**UNIVERSITATEA “ALEXANDRU IOAN CUZA”**

**DIN IAȘI FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

Titlu

*Personalized Teachable Machine*

**propusă de**

Balcan Bruno - Cristian

Sesiunea: *luna Iulie, anul 2025*

**Coordonator științific**

Prof. dr. Alboaie Lenuța

**Introducere**

***Personalized Teachable Machine*** este o aplicație web destinată tuturor studenților acestei facultăți dar și celor dornici să învețe bazele unui domeniu esențial pentru tehnologia modernă, **Machine Learning***.*

Scopul Acestei aplicații este de a ajuta studenții să înțeleagă mai bine conceptele prezentate la cursul de *Învățare Automată*, să rezolve exerciții, să experimenteze cu modul de aplicare al algoritmilor, cât și să observe modul în care aceștia pot fi puși în practică și să analizeze cazurile în care un algoritm poate fi performant pentru rezolvarea unei anumite probleme. Această ,,culegere” de informații și probleme aplicative este menită nu doar pentru a ușura munca studenților în procesul de învățare, ci și pentru a onora efortul profesorilor de a ne arăta toate aceste concepte, prin punerea la dispoziția noastră de materiale explicative, mai detaliate decât ceea ce se află în curs, tocmai pentru a ne provoca curiozitatea și setea de cunoaștere. Având în vedere cele spuse mai sus, este un moment potrivit pentru a vorbi despre motivația din spatele acestei lucrări, aceasta fiind dată materialele suplimentare menționate, care m-au ajutat enorm de mult să înțeleg mai bine prin explicații detaliate, să pot învăța mai ușor și mai eficient și care ofereau răspunsuri la întrebările ridicate în urma fiecărui curs. Toate acestea m-au făcut să apreciez importanța acestui domeniu pentru viitorul nostru ca și programatori, modul în care tehnologia poate avansa la un complet alt nivel în viitorul apropiat și modul în care putem integra această tehnologie pentru a digitaliza alte domenii precum educația, medicina, cercetările științifice, etc.

O sursă de inspirație pentru acest proiect este reprezentată de platforma *Teachable Machine* oferită de Google. Pe lângă informațiile teoretice pe care aplicația le oferă studenților, aceștia vor avea la dispoziție un mecanism de interacționare cu algoritmii prezentați în curs, în mod similar cu cel oferit de platforma *Teachable Machine*: ei vor putea încărca o mulțime de date, vor antrena un model pe baza algoritmului ales și apoi vor putea folosi modelul antrenat pentru rezolvarea unor exerciții practice. Am vorbit despre partea de *,,Teachable Machine”* a aplicației, dar unde apare ideea de *,,Personalized”*? Aici, utilizatorii se vor putea înregistra în aplicație, creând conturi și având opțiunea de a primi suport prin email. În plus, aceștia vor avea la dispoziție un asistent AI antrenat pentru a răspunde atât întrebărilor din domeniul Machine Learning, cât și neclarităților studenților asupra suportului de curs, ei putând solicita pagina de unde pot afla o anumită informație, parafrazări din culegere, etc.

Consider acest proiect, prin tehnologiile utilizate și îmbinarea lor într-o singură aplicație menită să faciliteze învățarea, unul de actualitate, având potențialul de a asista într-un mod competent activitatea la curs a studenților. De asemenea, acesta poate deschide oportunități noi altor cursuri cu aplicabilități practice, o platformă menită să completeze cursurile și să ofere un mediu interactiv care să sporească interesul și motivația, asemenea modului în care și eu am fost motivat de materialele adiționale, scrise sau video, puse la dispoziție de către profesori, în scopul de a mă sprijini în procesul de educație.

**Contribuțiile** pe care mi le-am propus să le aduc prin această lucrare au un scop educativ, aplicația fiind destinată în principal studenților pe parcursul cursului de Învățare Automată. Așa cum am menționat mai sus, aplicația oferă posibilitatea de a oferi informații utile, și pune în practică cunoștințele cumulate, studenții având la dispoziție metode de experimentare cu algoritmii atât pentru antrenarea unor modele cât și pentru testarea acestora.

Modul în care lecțiile și capitolele vor fi puse la dispoziție este similar platformei W3School, deoarece am folosit această platformă de foarte multe ori și sunt de părere că oferă o interfață simplă și extrem de eficientă care servește cel mai bine scopului educativ al acestei lucrări.

Studenții vor putea selecta lecțiile pe care vor să le urmeze, acestea fiind determinate de algoritmii elementari descriși în curs precum ID3. De asemenea, precum este cazul algoritmilor care construiesc arbori de decizie, aplicația va oferi un mod interactiv de afișare a acestora, prin care studenții vor putea urmări modul în care se aplică algoritmul și criteriile care stabilesc nodurile de decizie.

La finalul fiecărui capitol vor exista și exerciții practice în urma cărora studenții vor putea să antreneze și să testeze modele pentru a se familiariza și mai mult cu algoritmul învățat.

În final, deși aplicația oferă posibilitatea de a urmări lecțiile fără a fi conectat, utilizatorii vor avea la dispoziție funcționalități personalizate precum asistentul AI, o interfață pentru a-și actualiza profilul, și asistență email pentru înregistrarea în aplicație și schimbarea parolei.

În continuare, voi vorbi despre capitolele acestei lucrări, precum arhitecturile software folosite, scopul și rolul lor în funcționarea aplicației, modul în care acestea lucrează și cum are loc comunicarea la nivel de aplicație.

Cuprins

[1 Arhitectura Backend 5](#_Toc200225004)

[1.1 Alegerea platformei .NET 5](#_Toc200225005)

[1.2 Structura arhitecturală – Clean Architecture 5](#_Toc200225006)

[1.3 RESTful API 6](#_Toc200225007)

[1.3.1 API-uri bazate pe controllere 6](#_Toc200225008)

[1.3.2 Documentarea API-ului – Swagger 7](#_Toc200225009)

[1.4 Baza de date (SQLite) 7](#_Toc200225010)

[1.5 Securitate 7](#_Toc200225011)

[1.6 Alte servicii 7](#_Toc200225012)

[2 Implementarea algoritmilor ML – Serverul Python 8](#_Toc200225013)

[3 Aplicația Frontend – Interfața utilizatorului 9](#_Toc200225014)

[4 Idei pentru viitor și concluzii 10](#_Toc200225015)

[4.1 Idei și funcționalități potențiale 10](#_Toc200225016)

[4.2 Concluzii finale 11](#_Toc200225017)

# Arhitectura Backend

Reprezintă componenta principală a aplicației și punctul central de comunicare între toate tehnologiile software folosite. În această secțiune voi vorbi despre limbajele de programare folosite, motivul pentru care acestea au fost alese și rolul pe care îl îndeplinesc în cadrul proiectului.

## Alegerea platformei .NET

Încă de la început mi-am dorit să mă folosesc de acest proiect pentru a învăța să lucrez cu cât mai multe tehnologii noi și să experimentez într-un mediu bine documentat, dar strâns legat de cunoștințele dobândite deja pe parcursul anilor, față de care sunt familiarizat.

Am descoperit tot ceea ce căutam în C#, un limbaj de programare orientat pe obiect, cu o sintaxă asemănătoare celei din Java, și .NET, o platformă care oferă suport pentru acest limbaj, alături de o comunitate vocală și implicată activ în dezvoltarea sa. Astfel, .NET oferă diverse resurse și biblioteci care facilitează conexiunea cu baza de date, criptarea informațiilor sensibile ale utilizatorilor, convertirea automată de la un obiect la alt obiect, crearea validatorilor, citirea și încărcarea în memorie a documentelor PDF, etc. De asemenea, .NET asigură autentificarea prin JWT Bearer, oferă servicii de LLM (Ollama) și suport pentru dezvoltarea de API-uri RESTful, toate acestea fiind esențiale pentru dezvoltarea aplicației create de mine.

## Structura arhitecturală – Clean Architecture

În căutarea unei arhitecturi moderne și care să reprezinte cât mai bine stilul meu de lucru și scopul aplicației, am descoperit **Clean Architecture** (1). Această arhitectură se remarcă prin organizarea elementelor pe straturi, având scopul de a produce un sistem testabil, independent de *framework*, interfață, bază de date sau orice alți agenți externi. Astfel, aplicația devine scalabilă, flexibilă și ușor de întreținut.

Toate cele menționate mai sus sunt puse în evidență de principiul separării îndatoririlor. Implementarea acestei arhitecturi mă ajută să stabilesc reguli, unde diferite părți ale aplicației mele au responsabilități bine definite și sunt independente unele de altele. De exemplu, în cazul în care îmi doresc să schimb baza de date cu care lucrez sau să adaug o nouă bază de date în proiect, este suficient să mă folosesc de mecanismul de *Dependency Injection*[[1]](#footnote-1) oferit de .NET, fără a modifica codul sursă sau a schimba întreaga implementare a aplicației.

Acest mecanism este esențial pentru implementarea mea, deoarece îmi permite decuplarea aplicației în 4 straturi principale:

* Domain – unde se află logica de *business* și entitățile aplicației
* Application – un strat intermediar între domeniu și interfață, specifică modul în care circulă datele (prin interfețe) și definește logica aplicației (*use cases*)
* Infrastructure – se ocupa de comunicarea cu baza de date oferind servicii pentru a extrage sau salva date
* Web API (Presentation) – expune controllere API și reprezintă interfața de comunicare cu aplicația frontend folosită de utilizator

Astfel, această decuplare mă ajută să lucrez mai eficient, oferindu-mi o structură clară pe care o pot urmări și peste care pot construi noi funcționalități independente, care să nu necesite o rescriere a codului deja existent.

## RESTful API

Am ales această metodă de comunicare deoarece HTTP/REST (*Representational State Transfer*) rămâne în continuare standardul pentru comunicarea sincronă. Datorită principiilor REST, pot crea o aplicație server *stateless*, structurată pe mai multe straturi și independentă de client.

### API-uri bazate pe controllere

ASP.NET Core oferă suport pentru două metode pentru crearea API-urilor: API-uri bazate pe controllere și API-uri minimale. Pentru aplicația mea, prima opțiune este mult mai potrivită deoarece mă ajută să respect stilul arhitectural impus de *Clean Architecture*. Astfel, pot crea controllere(clase) pentru a servi într-un mod clar și structurat, fiecare serviciu expus de către aplicație. De asemenea, aceste clase se pot folosi de mecanismul de *dependency injection* prin *constructor injection* sau *property injection*.

În mod similar cu Spring Boot, ASP.NET Core utilizează **atribute** pentru a configura comportamentul unui controller sau tipul de cerere HTTP al unei metode ,,de acțiune”. Un aspect important al acestor clase controller este modul în care ele sunt activate, iar apoi distruse în funcție de cerere. Acest lucru este esențial pentru mecanismul de *binding source parameter inference[[2]](#footnote-2)* din .NET. În stratul *Application* din aplicația server, am definit un set de **comenzi și interogări** sub formă de clase, ale căror atribute conțin informații ce trebuie prelucrate de către server. Atunci când serverul primește o cerere HTTP, obiectul care deservește comanda este creat, iar serviciul asociat comenzii respective este executat prin intermediul bibliotecii **MediatR**. După finalizarea procesării, obiectul este eliberat. Voi detalia utilizarea bibliotecii MediatR în capitolele următoare.

### Documentarea API-ului – Swagger

**Swagger** este o specificație independentă de limbajul de programare, utilizată pentru a documenta API-uri. În ziua de astăzi, documentația este esențială pentru companiile care, prin aplicațiile lor, expun API-uri pentru a putea fi consumate și integrate de către clienți.

,,*A survey by* ***ProgrammableWeb*** *found that API consumers consider complete and accurate documentation as the biggest factor in their API decision making, even outweighing price and API performance.*” (2)

Am decis să implementez Swagger în aplicația mea deoarece este mai eficient din punct de vedere al dezvoltării aplicației și îmi oferă o viziune clară de ansamblu asupra API-urilor expuse, inclusiv scopul și categoria din care fac parte. De asemenea, documentația Swagger permite testarea rapidă a fiecărui *endpoint* pentru a verifica corectitudinea acestuia. În ASP.NET Core, biblioteca **Swashbuckle** ne ajută să generăm automat documentația Swagger, pe baza atributelor din controller, sub forma unui web UI. În plus, Swagger UI oferă suport pentru securizarea API-urilor folosind *bearer token*.

Astfel, pot dezvolta și valida API *endpoints* pentru a realiza o comunicare sigură și corectă între componentele aplicației.

## Baza de date (SQLite)

## Securitate

## Alte servicii

# Implementarea algoritmilor ML – Serverul Python

# Aplicația Frontend – Interfața utilizatorului

# Idei pentru viitor și concluzii

Din cauza constrângerilor de timp, multe idei sau încercări de implementare au trebuit să fie abandonate, dar alături de implementările menționate mai sus care permit scalabilitatea, ele constituie un bun plan pentru adaptarea proiectului și continuarea acestuia în viitor.

## Idei și funcționalități potențiale

1. Crearea unui sistem de progresie pe nivele
   * Utilizatorii vor putea parcurge lecțiile câștigând experiență și deblocând recompense
   * Aceste recompense pot varia, de la schimbarea fundalului aplicației la diferite efecte și animații pentru butoane.
   * Introducerea unui magazin unde utilizatorii pot cumpăra aceste efecte
2. Un clasament în care utilizatorii înregistrați în aplicație pot să-și urmărească progresul personal în comparație cu cel al celorlalți studenți.
3. Opțiunea de a putea contribui și crea lecții

* Utilizatorii vor putea semnala unui administrator diversele greșeli care pot apărea în timpul redactării lecțiilor prin serviciul de email pus la dispoziție în aplicație.
* De asemenea aplicație poate dispune de o secțiune de *workshop* unde utilizatorii vor putea propune lecții noi care pot fi apoi revizuite și aprobate pentru a fi adăugate la resursele principale.

1. Partajarea rezultatelor între utilizatori

* Tot la capitolul de *workshop* utilizatorii pot încărca propriile modele antrenate pentru a putea fi folosite de alți utilizatori în rezolvarea de exerciții.

1. Un grad ridicat de personalizare

* Implementarea cu ajutorul inteligenței artificiale unui sistem de schimbare a dificultății de înțelegere a informației, bazat pe nivelul utilizatorului.
* În acest sens, un utilizator mai puțin familiar cu termenii grei din acest domeniu va putea primi o versiune mai ușor de înțeles a lecției. Trecând prin aceasta, el va câștiga experiență și va debloca noi conținuturi și exprimări științifice.
* Evident această implementare va fi ca nivel de opțiune pe care utilizatorul o va putea activa/dezactiva oricând pentru a permite persoanelor noi dar cu experiență în domeniu să beneficieze de experiența completă oferită de aplicație.

1. **Extinderea spre alte domenii**

* În final dar poate una dintre cele mai interesante idei este de a extinde aplicația spre alte domenii studiate în facultate.
* Chiar în timp ce învățam pentru examene mi-am pus întrebarea: *cum ar fi să existe un site similar W3Schools dar pentru toate materiile din facultate?* Astfel, ideea acestei aplicații poate fi extinsă și să adapteze conținutul mai multor cursuri, în special celor care au și caracter aplicativ, nu doar teoretic.
* Această aplicație ar putea dispune de funcțiile descrise mai sus și ar constitui un mediu mai digitalizat în care studenții își pot găsi motivația pentru studiu, pot comunica cu alți studenți și profesori pentru a rezolva probleme, sau pot propune noi idei și subiecte de discuție pe care să le abordeze cu profesorii la curs sau chiar în aplicație.
* Într-adevăr, această idee este dificil de pus în practică și necesită multe resurse și timp. Cu toate acestea, într-o lume în care inteligența artificială domină piața și este strâns legată de viața unui student, consider că există și o metodă în care aceasta poate aduce beneficii, atunci când este folosită creativ, într-un mod educațional.

## Concluzii finale

**Bibliografie**

1. **Tekin, Semih.** What is the Clean Architecture? 4 August 2023.

2. **Swagger.** API Documentation Made Easy with OpenAPI & Swagger.

3. *ASP.NET CORE Documentation (.NET 9.0).* **Microsoft.** s.l. : Microsoft, 2024.

1. **Dependency Injection** (3)este un mecanism în .NET, ce are ca scop principal asigurarea principiului de ***dependency inversion***. Astfel, se creează un nivel înalt de abstractizare oferind aplicației ușurință în întreținere și scalabilitate. [↑](#footnote-ref-1)
2. *Binding source parameter inference* (3)- definește ce parte/părți din cerea HTTP vor fi preluate drept parametri într-o metodă din controller. De exemplu, atributul [FromQuery] atașat unui parametru al metodei specifică faptul că valoarea este preluată din *query string-ul* URL-ului și legată de parametrul respectiv. De asemenea, cuvântul „*inference*” se referă la faptul că sursa potrivită pentru parametru va fi dedusă automat. [↑](#footnote-ref-2)