## Követelményspecifikáció

### Síparadicsomot felügyelő rendszer Szoftverarchitektúrák tárgy házi feladat

**Feladatkiírás**

A hallgatók feladata egy síparadicsomot (ski resort) felügyelő elosztott rendszer megvalósítása. Ez egyrészt álljon egy központi komponensből (pl. webportálból), amely lehetővé teszi az személyzetnek a rendszer távoli irányítást (liftek terheltségének monitorozása, elindítás, leállítás), másrészt a lifteket képviselő alkalmazásokból. Nincsenek technológiai megkötések és nem szükségszerű a rendszer fizikai elosztottsága, akár egy gépen is megvalósítható.

**A** **fejlesztői csapat**

A csapat tagjai:

|  |  |  |
| --- | --- | --- |
| **csapattag neve** | **Neptun-kód** | **E-mail cím** |
| Kovács Bence Zoltán | S3OA6K | bence.kovacs.zoltan@gmail.com |
| Egyed Gábor | FIW1ZL | gabor.egyed.no1@gmail.com |

A csapatban a feladatokat többnyire frontend – backend feladatokra osztottuk. Egyed Gábor volt felelős a megjelenítésért, Kovács Bence pedig a backend megvalósításáért.

**Részletes feladatleírás**

A projekt során célunk egy olyan alkalmazás készítése, amely képes egy síparadicsom liftjeinek irányítására. A feladat tartalmazza egy állapotot generáló modell fejlesztését a liftek számára, egy a lifteket irányító egység fejlesztését, ami egy minimális kinézetű megjelenítésért és a liftek irányításáért felel. Ezentúl szükség van egy a liftek és a vezérlő egység közötti biztonságos kommunikáció megteremtésére.

Az említett egységek részletei:

* állapot generáló modell részletei
  + konfigurációból felolvasható értékek:
    - kezdő várakozók száma (a)
    - frissítés gyakorisága másodpercben (t)
    - frissítésenként leszálló emberek száma (b)
    - frissítésenként felszálló emberek száma (c)
    - frissítésenként [0-(d)] darab új várakozó jelenik meg (e) másodpercenként
    - a lift állapota bármikor változtatható. Az állapotok hatásai:
      * + *ideiglenesen lezárt* állapotban nem tudnak új utasok felszállni, a fentlévő utasok száma csökken (b)-vel frissítésenként de a várakozók száma folyamatosan nő [0-(d)]-vel (e) másodpercenként
        + *lezárt* állapotban nem tudnak új utasok felszállni, a fentlévő utasok száma csökken (b)-vel frissítésenként, de a várakozók száma is folyamatosan csökken [1-(f)]-el (e) másodpercenként
        + *nyitott* állapotban (c) új utas száll fel, (b) utas száll le, és (e) másodpercenként [0-(d)] darab új várakozó jelenik meg
* a vezérlő egység részletei
  + minimális megjelenítéssel listázza az aktuálisan elérhető lifteket, aminek tartalma automatikus frissül intervallumonként (nincs szó reaktív megjelenítésről, továbbfejlesztés keretében megvalósítható)
  + az elérhető liftekhez gombok tartoznak, melyekkel a lift állapota változtatható
  + a felületen belépés nélkül semmilyen műveletet nem lehet végezni, illetve megkülönböztetünk három felhasználói szerepet
    - liftek állapotát olvasni tudja
    - liftek állapotát olvasni és módosítani tudja
    - regisztrálhat új felhasználót, törölhet új felhasználót
  + a felületen regisztrálni lehet új felhasználót, ha van rá jogosultsága az aktuálisan belépett felhasználónak (ehhez szükség van legalább egy manuálisan regisztrált felhasználóra telepítéskor)
  + a felületen törölni lehet felhasználót, ha van rá jogosultsága (érdemes telepítéskor a default felhasználót törölni miután sikerült egy új törlésre és regisztrálásra alkalmas felhasználót létrehozni triviális biztonsági okokból)
* komponensek közti kommunikáció részletei
  + üzenet sor alapú biztonságos kommunikációs csatornán keresztül (a megbeszéltek alapján a liftek telepítése lehet manuális, így a csatornához való biztonságos csatlakozás manuálisan, pl certel fog történni)
  + a lift egységek frissítésenként közzéteszik az aktuális állapotukat, amire a vezérlő egység fel van iratkozva
  + a vezérlő egység cachelve tárolja a legfrissebb állapotát az éppen elérhető lift egységeknek
  + a vezérlő egység egy másik csatornán dedikált üzenettel jelzi a liftnek, hogy állapotváltásra van szükség, aminek sikeressége a felületen manuálisan ellenőrizhető

### Technikai paraméterek

A definiált alkalmazást Java platformra a Spring framework segítségével készítjük el annak érdekében, hogy platform függetlenül is lehessen használni. Az alkalmazáshoz szükséges üzenet sorhoz RabbitMQ brókert, a megjelenítéshez pedig Thymeleaf és Bootstrap mellett döntöttünk. A regisztrált felhasználók tárolásához perzisztens H2 adatbázist fogunk alkalmazni.

Technológiai indoklás, miért üzenetsor: (törölve lesz a beadott verzióból)

* Később úgy is lesz egy manager aki szeretne statisztikákat, illetve kimutatásokat csinálni, ehhez pedig nagyon jó integrált megoldások vannak egy üzenetsorhoz.
* Ígyis úgyis polling szerű dolgot kellett volna implementálni, nem szerettük volna feltalálni újra a kereket, érzésre rengeteg munkát spórol meg hogy nem kézzel kell az üzenetküldést kezelni.
* Nem tartom kizártnak, hogy később mondjuk egy már említett statisztikával dedikáltan foglalkozó egység telepítésére lenne szükség, ami aggregálja feldolgozza az adatokat és automatikusan jelez, ha valami abnormális működést észlel. Röviden könnyen bővíthetővé teszi az alkalmazást, könnyű mozdulni microservice irányba véleményünk szerint.
* Elérhetőek funkciók a legtöbb általam vizsgált üzenetsornál visszaigazolás implementálására, illetve engem nagyon meglepne, ha nem lennének erre standard megoldások. Jelenlegi specifikáció alapján manuálisan ellenőrizhető a felületen, hogy megérkezett-e az üzenet (hiszen látszik hogy milyen állapotban van) és szerintem elég komplex a feladat, hogy ne legyen szükség ennek a funkciónak az implementálására egy teljes értékű feladathoz. Ha mégis szükségét látod kérlek jelezd minél hamarabb, mert annak ellenére hogy van rá standard megoldás valószínűleg ahogy jelezted nem egyszerű dologról van szó.
* Illetve nem dolgoztam még üzenetsorokkal, és mindenképp egy fontos illetve hasznos technológiának találom amit jó lenne megismerni, ez pedig egy kiváló lehetőség rá.

TODO (minta maradvány)

### Szótár

**Dokumentum**: olyan adat (fájl vagy webcím), amely által reprezentált entitást a szoftverünk segítségével kezelni, nyilvántartani szeretnénk. Jelenleg dokumentumnak tekintjük az alábbi típusú fájlokat és webcímeket:

* Word fájl (.doc vagy .docx kiterjesztéssel)
* PDF-fájl (.pdf kiterjesztéssel)
* tetszőleges weboldal (teljes URL-jével megadva)
* Evernote jegyzet (azonosítójával megadva)

**Dokumentumok kezelése**: dokumentumok metaadatainak tárolása a program adatbázisában, melyek alapján szűrések, keresések végezhetők ezen, illetve a dokumentumok a programból megnyithatók, áthelyezhetők, törölhetők.

**Dokumentum beindexelése**: dokumentum felvétele a program adatbázisába, a dokumentum kezelése innentől kezdődhet.

**Dokumentum típusa**: a dokumentum jellege, amely meghatározza a hozzá tárolandó metaadatok körét (pl. konferenciacikk típusú dokumentum esetén tárolandó metaadat a konferencia neve, míg egy folyóiratcikk típusú dokumentum esetén nincs alapból

„konferencia neve” metaadat – kézzel hozzáadható).

**Dokumentumról tárolt metaadatok**: a dokumentumhoz fűzött adatok, melyeket a program adatbázisában tárol. Elemei lehetnek: dokumentum szerzője, dokumentum címe, dokumentum DOI-ja, dokumentumhoz fűzött megjegyzése, dokumentum eredeti helye, megjelenés éve, konferencia neve vagy folyóirat címe, dokumentumcímkék. A tárolt metaadatok köre függ a dokumentum típusától.

**Dokumentumcímke**: egy szöveg (pl. „munka”, „egyetem”), amely egy vagy több dokumentumhoz lehet rendelve. A dokumentumok ennek alapján csoportosíthatók. Egy dokumentumhoz több címke is tartozhat.

**DOI**: Digital Object Identifier, digitális objektumazonosító, a <http://www.doi.org/> oldalon

található szabványoknak és ajánlásoknak megfelelően.

**Indexelt (figyelt) mappa**: olyan helyi könyvtár a számítógépen, amelyben az újonnan létrehozott dokumentumokat automatikusan beindexeli a program futása közben.

### Essential use-case-ek

**Use-case** **diagram**

