UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

APLICAȚIE WEB PENTRU MANAGEMENTUL PROIECTELOR SOFTWARE

Absolvent
Ivan Bogdan

Coordonator științific
Conf. univ. dr. Boriga Radu-Eugen

București, iunie 2024

Rezumat

Această lucrare descrie evoluția și modul în care a fost implementată TaskPilot, o platformă software inovatoare pentru managementul proiectelor care integrează un chatbot ca manager virtual de proiect. TaskPilot facilitează comunicarea și coordonarea echipei și oferă o interfață simplă pentru a optimiza procesele de management al proiectelor în dezvoltarea software. Chatbot-ul, care este alimentat de inteligență artificială, răspunde rapid și precis la întrebări care se referă atât la metodologiile de dezvoltare, cât și la proiectele și sarcinile actuale. Acest lucru permite echipelor să se concentreze pe aspectele tehnice și creative, indiferent de sistemul lor de lucru (fizic, hibrid sau la distanță). Lucrarea descrie arhitectura aplicației și tehnologiile utilizate pentru implementare și subliniază potențialul impact pozitiv asupra eficienței și productivității echipelor.

Abstract

This paper describes the evolution and how TaskPilot, an innovative project management software platform that integrates a chatbot as a virtual project manager, was implemented. TaskPilot facilitates team communication and coordination and provides a simple interface to optimize project management processes in software development. The chatbot, which is powered by artificial intelligence, responds quickly and precisely to questions relating to both development methodologies and current projects and tasks. This allows teams to focus on technical and creative aspects, regardless of their working system (physical, hybrid or remote). This paper describes the application architecture and technologies used for implementation and highlights the potential positive impact on team efficiency and productivity.

Cuprins

Capitolul 1

Introducere

1.1 Motivație

Pe parcursul mai multor proiecte la care am lucrat la locul de muncă, am avut probleme în utilizarea aplicațiilor de management al proiectelor care erau suprasaturate de funcționalități. Aceste instrumente au o curbă de învățare abruptă, din cauza multitudinilor de capabilități și opțiuni. Adaptarea la aceste platforme este un proces descurajant și consumator de timp pentru cineva la început de drum, cum sunt eu.

Aceste instrumente ajung să fie greu de folosit, devenind mai degrabă o provocare decât o facilitare a managementului proiectelor, deoarece necesită mult timp pentru a înțelege cum să folosești toate funcțiile puse la dispoziție. Experiența personală m-a învățat că, de cele mai multe ori, funcționalitățile de bază sunt aproape singurele folosite. Ca urmare, am început să caut soluții mai simple și mai ușor de înțeles, care nu sacrifică funcționalitatea esențială.

O altă motivație importantă a fost descoperirea faptului că este necesară o comunicare constantă cu managerul proiectului pentru a obține informații suplimentare și clarificări. În multe echipe de dezvoltare software, project managerii sunt vitali pentru coordonarea și monitorizarea proiectului. Acest lucru implică un flux continuu de întrebări și solicitări din partea echipei, care poate încetini progresul și poate deveni frustrant pentru toți membri echipei. Astfel, mi-am dorit să introduc un asistent virtual capabil să răspundă la întrebările frecvente pentru a reduce contactul cu managerul proiectului.

1.2 Aplicații web similare

Pentru a înțelege mai bine platforma TaskPilot și locul său pe piață, este util să facem o comparație cu alte aplicații web similare. În diferite industrii, platformele prezentate mai jos sunt utilizate pentru managementul proiectelor, oferind o varietate de funcționalități. Această secțiune are ca scop prezentarea caracteristicilor principale ale fiecăreia.

1.2.1 Jira

Jira, o platformă puternică de management al proiectelor dezvoltată de Atlassian, este principala platformă de acest tip folosită de dezvoltatorii software. Este renumită pentru capacitatea sa de a gestiona fluxurile de lucru și de a rezolva probleme complexe. O serie de caracteristici importante ale Jira includ [1]:

- **Urmărirea problemelor și a bug-urilor:** gestionarea sarcinilor de dezvoltare, problemelor și bug-urilor.
- **Fluxuri de lucru personalizabile:** crearea de fluxuri de lucru adaptate nevoilor fiecărei echipe.
- **Raportare și analiză:** generare de rapoarte și statistici amănunțite pentru a urmări progresul și performanța echipei.

Deși Jira este o platformă extrem de puternică și versatilă, pentru utilizatorii noi poate fi complicată și greu de învățat.

1.2.2 Trello

Trello este o altă aplicație populară pentru managementul proiectelor. Este cunoscută pentru abordarea sa vizuală a diagramelor Kanban, utilizate pentru a arăta cum ar trebui să fie făcute sarcinile. Între caracteristicile principale ale Trello se numără [2]:

- **Tablouri Kanban:** Organizarea și monitorizarea sarcinilor prin utilizarea de liste, carduri și tablouri într-un mod vizual și intuitiv.
- **Colaborare în timp real:** Permite membrilor echipei să colaboreze în timp real, să facă observații și să atașeze diverse documente la carduri.
- **Puteri (Power-Ups):** Extensii pentru tablouri care includ calendare, voturi și integrări cu alte aplicații.

În ciuda faptului că Trello este foarte ușor de utilizat și de înțeles, are dezavantaje atunci când vine vorba de gestionarea proiectelor mai complexe care necesită o structură și funcționalități mai sofisticate.

1.2.3 GitHub Issues

GitHub Issues este un instrument integrat în platforma GitHub care este folosit în principal pentru gestionarea proiectelor de dezvoltare open-source. Caracteristicile distinctive ale sale includ [7]:

- **Asocierea cu pull requests:** Capacitatea de a lega problemele la pull requests facilitează urmărirea progresului și modificărilor de cod.
- **Automatizarea folosind GitHub Actions:** Automatizări care ajută la gestionarea problemelor și a proceselor de dezvoltare prin configurarea automată a fluxurilor de lucru.
- **Șabloane pentru issues:** Posibilitatea de a crea șabloane personalizate pentru issues, ceea ce permite standardizarea modului în care sunt raportate problemele și cerințele de funcționalități.

Deși GitHub Issues este o opțiune excelentă pentru proiectele care sunt deja găzduite pe GitHub și pentru colaborarea în comunitatea open-source, poate să nu ofere toate caracteristicile necesare pentru gestionarea proiectelor mai complexe.

1.3 Structura lucrării

Această lucrare de licență este structurată pentru a oferi o înțelegere clară și detaliată a dezvoltării și implementării platformei TaskPilot. Structura lucrării este împărțită în șase capitole principale, fiecare dintre acestea concentrându-se pe elemente cheie ale proiectului.

- **Capitolul 1: Introducere.** Acest capitol prezintă motivația din spatele dezvoltării TaskPilot și aplicații web similare și descrie structura generală a lucrării. În acest capitol sunt descrise contextul și importanța proiectului.
- **Capitolul 2: Preliminarii.** Acest capitol acoperă noțiunile de bază și tehnologiile utilizate în dezvoltarea TaskPilot.
- **Capitolul 3: Analiza soluției.** Capitolul de analiză se concentrează pe identificarea cerințelor funcționale și non-funcționale ale platformei TaskPilot. Se discută scenariile de utilizare și tipurile de utilizatori țintă, precum și rolul componentei AI și nevoile pe care aceasta le acoperă. Această analiză oferă o bază solidă pentru designul și implementarea ulterioară a platformei.
- **Capitolul 4: Designul și arhitectura aplicației.** Acest capitol prezintă arhitectura TaskPilot, subliniind modul în care este orientată pe microservicii. Se discută componentele principale ale aplicației și modul în care acestea lucrează împreună, conferind o privire de ansamblu asupra soluției.
- **Capitolul 5: Implementarea soluției.** În acest capitol se găsesc detaliile de implementare ale platformei TaskPilot, folosind tehnologiile descrise anterior. Este

descrisă implementarea bazei de date, precum și modul în care backend-ul interacționează cu aceasta și cu API-ul OpenAI. Se prezintă interfața aplicației și dezvoltarea acesteia având ca obiectiv o experiență cât mai plăcută a utilizatorului. Pentru a garanta scalabilitatea și portabilitatea soluției, este prezentată containerizarea aplicației folosind Docker.

- **Capitolul 6: Concluzii și perspective.** Capitolul final conține concluziile și lecțiile pe care le-am învățat în timpul dezvoltării TaskPilot. Se discută efectele și avantajele utilizării platformei în managementul proiectelor software, oferind și direcții de dezvoltare. Sunt evidențiate posibilele îmbunătățiri și ajustări ale funcționalităților, ce pot fi făcute pentru a satisface cerințele în continuă schimbare ale utilizatorilor.

Capitolul 2

Preliminarii

Acest capitol prezintă fundamentele teoretice și tehnologiile esențiale utilizate în dezvoltarea TaskPilot. Vom aborda noțiuni de management al proiectelor software, pentru o înțelegere aprofundată a temei și a platformei dezvoltate, și vom explora tehnologiile cheie care stau la baza aplicației și asigură funcționalitatea și scalabilitatea platformei.

2.1 Noțiuni specifice temei abordate

2.1.1 Managementul proiectelor software

Managementul proiectelor software este o disciplină complexă care implică planificarea, organizarea și gestionarea resurselor pentru a atinge obiective specifice într-un interval de timp și bugetar clar. Aceasta se ocupă de organizarea echipelor de dezvoltare, urmărirea progresului și asigurarea că toate elementele proiectului sunt aliniate cu cerințele și așteptările stakeholderilor.

De-a lungul deceniilor, managementul proiectelor software a evoluat semnificativ. Pentru primele proiecte software din anii 1950 și 1960, nu existau metodologii structurate și proiectele au fost gestionate ad-hoc, ceea ce a dus la multe eșecuri și depășiri ale bugetelor. Metodologii mai complexe, cum ar fi modelul Waterfall, propus de Winston W. Royce în 1970, au fost dezvoltate ca răspuns.

Modelul Waterfall [16], cunoscut și sub numele de model în cascadă, este un proces de dezvoltare software care necesită finalizarea fiecărei faze înainte de a trece la următoarea. Procesele comune ale modelului Waterfall includ: cerințele, designul, implementarea, verificarea și mentenanța.

Deși a fost popular, modelul Waterfall a avut unele limitări. Modelul presupunea că cerințele sunt bine înțelese la început și rămân neschimbate pe tot parcursul proiectului. În practică, cerințele software se schimbă frecvent, iar Waterfall nu poate face față acestor schimbări. În plus, abordarea secvențială însemna că erorile care au fost descoperite mai târziu în procesul de dezvoltare erau costisitoare și dificil de remediat.

În anii 1990 și 2000, a apărut o nouă metodă numită Agile pentru a aborda aceste probleme. Manifestul Agile [3], dezvoltat în 2001 de un grup de practicieni calificați, a subliniat că adaptabilitatea și cooperarea sunt esențiale în procesul de dezvoltare software.

Metodele Agile, cum ar fi Scrum și Kanban, pun accent pe livrarea incrementală și iterativă. Echipa Agile lucrează în sprinturi scurte, de obicei de două până la patru săptămâni, în loc să încerce să definească toate cerințele de la început. Aceasta permite grupurilor să răspundă rapid la modificări și să îmbunătățească continuu produsul în funcție de feedback.

Adoptarea metodologiilor Agile a adus numeroase beneficii:

- **Adaptabilitate și flexibilitate:** Grupurile sunt capabile să adapteze rapid cerințele și prioritățile.
- **Îmbunătățirea colaborării:** Membrii echipei sunt mai capabili să comunice și să lucreze împreună prin întâlnirile regulate de planificare și retrospectivă și sesiunile zilnice de stand-up.
- **Livrări regulate și incrementale:** Livrările frecvente de funcționalități noi permit părților interesate să monitorizeze progresul și să ofere feedback continuu.
- **Calitate îmbunătățită:** Procesul iterativ îmbunătățește calitatea produsului final, deoarece permite descoperirea și corectarea erorilor mai devreme.

Agile este mai bun pentru proiecte dinamice și complexe cu cerințe care se schimbă în timp, dar modelul Waterfall poate funcționa pentru proiecte cu cerințe clare și neschimbate. Waterfall oferă control mai strict asupra procesului de dezvoltare, dar cu costul flexibilității. În schimb, Agile facilitează un mediu de lucru adaptabil care încurajează colaborarea, care este esențială pentru succesul proiectelor software moderne [13].

2.1.2 Atribuțiile unui manager de proiect

Succesul unui proiect software depinde de managerul de proiect. Un manager de proiect este responsabil pentru planificarea, coordonarea, monitorizarea și finalizarea proiectelor, asigurându-se că acestea sunt livrate la timp, în buget și la nivelul de calitate așteptat [14].

- **Planificarea proiectului.** Definirea obiectivelor proiectului, crearea planurilor de lucru și stabilirea resurselor sunt toate responsabilitățile managerului de proiect. Identificarea fazelor principale ale proiectului, stabilirea termenelor limită și alocarea resurselor umane și materiale sunt toate incluse în acest proces.

- **Organizarea echipei.** Asigurându-se că fiecare membru al echipei are o înțelegere clară a rolurilor și sarcinilor pe care le îndeplinește, managerul de proiect este responsabil pentru organizarea echipei. Întâlniri regulate și sesiuni de feedback sunt organizate pentru a facilita comunicarea și colaborarea între membrii echipei și alte părți interesate.

- **Monitorizarea progresului.** Monitorizarea progresului proiectului este o sarcină esențială a unui manager de proiect. Acesta urmărește progresul sarcinilor, identifică riscurile și ia măsuri corective când este necesar. Indicatorii de performanță și rapoartele de progres sunt folosite pentru a evalua starea proiectului.

- **Administrarea resurselor.** Managerii de proiect sunt responsabili pentru gestionarea eficientă a resurselor, asigurându-se că acestea sunt utilizate în cel mai bun mod posibil. Aceasta include gestionarea bugetului, distribuirea echitabilă a sarcinilor și asigurarea că echipa are toate resursele și informațiile necesare pentru a-și îndeplini sarcinile.

- **Raportarea și comunicarea.** Este responsabilitatea managerului de proiect să mențină o comunicare deschisă cu toate părțile interesate. Acesta generează rapoarte periodice și înregistrează progresul proiectului, ceea ce facilitează o înțelegere clară a progresului și a obstacolelor pe care le întâmpină proiectul și echipa.

2.2 Tehnologii utilizate

2.2.1 Baza de date Elasticsearch

Elasticsearch este o bază de date non-relațională (NoSQL). Aceste tipuri de baze de date nu folosesc relații și tabele tradiționale pentru stocarea datelor, așa cum fac cele relaționale (SQL). Alternativ, Elasticsearch stochează datele în documente JSON indexate, ceea ce permite o căutare rapidă și eficientă. Această structură flexibilă permite gestionarea și stocarea unui număr mare de tipuri de date fără restricțiile impuse de modelele stricte ale bazelor de date relaționale.

Datorită structurii sale distribuite, Elasticsearch permite indexarea și căutarea rapidă a documentelor. Structura sa internă este bazată pe indici (corespondenții tabelelor din bazele de date relaționale), care conțin unul sau mai multe tipuri de documente (corespondentele liniilor din tabele). Fiecare document este format dintr-o colecție de câmpuri (coloanele din tabele), iar fiecare câmp poate conține una sau mai multe valori. Această flexibilitate permite gestionarea și stocarea eficientă a datelor complexe și variate.

Capacitatea de a realiza căutări extrem de rapide este o caracteristică remarcabilă a Elasticsearch. Aceasta este realizată prin utilizarea indexării inverse, o tehnică care

permite căutarea rapidă a documentelor care conțin anumite cuvinte cheie. În plus, Elasticsearch permite utilizatorilor să efectueze interogări amănunțite și să obțină rezultate precise, deoarece suportă operațiuni complexe de căutare și filtrare.

Elasticsearch este destinat să fie scalabil și distribuit. Un cluster Elasticsearch este format dintr-un set de noduri, fiecare nod conținând una sau mai multe replici ale fragmentelor de date, cunoscute și sub numele de shards. Această arhitectură ajută la performanțe ridicate și disponibilitate continuă, asigurând, de asemenea, redundanța datelor. Adăugarea de noi noduri la cluster permite scalarea orizontală a sistemului, oferind Elasticsearch capacitatea de a gestiona cantități mari de date și de a satisface cerințele în creștere ale aplicațiilor contemporane [4].

2.2.2 Limbajul de programare Python

Python este un limbaj de programare de nivel înalt, interpretat și renumit pentru simplitatea și ușurința de utilizare. Python, dezvoltat de Guido van Rossum și lansat pentru prima dată în 1991, a devenit unul dintre cele mai populare limbaje de programare. Acest lucru se datorează sintaxei sale ușor de înțeles și și versatilității în diverse domenii de aplicare [15].

Mai multe caracteristici importante ale limbajului Python îl fac popular, printre care:

- **Sintaxa simplă și ușor de citit:** Sintaxa Python este scrisă și citită ușor, ceea ce o face mai puțin dificilă pentru noii programatori și îmbunătățește productivitatea dezvoltatorilor. Python este adesea recomandat ca primul limbaj de programare pentru începători din mai multe motive, dintre care unul este acesta.

- **Versatilitatea:** Python este utilizat în multe domenii diferite, cum ar fi dezvoltarea web, știința datelor, inteligența artificială, automatizarea și dezvoltarea aplicațiilor desktop, fiind deosebit de util pentru dezvoltatori, datorită versatilității sale.

- **Bibliotecile extinse:** Deoarece Python are un ecosistem larg de biblioteci și framework-uri, dezvoltarea aplicațiilor este posibilă rapid și eficient.

- **Portabilitatea:** Codul Python poate fi rulat pe orice platformă care are instalat un interpretator Python, ceea ce îl face extrem de mobil și ușor de integrat în multe medii de dezvoltare.

- **Comunitatea activă:** O comunitate globală mare și activă de utilizatori Python ajută la dezvoltarea continuă a limbajului și oferă suport prin documentații, tutoriale și forumuri de discuții.

Datorită flexibilității, Python este utilizat în multe industrii și aplicații:

- **Dezvoltare web:** Utilizarea framework-urilor precum Django și Flask permite crearea rapidă a aplicațiilor web robuste și scalabile.
- **Data science and analysis:** Bibliotecile precum Matplotlib, NumPy și pandas sunt esențiale pentru analiza datelor și vizualizare.
- **Inteligență artificială și învățare automată:** Modelele de învățare automată pot fi create și antrenate folosind framework-uri precum TensorFlow și PyTorch.
- **Automatizare și scripting:** Python este utilizat frecvent pentru scrierea scripturilor simple și automatizarea sarcinilor repetitive.

2.2.3 API-urile de tip REST

Un standard arhitectural pentru crearea de servicii web scalabile și eficiente este reprezentat de API-urile de tip REST. Roy Fielding a introdus ideea de REST în teza sa de doctorat din anul 2000 [6] și, datorită simplității și flexibilității sale, a devenit rapid un model dominant pentru dezvoltarea API-urilor [9].

Principiul fundamental al API-urilor REST este acela de a fi stateless, ceea ce înseamnă că fiecare cerere de la client către server trebuie să conțină toate informațiile necesare pentru a o înțelege și a o procesa. Serverul nu stochează starea clientului între cereri succesive, ceea ce simplifică scalarea și îmbunătățește performanța sistemului.

Resursele sunt reprezentări ale datelor gestionate de server în arhitectura REST, iar fiecare resursă este identificată cu un URI (Uniform Resource Identifier). Acest lucru permite accesul direct și clar la resurse specifice prin cereri HTTP. Metodele HTTP obișnuite utilizate în REST includ:

- **GET** pentru a obține resurse,
- **POST** pentru a crea resurse noi,
- **PUT** pentru a actualiza resursele existente,
- **DELETE** pentru a șterge resursele existente.

Între client și server, informațiile sunt transmise în formate standardizate, cum ar fi JSON (JavaScript Object Notation) sau XML (eXtensible Markup Language). Datorită faptului că este ușor de utilizat și simplu în aplicațiile web moderne, JSON este cel mai frecvent utilizat.

API-urile REST sunt comune în dezvoltarea de aplicații moderne și sunt folosite pentru a conecta diferite aplicații și servicii. Pentru a facilita integrări simple și eficiente, majoritatea serviciilor cloud, platformelor de social media și serviciilor de e-commerce

expun API-uri REST. API-urile REST sunt, de asemenea, frecvent utilizate în arhitecturile bazate pe microservicii, fiecare microserviciu având câte o interfață REST pentru a comunica cu celelalte microservicii.

2.2.4 Framework-ul FastAPI

FastAPI este un framework eficient și de ultimă generație pentru crearea de API-uri web bazate pe Python. Lansat în 2018 de Sebastián Ramírez, FastAPI a câștigat rapid popularitate datorită vitezei sale și simplității de utilizare. Este bazat pe Pydantic, care gestionează validarea și serializarea datelor folosind tipuri de date Python convenționale, și Starlette, un framework web asincron de înaltă performanță.

Caracteristicile importante ale FastAPI includ [5]:

- **Performanță ridicată:** Datorită utilizării Starlette și Pydantic, FastAPI este unul dintre cele mai rapide framework-uri pentru construirea API-urilor în Python, comparabil cu NodeJS și Go. Acest lucru îl face potrivit pentru crearea de API-uri rapide și gestionarea de încărcări mari de cereri.
- **Rapid de codat:** FastAPI crește viteza de dezvoltare a funcționalităților cu 200% până la 300% și reduce erorile dezvoltatorilor cu 40%. Este ușor de utilizat și de învățat, și are un suport excelent pentru editoare, astfel reducând timpul necesar pentru citirea documentației.
- **Documentație automată:** FastAPI folosește standardul OpenAPI (cunoscut anterior sub numele de Swagger) pentru a crea automat documentație interactivă pentru API-uri. Acest lucru garantează că documentația este mereu sincronizată cu codul, bazându-se hint-urile de tip și signaturile funcțiilor.
- **Validare și serializare a datelor:** FastAPI validează automat cererile de intrare și le convertește în tipurile de date potrivite folosind hint-urile de tip din Python. Aceasta asigură că API-ul primește întotdeauna date corecte și ajută la prevenirea erorilor de programare frecvente.
- **Suport pentru programare asincronă:** FastAPI, care funcționează bazat pe asyncio, permite gestionarea cererilor concurente și executarea operațiunilor de tip I/O fără blocarea buclei de evenimente. Scrierea unui cod eficient și scalabil este astfel posibilă.
- **Sistem de injecție a dependențelor:** Un sistem de injecție a dependențelor puternic este oferit de FastAPI, ce permite gestionarea și partajarea resurselor între diferite componente ale aplicației. Scrierea codului și a componentelor reutilizabile sunt mai ușoare datorită acestui mecanism.

- **Suport pentru autorizare și autentificare:** FastAPI oferă middleware și decoratori pentru gestionarea logicii de autorizare și suportă mai multe mecanisme de autentificare, cum ar fi OAuth2 și JWT. Acest lucru face securizarea API-ului și protecția datelor sensibile ușoară.

2.2.5 Framework-ul NiceGUI

NiceGUI este un framework pentru crearea de UI în Python care a fost dezvoltat pentru a face dezvoltarea interfețelor grafice care rulează în browsere mai simplă. Deoarece NiceGUI este simplu de utilizat și are o curbă de învățare blândă, dezvoltatorii pot crea rapid aplicații web interactive fără a se preocupa de detaliile complicate ale dezvoltării web.

Cele mai importante caracteristici ale NiceGUI sunt [10]:

- **Simplu de utilizat:** NiceGUI este conceput pentru a fi ușor de înțeles și de utilizat, oferind o interfață simplă pentru crearea elementelor de UI, cum ar fi butoane, dialoguri, scene 3D și grafice. Această simplitate este perfectă pentru dezvoltatorii care doresc să construiască aplicații rapid fără a trebui să scrie cod HTML, CSS sau JavaScript.

- **Bazat pe Python:** NiceGUI permite dezvoltatorilor să construiască atât backend-ul, cât și frontend-ul folosind același limbaj de programare, eliminând nevoia de a învăța alte limbaje de programare pentru frontend. Acest lucru face procesul de dezvoltare mai simplu și scurtează timpul necesar pentru a deveni productiv.

- **Bazat pe FastAPI:** NiceGUI folosește FastAPI pentru a-și îmbunătăți performanța și a permite programarea asincronă. Integrarea simplă cu alte API-uri REST și gestionarea eficientă a cererilor concurente sunt posibile datorită acestui fapt.

- **Elemente UI predefinite:** NiceGUI oferă o varietate de elemente de interfață a utilizatorului predefinite, cum ar fi tabele, grafice și componente de interacțiune precum butoane și selecții. Aceste elemente sunt pregătite pentru utilizare și pot fi modificate pentru a satisface cerințele specifice ale aplicației.

- **Integrare cu Matplotlib și Pandas:** NiceGUI se integrează ușor cu Pandas și Matplotlib pentru analiza datelor și vizualizarea grafică, ceea ce permite afișarea directă a tabelelor și a graficelor în interfața utilizatorului. Astfel, NiceGUI este un instrument excelent pentru crearea de dashboard-uri și aplicații de monitorizare a datelor.

- **Flexibilitate și adaptabilitate:** Chiar dacă NiceGUI este ușor de utilizat și pentru începători, dezvoltatorii cu experiență au mai multe opțiuni de personalizare

avansate. Acest lucru permite crearea de interfețe unice și complexe care sunt personalizate pentru cerințele specifice ale proiectelor.

2.2.6 OpenAI API

Fiind un serviciu avansat de inteligență artificială, OpenAI API permite dezvoltatorilor să integreze capacități de procesare a limbajului natural în aplicațiile lor. API-ul folosește modele de tip GPT (Generative Pre-trained Transformer), cum ar fi GPT-3.5, GPT-4 și GPT-4o. Aceste modele au capacitatea de a crea text, de a răspunde la întrebări, de a traduce texte în alte limbi limbi și de a îndeplini o varietate de alte sarcini legate de limbaj.

API-ul permite utilizatorilor să personalizeze modelele pentru a se potrivi cerințelor specifice ale aplicațiilor, oferind o flexibilitate și o scalabilitate crescută. În plus, OpenAI API oferind o documentație detaliată și un mediu de testare interactiv, sunt facilitate experimentarea și integrarea în alte aplicații.

API-ul oferă măsuri de securitate la nivel enterprise, cum ar fi criptarea datelor, pentru OpenAI securitatea și conformitatea fiind prioritare. Diverse industrii folosesc această soluție pentru a automatiza procese, a îmbunătăți serviciile pentru clienți și a crea conținut inovator [12].

2.2.7 Docker

Docker este o platformă open-source ce permite dezvoltatorilor să automatizeze operarea aplicațiilor într-un container software, garantând că aplicația funcționează corect în orice mediu. Această tehnologie inovatoare a schimbat modul în care sunt dezvoltate, livrate și operate aplicațiile. Acest lucru a creat un mediu portabil și eficient pentru dezvoltarea, testarea și implementarea software-ului.

Între caracteristicile principale ale Docker se numără [11]:

- **Containerizarea:** Deoarece Docker folosește o abordare bazată pe containere, dezvoltatorii pot împacheta aplicațiile și dependențele lor într-un singur container. Aceste containere conțin toate componentele necesare pentru funcționarea aplicației, cum ar fi codul, librăriile, uneltele de sistem și configurațiile. Acest lucru asigură că mediul de dezvoltare este unul consistent și rezolvă problemele de compatibilitate.

- **Portabilitatea:** Containerele Docker sunt foarte ușor de transportat și pot funcționa pe orice sistem care utilizează Docker, indiferent de infrastructura acestuia. Acest lucru permite dezvoltatorilor să creeze aplicații o singură dată și să le ruleze pe mașini virtuale, servere locale sau medii cloud. Transferul aplicațiilor între diferite medii de producție și dezvoltare este ușor datorită portabilității.

- **Scalabilitatea:** Aplicațiile ce folosesc Docker pot gestiona fără efort sarcinile de lucru mai mari datorită capacităților sale de scalare. Dezvoltatorii pot extinde serviciile aplicațiilor prin alocarea mai multor resurse și distribuirea încărcăturii pe mai multe containere folosind framework-uri de orchestrare Docker, cum ar fi Docker Swarm sau Kubernetes. Astfel, aplicația funcționează mai bine și are o disponibilitate ridicată în perioadele de vârf.
- **Versionarea:** Docker facilitează versionarea și rollback-ul aplicațiilor software. Fiecare imagine Docker, care funcționează ca un plan de construcție pentru containere, are o versiune specifică. Prin folosirea de etichete, gestionarea versiunilor software și implementarea rapidă a versiunilor mai vechi, dacă este necesar, sunt simplificate. Astfel procesul de întreținere și testare a aplicațiilor este eficientizat.

Capitolul 3

Analiza soluției

În acest capitol, examinăm soluția TaskPilot, oferind o înțelegere a modului în care aplicația satisface cerințele utilizatorilor. Vom vorbi despre scenariile de utilizare și tipurile de utilizatori, precum și despre modul în care diferitele funcționalități ale aplicației pot fi folosite pentru a optimiza managementul proiectelor. Vom discuta, de asemenea, despre funcționalitățile componentei AI, dar și despre nevoile pe care aceasta le acoperă.

3.1 Rolul componentei AI

Componenta AI din TaskPilot joacă un rol crucial în transformarea modului în care utilizatorii interacționează cu platforma și își gestionează proiectele. Platforma integrează un asistent virtual de management al proiectelor care stă la dispoziția utilizatorului, oferindu-i suport continuu. Acest lucru crește productivitatea și eficiența, oferind sugestii bazate pe cele mai bune practici de management al proiectelor, dar și un acces rapid la date și informații vitale. Această secțiune prezintă atât funcționalitățile ce au la bază componenta AI, cât și nevoile specifice ale utilizatorilor pe care aceasta le satisface.

3.1.1 Funcționalități oferite de componenta AI

Componenta AI este esențială pentru a facilita gestionarea proiectelor și pentru a asigura utilizatorilor un suport constant și de înaltă calitate. Cele mai importante dintre acestea sunt:

- **Suport pentru întrebări și răspunsuri:** Informații detaliate în timp real sunt furnizate de asistentul virtual pentru a răspunde la întrebări legate de proiecte și tickete. Utilizatorii pot întreba despre statusul unui proiect sau al unui ticket, despre prioritățile acestora și despre detalii legate de obiectele asociate lor (utilizatori, bilete subordonate, comentarii etc.). Acest lucru asigură acces rapid la date importante, reducând timpul petrecut de utilizatori căutând informațiile manual.

- **Asistență în managementul proiectelor:** Asistentul AI oferă sugestii și bune practici pentru managementul proiectelor bazate pe metodologiile consacrate. Recomandări privind planificarea eficientă a proiectelor, alocarea resurselor și organizarea sarcinilor pot fi oferite utilizatorilor la cerere. Chatbot-ul joacă rolul unui project manager virtual, oferind asistență continuă și ghidare în timp real, ceea ce îmbunătățește calitatea deciziilor de management al proiectelor.

- **Interacțiune prin chat:** Prin intermediul unui chat integrat în fiecare pagină, utilizatorii pot interacționa cu asistentul AI și pot pune întrebări despre proiecte, tickete și bune practici de project management, facilitând accesul rapid și direct la suport. Chatbot-ul va oferi informații personalizate în funcție de proiectele sau sarcinile pe care utilizatorul le vizualizează raportat la poziția acestuia în fluxul aplicației la momentul plasării cererii, păstrând totodată și istoricul conversației pe parcursul navigării prin aplicație.

3.1.2 Nevoi acoperite de componenta AI

Prin funcționalitățile anterior prezentate, componenta AI satisface diversele nevoi ale utilizatorilor și îmbunătățește eficiența, accesibilitatea și calitatea managementului proiectelor. Această componentă acoperă următoarele nevoi principale:

- **Eficiență și productivitate:** Componenta AI îmbunătățește eficiența și productivitatea echipelor. Având acces rapid la răspunsuri și informații relevante prin intermediul chatului integrat, fără a naviga prin mai multe meniuri și secțiuni, utilizatorii pot aloca mai mult timp rezolvării sarcinilor din componența proiectelor și mai puțin timp celor administrative. Acest lucru permite o gestionare mai eficientă a timpului și a resurselor, ceea ce duce la performanțe mai ridicate ale echipei.

- **Calitate și consistență:** Toate echipele se pot asigura că urmează bunele practici și metodologiile standardizate, îmbunătățind calitatea și consistența proiectelor, cu ajutorul componentei AI, care oferă suport continuu și în timp real. Acest lucru ajută la menținerea unui nivel ridicat de profesionalism și eficiență în gestionarea proiectelor, care contribuie la atingerea obiectivelor organizaționale.

- **Suport decizional:** Managerii de proiect beneficiază de asistența și informațiile asistentului AI, ceea ce reduce incertitudinea și riscul asociat cu gestionarea proiectelor complexe. Asistentul AI își poate modifica raspunsurile pentru a se adapta cel mai bine modului de lucru al echipei, în funcție de metodologia pe care o preferă utilizatorul.

3.2 Utilizatori și scenarii de utilizare

Această secțiune explică diferitele scenarii de utilizare ale aplicației TaskPilot pentru diferitele tipuri de utilizatori. Descrierea tipurilor de utilizatori, scenariile specifice de utilizare, user stories și user personas sunt toate dezvoltate mai jos. Aceste componente sunt esențiale pentru a înțelege cum TaskPilot satisface cerințele specifice ale utilizatorilor și ajută la succesul proiectelor.

3.2.1 Tipuri de utilizatori

TaskPilot este conceput pentru un public divers, fiecare categorie de utilizatori având propriile nevoi și sarcini. Există două categorii principale de utilizatori:

- **Managerii de proiect:**
  - **Roluri și responsabilități:** Managerii de proiect pot planifica, organiza și urmări progresul proiectelor în platforma TaskPilot. Ei sunt responsabili pentru definirea obiectivelor proiectului, pentru alocarea sarcinilor și resurselor și pentru asigurarea că proiectul este finalizat în timp util și în conformitate cu bugetul.
  - **Funcționalități specifice:** dezvoltarea și gestionarea proiectelor, adăugarea de membri echipei, definirea și organizarea ticketelor și a relațiilor dintre ele și utilizarea chat-ului AI pentru asistență și consiliere.
- **Membrii grupului:**
  - **Roluri și responsabilități:** Membrii echipei sunt responsabili pentru finalizarea sarcinilor care le-au fost atribuite și pentru a lucra împreună pentru a atinge obiectivele proiectului. Ei gestionează sarcinile zilnice cu ajutorul platformei TaskPilot și raportează progresul lor.
  - **Funcționalități specifice:** colaborarea cu alți membri ai echipei, monitorizarea și actualizarea statusului ticketelor unde sunt asignați.

3.2.2 Scenarii de utilizare

Scenariile de utilizare explică modul în care diferitele tipuri de utilizatori folosesc TaskPilot pentru a-și atinge obiectivele și pentru a gestiona proiectele într-un mod eficient. Aceste scenarii ne ajută să înțelegem cum funcționează aplicația în lumea reală. Cele mai comune scenarii de utilizare sunt următoarele:

- **Planificarea și inițierea proiectelor:**

- **Descriere:** TaskPilot este un instrument util pentru managerii de proiect pentru a crea proiecte noi, a adăuga membri echipei și a defini tickete (Epics, Stories, Tasks, Bugs). Utilizatorii pot organiza ticketele în funcție de prioritate (Low, Normal, High, Critical) și status (Not Started, In Progress, Closed).
- **Beneficii:** dezvoltarea unui plan de proiect bine structurat și organizarea eficientă a resurselor, oferind claritate și direcție echipei.

- **Monitorizarea progresului proiectului:**
  - **Descriere:** TaskPilot permite urmărirea statusului și priorității ticketelor în timp real. Pe măsură ce proiectul progresează, managerii de proiect și membrii echipei pot actualiza statusul ticketelor. Acest lucru le permite să aibă o imagine clară a progresului proiectului.
  - **Beneficii:** o mai bună viziune asupra statusului proiectului și capacitatea de a reacționa rapid la modificări sau probleme, asigurându-se că proiectul continuă pe drumul cel bun.

- **Colaborarea în cadrul echipei:**
  - **Descriere:** TaskPilot ajută membrii echipei să lucreze împreună și să împărtășească actualizări și date despre ticketele pe care le gestionează.
  - **Beneficii:** Prin acces rapid la informații și feedback constant, sunt îmbunătățite comunicarea și colaborarea, se reduc întârzierile și se crește productivitatea.

3.2.3 User stories

În procesul de dezvoltare a software, user stories sunt o resursă vitală, care ajută la prezentarea caracteristicilor din punctul de vedere al utilizatorului final. Utilizatorii doresc anumite funcționalități ale aplicației în aceste povești scurte și clare. Mai jos sunt prezentate câteva user stories pentru platforma TaskPilot:

- **Ca manager de proiect**, vreau să pot crea un proiect nou, astfel încât să pot organiza și urmări sarcinile echipei mele.
- **Ca manager de proiect**, vreau să pot adăuga și modifica prioritatea ticketelor, astfel încât să pot gestiona mai bine resursele echipei.
- **Ca manager de proiect**, vreau să pot atribui tickete specific unor membri ai echipei, astfel încât să pot aloca sarcinile în mod clar și eficient.
- **Ca manager de proiect**, vreau să pot crea tickete de tip Epic, Story, Task și Bug, astfel încât să pot structura mai bine activitățile proiectului.

- **Ca membru al echipei**, vreau să pot vedea toate ticketele asignate mie, astfel încât să știu ce trebuie să fac.
- **Ca membru al echipei**, vreau să pot actualiza statusul task-urilor mele, astfel încât să reflecte progresul real al muncii mele.
- **Ca membru al echipei**, vreau să pot adăuga comentarii la tickete, astfel încât să pot comunica detalii suplimentare despre sarcini.
- **Ca membru al echipei**, vreau să pot întreba un chatbot despre statusul actual al unui ticket specific, astfel încât să obțin rapid informațiile necesare fără a căuta manual.

Aceste User Stories ajută la definirea cerințelor și priorităților din perspectiva utilizatorilor, asigurându-se că aplicația răspunde nevoilor acestora. Ele oferă un cadru clar pentru dezvoltarea și îmbunătățirea aplicației, pe baza experiențelor și așteptărilor utilizatorilor.

3.2.4 User personas

User personas sunt reprezentări fictive ale diferitelor tipuri de utilizatori ai aplicației, bazate pe date și cercetări reale. Acestea îmbunătățesc înțelegerea comportamentului și nevoilor utilizatorilor. Principalele exemple de user personas pentru platforma TaskPilot sunt prezentate mai jos:

- **Maria, Manager de proiect:**
  - **Vârstă:** 35 ani.
  - **Experiență:** 10 ani în managementul proiectelor IT.
  - **Obiective:** Să planifice și să gestioneze eficient proiectele echipei, să asigure respectarea termenelor și bugetelor.
  - **Nevoi:** Acces rapid la rapoarte de progres, suport în alocarea resurselor, și recomandări pentru bune practici.
- **Alex, Inginer software:**
  - **Vârstă:** 28 ani.
  - **Experiență:** 7 ani în dezvoltarea software.
  - **Obiective:** Să finalizeze eficient sarcinile asignate, să colaboreze cu echipa și să livreze cod de calitate.
  - **Nevoi:** Acces ușor la tickete, actualizări în timp real despre sarcini, și suport tehnic rapid.

Capitolul 4

Designul și arhitectura aplicației

În acest capitol sunt prezentate designul și arhitectura aplicației TaskPilot, precum și modul în care sunt conectate și structurate componentele principale ale sistemului. Vom vorbi despre arhitectura orientată pe microservicii, subliniind beneficiile sale, cum ar fi modularitatea și scalabilitatea. În plus, vom discuta despre structura bazei de date, a backend-ului și a frontend-ului aplicației, expunând modul în care aceste părți conlucrează pentru a oferi o experiență de utilizare coerentă și eficientă.

Vor fi utilizate diagrame pentru a ilustra structura și fluxurile în fiecare secțiune, facilitând înțelegerea interacțiunilor și a designului tehnic. Acest lucru ne va oferi o înțelegere completă a procesului de construire și de funcționare a aplicației TaskPilot și ne va permite să subliniem motivele deciziilor luate la nivel arhitectural.

4.1 Arhitectura orientată pe microservicii

O soluție populară pentru dezvoltarea aplicațiilor complexe este arhitectura bazată pe microservicii, care oferă flexibilitate, scalabilitate și o mai bună gestionare a componentelor individuale. Această arhitectură permite împărțirea aplicației TaskPilot în mai multe microservicii diferite care pot fi dezvoltate și scalate independent. Acest lucru facilitează procesele de dezvoltare și mentenanță, deoarece permite modificarea diferitelor părți ale platformei fără a perturba întregul sistem.

Componentele principale care alcătuiesc microserviciile aplicației TaskPilot sunt următoarele:

- **TaskPilot UI:** Componenta frontend, TaskPilot UI, construită cu ajutorul framework-ului NiceGUI, reprezintă interfața de acces a platformei TaskPilot expusă către utilizatori. Datorită framework-ului folosit, frontend-ul oferă o interfață vizuală intuitivă, ce le permite utilizatorilor să interacționeze cu aplicația. Pentru interacțiunea cu celelalte microservicii și cu componentele externe, TaskPilot UI trimite cereri către API, pe care acesta din urmă le gestionează.

- **TaskPilot API:** Fiind un API de tip REST, construit cu ajutorul framework-ului FastAPI, TaskPilot API reprezintă componenta backend. Acest microserviciu este responsabil de gestionarea logicii de business a aplicației și de interacțiunea atât cu celelalte microservicii, cât și cu componentele externe. API-ul expune endpoint-uri care sunt accesate de serviciul responsabil de UI.

- **Baza de date:** Ca sistem de stocare și indexare a datelor, Elasticsearch oferă capabilități puternice de căutare și analiză. Această componentă este vitală pentru gestionarea datelor proiectelor, deoarece permite un acces rapid și eficient la informația stocată. Elasticsearch este o opțiune excelentă pentru aplicațiile care necesită viteze mari de căutare și filtrare, cum ar fi platforma TaskPilot, datorită scalabilității sale și a capacității de gestionare a cantităților mari de date.

Pe lângă acestea, se adaugă și componenta externă **OpenAI API**, un serviciu ce oferă capabilități avansate de procesare a limbajului natural, pentru a putea beneficia de funcționalitățile AI. Legătura dintre API-ul OpenAI și sistemul TaskPilot se realizează la nivelul componentei responsabile de backend.

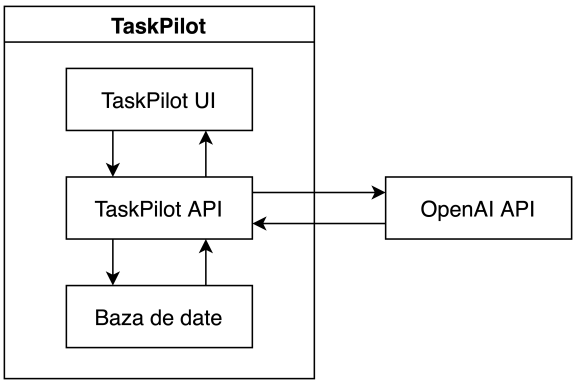

Figura 4.1: Arhitectura bazată pe microservicii a soluției TaskPilot

În ceea ce privește fluxul de comunicare între microservicii, conform diagramei din figura 4.1, utilizatorii interacționează cu TaskPilot prin intermediul interfeței UI, care trimite apoi cererile către API. Acesta din urmă procesează cererile utilizatorilor, interacționând cu baza de date Elasticsearch pentru a accesa sau a actualiza informațiile necesare. Pentru funcționalitățile AI, API-ul trimite cereri către OpenAI API și prelucrează răspunsurile primite, oferindu-le apoi utilizatorilor prin UI.

4.2 Designul bazei de date

Pentru implementarea soluției propuse au fost identificate patru tipuri de modele de date. Pentru o bună organizare a bazei de date și pentru a răspunde într-un mod eficient

la cererile API-ului, am ales ca fiecărei entități să îi fie atribuit câte un index în baza de date Elasticsearch. În figura 4.2 se pot observa tipurile de modele alese și modul în care acestea interacționează, urmând să fie explicate mai jos fiecare dintre ele.

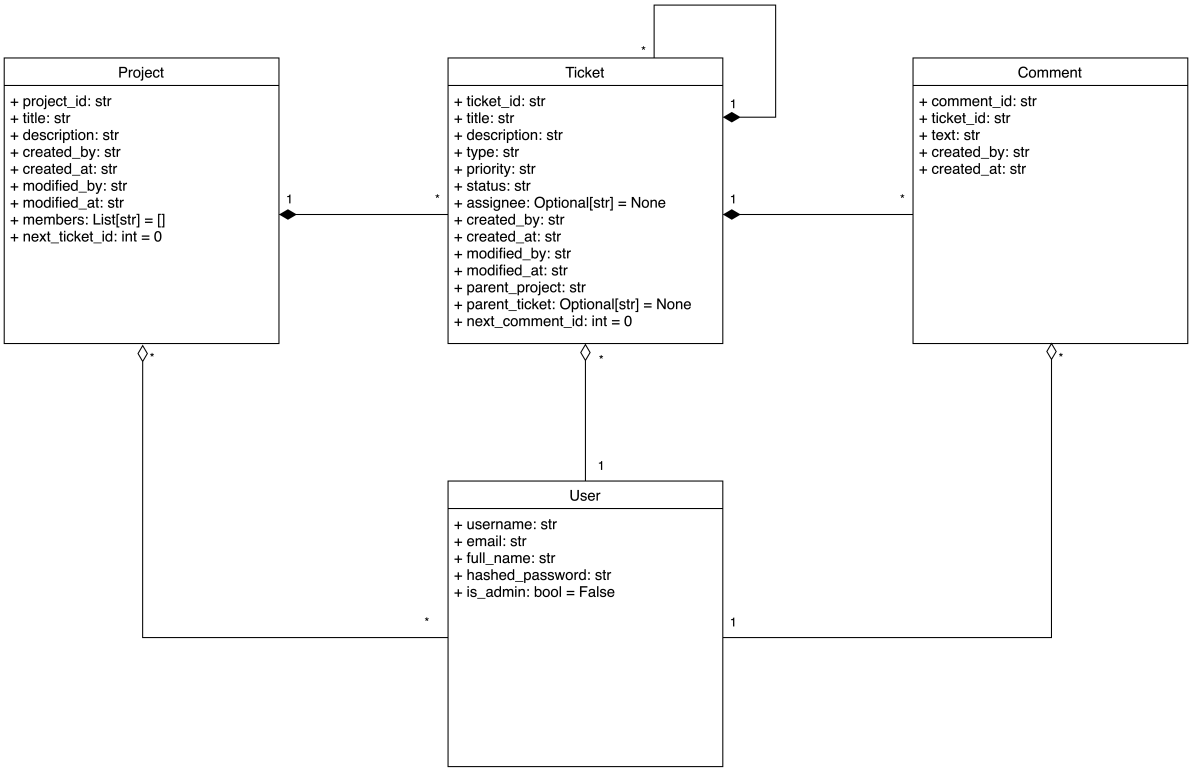

Figura 4.2: Diagrama de clase

- Modelul **User**:
  - **Descriere:** Modelul User reprezintă un utilizator înregistrat al platformei TaskPilot.
  - **Atribute:**
    - **username:** numele de utilizator, folosit ca identificator unic,
    - **email:** adresa de e-mail a utilizatorului,
    - **full_name:** numele complet al utilizatorului,
    - **hashed_password:** parola aleasă de utilizator, peste care a fost aplicată o funcție hash, pentru securitate,
    - **is_admin:** proprietate a unui utilizator de a avea rolul de admin, ceea ce îi oferă drepturi suplimentare.
  - **Relații:** Utilizatorii pot crea și șterge proiecte, tickete și comentarii, pot modifica proiecte și tickete și pot fi membri ai proiectelor sau asignați la un ticket.
  - **Indexul asociat din baza de date:** users.

- Modelul **Project**:
  - **Descriere:** Modelul Project reprezintă un proiect creat în platformă. Fiind entitatea principală din cadrul aplicației, în jurul lui sunt create toate celelalte entități specifice managementului proiectelor.
  - **Atribute:**
    * **project_id:** identificatorul unic al proiectului, setat de utilizator la crearea acestuia,
    * **title:** titlul proiectului,
    * **description:** descrierea proiectului,
    * **created_by:** numele de utilizator al utilizatorului care a creat proiectul,
    * **created_at:** data și ora creării proiectului,
    * **modified_by:** numele de utilizator al utilizatorului care a modificat ultima dată proiectul,
    * **modified_at:** data și ora ultimei modificări a proiectului,
    * **members:** lista de nume de utilizator a membrilor proiectului,
    * **next_ticket_id:** următorul ID disponibil în crearea ticket-elor subordonate, folosit la generarea incrementală a ID-urilor lor.
  - **Relații:** Proiectele conțin tickete, pot fi create, modificate și șterse de utilizatori și pot avea membri asociați.
  - **Indexul asociat din baza de date:** tickets.
- Modelul **Ticket**:
  - **Descriere:** Modelul Ticket reprezintă o sarcină de lucru, fiind unitatea atomică a unui proiect.
  - **Atribute:**
    * **ticket_id:** identificatorul unic al ticket-ului, generat automat într-un mod incremental la crearea acestuia, pe baza proiectului părinte,
    * **title:** titlul sarcinii de lucru,
    * **description:** descrierea sarcinii de lucru,
    * **type:** tipul ticket-ului, poate fi unul dintre următoarele:
      · **Epic:** un obiectiv mare sau o funcționalitate principală ce necesită un timp îndelungat de implementare, reprezentând o componentă majoră a unui proiect,
      · **Story:** o funcționalitate specifică sau o cerință a utilizatorului final, descrisă din perspectiva acestuia, care contribuie la realizarea unui Epic,

        - **Task:** o sarcină specifică și clar definită care trebuie realizată pentru a îndeplini un Story,
        - **Bug:** un defect sau o eroare în software care necesită remediere pentru a asigura funcționarea corectă a acestuia.
      * **priority:** prioritatea sarcinii de lucru, poate fi una dintre următoarele: Low, Normal, High, Critical,
      * **status:** statusul actual al ticket-ului, poate fi unul dintre următoarele:
        - **Not Started:** lucrul pentru acest ticket nu a fost încă început,
        - **In Progress:** lucrul pentru acest ticket a fost început, dar nu a fost încă finalizat,
        - **Closed:** sarcina de lucru a fost îndeplinită cu succes sau nu mai reprezintă o nevoie în cadrul proiectului.
      * **assignee:** numele de utilizator al utilizatorului care trebuie să realizeze sarcina,
      * **created_by:** numele de utilizator al utilizatorului care a creat ticket-ul,
      * **created_at:** data și ora creării ticket-ului,
      * **modified_by:** numele de utilizator al utilizatorului care a modificat ultima dată ticket-ul,
      * **modified_at:** data și ora ultimei modificări a ticket-ului,
      * **parent_project:** identificatorul unic al proiectului din care face parte sarcina de lucru,
      * **parent_ticket:** identificatorul unic al ticket-ului căruia îi este subordonat acest ticket,
      * **next_comment_id:** următorul ID disponibil în crearea comentariilor subordonate, folosit la generarea incrementală a ID-urilor lor.
  - **Relații:** Ticketele sunt create, modificate și șterse de utilizatori, fac parte dintr-un proiect, pot avea comentarii atașate și alte tickete subordonate și au asignat un utilizator pentru finalizarea sarcinii.
  - **Indexul asociat din baza de date:** tickets.

- Modelul **Comment**:

  - **Descriere:** Modelul Comment reprezintă un comentariu lăsat de un utilizator la un anumit ticket, oferind informații suplimentare despre modul de implementare, statusul actual ș.a.m.d.
  - **Atribute:**
    * **comment_id:** identificatorul unic al comentariului, generat automat într-un mod incremental la crearea acestuia, pe baza ticket-ului părinte,

    - **ticket_id:** identificatorul unic al ticket-ului la care a fost lăsat comentariul,
    - **text:** mesajul efectiv al comentariului,
    - **created_by:** numele de utilizator al utilizatorului care a lăsat comentariul,
    - **created_at:** data și ora la care a fost lăsat comentariul.
  - **Relații:** Comentariile sunt atașate ticketelor și create și șterse de utilizatori.
  - **Indexul asociat din baza de date:** comments.

4.3 Structura backend-ului

Backend-ul aplicației TaskPilot este construit folosind framework-ul FastAPI. Acesta oferă o performanță sporită pentru dezvoltarea de API-uri web în Python, ceea ce îl face o alegere ideală pentru dezvoltarea acestei componente. Această parte a aplicației gestionează logica de business, comunicarea cu baza de date Elasticsearch și interacțiunile cu servicii externe, cum ar fi OpenAI API, devenind astfel singurul mod prin care se poate face interacțiunea dintre componenta UI și acestea din urmă.

Endpoint-urile API-ului sunt configurate pentru a gestiona cererile HTTP de la clienți, procesându-le și oferind răspunsuri adecvate. Fiecare endpoint îndeplinește o funcționalitate specifică, iar URI-urile (Uniform Resource Identifiers) sunt structurate logic pentru a face navigarea și utilizarea API-ului mai ușoară.

Pentru a face proiectarea API-urilor simplă și ușor de înțeles, există convenții de denumire, pe care acest serviciu le respectă [8]:

- Utilizarea literelor mici și a liniuțelor pentru separarea cuvintelor în URI-uri.
- Resursele sunt denumite la plural pentru a reflecta colecțiile de obiecte.
- Utilizarea substantivelor pentru definirea resurselor și evitarea utilizării verbelor.
- Verbele HTTP sunt utilizate pentru a specifica acțiunea care trebuie efectuată asupra resursei astfel: GET pentru citire, POST pentru scriere, PUT pentru actualizare, DELETE pentru ștergere.
- Resursele sunt structurate ierarhic pentru a reflecta relațiile dintre ele.
- Parametrii variabili sunt incluși în URI între acolade pentru a specifica elementele dinamice.

Astfel au rezultat următoarele endpoint-uri grupate în funcție de resursele pe care acestea operează:

- Endpoint-uri specifice **utilizatorilor**:
  - `GET /api/users`: furnizează o listă cu toți utilizatorii existenți.
  - `POST /api/users`: creează un utilizator.
  - `GET /api/users/{user_id}`: furnizează un utilizator specific în funcție de numele de utilizator.
  - `PUT /api/users/{user_id}`: modifică un utilizator specific în funcție de numele de utilizator.
  - `DELETE /api/users/{user_id}`: șterge un utilizator specific în funcție de numele de utilizator.
  - `POST /api/users/search`: caută utilizatorii după diverse atribute.
  - `GET /api/users/{user_id}/tickets/assigned`: furnizează o listă cu toate ticketele asignate unui utilizator specific, în funcție de numele de utilizator.
  - `POST /api/users/login`: returnează dacă datele de autentificare sunt corecte pentru un anumit utilizator.
  - `GET /api/users/{user_id}/projects`: furnizează o listă cu toate proiectele la care utilizatorul are acces.
- Endpoint-uri specifice **proiectelor**:
  - `GET /api/projects`: furnizează o listă cu toate proiectele existente.
  - `POST /api/projects`: creează un proiect.
  - `GET /api/projects/{project_id}`: furnizează un proiect specific în funcție de ID-ul său.
  - `PUT /api/projects/{project_id}`: modifică un proiect specific în funcție de ID-ul său.
  - `DELETE /api/projects/{project_id}`: șterge un proiect specific în funcție de ID-ul său.
  - `POST /api/projects/search`: caută proiectele după diverse atribute.
  - `GET /api/projects/{project_id}/tickets`: furnizează o listă cu toate ticketele subordonate unui proiect specific în funcție de ID-ul său.
  - `GET /api/projects/{project_id}/members/{user_id}`: verifică dacă un anumit utilizator este membru al unui proiect specific în funcție de ID-ul său și de numele de utilizator.
  - `GET /api/projects/{project_id}/owners/{user_id}`: verifică dacă un anumit utilizator are drepturi de editare și de ștergere asupra unui proiect specific în funcție de ID-ul său și de numele de utilizator.

- Endpoint-uri specifice **ticketelor**:
  - `GET /api/tickets`: furnizează o listă cu toate ticketele existente.
  - `POST /api/tickets`: creează un ticket.
  - `GET /api/tickets/{ticket_id}`: furnizează un ticket specific în funcție de ID-ul său.
  - `PUT /api/tickets/{ticket_id}`: modifică un ticket specific în funcție de ID-ul său.
  - `DELETE /api/tickets/{ticket_id}`: șterge un ticket specific în funcție de ID-ul său.
  - `POST /api/tickets/search`: caută ticketele după diverse atribute.
  - `GET /api/tickets/{ticket_id}/comments`: furnizează o listă cu toate comentariile subordonate unui ticket specific în funcție de ID-ul său.
  - `GET /api/tickets/{ticket_id}/child-tickets`: furnizează o listă cu toate ticketele subordonate unui ticket specific în funcție de ID-ul său.
  - `PUT /api/tickets/{ticket_id}/status`: modifică statusul unui ticket specific în funcție de ID-ul său.
  - `PUT /api/tickets/{ticket_id}/assignee`: modifică utilizatorul care trebuie să rezolve ticket-ul, în funcție de ID-ul său.
  - `GET /api/tickets/{ticket_id}/owners/{user_id}`: verifică dacă un anumit utilizator are drepturi de editare și de ștergere asupra unui ticket specific în funcție de ID-ul său și de numele de utilizator.
- Endpoint-uri specifice **comentariilor**:
  - `GET /api/comments`: furnizează o listă cu toate comentariile existente.
  - `POST /api/comments`: creează un comentariu.
  - `GET /api/comments/{comment_id}`: furnizează un comentariu specific în funcție de ID-ul său.
  - `DELETE /api/comments/{comment_id}`: șterge un comentariu specific în funcție de ID-ul său.
  - `POST /api/comments/search`: caută comentariile după diverse atribute.
  - `GET /api/comments/{comment_id}/owners/{user_id}`: verifică dacă un anumit utilizator are drepturi de ștergere asupra unui comentariu specific în funcție de ID-ul său și de numele de utilizator.

4.4 Structura frontend-ului

Frontend-ul aplicației TaskPilot este construit folosind NiceGUI, un framework modern și ușor de utilizat pentru dezvoltarea de interfețe grafice în Python. Aceasta componentă a aplicației gestionează interacțiunea cu utilizatorul final și asigură o experiență de utilizare intuitivă și eficientă. Modularitatea structurii frontend-ului facilitează dezvoltarea, întreținerea și scalabilitatea aplicației.

Frontend-ul platformei TaskPilot este alcătuit din mai multe module esențiale, fiecare dintre ele îndeplinind un rol specific în definirea și funcționarea interfeței:

- **Pagini de autentificare (Figura A.1 și Figura A.2):**
  - **Descriere:** Aceste pagini gestionează autentificarea și autorizarea utilizatorilor. Pentru a putea utiliza funcționalitățile aplicației, utilizatorii trebuie să se autentifice, sau, dacă nu au un cont creat, să se înregistreze.
  - **Funcționalități:** Formulare de înregistrare și autentificare, validarea drepturilor utilizatorilor.
- **Pagini de proiecte:**
  - **Descriere:** Utilizatorii pot vizualiza și gestiona proiectele cu ajutorul acestui modul. Toate proiectele pot fi vizualizate într-o pagină dedicată, iar fiecare proiect are o pagină specială care oferă utilizatorilor acces la informații despre proiect, precum și la o listă de tickete care sunt subordonate lui.
  - **Funcționalități:** Vizualizarea listei de proiecte, crearea de noi proiecte, modificarea și ștergerea proiectelor existente (Figura A.6).
- **Pagini de tickete (Figura A.3, Figura A.4 și Figura A.5):**
  - **Descriere:** Asemănător paginilor de proiect, utilizatorii pot vizualiza și gestiona sarcinile de lucru cu ajutorul acestui modul. Toate ticketele pot fi vizualizate într-o pagină dedicată, iar fiecare ticket are o pagină specială care oferă utilizatorilor acces la informații despre el, precum și la o listă de tickete care sunt subordonate lui.
  - **Funcționalități:** Vizualizarea listei de tickete, crearea de noi tickete, modificarea și ștergerea ticketelor existente, adăugări și ștergeri de comentarii.
- **Header și meniu:**
  - **Descriere:** Acest modul facilitează navigarea între secțiunile aplicației. Include elementele de navigare și meniul principal.
  - **Funcționalități:** Navigare simplă între pagini, deautentificare.

- **Chat cu asistent virtual (Figura A.7 și Figura A.8):**
  - **Descriere:** Cu ajutorul acestui modul utilizatorii pot interacționa cu un manager virtual de proiect prin chat. O fereastră de chat apare într-un colț al ecranului atunci când este apăsat butonul de chat.
  - **Funcționalități:** Oferirea de informații generale despre managementul proiectelor, răspuns la întrebări legate de proiecte și tickete și interacțiune prin chat cu project managerul virtual.

Diagrama de mai jos (Figura 4.3) ilustrează modul simplificat de interacțiune dintre paginile principale ale aplicației TaskPilot. Aceasta arată cum utilizatorii navighează de la o pagină la alta, începând cu paginile de autentificare și înregistrare, continuând cu pagina principală, și ajungând la paginile de proiecte și tickete. Fiecare proiect și ticket are, de asemenea, pagini care pot fi accesate pentru a vedea informații detaliate.

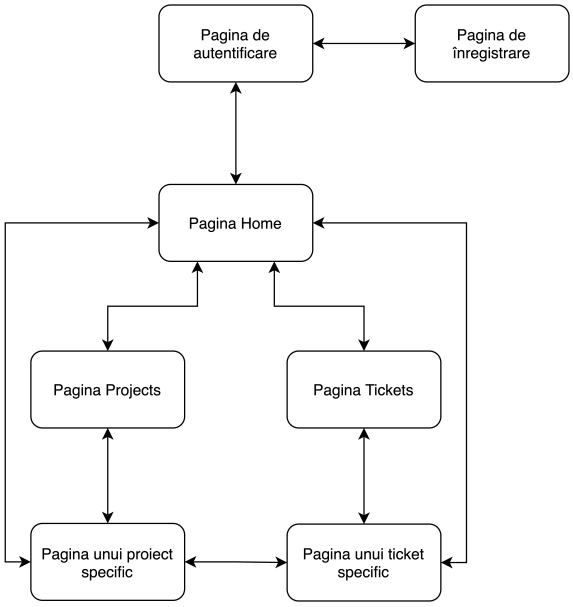

Figura 4.3: Interacțiunea dintre paginile aplicației

Capitolul 5

Implementarea soluției

Acest capitol detaliază procesul de implementare a soluției TaskPilot, acoperind fiecare componentă majoră a aplicației. Vor fi discutate implementarea bazei de date, a frontend-ului și a backend-ului platformei, precum și containerizarea acesteia. Pentru a ilustra modul în care au fost construite și integrate aceste componente, fiecare subsecțiune va include detalii tehnice și exemple de cod.

5.1 Implementarea bazei de date

Implementarea bazei de date în cadrul aplicației TaskPilot se realizează prin utilizarea Elasticsearch, acest tip de bază de date fiind ales datorită flexibilității și performanței ridicate în gestionarea și interogarea unui volum mare de date.

Scriptul de orchestrare verifică prezența unui container Elasticsearch existent pentru a asigura persistența datelor în baza de date. Containerul este pornit dacă este găsit, în caz contrar, fiind creat și configurat unul nou. Acesta este configurat pentru a rula ca un singur nod cu securitatea dezactivată, ceea ce simplifică configurarea și comunicarea în rețeaua Docker care a fost creată.

```
if [ $( docker ps -a | grep taskpilot-elastic | wc -l ) -gt 0 ]; then
  docker start taskpilot-elastic
else
  docker run --net taskpilot --name taskpilot-elastic -p 9200:9200 -p
     ↪ 9300:9300 -e "discovery.type=single-node" -e "xpack.security.
     ↪ enabled=false" docker.elastic.co/elasticsearch/elasticsearch
     ↪ :8.11.4 &
fi
```

Aceste comenzi configurează Elasticsearch pentru a rula pe porturile 9200 și 9300. Acest lucru îi permite accesul la cereri HTTP (prin portul 9200) și la comunicarea internă între noduri (prin portul 9300 - port de ”bârfă”). În plus, setările de configurare specifica-

te, cum ar fi `discovery.type=single-node` și `xpack.security.enabled=false`, permit rularea rapidă și simplă a unei instanțe Elasticsearch pentru dezvoltare și testare.

5.2 Implementarea backend-ului

Implementarea backend-ului pentru TaskPilot implică dezvoltarea și configurarea API-ului care gestionează logica de business, dar și a modulelor care asigură comunicarea cu baza de date și interacțiunile cu serviciile externe. Această secțiune va detalia procesul de creare a backend-ului folosind framework-ul FastAPI.

5.2.1 Interacțiunea cu baza de date

Un set de operații controlează comunicarea cu baza de date a platformei TaskPilot. Aceste operații includ adăugarea, actualizarea, ștergerea și căutarea documentelor stocate în Elasticsearch. Au fost create funcții pentru fiecare dintre acestea, ce ulterior au fost folosite de API pentru crearea endpoint-urilor ce operează pe fiecare resursă specifică și implementează logica de business. O parte dintre implementările lor sunt prezentate în codul de mai jos, toate funcțiile urmărind aceeași structură, bazată pe prinderea și tratarea timpurie a excepțiilor și pe reutilizarea codului.

```
def get_connection() -> Optional[Elasticsearch]:
    try:
        conn = Elasticsearch(config_info.DB_URL)
        logger.info("Generated␣Elasticsearch␣client")
    except Exception:
        logger.error(f"Failed␣to␣generate␣Elasticsearch␣client:␣{exception
            ↪ }")
        conn = None
    return conn
```

În secvența de cod de mai sus se observă funcția care stabilește conexiunea cu baza de date. Aceasta este folosită de toate celelalte funcții responsabile de diverse operații pe baza de date. Încearcă crearea unui obiect de tip Elasticsearch pe baza URL-ului serviciului, capabil să gestioneze operațiunile uzuale. În cazul în care conexiunea nu poate fi stabilită, este returnat un obiect `None`, care va ridica excepții ce vor fi tratate în fiecare funcție în parte, astfel evitând erorile încă de la cel mai de jos nivel.

Pentru a deține controlul total asupra ID-urilor obiectelor din baza de date, funcția responsabilă de inserare gestionează generarea unuia, în cazul în care nu a fost unul pasat ca parametru.

```
def create_item(index: str,
                item: Dict[str, Any],
                item_id: Optional[str] = None) -> Optional[str]:
    if not item_id:
        item_id = str(uuid.uuid4())
    conn = get_connection()
    try:
        response = conn.index(
            index=index,
            id=item_id,
            body=item,
            op_type="create"
        )
    except Exception as exception:
        logger.error(
            f"Failed to create item with id {item_id} in index {index}:"
            f" {exception}"
        )
        return None
    logger.info(f"Created item with id {item_id} in index {index}: {item}"
        ↪ )
    return response["_id"] if response["result"] == "created" else None
```

Pentru a evita suprascrierea obiectelor, în momentul apelării metodei `.index()` specifică obiectului de tip Elasticsearch, se folosește opțiunea `op_create="create"` astfel încât dacă atunci când se încearcă inserarea există deja acel ID în indexul cerut, va fi returnată o eroare. Funcția returnează ID-ul obiectului nou creat sau un obiect `None`, dacă crearea a eșuat.

5.2.2 Interacțiunea cu OpenAI API

În mod asemănător, pentru a interacționa cu API-ul OpenAI, este nevoie de generarea unui client. Acest lucru se realizează în funcția `get_openai_client()` prin pasarea cheii API generată în platforma OpenAI. Pentru conversațiile cu modelul **gpt-4o**, cel mai recent și performant model pus la dispoziție de platformă, este folosită funcția `get_openai_response()`, care primește ca parametri un prompt (instrucțiunea din partea utilizatorului), un prompt de sistem (instrucțiuni de sistem, pentru a personaliza modelul) și istoricul conversației ca o listă de obiecte de tip JSON. Structura chat history-ului este următoarea:

- **role:** reprezintă rolul respectivului participant la conversație, poate fi de următoarele tipuri:
  - **user:** utilizatorul care a pus întrebarea,
  - **system:** sistemul, care oferă instrucțiuni modelului,
  - **assistant:** chatbot-ul, care răspunde.
- **content:** reprezintă mesajul propriu-zis din conversație.

Mesajul și, dacă există, instrucțiunile de sistem se inserează în istoricul conversației. Această listă de mesaje, ce alcătuiește istoricul, este trimisă către modelul AI prin API, iar din răspunsul primit, care poate conține mai multe variante de răspuns, se extrage prima variantă generată. Funcția returnează mesajul primit de la GPT, împreună cu istoricul actualizat al conversației. Implementarea acestei interacțiuni poate fi urmărită în secvența de cod următoare:

```
def get_openai_client() -> openai.OpenAI:
    client = openai.OpenAI(api_key=config_info.OPENAI_API_KEY)
    logger.info("Generated␣OpenAI␣client")
    return client

def get_openai_response(prompt: str,
                        system_prompt: Optional[str] = None,
                        chat_history: Optional[List[Dict[str, str]]] = None
                        ) -> Tuple[str, List[Dict[str, str]]]:
    if chat_history is None:
        chat_history = []
    if system_prompt is not None:
        chat_history.append({
            "role": "system",
            "content": system_prompt
        })
    chat_history.append({
        "role": "user",
        "content": prompt
    })

    client = get_openai_client()
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=chat_history
```

```
    )
    response_dict = response.dict()
    response_choices = response_dict.get("choices", [])
    response_message = (
        response_choices[0]["message"]["content"]
        if len(response_choices) > 0
        else "Unable to generate response"
    )

    chat_history.append({
        "role": "assistant",
        "content": response_message
    })

    logger.info(f"Generated OpenAI response for request '{prompt = }':"
                f" {response_message = } {chat_history = }")
    return response_message, chat_history
```

5.2.3 Implementarea API-ului

Implementarea API-ului TaskPilot implică gestionarea unui set de endpoint-uri care permit realizarea operațiunilor necesare pentru gestionarea diferitelor tipuri de modele de date. Aceste endpoint-uri sunt organizate conform resurselor și operațiunilor specifice, asigurându-se că cererile și răspunsurile sunt gestionate în mod eficient. Definirea endpoint-urilor a fost făcută în secțiunea 4.3.

Pentru o gestionare eficientă și pentru a minimiza numărul de erori de natură umană, am considerat că operațiile acceptate și rutele endpoint-urilor ar trebui definite într-un fișier separat. Ținând cont că acestea vor fi folosite și de alt microserviciu, am creat un director intitulat `common`, care este independent, neaparținând niciunui microserviciu. În acest folder, în fișierul de configurare, `config_info.py`, a fost definită clasa de constante `APIOperations`, ce conține enumerate toate operațiile permise în API. Apoi, un dicționar numit `API_ROUTES` mapează fiecare operație la URI-ul corespunzător. Următoarea secvență de cod ar trebui să ofere claritate asupra mecanismului:

```
class APIOperations:
    USERS_GET = "users_get"
    USERS_CREATE = "users_create"
    # ...

API_ROUTES = {
```

```
    APIOperations.USERS_GET: "/api/users/{user_id}",
    APIOperations.USERS_CREATE: "/api/users",
    # ...
}
```

Pentru serializarea datelor cu care lucrează API-ul am creat clase de cerere și de răspuns, care moștenesc clasa `BaseModel` din librăria **Pydantic**. Aceste clase sunt folosite în signaturile funcțiilor ce vor fi transformate în endpoint-uri cu ajutorul FastAPI. Exemple de clase de cerere se găsesc în următoarea secvență de cod:

```
class CreateUserRequest(BaseModel):
    username: str
    email: str
    full_name: str
    password: str
    is_admin: bool = False

class AIRequest(BaseModel):
    prompt: str
    system_prompt: Optional[str] = None
    chat_history: Optional[List[Dict[str, str]]] = None
```

Pentru standardizarea răspunsurilor oferite de API am creat o clasă de răspuns generică, care este apoi moștenită de alte clase de răspuns mai specifice, care adaugă câmpuri suplimentare. Exemple de astfel de clase de răspuns se găsesc în secvența următoare:

```
class Response(BaseModel):
    message: str
    code: int = 200
    result: bool = True

class GetUserResponse(Response):
    user: Optional[models.User] = None

class AIResponse(Response):
    response: Optional[str] = None
    chat_history: Optional[List[Dict[str, str]]] = None
```

Toate acestea pregătesc scopul final, și anume crearea endpoint-urilor. În fișierul `api_main.py` se găsesc componentele specifice FastAPI, în timp ce funcțiile care implementează logica efectivă se găsesc în fișierul `api_endpoint_helpers.py`. Un exemplu de definire a unui endpoint cu tot cu implementarea funcționalității sale este următorul:

```python
# api_main.py
app = fastapi.FastAPI()

@app.get(config_info.API_ROUTES[config_info.APIOperations.USERS_GET],
        tags=["Users"])
async def get_user(user_id: str) -> api_resp.GetUserResponse:
    """Get a user by id"""
    response = api_help.get_user(user_id)
    return response
```

```python
# api_endpoint_helpers.py
def get_user(user_id: str) -> api_resp.GetUserResponse:
    user_id = user_id.lower()
    index = config_info.DB_INDEXES[config_info.Entities.USER]
    db_get_result = db.get_item(index, user_id)

    if not db_get_result:
        response = api_resp.GetUserResponse(
            message=f"User␣with␣id␣'{user_id}'␣not␣found",
            code=404,
            result=False
        )
        logger.error(response.message)
        return response

    user = models.User.parse_obj(db_get_result)
    response = api_resp.GetUserResponse(
        message=f"User␣with␣id␣'{user_id}'␣retrieved␣successfully",
        user=user
    )
    logger.info(response.message)
    return response
```

În modulul `models.py` sunt implementate modelele conform diagramei de clase din figura 4.2. Clasa `Entities` și dicționarul `DB_INDEXES` sunt folosite pentru a defini niște constante în același mod în care au fost folosite și operațiile acceptate împreună cu rutele către endpoint-uri:

```python
class Entities:
    USER = "user"
```

```
    PROJECT = "project"
    TICKET = "ticket"
    COMMENT = "comment"

DB_INDEXES = {
    Entities.USER: "users",
    Entities.PROJECT: "projects",
    Entities.TICKET: "tickets",
    Entities.COMMENT: "comments"
}
```

5.3 Implementarea frontend-ului

Framework-ul NiceGUI este folosit pentru a implementa frontend-ul platformei TaskPilot. Această secțiune prezintă detaliile din spatele configurării paginilor și a rutelor, ale implementării sistemului de autentificare și ale creării chat-ului cu asistentul virtual. Fiecare subsecțiune va oferi o prezentare generală a procesului de implementare și va include exemple de cod pentru a prezenta cele mai importante componente ale frontend-ului.

5.3.1 Definirea paginilor și a rutelor

Framework-ul NiceGUI având la bază FastAPI, modul de lucru în ceea ce privește definirea paginilor și a rutelor este foarte asemănător cu cel al definirii endpoint-urilor. Într-o manieră similară am definit în `config_info.py` clasa UIPages și dicționarul `UI_ROUTES`:

```
class UIPages:
    HOME = "home"
    LOGIN = "login"
    REGISTER = "register"
    PROJECTS = "projects"
    PROJECT = "project"
    TICKETS = "tickets"
    TICKET = "ticket"

UI_ROUTES = {
    UIPages.HOME: "/",
    UIPages.LOGIN: "/login",
    UIPages.REGISTER: "/register",
    UIPages.PROJECTS: "/projects",
```

```
    UIPages.PROJECT: "/projects/{project_id}",
    UIPages.TICKETS: "/tickets",
    UIPages.TICKET: "/tickets/{ticket_id}"
}
```

În același stil, maparea rutelor la funcții se petrece în fișierul `ui_main.py`, iar definirea efectivă a lor în fișierul `ui_page_helpers.py`. Din cauza complexității paginilor am simțit nevoia de a le regrupa în fișiere mai mici, pentru a le avea într-o structură mai prietenoasă și mai ușor de menținut. Astfel au apărut fișierele `auth_pages.py`, ce conține paginile de autentificare și înregistrare, `header_page.py`, ce conține header-ul și chat-ul cu project managerul virtual (elementele ce se găsesc pe aproape toate paginile platformei), `projects_pages.py`, ce conține atât pagina cu toate proiectele, cât și cea ce conține un singur proiect specific, și `tickets_pages.py`, ce conține atât pagina cu toate ticketele, cât și cea ce conține un singur ticket specific.

Un exemplu de definire a unei pagini și de mapare a ei la o rută se regăsește în următoarea secvență de cod:

```
# api_main.py
@ui.page(config_info.UI_ROUTES[config_info.UIPages.HOME])
def main_page() -> None:
    set_context()
    return ui_help.main_page()
}
```

```
# api_endpoint_helpers
@apply_header
def main_page() -> None:
    ui.markdown(
        f"Welcome␣back,␣**{app.storage.user.get('username',␣'')}**!"
    ).classes("text-5xl␣items-center␣justify-between␣w-full␣self-center"
              "␣px-6␣py-2")
# ......
```

Pentru interacțiunea cu componenta backend, din serviciul UI se vor trimite cereri de API către endpoint-urile definite anterior. Răspunsul va fi prelucrat folosind modelele de date și apoi transpus în componentele de UI disponibile, puse la dispoziție de framework, pentru a fi afișate utilizatorilor. Un fragment de cod, preluat din pagina principală ilustrează această interacțiune dintre cele două componente:

```
# .....
with ui.column().classes("items-center␣w-full␣self-center␣px-6␣py-2"):
    assigned_tickets = requests.get(
```

```
        config_info.API_URL
        + config_info.API_ROUTES[APIOps.USERS_ALL_ASSIGNED_TICKETS].format
            ↪ (
            user_id=app.storage.user.get("username", ""))
    ).json()["tickets"]

    assigned_tickets = [
        models.Ticket.parse_obj(ticket) for ticket in assigned_tickets
        if ticket["status"] != config_info.TicketStatuses.CLOSED
    ]

    for ticket in assigned_tickets:
        with ui.card().classes("w-full"):
            ui.chip(
                text=ticket.ticket_id,
                icon="arrow_right",
                on_click=lambda t=ticket: ui.navigate.to(
                    config_info.UI_ROUTES[
                        config_info.UIPages.TICKET].format(
                        ticket_id=t.ticket_id
                    )
                )
            ).classes("text-white␣text-base")
            ui.label(ticket.title).classes("text-2xl␣font-bold")
# .....
```

Aici se preiau din baza de date prin intermediul backend-ului ticketele asignate utilizatorului, care sunt transformate din format JSON în obiecte de tip Ticket. Pentru fiecare ticket în parte se va afișa titlul acestuia, precedat de un buton, care va face navigarea către pagina proiectului părinte (proiectul din care face parte sarcina de lucru).

5.3.2 Implementarea sistemului de autentificare

Sistemul de autentificare al platformei TaskPilot este integrat în componenta UI și este bazat pe nume de utilizator și parolă. Există două pagini legate de autentificare:

- **Pagina de autentificare:** Aceasta este pagina unde utilizatorii existenți se pot autentifica în aplicație. Utilizatorul introduce numele de utilizator și parola, moment în care se face o cerere de tip `POST` către endpoint-ul `/api/users/login` al backend-ului. Acest endpoint aplică o funcție hash peste parolă și verifică dacă

perechea (nume de utilizator, parola hash-uită) există în baza de date. Dacă autentificarea este reușită, informațiile utilizatorului sunt stocate la nivel local pe client și utilizatorul este redirecționat către pagina dorită.

- **Pagina de înregistrare:** Aceasta este pagina unde utilizatorii se pot înregistra în platformă. Utilizatorul introduce numele de utilizator dorit, adresa de e-mail, numele complet și parola, iar aplicația face o cerere `POST` către endpoint-ul `/api/users` din backend. Acesta aplică funcția hash peste parolă și încearcă să introducă datele primite și prelucrate în baza de date. Dacă înregistrarea s-a făcut cu succes, informațiile utilizatorului sunt stocate local și utilizatorul este redirecționat către pagina principală.

Aceasta este codul puțin simplificat al paginii de autentificare, astfel încât să fie păstrate doar funcționalitățile esențiale. Diferențele dintre cele două pagini fiind minore, nu mai este necesară și prezentarea funcției de înregistrare.

```
def is_user_authenticated() -> bool:
    return app.storage.user.get("authenticated", False)

def login_page() -> None:
    def try_login() -> None:
        url = config_info.API_URL + config_info.API_ROUTES[APIOps.
            ↪ USERS_LOGIN]
        login_response = requests.post(url=url, json={
            "username": username.value,
            "password": password.value
        })
        if login_response.json().get("result", False):
            app.storage.user.update({
                "username": username.value,
                "authenticated": True
            })
            ui.navigate.to(app.storage.user.get("referrer_path", "/"))
        else:
            ui.notify("Wrong␣username␣or␣password", color="negative")
    if is_user_authenticated():
        ui.navigate.to(config_info.UI_ROUTES[config_info.UIPages.HOME])
    with ui.card().classes("absolute-center"):
        username = ui.input("Username").on("keydown.enter", try_login)
        password = ui.input(label="Password", password=True,
            password_toggle_button=True).on("keydown.enter", try_login)
```

O altă componentă importantă a sistemului de autentificare este middleware-ul, care verifică dacă utilizatorul este autentificat și pagina pe care dorește să o acceseze este una existentă. Acesta folosește inclusiv expresii regulate, pentru a verifica validitatea rutelor dinamice (care depind, de exemplu, de ID-ul unei entități). Dacă utilizatorul încearcă să acceseze o pagină restricționată și nu este autentificat, se reține ruta către care a încercat să navigheze și este redirecționat către ea în momentul în care a reușit să se autentifice cu succes. Mai jos este expusă secvența de cod responsabilă de middleware.

```
class AuthMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        path = request.url.path
        routes = Client.page_routes.values()
        is_authenticated = app.storage.user.get("authenticated", False)

        if not is_authenticated and path not in config_info.
            ↪ UNRESTRICTED_PAGE_ROUTES:
            is_valid_route = False
            for route in routes:
                route_pattern = re.sub(
                    r"\{[^{}]*\}", r"[^/]+", route)
                pattern = re.compile(f"^{route_pattern}$")
                if pattern.fullmatch(path):
                    is_valid_route = True
                    break

            if is_valid_route:
                app.storage.user["referrer_path"] = path
                return RedirectResponse(config_info.UI_ROUTES[config_info.
                    ↪ UIPages.LOGIN])

        return await call_next(request)

app.add_middleware(AuthMiddleware)
```

5.3.3 Implementarea chat-ului cu asistent virtual

Indiferent de pagina pe care se află, utilizatorul, odată ce s-a autentificat, are acces la un buton în partea din dreapta jos a paginii pentru interacțiunea cu asistentul virtual. Acest buton, când este apăsat, deschide o fereastră în colțul paginii în care se află chat-ul cu modelul AI. Conversația este persistentă și există și un buton pentru ștergerea acesteia.

Pentru ca asistentul AI să aibă acces la ceea ce ”vede” utilizatorul, în funcție de pagina pe care o accesează, se setează un context. Acesta este format din două câmpuri, **project** și **ticket**, care reține ID-urile proiectului și sarcinii de lucru pe a căror pagină se află utilizatorul. Pentru o pagină generică, care nu este specifică unui proiect sau ticket anume, acele câmpuri sunt nule. Contextul este salvat la client, în storage-ul local. Până la începerea conversației, istoricul mesajelor este și el gol.

În momentul trimiterii unui mesaj se extrage chat history-ul de până acum și se extrag toate documentele din baza de date la care utilizatorul are acces (informațiile despre proiectele pe care le vede, alături de ticketele subordonate și comentariile lor). Acest context este inserat într-un prompt de sistem și se trimite către componenta backend alături de mesajul utilizatorului. Dacă contextul este setat pentru un proiect sau ticket, se oferă informații în prompt-ul de sistem și despre entitatea la care se uită. În momentul în care utilizatorul a părăsit o pagină specifică sau a accesat o alta, se modifică automat contextul.

De pregătirea request-ului de API și trimiterea acestuia către backend, pentru ca apoi să fie redirecționat către API-ul OpenAI, este responsabilă această bucată ce cod:

```
chat_history = app.storage.user.get("chat_history", [])
openai_request = api_req.AIRequest(prompt=message, chat_history=
    ↪ chat_history)
general_context = get_overall_context(app.storage.user.get("username", ""
    ↪ ))
specific_context = app.storage.user.get("context", {"project": None, "
    ↪ ticket": None})
context_project = specific_context.get("project", None)
context_ticket = specific_context.get("ticket", None)
if context_ticket:
    specific_context_message = (config_info.AI_TICKET_CONTEXT_TEMPLATE.
        ↪ format(
        ticket_id=context_ticket))
elif context_project:
    specific_context_message = (config_info.AI_PROJECT_CONTEXT_TEMPLATE.
        ↪ format(
        project_id=context_project))
else:
    specific_context_message = config_info.AI_NO_CONTEXT_TEMPLATE

if not chat_history:
    openai_request.system_prompt = (config_info.AI_INSTRUCTIONS + "␣"
        + config_info.AI_CONTEXT_TEMPLATE.format(context=general_context))
```

```
if openai_request.system_prompt is None:
    openai_request.system_prompt = ""
openai_request.system_prompt += "␣{specific_context_message}"

chat_history.append({
    "role": "user",
    "content": message
})
app.storage.user.update({"chat_history": chat_history})
get_chat_history.refresh()

openai_response = requests.post(
    url=f"{config_info.API_URL}/{config_info.API_ROUTES[APIOps.AI]}",
    json=openai_request.dict()
).json()
app.storage.user.update({"chat_history": openai_response["chat_history"
    ↪ ]})
get_chat_history.refresh()
```

5.4 Containerizarea aplicației

Asigurând un mediu consistent și izolat pentru rularea componentelor aplicației, containerizarea este un element esențial în dezvoltarea și distribuția aplicațiilor moderne. Toate serviciile platformei TaskPilot, incluzând componentele backend și frontend, precum și baza de date Elasticsearch, sunt containerizate în Docker. În această secțiune este prezentat procesul de containerizare, care explică utilizarea Dockerfile-urilor și a scriptului `build_runner.sh` pentru a orchestra crearea și rularea containerelor.

5.4.1 Redactarea Dockerfile-urilor

Dockerfile-urile definesc pașii necesari pentru a construi imaginea Docker a serviciilor. Acești pași includ setarea imaginii de bază, instalarea dependențelor și copierea și rularea codului aplicației. Imaginea Docker rezultată poate fi văzută ca un șablon după care sunt construite containerele.

Un exemplu de Dockerfile este cel al microserviciului TaskPilot API, prezentat mai jos. Singura diferență între acesta și cel al microserviciului TaskPilot UI este dată de diferența căilor către fișiere (trebuie înlocuit `api` cu `ui`) și de portul expus pe care va rula

serviciul (8081, în loc de 8080). Cele două microservicii au fost dezvoltate într-un mod similar, asigurând standardizarea acestora.

```
FROM --platform=linux/amd64 python:3.8.11-bullseye

WORKDIR /taskpilot

ENV PYTHONPATH="${PYTHONPATH}:/."

COPY taskpilot/api/requirements.txt /taskpilot/api/requirements.txt
RUN pip install --upgrade pip
RUN pip install -r api/requirements.txt

COPY taskpilot/common ./common
COPY taskpilot/api ./api

EXPOSE 8080

CMD ["python3", "api/api_main.py"]
```

Pașii urmați în Dockerfile sunt următorii:

- **Setarea imaginii de bază:** Imaginea de bază utilizată este `python:3.8.11-bullseye`, care asigură un mediu Linux compatibil cu arhitectura amd64 și versiunea specificată de Python.
- **Setarea directorului de lucru:** Directorul de lucru este setat la `/taskpilot`.
- **Setarea variabilei de mediu:** `PYTHONPATH` este extins pentru a include directorul rădăcină al proiectului, asigurând accesul la modulele Python.
- **Copierea și instalarea dependențelor:** `requirements.txt` este copiat în container și utilizat pentru a instala toate dependențele necesare folosind `pip`.
- **Copierea codului aplicației:** Codul sursă al aplicației este copiat în interiorul containerului.
- **Expunerea portului:** Portul `8080` este expus pentru a permite accesul la API.
- **Comanda de rulare:** Comanda specificată rulează aplicația prin intermediul fișierului `api_main.py` folosind `python3`.

5.4.2 Orchestrarea prin script bash

Scriptul `build_runner.sh`, care se află la rădăcina proiectului, este folosit pentru a orchestra crearea și rularea containerelor Docker pentru toate serviciile care fac parte din componența soluției TaskPilot. Acest script este responsabil pentru configurarea unei rețele Docker, pornirea containerului Elasticsearch și pornirea serviciilor TaskPilot API și TaskPilot UI. Crearea și pornirea containerului Elasticsearch a fost deja acoperită în secțiunea 5.1.

```
if [ $( docker network ls | grep taskpilot | wc -l ) -eq 0 ]; then
  docker network create taskpilot
fi
```

Se verifică existența rețelei Docker `taskpilot`. Dacă aceasta nu este prezentă, va fi creată.

```
start_service () {
  docker stop taskpilot-"$1" && docker rm taskpilot-"$1"
  docker rmi taskpilot-"$1"

  docker build -f taskpilot/"$1"/Dockerfile_"$1" -t taskpilot-"$1" .

  docker run --net taskpilot -p "$2":"$2" --name taskpilot-"$1" taskpilot
      ↪ -"$1" &
}

start_service api 8080
start_service ui 8081
```

Funcția `start_service()` automatizează procesul manual de construire a containerelor. Pentru pregătirea containerului nou, mai întâi este oprit și șters containerul vechi ce poartă același nume. Eliminând dependențele, imaginea Docker cu același nume poate fi și ea ștearsă. Se începe construirea imaginii noi Docker, cu numele specificat după parametrul `-t`. Într-un final, se rulează containerul bazat pe imaginea proaspăt creată, cu numele specificat și pe portul asociat. Cele două servicii folosesc această funcție pentru a-și construi containerele funcționale.

Capitolul 6

Concluzii și perspective

6.1 Concluzii

Implementarea aplicației TaskPilot a demonstrat utilizarea eficientă a tehnologiilor moderne în dezvoltarea unei platforme robuste și scalabile pentru gestionarea proiectelor software. Această lucrare de licență a explorat întregul proces de dezvoltare, de la analiza soluției și designul arhitectural, până la implementarea propriu-zisă.

Analiza soluției a fost un pas important în definirea cerințelor utilizatorilor țintă. Pentru a identifica caracteristicile esențiale și pentru a evalua beneficiile și dezavantajele fiecăruia dintre acestea, am examinat mai multe aplicații de management al proiectelor de pe piață, cum ar fi Jira, Trello și GitHub Issues. Cu ajutorul acestei analize, au fost dezvoltate cerințele și specificațiile pentru TaskPilot, astfel încât să se poată asigura că aplicația va îndeplini nevoile utilizatorilor.

Utilizarea unei arhitecturi orientate pe microservicii, care permite scalabilitatea și întreținerea simplă a aplicației, a fost scopul designului arhitectural. Fiecare componentă importantă, cum ar fi baza de date, API-ul și interfața, a fost concepută pentru a funcționa ca un serviciu separat, fiind interconectate printr-o rețea Docker.

Elasticsearch, fiind o bază de date distribuită, este cunoscută pentru capacitățile sale de căutare eficientă și de analiză a datelor în timp real. Aceasta s-a dovedit a fi alegerea potrivită pentru baza de date a aplicației TaskPilot, alegere ce a fost dictată de necesitatea gestionării unui volum mare de date într-o manieră eficientă și scalabilă.

Backend-ul aplicației a fost dezvoltat folosind FastAPI, un framework modern și performant pentru dezvoltarea de API-uri web în Python. FastAPI a permis crearea rapidă a endpoint-urilor expuse de API și a asigurat validarea automată a datelor. Implementarea backend-ului a inclus gestionarea diverselor modele de date, precum și integrarea cu OpenAI API pentru funcționalitățile de chat ale asistentului virtual.

Frontend-ul aplicației a fost construit cu NiceGUI, un instrument eficient și ușor de utilizat pentru crearea de interfețe grafice atractive și funcționale. Paginile implementate

și fereastra de chat cu asistentul virtual au asigurat o experiență de utilizare plăcută, fluidă și intuitivă.

Utilizarea Docker a permis containerizarea aplicației, permițând crearea de medii izolate și consistente pentru fiecare componentă a aplicației. Pentru a asigura o implementare eficientă și scalabilă, scriptul bash a fost utilizat pentru a organiza crearea și rularea containerelor Docker.

TaskPilot reprezintă, așadar, un exemplu solid de integrare a tehnologiilor moderne pentru a crea o platformă eficientă de gestionare a proiectelor software. Proiectul a demonstrat că pentru a crea o aplicație scalabilă și performantă, este necesară o analiză atentă, un design arhitectural bine structurat și o implementare strictă. Cu toate acestea, au fost identificate și unele limitări, în special în ceea ce privește securitatea și testarea automată, care oferă oportunități pentru îmbunătățiri viitoare.

6.2 Perspective

Dezvoltarea TaskPilot deschide numeroase perspective pentru îmbunătățiri și extinderi viitoare. Printre direcțiile de dezvoltare care merită explorate se numără:

- **Îmbunătățirea securității:** Securitatea aplicației este unul dintre cele mai importante aspecte care trebuie abordate. Implementarea autentificării și autorizării la nivel de backend, utilizând token-uri JWT și gestionarea rolurilor utilizatorilor, va garanta protecția datelor și accesul sigur la resurse.

- **Extinderea funcționalităților de management al proiectelor:** Adăugarea de noi funcționalități, cum ar fi raportarea avansată, urmărirea deadline-urilor și integrarea cu alte instrumente de productivitate și colaborare, ar putea face platforma TaskPilot să fie și mai utilă pentru echipele de dezvoltare. În plus, generarea de șabloane pentru diverse tipuri de proiecte sau de metodologii ar putea atrage o masă mai mare de utilizatori.

- **Îmbunătățirea capacităților AI:** Utilizatorii pot obține experiențe îmbunătățite și funcționalități suplimentare, cum ar fi recomandări personalizate și automatizarea proceselor de management al proiectelor, dacă sunt dezvoltate și implementate modele AI mai complexe pentru chatbot.

- **Testarea automată:** Stabilirea stabilității și fiabilității aplicației pe termen lung va fi facilitată prin implementarea unui sistem complet de testare automată, care include atât teste funcționale, cât și de performanță.

- **Distribuția și scalabilitatea:** Continuarea utilizării containerelor Docker pentru a extinde aplicația și pentru a explora posibilitățile de implementare în cloud va

permite platformei TaskPilot să se adapteze nevoilor în creștere ale utilizatorilor și să garanteze disponibilitatea serviciului în întreaga lume.

TaskPilot are, așadar, un potențial semnificativ de dezvoltare și îmbunătățire. Direcțiile viitoare de dezvoltare vor contribui la consolidarea și extinderea funcționalităților aplicației, asigurându-se astfel că TaskPilot reprezintă un instrument de management al proiectelor de înaltă calitate pentru echipele de dezvoltare software. Pe măsură ce tehnologia evoluează, TaskPilot va continua să se adapteze și să integreze cele mai recente inovații pentru a rămâne un instrument valoros și relevant în domeniul managementului de proiecte.

Bibliografie

[1] Atlassian, *Jira Features*, Accessed: 2024-06-11, URL: https://www.atlassian.com/software/jira/features.

[2] Atlassian, *Trello Home*, Accessed: 2024-06-11, URL: https://trello.com/home.

[3] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland și Dave Thomas, *Manifesto for Agile Software Development*, Accessed: 2024-06-11, Feb. 2001, URL: https://agilemanifesto.org/.

[4] Elastic, *What is Elasticsearch? | Elasticsearch Guide [8.14]*, Accessed: 2024-06-11, URL: https://www.elastic.co/guide/en/elasticsearch/reference/8.14/elasticsearch-intro.html#elasticsearch-intro.

[5] FastAPI, *FastAPI - Official Documentation*, Accessed: 2024-06-11, URL: https://fastapi.tiangolo.com/.

[6] Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Accessed: 2024-06-11, 2000, cap. 5, URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[7] GitHub, *GitHub Issues · Project planning for developers*, Accessed: 2024-06-11, URL: https://github.com/features/issues.

[8] Lokesh Gupta, *REST API URI Naming Conventions and Best Practices*, Accessed: 2024-06-13, Nov. 2023, URL: https://restfulapi.net/resource-naming/.

[9] Lokesh Gupta, *What is REST?*, Accessed: 2024-06-11, Dec. 2023, URL: https://restfulapi.net/.

[10] NiceGUI, *NiceGUI - Official Documentation*, Accessed: 2024-06-11, URL: https://nicegui.io/.

[11] NoobToMaster, *Overview of Docker's key features and its role in modern software development*, Accessed: 2024-06-11, URL: https://noobtomaster.com/docker/overview-of-dockers-key-features-and-its-role-in-modern-software-development/.

[12] OpenAI, *OpenAI API*, Accessed: 2024-06-11, URL: https://openai.com/api/.

[13] Dan Radigan, *Agile vs. waterfall project management*, Accessed: 2024-06-11, URL: https://www.atlassian.com/agile/project-management/project-management-intro.

[14] Teamhub, *The Role of a Software Development Project Manager*, Accessed: 2024-06-11, Feb. 2024, URL: https://teamhub.com/blog/the-role-of-a-software-development-project-manager/.

[15] Wikipedia, *Python (programming language)*, Accessed: 2024-06-11, 2024, URL: https://en.wikipedia.org/wiki/Python_(programming_language).

[16] Wikipedia, *Waterfall model*, Accessed: 2024-06-11, URL: https://en.wikipedia.org/wiki/Waterfall_model.

Anexa A

Capturi de ecran din aplicația TaskPilot

Această anexă conține capturi de ecran din aplicația TaskPilot, prezentând principalele funcționalități și interfețe. Capturile de ecran oferă o imagine vizuală asupra modului în care utilizatorii interacționează cu aplicația și asupra elementelor de design implementate.

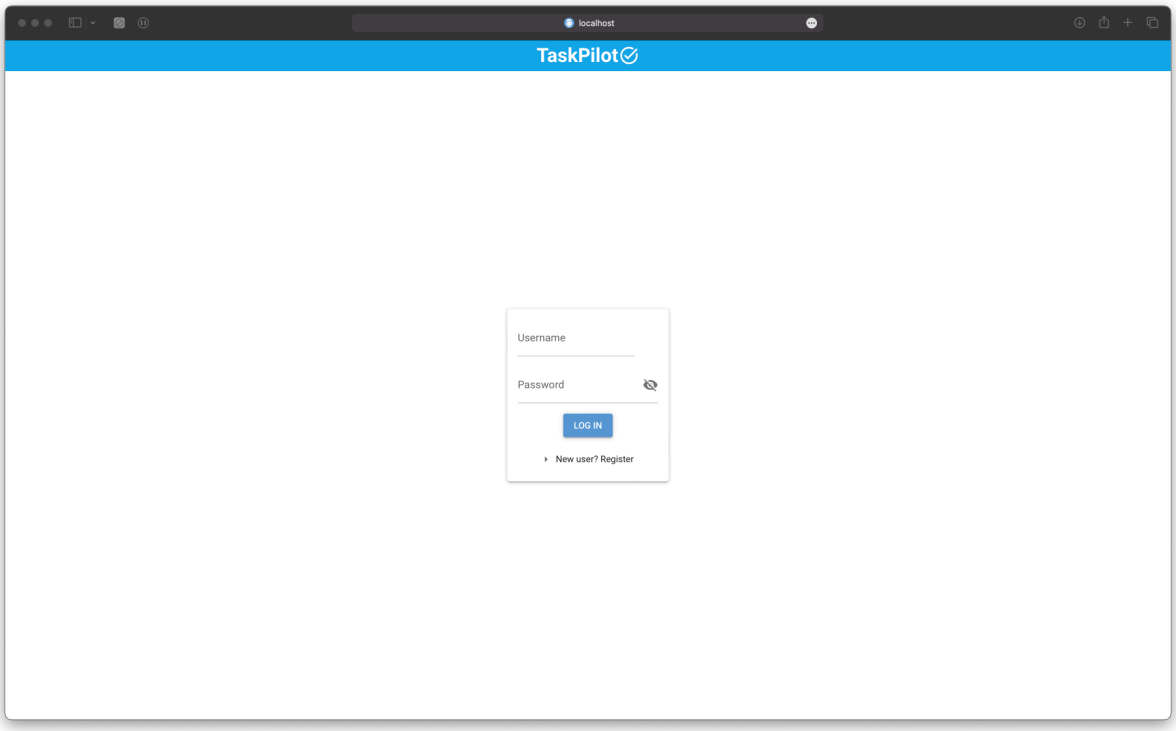

Figura A.1: Pagina de autentificare

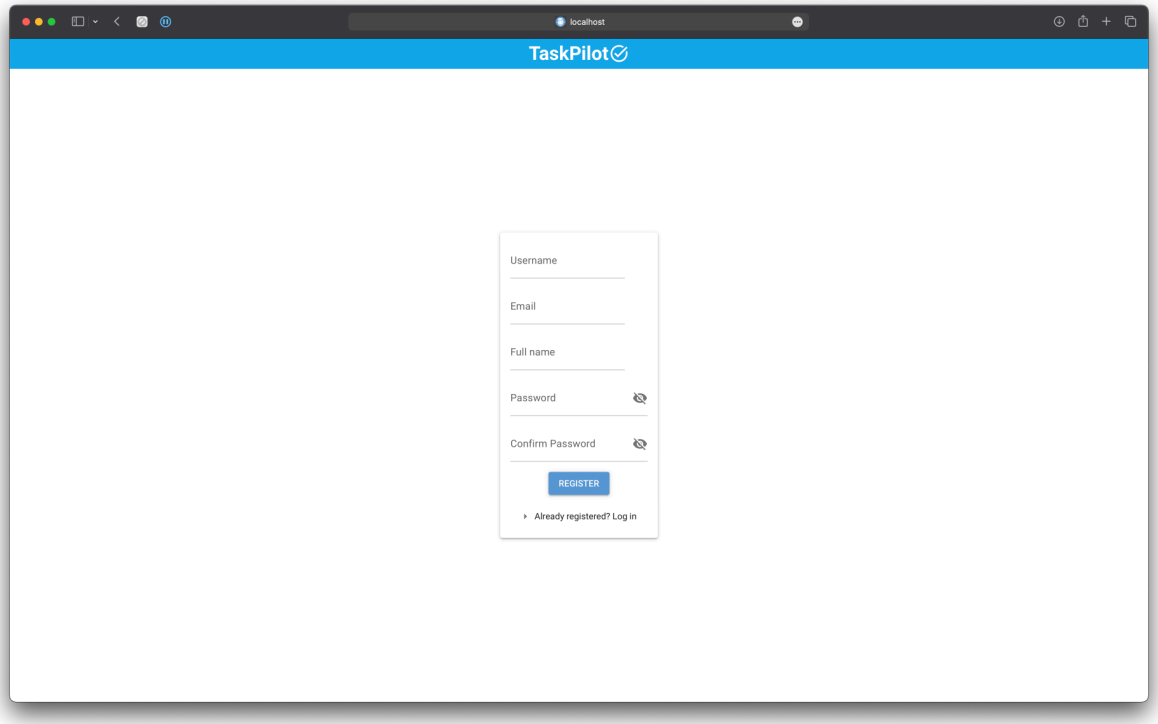

Figura A.2: Pagina de înregistrare

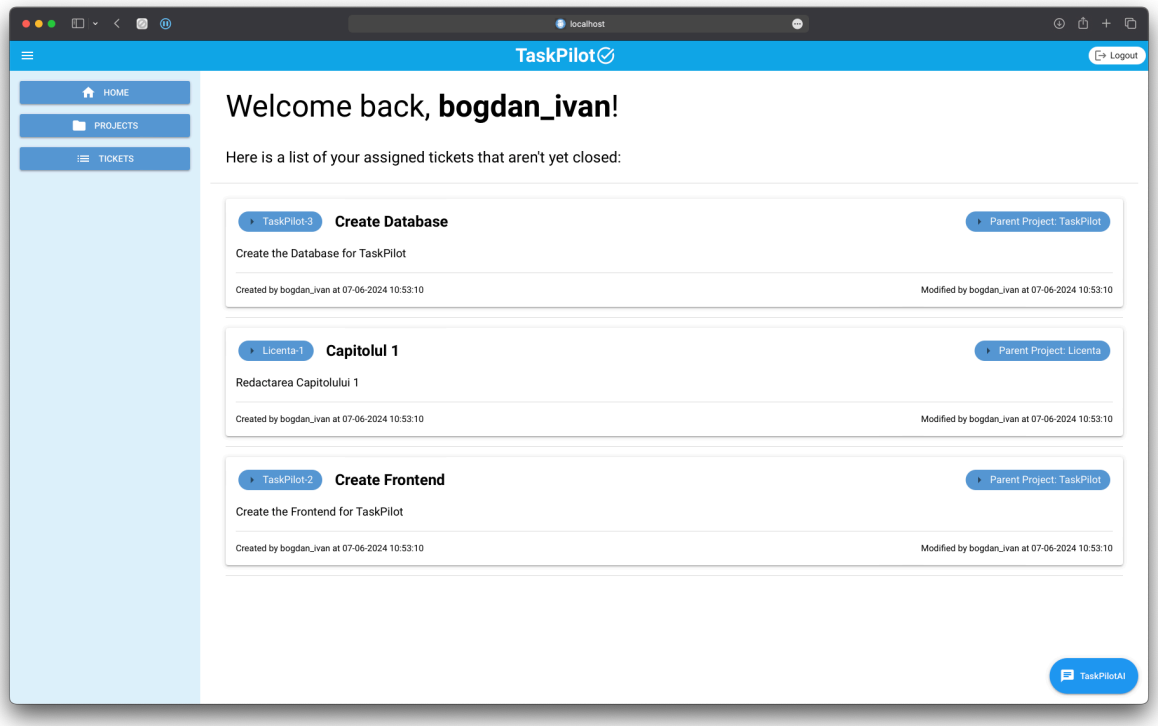

Figura A.3: Pagina principală ”Home”

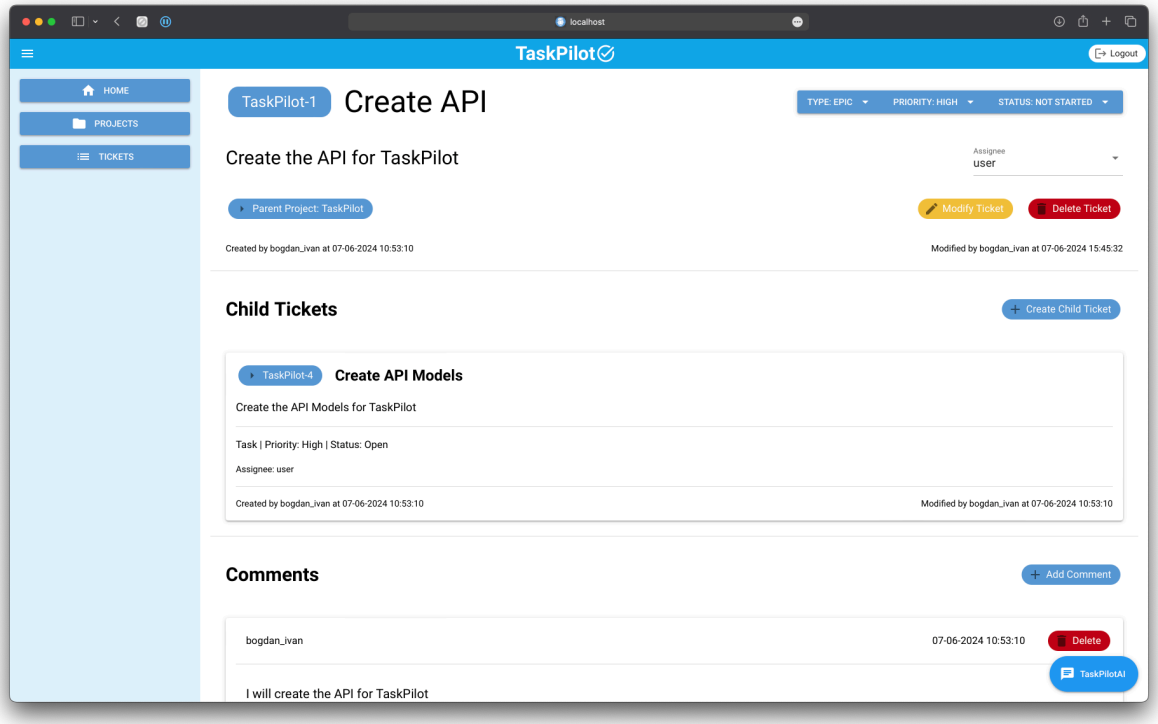

Figura A.4: Pagina ticket-ului cu ID TaskPilot-1

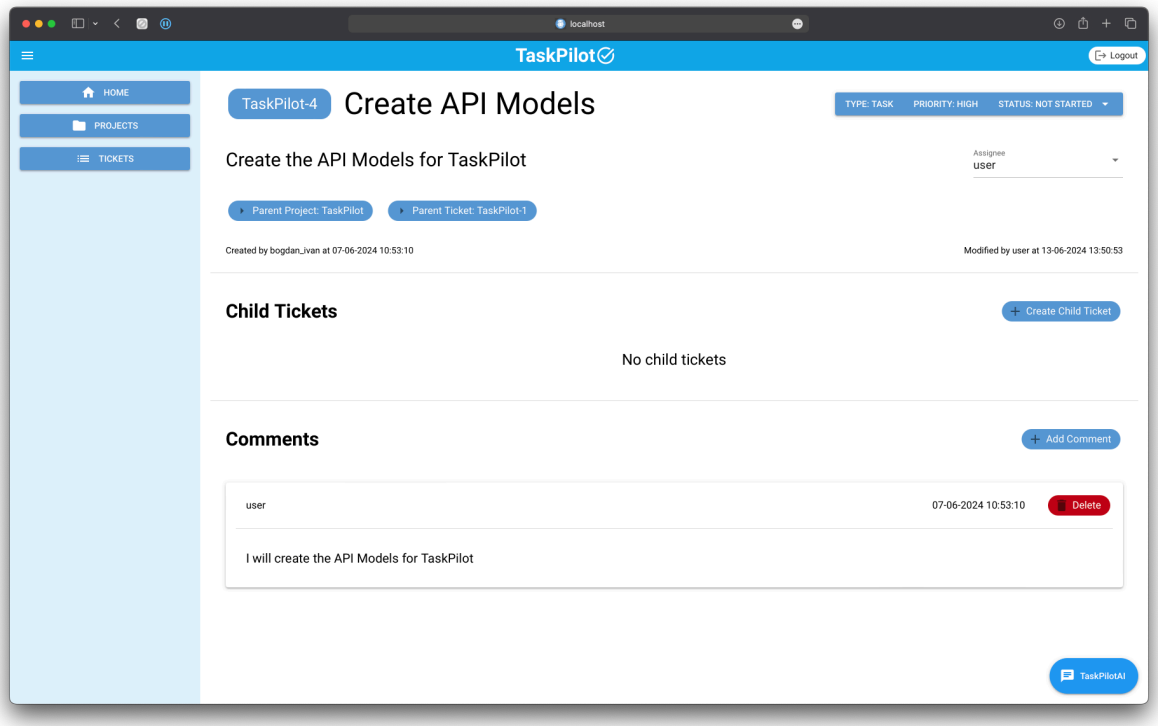

Figura A.5: Pagina ticket-ului cu ID TaskPilot-4

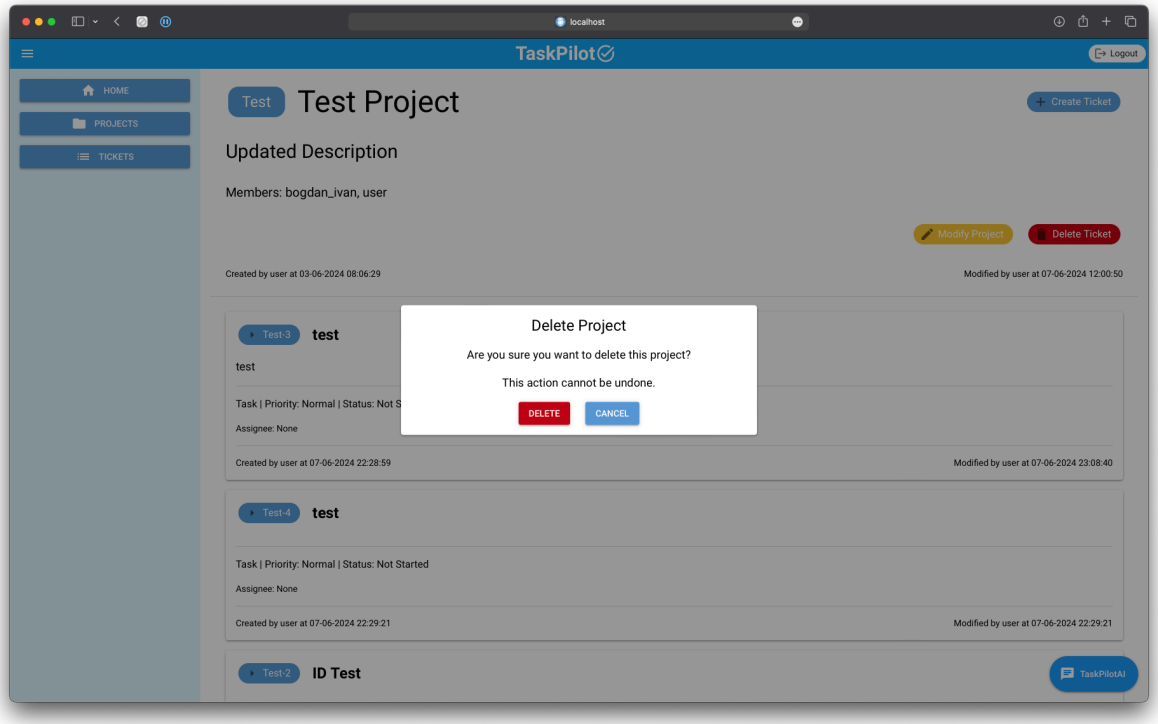

Figura A.6: Ștergerea unui proiect

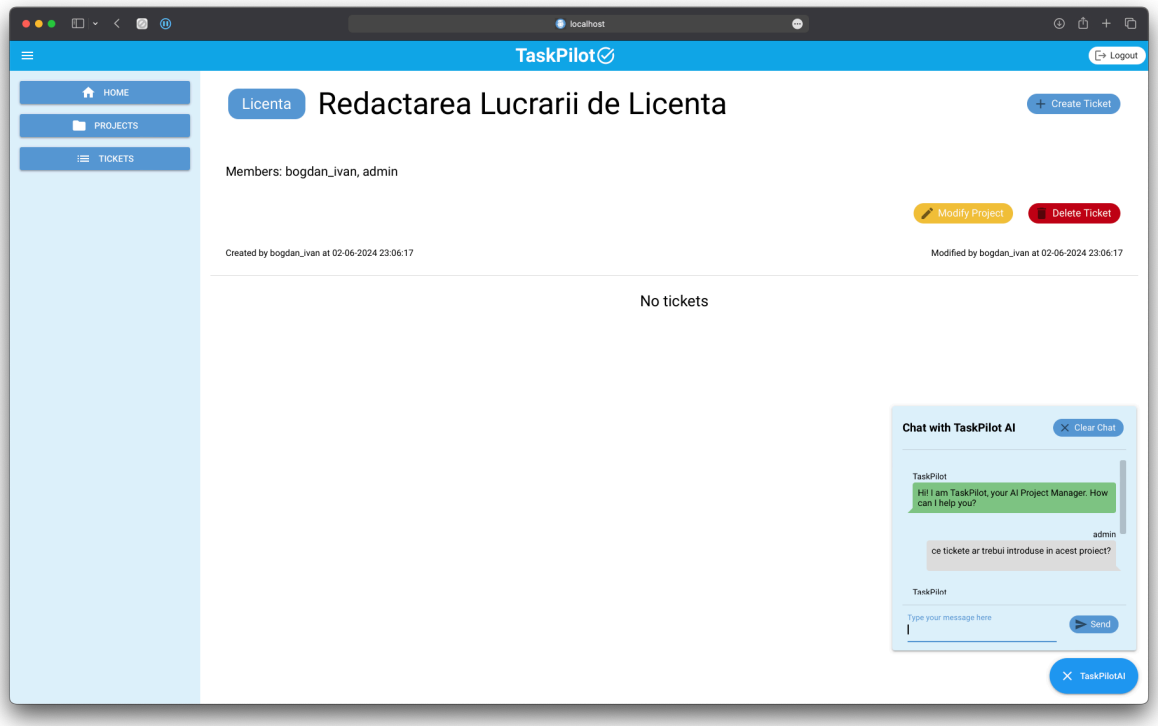

Figura A.7: Întrebare adresată asistentului AI

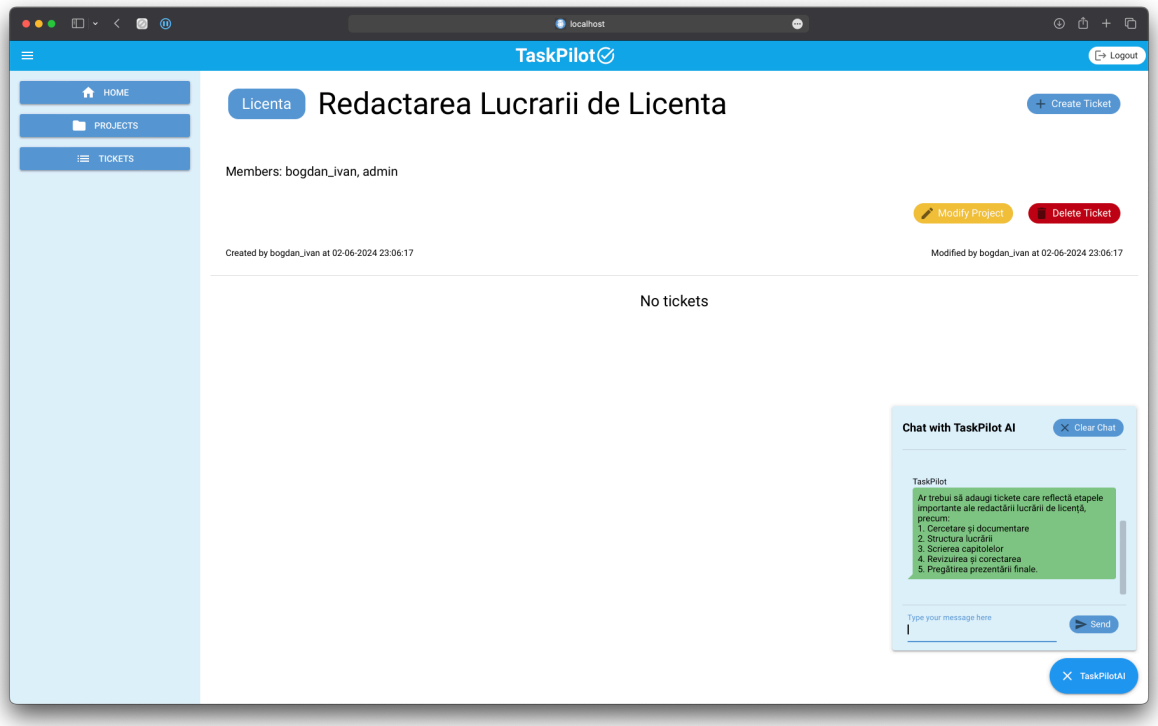

Figura A.8: Răspuns primit de la asistentul AI