzenethez közvetlen üzenetváltás jön létre. A sífelvonók a soraikat a megfelelő közvetlen központokhoz kötik, az egyedi skilift_id-t használva routing_key-ként. Ezzel a megközelítéssel, feltételezve, hogy a kapcsolat és a csatorna már be van állítva, és a megfelelő központ létezik, közvetlen üzenetek küldhetők a route_key beállításával a kívánt sílift azonosítójára.

- var skiLiftOneId = "123abc"
- channel.publish(exchange, skiLiftOneId, Buffer.from("Hi ski lift 1!"))
- Suggestions
- exhchange name: direct_suggestion
- exhchange_type: direct
- routing_key: <lift_id>

A javaslatok felhasználhatók ajánlások küldésére a sífelvonók üzemeltetőinek. Ezek a javaslatok megjelennek a kezelő képernyőjén. A sífelvonók az üzenetek szövegét a következő formátumban várják:

```
{
  "messageKind": "suggestion"
  "user": "abc123",
  "severity": "INFO",
  "timestamp": "2024-11-10T19:26:05.199209",
  "message": "Something less important."
}
```

Emergency Stop

exhchange name: direct_emergency_stop , exhchange_type: direct, routing_key: <lift_id>

A vészleállítók segítségével távolról is meg lehet állítani egy sífelvonót. Leírható a leállítás indoklása, valamint egy késleltetési időszak (másodpercben), amely alatt a kezelő a stop paranccsal törölheti a leállítást.

A sífelvonók az üzenetek szövegét a következő formátumban várják:

```
{
  "user": "abc123",
  "timestamp": "2024-11-10T19:26:05.199209",
  "message": "problem",
  "abortTime": 15
}
```

Konténerizáció és orkesztráció

A rendszert kétféle környezetben teszteltük: egy lokális fejlesztői környezetben Docker Compose segítségével, és egy Minikube klaszteren, amihez egy manifest fájlt írtunk, ennek a segítségével egy paranccsal el tudjuk indítani az alkalmazást. A rendszer összes általunk írt komponensét (frontend, backend, síliftek) dockerizáltuk, és feltöltöttük Dockerhubra.

Docker Compose alapú fejlesztői környezet

A következőkben ismertetjük a docker-compose.yaml fájlban lévő főbb komponenseket.

Frontend szolgáltatás

Angular alapú webalkalmazás Nginx webserverral és környezeti változókkal konfigurálható backend kapcsolatokkal:

```
frontend:
  image: frontend
  container_name: frontend
  build:
    context: ./client
    args:
      - ANGULAR_PORT=${ANGULAR_PORT}
  ports:
    - ${ANGULAR_PORT}:${ANGULAR_PORT}
  environment:
    BACKEND_URL: ${BACKEND_URL}
    RABBITMQ_WS_URL: ${RABBITMQ_WS_URL}
  volumes:
    - ./client/nginx.conf:/etc/nginx/nginx.conf:ro
  ...
```

Backend szolgáltatás

Spring Boot alkalmazás integrált Keycloak és RabbitMQ kapcsolattal:

```
backend:
  image: backend
  build:
    context: ./backend
  environment:
    SPRING_PROFILES_ACTIVE: ${SPRING_PROFILES_ACTIVE}
    SPRING_DATASOURCE_URL: ${SPRING_DATASOURCE_URL}
    KEYCLOAK_AUTH_SERVER_URL: ${KEYCLOAK_AUTH_SERVER_URL}
    SPRING_RABBITMQ_HOST: ${RABBITMQ_HOST}
  ...
```

Sífelvonó

```
ski-lift-1:
  image: krisztianszenasi/ski-lift:latest
  environment:
    LIFT_ID: ${LIFT_1_ID}
    START_LAT: ${LIFT_1_START_LAT}
    START_LON: ${LIFT_1_START_LON}
    LINE_SPEED: ${LIFT_1_LINE_SPEED}
    CARRIER_CAPACITY: ${LIFT_1_CARRIER_CAPACITY}
  ...
```

RabbitMQ

Menedzsment felülettel és WebSocket támogatással:

```
rabbitmq:
  image: rabbitmq:3-management
  hostname: rabbitmq
  ports:
    - "${RABBITMQ_AMQP_PORT}:${RABBITMQ_AMQP_PORT}"
    - "${RABBITMQ_MANAGEMENT_PORT}:${RABBITMQ_MANAGEMENT_PORT}"
    - "${RABBITMQ_WEBSOCKET_PORT}:${RABBITMQ_WEBSOCKET_PORT}"
  volumes:
    - rabbitmq_data:/var/lib/rabbitmq
    - rabbitmq_logs:/var/log/rabbitmq
```

Keycloak

Dedikált PostgreSQL adatbázissal:

```
keycloak:
  environment:
    KC_DB: ${KC_DB}
    KC_DB_URL: ${KC_DB_URL}
    KEYCLOAK_ADMIN: ${KEYCLOAK_ADMIN}
    KEYCLOAK_ADMIN_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD}
```

Kubernetes/Minikube környezet

A következőkben ismertetjük a manifest_prod.yaml fájlban található főbb komponenseket és konfigurációkat.

ConfigMap

Az alkalmazás konfigurációs értékeit tartalmazza, elsősorban a sífelvonók beállításait:

```
kind: ConfigMap
metadata:
  name: app-config
data:
  RABBITMQ_HOST: "rabbitmq"
  RABBITMQ_PORT: "5672"
  LIFT_1_ID: "e93206d2-e357-4d2d-a43b-c194744dac9a"
  LIFT_1_START_LAT: "46.69539731050563"
  LIFT_1_START_LON: "13.914595364129971"
  ...
```

Perzisztens tárolók (PVC-k)

Az alkalmazás állandó tárolóinak definíciói:

```
kind: PersistentVolumeClaim
metadata:
  name: rabbitmq-data-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi ...
```

Frontend szolgáltatás

Az alkalmazás Angular alapú webalkalmazása Nginx webszerverrel:

```
kind: Service
metadata:
  name: frontend
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: frontend
---
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  template:
```

```

spec:
  containers:
    - name: frontend
      image: bambika/frontend:latest
    env:
      - name: BACKEND_URL
        value: "http://backend:8080"
      - name: RABBITMQ_WS_URL
        value: "ws://rabbitmq:15674/ws" ...

```

Backend szolgáltatás

Spring Boot alkalmazás Keycloak és adatbázis integrációval:

```

kind: Service
metadata:
  name: backend
spec:
  ports:
    - port: 8080
      targetPort: 8080
---
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  template:
    spec:
      containers:
        - name: backend
          image: bambika/backend:latest
          env:
            - name: SPRING_PROFILES_ACTIVE
              value: "docker"
            - name: SPRING_DATASOURCE_URL
              value: "jdbc:postgresql://db:5432/postgres" ...

```

RabbitMQ szolgáltatás

```

kind: Service
metadata:
  name: rabbitmq
spec:
  type: LoadBalancer
  ports:
    - name: amqp
      port: 5672
    - name: management

```

```

        port: 15672
      - name: websocket
        port: 15674
    ---
  kind: Deployment
  metadata:
    name: rabbitmq
  spec:
    replicas: 1
    template:
      spec:
        containers:
          - name: rabbitmq
            image: rabbitmq:3-management
            volumeMounts:
              - name: rabbitmq-data
                mountPath: /var/lib/rabbitmq
              - name: rabbitmq-logs
                mountPath: /var/log/rabbitmq ...

```

Keycloak szolgáltatás

Itt a docker-compose-al ellentétben egy online imaget töltünk be.

```

  kind: Service
  metadata:
    name: keycloak
  spec:
    type: LoadBalancer
    ports:
      - port: 9090
        targetPort: 8080
    ---
  kind: Deployment
  metadata:
    name: keycloak
  spec:
    replicas: 1
    template:
      spec:
        containers:
          - name: keycloak
            image: quay.io/keycloak/keycloak:22.0.5
            args: ["start-dev", "--import-realm"]
            env:
              - name: KC_DB
                value: "postgres"
              - name: KC_DB_URL

```

```
value: "jdbc:postgresql://keycloak-postgres:5432/keycloak"
```

```
...
```

PostgreSQL adatbázisok

```
kind: Service
metadata:
  name: db
spec:
  ports:
    - port: 5432
```

```
---
```

```
kind: Deployment
metadata:
  name: db
spec:
```

```
  containers:
    - name: postgres
      image: postgres:14.1-alpine
  ... + a keycloak egy hasonló adatbázis konténert használ.
```

Sífelvonó szolgáltatások

Három független sífelvonó példány, azonos konfigurációs struktúrával:

```
kind: Deployment
metadata:
  name: ski-lift-1
spec:
  replicas: 1
  template:
    spec:
      containers:
        - name: ski-lift
          image: krisztianszenasi/ski-lift:latest
          env:
            - name: LIFT_ID
              valueFrom:
                configMapKeyRef:
                  name: app-config
                  key: LIFT_1_ID
  ... +további környezeti változók, amiket a ConfigMap-ban adunk meg.
```

Telepítési leírás

A SnowPeak rendszer fejlesztése során kiemelt figyelmet fordítottunk a telepítés és üzemeltetés egyszerűségére, miközben biztosítjuk a komplex elosztott rendszer megbízható működését. A szoftvert úgy terveztük, hogy minimális konfigurációval is üzembe helyezhető legyen, ugyanakkor megfeleljen a modern konténerizált környezetek követelményeinek.

Rendszerkövetelmények

A rendszer futtatásához az alábbi komponensek szükségesek:

- Kubernetes klaszter vagy Minikube (minimum 1.20-as verzió)
 - Minimum 2 CPU mag
 - Legalább 4 GB RAM
 - Minimum 20 GB szabad lemezterület
- Konténer futtató környezet (ajánlott: Docker)
 - Docker Engine 20.10.0 vagy újabb verzió
- kubectl parancssori eszköz
- Operációs rendszer:
 - Windows 10/11 (64-bit)
 - vagy bármely modern Unix alapú rendszer (Ubuntu 20.04+, macOS 10.15+)

Telepítési útmutató

A telepítés egyszerűen elvégezhető az alábbi lépések követésével:

1. Klónozza le a projekt repository-ját:

```
git clone https://github.com/Pillangocska/SnowPeak.git
```

```
cd snowpeak
```

2. A telepítéshez használja az operációs rendszerének megfelelő telepítő szkriptet:

Windows rendszeren:

```
.\k8s\start_on_windows.ps1
```

Linux/macOS rendszeren:

```
./k8s/start_on_unix_based.sh
```

Docker Compose használatával:

```
docker-compose up -d --build
```


A telepítő szkriptek automatikusan létrehozzák a szükséges konténereket, konfigurálják a hálózati beállításokat és elindítják a szolgáltatásokat. Az első indításkor a rendszer automatikusan inicializálja az adatbázist és létrehozza az alapértelmezett konfigurációs fájlokat.

Fontos megjegyzések

- A rendszer első indítása során ellenőrizni kell, hogy a szükséges portok (8080, 8443, 9090) nem foglaltak-e más alkalmazások által.
- Windows környezetben meg kell győződni meg róla, hogy a Hyper-V vagy WSL2 megfelelően van konfigurálva.
- Unix alapú rendszereken ellenőrizni kell, hogy a telepítő szkriptnek megfelelő futtatási jogosultságai vannak-e (chmod +x).
- A rendszer tesztelése során javasolt a mellékelt példa konfigurációs fájlok használata.

Támogatott platformok

A rendszert az alábbi környezetekben teszteltük:

- Windows 11 Pro (64-bit)
- Ubuntu 22.04 LTS
- macOS Sequoia (15.0+)

Más Linux disztribúciókon való futtatás is lehetséges, de ezeken a platformokon külön tesztelést igényelhet.

A program készítése során felhasznált eszközök

GitHub: forráskód verziókezelésére és csapatmunka koordinálására.

Visual Studio Code: általános célú, lightweight forráskód szerkesztő és fejlesztőkörnyezet.

IntelliJ IDEA: Java alapú alkalmazások fejlesztésére szolgáló integrált fejlesztőkörnyezet.

Miniconda [1]: Python környezetek és csomagok kezelésére szolgáló disztribúció.

Pyenv [2]: különböző Python csomagok telepítésére és kezelésére.

Microsoft Word: dokumentáció és felhasználói kézikönyv készítésére.

PostgreSQL [3]: relációs adatbázis-kezelő rendszer az alkalmazás adatainak tárolására.

Keycloak [4]: az alkalmazásban lévő Auth folyamatokat implementáltuk a segítségével.

Spring Boot [5]: Java alapú alkalmazások és microservicek fejlesztésére szolgáló keretrendszer, amelyet a backenden használtunk.

Angular [6]: modern, komponens alapú felhasználói felületek fejlesztésére szolgáló keretrendszer, amit a frontenden használtunk.

Docker: az alkalmazás konténerizált környezetben történő futtatására és terjesztésére.

Docker-compose [7]: több konténerből álló alkalmazások konfigurálására és orkesztrálása, amelyet lokális fejlesztéshez használtunk.

Minikube [8]: lokális Kubernetes környezet létrehozására és tesztelésére fejlesztés során.

Összefoglalás

Munkánk során megterveztük, implementáltuk és dokumentáltuk a SnowPeak nevű valós idejű sífelvonó felügyeleti rendszert. Az elkészített elosztott alkalmazás segítségével sífelvonók működését monitorozhatjuk és vezérelhetjük központosított módon.

A megvalósított rendszer modern microservice architektúrát használ, amely több rétegből épül fel: frontend szolgáltatás, backend szolgáltatás, üzenetkezelő rendszer, autentikációs szolgáltatás és adattárolási réteg. Az alkalmazás az adatokat PostgreSQL adatbázisban tárolja, míg a valós idejű kommunikációt RabbitMQ üzenetközvetítő rendszer biztosítja. A felhasználók azonosítását és jogosultságkezelését Keycloak szolgáltatás végzi, amely külön adatbázissal rendelkezik.

Az alkalmazásunk képes több sífelvonó egyidejű felügyeletére, valós idejű adatgyűjtésre és beavatkozásra. A rendszer magas rendelkezésre állást biztosít a kritikus komponensek redundáns kialakításával. A felhasználói felület modern webes technológiákkal készült, amely platformfüggetlen hozzáférést tesz lehetővé.

Munkánk során részletes rendszertervet készítettünk – amely jelen dokumentáció részét képezi – és jelentős mennyiségű implementációs munkát végeztünk konténerizált környezetben. Ennek eredményeként egy skálázható, megbízható és biztonságos rendszert hoztunk létre, amely nem csak az alapvető felügyeleti igényeknek felel meg, hanem a modern felhő-natív alkalmazások követelményeinek is.

Továbbfejlesztési lehetőségek

Számos ötletünk volt a fejlesztés során, amelyeket idő hiányában nem sikerült megvalósítani. A teljesség igénye nélkül felsorolunk most néhányat:

- A frontenden jelenleg csak emergency stop, és suggestion típusú parancsok adhatóak ki az egyes lifteknek, de a rendszer jelenlegi állapotában felvan készítve a sílifteken lokálisan is kiadható parancsok fogadására.
- A jelenlegi CLI felület helyett az egyes sílifteken egy Textual [9] alapú GUI kapott volna helyet.
- A jelenlegi Erlang-C sorbanállási modell csak statikus a konfigurálásnál megkapott adatokkal dolgozik.

A fejlesztés folyamán is odafigyeltünk ezekre a továbbfejlesztési irányokra és igyekeztünk olyan tervezői döntéseket hozni, amelyek segítik a program fejlesztésének folytatását.

Hivatkozások

- 1] Miniconda dokumentáció
<https://docs.anaconda.com/miniconda/>
- 2] Pyenv GitHub
<https://github.com/pyenv/pyenv>
- 3] PostgreSQL honlapja
<https://www.postgresql.org/>
- 4] Keycloak honlapja
<https://www.keycloak.org/>
- 5] Spring Boot honlapja
<https://spring.io/projects/spring-boot>
- 6] Angular honlapja
<https://angular.dev/>
- 7] Docker Compose dokumentáció
<https://docs.docker.com/compose/>
- 8] Minikube dokumentáció
<https://minikube.sigs.k8s.io/docs/>
- 9] Textual honlapja
<https://textual.textualize.io/>