

DIPLOMAMUNKA

Mezei Botond, Szabó Benedek

Debrecen
2023

Debreceni Egyetem
Informatikai Kar
Számítógéptudományi Tanszék

Edzőterem működését támogató szoftver PureScript és Vue.js alapokon

DIPLOMAMUNKA

KÉSZÍTETTE:

Mezei Botond és Szabó Benedek
programtervező informatika szakos hallgatók

TÉMAVEZETŐ:

Dr. Battyányi Péter
adjunktus

Debrecen
2023

Tartalomjegyzék

Bevezetés	2
1. Szakirodalmi áttekintés	4
1.1. Funkcionális programozás	4
1.1.1. A funkcionális programozás előnyei	5
1.1.2. A funkcionális programozás nehézségei	6
1.2. PureScript	7
1.3. Vue.js	10
1.3.1. JavaScript keretrendszerek	10
1.3.2. A Vue.js rövid története és tulajdonságai	11
1.4. Java	11
1.4.1. Spring	11
2. A fejlesztés bemutatása	12
2.1. PureScript	12
2.2. Java - Spring backend	12
2.3. Vue.js	12
3. Az eredmények ismertetése	12
3.1. A fejlesztés tapasztalatai	12
3.2. Mérési eredmények	12
Összefoglalás	15
Irodalomjegyzék	15

Bevezetés

A funkcionális programozás egy kevésbé elterjedt programozási paradigma. Noha számos népszerű nyelv használ funkcionális elemeket [7] [12] [13], a teljesen funkcionális fejlesztés nem annyira népszerű. Megismerése az oktatásban sem igazán hangsúlyos. Ennek ellenére számos előnnyel rendelkezik, melyeket a későbbi fejezetekben fogunk részletezni. Másfajta gondolkosámódot igényel, mint az imperatív megközelítés: a "hogyan?" helyett a "mit csináljon a program?" kérdésre adja meg a választ.

A dolgozatban tárgyalt szoftver backend komponensének elkészítésére a PureScript nyelvet választottuk. Ez egy tisztán funkcionális és erősen típusos programozási nyelv, mely Javascript-re fordul. Azért választottuk ezt a nyelvet, mert tisztán funkcionális, a közelmúltban megjelent nyelv, mely a többi funkcionális nyelvhez (Haskell, Elm) képest modernebb, erőteljesebb, könnyebben használható és teljesebb [1].

Munkánk elsődleges célja egy edzőterem működését támogató szoftver elkészítése PureScript backend és Vue.js frontend használatával. Ezen felül szeretnénk megvizsgálni és bemutatni a funkcionális nyelvek használatának előnyeit és lehetőségeit leginkább webalkalmazások fejlesztése során. A szoftver elkészültét követően implementálásra kerül ugyanez a backend Java nyelven a Spring keretrendszer használatával. Célunk a két verzió sebességének és fejlesztési tapasztalatainak összehasonlítása. A sebesség összehasonlítására többféle operációs rendszerrel és különböző specifikációkkal rendelkező számítógépeken ugyanazokat a kéréseket 10, 50 és 100-as kötegekben küldjük el a két különböző szervernek, majd elemezzük az így kapott eredményeket.

A huszonegyedik században egy korszerű edzőteremnek szüksége van egy szoftverre, amely a mindennapi üzletmenetet támogatja. Választásunk azért erre a területre esett, mert egyrészt mindketten szeretünk edzeni járni, másrészt pedig a szoftver összetettsége már alkalmas lehet a PureScript nyelv megismerésére, lehetőségeinek és előnyeinek bemutatására, illetve egy elterjedtebb technológiával (Java, Spring) készült változatával való összehasonlításra.

Az alkalmazás felhasználói az edzőterem recepciós kollégái. A rendszerben lehetőség van rögzíteni:

- vendégeket
- bérlet típusokat (jegy típusokat)
- a felhasználók jegyeit, bérleteit
- öltöző szekrényeket

A *Vendégek* oldalon listázhatjuk a regisztrált vendégeket, újat regisztrálhatunk, a meglévők adatait módosíthatjuk, vendéget törölhetünk. A vendégek számára bérleteket vagy jegyeket lehet vásárolni. A továbbiakban bérletnek nevezzük azt a tagásgot, ami egy időtartamra engedélyezi a terem használatát, jegynek pedig ami alkalmakhoz kötött. A jegyeknek is van egy elévülési ideje. Minden belépőnek három ára van: teljes, diák és klub ár. A felhasználóknak klubtagság vásárlására van lehetőségük, amely különböző előnyökkel jár, például kedvezőbb bérletárak.

A vezérlőpulton lehetőség van a vendégek beléptetésére: a regisztrált vendégek listájából ki kell választani az illetőt. A listában csak az aktív bérlettel vagy érvényes jeggyel rendelkezők jelennek meg. A illetőhöz hozzárendelődik a neve szerint egy szabad öltözőszekrény. Jeggyel rendelkező vendégek számára a jegyen található fennmaradó alkalmak száma eggyel csökken. Az öltözőszekrények megtekintésére egy külön lap szolgál. Itt nemek szerint meg lehet tekinteni minden szekrény állapotát, illetve az előzménytörténetét, vagyis mikor és ki használta. A foglalt szekrények piros színnel jelennek meg.

A vezérlőpulton a beléptetett vendégek kijelentkeztetésére is van lehetőség. Ezt úgy tehetjük meg, hogy a feljövő listából kiválasztjuk az illetőt. Ilyenkor felszabadul a szekrénye.

Minden oldal tetején egy navigációs sáv található, mellyel a megfelelő lapra ugorhatunk.

A közös munka során az egyik legnépszerűbb verziókezelő rendszert, a git-et használtuk a Github szolgáltatóval. Az extrém programozás módszerei közül többet is alkalmaztunk, például a páros programozást, mely hatékonynak bizonyult. Lényege, hogy az egyik fejlesztő írja a kódot, a másik pedig segíti közben.

1. Szakirodalmi áttekintés

1.1. Funkcionális programozás

A funkcionális programozás egy programozási paradigma. Több programozási nyelv is tartozik ide, melyeket bizonyos tulajdonságok, módszereik, lehetőségeik, gondolkodási logikájuk köti össze. Az egyik korai funkcionális nyelv a Lisp, melyet John McCarthy alkotott meg az 1950-es évek végén [4]. Ismertebb funkcionális nyelvek például Haskell, Elm, Erlang, Scala vagy a PureScript. A funkcionális programozás során a fejlesztő azt specifikálja a programban, hogy mit kell kiszámítani, és nem azt, hogy hogyan, milyen lépésekben [3]. A program függvények hívásából és ezek kiértékeléséből áll, nincs értékadás, csak érték kiszámítás [3]. A függvényt leginkább úgy értjük, hogy egy leképezés egy adott halmazról egy másik halmazra [8]. A listák (vagy halmazok) kezelésének ezért kiemelt szerep jut a funkcionális nyelvekben. A program tartalmazhat a nyelvben előre definiált, és a programozó által definiált függvényeket. A függvények névvel és opcionálisan argumentumokkal rendelkeznek. Az előállított érték(ek) megegyező paraméterekkel mindig ugyanaz. A rekurzió egy nagyon gyakori koncepció funkcionális nyelvekben.

Egy másik nagy paradigma az imperatív programozás. Az imperatív nyelvek lényege, hogy a programozó a lépéseket definiálja a kódban, melyet a számítógépnek el kell végeznie (utasítások). A legismertebb imperatív nyelv a C.

A logikai programozási paradigma lényege, hogy a programozó állításokat, szabályokat rögzít, melyek használatával a gép automatikusan kikövetkezteti, hogy egy kérdéses állítás igaz, vagy sem. Ilyen nyelv például a Prolog.

Manapság az egyik legelterjedtebb paradigma az objektumorientált programozás. Ez egy gondolkodásmód, tervezési módszer is egyben. A valós világot osztályok formájára képezi le, melynek egyedeit objektumok személyesítik meg. Legfőbb elvei az egységbezárás (encapsulation: az adatmodell és az eljárásmodell szétválaszthatatlansága), öröklődés (újrafelhasználhatóság kiterjesztése), hozzáférés-szabályozás és többalakúság (polimorphism: lehetővé teszi, hogy ugyanarra az üzenetre különböző objektumok a saját módjukon válaszoljanak) [3].

A ma leginkább használatos nyelvekre általánosságban igaz az, hogy nem csupán egyetlen paradigmát követnek tisztán, hanem többet vegyítenek. Ennek az az oka, hogy minden paradigmának megvan a maga előnye (és persze hátránya is), és ezeket az előnyöket érdemes kihasználni. Például a Java (2023. áprilisában a harmadik legtöbbet használt nyelv [6]) alapjaiban imperatív, objektumorientált, de a nyolcas verziótól kezdve [7] megjelennek benne funkcionális

elemek, például a lambda kifejezések. A nyelv hivatalos oldala [7] "erőteljes kiterjesztés"-nek nevezi ezt a lépést. A Java mellett több elterjedt, nem tisztán funkcionális nyelv, mint például a Python vagy a C++ is épít be funkcionális elemeket [12] [13].

1.1.1. A funkcionális programozás előnyei

A funkcionális programozás, mint paradigma, számos előnnyel rendelkezik. Az imperatív gondolkodás kötöttségét egy másfajta megközelítéssel oldja fel. A lényeg, hogy *mit* csináljon a program, és nem az, hogy *hogyan*. Ez az ötlet a programozótól is másfajta megközelítést, gondolkodásmódot kíván, és máshogy strukturált kódot is fog eredményezni.

Ránézésre jobban érthető, átláthatóbb, esztétikusabb kód. A szoftver "viselkedése" jobban olvasható [1]. Ez az előny egyszerűen a funkcionális szintaktikából és gondolkodásmódból adódik. A következő Java kódrészletek Boris Radojicic 2022-es cikkéből [8] származnak. Az első az iteratív megközelítés látható:

```
public List<String> getAddresses(List<Person> persons) {
    List<String> addresses = new ArrayList<>();
    for (int i = 0; i < persons.size(); i++) {
        Person person = persons.get(i);
        if (person.isValidData()) {
            String address = person.getAddress();
            addresses.add(address.trim());
        }
    }
    return addresses;
}
```

A következő kódrészlet pedig a funkcionális megközelítést alkalmazza:

```
public List<String> getAddresses(List<Person> persons) {
    return persons.stream()
        .filter(person -> person.isValidData())
        .map(person -> person.getAddress())
        .map(address -> address.trim())
        .collect(Collectors.toList());
}
```

Míg a két kód ugyanazt a viselkedést eredményezi, a kettőre ránézve elmondható, hogy a második olvashatóbb és tömörebb.

Nagy fokú újrafelhasználhatóság. Mivel egy funkcionális program gyakorlatilag függvények deklarálásából és függvényhívásokból áll, ezeket a függvényeket nagy mértékben újra lehet hasznosítani, elkerülve a feleslegesen duplikált kódrészleteket. A felesleges kód rontja az átláthatóságot és több erőforrást is igényel.

Könnyebb tesztelhetőség. Mivel a funkcionális programban nincsenek állapotok és mellékhatások, így könnyebb lefedni az összes esetet. Illetve ebből kifolyóan, a függvénynek bizonyos bemenő paraméterekre mindig ugyanazt az egyértelmű kimenetet kell előállítania.

A rekurzió hatékony használata. A rekurzió a funkcionális nyelvek gyakori eleme. Átláthatóvá és egyértelművé teszi a kódot teszi a kódot bizonyos problémák esetében, ahol a rekurzív definiálás a természetesebb megoldás. A rekurciónak egy speciális esete a farokrekurzió. Ez azt jelenti, hogy a rekurzív hívás a legutolsó művelet, amit a függvény végrehajt. Nagy előnye, hogy idő és erőforrás takarékos a nem farokrekurzív függvényekkel szemben, ugyanis a fordító kevesebb információt kell, hogy tároljon ilyen módon a veremben.

1.1.2. A funkcionális programozás nehézségei

Az előnyök között első helyen állt az átláthatóság és érthetőség. Ehhez viszont hozzátartozik az, hogy a fejlesztő előtte megismerje és elsajátítsa a funkcionális programozás gondolkodásmódját, ami néha bizony nem egyszerű és nem a legtermészetesebb megoldásnak tűnhet.

Memóriahasználat és teljesítmény. Mivel a funkcionális nyelvek nem használnak érték-átadást, sokszor egy változó értékének megváltozása helyett létrejön egy újabb, ami így több erőforrást igényel [9]. A legtöbb funkcionális nyelv ennek ellenére számos módon igyekszik ezt kompenzálni: farokrekurzió, lusta kiértékelés, "smart linking". A több paradigmát támogató vagy OOP nyelvek nem feltétlen implementálják ezeket az optimalizációkat [9].

Mivel a funkcionális nyelvek használata kevésbé elterjedt, kevesebb eszköz, keretrendszer áll rendelkezésre és kevesebb a felhasználók, szakértők száma. Ennek következtében a fejlesztést nehezíti néha a dokumentáció hiányossága vagy teljes hiánya, illetve az interneten fellelhető minta kódok hiánya és elavultsága.

1.2. PureScript

A nyelvet Phil Freeman tervezte 2013-ban, ugyanis nem volt megelégedve a Haskell-t Javascript-re fordító próbálkozásokkal (például Fay, Haste, vagy GHCJS) [15]. A nyelv teljes forráskódja és dokumentációja megtalálható a GitHubon [16]. Hasznos dokumentációs forrás továbbá a Pursuit [18], illetve a nyelv alkotója által publikált PureScript by Example könyv [2].

A PureScript egy erősen típusos, tisztán funkcionális nyelv. Javascript kódra fordul. Ennek előnye, hogy a kód írására egy szép, könnyen áttekinthető és egyértelmű nyelvet használunk, amiből egy hatékony Javascript kód generálódik. Amit Javascriptben meg lehet írni, azt nagyjából PureScriptben is [1]. A böngészőben és a szerveren egyaránt futtatható.

A nyelv alapja a Haskell, nagyban hasonlít rá. Maga a szintkaxis annyira hasonló, hogy néhány fejlesztői eszköz a kód megjelenítéséhez a Haskell kiemelését használja. A nyelv fordítója is Haskell nyelven íródott [1]. Charles Scalfani [1] szerint erősebb, de egyben könnyebben is használható, mint a Haskell.

A nyelv komoly előnye, hogy frontend és backend fejlesztésre egyaránt alkalmas lehet [1].

A PureScriptnek saját csomagkezelő szoftvere van, a Spago [17]. Forráskódja szintén elérhető a GitHubon. Munkánk során mi is ezt használtuk, véleményünk szerint nagyon egyszerűen használható, és probléma sem merült fel vele kapcsolatban.

A PureScript rendelkezik egy parancsoros végrehajtási móddal, ezt PSCI-nek vagy REPL-nek nevezik. Soronként történik a kód begépelése és végrehajtása. Több sort is megadhatunk egyszerre a *:paste* paranccsal (kilépés: ctrl + D). A nyelvvel való ismerkedés és a tesztelés során hasznos lehet, ugyanis lépésről lépésre tudjuk irányítani a kódot, például típusok lekérdezéséhez.

A következő kódcsipet a *Hello, world!* program PureScript változata:

```
module Main where

import Prelude

import Effect (Effect)
import Effect.Console (log)

main :: Effect Unit
main = log "Hello World!"
```

Előre definiált könyvtárakból lehetőség van függvényeket importálni, ezt a fájl elején tehetjük meg. Elég explicit módon kell megadni a konkrét behívandó

függvényeket és komponenseket. Ezután (mint tiszta funkcionális nyelvhez illik) függvények deklarációja és definíciója következik. A következő kódcsipet egy függvényt ábrázol:

```
add :: Int -> Int -> Int
add a b = a + b
```

Az első sor nem kötelező, de az átláthatóság érdekében érdemes megadni. Azt lehet belőle leolvasni, hogy az `add` egy olyan függvény, amely két `Int` típusú input paraméterrel rendelkezik, és kimeneti értéként egy `Int`-et állít elő. A függvény a következőt csinálja: [1]:

- kap egy vagy több bemeneti értéket
- elvégez bizonyos számításokat
- visszaad egy értéket

Egy tiszta függvény mellékhatás mentes, és mindig ugyanazt a kimenetet állítja elő[1]. A tiszta funkcionális elvek szerint egy függvénynek csak egy kimeneti értéket adhat vissza. Ez nehézségnek tűnhet, de az úgy nevezett *Currying* koncepció segíthet: egy több paraméteres függvény egy paraméteres függvények halmazára cserélését jelenti.

A nyelv további jellemzője, hogy minden változó "Immutable", vagyis a létrejöttük után nem módosulhatnak. Ez is egy érdekes tulajdonság az imperatív világban járatosabb fejlesztők számára. Az imperatív nyelvekben is van lehetőség objektumoknak ilyen tulajdonság beállítására. PureScriptben a *let* kulcsszó gyakran segítségünkre lehet ilyenkor, például:

```
add :: Int -> Int
add x = do
  let y = 20
  x + 20
```

A fenti függvény visszaadja a bemeneti értéknél hússzal nagyobb integert. PureScriptben a sorok behúzása is lényeges, a Pythonéhoz hasonló elven működik. Ott, ahol a legtöbb nyelv zárójeleket használna, a PureScriptben behúzás található: eggyel bentebbi behúzás egy zárójelezett blokknak felel meg.

Az imperatív nyelvek egyik alap építőelemét, a ciklusokat is el kell felejteni PureScriptben. Más megközelítés helyettük például a rekurzió használata, mely a funkcionális nyelvekben gyakori koncepció. Példaként Charles Scalfani könyvéből [1] mutatok két Javascript kódcsipetet, melyek ugyanazt a célt szolgálják. A ciklusos megközelítés:

```
const factorial = n => {
  var result = 1;
  for (var i = 1; i <= n; ++i)
    result = result * i;
  return result;
};
```

Funkcionális módon:

```
const factorial = n => n === 0 ? 1 : n * factorial (n - 1);
```

A második kódcipet úgy definiálja a függvényt, hogy azt mondja meg, *mit* számoljon ki, míg az első a *hogyan* kérdésre adja meg a választ. Ugyanez a kód PureScriptben:

```
factorial :: Int -> Int
factorial n =
  if n = 1 then 1 else factorial $ n - 1
```

A fenti két funkcionális függvény farokrekurzív, mivel a rekurzív hívás maga a függvényben az utolsó hívás. A rekurzió egy előnyös fajtája, mivel kevesebb memóriát igényel. Ez annak köszönhető, hogy ily módon a veremben kevesebb információ tárolódik, és a sebesség is szignifikánsan jobb lesz egy nem farokrekurzív függvényhez képest.

A PureScript továbbá statikus nyelv. Ez azt jelenti, hogy a típusok ellenőrzése fordítás alatt zajlik, és nem futásidő alatt. Előnye, hogy a hibák már fordítás közben kiderülnek, kevesebb tesztelés szükséges. Hátránya pedig, hogy a típusokra több figyelmet kell fordítani, limitálja a programozói szabadságot [1].

A nyelvvel kapcsolatban kissé negatívum élményként tapasztaltuk, hogy az általános dokumentáltság (nem magának PureScript nyelvnek a dokumentációja, inkább a rendelkezésre álló egyéb oktatási anyagok, könyvek, példák, megválaszolt kérdésekre gondolva) hiányos, sokszor nehéz volt emiatt haladni a fejlesztéssel. Ez véleményünk szerint egyrészt a kevésbé népes funkcionális fejlesztői közösségnek köszönhető, másrészt pedig a nyelv fiatal korának.

1.3. Vue.js

1.3.1. JavaScript keretrendszerek

A Vue.js egy klines-oldali keretrendszer, mely a JavaScript programozási nyelvre épül. A JavaScript 1996-os indulása óta mára már megkerülhetetlen része a webnek, a weben található oldalak több mint 95%-án használatban van valamilyen formában [14]. Az évek során a nyelvvel dolgozó fejlesztőknek köszönhetően számos különböző problémára megoldást nyújtó könyvtár született, mely mind a web mai formájának alakulásához járult hozzá.

A keretrendszerek is ilyen könyvtárak. Meghatározzák egy abban írt szoftver felépítését, mely olyan előnyökkel jár mint a kiszámíthatóság, fenntarthatóság illetve a skálázhatóság. A keretrendszerek megépítésének motivációja a fejlesztési munkamenet egyszerűsítésében keresendő. Nem várta fel új képességekkel az eredeti programozási nyelvet (jelen esetünkben a JavaScriptet), hanem átláthatóbbá valamint könnyebben elérhetővé teszi azt. Általuk a fejlesztőknek csak azt kell meghatározniuk, hogy hogyan nézzen ki az általuk elkészíteni kívánt felhasználói felület, a megvalósításról pedig már a keretrendszer gondoskodik.

Az alább felsorolt tulajdonságok szintén a fejlesztési folyamatokat megkönnyítéséhez járulnak hozzá:

- tesztelési és kódellenőrzési eszközök biztosítása,
- felhasználói felület részekre (komponensekre) osztása,
- útvonalkezelés megkönnyítése.

Az elvitathatatlan előnyökön felül természetesen van más nézőpont is ami felől vizsgálandó szükséges egy fejlesztőnek, mielőtt egy keretrendszer használatába kezd, az alap JavaScriptet elfeledve. Fontos észben tartani, hogy a keretrendszerek elsajátításához időre van szükség, hogy az általa biztosított eszközöket a lehető leghatékonyabban tudja felhasználni a fejlesztő a céljai elérésére. Elenghetetlen továbbá átgondoli az elkészítendő program célját. Egy olyan alkalmazás, amely semmilyen felhasználói interakcióval nem rendelkezik, nem biztos, hogy igényel egy olyan professzionális fejlesztői eszközt, mint egy keretrendszer, mely komplex struktúrája feleslegesen bonyolítja a fejlesztési folyamatot.

Ahhoz, hogy az elérhető keretrendszerek közül a legmegfelelőbb kerüljön kiválasztásra, érdemes szem előtt tartani néhány egyszerű szempontot. A választott keretrendszer dokumentációjának minősége és elérhetősége. Minél részletesebb, minél több példával rendelkezik, annál könnyebb lesz a fejlesztés

során felmerülő problémákra megoldást találni vagy az implementációs kérdésekben döntést hozni. A dokumentációt kiegészítő szempont a fejlesztői közösség aktivitása, amely a speciális esetekben nyújtat segítséget, akár korábban feltett és megválaszolt kérdéseken keresztül.

1.3.2. A Vue.js rövid története és tulajdonságai

A Vue.js ötlete Evan You, Google alkalmazott fejében született meg. A munkája során szüksége lett volna egy olyan hatékony eszközre, ami gyorsan képes nagy mennyiségű HTML dokumentumot előállítani adatok és prototípusok felhasználásával. A már létező eszközök nem tudták teljes mértékben kielégíteni az igényeit. Volt olyan, amelyik csupán struktúrát biztosított, de az adatkezelése nem volt ideális (Backbone), de volt olyan is, amelyik túl szigorú, kötött szabályrendszerével nem csak a fejlesztendő alkalmazás struktúráját, de a kódolási folyamatot is túlságosan nagy mértékben kívánta befolyásolni (Angular). A megfelelő megoldás hiánya arra készítette, hogy saját maga hozza létre az általa keresett keretrendszert. Így jelent meg a keretrendszer 2014-ben, mely azóta egy lelkes fejlesztő kis magánprojektjéből széles fejlesztői körökben elterjedt és folyamatosan fejlődő keretrendszerré nőtte ki magát.

1.4. Java

1.4.1. Spring

2. A fejlesztés bemutatása

2.1. PureScript

2.2. Java - Spring backend

2.3. Vue.js

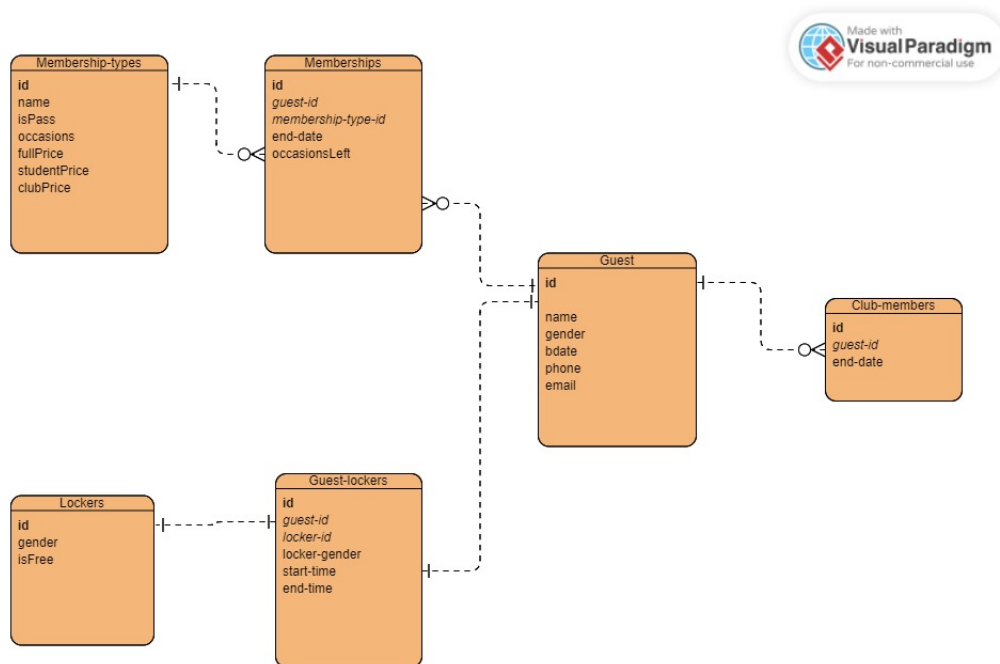
3. Az eredmények ismertetése

3.1. A fejlesztés tapasztalatai

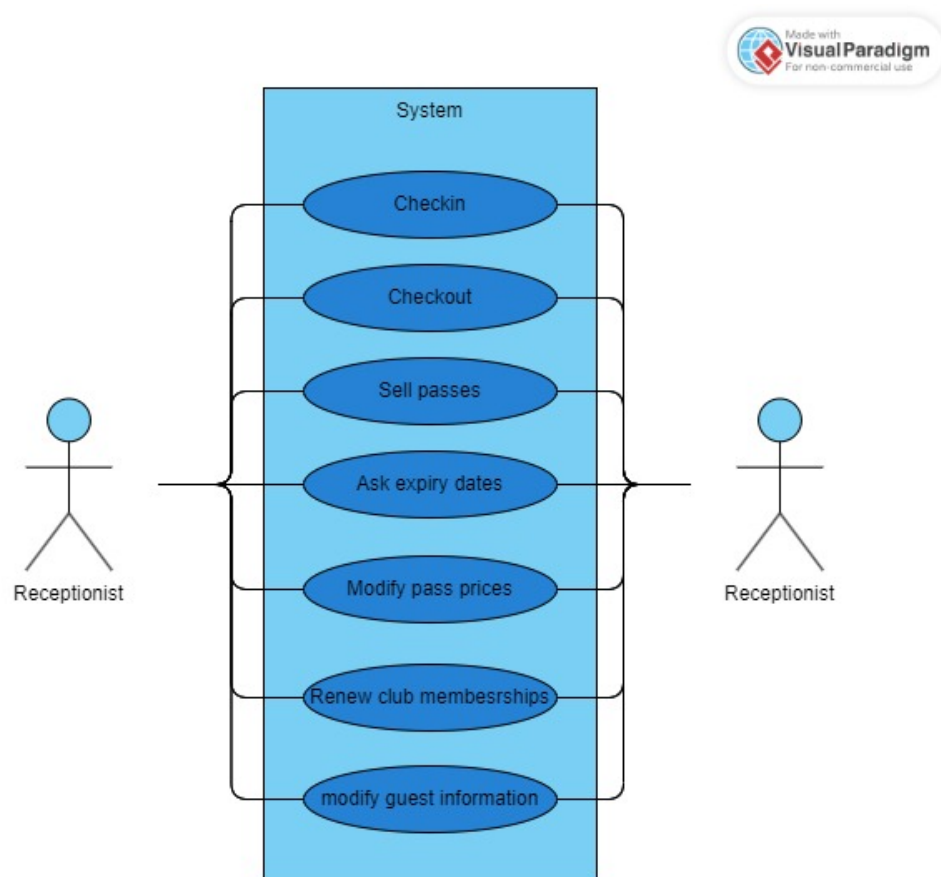
3.2. Mérési eredmények

Metódus	Útvonal	Leírás
GET	/guest/getAll	Az összes vendég lekérése.
GET	/guest/getById/{id}	Egy vendég lekérése az azonosítója alapján.
POST	/guest/insertGuest	Egy új vendég hozzáadása.
PUT	/guest/updateGuest/{id}	Egy vendég kártyájának szerkesztése
DELETE	/guest/deleteGuest/{id}	Egy vendég törlése
...

1. táblázat. A kérések listájának részlete.



1. ábra. A szoftver adatbázis modellje.



2. ábra. Használati eset (use case) diagram.

Összefoglalás

Irodalomjegyzék

- [1] C. Scalfani. Functional Programming Made Easier: A Step-by-Step Guide. 2021.
- [2] P. Freeman. PureScript by Example. 2014 - 2017. <https://book.purescript.org/>
- [3] Dr. Vadász Dénes: Programozási paradigmák, programozási nyelvek (letölthető egyetemi oktatási anyag) <https://web.archive.org/web/20150501083657/http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/DVadasz/GEIAL401/Progpar-4-fejezet.pdf#>
Hozzáférés dátuma: 2023.04.11.
- [4] John McCarthy: The implementation of LISP. 1996. <http://www-formal.stanford.edu/jmc/history/lisp/node3.html>
- [5] Szuromi Zs.: Programozási paradigmák, kézirat. ME, 1996.
- [6] The TIOBE Programming Community index. <https://www.tiobe.com/tiobe-index/> Hozzáférés dátuma: 2023.04.11.
- [7] A Java nyelv hivatalos honlapja. <https://dev.java/learn/lambda/>
Hozzáférés dátuma: 2023.04.11.
- [8] B. Radojicic.: Imperative to Functional Programming in Java. 2022 <https://symphony.is/blog/imperative-to-functional-programming-in-java> Hozzáférés dátuma: 2023.04.11.
- [9] J. Neumann.: Advantages and disadvantages of functional programming. 2022. <https://medium.com/twodigits/advantages-and-disadvantages-of-functional-programming-52a81c8bf446>
Hozzáférés dátuma: 2023.04.12.
- [10] <https://www.purescript.org/>
- [11] <https://vuejs.org/guide/introduction.html>
- [12] J. Sturtz.: Functional Programming in Python: When and How to Use It. Hozzáférés dátuma: 2023.04.20. <https://realpython.com/python-functional-programming/>

- [13] D. Cravey.: Functional-Style Programming in C++.
Hozzáférés dátuma: 2023.04.20. <https://learn.microsoft.com/en-us/archive/msdn-magazine/2012/august/c-functional-style-programming-in-c>
D. Cravey.: Functional-Style Programming in C+.
Hozzáférés dátuma: 2023.04.20.
- [14] Kliens-oldali programozási nyelvek használatát bemutató statisztika.
<https://w3techs.com/technologies/details/cp-javascript>
Hozzáférés dátuma: 2023.04.23.
- [15] A PureScript nyelv hivatalos dokumentációja. <https://github.com/purescript/documentation> Hozzáférés dátuma: 2023.04.24.
- [16] A PureScript nyelv hivatalos forráskódja. <https://github.com/purescript/purescript> Hozzáférés dátuma: 2023.04.24.
- [17] A Spago hivatalos dokumentációja. <https://github.com/purescript/spago> Hozzáférés dátuma: 2023.04.24.
- [18] Pursuit. Hivatalos PureScript dokumentáció. <https://pursuit.purescript.org> Hozzáférés dátuma: 2023.04.24.