

# 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

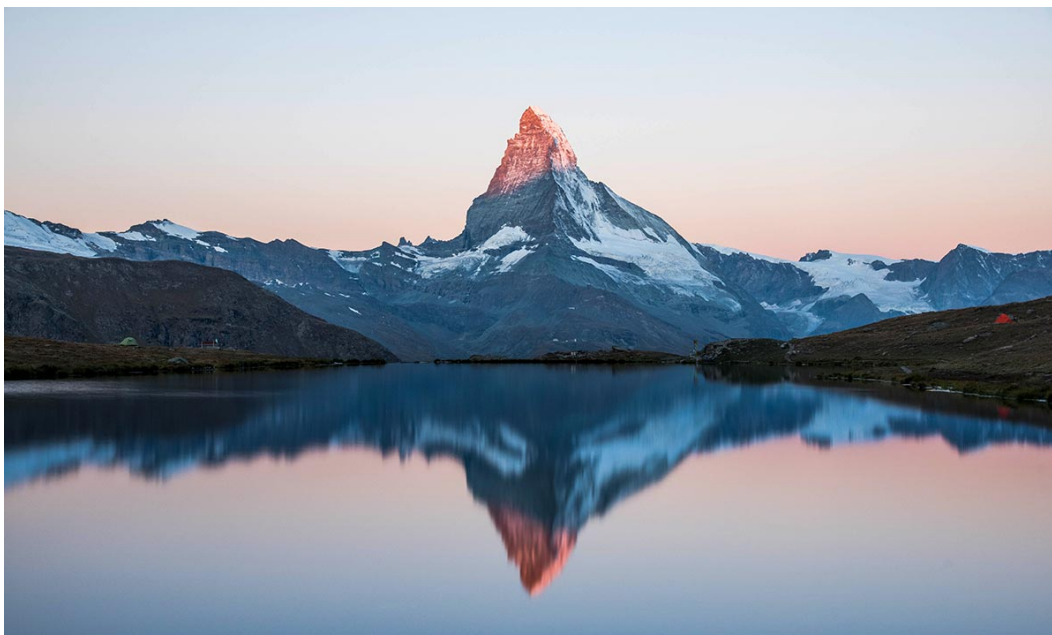
TÉMAVEZETŐ:

Dr. Battyányi Péter  
adjunktus

Debrecen  
2023

# Tartalomjegyzék

Bevezetés	2
1. Szakirodalmi áttekintés	3
1.1. Funkcionális programozás . . . . .	3
1.1.1. A funkcionális programozás előnyei . . . . .	4
1.1.2. A funkcionális programozás nehézségei . . . . .	5
1.2. PureScript . . . . .	5
1.3. Vue.js . . . . .	5
2. Eredmények és azok értékelése	6
3. Továbbfejlesztési lehetőségek	7
Összefoglalás	8
Irodalomjegyzék	8



1. ábra. Az ARIS objektumai, és az ARIS ház felépítése. A jobb oldali ábra forrása: *Fonó, 2005*

## Bevezetés

miert ezt választottuk

mi a munkánk és a dolgozat célja

A huszonegyedik század harmadik évtizedében minden vállalatnak, cégnek vagy szervezetnek elengedhetetlenül szükségessé vált az informatika bevonása a mindennapi működésbe. A cégek versenyképességük megőrzése, a hatékonyság és profitabilitás növelése érdekében már gyakorlatilag egy mikrovállalkozás méretétől kezdve rászorulnak valamiféle speciális céges szoftver, vállalatirányítási rendszer használatára. Egy kicsit nagyobb vállalatnál pedig már egészen elképzelhetetlen lenne a működés e szoftverek nélkül. A költségcsökkentés és automatizálás mellett meg kell említeni az információ növekvő szerepét, egy vállalatirányítási rendszer biztosíthatja a gyors (valós idejű) információ hozzáférést a munkatársaknak, illetve adatokkal látja el a döntéshozókat.

# 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.

### 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 elsőt 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.

## **1.2. PureScript**

## **1.3. Vue.js**

## 2. Eredmények és azok értékelése



### 3. Továbbfejlesztési lehetőségek

A vizsgált és specifikációk készítésekor használt modellezési nyelvek (ARIS, BPMN, UML, Petri-hálók) közül én az ARIS-t találtam jelen feladathoz a legmegfelelőbbnek. A különböző diagramtípusok integrációja és azok rendszerezett kezelése mindenképp kiemeli a többi közül. Ezen felül leginkább a folyamatok leírásakor az általa kínált információtöbbletet is hasznosnak találtam, véleményem szerint a többinél nagyobb kifejezőerővel rendelkezik ez az eszköz.

## Összefoglalás

A mai korszerű vállalatoknak versenyképességük megőrzése céljából elengedhetlenné vált üzleti szoftverek alkalmazása. Dolgozatom célkitűzése egy debreceni kereskedelmi profilú kisvállalkozás folyamatainak, működésének megismerése, modellezése és a számára készülő vállalatirányítási rendszer üzleti követelmény specifikációjának elkészítése volt.

## 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.