ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Szvoboda Bálint**

2022. Hallgató törzskönyvi száma: **T/005280/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Szvoboda Bálint**
Törzskönyvi száma: T/005280/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Horgász napló és fogás elemző mesterséges intelligencia segítségével webes és mobil környezetben
Fihsing diary and catch analyst with artificial intelligence in web and mobil environment

Intézményi konzulens: Balázs Elemér
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

Készítsen egy olyan rendszert, melyben területenként rögzíthetők legyenek horgászatok közben fogott halak adatai és a horgászatok körülményei (csali, időjárás, stb.). Az alkalmazás részben az adatok rendszerezett tárolásában nyújtson segítséget, melyet az érintett felhasználók grafikus formában meg tudjanak tekinteni. Mindezek mellett a program adjon javaslatot arra, hogy az aktuális időjárási viszonyok függvényében, milyen csalival érdemes horgászni. Az ajánlás előállításához gépi tanulás alapú megközelítést alkalmazzon.
A rögzített adatokról készítsen különféle statisztikákat, melyek típusait a tervezési szakaszban határozza meg.
Az alkalmazás legyen elérhető webes, illetve mobilos környezetben egyaránt.

A dolgozatnak tartalmaznia kell:

- a feladat leírását,
- a kapcsolódó irodalom részletes feldolgozását,
- hasonló rendszerek bemutatását és jellemzését,
- az alkalmazás teljeskörű tervezését és a kiválasztott technológiák indoklását,
- a kialakított gépi tanuláson alapuló módszer ismertetését,
- az adatokból kinyerhető statisztikák meghatározását,
- az alkalmazás tesztelési jegyzőkönyvét és eredmények bemutatását,
- továbbfejlesztési lehetőségeket,
- a dokumentációt, a programot, a szükséges inputadatokat, valamint a rendszert bemutató honlapot és prezentációt CD mellékleten.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * Budapest
Ph.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. május 10.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: SZVOBODA BÁLINT

Neptun kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

HORGÁSZ NAPLÓ ÉS FOGÁS ELEMZŐ MESTERSÉGES INTELLIGENCIA SEGÍTSÉGÉVEL WEBES ÉS MOBIL KÖRNYEZET BEN

Szakdolgozat / Diplomamunka[2] címe angolul:

FISHING DIARY AND CATCH ANALYST WITH ARTIFICIAL INTELLIGENCE IN WEB AND MOBIL ENVIRONMENT

Intézményi konzulens: BALÁZS ELEMÉR

Külső konzulens: —

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 12.	PROJEKT INICIALIZÁLÓ MEGBESZÉLÉS	
2.	2021. 10. 03.	HASONLÓ RENDSZEREK ÁTTEKINTÉSE	
3.	2021. 11. 04.	RENDSZERTERV PONTOSÍTÁSA	
4.	2021. 12. 05.	ELÉRT EREDMÉNYEK VÉLEMÉNYEZÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 20.21. 12. 09.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: SZVOBODA BALINT

Neptun kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

HORGÁSZ NAPLÓ ÉS FOGÁS ELEMZŐ MESTERSÉGES INTELLIGENCIA SEGÍTSÉGÉVEL WEBES ÉS MOBIL KÖRNYEZETBEN

Szakdolgozat / Diplomamunka[2] címe angolul:

FISHING DIARY AND CATCH ANALYST WITH ARTIFICIAL INTELLIGENCE IN WEB AND MOBIL ENVIRONMENT

Intézményi konzulens: BALÁZS ELEMÉR

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.22.	BESZÁMOLÓN JAVASOLT ÁTALAKÍTÁSOK MEGBESZÉLÉSE	
2.	2022.03.29.	CSALI AJÁNLÓ MODUL BEMUTATÁSA	
3.	2022.04.23.	SZERVER KOMPONENS DEMONSTRÁLÁSA	
4.	2022.05.08.	TESZTELÉS, LEADANDÓ DOKUMENTUMOK ÁTTEKINTÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.05.08.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

1. Bevezetés

Az alkalmazás egy a horgászathoz használható fogási naplót valósít meg részben, amelyben a felhasználó eltárolhatja a horgászatot, annak időpontját és körülményeit. Az adott horgászathoz el lehet menteni a kifogott halakat, illetve, hogy milyen csalival sikerült megfogni őket. Később ezeket a horgászatokat vissza lehet nézni. Statisztika mutatja, hogy melyik időszakban, melyik csalival sikerült a legtöbb halat zsákmányolni, ezt lehet horgászatonként és összesítve is. Továbbá meg lehet nézni a legnagyobb kifogott halak listáját összesítve és halfajonként is.

A korábbi tapasztalatok és az általános vélekedések alapján a rendszer egy gépi tanulás algoritmus segítségével ad egy ajánlást, hogy az adott körülmények között a horgász milyen csalival próbálkozzon, amivel, jó eséllyel sikeres lehet.

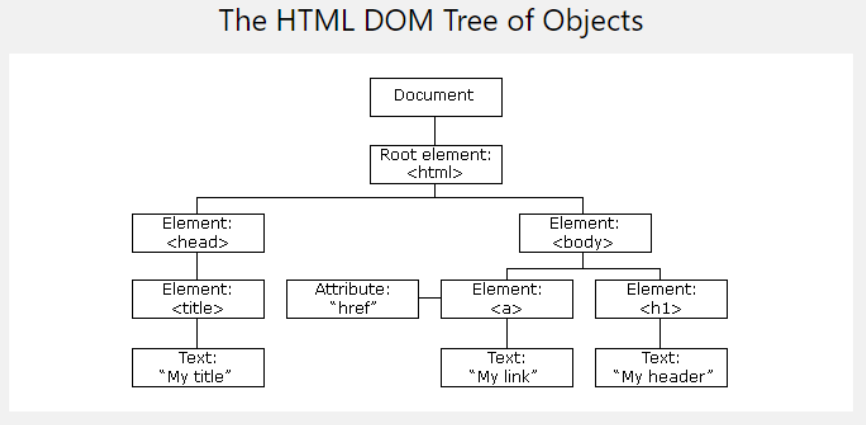
A rendszer áll egy web service-ből, ami a csali ajánló rendszerrel és több különböző felhasználói felülettel is kommunikál. Az alkalmazás webes és mobilos platformon is elérhető lesz az egységes JavaScript alapú kliensnek köszönhetően.

2. Irodalomkutatás

Az irodalomkutatás során azokat a technológiákat vettem górcső alá, amelyek potenciálisan felhasználhatóak a projekt során.

2.1. HTML

A HTML (HyperText Markup Language) egy leíró nyelv. Ennek segítségével meghatározható a weboldal struktúrája. Különböző tartalmakat, elemeket adhatunk hozzá, mint például paragrafusok, listák, címsorok, képek és linkek. A HTML oldalak különböző elemekből épülnek fel, ezeknek van nyitó és záró tag-je, ezen belül lehetnek attribútumai is. Az egyik ilyen például a class amiben különböző stílusokat adhatunk át az elemnek, hogy hogyan nézzen ki. A HTML oldalt két nagy részre lehet bontani a head és a body. A fejben van az oldal leírása, címe, a karakter kódolás. A másik részben pedig a tartalom, ami megjelenik az oldalon. [1]
A DOM (Document Object Model) a HTML oldal objektumai, amik egy fa struktúrába rendeződnek *(2.1. ábra)*. Definiálja az elemek tulajdonságait, metódusait és eseményeit is. [2]



2.1. ábra: Egy példa DOM fa felépítése (forrás: [2])

2.2. CSS

A CSS (Cascading Style Sheets) segítségével adhatunk stílust a különböző HTML elemeknek. Ezt megadhatjuk, az adott elemen belül, létrehozhatunk neki egy külön blokkot vagy egy másik fájlban megírható a felhasználandó stílus. Előre megírt keretrendszerek is vannak, amiket fel lehet használni. Ilyenek például a Bootstrap, Tailwind, Semantic UI, Foundation, Bulma. Ezeket különböző csomagkezelőkből (npm, gem, yarn) lehet telepíteni.

2.2.1. Bootstrap 4

Az egyik legnépszerűbb front-end keretrendszer, nyílt forráskódú. 2010-ben készült el a Twitter fejlesztői által. Jelenleg a 4. verziónál tart. [3] Könnyen készíthető vele olyan weboldal, ami mobil eszközökön is reszponzív lesz.

2.3. JavaScript

A világ egyik legnépszerűbb programozási nyelve. Legtöbben a weboldalak parancsnyelveként ismerik, de sok a webböngészőn kívüli környezetben is használják. Ilyen a Node.js, Apache CouchDB és az Adobe Acrobat. A JavaScript egy prototípus-alapú, dinamikus nyelv, ami támogatja az objektum orientált programozási stílust is. A JavaScript szabványa az ECMAScript. 2015-ben volt egy nagy változtatás ekkor adták ki az ES6 verziót. Azóta a szabványokat éves ciklusokban adják ki. Manapság egyre jobban elterjedtek és előszeretettel használják a JavaScript keretrendszereket, ezek közül is a három legelterjedtebb az Angular, a React.js és a Vue.js. [4]

2.3.1. React.js

A React.js virtual DOM technikát használ csak azokat a komponenseket frissíti és jeleníti meg újra, ahol változások történtek. Támogatja a komponens alapú fejlesztést, hogy egy elemet többször is fel lehessen használni. A keretrendszer ad lehetőséget Hook-ok használatára. ezekből kétféle van State Hook és Effect Hook. Az előbbi értesítés küld, ha megváltozik az objektum így újra meg lehet jeleníteni a virtual DOM segítségével. A másik segítségével mellékhatásokat válthatunk ki a komponensekben. Például, ha betöltött egy komponens akkor hajtson végre valamilyen utasítást. [5]

2.4. Cross-platform mobil alkalmazások

Két féle cross-platform mobilalkalmazás létezik a natív cross-platform és a hibrid. A natív cross-platfrom alkalmazás. Minden operációs rendszer rendelkezik a saját SDK-jával. Ez Andorid esetén lehet Java vagy Kotlin, iOS alkalmazásoknál pedig Objective-C vagy Swift. A fejlesztők létrehoztak egy egységesített API-t, ami a natív SDK fölött fut. Így azonos kódbázist használnak iOS és Android alatt is. A natív cross-platfrom alkalmazásokat többnyire Xamarinnal és React Nativevel készítik. A hibrid alkalmazások, amióta az Android és iOS SDK-k tartalmaznak webes komponenseket is, azóta a felhasználói felületek készíthetők HTML, CSS és JavaScript segítségével. A kódot becsomagolják úgy az alkalmazásba mintha egy böngészőben futna. Az alkalmazások korlátozottan férnek hozzá az okostelefonok funkcióihoz. A leggyakrabban használt hibrid keretrendszerek az Apache Cordova és a Flutter. A cross-platform alkalmazások előnyei. Rövidebb fejlesztési idő. A kód újrahasználhatóság miatt. Nagy elérhetőség. Sokféle eszközön, illetve operációs rendszeren tud futni az alkalmazás. Könnyebb frissíteni az alkalmazást. Manapság sűrűn változnak a szoftverek, nem kell külön platformonként elvégezni a frissítést. A másik oldalról pedig a hátrányai. Teljesítmény, nagy CPU és GPU erőforrást igénylő feladatok esetén jelentős különbség van natív és hibrid alkalmazások között. Nehezebben tanulható. Aki platform független alkalmazást kell írnia, ismernie, kell mindkét oldalt az iOS-t és az Androidot is. Korlátozott támogatás, nem minden külső könyvtár és SDK működik jól a cross-platform keretrendszerben. [6]

2.4.1. Xamarin

C#-ban és .Net-ben íródott. Cross-platform alkalmazásokat lehet vele készíteni Android, iOS, tvOS, macOS és Windows alkalmazásokat. Ha platform specifikus interfészekre van szükség, akkor a fejlesztő tudja használni a különböző könyvtárakat hozzá. Lehetőség van az eszközök szenzorjait és egyéb funkcionalitásait is igénybe venni. [7]

2.4.2. React Native

A Facebook mutatta be. A React JavaScript könyvtár alapjaira építették. A React Native közel 80%-ban azonos kódbázist használ. Az okostelefon kamerájának és egyéb funkcióinak használatához külön kódot kell írni Androidra és iOS-re. [8]

2.4.3. Flutter

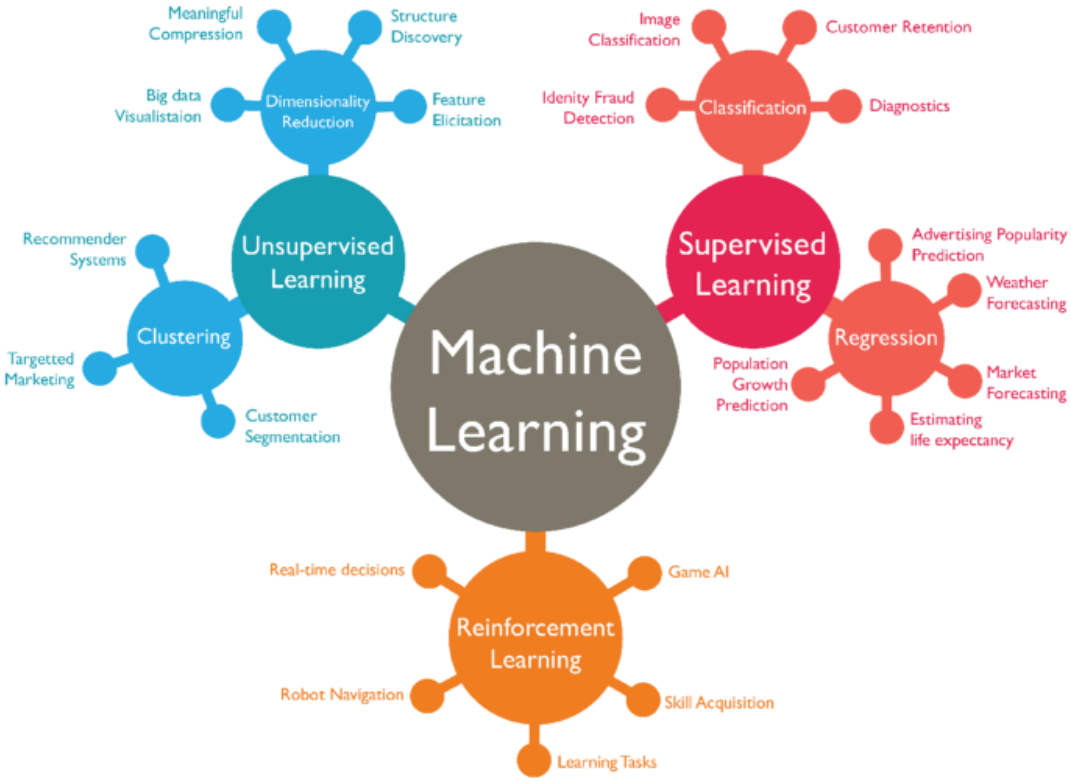
A Flutter a Googlenek a felhasználói felület eszköztára. Azonos kódbázissal lehet készíteni alkalmazásokat mobilra, böngészőbe és asztali számítógépre is. A Flutter alapjai Dart nyelven íródtak. Ez egy újabb programozási nyelv, ami a Swift és Kotlin kódokat átfordítja JavaScriptre. A keretrendszer jól illeszkedik az MVP (Minimum Viable Product) fejlesztéshez. [9]

2.5. ASP.NET MVC

Az ASP.NET MVC egy keretrendszer webalkalmazásokhoz, amit a Microsoft fejlesztett ki. Amiben ötvözte az MVC (Model-View-Controller) architektúrát a már meglévő ASP.NET platformmal, így lehetővé teszi a naprakész technológiák és az agilis módszertan használatát. A korábbi verziója az ASP.NET Web Forms volt. A keretrendszernek köszönhetően jól karbantartható a kód jól feloszthető külön egységekre az MVC tervezési minta alapján. Támogatja a TDD (Test-Driven Development) fejlesztési módszertant. [10]

2.6. Gépi tanulás

A gépi tanulás során olyan rendszerek jönnek létre, amelyek képesek az adatokból történő tanulásra és következtetésre. Olyan összetett problémák vagy nagy adatmennyiség esetén alkalmazhatjuk, amikor a hagyományos megközelítés során túl sok finomhangolásra és összetett szabályokra lenne szükség.
Több csoportra oszthatók a tanulási módszerek: felügyelt tanulás, felügyelet nélküli tanulás, megerősítéses tanulás *(2.2. ábra)*.



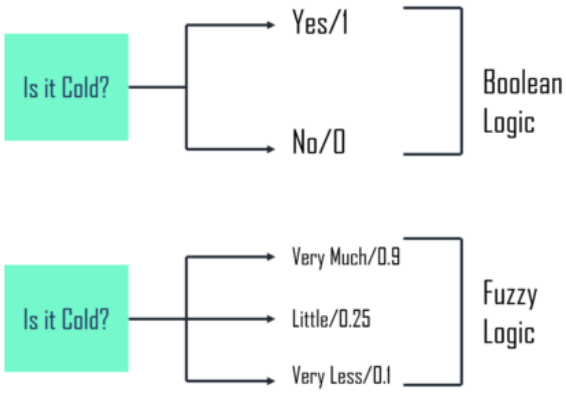
2.2. ábra: Gépi tanulási módszerek típusai (forrás: [11])

A felügyelt tanítás során az adathalmaz tartalmazza a kívánt megoldásokat, így mérhető a pontosság. A megoldandó feladatok a kimeneti érték típusa szerint tovább bontható csoportokra: regresszió, klasszifikáció. A regresszió esetében a kimenet számszerű. A klasszifikáció során a kimenet kategorikus, megmutatja, hogy az adott elem melyik osztályba sorolható a paraméterei alapján. Lehet bináris vagy multi klasszifikáció, ezt az osztályok száma határozza meg. A leggyakrabban használt algoritmusok: K-legközelebbi szomszéd, lineáris regresszió, döntési fa, véletlen erdő és neurális hálózatok.
Felügyelet nélküli tanításról beszélünk, amikor a tanítómintában címkézetlen adatok szerepelnek, vagyis nem tartalmazza a megoldásokat. Ilyenkor az adatok közötti összefüggések felismerése és csoportok / klaszterek létrehozása a cél. A félig felügyelt tanulás során részben címkézett adatok szerepelnek. A legtöbb esetben ilyenkor a tanulási algoritmus felügyelet nélküli és felügyelt tanítás kombinációja.
A megerősítéses tanulás megvizsgálja a környezetét, választ egy műveletet és eszerint jutalmat, illetve büntetést kap. Magától kell megtanulja az optimális stratégiát ez a jutalmak maximalizálásával történik.
Nehézséget okoz a gépi tanulás során, ha nincsen megfelelő mennyiségű adat, ha nem megfelelő a minőségük vagy, ha nem reprezentatív az adat. Az adatokat célszerű két részre bontani: tanító halmazra és teszt halmazra. Általában 80:20 arányban szokás szétosztani őket. [12]

2.6.1. Fuzzy logika

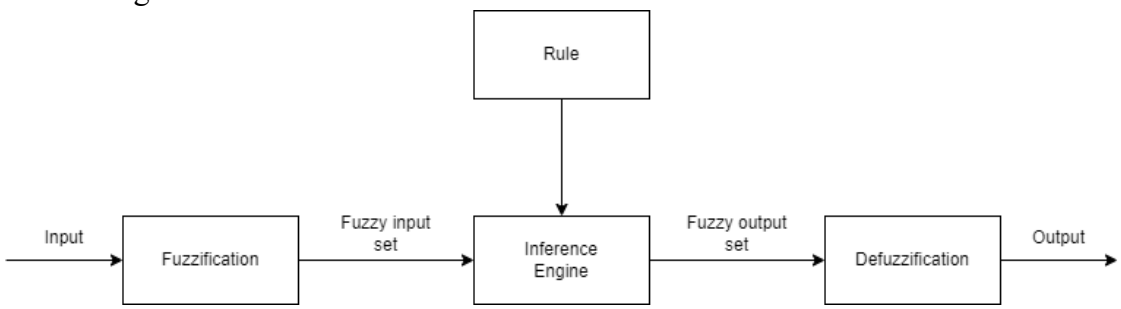
A Boole-algebrában igaz (1) vagy hamis (0) értékek szerepelnek, csupán két lehetőség közül lehet választani. A fuzzy logika ebben hoz újat, hogy 0 és 1 közötti értékek is

szerepelnek. Az ember is a döntésénél nem feltétlenül teljesen biztos lehet ez csak félig vagy kicsit is. A két logika közötti különbséget a *2.3. ábra* szemlélteti.



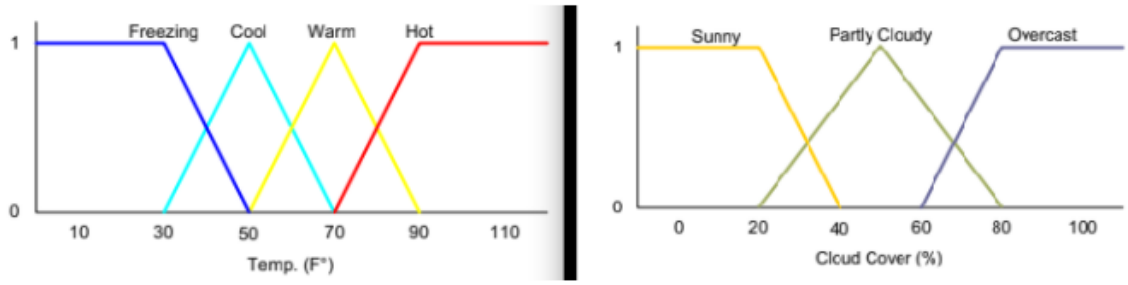
2.3. ábra: A boolean logika és a fuzzy logika összehasonlítása (forrás: [13])

A fuzzy logika architektúrájának *(2.4. ábra)* főbb elemei: Fuzzification, Inference Engine, Rule, Defuzzification. A Fuzzification komponens számszerűsíti az állításokat. Például a hőmérsékletnél definiálja a dermesztő hideget, hideget, meleget, forrót. Az Inference Engine felel, hogy a szabályok által milyen döntést hozzon a kapott adatok alapján. Például „ha meleg idő van, akkor hideg italt iszom”, „ha az időjárás hűvös, akkor forró italt iszom”. A Defuzzification egység a kapott fuzzy kollekciók alapján előállítja a kimenetet. Többféle módszert is lehet használni. A leggyakoribbak az unió, a metszet és a különbség.



2.4. ábra: Fuzzy logika architektúrája (forrás: saját ábra)

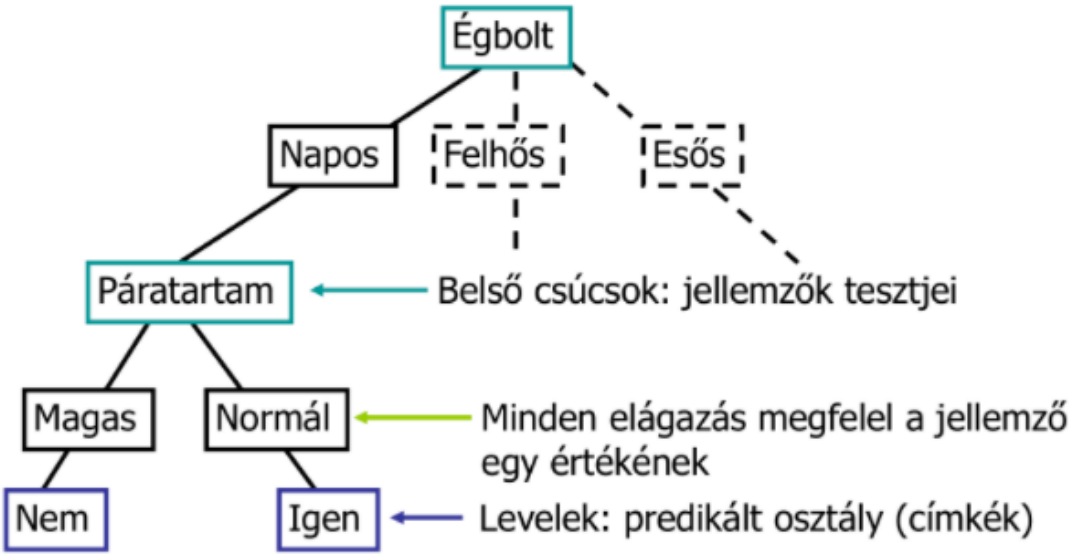
A tagsági függvény *(2.5. ábra)* a bemenő adatokhoz hozzárendel egy tagsági értéket, ami 0 és 1 közötti szám. A függvény folytonos, alakja szerint lehet háromszög, trapéz fűrészfog stb.



2.5. ábra: Példák tagsági függvényre (forrás: [14])

2.6.2. Döntési fa (Decision Tree)

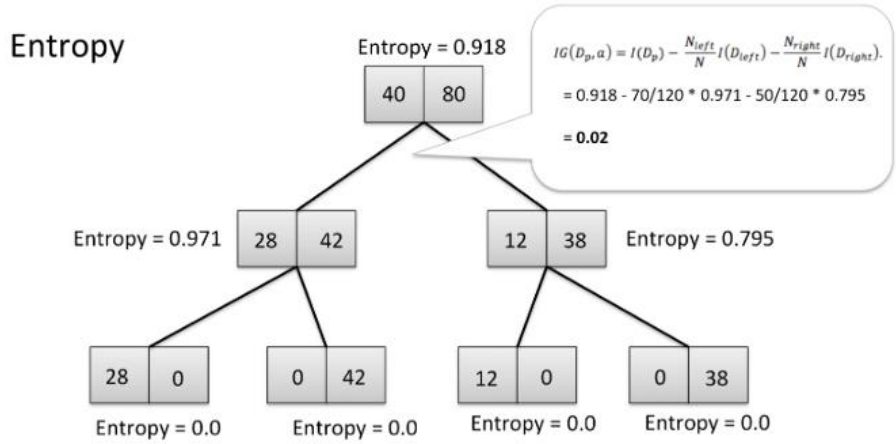
A döntési fa egy olyan tanuló algoritmus, ami a jellemzők közti kapcsolatok explicit gépi tanulásán alapul. A fa belső csúcsai egy-egy jellemzőt tartalmaznak. Egy csúcs gyerekei az adott jellemző lehetséges értékeit jelenti, ahány gyereke van annyi lehetséges értéke lehet. A levelek felelnek meg az osztályoknak *(2.6. ábra)*.



2.6. ábra: A döntési fa elemei (forrás: [15])

Nem minden jellemzőnek kötelező szerepelnie a fában, ha nem változtatja a kimenetet azon az ágon. A fában egy jellemző többször is szerepelhet, de egy gyökértől a levélig tartó úton csak egyszer jelenhet meg. A leveleknek nem kell ugyanazon a mélységen lenniük. A döntési fa logikai szabályokkal írható le. A gyökértől a levélig vezető úton a jellemzők értékei között ÉS kapcsolat van, míg a levelekig vezető utak között VAGY kapcsolat található. A döntési fák előnyei: diszkrét jellemzők közti explicit kapcsolatot tanul, a fa emberi szemmel is jól értelmezhető. A hátránya, ha sok jellemző van, akkor a közöttük lévő kapcsolatok tanulásához nagy méretű tanító minta szükséges. Továbbá csak a döntés megszületéséhez legszükségesebb, leghasznosabb jellemzőket építi be a fába, amik kevésbé hasznosak nem kerülnek bele a, így az információtartalmuk elvész. A döntési fákat akkor célszerű alkalmazni, ha viszonylag kevés (pár száz) diszkrét jellemzőnk van és feltételezzük, hogy köztük bonyolultabb kapcsolat létezik A döntési fák hátrányainak eltüntetésére van a véletlen erdő osztályozás (Random Forest Classification). Itt a modell nem egyetlen fa, hanem több kisebb döntési fából áll és azok jóslatából tevődik össze a végső predikció. Különböző jellemzőket tartalmaznak a kisebb fák, így különböző nézőpontokból tudnak értékelni. [15]
A döntési fa felépítése során az ágak feldarabolásra kerülnek. Erre a két legismertebb módszer az ID3 (Iterative Dichotomizer) és a CART (Classification and Regression Tree). A feldarabolás során különböző mérőszámokat alkalmaznak az ID3 az információszerzést és ehhez az entrópiára van szüksége. Az entrópia a rendezetlenség mértékegysége, jelen esetben a hiányos, szennyezett információ. A döntési fa ágainak darabolásával az algoritmus egyre alacsonyabb entrópia felé halad *(2.7. ábra)*, így

kitisztul, hogy melyik útvonal melyik osztályba vezet. Az ID3 célja a magas információszerzés és az alacsony entrópia, ezáltal egy jobb átláthatóságot biztosít.



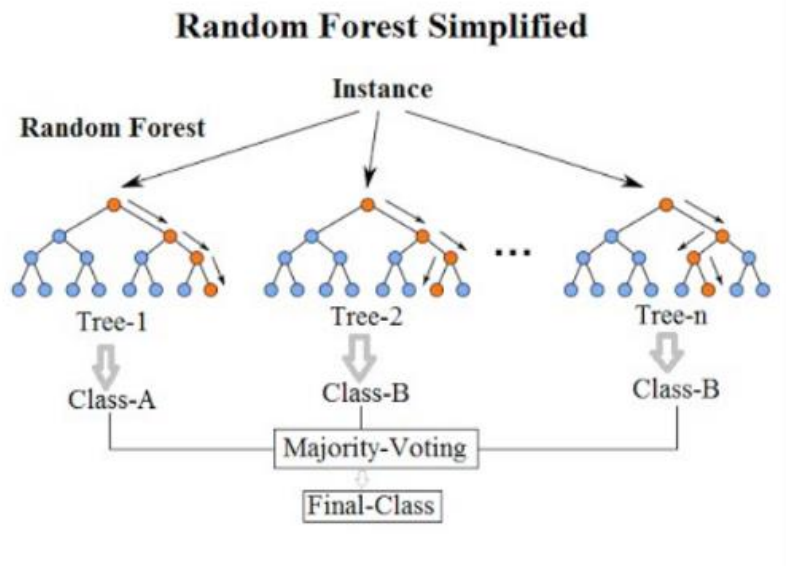
2.7. ábra: Az entrópia változása a darabolás során (forrás: [16])

A CART a Gini indexet használja, ami megmutatja, hogy egy véletlenül kiválasztott adat mekkora valószínűséggel sorolható be rossz osztályba. A célja, hogy minimalizálja ezt az értéket.

A döntési fák hajlamosak a túltanulásra, ami azt jelenti, hogy nehezen címkézi fel azt az adatot, amit korábban nem látott még. A tanító minta nagyon rögzült benne és nehezen tud róla váltani. [16]

2.6.3. Véletlen erdő (Random Forest)

A véletlen erdő osztályozásnál a modell nem egyetlen fából, hanem több kisebb döntési fából épül fel. Ezek a kisebb egységek a kapott bemenetet külön-külön értékelik ki és kapnak egy eredményt, utána egy többségi szavazó segítségével előáll a végső eredmény *(2.8. ábra)*.

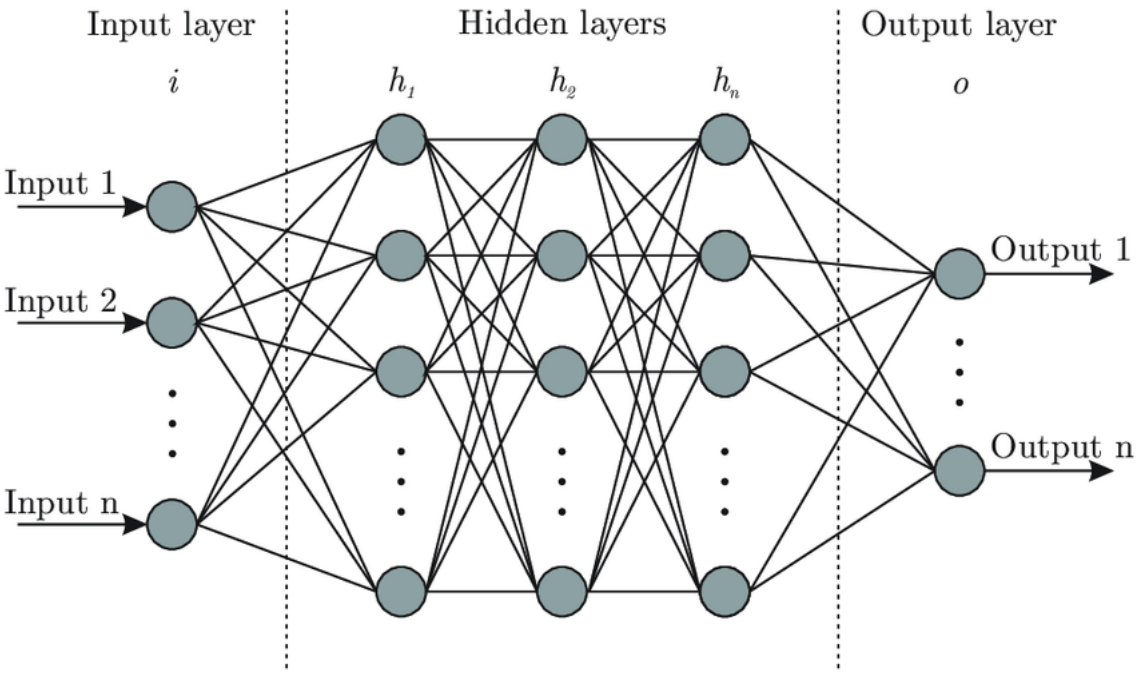


2.8. ábra: A véletlen erdő felépítése (forrás: [16])

A véletlen erdőt azért tervezték, hogy a zsákoló algoritmus segítségével tovább csökkentse a túltanulás lehetőségét. Az algoritmus az eredeti adathalmazból kiválaszt véletlenszerűen sorokat, ezek ismétlődhetnek is és előállít adatszerkezeteket belőlük. Ezekből épülnek majd fel a döntési fák, amik egymástól teljesen függetlenek lesznek. Mivel eltérő adatokból épülnek fel, így jobban tudnak reagálni a teljesen új, addig ismeretlen adatokra. [16]

2.7. Neurális hálózat

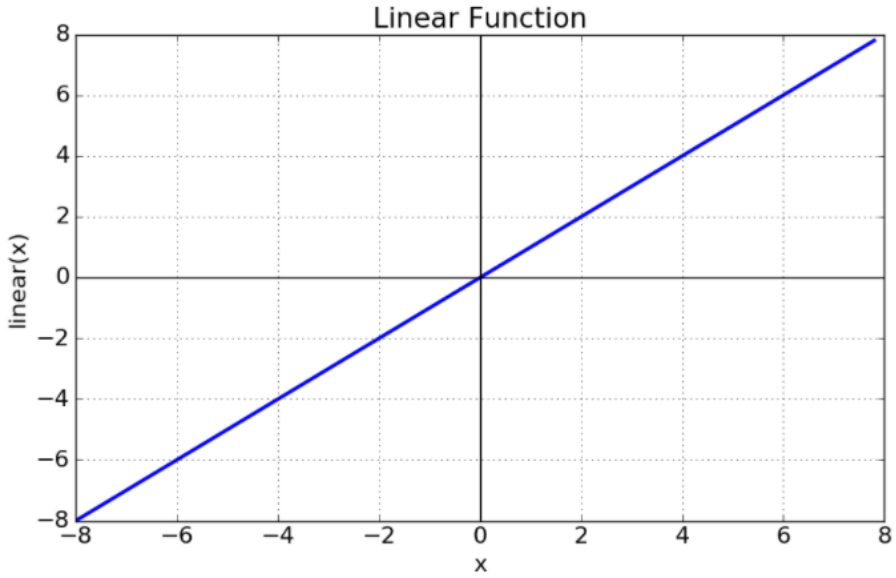
A neurális hálózat komplex adat-vezérelt problémák megoldására használható. Alapvetően három részre bonható fel *(2.9. ábra)*. Az első a bementi réteg itt az adatokat töltjük be a rendszernek. Ez a bemeneti paraméterek számától függ, képek esetén a kép pixel számával megegyező neuronra van szükségünk. A második a rejtett rétegek, ez a belső rész a bemeneti és a kimeneti rétegek között helyezkedik el, ahol a neuronok a súlyozott bemenetek halmazát veszik fel és az aktivációs függvény révén kimenetet produkálnak. Az utolsó a kimeneti réteg, ami a rendszer válasza a betöltött bemenetre.



2.9. ábra: Neurális hálózat (forrás: [13])

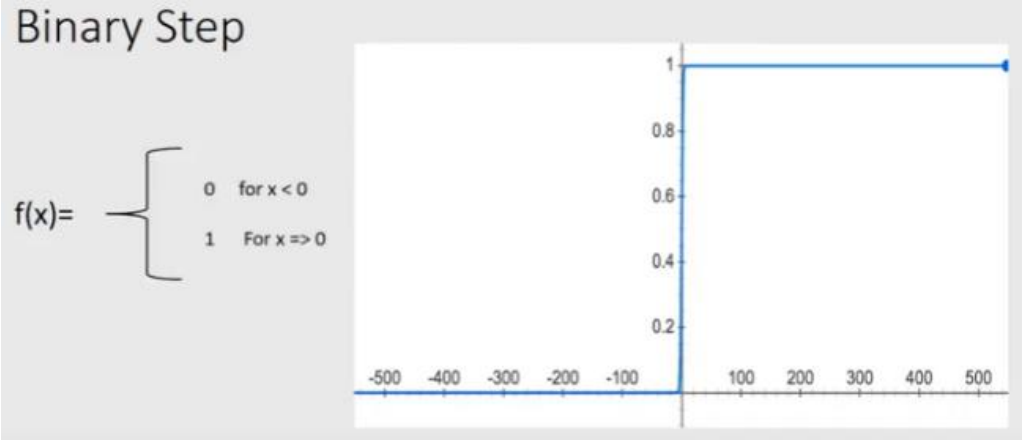
Amikor az adat bekerül a rejtett rétegbe, akkor egy véletlenszerű kezdő értéket kapcsolunk mindegyikhez. Ezután megszorozzuk a hozzá tartozó súllyal majd a végeredményhez hozzárendeljük a bias értéket. Utolsóként az aktivációs függvény hajtódik végre és tovább kerül a következő rétegbe. Ez mindaddig ismétlődik amíg végig nem érünk a rejtett rétegeken és elérünk a kimeneti réteghez. Ahol megkapjuk a végeredményt.

A backpropagation módszerrel visszafelé haladunk a hálózatban és összehasonlítjuk a tényleges kimenetet az elvárt kimenettel, majd a súlyokat úgy változtatja, hogy a hiba minimalizálódjon. [18]
A neuronnál található aktivációs függvénnyel próbálja a rendszer a kapott értéket adott határon belül tartani, ez többnyire -1 és 1 közötti érték szokott lenni, de aktivációs függvényenként eltérő lehet. Vannak lineáris és nem lineáris függvények. A lineáris függvények *(2.10. ábra)* nem segítenek feloldani a hálózat komplexitását, ezért nem szokás használni. [19]



2.10. ábra: Lineáris aktivációs függvény (forrás: [19])

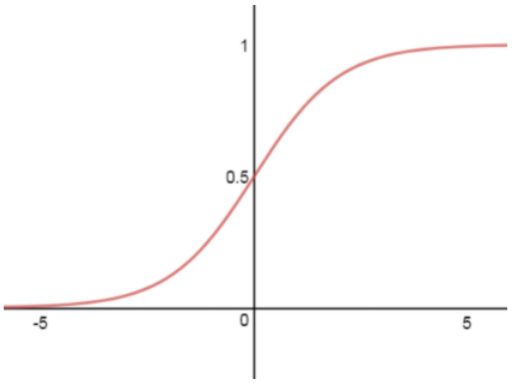
A nem lineáris függvényeknél a legkezdetlegesebb a bináris, itt csupán két értéket vehet fel a függvény 0 és 1. *(2.11. ábra)*



2.11. ábra: Bináris aktivációs függvény (forrás: [20])

A Sigmoid *(2.12. ábra)* aktivációs függvény, a bemenő értékeket 0 és 1 közötti tartományba képezi le. Ez a függvény két fő hátránnyal rendelkezik. Az első, hogy a rendkívül kis tartomány miatt a nagy pozitív és negatív számoknál nem tud jelentős változásokat végrehajtani, így megnehezítené a tanulás folyamatát. A másik hátrány, hogy a függvény nem nulla központú, ami szintén nehezíti a gyors tanulást.

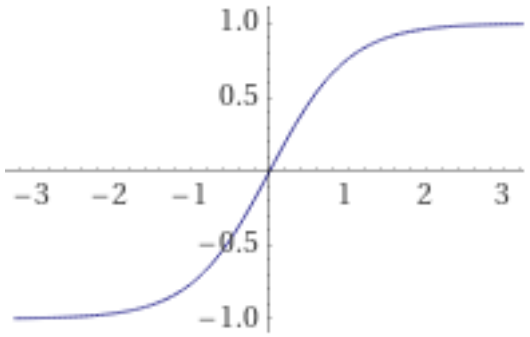
$$sigmoid(x) = \frac{e^x}{1 + e^x} \tag{2.1.}$$



2.12. ábra: Sigmoid aktivációs függvény (forrás: [21])

A Tanh aktivációs függvény *(2.13. ábra)*, azaz a tangens hyperbolicus annyiban különbözik az elődjétől, hogy már nulla központú. A [-1,1] tartományba képezi le a bemenetet. Ezért előnyben szokták részesíteni a Sigmoidal szemben.

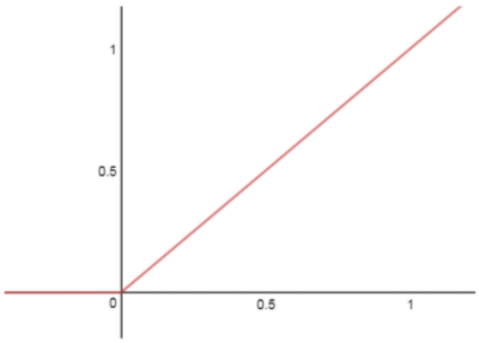
$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2.}$$



2.13. ábra: Tanh aktivációs függvény (forrás: saját ábra)

A ReLU (Rectified Linear Unit) aktivációs függvénynek *(2.14. ábra)* a bemenete, ha nagyobb, mint nulla akkor egy lineáris leképezést kapunk, viszont, ha kisebb akkor minden esetben nullát. Míg más számításigényes függvények (Tanh, Sigmoid) sok műveletet foglalnak magukba, addig a ReLU könnyen megvalósítható. Hátránya, hogyha a bemenet nullánál kisebb akkor mindig nullát ad vissza ez egy nagy tartomány, ami megnehezíti a tanulást.

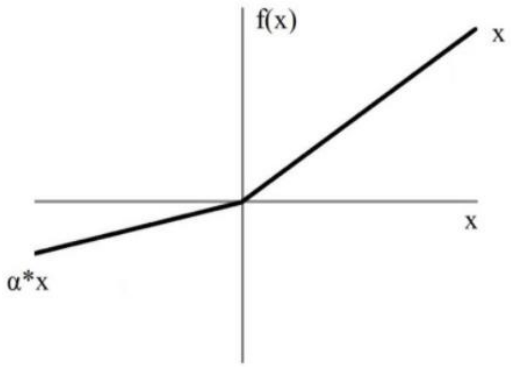
$$R(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & otherwise \end{cases} \tag{2.3.}$$



2.14. ábra: ReLU aktivációs függvény (forrás: [21])

A Leaky ReLU *(2.15. ábra)* az előző tovább fejlesztése, mivel az sokszor „megölte” a neuront és utána már soha nem aktiválódott. Abban az esetben, hogyha a bemenő paraméter kisebb, mint nulla akkor megszorozzuk egy α számmal és lineárisan leképezzük. Így elkerülhetjük, hogy „meghaljon” a neuron.

$$R(x) = \begin{cases} x & if\ x \geq 0 \\ \alpha x & otherwise \end{cases} \tag{2.4.}$$



2.15. ábra: Leaky ReLU aktivációs függvény (forrás: [21])

A Softmax aktivációs függvény *(2.5. képlet)* egy kicsit különbözik a többitől. Csak az utolsó rétegben használjuk, ha klasszifikációs feladat van és meg akarjuk jósolni a valószínűségeket, hogy melyik eredményre milyen esély van. Az eredményeket 0 és 1 közötti számmal reprezentálja. [21]

$$y_i = \frac{e^{z_i}}{\sum_{i=0}^{m} e^{z_i}} \tag{2.5.}$$

A *loss* függvény a neurális hálózatok egyik alapvető eleme, ami a hálózat hibájára mutat rá, hogy menyire ad pontos választ a rendszer. A *loss* segít kiszámítani a gradienst, ami a súlyok változtatásáért felel. Többféle függvény is létezik a *loss* kiszámolására ilyenek például a Cross Entropy és a Mean Squared Error. A gradiens azt mutatja meg, hogy a bemenet kis változtatásakor a kimenet milyen mértékben változik meg. Ha a gradiens értéke nagy, akkor a rendszer gyorsan tud tanulni, mivel kis eltérésű bemenetek hatására is megismeri a kimenetek változását. A kis vagy nulla értékű gradiensnél a tanulás lelassul, vagy megáll. Ez a folyamat elengedhetetlen a neurális hálózat tanításánál. [18]

2.8. TensorFlow

A TensorFlow egy nyílt forráskódú könyvtár, amit a Google fejlesztett ki, hogy támogassa a mély tanulást. Eredetileg nagy számítás igényű feladatokhoz alkották meg. Azonban ez nagyon hasznos a mély tanulást használó rendszerek fejlesztésnél is. Az adatokat multi dimenziós tömbökben tárolja, így nagy mennyiségű adatot könnyebb kezelni. A TensorFlow C++ és Python API-kal rendelkezik. A magas szintű API-nak köszönhetően nincsen szükség komplex kód írására, ha neurális hálózatot vagy egy neuront akarunk konfigurálni. A könyvtár ezeket a feladatokat mind elvégzi. A deep learning alkalmazások nagyon összetettek, a tanulás alatt sok számítást, mátrix szorzást és különböző matematikai műveleteket végeznek. Ezért támogatják, hogy a kódot ne csak CPU-n hanem GPU-n is lehessen futtatni a gyorsabb eredmény érdekében. [22]

2.9. Keras

A Keras egy deep learning API Python nyelven, ami a TensorFlow fölött fut. A témában kezdők és a kutatók számára készítették, mert gyorsan és egyszerűen lehet vele fejleszteni. A kutatás szempontjából fontos, hogy az ötlettől a megvalósításig minél gyorsabban eljusson. [23]

2.10. Technológiák összegzése

A rendszer fejlesztése során a fentebb említett technológiák használhatóak fel. A különböző CSS könyvtárak megkönnyítik a felhasználói felületek elkészítését. Ezáltal nem kell külön tiszta CSS fájlokat létrehozni. Hanem használhatóak az előre elkészített osztályok, amelyeket a különböző HTML elemeknél hívhatóak meg. A React és React Native hasonló felépítésének köszönhetően az egyik ismeretével könnyebben fejleszthető a másik. A mögöttes kód, ami az adatszervezési logikát valósítja meg mind a két technológiánál megírható JavaScript/TypeScript nyelven. A többi mobil technológiák eltérő nyelveket használnak, amik a webes környezetnél nem használhatóak. Az ASP.NET segítségével készülhet el a Web API, ami egy jól használható keretrendszer és könnyen készíthetünk el vele különböző web szolgáltatásokat. A modern elvárásoknak megfelelően automatikusan szerializálja a JSON objektumokat, de természetesen egyedileg beállíthatóak, ha az szükséges a különleges követelmények miatt. Támogatja a JWT (JSON Web Tokens) alapú hitelesítést, engedélyezést. A gépi tanulási algoritmusok, akkor hasznosak, ha nem áll rendelkezésre megfelelő mennyiségű adat, amiből be lehetne tanítani egy modellt. Ezek közül is leginkább a döntési fákból álló véletlen erdő tudja

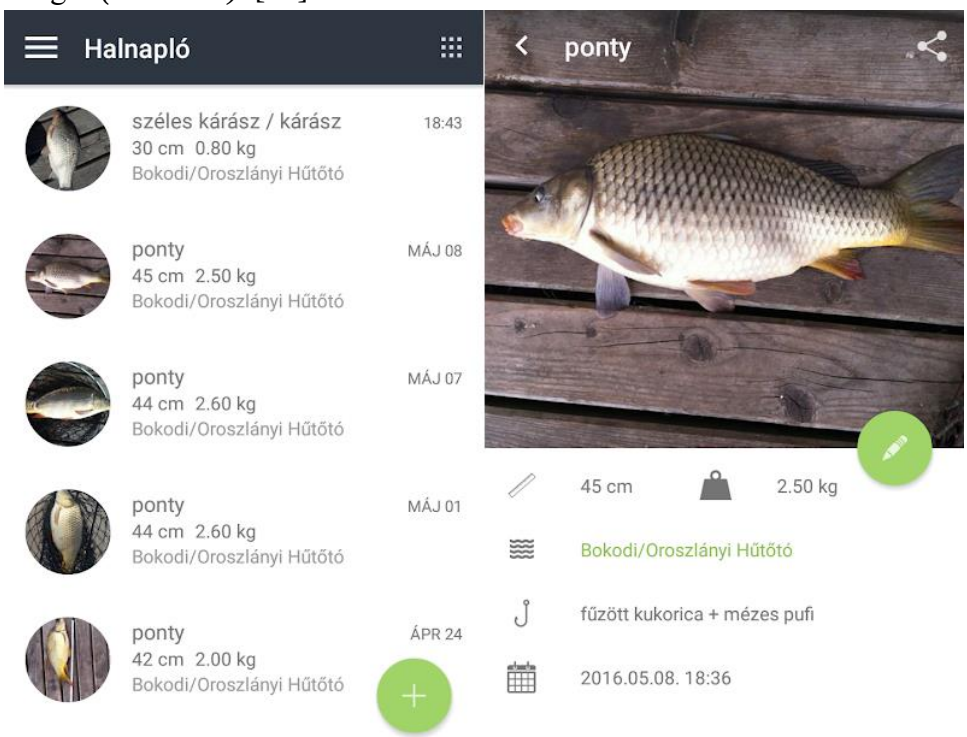
adni a legpontosabb eredményt. A neruális hálózatok nagy mennyiségű adatok befogadására alkalmasak, amikben képes megtalálni egy mintát. Így a betanítás után jó predikciókat képes adni.

3. Hasonló rendszerek

Ebben a fejezetben a készítendő alkalmazáshoz hasonló rendszereket gyűjtöttem össze.

3.1. Halnapló

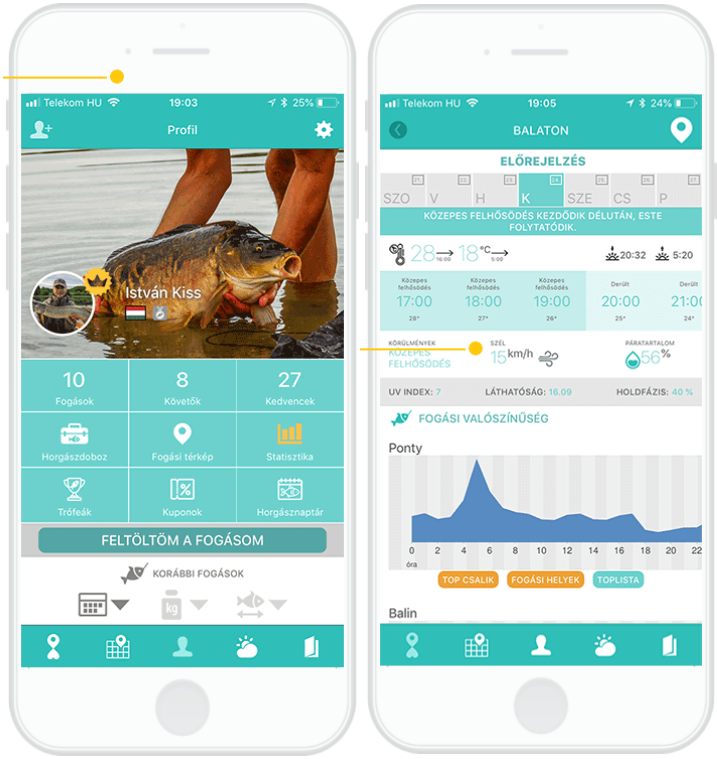
Online fogási napló és közösségi portál horgászoknak. Fogásait megőrizheti és megoszthatja másokkal. Főbb funkciók: fogások rögzítése fotókkal, automatikus horgászvíz meghatározás GPS pozíció alapján, horgászvíz keresése, a fogások tetszés szerint megoszthatók. A képernyőfotókon látszódnak a rögzített fogások, illetve egy konkrét fogás *(3.1. ábra)*. [24]



3.1. ábra: Halnapló képernyőfotók (forrás: [24])

3.2. Fishinda

Android és iOS operációs rendszerrel rendelkező mobiltelefonoknál is elérhető az alkalmazás. Egy közösségi horgász applikáció, ahova a felhasználó feltöltheti a képekkel illusztrált fogásait és később visszanézheti mi az, ami jól működött *(3.2. ábra)*. A hírfolyamon láthatja hol és milyen csalival sikerült fogni a legszebb halakat. Kapcsolatba léphet más horgásztársakkal és tapasztalatot cserélhet velük. A sok összegyűjtött információ alapján kikövetkeztetheti a fogási esélyeket vizekre és halakra lebontva. [25]



3.2. ábra: Fishinda Saját Profil és Fogás Előrejelzés (forrás: [25])

3.3. Ingatlan értékbecslő rendszer neurális hálózattal

Ebben az alkalmazásban szükségünk van nagy mennyiségű adatra, ez minden tanításnál létfontosságú kérdés. Az adathalmaz (forrás: [27]) 13 bemeneti információból és 1 elvárt eredményből áll. A fejlesztés során a TensorFlow szoftverkönyvtára volt használva, ami ingyenes és nyílt forráskódú. Neurális hálózatok létrehozásához használható. Első lépésként az tanító adatok kerülnek beolvasásra. Az adatok különböző tartományokban mozognak, ami nem jó a gradient descent[1] eljárásnak. Ezért célszerű normalizálni őket 0 és 1 közötti értékekre, így mindegyik értéket ugyan olyan fontossággal fogva kezelni. Amennyiben, hogyha az adat beolvasás kész el lehet kezdeni felépíteni a neurális hálót. Egyesével hozzáadjuk a rétegeket, meg kell adni a neuronok számát, illetve az aktivációs függvényt ez jelen esetben a ReLU. Az utolsó réteg a kimenet itt egy neuron használunk és nem adunk meg aktivációs függvényt. A végén hozzáadjuk a modellhez a *loss* függvényt és az optimalizálást. Majd külön kezelünk egy X értéket, ami a jósolt válasz

[1] Optimalizáló algoritmus neurális hálózatoknál, ami a függvény lokális minimumát keresi.

és egy Y értéket, ami az elvárt válasz. A hálózat készen áll a tanításra. A TensorFlow segítségével beállíthatjuk, hogy hányszor fusson le a tanítás és közben keverje össze az adathalmaz sorait, hogy ne mindig ugyan abban a sorrendben olvassa be az adatokat. A végén pedig megjelenítjük a kapott adatokat így látjuk mennyire sikerültek jól a becslések. [26]

3.4. Need for mobile application in fishing

Indiában 7 millió ember él a part mentén, akiknek a megélhetése a horgászattól függ. A horgászoknak szükségük van az időjárási körülményekre és a vízállás is, így kideríthetők a potenciális horgászhelyek. GPS navigáció segítségével felkereshetők majd elmenthetők a kiszemelt helyek.
Jelenleg az időjárás előrejelzést rádión hallgatják, az aktuális és a pár napot átfogó adatokat közlik. A mobil applikációnál részletesebb és bármikor visszanézhető adatokat lehet feltüntetni, így jobban lehet előre tervezni a horgászatot. Térképen megjeleníthető a potenciális horgászhelyek listája, ahova navigáció segítségével könnyen oda lehet jutni. A hal rajok mozgása is berajzolható az adott időjárási viszonyok tekintetében. A horgászok láthatják egymás hajóit, így tudják, melyik területeket fedik le.
A mobil alkalmazás lenne az elsőszámú erőforrása a horgászoknak. Az amatőr és a profi horgász is ugyanúgy tudja használni az alkalmazást, viszont a korábban megszerzett tapasztalat nélkülözhetetlen a jobb fogási eredmények elérésében. Az alkalmazás egy értékes eszközzé válhat a horgászok kezében, nézhetnek szolunáris, időjárás előrejelzéseket, radar térképeket, feljegyezhetik fogásaikat, megjelölhetnek potenciális horgászhelyeket esetleg segélykéréseket adhatnak le, ha bajba kerülnek a vízen. [28]

3.5. Picking a vacation destination – Decision Tree

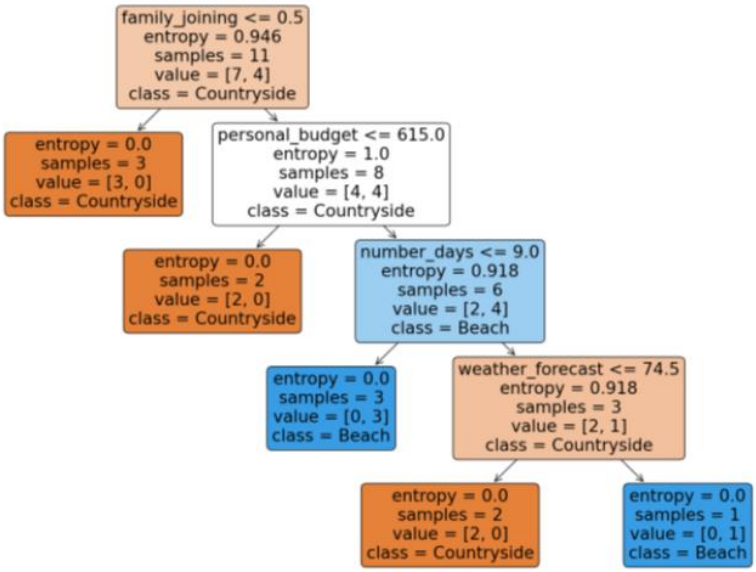
Egy nyaralás tervező alkalmazás, ami a bemenő adatok alapján megmondja, hogy tengerpartra vagy vidékre menjen a család nyaralni. A bemenő adatok, hogy hány napos legyen a nyaralás, akar-e csatlakozni a nagy család többi tagja, mennyibe fog kerülni személyenként, az időjárás előrejelzés milyen hőmérsékletet jósol és szeretnének-e új helyeket felfedezni. Az adatok a korábbi nyaralások alapján készültek *(3.3. ábra)*.

Number of days	Family joining	Personal budget ($)	Weather forecast (Avg Temp in F)	Explore new places	Vacation
10	Yes	500	75	Yes	Countryside
7	No	1000	78	Yes	Countryside
15	Yes	750	80	No	Beach
8	Yes	550	67	Yes	Countryside
10	No	800	73	No	Beach

3.3. ábra: Adathalmaz részlet a feladathoz (forrás: [29])

A ScikitLearn Decision Tree algoritmusa nem tudja kezelni az összes adat típust csak a numerikusakat. Ezért egy előfeldolgozás során át kell alakítani a szöveges adatokat számmá. A rendelkezésre álló adatszerkezetet felbontja két részre 80:20 arányban, az első rész a tanító minta lesz a második pedig a teszteléshez használható adathalmaz. A ScikitLearn alapértelmezetten a Gini indexet használja veszteség függvényként, de

használhatjuk az entrópiát is. A betanított modellen *(3.4. ábra)* jól látszik, hogy hiányzik az *explore_new_places* fa elem.

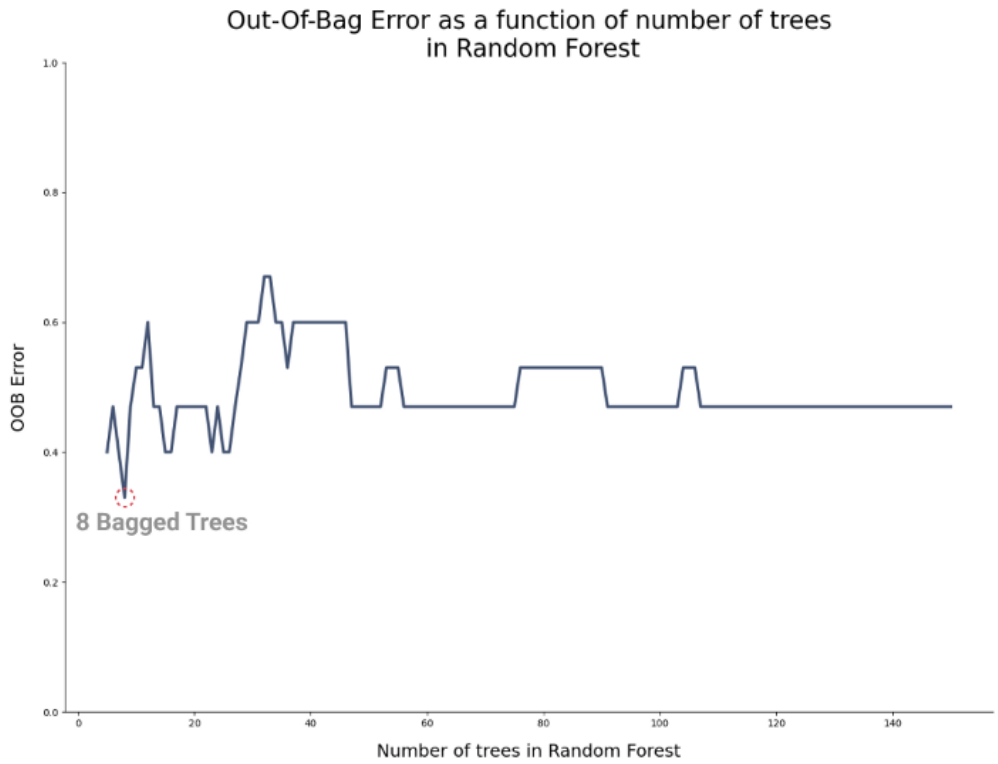


3.4. ábra: A betanított model (forrás: [29])

A DecisionTreeClassifier modellnek van egy *feature_importances* tulajdonsága, ami azt mondja meg, hogy az egyes elemek mennyire járulnak hozzá a modell pontosságához. Ezt egy tömbben adja vissza, amiben az *explore_new_places* értéke 0. Ezért ebben az estben nem szerepel a fában. Végül a kiértékelés során a tanító adathalmazon 100%-os pontosság, viszont a teszt adatokon csak 67%-os pontosság volt elérhető. A modell túltanult, túlságosan betanulta a tanító mintát, ezért nem tudott általánosítani és rossz választ adott azokra az adatokra, amiket korábban nem látott. A pontosság növelésére az egyik megoldás a tanító minta növelése lehet.
A döntési fák hajlamosak a túltanulásra. Amikor nagyon magas a fa és sokszor fel vannak darabolva a jellemzők, akkor valószínűleg túltanulta a tanító mintát. A túl komplex fát nehéz olvasni és értelmezni is. A másik oldalról, ha nagyon kicsi a fa, akkor az alultanulás kockázata jelenik meg. Egy fa egyedül tipikusan nem ad kellően jó predikciót, de több fa esetében ez javulhat. A véletlen erdő algoritmus több fát kombinál, hogy jobb teljesítményt biztosítson, mint egy sima fa. [29]

3.6. Picking a vacation destination – Random Forest

A fentebb található nyaralás tervező feladat megoldása véletlen erdővel. Az előző megoldásban 0,67 volt az MSE (Mean Squared Error) értéke, ezt egyetlen egy fa érte el. A ScikitLearn könyvtár segítségével lett tanítva a modell, alapértelmezetten 100 fát hozott létre. Az Out-Of-Bag Error értéke 0,47 lett, nagyjából 30%-kal sikerült csökkenteni a hibákat. Az OOB értéke a fák felépítése során a zsákoló algoritmus miatt kimaradt adatok segítségével készült predikciók hibájának az átlaga. A fák optimális számának megtalálásához legenerálásra kerültek erdők 5-150 darab fával. A *3.5. ábráról* könnyen leolvasható, hogy az OOB Error a 8 fából álló erdőnél volt a legkisebb 0,33. 50 darab fától alig változik az OOB Error értéke, innentől egy síkság található. A hiba mértékét jelentősen sikerült csökkenteni a véletlen erdő megvalósításával. [30]



3.5. ábra: Out-Of-Bag Error szemléltetése fa darabszámonként (forrás: [30])

3.7. Hasonló rendszerek összegzés

A Halnapló és a Fishinda egy egyszerűbb CRUD alkalmazás, néhány extra funkcióval. Ilyen a horgászvíz meghatározás GPS koordináták alapján. A Fishinda lépett a social media irányába és egy közösségi alkalmazást készített. Továbbá ügyes statisztikákat készít a horgászatok alapján, mint például a nap folyamán melyik időszakok szoktak eredményesek lenni. A horgászvíz meghatározást és az ötletes statisztikákat később én is felhasználhatom az alkalmazásomban.

Az ingatlan értékbecslő rendszernél a neurális hálózat, akkor működik jól, ha sok adat áll rendelkezésünkre. A nagy méretű tanító minta segítségével eredményesen lehet tanítani a hálózatot és jó predikciókat fog később szolgáltatni. Jelen esetben az alkalmazás indulásakor nem áll rendelkezésünkre nagy mennyiségű adat, ami alapján később tudnánk ajánlást adni. Az alkalmazás hosszú távú használata során gyűjthető össze a kellő mennyiségű információ a tanítómintához, ezután tudna eredményes működni ez a módszer.

A nyaralás tervező alkalmazásoknál látszik a döntési fa gyengepontja, hogy kicsi tanítóminta esetén hajlamos a túltanulásra. Ennek a problémának a kezelésére növelni kell a tanítóminta méretét vagy több fát kell használni és áttérni a véletlen erdő algoritmusra. A jelen helyzetben a csali ajánláshoz a saját tapasztalataim, illetve az általános vélekedések alapján vannak adataim. Amennyiben ezek nem elegendőek a döntési fa optimális működéséhez a továbbfejlesztett megoldást a véletlen erdő algoritmust alkalmaznám.

4. Tervezés

Ebben a fejezetben a rendszer megtervezéséről, funkcióiról lesz szó és hogy milyen komponensekből fog állni.

4.1. Funkció lista

Az alkalmazás során szükség van a felhasználók és a horgászatok, fogások kezelésére. A kapott adatok alapján statisztikák készítésére és csalik ajánlására. Az alkalmazás használatához bejelentkezés szükséges, így a felhasználó kezelés is elengedhetetlen.

4.1.1. Felhasználó kezelés

Az alkalmazásban minden felhasználónak saját profilja van, amivel létrehozhat horgászatokat és fogásokat, majd később megtekintheti. Ezeket más felhasználók nem láthatják. Ehhez regisztrációra, majd bejelentkezésre van szükség. A bejelentkezéshez egy e-mail cím és jelszó páros szükséges, ekkor a szerver generál egy egyedi tokent amivel a szerver azonosítja a belépni kívánni akaró felhasználót. Minden felhasználó azonos jogkörrel fog rendelkezni. Van lehetőség új jelszó igénylésére, amennyiben nem emlékszik a régire. Ezt a regisztráció során megadott e-mail címre küldi ki a rendszer.

4.1.2. Horgászatok felvitele / karbantartása

A felhasználó létrehozhat horgászatokat az időpont és a helyszín megadásával. A horgászhelyet egy legördülő listából lehet kiválasztani. Hozzáfűzhet megjegyzéseket, ha valamilyen információt szeretne még eltárolni a horgászattal kapcsolatban. Később a felhasználó ezeket a horgászatokat módosíthatja vagy törölheti is.

4.1.3. Fogások felvitele / karbantartása

A horgász az adott horgászatokhoz felviheti a fogásait. Ehhez meg kell adnia a kifogott hal faját, méretét (súly vagy hossz) és hogy milyen csalival sikerült horogra csábítani. Később itt is van lehetőség módosítani vagy törölni a fogásokat.

4.1.4. Statisztikák

Az alkalmazásban meg lehet nézni a korábbi horgászatok és fogások alapján készített statisztikákat. Többféle statisztika közül lehet választani. Ezeket a statisztikákat globálisan is meg lehet nézni, ahol az összes horgászat szerepel a statisztikákban, nincsenek külön lebontva. Továbbá külön horgászatonként is elérhetőek ezek a statisztikák.

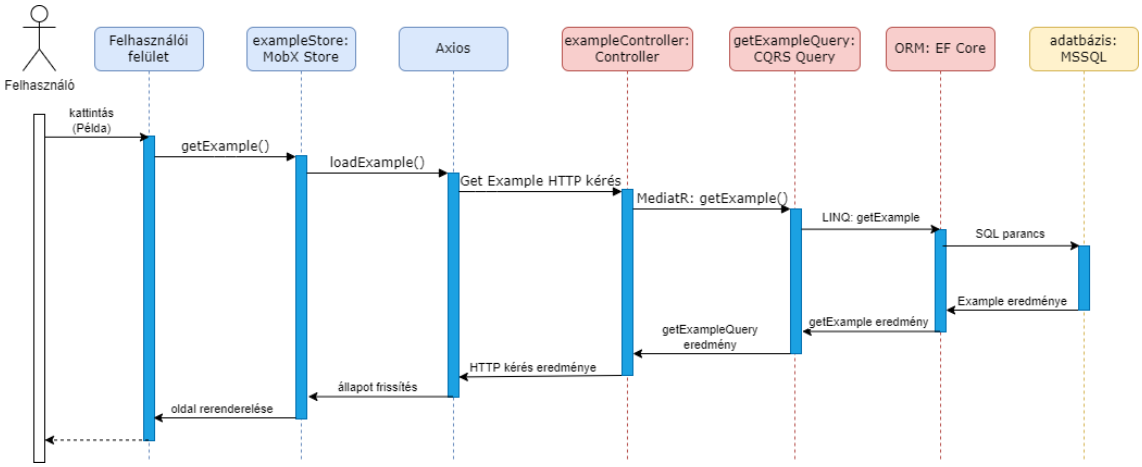
- legnagyobb kifogott hal
- legeredményesebb csalik (maximum 3 szerepelhet a toplistában)
- kifogott halak darabszáma
- kifogott halak összsúlya
- leggyakrabban kifogott halfaj
- horgászatok száma havi bontásban

4.1.5. Csali ajánlás

Az aktuális időjárási körülmények alapján a rendszer egy mesterséges intelligencián alapuló modell segítségével ad egy csali ajánlást, ami a kezdő horgászok számára segítség lehet.

4.2. Általános munkafolyamat egy-egy felhasználói interakció során

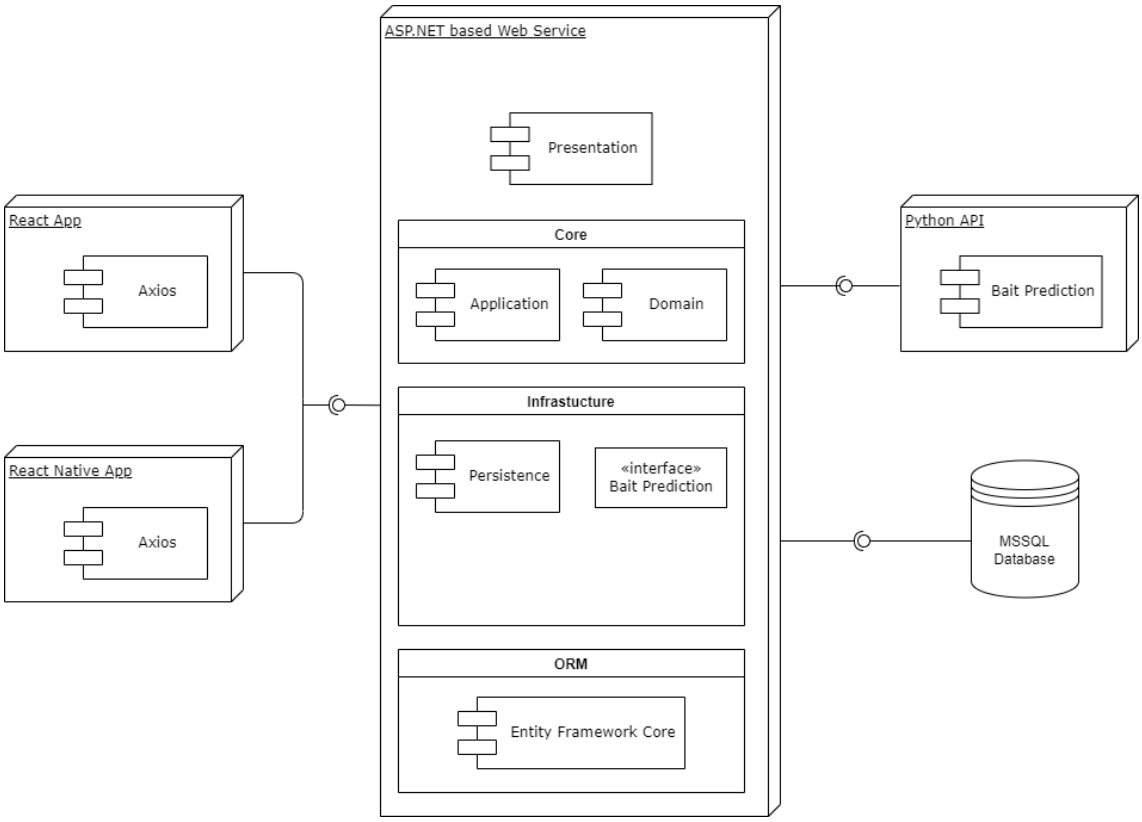
Az alábbi szekvencia diagram *(4.1. ábra)* mutatja be, hogy egy általános felhasználói interakció során hogyan jeleníti meg a kért adatot az alkalmazás. A „Példa” funkció segítségével szemléltetem a kommunikációt a front-end, back-end és végül az adatbázis között. A felhasználó rákattint a „Példa” gombra, amit a felhasználói felület továbbít az állapot menedzsernek, a Redux Storenak. Utána az Axios elindítja a HTTP kérést a back-end felé. A kérés beérkezik a back-enden lévő Controllerbe, amely létrehoz egy CQRS (Command Query Responsibility Segregation) query-t. Ez a query fogja megkérni a technológiához tartozó ORM-től LINQ segítségével az információt, majd az eredményt továbbítja a Controller felé. A Controller az eredményt JSON formátumba szerializálja és visszaküldi a kliensnek. A kliensen az Axios deszerializálja a JSON objektumot, majd frissíti az állapot menedzsert. A store állapotára feliratkozott felhasználói felület elemei újra renderelődnek, így megjelenik a kért adat a felhasználói felületen.



4.1. ábra: A "Példa" szekvencia diagramja (forrás: saját ábra)

4.3. A rendszer komponensei

A 4.2. ábrán megjelenő diagram mutatja az általam elkészítendő projekt elemeit. Két különböző platformra épülő front-end-et tud majd használni a felhasználó. A React alapú alkalmazást webes felületen és az ennek az alapjaira épülő React Native alkalmazást pedig mobilos környezetben. A rendszer fő része a back-end, ahol az üzleti logika van. Az Infrastructure rétegben lesz az a modul, ami a csali ajánláshoz szükséges kommunikációt biztosítja a Python API-val.



4.2. ábra: A rendszer komponens diagramja (forrás: saját ábra)

5. Prototípus

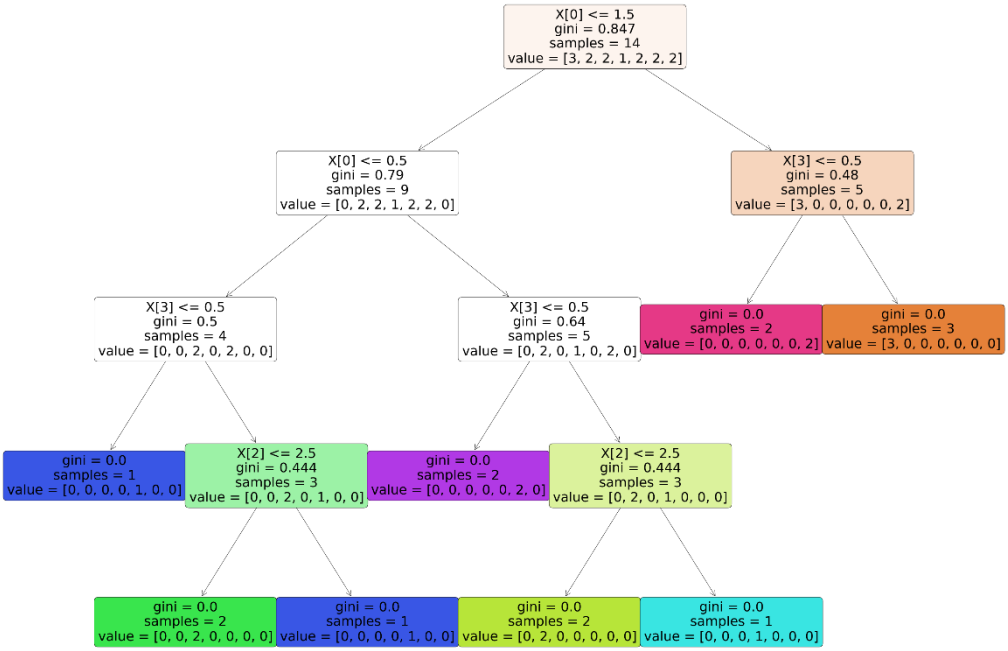
A fejlesztést a csali ajánló rendszer elkészítésével kezdtem. Ez egy különálló Python API, amihez, ha beérkezik a kérés az időjárási adatokkal, akkor azok alapján visszaküldi a csali ajánlást.

Először elkészítettem a saját tapasztalataim, illetve a horgászok közötti általános vélekedések alapján a tanító mintát egy csv fájlban. A benne szereplő időjárás adatok intervallumokat fednek le *(5.1. ábra)*. Később, amikor egy időjárás API szolgáltatja majd az adatokat, azokat egy előfeldolgozás során konvertálom a tartományokba és azokkal számolok tovább.

	Water temperature	Air pressure	Cloudy	Wind speed	Bait
0	cold	low	clear	no	SERIE WALTER WAFTER 6-8MM STRAWBERRY
1	cold	medium	partly cloudy	weak	SERIE WALTER WAFTER 6-8MM STRAWBERRY
2	cold	heigh	overcast	weak	PROMIX POP UP PELLET 8MM CHOCOLATE
3	cold	medium	overcast	strong	PROMIX POP UP PELLET 8MM CHOCOLATE
4	medium	heigh	partly cloudy	no	SERIE WALTER WAFTER 8-10MM ORANGE

5.1. ábra: Tanítóminta a modellhez (forrás: saját ábra)

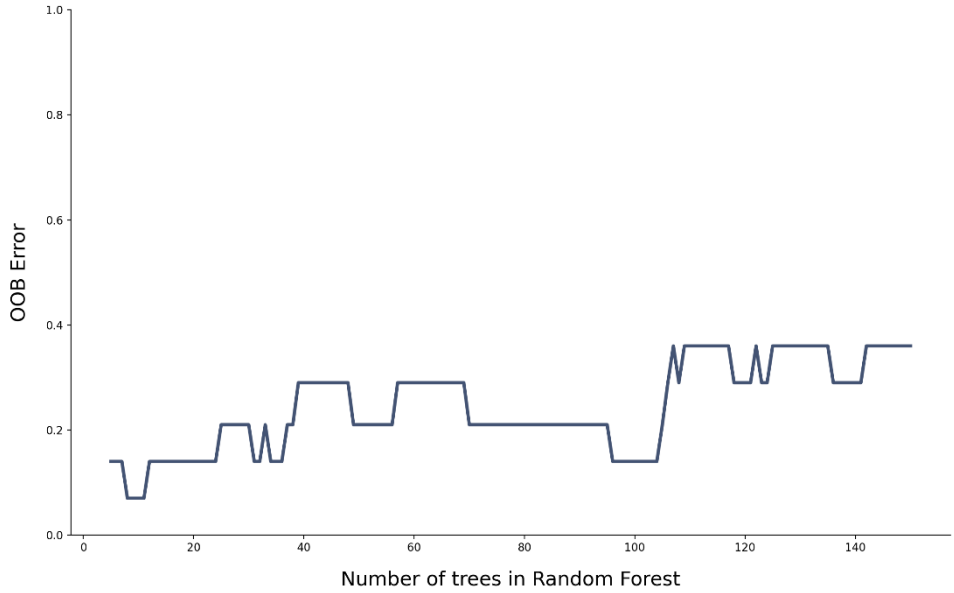
Az ajánlás elkészítéséhez először egy döntési fát készítettem. Az csv fájlt egy dataframe objektumba olvasom be. Ahol a szöveges adatokat egy LabelEncoder segítségével átalakítom egész számmá, amivel később lehet dolgozni. A döntési fát az alapértelmezett ScikitLearn beállításokkal hoztam létre és adtam át neki a tanító mintát. A fa elkészítése során nem volt megadva maximális mélység, illetve a feldaraboláshoz a Gini indexet használta *(5.2. ábra)*.



5.2. ábra: Az elkészült döntési fa (forrás: saját ábra)

A tanítás után a predikciók során látszik, ha a tanítómintában megegyező adat érkezik be akkor jó ajánlást ad. Viszont amikor az egyik paraméter már eltér, akkor nagyjából 60%-os pontossággal ad jó választ.

A pontosság javítása és a túltanulás eltüntetése érdekében kipróbáltam a véletlen erdő megoldással is. Az eleje a beolvasás és az előfeldolgozás teljesen megegyezik az előző megoldással. Az optimális mennyiségű fa megtalálásához először létrehoztam 150 darab erdőt 5-150 darab fával és megnéztem mindegyiknek az OOB hiba értékét *(5.3. ábra)*. A diagramon jól látszik a minimum helye, ez a síkság a 8-11 darab fánál található és az értkéke 0,07.



5.3. ábra: Az erdők OOB hibájának értéke (forrás: saját ábra)

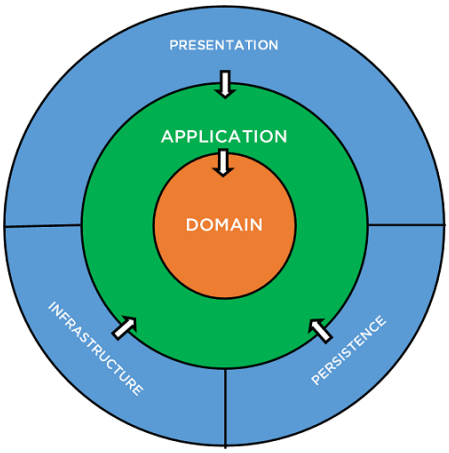
Így egyértelműen kiderült mennyi fával kell létrehozni az erdőt az optimális megoldáshoz. Az új megoldás jól reagál azokra az input adatokra is, amik nem szerepeltek a tanítómintában. Azokban az esetekben amikor az egyik paraméter eltér a tanítómintában szereplőtől, akkor is megközelítőleg 90%-ban jó eredményt ad válaszul.

6. Fejlesztés

Ebben a fejezetben a komponensek fejlesztéséről és az egymás közötti kapcsolatok kiépítéséről lesz szó.

6.1. A szerver architektúrájának implementálása

A felépítése a Clean Architecture elveit követi [31]. Először létrehoztam az elszeparált rétegeket. A Domain réteget, amiben az entitások lesznek. Az Application réteget, amiben a szolgáltatások, model osztályok, validátorok és mapperek lesznek. A Presentation réteget, ami lesz maga az API a kontrollerekkel, majd a Persistance és Infrastucture rétegeket, amiben az adatbázis elérés és a repository-k, illetve az utóbbi rétegben a csali ajánló rendszer elérése lesz. Az elkészült rétegek után a függőségek beállítása következet a hagyma szerű felépítés szerint befelé. [32]



6.1. ábra: Clean Architecture (forrás: [31])

6.2. Alapvető funkciók implementálása

Először a Domain réteget hoztam létre a szükséges entity-kel (ApplicationUser, Catch, Fishing, Species). A felhasználó entitás és a szerepkörök létrehozásánál a Microsoft Identity csomagját használtam segítségül. Az Application rétegben a CQRS (Command Query Responsibility Segregation) alapján készítettem el külön az író és külön az olvasó műveleteket [33]. Nagy előnye a terhelés kiegyenlítés. Ugyan azon szerverrel több kérést is ki lehet szolgálni, hogyha jellemzően több olvasás van, mint írás. Mindemellett könnyebben kezelhető és tesztelhetőbb kódot kapunk.
Az Application réteg a Presentation réteggel van kapcsolatban, ahol a Controller segítségével tud kommunikálni a kliens alkalmazásokkal. A rétegek közti adatküldés DTO-k (Data Transfer Object) segítségével valósul meg. A DTO-k és Entity-k közti átalakítást az AutoMapper végzi. A Persistance rétegben létrehoztam a repository-kat és a DAO-kat (Data Access Object) a hozzájuk illő domain entity-khez. A repository-kat az adatok módosításához, míg a DAO-kat csak az egyszerű adat lekéréshez használom. A Controllerek létrehozása során definiáltam a kellő HTTP kérésekhez szükséges függvényeket. Ennek a rétegnek a feladata a JSON alapú objektumok deszerializálása, majd tovább küldése az Application rétegnek. A visszakapott eredményt szerializálja és visszaküldi a kliens alkalmazásnak.

6.3. Adatbázis szerver kiépítése

A fejlesztést egy MacBookon végeztem, ezért szükségem volt egy olyan környezetre, ahol futtathatom az MSSQL szervert. Az MSSQL szerver jól működik együtt a .NET keretrendszerrel a LINQ segítségével könnyen készíthetünk lekérdezéseket. A Docker Desktop segítségével létrehoztam egy konténert, amiben a kellő image-t felhasználva elkészítettem az adatbázis szervert, amit lokálisan el tudtam érni. Így már az Entity Framework használatával a korábban generált migration script futtatásával elkészült az adatbázis.
A konténerek és a virtuális gépek hasonló erőforrás virtualizációs technológiák. A konténerek egy egyszerű szoftver csomagok, amelyek tartalmazzák a szoftvert és a hozzá szükséges függőségeket a futtatáshoz. A konténerben lévő függőségek az operációs rendszernél magasabb szinten léteznek. Több konténert is lehet futtatni ugyanazon a gépen megosztott operációs rendszerrel, de minden egyes konténer külön álló folyamatként működik. Előnye, hogy egyszerű és csak magas szintű szoftvereket tartalmaz, így könnyen és gyorsan módosíthatóak. Továbbá kevesebb helyet foglalnak a virtuális gépekkel szemben, ahol a minden egyes virtuális gép tartalmazza az operációs rendszert is. Sok publikus előkészített konténer létezik, amiket fel lehet használni, ez időt spórol, jelentősen megkönnyíti a fejlesztők dolgát. Hátránya a virtuális gépekkel szemben, hogy nem lehet teljesen különálló, független rendszerként futtatni. Lehetséges, hogy egyik konténer hatással van a másikra a megosztott operációs rendszer miatt. Én az egyszerűség miatt választottam a konténerizációt, a Dockert pedig a népszerűség és a sok felhasználó miatt. Továbbá a Docker Hub egy hatalmas publikus tárhely, ahonnan a konténerek könnyen letölthetőek és telepíthetőek a lokális futtató környezetbe.

6.4. Felhasználó kezelés

A horgászatok felhasználókhoz vannak kötve, hogy minden egyes felhasználó csak a saját horgászatait és fogásait láthassa. Az alkalmazás használata előtt szükség van egy regisztrációra, ahol csak egy nevet, e-mail címet és jelszót kell megadni. A rendszer ellenőrzi, hogy minden adat meg van-e adva és utána létrehozza az új felhasználót. Amennyiben elfelejti a jelszavát van lehetőség egy új jelszó igénylésére. Ekkor az e-mail cím alapján megkeresi a rendszer a felhasználót, generál egy új jelszót majd a token segítségével beállítja azt és küld egy e-mailt, ami tartalmazza a friss jelszót. Az e-mail küldéséhez a beépített SmtpClient osztályt használtam.

6.5. A csali ajánló rendszer integrálása a web szolgáltatásba

A prototípusként korábban elkészített rendszert az Infrastructure rétegben hívom meg az átadott időjárási adatokkal. Egy post kérést küld a Python API-nak, ami az eredményt JSON formában küldi vissza. A kommunikációhoz a HttpClient osztályt használtam.
Alapvetően két variáció volt az ajánló rendszer elkészítésére az egyik a C# és az ML.NET keretrendszer használata, a másik a Python és a Scikit-learn könyvtár használata. A második lehetőségnél jelentősen több példa kód és ismeretanyag állt rendelkezésre, abból adódóan, hogy népszerűbb és könnyen használható. Továbbá sok olyan könyvtár van hozzá, ami segítséget nyújthat, megkönnyíti a fejlesztő dolgát, ilyen például az

előfeldolgozás vagy a címkézés. Egyszerűbb, rövidebb kódot lehet írni vele, viszont lassabb lesz mintha az ML.NET-et használná a fejlesztő.

6.6. A statisztikák elkészítése

A horgászatokról és összesítve is különböző statisztikák készülnek, amikből a horgász hasznos következtetéseket vonhat le. Az idei év horgászatainak száma havi bontásban statisztika a főoldalon fog megjelenni, így látja a horgász, hogy mennyi időt töltött a vízparton. A többi:

- kifogott halak darabszáma
- legnagyobb halak (max. 3db)
- leggyakrabban fogott halfaj
- legsikeresebb csalik (max. 3db)
- kifogott halak összesített súlya

a külön statisztikai oldalon fog megjelenni összesítve és külön megjeleníthetőek horgászatonként is.

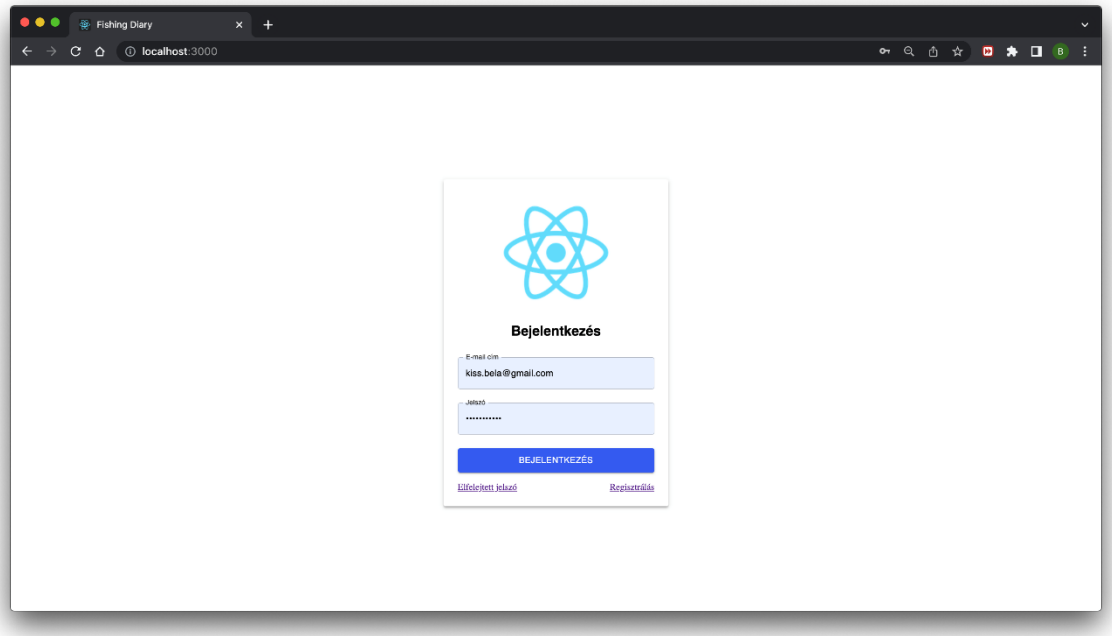
6.7. Kliens alkalmazások architektúrájának felépítése

A React és React Native alkalmazások hasonló architektúrára épülnek, külön vannak a model osztályok, a megjelenítéshez a logika ezek a mobx store-ok, amikben az aktuális állapotok vannak tárolva és az agent.ts fájl, amiben a kommunikáció valósul meg a szerverrel az axios csomag segítségével. A többi könyvtár az oldalak közötti navigáció és a megjelenítés, amik külön oldalakra vannak szeparálva.

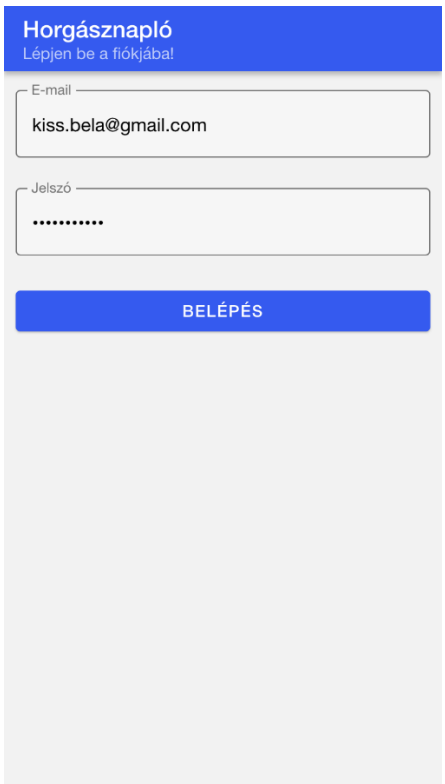
6.8. A kliens alkalmazások kinézetének implementálása

A fejlesztés során a mostanság modern TypeScript nyelvet használtam. Amit később a fordító átalakít JavaScript-re. Ennek köszönhetően típusos nyelvvel lehet fejleszteni, mely segít kiküszöbölni a típusbizonytalanságból eredő potenciális hibákat. A web alkalmazás megjelenítéséhez a Material UI keretrendszert használtam, míg a mobil alkalmazásnál a React Native Paper-t, mind a kettő az npm csomagkezelőn keresztül egyszerűen telepíthető. Egyszerű letisztult megjelenést biztosít mind a kettő az alkalmazásnak. A fejlesztés során elkészítettem a külön oldalakat, melyeket utána összekötöttem a menün keresztül történő navigációval.

A szoftver elindításakor egy bejelentkező felület jelenik meg, ahol lehet regisztrálni, illetve az elfelejtettjelszó helyet újat igényelni *(6.2. ábra és 6.3 ábra)*.

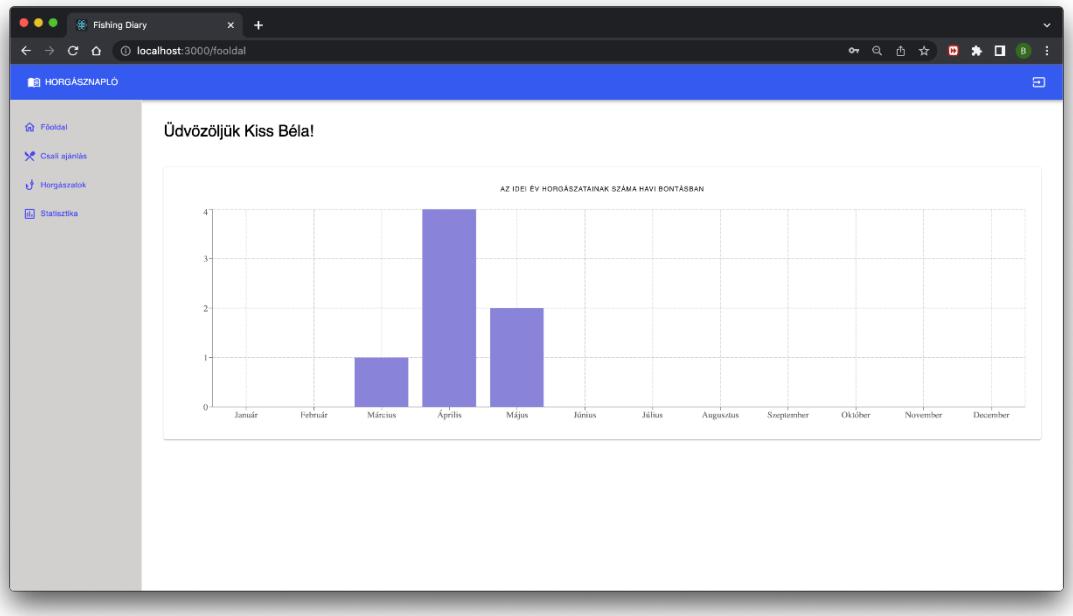


6.2. ábra: Webes kliens: bejelentkezés (forrás: saját ábra)

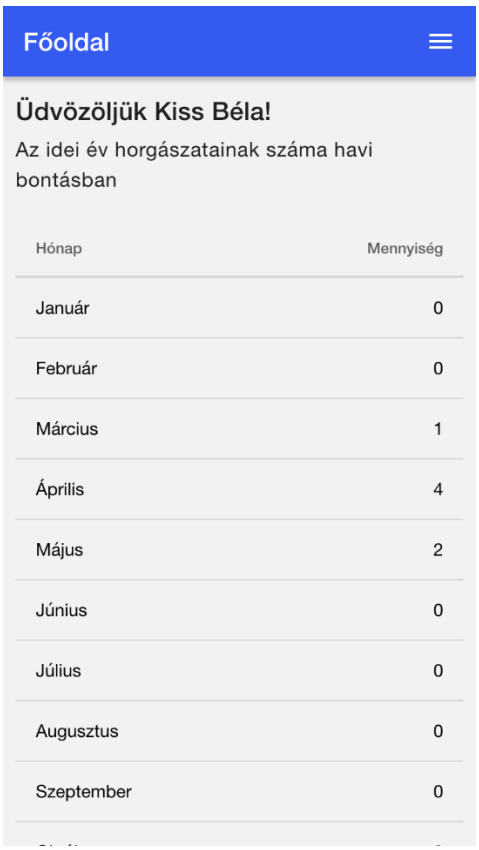


6.3. ábra: Mobil kliens: bejelentkezés (forrás: saját ábra)

A bejelentkezés után megérkezünk a Főoldalra *(6.4. ábra és 6.5 ábra)*. Itt a felhasználó látja, hogy az idei évben hány horgászaton vett részt havi bontásban. A menü pontok webes alkalmazásnál bal oldalt a mobilosnál pedig a jobb felső sarokban találhatóak. Ezek használatával tudunk navigálni az alkalmazásokban.

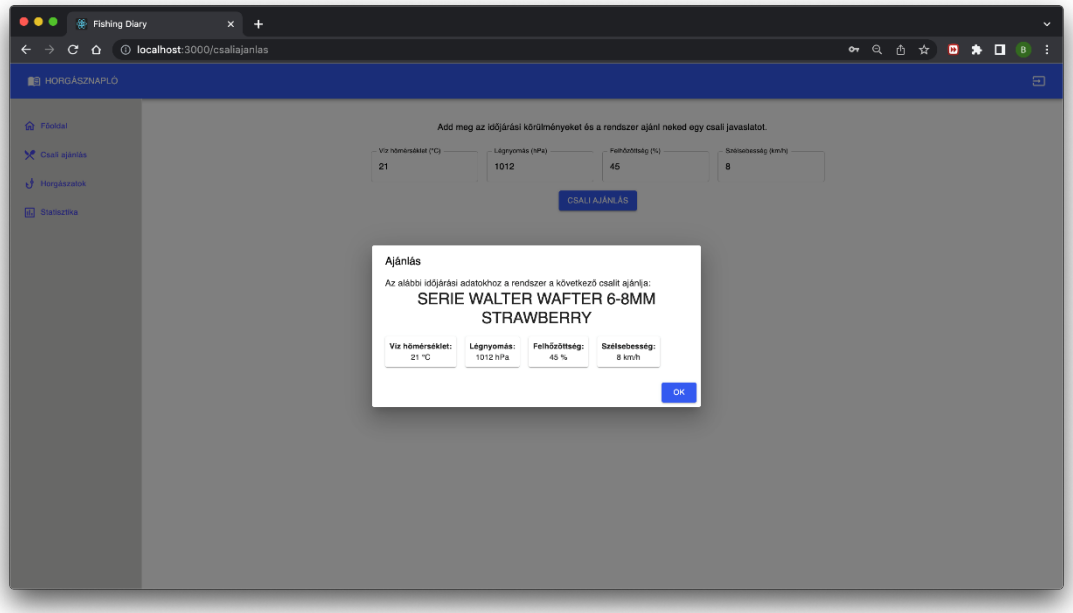


6.4. ábra: Webes kliens: Főoldal (forrás: saját ábra)

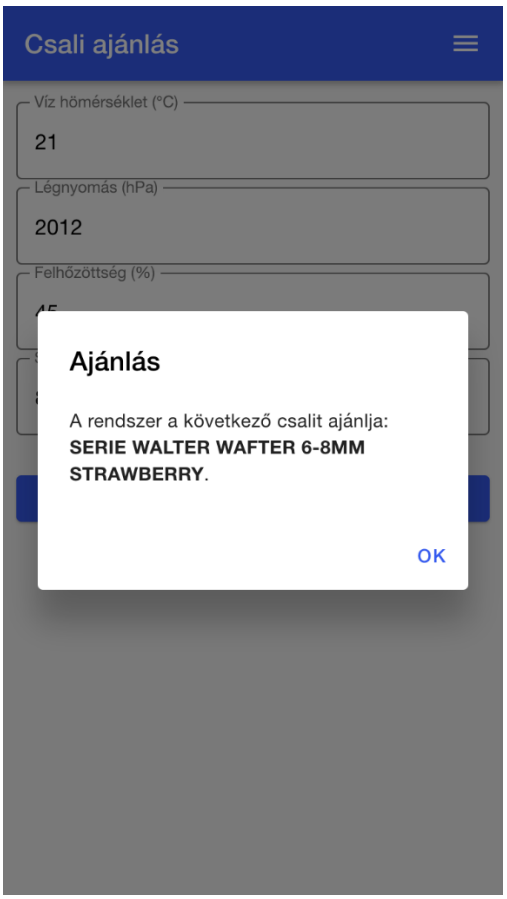


6.5. ábra: Mobil kliens: Főoldal (forrás: saját ábra)

A Csali ajánló oldalon *(6.6. ábra és 6.7 ábra)* az időjárási adatok megadása után a rendszer a véletlen erdő modell segítségével ad egy ajánlást, hogy milyen csalival érdemes horgászni ezen körülmények között.

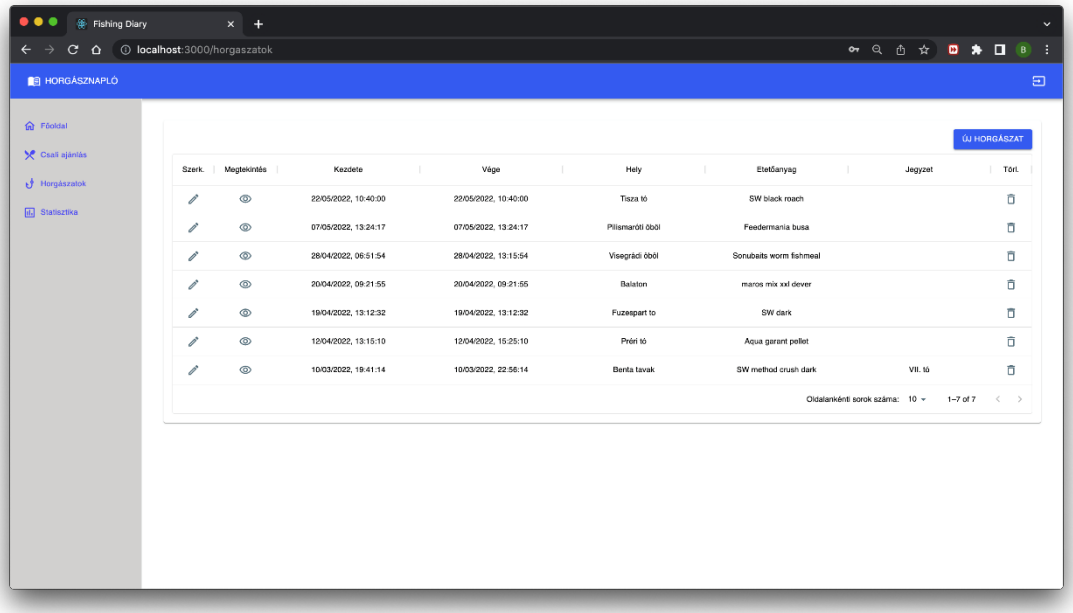


6.6. ábra: Webes kliens: Csali ajánló oldal (forrás: saját ábra)



6.7. ábra: Mobil kliens: Csali ajánló oldal (forrás: saját ábra)

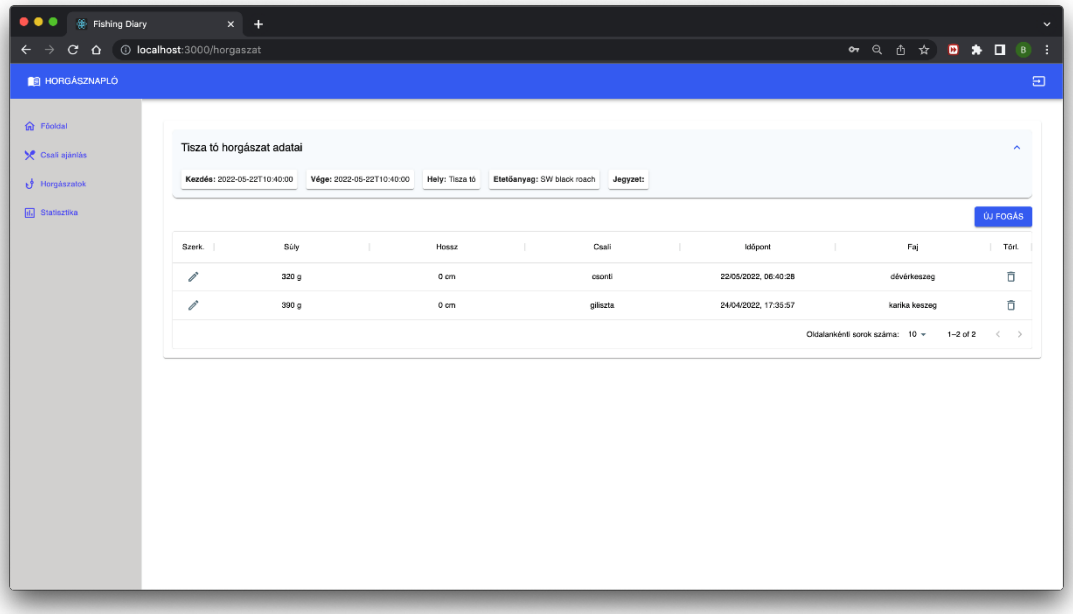
A Horgászatok oldalon lehet böngészni a horgászatok között *(6.8. ábra és 6.9 ábra)*. Törölni és módosítani is lehet a horgászatokat, illetve újakat felvinni. A megtekintésnél megjelennek a részletes adatok és láthatóak lesznek a horgászathoz tartozó fogások.



6.8. ábra: Webes kliens: Horgászatok oldal (forrás: saját ábra)

6.9. ábra: Mobil kliens: Horgászatok oldal (forrás: saját ábra)

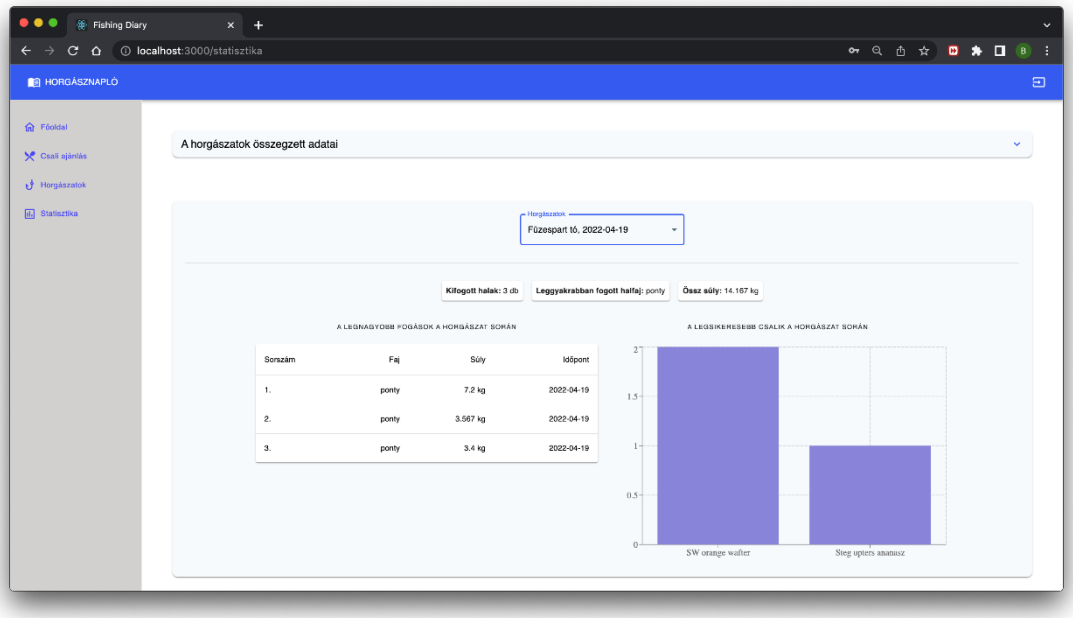
A Fogások oldalon a horgászathoz tartozó fogások vannak listázva *(6.10. ábra és 6.11 ábra)*. Megjelennek még a horgászat adatai, hogy tudjuk melyikbe léptünk bele. Itt is lehet felvinni új fogásokat vagy meglévőt módosítani, illetve törölni. A megtekintés után megjelenik a fogász részletes adatai ez a mobil kliensnél elengedhetetlen, mert a limitált megjelenítése felület miatt nem fér ki minden fontos adat a kijelzőre.



6.10. ábra: Webes kliens: Fogások oldal (forrás: saját ábra)

6.11. ábra: Mobil kliens: Horgászat oldal (forrás: saját ábra)

A Statisztika oldalon az összesített statisztikák vagy az adott horgászatra leszűrt statisztikák tekinthetőek meg *(6.12. ábra és 6.13 ábra)*. A webes alkalmazásnál egy oldalon jelenik meg mind a kettő az összesített statisztikák egy lenyíló accordion komponensben találhatóak.



6.12. ábra: Webes kliens: Statisztika oldal (forrás: saját ábra)

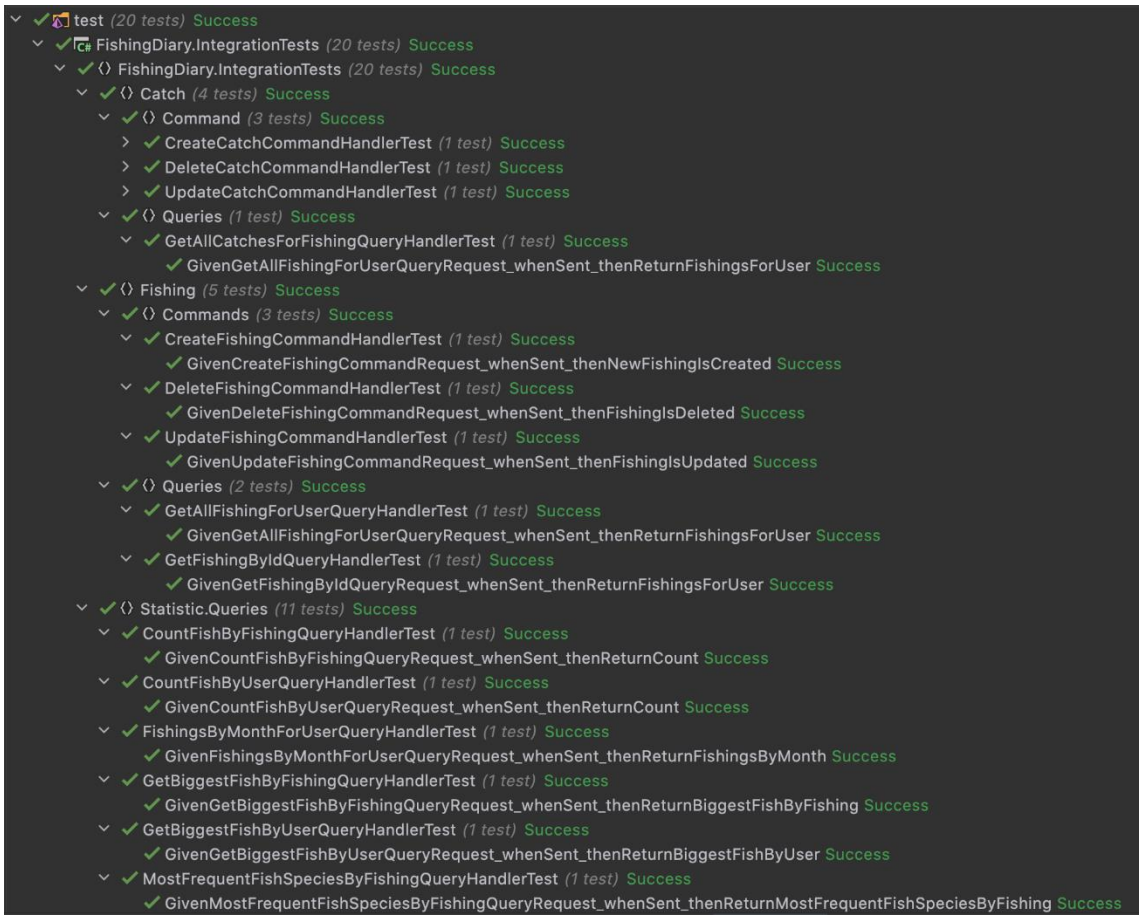
6.13. ábra: Mobil kliens: Statisztika oldal (forrás: saját ábra)

7. Tesztelés és értékelés

A fejezet során az elkészített rendszer teszteléséről, illetve az értékeléséről esik szó.

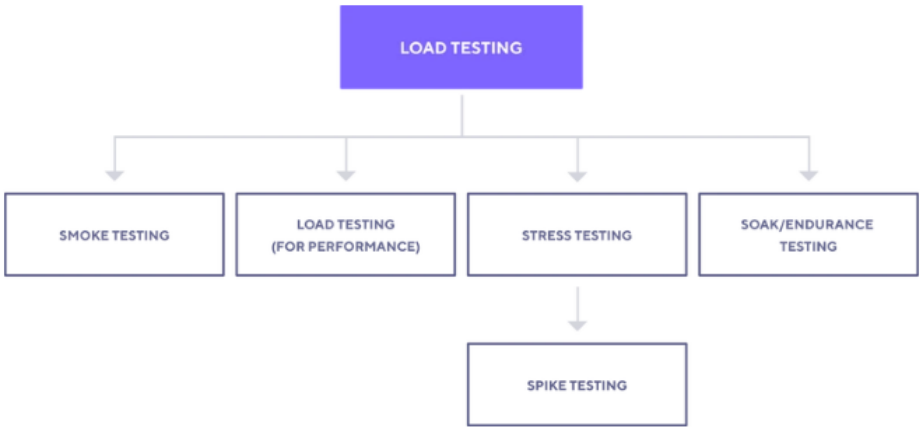
7.1. Az alkalmazás tesztelése

A szerver oldali rétegek lettek tesztelve. Ehhez egység és integrációs teszteket használtam. Az integrációs teszteket nevezném inkább fél-integrációs teszteknek, mert ezek nem egy úgynevezett teszt környezetben futottak le. A teszt környezetnek a lehetőségek szerint legjobban kell hasonlítania a végleges környezetre. A tesztelés során a tesztek a fejlesztési környezetben futottak le. Az egység tesztek során az összes függőséget ki kell mockolnunk, ezekre a tesztekre többnyire akkor van szükség, amikor egy adott eljárás vagy függvény mélyebb, összetettebb logikát foglal magába. A fél-integrációs tesztek futtatása során is metódusokat ellenőrzünk, hogy jól működnek-e. Azzal a különbséggel, hogy ebben az esetben a függőségeket nem mockoljuk ki, hanem valóban létrehozzuk. A tesztelés során az xUnit, az NFluent, a Shouldly és a Moq könyvtárakat használtam. Az xUnit segítségével egység teszteket lehet készíteni. Az NFluent és a Shouldly egy asserter könyvtár, amivel a kapott eredményt lehet ellenőrizni. A Moq használatával tudjuk megszüntetni a függőségeket, amelyekre nincsen szükségünk az egység tesztek során. A tesztek során a névadást a Given-When-Then minta alapján végeztem. A Given a kiinduló helyzet, a feladat végrehajtás előtti állapota. A When, hogy mit csinálunk a kiinduló állapottal. A Then pedig megmutatja mit várunk el, miután a feladat végrehajtódott. Egy metódus készítése során fontos a jó, leíró névválasztás és ez ugyan úgy fontos a teszteseteknél is. Amennyiben ezek teljesülnek, nincs szükség a metódus fölé kommenteket írni és clean code-ot kapunk. A fél-integrációs és az egység tesztek során a AAA szabályt követtem, tehát az Arrange, Act, Assert felépítést. Az Arrange részben beállítjuk a teszthez szükséges követelményeket. Az Act során hívódik a tesztelni kívánt metódus. Az Assert részben pedig az ellenőrzés történik, hogy az elvárt eredményt kaptuk-e. Ennek a szabálynak köszönhetően a tesztek átláthatóak, könnyen megérthetőek lesznek.
A fél-integrációs tesztek segítségével sikerült teljeskörűen tesztelni az Application réteget. Ehhez készítettem egy ősosztályt, amiben a teszteléshez szükséges függőségeket állítottam be. Minden teszt futása előtt újra inicializálja az in-memory adatbázist, hogy egymástól függetlenül tudjanak futni, ne legyenek hatással egymásra a tesztesetek. A Persistence rétegben találhatóak az adatbázis műveletek, amelyeket az Application réteg is használ. Ezáltal az a réteg is tesztelésre került. Amennyiben hibás teszt volt a futtatás után, pontosan tudtam hol kell keresni az adott hibát. Ebben a fejlesztői környezetek által tesztekhez készített futtatói felületek voltak a segítségemre, amik pontos hiba üzeneteket adtak. A tesztelés során 20 darab tesztel minden egyes command-ot, illetve query-t sikerült tesztelni *(7.1. ábra)*, amik biztosították a helyes működését a rendszernek.



7.1. ábra: Integrációs tesztek (forrás: saját ábra)

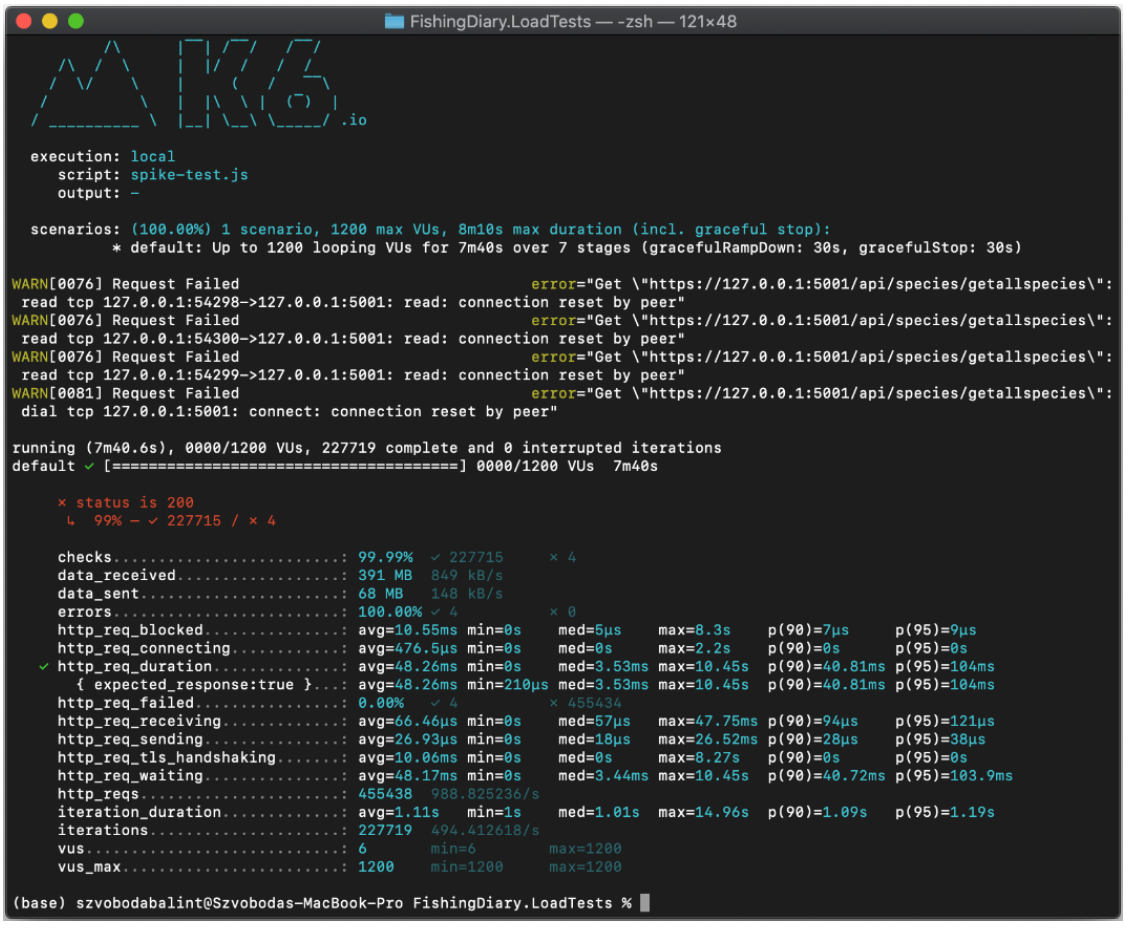
Az API tesztelése során szokás terheléses teszteket is elvégezni, melyeket nem funkcionális teszteknek neveznek. Ezeket a teszteket is mint az integrációs teszteket célszerű egy olyan teszt környezetben futtatni, ami minél jobban hasonlít majd a végleges környezetre. A terheléses teszteknek több fajtája is van *(7.2. ábra)*.



7.2. ábra: A terheléses tesztek típusai (forrás: [34])

A smoke teszt feladat annak az ellenőrzése, hogy a rendszer képes-e kezelni a minimális terhelést, minden rendesen működik-e. A teljesítmény tesztet a rendszer viselkedésének meghatározására használható normál és magas használat között. A cél, hogy stabilan üzemeljen a rendszer, ha egyszerre több felhasználó is működteti. A stressz és a spike

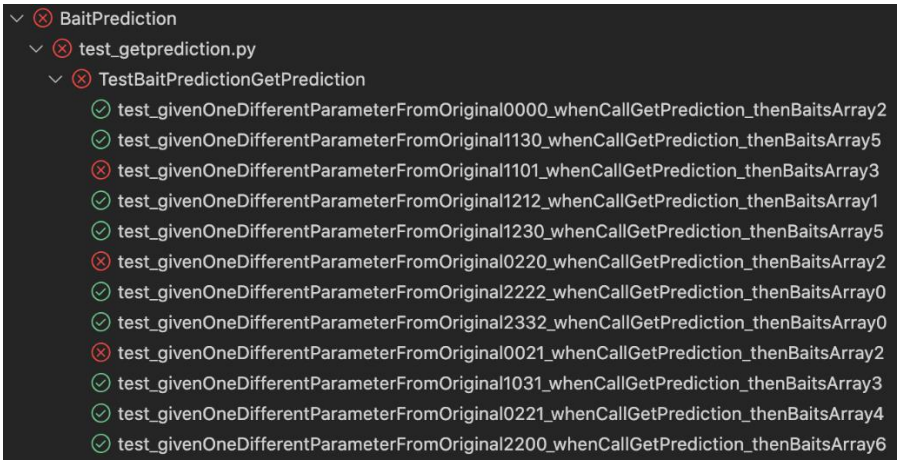
teszt feladata a rendszer határainak megtalálása extrém körülmények között. Ezeknél a teszteknél egy nagyobb terhelést kap a rendszer, mint egy teljesítmény tesztnél. A stressz és a spike teszt közötti az a különbség, hogy a spike teszt azonnal nagy terhelés alá vonja a rendszert, míg a stressz teszt fokozatosan építi fel az extrém terhelést. A soak vagy endurance teszt a hosszabb távú megbízhatósággal foglalkozik. Feltárja azokat a teljesítmény és megbízhatósági problémákat, amelyek abból adódnak, hogy a rendszer hosszú ideig van terhelés alatt. Az API tesztelése során nem funkcionális teszteket is alkalmaztam, amihez a Grafana Labs k6 tesztelő eszközét használtam. Ez egy nyílt forráskódú, könnyen használható eszköz, amiben JavaScript nyelven lehet írni a teszteket. A teljesítmény teszt során egy általános és magas használatot szimuláltam ez 500-700 felhasználó együttes használata volt. A stressz és spike teszt során a csúcsérték 1200 felhasználó volt, ezt hosszabb, illetve rövidebb idő alatt érték el. A tesztek során a HTTP kérések átlagos válaszideje 19 és 105 ms között volt, erre beállítottam egy 200 ms küszöb értéket. A tesztek ideje alatt végig a küszöb érték alatt maradtak a válaszidők. Emellett még a sikertelen HTTP kéréseket vizsgáltam, ami egyedül a spike teszt során jelentkezett. Akkor történtek a sikertelen HTTP kérések, amikor hirtelen rövid idő alatt jelentősen megugrottak a kérések száma és ezt nem tudta kezelni a rendszer *(7.3. ábra)*.



7.3. ábra: Spike teszt (forrás: saját ábra)

A csali ajánló rendszernél egység tesztekkel ellenőriztem, hogy a rendszer a beérkező adatokat, jól dolgozza-e fel és a megfelelő címkét kapják-e. A véletlen erdő, nehezen tesztelhető, mert nem áll rendelkezésre teszt minta. Korábban a modell létrehozásánál az

OOB hiba megmutatta hány fából álljon az erdő, ekkor a tanító mintából kimaradt adatokkal tesztelte a modellt. Az előre megadott szabályoktól egy paraméterrel eltérő adatokat adtam meg a teszteseteknél. A rendszer így 75 százalékban adott jó választ *(7.4. ábra)*.



7.4. ábra: Véletlen erdő modell tesztelése (forrás: saját ábra)

7.2. Az alkalmazás értékelése

Nagyrészt sikerrel zárult a rendszer fejlesztése. Az elején lefektetett célok megvalósultak, minden fő funkció implementálása sikerrel zárult. Az alkalmazás fejlesztése alatt igyekeztem minél több best practice-t (legjobb gyakorlat) alkalmazni. Az egyik ilyen a MediatR csomag használata volt, aminek köszönhetően a szerver oldalon történő hibákat, ha nem is lettek explicit kezelve, akkor sem fog leállni. A hagyma architektúrának köszönhetően egy könnyen kezelhető, moduláris alkalmazás készült. Amelyben a rétegek könnyen kicserélhetőek. Ehhez elengedhetetlen volt az IoC (Inversion of Control) implementálása. Amelynek köszönhetően egy jól tesztelhető kód született. Sikerült egy olyan felhasználói felületet készíteni, ami modern és felhasználó barát is egyben. Erre meglehetősen sok időt fordítottam, hiszen a felhasználó ezen a felületen tudja kezelni az alkalmazást és csak ezzel tud érintkezni. A csali ajánló rendszer is tudja segíteni a kezdő, tapasztalatlan horgászokat egy mesterséges intelligencia által.

8. Befejezés

Az alkalmazás több területén rejlik még fejlesztési potenciál. Az elkészült megoldások átírásával, tovább fejlesztésével és a felhasználói élmény javításával lehetne még jobbá és hatékonyabbá tenni a befejezett szoftvert.

8.1. A felhasználói élmény javítása

Jelenleg a mobil eszközökre készített kliens csak az adatok megjelenítésére alkalmas és a módosítására nem. Viszont a böngészőben futó alkalmazás képes a reszponzív működésre, így az okoseszközökön is lehet teljes értékűen használni. De a mobil alkalmazás funkcióinak kibővítésével a felhasználók kényelmesebben tudnák kezelni a szoftvert.

8.2. Választott megoldások javítása

A rendszer szerver oldali moduljai jelenleg HTTP kéréseken és válaszokon keresztül kommunikál egymással. Mind biztonság béli mind teljesítmény béli problémákhoz vezethet. Az előbbi esetnél nem kívánatos személyek is elérhetik az egyes modulokat, mivel külön entitásként viselkednek, ezért bárki számára elérhetők. Az utóbbi probléma is az előbb említett problémából fakad, hiszen az egyes modulok nem valamilyen kommunikációs csatornán keresztül lépnek kapcsolatba egymással. Hanem kénytelenek az interneten keresztül a HTTP protokoll segítségével kommunikálni, ami általánosságban véve egy lassú műveletnek számít. A problémák megoldására egy platform független üzenetküldő szolgáltatás használata lenne, ilyen például a Kafka és a RabbitMQ. Amelyek az aszinkron kommunikációs típust használják, míg a HTTP szinkron. Az AMQP protokoll is aszinkron üzenet küldést használ, aminek az egyik előnye a gyorsaság.

A teljesítmény béli javítására a cache-lés is megoldást nyújt. Amellett, hogy a CQRS minta megvalósításával már sikerült gyorsítani az alkalmazáson, tömérdek alkalommal lép interakcióba az adatbázissal, amik költséges műveletek. Ezek számának csökkentésére általában valamilyen cache-lési mechanizmust alkalmaznak. A projektemben számos adatbázis lekérdezés van, ilyenek a statisztikák lekérése, horgászatok és fogások betöltése. Nyilvánvaló megoldás lenne ezeket eltárolni, hogy ne legyenek fölösleges adatbázis interakciók. Több lehetőség is van erre, az egyik a felhasználó eszközének memóriájába való cache-lés. Amely viszont hosszú távú használat esetén teljesítmény béli romláshoz vezethet, amit a memória terület csökkenése okoz. Ebben az esetben a felhasználó eszközének memória kapacitása sorsdöntő. A másik lehetőség, ha nem a felhasználó készülékének a memóriáját használjuk, hanem magának a szervernek a memóriáját. Az Azure Redis Cache és az NCache is ezt valósítja meg. Ekkor a fejlesztők biztosan tudják mekkora memória áll rendelkezésükre.

A rendszer fejlesztése során lehetett volna használni a TDD (Test Driven Development) szoftverfejlesztési gyakorlatot a kód minőségének javítása érdekében. Ennél a módszernél a teszteket írjuk meg először az adott metódushoz, majd utána implementáljuk az éppen szükséges kóddal, hogy a teszt sikeresen fusson le. Következő lépésben refaktorálunk, majd kezdjük elölről az egészet. Így a hibákat már az elején, a

fejlesztés korai szakaszában kiszűrhetjük. A tesztek megírása közben hamarabb eszünkbe jutnak a szélsőséges esetek, így már előre felkészíthetjük rá a kódunkat. Továbbá a tesztelés egyfajta dokumentáció is, ha új fejlesztő érkezik a csapathoz, akkor a mindig aktuális tesztek alapján könnyebben megértheti az alkalmazás működését.

8.3. Csali ajánló rendszer továbbfejlesztése

Jelenleg ez az ajánló rendszer egy döntési fákból álló véletlen erdő. Aminek a tanításához a horgász körökben tapasztalható általános vélekedéseket használtam. Az alkalmazás hosszú távú használata során visznek fel folyamatosan új fogásokat a horgászok. Az új fogás résznél ki lehet bővíteni a megadandó adatokat az időjárási adatokkal, amiket az adatbázisban el lehetne tárolni akár egy külön álló táblában. Így már nagyjából egy év után elegendő mennyiségű adatot gyűjtene a rendszer egy neurális hálózat tanításához. A hálózat megtalálna valamilyen mintát és fel tudna épülni egy modell. Az évente gyűjtött új adatokkal együtt lehet újra elvégezni a tanítást, a biztosabb ajánlások érdekében. A neurális hálózat létrehozásához a jelenleg is használatos python és TensorFlow könyvtárat, illetve a NuGet csomag kezelőben elérhető CTNK könyvtárat lehet használni. A CNTK könyvtár használatával nem kell egy külön külső szolgáltatást létrehozni, hanem az Infrastructure rétegben is lehet implementálni.

Tartalmi összefoglaló

A szakdolgozatom céljaként azt tűztem ki, hogy készítsek egy olyan rendszert, ami segíti a horgászok horgászatait és fogásait rendszerezni. Továbbá a kezdő horgászoknak tud segítséget nyújtani a csali választásban egy ajánlással. Ennek a rendszernek elérhetőnek kell lennie böngészőn keresztül, illetve mobil eszközökön.

Az irodalomkutatás során körbe jártam azokat a technológiákat, amiket a projekt során fel lehet használni. Egy CSS könyvtár és az egyik legelterjedtebb JavaScript keretrendszer bemutatása után a három legismertebb cross-platform lehetőség körbejárása következett. Esett még szó a Microsoft webalkalmazásokhoz készített technológiájáról az ASP.NET-ről. A fejezet végén bemutatásra kerültek gépi tanulási algoritmusok, illetve a neurális hálózatok, amik a csali ajánló rendszer kifejlesztéséhez használhatóak a projekt során. Ezekhez az algoritmusokhoz, illetve a neurális hálózathoz a leggyakrabban használt könyvtár a TensorFlow is ismertetésre került.

A következő fejezet a hasonló rendszerek volt, ebben más horgásznapló alkalmazásokat tanulmányoztam. Továbbá olyan alkalmazásokat kerestem, amikben előfordulnak olyan technológiák, amiket felhasználhatok a csali ajánláshoz. A fejezet végén összegeztem a tapasztalatokat, hogy melyik alkalmazásból, mit és hogyan lehetne felhasználni az én projektemhez is.

A tervezés során ismertettem, hogy milyen funkciókat szeretnék implementálni a fejlesztés során. Bemutattam egy általános munkafolyamatot egy szekvencia diagramon keresztül. Hogyan indul el a folyamat a felhasználói felületen való kattintástól az adatbázisból lekért információig majd az egész, hogyan jut vissza a felhasználóig. Azután a rendszer bemutatása egy komponens diagram segítségével következett.

A tervezés után a prototípus elkészítésén volt a sor. Az ebben szakaszban elkészült a csali ajánló rendszer, ami egy véletlen erdő algoritmuson alapszik. A működéséhez először az adatok feldolgozására volt szükség.

A fejlesztés szakaszban elkészült a teljes rendszer. A többi komponens, amik a web szolgáltatás, a webes- és a mobil kliens, befejezése, és a web szolgáltatás a korábban elkészített csali ajánló modullal való összekapcsolása. A kommunikációjuk HTTP kérések és válaszok formájában történik.

A tesztelés során a web szolgáltatást és a csali ajánló modult teszteltem egység és integrációs tesztek segítségével. Az API terheléses teszteken is átesett, amiből kiderült a teljesítménye és stabilitása. A fejezet második felében a rendszer értékelése következett.

Az utolsó fejezetben a lehetséges továbbfejlesztések ismertetése és a felhasználói élmény javítása került bemutatásra, illetve a csali ajánló rendszer egy másik lehetséges implementálása a korábban begyűjtött adatok segítségével.

Abstract

The goal I set to accomplish with my thesis was to build a system that helps anglers organize their catches and fishing. Furthermore, I wanted it to assist amateurs in selecting bait with recommendations. This system needs be available both in browser and on mobile.

During my literature research I assessed the technologies that could fit the project. After exploring a CSS library and one of the most prevalent JavaScript frameworks ensued the presentation of the three most well known cross-platform solutions. Microsoft's technology for web applications, the ASP.NET was also considered. At the end of the chapter I showcased machine learning algorithms and neural networks that may be used for the bait recommendation system. I expounded the most commonly used library, the TensorFlow, for these algorithms and neural networks as well.

The next chapter was about similar systems, where I examined other fishing diary applications and searched for ones that utilized technologies that I could implement in suggesting baits. At the end of the chapter I summarized my experiences, what could be used and why.

During the design process I layed out what functions I would like to implement in development. I described a general workflow through a sequence diagram. How the process starts from the click on the user interface, to the queried information from the database, and how it returns to the user. After that, I showcased the system on a component diagram.

The next step after the design was the creation of a prototype. The bait recommendation component, which is based on a random forest algorithm, was finished during this stage. For it to operate, data processing was required.

The whole system was finished during the developmental stage. The seperate components, which are the web service, the web and mobile clients, were completed, and the web service was connected to the previously finished bait recommendation module. The communication is done through HTTP requests and responses.

I tested the web service and the bait recommendation module with unit tests and integration tests during the testing phase. The API underwent load tests, which measured its' performance and stability. The second half of the chapter is the evaluation of the system.

The last chapter presents possible ways of future improvements, betterments to the user experience and a different implementation of the bait recommendation component using previously collected data.

Ábrajegyzék

Irodalomjegyzék

[1] MDN – Structuring the web with HTML (https://developer.mozilla.org/en-US/docs/Learn/HTML), utoljára megtekintve: 2020.11.20

[2] W3schools – What is HTML DOM (https://www.w3schools.com/whatis/whatis_htmldom.asp), utoljára megtekintve: 2020.11.20

[3] Bootstrap v4.5 – About (https://getbootstrap.com/docs/4.5/about/overview/), utoljára megtekintve: 2020.11.20

[4] MDN – JavaScript (https://developer.mozilla.org/hu/docs/Web/JavaScript), utoljára megtekintve: 2020.11.20

[5] React – A JavaScript library for building user interfaces (https://reactjs.org/), utoljára megtekintve: 2020.11.22

[6] Cross-platform vs Native Mobile App Development: Choosing the Right Development Tools for Your Project (https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81), utoljára megtekintve: 2021.05.19

[7] Xamarin – Open-source mobile app platform for .NET (https://dotnet.microsoft.com/en-us/apps/xamarin), utoljára megtekintve: 2021.05.19

[8] React Native – Learn once, write anywhere (https://reactnative.dev/), utoljára megtekintve: 2021.05.19

[9] Flutter – Build apps for any screen (https://flutter.dev/), utoljára megtekintve: 2021.05.20

[10] ASP.NET – Open-source web framework for .NET (https://dotnet.microsoft.com/en-us/apps/aspnet), utoljára megtekintve: 2021.05.20

[11] Towards Data Science – Machine Learning Algorithms In Layman's Terms, Part 1 (https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-1-d0368d769a7b), utoljára megtekintve: 2021.12.29

[12] Towards Data Science – Coding Deep Learning For Beginneres (https://towardsdatascience.com/coding-deep-learning-for-beginners-types-of-machine-learning-b9e651e1ed9d), utoljára megtekintve: 2021.12.29

[13] Edureka! – What is Fuzzy Logic in AI and What are its Applications? (https://www.edureka.co/blog/fuzzy-logic-ai/), utoljára megtekintve: 2021.12.30

[14] Medium – Fuzzy Logic: Approximating Imprecise Statement (https://medium.com/it-paragon/fuzzy-logic-approximating-imprecise-statement-54d444237820), utoljára megtekintve: 2021.12.30

[15] SZTE – Gépi Tanulás A Gyakorlatban, Döntési Fa Osztályozó (https://www.inf.u-szeged.hu/~rfarkas/ML20/dontesi_fa.html), utoljára megtekintve: 2022.01.02

[16] Towards Data Science – Machine Learning Algorithms In Layman's Terms, Part 2 (https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-2-a0a74df9a9ac), utoljára megtekintve: 2022.01.02

[17] Prediction of wind pressure coefficients on building surfaces using Artificial neural network architecture (https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051), utoljára megtekintve: 2021.05.20

[18] Society of AI – Introduction to Neural Networks and Deep Learning (https://medium.com/@societyofai/introduction-to-neural-networks-and-deep-learning-6da681f14e6), utoljára megtekintve: 2021.05.20

[19] Towards Data Science – Activation Functions in Neural Networks (https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6), utoljára megtekintve: 2021.05.20

[20] Towards Data Science – Activation Functions (https://towardsdatascience.com/activation-functions-in-neural-networks-58115cda9c96), utoljára megtekintve: 2021.05.20

[21] Towards Data Science – Activation Functions in Deep Learning (https://towardsdatascience.com/activation-functions-in-deep-neural-networks-aae2a598f211), utoljára megtekintve: 2021.05.20

[22] TensorFlow – Guide (https://www.tensorflow.org/guide), utoljára megtekintve: 2021.05.27

[23] Keras – About (https://keras.io/about/), utoljára megtekintve: 2021.05.27

[24] Google Play – Halnapló (https://play.google.com/store/apps/details?id=hu.halnaplo.app&hl=hu), utoljára megtekintve: 2021.05.21

[25] Fishinda (http://app.fishinda.com/hu/#/), utoljára megtekintve: 2021.05.21

[26] Medium – Simple Housing Price Prediction Using Neural Networks with TensorFlow (https://medium.com/@robertjohn_15390/simple-housing-price-prediction-using-neural-networks-with-tensorflow-8b486d3db3ca
), utoljára megtekintve: 2021.05.22

[27] Kaggle – Boston Housing Price Prediction dataset (https://www.kaggle.com/c/boston-housing), utoljára megtekintve: 2021.05.22

[28] C. Mercy Amrita and P. Karthickumar, NEED FOR MOBILE APPLICATION IN FISHING (https://www.ijset.net/journal/1227.pdf), utoljára megtekintve: 2021.05.27

[29] Towards Data Science – Decision Tree Classifier explained in real-life: picking a vacation destination (https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575), utoljára megtekintve: 2022.01.07

[30] Towards Data Science – Random Forest Algorithm explained in real-life example and some Python code (https://towardsdatascience.com/random-forests-algorithm-explained-with-a-real-life-example-and-some-python-code-affbfa5a942c), utoljára megtekintve: 2022.01.09

[31] Playing with Clean Architecture and CQRS pattern using asp.net core, EF and Dapper (https://mahedee.net/playing-with-clean-architecture-and-cqrs-pattern-using-asp.net-core-ef-and-dapper/), utoljára megtekintve: 2022.05.04

[32] Robert C. Martin: Clean Architecture. Pearson Education, 2017

[33] Martin Fowler: Patternf of Enterprise Application Architecture. Pearson Education, 2002

[34] Grafana k6 – Test Types, Introduction (https://k6.io/docs/test-types/introduction/), utoljára megtekintve: 2022.05.05