

UTILIZAREA I.A. ÎN INGINERIA PROGRAMELOR SOFTWARE

*analiză privind oportunitățile oferite de dezvoltarea inteligenței artificiale
în ingineria dezvoltării software în general și a aplicațiilor web în special*

*Tomasciuc Gheorghe Bogdan, Universitatea “Ștefan cel Mare”, Suceava
Paranici Ștefan Iulian, Universitatea “Ștefan cel Mare”, Suceava*

Rezumat

Utilizarea inteligenței artificiale (I.A.) devine o parte din ce în ce mai importantă în majoritatea activităților noastre – de la industriile puternic bazate pe tehnologie până la ramuri care sunt mult mai puțin influențate de dezvoltările din zona IT. cum ar fi industria muzicală, arhitectura sau designul. Cu atât mai mult, dezvoltarea inteligenței artificiale și ubicuitatea de care se bucură la această dată afectează într-o manieră considerabilă industriile care folosesc în mod intrinsec tehnologiile curente și se bazează pe cele mai recente descoperiri din domeniu. În această lucrare vom face cunoștință cu conceptele discutate cu ajutorul lucrărilor în domeniu și prin propriile prisme, vom trece în revistă cele mai cunoscute „avataruri” ale inteligenței artificiale și vom vedea cum acestea ne ajută în mod concret cu optimizarea timpilor petrecuți cu proiectarea și dezvoltarea efectivă a unui proiect practic de aplicație web. În final vom încerca să tragem niște concluzii legate de acest subiect și de direcții viitoare de abordare a acestei chestiuni.

Introducere în inteligența artificială

Inteligența artificială este un gând care i-a fascinat pe oameni încă din timpuri imemorabile. Există multiple mituri, povești și istorisiri în carea au apărut de-a lungul timpului legături de această fascinație a oamenilor legate de lucruri construite care primesc într-o mai mare sau mai mică măsură caracteristici pe care acum le atribuim conceptului de inteligență artificială. [1] Exemple ale acestei fascinații pentru construcții dotate cu inteligență întâlnim în povești precum “Vrăjitorul din Oz” prin personajul omului de tinichea [2] sau în personaje mitice precum “uriașul din Talos” construit de Hefaistos și însărcinat cu protejarea insulei Creta sau cu statuile sculptate de Pymalion. De asemenea, în istorisiri vechi și

legende alchemice există tot felul de “rețete” pentru crearea de golemi însuflețiți creați din pământ sau alte materiale anorganice. În epoca modernă facinația precursorilor pentru acest subiect s-a mutat în avataruri moderne care ar putea fi gazde pentru inteligența artificială cum ar fi automatonii sau mașini (sisteme de calcul) capabile să gândească sau cel puțin să ofere răspunsuri într-o manieră care nu ne permite să distingem între raționalizare și algoritmi computaționali folosiți. [1]

Inteligența artificială și știința calculatoarelor

Mai departe ne vom concentra pe ce cunoaștem acum ca și inteligență artificială în domeniul științei calculatoarelor. Acest concept, deși a apărut cu multe decade în urmă, nu a cunoscut un avânt puternic în dezvoltare până la începutul anilor 2020. Conceptul inițial a fost propus de către Alan Turing inițial în anul 1941 și ulterior în articolul său “Computing Machinery and Intelligence” publicat în revista Mind în anul 1950. [3] Acest articol avea să devină piatra de temelie în dezvoltarea inteligenței artificiale. Într-un mod inovator, Turing renunță în acest articol la efortul de a demonstra dacă un sistem informatic (computer) gândește sau nu și se întreabă dacă putem distinge acest lucru doar analizând rezultatele obținute de la acesta fără să știm dacă rezultatele pe care le avem provin de la un calculator sau de la o persoană. Desigur, articolul abordează subiectul într-o manieră mult mai detaliată și convingătoare însă ideea de bază care se poate desprinde este că, având acces la suficiente informații care să constituie o pseudo experiență anterioară, s-ar putea construi, în mod teoretic, un creier artificial asemănător din punct de vedere al funcționării cu creierul uman însă mult mai eficient – desigur în aria în care a fost construit. Pe baza acestui mod de abordare s-au

pus mai târziu bazele rețelelor neuronale cu ajutorul cărora s-au construit primele modele de învățare asistată, așa-numitele mașini Perceptron construite de către Frank Rosenblatt. [4] Următorul pas firesc pentru a micșora distanța dintre om și mașină și apropierea, astfel, de imaginea ideală de interacțiune dintre om și mașină imaginată de către grecii antici a fost introducerea conceptului de limbaj natural adică introducerea algoritmilor prin care un sistem informatic să decodeze limbajul natural uman și să poată furniza răspunsuri adecvate pe baza întrebărilor sau a instrucțiunilor primite în această manieră. Pe baza descoperirilor în această zonă de cercetare s-au creat initial primii chatboți precum ELIZA [5] și, în perioada contemporană, platformele de inteligență artificială precum ChatGPT, Google Bard (acum Gemini), Claude, ș. a. m. d..

Toate aceste descoperiri au necesitat ani de dezvoltare și resurse considerabile investite în dezvoltarea acestor tehnologii. Care ar fi, așadar, scopul principal al acestor eforturi considerabile? Într-un naiv, după cum ar putea considera unii, cred că scopul principal al acestor dezvoltări, cel puțin din perspectiva cercetătorilor care le-au condus, a fost îmbunătățirea cunoașterii în aceste domenii astfel încât, pe viitor, să devenim mai buni și mai eficienți. Este, de fapt, recunoașterea limitelor biologice ale noastre ca oameni, dar dorința vie de a le depăși, de a vedea ce este dincolo de limitările noastre, de a întrezări orizonturi noi pe care nu le putem observa cu setul nostru limitat de simțuri și cu capacitatea noastră de concentrare, analiză și sinteză. Desigur că fiecare reprezentant al scenei sociale vede în dezvoltarea inteligenței artificiale beneficiile pe care și le dorește în propriul domeniu și, privind la sursele finanțării, există multe aplicații comerciale sau militare pentru inteligența artificială. Se spune că empatia și etica sunt unele dintre barierele pe care oamenii trebuie să le depășească pentru a putea privi unele probleme într-un mod obiectiv și a ajunge la rezultate optime. Spre exemplu problema tramvaiului sau alte probleme similare nu ar pune presiune psihologică asupra unei inteligențe artificiale care ar privi problema într-un mod total detașat și, pe baza datelor de intrare ar produce un rezultat

optim în concordanță cu scopul urmărit. Suntem oare, pregătiți, însă, să acceptăm astfel de soluții? Viteza și specificitatea cu care algoritmii de inteligență artificială produc rezultate fac pe mulți dintre noi să înțelegem că inteligența artificială a devenit o parte integrantă din viața noastră și pe viitor va deveni la fel de prezentă precum telefoanele mobile sunt acum. Utilizare, așadar a inteligenței artificiale ar aduce beneficii majore în ingineria software și în acest moment suntem pregătiți să absorbim și să acceptăm aceste beneficii oferite de inteligența artificială. Acesta este folosită sau se planifică introducerea în aproape toate stadiile de dezvoltare software: de la planificare, analiza riscului și dezvoltare până la operațiunile periodice de mentenanță, debugging și feature support. [6] Iar acest lucru se poate deduce din frecvența cu care termenul de inteligență artificială în corelație cu diferiți alți termeni sunt folosiți în diferite lucrări de la începutul secolului 21. Se poate observa așadar, în urma unei astfel de analize, că utilizarea I.A. în ingineria software reprezintă o preocupare din ce în ce mai mare pentru cercetătorii în acest domeniu. În acest sens se observă o tendință emergentă de a schimba întrebarea “Cum programăm un PC?” către întrebarea care ar avea un răspuns mai avantajos “Cum programăm un PC să se programeze pe sine?”. [7] Răspunsul la această întrebare ar avea avantaje evidente în domeniul ingineriei software întrucât s-ar baza pe principiile ingineriei software în toate fazele dezvoltării, principii care ar fi fost introduse în faza de antrenare și pe care inteligența artificială ar putea să le sintetizeze în soluții viabile într-un timp mult mai mic și cu costuri mult mai mici decât o echipă constituită în acest scop.

Ingineria software

Ingineria software reprezintă aplicarea metodelor ingineresti asupra stagiilor de planificare, dezvoltare și mentenanță a soluțiilor software. Acest concept a apărut ca și consecință a observării unei deteriorări continue și agravate a programelor software. [8] O mare parte a programelor pentru calculator sufereau de probleme grave cum ar fi: nelivrarea la timp a produsului, creșterea necontrolată a costurilor de dezvoltare, erori grave în operare sau chiar, în

unele cazuri, livrarea chiar a unui produs nefuncțional. În acest context s-a cristalizat nevoia dezvoltării unui sistem standardizat de dezvoltare a produselor software astfel încât să se depășească aceste probleme și să se obțină, astfel, produse software de calitate și la timp. Câteva definiții notabile (en.) ale acestui concept sunt date mai jos:

"The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software"

—The Bureau of Labor Statistics—IEEE Systems and software engineering – Vocabulary

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"—IEEE Standard Glossary of Software Engineering Terminology [9]

Scopul fundamental al ingineriei software este, așadar, de a furniza produse software (livrabile) de calitate și la timp. În atingerea acestui scop ingineria software conturează câteva obiective pe care produsul final trebuie să le atingă:

- *fiabilitate*, adică abilitatea de a își îndeplini funcțiile pentru o perioadă indefinită de timp;
- *reutilizarea*, posibilitatea de a fi refolosit în tot sau în parte;
- *servisabilitate*, posibilitatea de a putea fi îmbunătățit sau adaptat cu ușurință la cerințele actuale;
- *testabilitate*, posibilitatea de a testa și a evalua performanțele software-ului în funcție de anumite criterii;
- *corectitudine*, determinabilă în funcție de gradul de îndeplinire a caracteristicilor detaliate în caietul de sarcini;
- *adaptabilitate*, soluția software ar trebui să poată fi adaptată cu un minim de efort la nevoile utilizatorului;
- *portabilitate*, soluția software ar trebui să poată fi instalată pe multiple sisteme sau să poată fi transferată de pe un sistem pe altul. [8]

Pentru a putea atinge aceste obiective s-au conturat, de-a lungul timpului diferite metode de abordare a tehnicilor de dezvoltare și s-au propus diferite variante pentru a obține în el mai eficient

mod soluții software de calitate. Printre aceste soluții amintim tehnicile *waterfall*, *spiral*, *iterative*, *incremental* sau *agile*. Nu intrăm în detaliile acestora întrucât există resurse foarte bune care explorează în mod detaliat beneficiile și limitările sau neajunsurile fiecărei metode.

Starea actuala

În ultimii ani s-a observat o dezvoltare accentuată a utilizării inteligenței artificiale în aproape toate ramurile industriei. Există modele de inteligență artificială modelată după diferite arii de implementare: de la întrebări generale până la întrebări foarte specifice din anumite industrii (ex. medicale sau industriale).

Atributele caracteristice ale inteligenței artificiale, capacitatea sa superioară de a analiza seturi mari de date, folosindu-se de acestea într-un mod similar în care oamenii se bazează pe experiența anterioară, capacitatea extraordinară de analiză și sinteză a datelor și de a le extrapola pentru a furniza un răspuns într-o fracțiune din timpul necesar unei echipe sunt motivele pentru care inteligența artificială s-a integrat cu succes în arii precum: jocurile pe calculator, procesarea limbajului natural, recunoașterea formelor și a șabloanelor, sisteme experte (în diferite arii), sisteme de recunoaștere automată vizuală, recunoașterea vorbirii, roboți inteligenți. [7]

Așadar putem spune cu certitudine că beneficiile aduse de IA industriilor în care sunt utilizate înclină balanța atât de mult în favoarea sa încât ne așteptăm ca prezența IA în viața noastră se va accentua pe măsură ce crește rata de adopție a acestei tehnologii în diferitele ramuri ale industriei.

LLM-urile (Large Language Models) sunt cele mai reprezentative implementări ale inteligenței artificiale și cele care s-au dovedit a avea cea mai puternică atracție în adopție. Acestea se bazează pe repoziitoare uriașe de informații și pe vitezele de procesare mari pe care le oferă posibilitățile actuale de procesare și stocare oferite de clusterelor de calculatoare. [10] Acesta este, probabil, și motivul pentru care tehnologia aceasta a avut nevoie de atât de mult timp pentru a lua avânt. Nu am avut posibilitatea de a stoca și utiliza într-un mod eficient cantitățile de informații de care era nevoie pentru a crea un

astfel de mediu în care să antrenăm inteligența artificială. Însă odată cu creșterea acestor capacități – conform legii lui Moore – a apărut o explozie în dezvoltarea LLM. După cum se poate vedea în figura 1 de mai jos, nu numai că s-au dezvoltat multiple astfel de tehnologii în ultimii ani – evidențiate cu buline gri – dar liderii înregistrează, de asemenea, performanțe din ce în ce mai bune în generarea de cod.

răspunsuri în altă limbă decât cea în care are, probabil, materialele de antrenare la dispoziție.

Prompturile furnizate conțin întrebări specifice legate de tehnicile legate de ingineria software și de dezvoltarea concretă a unui proiect în vederea creării unei aplicații web de generare adevărate de student.

Modalitatea de evaluare a performanțelor fiecărui LLM va fi realizată în funcție de punctaje între 1 și 10 după următoarele criterii:

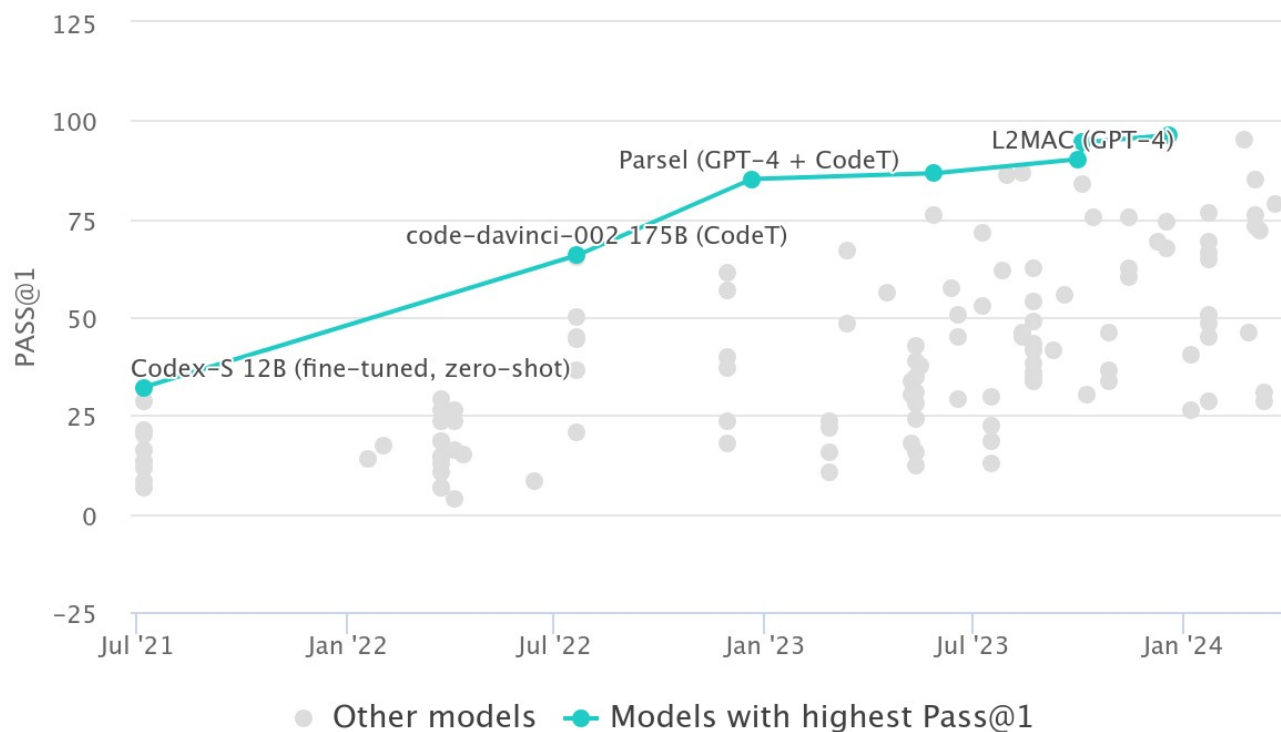


Figura 1: Lideri LLM în generarea de cod (imagine generată pe <https://paperswithcode.com/sota/code-generation-on-humaneval> și preluată în Aprilie 2024)

Modalitate analiza

Pentru raportul curent privind utilizarea IA în ingineria software am ales trei reprezentanți ai celor mai răspândite manifestări ale inteligenței artificiale sub forma LLM la care putem avea acces în mod liber. Acestea sunt:

- ChatGPT – versiunea 3.5;
- Gemini (Google Bard);
- Copilot (Microsoft).

Prompt-urile furnizate au fost construite în limba română, în acest fel putând testa, pe lângă capacitatea de a genera cod și capacitatea de a traduce și înțelege cerințe și de a genera

- ✓ înțelegerea întrebării din contextul oferit și oferirea unui răspuns fără a fi necesară introducerea de prompt-uri suplimentare pentru clarificarea sarcinii;
- ✓ furnizarea unui răspuns clar și specific întrebării (negeneral);
- ✓ adăugarea de comentarii sau de blocuri de text pentru explicarea codului generat;
- ✓ ușurința în utilizare;
- ✓ disponibilitate.

Lucrare practica

În realizarea acestui experiment am încărcat ca și prompturi următoarele texte pentru a testa capacitățile fiecărui LLM de a acționa în sprijinul

inginerilor software. Detaliile prompturilor se regăsește în Anexa A din documentul Anexe atașat.

1. Cunoști principiile de inginerie software și cum se aplică acestea în cadrul proiectelor de dezvoltare software a aplicațiilor web? (ANEXA 1: Q-1)

Răspuns ChatGPT (Scor 3/5)

Răspuns Gemini (Scor 5/5)

Răspuns Copilot (4/5)

2. Având în vedere principiile anterioare care sunt pașii pe care i-ai parcurge pentru a dezvolta o aplicație web cu următoarele cerințe? Aplicația va fi folosită de trei categorii de utilizatori: studenți, secretar și administrator. În afară de utilizatorii cu rol administrator care se vor conecta pe bază de utilizator și parolă, ceilalți utilizatori se vor conecta cu Google Sign In. Adresele de email ale utilizatorilor vor fi încărcate la instalarea aplicației de către administrator pe baza unui fișier Excel. Acesta va putea, din lista de adrese să desemneze care utilizatori vor îndeplini ulterior rolul de secretar sau secretară din lista de utilizatori. Utilizatorii de tip student vor înregistra cereri de eliberare adeverințe prin completarea unui formular Google unde vor introduce doar motivul cererii și adresa de email. Informațiile din formular vor fi, ulterior, consultate de către aplicație și încărcate într-o bază de date separată. Utilizatorii cu rol secretar se vor conecta tot pe baza Google SignIn însă la o interfață proprie în care vor vedea o listă cu cererile înregistrate de studenți și va putea aproba sau respinge cererea. În cazul respingerii cererii va trebui să furnizeze un motiv. Administratorii vor avea, și ei, acces la interfața secretarului și, în plus, vor putea să introducă detaliile de identificare ale facultății, să introducă fișierul Excel pentru lista inițială de utilizatori și să realizeze operațiuni de salvare a bazei de date și de reinițializare a întregului sistem.

(ANEXA 1: Q-2)

Răspuns ChatGPT (4/5)

Răspuns Gemini (5/5)

Răspuns Copilot (4/5)

3. Pentru dezvoltarea aplicației am decis să folosesc următoarele tehnologii: Google pentru autentificarea utilizatorilor non-administratori conform cerințelor, Google Forms pentru preluarea cererilor în baza de date a aplicației, NextJS, NestJS, Sqlite3 și TypeORM. În acest scenariu poți crea codul necesar care să fie folosit la autentificarea utilizatorilor de tip secretar știind că Google ca autentifica utilizatorul pe baza adresei de email iar în baza de date a aplicației vom avea o coloană rol în tabelul cu utilizatori? (ANEXA 1: Q-3)

Răspuns ChatGPT (3/5)

Răspuns Gemini (4/5)

Răspuns Copilot (5/5)

4. Folosind aceleași tehnologii menționate mai sus scrie un cod pentru o rutină automată prin care, la final de zi, la ora 17:00 toți utilizatorii cu rol administrator să primească un raport Excel cu toate adeverințele emise în ziua respectivă. Coloanele sunt: numele și prenumele solicitantului, adresa de email, motivul cererii, stare: dacă a fost aprobată sau nu, numărul cererii: în cazul în care fost aprobată sau textul „respinsă” dacă a fost respinsă și motivul. (ANEXA 1: Q-4)

Răspuns ChatGPT (5/5):

Răspuns Gemini (4/5):

Răspuns Copilot (4/5 – a pierdut contextul):

5. În contextul acestui proiect ce sugestii poți aduce pentru a crește productivitatea activității utilizatorilor cu rol de secretar și a face experiența de lucru cât mai plăcută? (ANEXA 1: Q-5)

Răspuns ChatGPT (5/5):

Răspuns Gemini (5/5):

Răspuns Copilot (3/5):

Medii scor raspunsuri

La acest moment devenit evident ca instrumentele testate obțin în mod consecvent scoruri mari și apropiate și nu mai are sens notarea răspunsurilor. Mediile scorurilor obținute până acum:

ChatGPT 3.5: 22/25 = 88%

Gemini: 23/25 = 92%

Copilot: 20/25 = 80%

Având în vedere aceste rezultate preliminare putem să spunem cu certitudine că IAG are deja capacitatea de a contribui într-un mod extrem de folositor în dezvoltarea produselor software începând cu cele mai incipiente stadii ale ingineriei produsului și până la finalul ciclului de viață al produsului.

În continuare vom da exemple de întrebări și răspunsuri concrete cu care IAG au contribuit în dezvoltarea unei soluții software în diferite stagii. Răspunsurile IAG for fi prezentare sub formă de imagini trunchiate. Răspunsul complet la întrebările următoare poate fi găsit în ANEXA1 pentru a fi studiat în întregime. Am adresat întrebări atât în engleză cât și în limba română.

IAG în faza de analiză a proiectului:

- analiza cerințelor (ANEXA 1: Q-6)

~v03.03- Caiet de sarcini TWAAOS-SIC-... PDF

Aici sunt requirement-urile unei aplicații pentru un proiect la facultate. Fa-mi o lista pe scurt cu tot ce ar trebui sa faca aplicatia.

Lista cerințelor pentru aplicația "Sistem de generare a adeverințelor electronice pentru studenți"

1. Prezentare generală

- Obiective:**
 - Simplificarea procesului de obținere a adeverințelor pentru studenți.
 - Reducerea timpului necesar eliberării adeverințelor.
 - Asigurarea corectitudinii adeverințelor generate.
- Funcționalități generale:**
 - Generarea adeverinței la primirea unei cereri electronice.
 - Validarea adeverinței de către secretariat.
 - Informarea studentului despre starea cererii prin email.
 - Alocarea unui număr de înregistrare unic adeverinței.
 - Generarea de rapoarte zilnice sau lunare privind adeverințele emise [?].

2. Categoriile de utilizatori ai aplicației

2.1. Rol: Student

- Acces aplicație:**
 - Autentificare prin "Sign In With Google" cu adresa de email asociată facultății

- conturarea obiectivelor majore (inclusiv pentru colaborarea cu alte unelte) (ANEXA 1: Q-7)

Fa-mi o lista cu checkboxes, ca sa o pot pune in notion

Lista cerințelor pentru aplicația "Sistem de generare a adeverințelor electronice pentru studenți"

1. Prezentare generală

- ☐ Simplificarea procesului de obținere a adeverințelor pentru studenți.
- ☐ Reducerea timpului necesar eliberării adeverințelor.
- ☐ Asigurarea corectitudinii adeverințelor generate.
- ☐ Generarea adeverinței la primirea unei cereri electronice.
- ☐ Validarea adeverinței de către secretariat.
- ☐ Informarea studentului despre starea cererii prin email.
- ☐ Alocarea unui număr de înregistrare unic adeverinței.

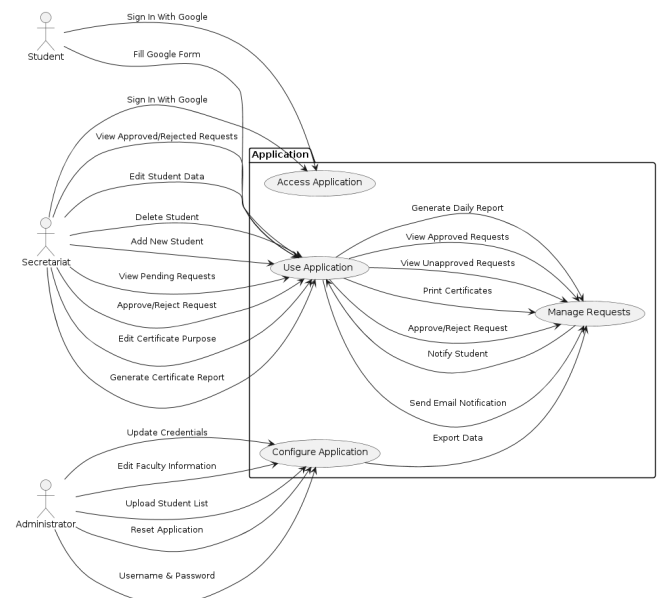
- identificarea utilizatorilor și crearea diagramei de utilizatori prin generarea codului UML necesar (ANEXA 1: Q-8)

To create the UML diagram for the users involved in the Next.js and Nest.js project described, we'll first identify the actors and their relationships. Here's a brief summary of the roles:

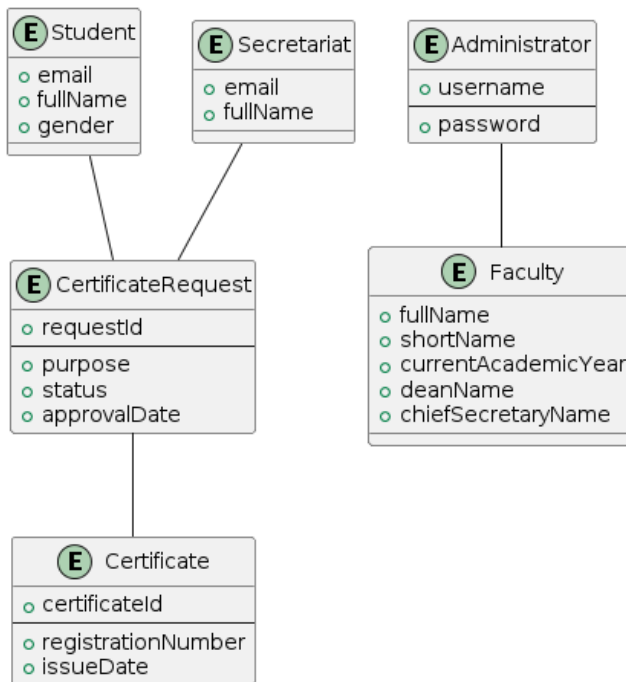
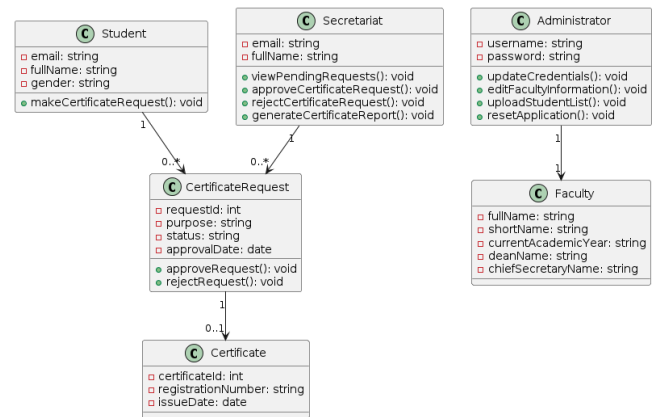
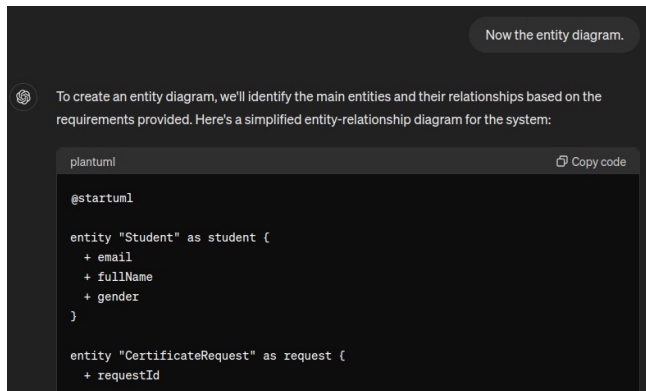
- Student:** Interacts with the application to request certificates.
- Secretariat:** Manages the approval and processing of student certificate requests.
- Administrator:** Oversees the configuration and maintenance of the application.

Now, let's represent these roles and their interactions in a UML diagram using PlantUML syntax:

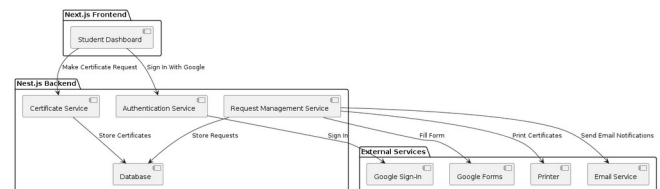
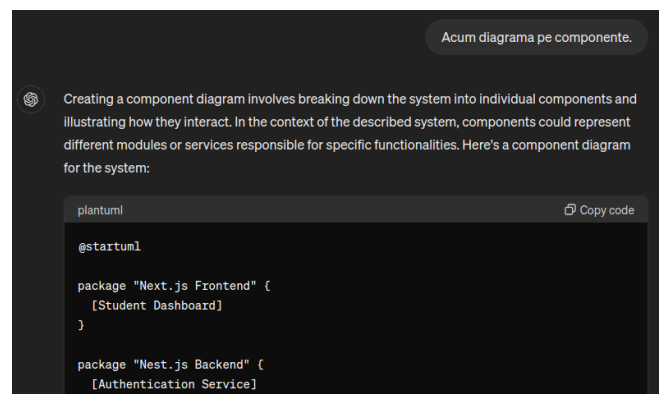
```
plantuml
@startuml
left to right direction
actor Student as S
actor Secretariat as Sec
actor Administrator as Admin
```



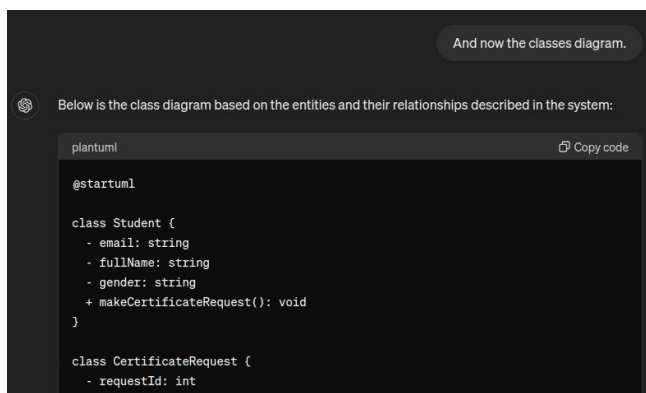
- identificarea și crearea diagramei de entități:
(ANEXA 1: Q-9)



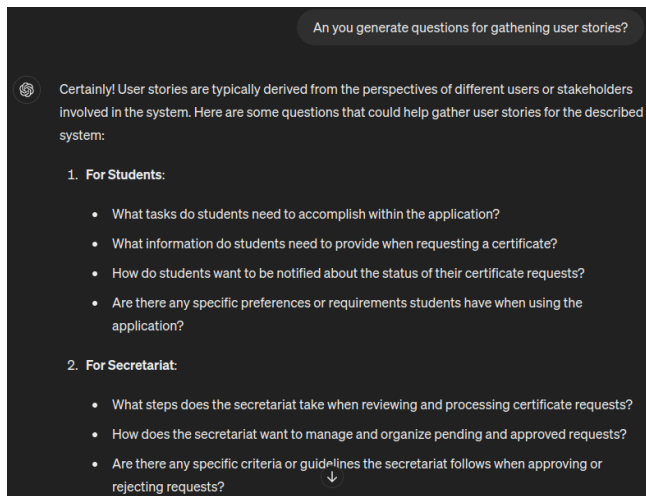
- identificarea și crearea diagramei de componente:
(ANEXA 1: Q-11)



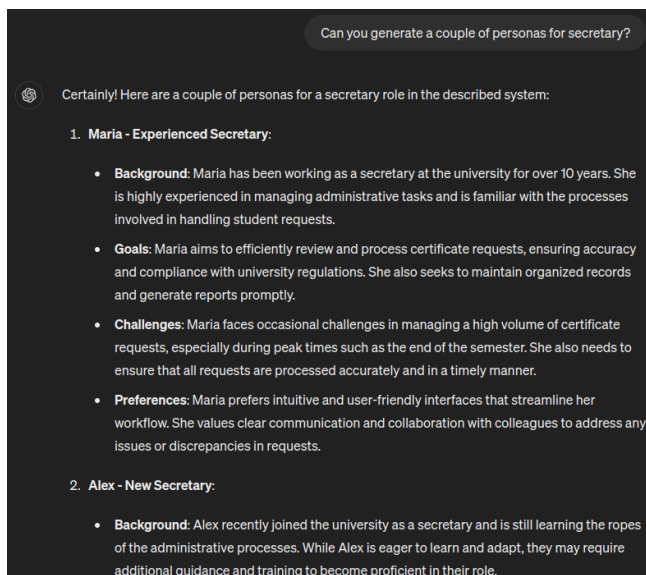
- identificarea și crearea diagramei de clase:
(ANEXA 1: Q-10)



- generarea întrebărilor pentru „user stories”: (ANEXA 1: Q-12)



- generarea de personas: (ANEXA 1: Q-13)



În mod similar s-au putut obține și alte diagrame necesare. În această fază IAG s-a comportat neașteptat de bine – a furnizat informații valoroase care au putut fi folosite cu unele adaptări în practică. Încă apar confuzii, încă se mai pierde contextul și nu suntem în situația în care furnizăm IA caietul de sarcini și acesta crează toată aplicația și tot fluxul fără intervenția unui inginer / programator și, probabil, nici nu ne dorim asta. Însă răspunsurile furnizate, în mâinile persoanei potrivite poate ajuta la creșterea vitezei cu care proiectul parcurge aceste etape – fiind

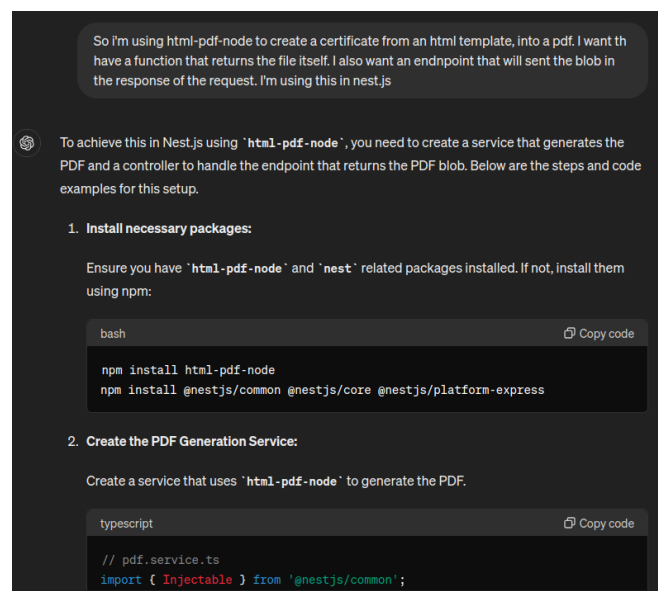
necesară mai mult ghidarea IA către răspunsul corect și mai puțină muncă brută din partea inginerului sau a programatorului.

În faza de dezvoltare

În faza de dezvoltare utilizarea IA este uneori atât de subtilă încât aproape nici nu ne mai dăm seama că o parte din codul scris vine de la IA. Ne referim aici, desigur la instrumentele de completare automată a codului în cadrul editoarelor folosite în dezvoltare și în special la Github Copilot integrat în VS Code. Dar resurse similare sunt și pentru majoritatea IDE de utilizare largă: Leo, Codeium, AI Assisant, etc. Pe lângă adresarea de întrebări concrete IAG referitor la probleme specifice precompletarea în mod inteligent a codului în timp ce îl scriem este de mare ajutor atunci când IA reușește să își dea seama ce bloc de cod dorim să scriem. Pe lângă scrierea mult mai rapidă a blocului de text – acesta se și folosește de obiceiuri considerate bune de scriere a codului.

Mai jos punem câteva exemple concrete de utilizare a IAG în dezvoltarea unor componente din cadrul proiectului.

- Generarea fișierului pdf cu conținutul adeverinței.



- Generarea pdf multipagină pentru a putea lista în bloc adeverințe

then i want an endpoint that will help me retrieve multiple requests, all concatenated in a single pdf

To create an endpoint that retrieves multiple requests and concatenates their responses into a single PDF, we need to perform the following steps:

1. Modify the `PdfService` to handle multiple HTML contents and concatenate them into a single PDF.
2. Create an endpoint in the `PdfController` that accepts multiple HTML contents, generates individual PDFs, and concatenates them.
3. Update the Nest.js application to support these changes.

Let's start with the implementation:

1. Modify the `PdfService` to Handle Multiple HTML Contents

First, update the `PdfService` to concatenate multiple PDFs into a single PDF.

```
typescript
// pdf.service.ts
import { Injectable } from '@nestjs/common';
import * as htmlPdf from 'html-pdf-node';
import * as PDFMerger from 'pdf-merger-js';
import { promises as fs } from 'fs';

@Injectable()
export class PdfService {
  async generatePdf(html: string): Promise<Buffer> {
    const file = { content: html };
    const pdfData = { content: '1' };
  }
}
```

- integrarea autentificării Google cu Node prin librăria Passport

how can i get the google client id

To obtain your Google Client ID for OAuth2 authentication, follow these steps:

Step 1: Go to Google Cloud Console

Navigate to the [Google Cloud Console](#).

Step 2: Create a New Project or Select an Existing One

1. Click on the project dropdown at the top of the page.
2. Click on "New Project" to create a new project or select an existing project from the list.

Step 3: Enable Google APIs

1. With your project selected, go to the [API & Services Dashboard](#).
2. Click on "Enable APIs and Services".
3. Search for "Google+ API" (or "Google Identity" if Google+ API is deprecated) and click on it.
4. Click on "Enable".

Step 4: Configure OAuth Consent Screen

1. Go to the [OAuth consent screen](#).
2. Select "External" and click "Create".
3. Fill out the required fields (App name, User support email, etc.).
4. Add your email address and the domains that will use the OAuth service.
5. Click "Save and Continue" through the steps, filling out any required information.

- asistență în depanarea problemelor de implementare;

Help me integrate google authentication in a nest.js app, using passport strategy

Integrating Google Authentication into a Nest.js application using Passport involves several steps. Below is a detailed guide on how to accomplish this:

Step 1: Install Required Packages

First, you need to install the necessary packages. Run the following command in your terminal:

```
bash
npm install @nestjs/passport passport passport-google-oauth20
```

Step 2: Create a Google Strategy

Create a file named `google.strategy.ts` in your authentication module directory:

```
typescript
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy, VerifyCallback } from 'passport-google-oauth20';

@Injectable()
export class GoogleStrategy extends PassportStrategy(Strategy, 'google') {
  constructor() {
    super({
      clientID: 'YOUR_GOOGLE_CLIENT_ID',
      clientSecret: 'YOUR_GOOGLE_CLIENT_SECRET',
      callbackURL: 'http://localhost:3000/auth/google/callback',
      scope: ['email', 'profile'],
    });
  }
}
```

- asistență în crearea cheilor de comunicare cu API Google: sumarizarea documentației și furnizarea de pași concreți pentru a obține cheia de comunicație necesară într-un mod descriptiv ușor de urmat.

I seem to have some problem with the callback

If you are encountering issues with the callback, there are several potential areas to check. Let's go through a comprehensive setup and troubleshooting guide to ensure everything is configured correctly.

Step-by-Step Setup and Troubleshooting Guide

1. Ensure Correct Callback URLs

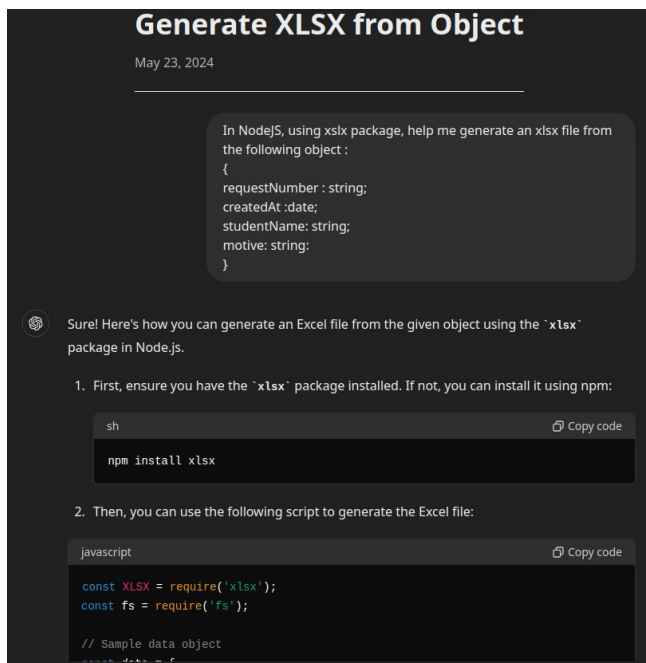
Verify that the callback URL configured in your Google Cloud Console matches the one in your Nest.js strategy setup.

- **Google Cloud Console:** The callback URL should be `http://localhost:3000/auth/google/callback`.
- **Nest.js Strategy:** Ensure the callback URL is set correctly in your `GoogleStrategy`:

```
typescript
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy, VerifyCallback } from 'passport-google-oauth20';

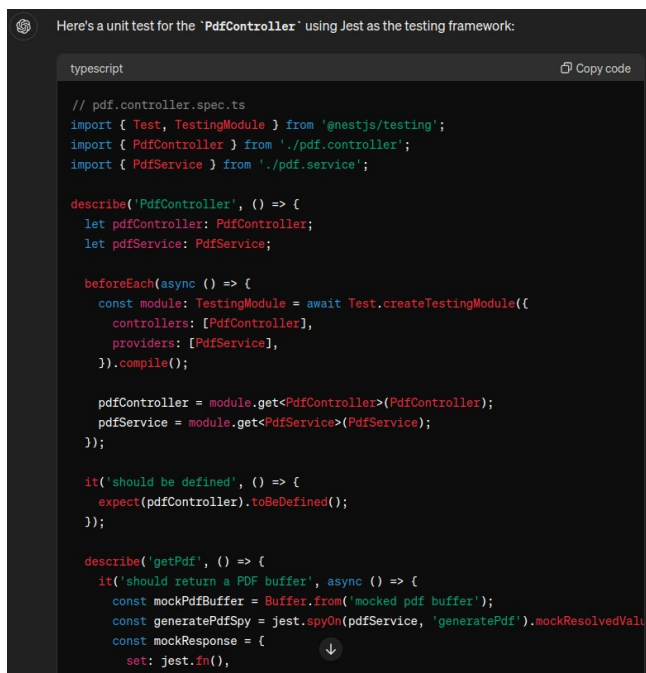
@Injectable()
export class GoogleStrategy extends PassportStrategy(Strategy, 'google') {
  constructor() {
    super({
      clientID: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
      callbackURL: 'http://localhost:3000/auth/google/callback',
      scope: ['email', 'profile'],
    });
  }
}
```

- exportarea datelor în format xlsx



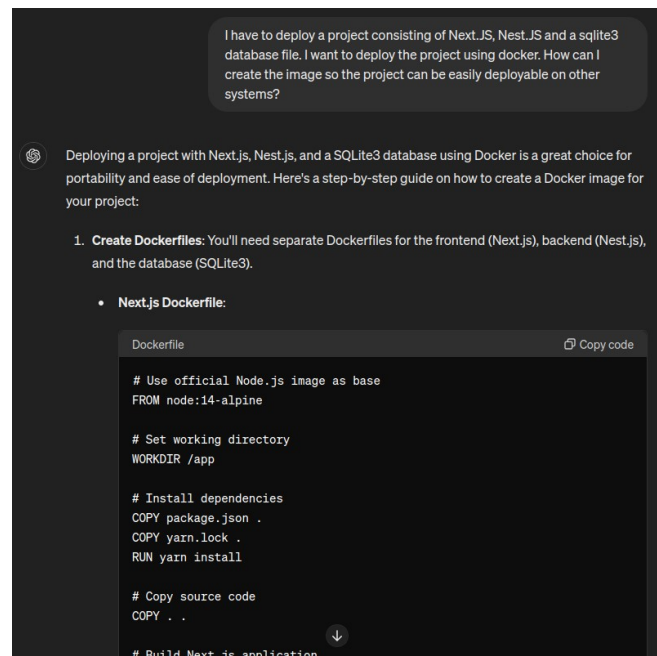
În faza de testare

- scrierea de cod pentru testarea unitară a codului generat folosind Jest

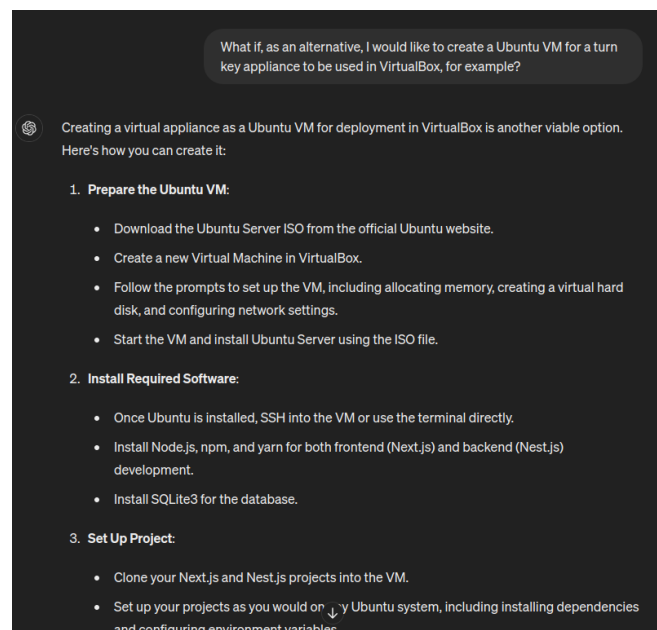


În faza de instalare

- instrucțiuni detaliate privind crearea unei imagini pentru instalarea ușoară a proiectului

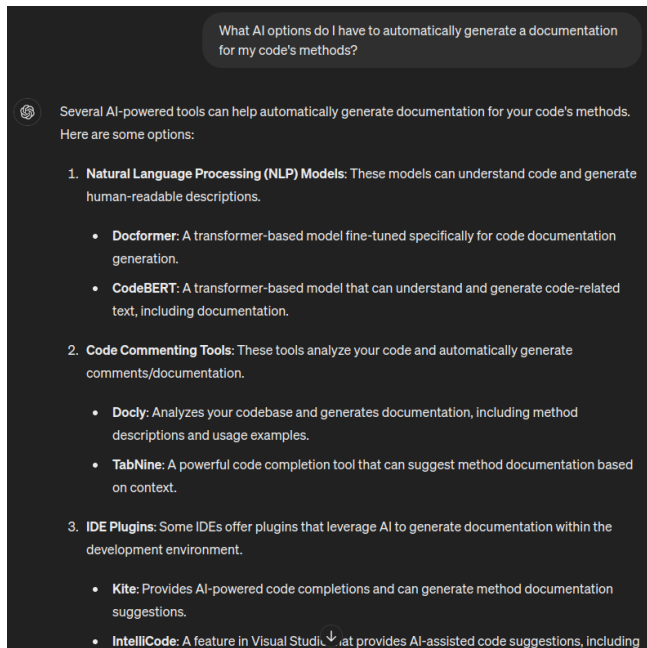


- variantă alternativă de distribuire a proiectului folosind o mașină virtuală cu Ubuntu

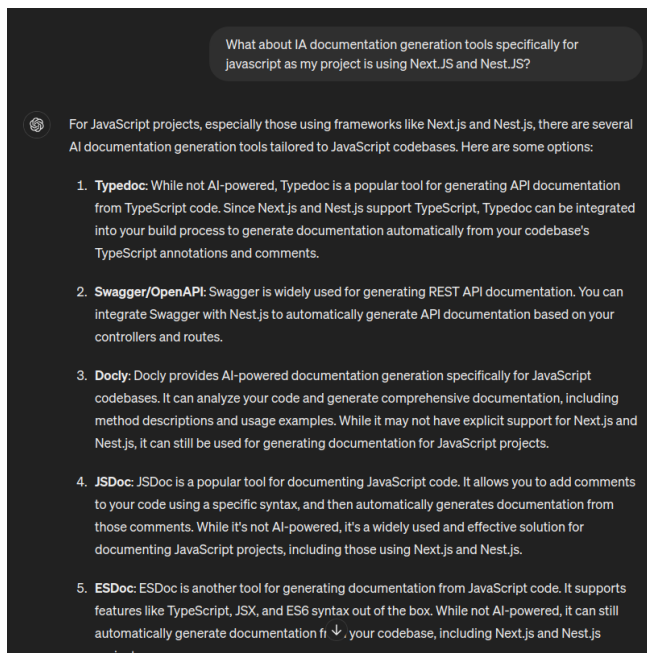


În faza de documentare a codului

- identificarea de unelte generale pentru generarea documentației codului proiectului



- cristalizarea rezultatelor prin refixarea contextului, respectiv prin specificarea clară a tehnologiilor folosite pentru care este necesară generarea documentației



Concluzii

În urma acestui experiment putem completa raportul acesta cu câteva concluzii legate de utilizarea inteligenței artificiale (în speță inteligența artificială generativă) în fluxurile

activităților de inginerie software. În primul rând apreciem că tehnologia s-a maturizat atât de mult încât răspunsurile au o rată de halucinație foarte mică atunci când contextul este bine stabilit de la început sau când contextul este clarificat prin prompt-uri suplimentare. Experiența în utilizarea soluțiilor este una foarte prietenoasă după cum se poate vedea în răspunsurile primite de la platformele testate.

De asemenea se poate vedea că răspunsurile au într-adevăr o valoare extraordinară în utilizare, atât în planificarea proiectului cât și în stagiile de dezvoltare efectivă a soluției prin furnizarea de cod funcțional și care respectă principii solide de dezvoltare.

Toate platformele testate au furnizat, după conturarea contextului, informații corecte și concrete într-o fracțiune din timpul necesar unei echipe de programare pentru a realiza secvențele necesare de cod. De asemenea, la o parte din întrebări, am primit și răspunsuri la contexte suplimentare dar conexe de care am fi putut fi interesați precum și explicații detaliate legate de funcționarea codului oferit sau motivele furnizării răspunsului în cazul în care s-a luat o decizie.

În ceea ce privește zona de dezvoltare efectivă a aplicațiilor se observă că deja s-a trecut de perioada în care se considera că I.A. este ceva inovator și s-a trecut într-o zonă de competitivitate între soluțiile de I.A. generativ care ajută în această fază. Soluții de *Intellisense* cu un motor I.A. în spate concurează acum pentru a câștiga preferințele utilizatorilor nu doar prin calitatea răspunsurilor ca a ajuns, după cum s-a văzut mai sus, la un nivel foarte ridicat dar și prin elemente de îmbunătățire a calității vieții (quality of life improvements) cum ar fi sugestiile de tip umbră, adoptarea unei politici neopinate în furnizarea sugestiilor, etc., pentru a fi cât mai utile dar și cât mai ușor de folosit.

La întrebări mai puțin tehnice – cum este cea legată de crearea unui mediu „plăcut” de lucru am primit răspunsuri pertinente care iau în considerare personalizarea mediului de lucru și alte aspecte similare care țin mai mult de aspecte psihologice decât de aspecte tehnice, lucru care ne duce cu gândul la un viitor apropiat în care inteligența artificială va fi un asistent personal aproape indispensabil.

Se observă de asemenea un trend nou de punere în valoare a ideilor și de promovare a dezvoltării de aplicații fără niciun fel de limbaj de programare învățat. Există tot mai multe platforme care promit dezvoltarea unei aplicații doar pe baza unor prompt-uri descriptive [11] [12] fără a scrie vreo linie de cod – astfel de platforme indicând prin simpla lor existență maturitatea soluțiilor A.I. în dezvoltarea aplicațiilor și încrederea de care se bucură pentru a crea un produs software plecând, practic, doar de la o idee.

Este evident că paradigma de dezvoltare a soluțiilor software se va schimba mai devreme decât mai târziu și trebuie să ne pregătim pentru o lume în care inteligența artificială va face munca grea – așa-numita „heavy lifting” – iar valoarea reală a aplicațiilor va proveni din valoarea lor intrinsecă pentru utilizatori – adică în ce mod rezolvă o problemă a acestora și mai puțin de calitatea dezvoltării care va fi ținută la un standard înalt de către unelte de dezvoltare pe bază de inteligență artificială.

Direcții viitoare de cercetare

În viitor s-ar putea avea în vedere studierea mai multor răspunsuri din diferite faze de implementare a proiectelor de dezvoltare software, eventual modul în care ar reacționa la situații excepționale în dezvoltarea proiectelor. De asemenea ar avea valoare o investigație comparativă legată de implementarea unor mecanisme de testare și îmbunătățire continuă a soluției software. Un scenariu, de asemenea, interesant de explorat ar fi introducerea unei zone izolate de testare într-un mediu real în care inteligența artificială să prototipeze o parte din livrabile și apoi să le evalueze pe baza unor criterii prestabilite.

Alte posibile subiecte care ar putea constitui teme de cercetare legate de această temă ar putea constitui găsirea unui mecanism de evaluare între diferite motoare de inteligență artificială, poate chiar concurente, în vederea optimizării răspunsurilor primite pentru diferite teme, un fel de colaborare între I.A. în vederea găsirii celor mai bune soluții la problemele puse.

Credit

Multe din ideile generale din această lucrare au fost inspirate din cursurile și laboratoarele de la disciplina Inginerie Software Avansată. [13] Contribuțiile personale originale ale autorilor derivă, în mare parte, din idei sau concepte desprinse din discuții de la aceste cursuri și laboratoare.

Bibliografie & Referinte

- [1] "History of artificial intelligence," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/History_of_artificial_intelligence
- [2] A. Rockwell, "The History of Artificial Intelligence," Harvard Graduate School of the Arts and Sciences - SITN. [Online]. Available: <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>
- [3] A. Turing, "Computing Machinery and Intelligence," *Mind*, pp. 433–460, 1950.
- [4] N. J. Nilsson, *The Quest for Artificial Intelligence*, 1st ed. Cambridge University Press, 2009. doi: 10.1017/CBO9780511819346.
- [5] "ELIZA," Wikipedia. [Online]. Available: <https://en.wikipedia.org/wiki/ELIZA>
- [6] A. Mashkoor, T. Menzies, A. Egyed, and R. Ramler, "Artificial intelligence and software engineering: Are we ready?," *Computer*, vol. 55, no. 3, pp. 24–28, 2022.
- [7] M. P. Nath, P. Pandey, K. Somu, and P. Amalraj, "Artificial intelligence & machine learning: the emerging milestones in software development," *IJRSI*, vol. 5, no. IX, pp. 36–44, 2018.
- [8] M. Shehab, L. Abualigah, M. I. Jarrah, O. A. Alomari, and M. S. Daoud, "(AIAM2019) artificial intelligence in software engineering and inverse," *Int. J. Comput. Integr. Manuf.*, vol. 33, no. 10–11, pp. 1129–1144, 2020.
- [9] "Software engineering," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Software_engineering
- [10] A. Fan *et al.*, "Large language models for software engineering: Survey and open problems," *ArXiv Prepr. ArXiv231003533*, 2023.
- [11] "AI App Generator to Generate Your App," Appy Pie. Accessed: May 31, 2024. [Online]. Available: <https://www.appypie.com/ai-app-generator>
- [12] "Builder.ai® - Award Winning App Development Platform." Accessed: May 31, 2024. [Online]. Available: <https://www.builder.ai/>
- [13] "FIȘA DISCIPLINEI - INGINERIE SOFTWARE AVANSATĂ, Prof.dr.ing. Cristina TURCU, as.dr.ing Ovidiu GHERMAN." [Online]. Available: https://fiesc.usv.ro/wp-content/uploads/sites/17/2023/02/08.USV_.FIESC_.SIC_.DAP_.02.O.08-ISA-Turcu-Cr.pdf