EÖTVÖS LORÁND TUDOMÁNYEGYETEM
Informatikai Kar
Programozási Nyelvek és Fordítóprogramok Tanszék

Idősor vizualizációs webalkalmazás

Pataki Norbert
Adjunktus

Törteli Olivér Máté
programtervező informatikus, BSc

SZAKDOLGOZAT / DIPLOMAMUNKA

EREDETISÉG NYILATKOZAT

Alulírott TÖLTELI OLIVÉR MÁRK Neptun-kód: XVP81Z

ezennel kijelentem és aláírásommal megerősítem, hogy az Eötvös Loránd Tudományegyetem Informatikai Karának, PROGRAMOZÁSI NYELVEK ÉS FORDÍTÓPROGRAMOK Tanszékén írt, IDŐSOR VIZUALIZÁCIÓS WEBALKALMAZÁS

című szakdolgozatom/diplomamunkám saját, önálló szellemi termékem; az abban hivatkozott szakirodalom felhasználása a szerzői jogok általános szabályainak megfelelően történt.

Tudomásul veszem, hogy szakdolgozat/diplomamunka esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Budapest, 2020. 05. 30.

hallgató aláírása

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:

Név: Törteli Olivér Máté

Neptun kód: XVP8IZ

Képzési adatok:

Szak: programtervező informatikus, alapképzés (BA/BSc)

Tagozat: Nappali

Belső témavezetővel rendelkezem

Témavezető neve: *Pataki Norbert*

munkahelyének neve: ELTE IK, Prog. Nyelvek és Fordprog. Tanszék

munkahelyének címe: 1117. Bp. Pázmány Péter sétány 1/C

beosztás és iskolai végzettsége: Adjunktus, PhD

A szakdolgozat címe: Idősor vizualizációs webalkalmazás

A szakdolgozat témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

A szakdolgozat célja egy webes környezetben futó alkalmazás, melynek fő funkciója, hogy különféle adatforrásokat (pl. egérmozgatást, telefon giroszkóp adatokat) különböző grafikonok segítségével vizuálisan ábrázoljon (pl. sebesség grafikon). Ezek az adatforrások idősorként reprezentálhatóak, valamint különféle vizuális ábrázolásuk megadható. Az adatforrások file-ban kerülnek elmentésre a szerveren későbbi újranyitás céljából. A webes alkalmazás különálló backendből és frontendből áll, a kettő között hálózati kommunikáció zajlik. A webes felületet a felhasználó saját ízlésének megfelelően alakíthatja. A felületen megtalálható többek között az adatforrás, a konfiguráció, a metaadatok és a különféle grafikonok. Az alkalmazás Ovád Nóra közreműködésével készül.

Budapest, 2019.11.07.

Tartalomjegyzék

Bevezetés

A manapság egyre nagyobb teret hódít az idősoros adatok gyűjtése és azok feldolgozása. Az adatok feldolgozásának egy jókora (meghatározó) része a nyers adatok értelmezése, átalakítása, megjelenítése és tárolása.

Napjaink böngészői egyre jobb teljesítményre képesek (ez részben a folyamatos hardveres fejlődésnek is köszönhető). A böngészők nagy előnye pedig, hogy platform független (legalábbis a standard által előírt működésben [1]), így ha nem nagyon speciális (például hardver vagy szoftver specifikus, nagy számítási igényű) alkalmazást szeretnénk készíteni, akkor érdemes a webes technológiákat használni.

A dolgozatom célja, hogy webes technológiákra épülő idősoros adat mentő és megjelenítő alkalmazás szerver-oldali részét implementáljam.

Szerettem volna olyan keretrendszerben elkészíteni a szerver alkalmazást, hogy továbbfejlesztő lehessen akár nagyobb kaliberű alkalmazássá is – például nagyobb terhelés kiszolgálására alkalmas lehessen. Ezért választottam az Erlang-ot a szerveralkalmazás programozási nyelvének.

Felhasználói dokumentáció

Ismertető

Az alkalmazás azért jött létre, hogy egy könnyen használható eszközt, felületet biztosítson idősoros adatok rögzítésére, tárolására és az adatok megjelenítésére (minőségének ellenőrzésére).

Az alkalmazás két fő egységre bontható fel. A szerver – főleg erről az egységről szól a leírás – amely az adatok tárolásáért, kiszolgálásáért felelős. Illetve a kliens – amelyet csak néhány egyszerű felhasználási lehetőség bemutatásáig említek – amely az adatokba való betekintésért felelős.

A timeseries alkalmazás kezdő adatelemzőknek, olyan felhasználóknak készült akik adatot szeretnének gyűjteni, és közben ellenőrizni azokat. A telepítéshez és használathoz alapvető Debian operációs rendszer, illetve webes technológiák ismerete szükséges. A kliens használatához a böngészős ismeretek szükségesek.

Telepítés

Szükséges a hálózati elérés a timeseries szerverhez, ha nem ugyanazon a gépen fut a szerver, mint ahol a kliens oldalt betöltjük.

Hardver igények

Minimális hardver igény

- 1 CPU mag
- 2 GB RAM
- 100 MB merevlemez kapacitás

Optimális hardver igény

- 2 CPU
- 4 GB RAM
- 500 MB merevlemez kapacitás

Kliens függőségek telepítése

Firefox böngésző telepítése szükséges az elemző felület betöltéséhez. (Használható Chrome vagy Chromium böngésző is, de a leírás ezekre nem tér ki) A kliens használatához csak egy Firefox böngészőre van szükség, amely telepíthető:

```
# Firefox
sudo apt install firefox-esr
```

A további kliens függőségeket szerver függőségként kezeli a leírás, mivel a szerverrel együtt kell lefordítani, majd azt kiszolgálni.

Szerver függőségek telepítése

Az alkalmazás Debian-alapú Linux operációs rendszereken/rendszerekre készült, például Debian 10, Ubuntu v20.04 LTS. A telepítési leírás ezekre a rendszerekre vonatkozik.

Az alkalmazás más platformon is futtatható lehet a megfelelő (akár helyettesítő) függőségek telepítésével, ez az eset nem része a leírásnak.

A következő parancsokkal a függőségeket tudja feltelepíteni, a folyamat percekig eltart.

A közvetett függőségek telepítése

```
sudo apt install git \
                 wget curl \
                 build-essential make \
                 gunzip unzip

# asdf
git clone https://github.com/asdf-vm/asdf.git \
          ~/.asdf \
          --branch v0.7.8
```

A szerver függőségek telepítése

```
# erlang
asdf plugin-add erlang
asdf install erlang 22.3

# rebar
asdf plugin-add rebar
asdf install rebar 3.9.1
```

A kliens függőségek telepítése

```
# elm
asdf plugin-add elm
asdf install elm 0.19.1

# nodejs
asdf plugin-add nodejs
asdf install nodejs 9.9.0
```

Fordítás

Majd a következő parancsokat a timeseries projektben hajtsa végre az alkalmazás lefordításához:

```
# Fordítás
make build

# Tesztek lefuttatása, hogy tényleg sikerült-e a telepítés
make test
```

Ha az előzőekben leírt folyamat sikeresen lefutott, akkor készen áll a szoftver a használatra.

Használati útmutató

Adatformátum

Alapvetően minden adat, ami szerver webes API-ján keresztül utazik az JSON formátumú, így a HTTP lekérdezések megfelelő fejléc paramétereket mindig be kell állítani („Content-Type”, „Accept”).

Összesítés
{„token1”: <token1 adatsor hossza>, „token2”: …}
Mérési esemény
{„t”: <idő1>, „value1”: <érték1>, …}
Adatsor
{”token”: <token>, ”events”: [<esemény1>, …]}

Szerver használata

Minden parancs a timeseries mappa gyökerében állva értendő.

Szerver elindítása

A szerver elindítása a következő paranccsal érhető el (a példában látható log üzenetekből egy részlet, hogy megerősítsen minket abban, hogy sikeres volt az indítás):

```
$ make shell
/home/omt/.asdf/shims/rebar3 release


…
=PROGRESS REPORT==== 01-Jan-1900::23:50:58.251587 ===
    supervisor: {local,timeseries_sup}
    started: [{pid,<0.495.0>},
              {id,timeseries_server},
              {mfargs,
                  {timeseries_server,start_link,
                      [...]}},
              {restart_type,permanent},
              {shutdown,5000},
              {child_type,worker}]
…

(timeseries@localhost)1>
```

A szerver a 8080-as porton hallgat alapértelmezetten.

Ha a port már használatban van, akkor a következő hibaüzenetet kaphatjuk:

```
…
=INFO REPORT==== 01-Jan-1900::23:48:52.019680 ===
    application: timeseries
    exited: {bad_return,
             {{timeseries_app,start,[normal,[]]},
              {'EXIT',
               {{badmatch,{error,eaddrinuse}},
               [{timeseries_app,start,2,
                 [{file,
                   ".../apps/timeseries/src/timeseries_app.
erl"},
                  {line,64}]},
               {application_master,start_it_old,4,
                           [{file,"application_master.erl"
},{line,277}]}]}}}}
    type: permanent
…
```

A port átállításához, szerkessze config/sys.config fájlban a port paramétert majd indítsa újra a szervert.

Szerver leállítása

A szerver megállítása a q() függvény meghívásával érhető el:

```
(timeseries@localhost)1> q().
ok
(timeseries@localhost)2>
$ # ez már a linux shellünk
```

Parancssori alkalmazás – timeseries_cli

Minden CLI parancs elindított szerver alkalmazással fog tudni működni.

Adatsor feltöltése

A timeseries_cli --upload <fájl> paranccsal tölthetjük fel az adatsorunkat:

```
$ ./timeseries_cli --upload example_data/ba.json

Uploading succeeded.
```

HTTP végpontok

Az összesítés lekérdezésére, illetve adatsorok fel- és letöltésére a következő HTTP végpontok érhetőek el:

Összesítés letöltése	
Útvonal	/summary
Metódus	GET
Fejlécek	Accept: application/json

Adatsor feltöltése	
Útvonal	/upload/:token
Metódus	POST
Fejlécek	Content-Type: application/json
Tartalom	Adatsor (lásd Adatformátum 8)

Adatsor letöltése	
Útvonal	/download/:token
Metódus	GET
Fejlécek	Accept: application/json

A következő példában néhány parancssori (curl program segítségével) lekérdezéssel szemléltetem az összesítéshez, illetve az adatsor feltöltéséhez tartozó HTTP végpontok működését:

```
# Összesítés
$ curl --request GET \
       --header "Accept:application/json" \
       http://127.0.0.1:8080/summary
{}

# Adatsor feltöltése
$ curl --request POST \
       --header "Content-Type:application/json" \
       --data @example-data/ba-stocks.json \
       http://127.0.0.1:8080/upload/BA-stocks
{"result":"ok"}

# Adatsor feltöltése
$ curl --request POST \
       --header "Content-Type:application/json" \
       --data @example-data/london-humidity-Aug.json \
       http://127.0.0.1:8080/upload/london-humidity-Aug
{"result":"ok"}
# Összesítés
$ curl --request GET \
       --header "Accept:application/json" \
       http://127.0.0.1:8080/summary
{"london-humidity-Aug":24,"BA-stocks":1420}
```

A következő példában pedig néhány parancssori (curl program segítségével) lekérdezéssel szemléltetem az összesítéshez, illetve az adatsor letöltéséhez tartozó HTTP végpontok működését:

```
# Összesítés
$ curl --request GET \
       --header "Accept:application/json" \
       http://127.0.0.1:8080/summary
{"london-humidity-Aug":24,"BA-stocks":1420}

# Adatsor letöltése
$ curl --request GET \
       --header "Accept:application/json" \
       http://127.0.0.1:8080/download/london-humidity-Aug
{"token":"london-humidity-Aug","events":
[{"t":0,"humidity":87},{"t":1,"humidity":88}, … ]}

# Adatsor letöltése
$ curl --request GET \
       --header "Accept:application/json" \
       http://127.0.0.1:8080/download/BA-stocks
{"token":"BA-stocks","events":
[{"t":0,"open":50.88,"close":50.0},
{"t":1,"open":50.75,"close":51.0}, … ]}
```

WebSocket végpontok

Az adatsorok fel- és letöltésére a következő WebSocket végpontok érhetőek el:

Adatsor feltöltése	
Útvonal	/save/:token
Bemeneti adatkeretek	Mérési esemény (lásd Adatformátum 8)
Kimeneti adatkeretek	-
Terminálás	A dolga végeztével a kliens megszakítja a kapcsolatot.

Adatsor letöltése	
Útvonal	/load/:token
Bemeneti adatkeretek	-
Kimeneti adatkeretek	Mérési esemény (lásd Adatformátum 8)
Terminálás	A dolga végeztével a szerver megszakítja a kapcsolatot.

A következő parancssori (wscat [2] program segítségével) utasításokkal szemléltetem egy adatsor feltöltéséhez tartozó WebSocket végpont működését:

```
$ wscat --connect ws://127.0.0.1:8080/save/token-1
Connected (press CTRL+C to quit)
> {"t":1,"x":2}
> {"t":3,"x":4}
> {"t":5,"x":6}
# Ctrl + c
```

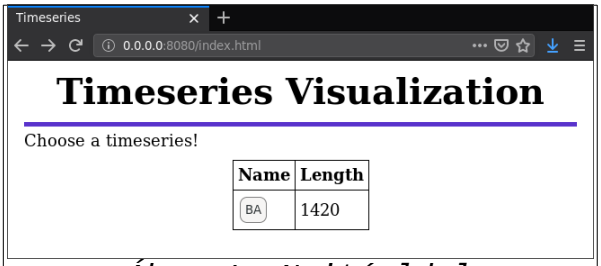
A következő parancssori (wscat [2] program segítségével) utasítással szemléltetem egy adatsor letöltéséhez tartozó WebSocket végpont működését:

```
$ wscat --connect ws://127.0.0.1:8080/load/token-1
Connected (press CTRL+C to quit)
< {"x":2,"t":1}
< {"x":4,"t":3}
< {"x":6,"t":5}
Disconnected (code: 1000, reason: "")
```

Kliens használata

A kliens használatához szükség van a szerver elindítására. (lásd Szerver elindítása 8)

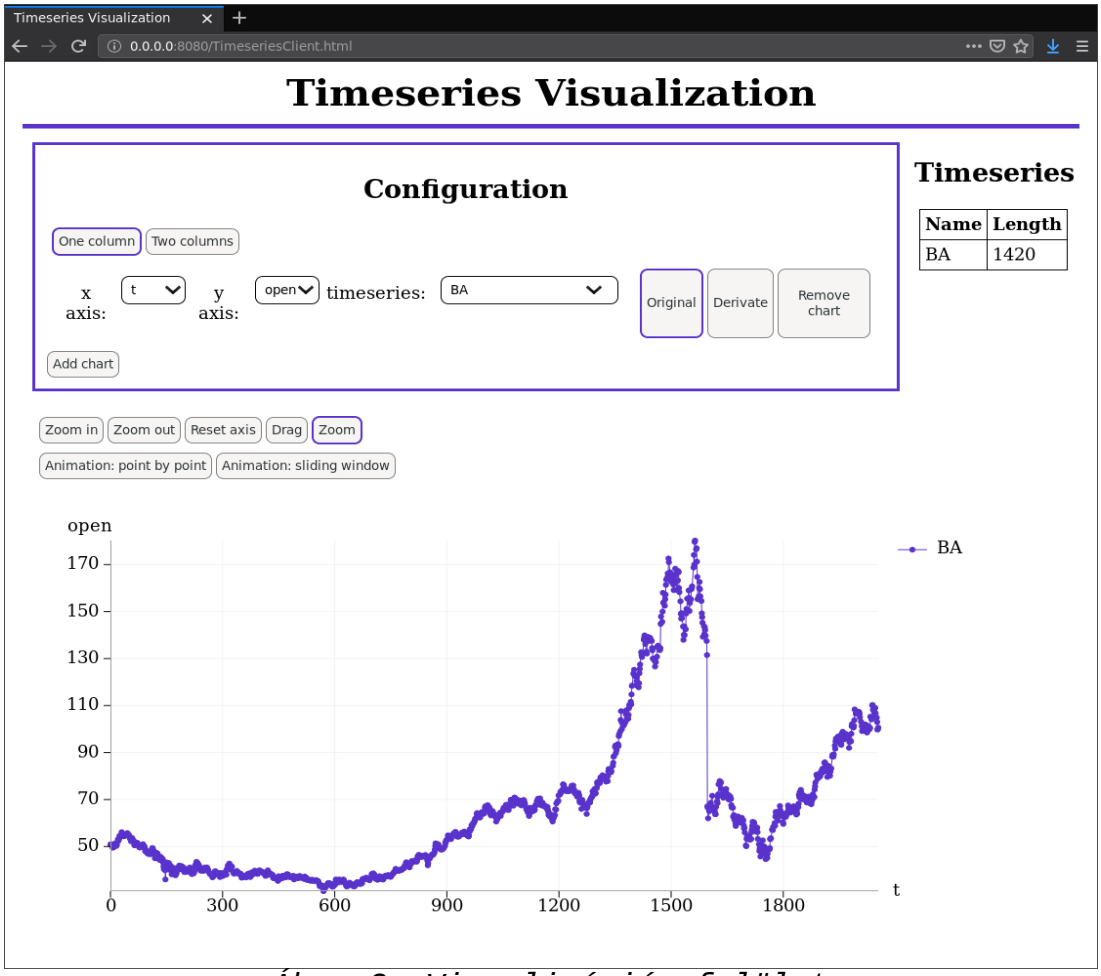
A kliens nyitóoldala az /index.html betöltésével érhető el böngészőjébe (lásd Ábra 1). Ezen az oldalon láthatja listába szedve milyen adatsorok érhetően el az alkalmazásban. Minden adatsor tokenje mellett az adatsor hossza is látható.



Ábra 1: Nyitóoldal

Egy konkrét adatsor tokenjére kattintva rajzoltathatja ki a kívánt adatsort (lásd Ábra 2). Ezen a felületen tudja leellenőrizni a feltöltött adatsorokat, akár egyszerre kettőt is. Megnézheti az adatsorok koordinátáit, akár derivált értékeit. Rá tud közelíteni az ábrákra.

Ennél részletesebb leírás nem része ennek a leírásnak, mivel ez a dolgozat a szerver-oldali implementációról szól.



Ábra 2: Vizualizációs felület

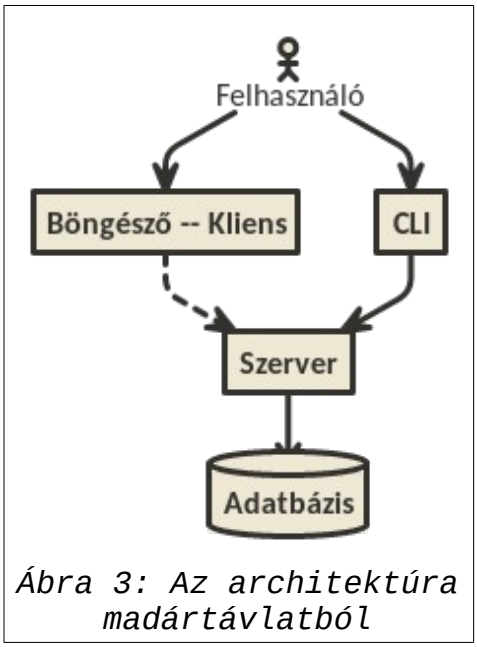
Fejlesztői dokumentáció

Áttekintés

Az alkalmazás célja, hogy kényelmesen és egyszerűen lehessen idősoros adatokat menteni, tárolni és ellenőrizni.

Az alkalmazás 2 fő és 2 kisebb részből épül fel (lásd Ábra 3) a szerverből illetve a kliensből. Továbbá a felhasználói élmény javítása érdekében található egy parancssori felület (továbbiakban CLI), illetve része az alkalmazásnak egy fájlrendszer alapú adatbázis is – opcióként választható a memória alapú tárolás is.

A kliens alkalmazásra nem tér ki ez a leírás részletesebben.



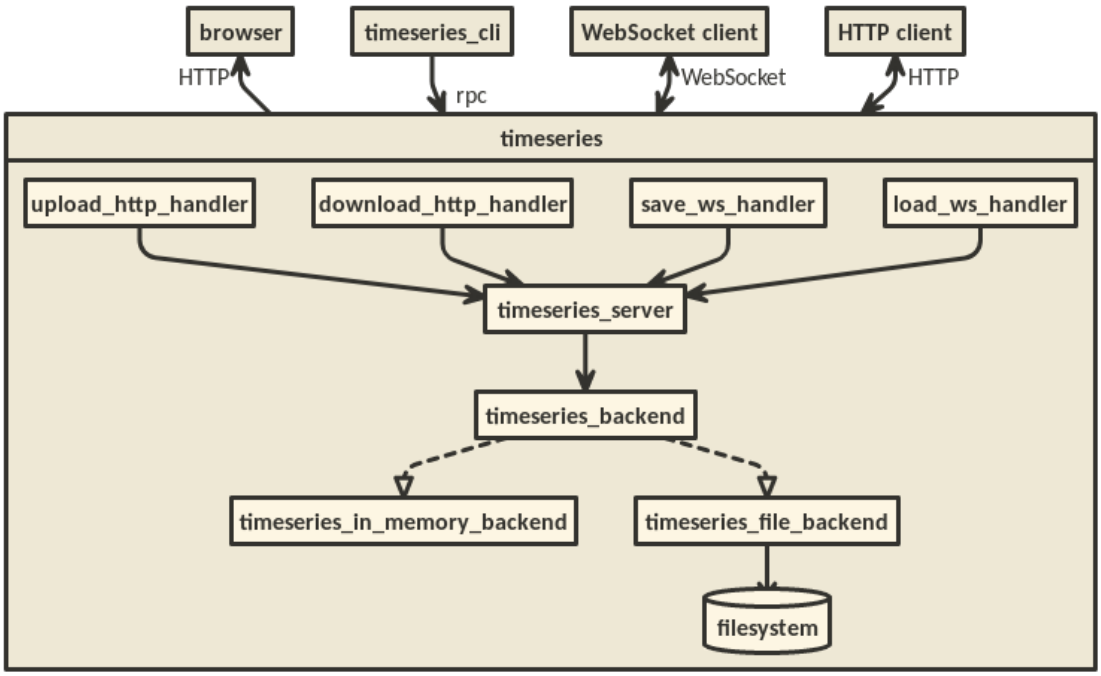
Ábra 3: Az architektúra madártávlatból

Az alkalmazás továbbá webes felületet biztosít az adatsorok fel- és letöltésére. Így akár közvetlenül összeköthető egy adatgyűjtő alkalmazással is.

A szerver alkalmazásnak 3 főbb része van (lásd Ábra 4):

- A hálózati lekérdezéseket kezelő modulok

- Az adatsorokat kezelő szerver: kezeli adatsorokat, itt található az üzleti logika, itt indulnak az adatbázis műveletek.
- Az adatsorokat tároló adatbázis: ez a felelős az adatok tárolásáért.



Ábra 4: Részletes architektúra

A következő alfejezetekben az szerver alkalmazás különböző részeinek – rétegeinek, moduljainak, hívásainak – leírása található.

Az alkalmazás felépítése

Webes interfész moduljai

Ezek a modulok cowboy [3] handler modulok, amelyek a beérkező HTTP vagy WebSocket kapcsolatokat kezelik.

summary_http_handler

Ez a modul a /summary útvonalra beérkező HTTP lekérdezéseket kezeli. A lekérdezés megfelelő paraméterezésével az összesítés tölthető le.

Lekérdezés	
Útvonal	/summary
Metódus	GET
Fejlécek	Accept: application/json

Válasz	
Státusz kód	200
Fejlécek	Content-Type: application/json
Tartalom	Összesítő objektum (lásd Adatformátum 8)

Amennyiben nem megfelelő a paraméterezés a REST [4] által előírt módon jár el ez a modul.

upload_http_handler

Ez a modul a /upload/:token útvonalra beérkező HTTP lekérdezéseket kezeli. A lekérdezés megfelelő paraméterezésével egy adatsort lehet feltölteni.

Helyes lekérdezés:

Lekérdezés	
Útvonal	/upload/:token
Metódus	POST
Fejlécek	Content-Type: application/json
Tartalom	Adatsor (lásd Adatformátum 8)

Válasz	
Státusz kód	200

Amennyiben nem megfelelő a paraméterezés a REST [4] által előírt módon jár el ez a modul.

download_http_handler

Ez a modul a /download/:token útvonalra beérkező HTTP lekérdezéseket kezeli. A lekérdezés megfelelő paraméterezésével egy adatsort lehet letölteni.

Lekérdezés	
Útvonal	/download/:token
Metódus	GET
Fejlécek	Accept: application/json

Válasz	
Státusz kód	200
Fejlécek	Content-Type: application/json
Tartalom	Adatsor (lásd Adatformátum 8)

Amennyiben nem megfelelő a paraméterezés a REST [4] által előírt módon jár el ez a modul.

save_ws_handler

Ez a modul a /save/:token útvonalra csatlakozó WebSocket kapcsolatot kezeli. A megfelelő WebSocket adatkeretek küldésével lehet elmenteni (mérési) eseményeket az útvonalban megadott tokenhez.

Kommunikáció	
Útvonal	/save/:token
Bemeneti adatkeretek	Mérési esemény (lásd Adatformátum 8)
Kimeneti adatkeretek	-
Terminálás	A dolga végeztével a kliens megszakítja a kapcsolatot.

load_ws_handler

Ez a modul a /load/:token útvonalra csatlakozó WebSocket kapcsolatot kezeli. A megfelelő WebSocket adatkeretek küldésével lehet letölteni (mérési) eseményeket az útvonalban megadott tokenhez tartozó adatsorból.

Kommunikáció	
Útvonal	/load/:token
Bemeneti adatkeretek	-
Kimeneti adatkeretek	Mérési esemény (lásd Adatformátum 8)
Terminálás	A dolga végeztével a szerver megszakítja a kapcsolatot.

timeseries_cli

Ez az Erlang alkalmazás a egy egyszerű escript alkalmazás amellyel a megszokott Unixos parancssori felületen léphetünk a szerverrel interakcióba.

A timeseries_cli célja, hogy a meghívásakor automatikusan csatlakozzon az adott gépen elindított szerverhez és Erlangos RPC hívásokon keresztül végrehajtsa a kívánt akciókat. Mindezt úgy, hogy a felhasználónak nem kell ismernie az Erlang környezetet.

Adatsor feltöltése

A „timeseries_cli --upload <fájlnév>” parancs egy adatsor feltöltésére szolgál.

Például az egyik példa adatsor felöltése így néz ki.

```
$ ./timeseries_cli --upload example_data/ba.json

Uploading succeeded.
```

timeseries

A timeseries modul tartalmazza:

- az adatsorokat reprezentáló timeseries objektumot. Ez az objektum tartalmazza az összes szükséges információt, amely egy konkrét adatsorhoz tartozik, mint a token és a mérési események,
- az összesítő objektumot,
- továbbá azokat az API hívásokat, amellyel a timeseries objektumot tudjuk létrehozni, módosítani vagy az adott objektumból lekérdezni a kívánt adattagját.

Típusok

```
-record(timeseries, {token :: token(),
                     events :: [event()],
                     state :: any()}).

-type token() :: binary().
-type event() :: #{<<"t">> := any(), binary() => any()}.
-type info() :: non_neg_integer().
-type timeseries() :: #timeseries{}.

-type summary() :: #{token() => info()}.
```

A timeseries objektumot Erlang rekordként implementáltam. Ezt az objektum úgynevezett opaque típusként használom az alkalmazásban, tehát minden műveletet amit az objektumon

szeretnék végrehajtani azt a timeseries modul által biztosított API hívásokon keresztül tehetem meg.

Az Erlang rekordok használatában az a nehézség, hogy csak akkor használható másik modulból is ha a megfelelő fejléc fájlba ki van rakva a rekord definíciója és ezt a fejlécet be tudja tölteni a felhasználó modul.

A jobb enkapszuláció és az imént említett nehézség miatt is döntöttem a mellett, hogy minden timeseries objektumon értelmezett műveletet a timeseries modulban implementálom le.

Így ebben a modulban található szintén a timeseries objektumhoz tartozó API egységtesztjei is. (lásd Egységtesztek a timeseries objektumon 38)

timeseries API függvények

- new/1, new/2: Létrehoz egy új adatsort az adott tokennel, vagy opcionálisan a megadott mérési eseményekkel
- add/2: Hozzáad egy mérési eseményt az adatsorhoz
- info/1: Visszaadja az adatsorhoz tartozó összesítést, jelen implementációban a length/1 hívás eredményét
- length/1: Visszaadja az adatsor hosszát – jelen implementációban a mérési eseményeinek számát
- token/1: Visszaadja az adatsor tokenjét
- events/1: Visszaadja az adatsor mérési eseményeit
- is_valid/1: Eldönti, hogy az adott adatsor vagy esemény eleget tesz-e az objektumtól elvárt formátumnak

timeseries_server

Ez a szerver hivatott az adatsorok kezelni, kiszolgálni, végrehajtatni a szükséges adatbázis tranzakciókat.

Erlang OTP eleve tartalmazza a gen_server megoldást, amellyel könnyen implementálhatjuk szerveralkalmazásunkat. Ez a behaviour régóta rész az Erlangos környezetnek, sok tapasztalatot szerezve fejlesztették éveken át és a mai napig használják éles szoftverekben. Előnye, hogy számtalan eszköz áll rendelkezésre, amellyel kényelmesebben fejleszthetjük, használhatjuk a szerveralkalmazásunkat.

A timeseries_server ezért készült el gen_server implementációval, mert egyrészről egy kiforrott technológia, másrészről pedig a jövőben továbbfejlesztést nagyban elősegíti egy ilyen sokat használt technológia, például az alkalmazás felügyelete vagy a skálázása. (További információk [4])

Ebben a fejezetben leírt függvény szignatúrák a timeseries modul típusait használja fel. (a könnyebb olvashatóság érdekében nem írom ki a modul prefixet, például timeseries:token() helyett token() szerepel)

gen_server függvények:

- init/1: gen_server elindítása az adott konfigurációval.
- handle_call/3: Beérkező szinkron hívás kezelése. Jelen implementációban minden alkalmazás API hívás, ilyen szinkron hívás.
- handle_cast/2: Beérkező aszinkron hívás kezelése. Jelen implementáció nem tartalmaz aszinkron hívást.

- handle_info/3: Minden más szerverbe érkező üzenet kezelése. Jelen implementáció nem tartalmaz más üzenetet.
- terminate/2: A szerver megállásával hívódik meg ez a függvény.
- code_change/3: A futás közben végrehajtandó kód frissítéskor hívódik meg ez a függvény. Jelen implementáció nem támogatja a futás közben végrehajtott kód frissítést.

A következő API hívások az éppen felkonfigurált timeseries_backend (továbbiakban backend) hívásait használják az adatsorok tárolására.

Összesítés

Ennek a függvénynek a célja, hogy összesítse az adatsorok aktuális állapotát, átfogó képet adjon az elérhető adatsorokról.

Függvény szignatúra:

```
summarize() -> {ok, Summary}
   • Summary = summary()
```

Összegyűjti az éppen aktuálisan elérhető összes adatsort, illetve ezeknek az adatsoroknak a hosszát – a rögzített mérési események számát.

Nyilvánvaló továbbfejlesztési lehetőség kibővíteni ezeknek az összesített információknak a halmazát (például: mérési események mezői).

Ez a függvény hívja meg a backend summary függvényét.

Adatsor létrehozása

Ennek a függvénynek a célja, hogy létrehozzon egy új adatsor a megadott tokenhez.

Függvény szignatúra:

```
new(Token) -> ok | {error, token_already_exist}
   • Token = token()
```

Leellenőrzi, hogy elérhető-e még az adott token – nincs-e már egy adatsor ehhez a tokenhez eltárolva –, amennyiben már foglalt, akkor hibával tér vissza a függvény. Egyébként létrehoz egy üres adatsort, amelybe már lehet mérési eseményeket menteni.

Ez a függvény hívja meg a backend save függvényét, hogy tárolja a létrehozott adatsort objektumot.

Esemény hozzáadása

Ez a függvény hozzáad egy mérési eseményt az megadott tokenhez tartozó adatsorhoz.

Függvény szignatúra:

```
add(Token, Event) -> ok | {error, unknown_token}
   • Token = token()
   • Event = event()
```

Leellenőrzi, hogy elérhető-e az adott token, ha egy nem létező adatsorhoz akar a hívó hozzáadni a mérési eseményt, akkor hibával tér vissza. Egyébként pedig hozzáadja az adott adatsorhoz a mérési eseményt a backend add hívásán keresztül.

Adatsor betöltése

Ez a függvény betölti a megadott tokenhez tartozó adatsort.

Függvény szignatúra:

```
load(Token) -> {ok, Timeseries} | {error, unknown_token}
```

- Token = token()
- Timeseries = timerseries()

Leellenőrzi, hogy elérhető-e az adott token, ha egy nem létező adatsort akart a hívó lekérdezni, akkor hibával tér vissza. Egyébként lekérdezi a backend load hívásán keresztül a kívánt adatsort, majd visszatér ezzel.

Adatsor elmentése

Ez a függvény elmenti a megadott adatsort.

Függvény szignatúra:

```
save(Timeseries) -> ok | {error, token_already_exist}
```

- Timeseries = timerseries()

Leellenőrzi, hogy elérhető-e még az adott token – nincs-e már egy adatsor ehhez a tokenhez eltárolva –, amennyiben már foglalt, akkor hibával tér vissza a függvény. Egyébként elmenti az alkalmazást az adatsort a backend save függvényével.

timeseries_backend

Ez egy Erlangos „behaviour” – ez az absztrakt osztályoknak vagy az interfészeknek felel meg – amely azt hivatott előírni, hogy milyen funkcionalitást kell biztosítani az adatbázis rétegnek az adatsorok tárolásához.

A következőkben az interfész függvények leírásait adom meg. Minden hívás (kivéve az initialize) bemeneti paraméterként és kimeneti értékként is tartalmazza az adatbázis állapotát – ennek az állapotnak a tartalma és menedzselése a konkrét implementáció része.

Típusok

```
-type config() :: any().
-type state() :: any().
```

Inicializálás

Ez a hívás felelős az adatbázis elindításáért. Bemeneti paramétere a konfiguráció, kimenete pedig az inicializált adatbázis réteg állapota.

Függvény szignatúra:

```
initialize(Config) -> {ok, State}
   • Config = config()
   • State = state()
```

Összesítés

Ezzel a hívással a lekérdezéskor elérhető adatsorokról adok egy összesítést, ami jelen esetben az adatsor hossza. Nyilvánvaló továbbfejlesztési lehetőség ennek az összesítésnek a finomítása részletesebb információkkal az adatsorokról és/vagy általánosabb adatbázis információkkal.

Függvény szignatúra:

```
summarize(State) -> {{ok, Summary} | {error, Reason},
State}
   • State = state()
   • Summary = timeseries:summary()
   • Reason = any()
   • State = state()
```

Token elérhetősége

Szükségem van erre a függvényre ahhoz, hogy az adatsorok azonosítói egyediek maradhassanak és ne lehessen felülírni

azokat. Eldöntöm egy megadott tokenről, hogy elérhető-e (vagy sem).

Függvény szignatúra:

```
is_available(Token, State) -> {{ok, IsAvailable}, State}
   • Token = timeseries:token()
   • State = state()
   • IsAvailable = boolean()
```

Adatsor elmentése

Függvény szignatúra:

```
save(Timeseries, State) -> {ok | {error, unknown_token}, State}
   • Timeseries = timeseries:timeseries()
   • State = state()
```

Adatsor betöltése

Függvény szignatúra:

```
load(Token, State) -> {{ok, Timeseries} | {error, unknown_token}, State}
   • Token = timeseries:token()
   • State = state()
   • Timeseries = timeseries:timeseries()
```

Esemény elmentése

A streaming megoldás érdekében szükség van arra, hogy egy mérési eseményt hozzá tudjunk adni egy létező adatsorhoz. Ez a hívás egy adott tokenhez tartozó adatsorhoz adja hozzá a megadott eseményt.

Függvény szignatúra:

```
add(Token, Event, State) -> {ok | {error, unknown_token},
State}
```

- Token = timeseries:token()
- Event = timeseries:event()
- State = state()

timeseries_in_memory_backend

Ez a modul egy implementációja timeseries_backend interfésznek. A függvény szignatúrák eleget tesznek a timeseries_backend-ben leírtaknak, az inicializáláshoz használt konfigurációban, illetve az állapotban (”State”) különbözik – konkrétabb, mint amit timeseries_backend definiál.

A backend célja, hogy csak a memóriába menti el, tárolja az adatsorokat. Erre akkor lehet szükség, ha mondjuk csak adatsorok továbbítására szeretnénk használni az alkalmazást. Így kevesebb erőforrást fog felhasználni az alkalmazásunk.

A backend Erlang map-be – hash tábla – menti el az adatsorokat az tokenjükkel indexelve (erre a map-re a továbbiakban tároló táblaként fogok hivatkozni). Így hatékonyan megtalálhatóak az adatsorok.

Típusok

```
-type config() :: #{}.
-type state() :: #{token() => timeseries()}.
```

Inicializálás

Nincs konfigurálható paramétere ennek a backend-nek. Inicializáláskor létrehoz egy üres map-et, amelybe az adatsorokat fogja tárolni.

Összesítés

A tároló táblában található kulcsokhoz – tokenekhez – lekérdezi a hozzájuk tartozó adatsorok hosszát .

Token elérhetősége

Leellenőrzi, hogy található-e a megadott token a tároló táblában, amennyiben nem található a token elérhető, ellenkező esetben pedig nem elérhető.

Adatsor elmentése

Lekérdezi elmenteni kívánt adatsor tokenjét, leellenőrzi, hogy létezik-e már és ha nem, akkor hozzáadja a tároló táblához.

Például, a „példa” tokennel rendelkező adatsor elmentése egy üres tároló táblába ilyen táblákat eredményez:

```
% Tároló tábla a mentés előtt
#{}

% Tároló tábla a mentés után
#{<<”példa”>> => {timeseries, …}}
```

Adatsor betöltése

Leellenőrzi, hogy a betölteni kívánt tokenhez tartozik-e adatsor a tároló táblában. Ha létezik az adatsor, akkor lekérdezi azt a táblából és visszatér azzal. Ellenkező esetben hibával tér vissza.

Esemény hozzáadása

Leellenőrzi, hogy a betölteni kívánt tokenhez tartozik-e adatsor a tároló táblában. Ha létezik az adatsor, akkor lekérdezi azt a táblából, hozzáadja mérési eseményt az

adatsorhoz, majd felülírja az adatsor korábbi állapotát. Ellenkező esetben hibával tér vissza.

timeseries_file_backend

Ez a modul egy implementációja timeseries_backend interfésznek. A függvény szignatúrák eleget tesznek a timeseries_backend-ben leírtaknak, az inicializáláshoz használt konfigurációban, illetve az állapotban (”State”) különbözik – konkrétabb, mint amit timeseries_backend definiál.

A konfiguráció része, hogy hová mentse az alkalmazás az adatokat.

A megadott mappába menti el az adatsorokat az alkalmazás a token nevével ellátott fájlba. A fájl tartalmazza az adatsorhoz tartozó mérési eseményeket.

Típusok

```
-type config() :: #{data_dir := string()}.
-record(state, {directory :: string()}).
-type state() :: #state{}.
```

Inicializálás

A konfigurációban megadott elérési utat – mappát – biztosítja – létrehozza, ha nem létezik még.

Összesítés

A mappában található fájlok nevét – az adatsor tokenjét – és a fájlok sorainak számát adja vissza összesítésként.

Token elérhetősége

Ha megtalál a mappában a keresett fájlnevet, akkor már nem elérhető az adott token.

Adatsor elmentése

Létrehozza tokennel elnevezett fájlt az alkalmazás és soronként elmenti a mérési eseményeket.

A következő példában a BA-stocks tokenhez tartozó fájl tartalma, formátuma látható:

```
$ cat BA-stocks
{"t":0,"open":50.88,"close":50.0}
{"t":1,"open":50.75,"close":51.0}
{"t":2,"open":51.0,"close":50.5}
…
```

A fájlba azért soronként kerülnek bele a mérési események – és nem pedig egy JSON vektorba –, mert így folytonosan tudja az alkalmazás beleírni a fájlba a beérkező eseményeket, így sokkal hatékonyabb.

Adatsor betöltése

A kívánt tokennel elnevezett fájlt felolvassa az alkalmazás soronként, melynek sorai a mérési események.

Ezekből az adatokból pedig létrehoz az alkalmazás egy timeseries objektumot.

Esemény elmentése

A kívánt tokennel elnevezett fájlhoz végére írja – egy új sorba – a mérési eseményt.

Függőségek

Fejlesztői eszközök

asdf (+ plugins)

Az asdf[6] program a különböző programozási környezetek telepítését segíti. Egy egyszerűen használható, egyesített felületet nyújt, amellyel könnyen feltelepíthetjük és konfigurálhatjuk a kívánt programozási környezetet.

Make

A make fejlesztési folyamatok, részfeladatok automatizálására, rendszerezésre készített eszköz. Unix operációs rendszereken évtizedek óta nagy népszerűségnek örvend.

Rebar3

A rebar3 egy, kifejezetten Erlang projekt menedzselésére készített, parancssori alkalmazás, ez az alkalmazás az iparági sztenderd.

Wscat

Ez az alkalmazás egy parancssori Websocket kliensalkalmazás. Abban segít, hogy könnyen letesztelhetőek legyenek a szerveralkalmazás WebSocket végpontjai, akár kézzel is. Több különböző eszközt kipróbáltam erre a célra, de végül az egyszerűsége miatt – mind telepíthetőségében, mind használatában – ezt a programot választottam.

Felhasznált kész komponensek

Az Erlangos környezet számtalan hasznos beépített alkalmazást tartalmaz. A következő felsorolás a külső függőségeket tartalmazza.

Cowboy

A cowboy Erlang alkalmazás egy nagy teljesítményű HTTP szerver, ez az alkalmazás a legnépszerűbb az Erlangos környezetben erre a feladatra.

Jiffy

A jiffy egy JSON konverter Erlang alkalmazás. Az Erlangos JSON konverterek között ezt használják a legtöbben, így jó választásnak bizonyult.

Gun

Ez egy tesztelésre használt HTTP és WebSocket kliens alkalmazás. Az integrációs tesztek ezzel a klienssel állítottam össze.

Tesztelés

Az alkalmazás fejlesztése közben elkészültek „zöld utas” tesztek, amelyek a szoftver helyes működését hivatottak leírni, leellenőrizni.

Az alkalmazás kódja tartalmaz

- integrációs teszteket a HTTP, illetve a WebSocket végpontok működéséről,
- teljesítmény teszteket a HTTP végpontok áteresztőképességéről,
- egységteszteket az adatsorokat reprezentáló timeseries objektum helyes működéséről,
- illetve integrációs tesztet a parancssoros alkalmazás helyes működéséről.

Alkalmazás integrációs tesztje

A projekt apps/timeseries/test/integration_SUITE.erl fájlában a basics, illetve basics_with_file_backend függvény tartalmazzák az alkalmazás integrációs tesztjét. A két teszteset kettő adatsort tölt fel a szerverre, lekérdez egy összesítést majd tölti le azokat az adatsorokat. A fel- és letöltés mind HTTP és WebSocket-en keresztül is megtörténik. A két teszteset között annyi a különbség, hogy az előbbi memóriába, az utóbbi pedig fájlba menti az adatsorokat.(lásd timeseries_backend 27)

Alkalmazás integrációs HTTP teljesítmény tesztje

A projekt apps/timeseries/test/integration_SUITE.erl fájlában a performance_test_sequential_with_file_backend_via_HTTP, illetve a

performance_test_parallel_with_file_backend_via_HTTP függvények tartalmazzák az alkalmazás integrációs HTTP teljesítmény tesztjét.

A két teszteset 10, 20, 30 darab különböző hosszúságú – 100, 500, 1000, illetve 5000 mérési eseményt tartalmazó – adatsor fel- és letöltésének időtartalmát méri le. A két teszteset között az a különbség, hogy az előbbi sorban hajtja végre a lekérdezéseket, míg az utóbbi párhuzamosan.

A sorban végrehajtott lekérdezésekkel lemérhetjük, hogy egy felhasználó egy lekérdezése mennyi idő alatt teljesül (továbbiakban referencia idő).

A párhuzamosan végrehajtott lekérdezésekkel pedig szimulálhatjuk több felhasználó működését, felmérhetjük, hogy a referencia időhöz képest mennyivel több ideig tartanak a lekérdezések – mennyivel esik vissza a szerver teljesítménye.

Szekvenciális feltöltés (milliszekundum / 10 adatsor)						
Felhasználó \ mérési esemény	100	500	1000	5000	10000	20000
10	6.61	17.12	27.42	167.72	322.22	613.70
20	5.77	18.30	34.40	151.31	288.85	617.82
30	5.21	17.30	36.31	153.21	297.23	614.64
50	4.46	16.78	30.33	141.47	285.21	607.34
Átlagos	**~5.5**	**~17.6**	**~32.2**	**~153.2**	**~309.3**	**~613.9**

Szekvenciális letöltés (milliszekundum / 10 adatsor)						
Felhasználó \ mérési esemény	100	500	1000	5000	10000	20000
10	4.57	8.89	12.71	48.43	94.25	187.20
20	4.80	8.52	13.15	48.55	92.12	187.59
30	4.45	10.23	12.52	48.01	93.00	183.36
50	4.42	8.02	14.40	49.89	95.38	190.12
Átlagos	**~4.5**	**~9.0**	**~13.4**	**~48.8**	**~93.2**	**~187.2**

Párhuzamos feltöltés (milliszekundum / 10 adatsor)						
Felhasználó \ mérési esemény	100	500	1000	5000	10000	20000
10	4.02	14.69	28.60	123.49	253.05	481.91
20	3.81	14.03	28.72	139.98	276.30	487.43
30	3.82	14.67	27.03	139.01	308.31	490.22
50	3.64	14.56	29.94	154.64	358.80	537.54
Átlagos	**~3.8**	**~14.4**	**~28.5**	**~139.3**	**~300.0**	**~497.3**

Párhuzamos letöltés (milliszekundum / 10 adatsor)						
Felhasználó \ mérési esemény	100	500	1000	5000	10000	20000
10	3.86	7.22	10.85	44.82	87.48	176.10
20	3.79	7.19	10.51	47.25	90.51	179.73
30	4.05	7.08	10.82	47.32	114.87	178.52
50	3.80	7.09	10.91	62.81	115.80	195.38
Átlagos	**~3.8**	**~7.1**	**~10.7**	**~50.1**	**~103.3**	**~181.1**

A teljesítmény nem esett vissza (sőt néhol jobban is teljesített) még 50 párhuzamos lekérdezés mellett sem. Az Erlang kifejezetten magas hibatűrésű környezet, eredetileg telekommunikációs rendszerek létrehozására tervezték. Az alkalmazás célja az, hogy kisebb csapatok, projektek egyszerűen kezelhessék adatsoraikat, 50 lekérdezés párhuzamosan bőven több mint amit egy ilyen helyzet igényelhet.

Egységtesztek a timeseries objektumon

Az adatsorokat timeseries objektumokban tárolja az alkalmazás. Az objektumon értelmezett állapot átmenetek tesztelésére – létrehozás, mérési esemény hozzáadása – az objektum moduljában (apps/timeseries/src/timeseries.erl) található egységteszteket hoztam létre.

Tanulságok, következtetések

Az imént említett tesztesetek lefedik a kívánt használat mellett felmerülő esetek túlnyomó részét, kivéve a hibás használat teremtette helyzeteket – ez néhol nehezítette a fejlesztést is. A hibás használatot leíró tesztesetek hiánya azt is jelenti, hogy így nehezebb lenne egy másik alkalmazással összekötni, integrálni. Mivel az integráció nem volt elsődleges célja a projektnek, ezért ez most egy nyilvánvaló továbbfejlesztési lehetősége a projektnek.

Összefoglalás

Elkészült egy webes adatsor vizualizációs alkalmazás, amelynek kliens-oldali részét Ovád Nóra készítette el. Ennek a dolgozatnak a szerveralkalmazás elkészítése volt a célja, amellyel kiszolgálhatom a kliensalkalmazást, tudok adatsorokat tárolni, kiszolgálni, mindezt webes technológiákkal. Ez az alkalmazás jó szolgálatot tenne kisebb méretű, adatsorokat használó, feldolgozó projektekben.

Az alkalmazás létrehozásakor szempont volt, hogy időtálló technológiákat, megoldásokat válasszunk, még úgy is, hogy jelenleg egy szűkebb, jól körülhatárolható funkcionalitással szállítjuk le ezt a megoldást. A továbbfejlesztés lehetőségét figyelembe vettük, mind teljesítményben, mind funkcionalitás terén.

A projekt GitHub-on [7] elérhető, így a valós felhasználásra és talán az alkalmazás továbbfejlesztésére is sor kerülhet később.

Köszönet a dolgozat témavezetőjének, Pataki Norbertnek munkáját és segítőkészségét. Illetve az alkalmazás kliens-oldali fejlesztőjének, Ovád Nórának kitartó munkájáért és lelkesedéséért.

Irodalomjegyzék

[1] W3 sztenderd
https://www.w3.org/standards
(elérhető: 2020-05-10)
[2] wscat program
https://www.npmjs.com/package/wscat
(elérhető: 2020-05-24)
[3] cowboy könyvtár dokumentációja
https://ninenines.eu/docs/en/cowboy/2.8/guide/
(elérhető: 2020-05-02)
[4] REST API
https://en.wikipedia.org/wiki/Representational_state_transfer
(elérhető: 2020-05-16)
[5] gen_server
http://erlang.org/doc/design_principles/gen_server_concepts.html
(elérhető: 2020-05-23)
[6] asdf – függőség kezelő alkalmazás
https://asdf-vm.com
(elérhető: 2020-05-13)
[7] timeseries – GitHub
https://github.com/tortelio/timeseries
(elérhető: 2020-05-30)