UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

APLICAȚIE WEB PENTRU GESTIONAREA EFICIENTĂ A ANGAJAȚILOR ȘI A SARCINILOR DE LUCRU

Absolvent
Alexandra-Diana Ciocan

Coordonator științific
Lect.dr. Natalia Gabriela Ozunu

București, iunie 2024

Rezumat

Un instrument de ajutor pentru orice echipă ce dorește să lucreze organizat este o aplicație de management a sarcinilor de lucru. Aceasta permite centralizarea datelor necesare îndeplinirii muncii de către fiecare membru al echipei și analiza modului de muncă atât individual, cât și la nivel de grup. Totuși, gestiunea datelor dintr-o astfel de aplicație trebuie să fie cât mai simplă pentru a nu distrage atenția de la proiect către operații administrative.

Din acest motiv, această lucrare se concentrează pe crearea unei astfel de aplicații cu integrarea unei functionalități specializate în asignarea automată, într-un mod eficient și echitabil, a atribuțiilor.

Abstract

A task management application is a helpful tool for any team that strives to work in an organized manner. It allows users to centralise the data needed for each team member to carry out their work and analyse how the team works both individually and as a group. However, data management in such an application should be as simple as possible so as not to distract attention from the project to administrative things.

For this reason, this paper focuses on the creation of such an application with the integration of functionality specialised in the automatic assignment of tasks in an efficient and fair way.

Cuprins

Capitolul 1

Introducere

1.1 Motivație

Prin contactul meu de până la momentual acutal cu mediul corporate și, prin extensie, cu metodologia Agile, am observat că o piedică în maximizarea performanței unei echipe o constituie asignarea haotică a sarcinilor de lucru. Aceasta rezultă într-o lipsă de echilibru în volumul de muncă, împreună cu posibilitatea neîncheierii sarcinilor până la data limită a livrării.

Astfel, aplicația centrală acestei lucrări își propune simplificarea gestionării proiectelor din cadrul unei companii și creșterea productivității prin automatizarea procesului de atribuire a sarcinilor de lucru. Utilizatorii cheie sunt managerii, care pot crea panouri cu sarcini de lucru și distribui eficient volumul de muncă, și angajații obișnuiți, care își pot vizualiza sarcinile zilnice și oferi noutăți legat de statusul lor.

1.2 Domenii abordate

Această lucrare are în vedere, în principiu, domeniul dezvoltării aplicațiilor web și al inteligenței arficiale. În cadrul proiectării aplicației, am folosit tehnologii mature precum Java & Spring pentru back-end, Typescript & React pentru front-end și baze de date relaționale, prin intermediul PostgreSQL [de adaugat detalii dupa implementarea agentului inteligent]

1.3 Analiza competitorilor

Investigând cele mai populare aplicații web de gestiune a sarcinilor de lucru în cadrul unei echipe, competitorii principali identificați sunt: Jira, Monday.com și Azure DevOps.

Fiecare dintre acestea excelează în puncte diferite. Jira este foartre bine adaptat metodologiei Agile, constituind un sistem bun de organizare a unui workflow, Monday.com

oferă căi de comunicare pentru echipă, iar Azure DevOps include chiar si funcționalități de version control, facilitând CI/CD.

Atunci când vine vorba de automatizare, Jira și Monday.com pun la dispoziție un sistem bazat pe triggeri. Utilizatorul trebuie să configureze o regulă de forma ”atunci când condiția … este îndeplinită, execută …”. Astfel, e nevoie de efortul de analiză și decizie al utilizatorului pentru a configura toate aceste condiții. Azure DevOps, pe de alta parte, își axează posibilitățile de automatizare spre sarcinile manuale repetitive, în zona de CI/CD.

1.4 Nevoi îndeplinite de aplicație

Aplicația care este subiectul tezei își propune să construiască un algoritm inteligent care să scadă implicarea utilizatorului în procesul de decizie și analiză a informațiilor adunate, pentru a simplifica munca managerilor sau a Scrum Master-ilor, a crește echitabilitatea împărțirii volumului de muncă, și a crește per total mulțumirea și productivitatea în cadrul echipei.

Mai mult, această procesare a informației poate detecta și nivele neobișnuite de productivitate pentru un anumit angajat, pentru a fi ușor atât pentru acesta, cât și pentru manager, să depisteze semnale de alarmă pentru sănătatea mintală a angajului și să acționeze corespunzător.

1.5 Structura lucrării

[de adaugat descriere a fiecarui capitol dupa finalizarea lucrarii]

Capitolul 2

Preliminarii

2.1 Funcționalități principale

2.1.1 Actori

Principalii actori ai cazurilor de utilizare sunt:
- Membru de echipă
- Manager
- Administrator

2.1.2 Cazuri uzuale de utilizare

- Manager:

  1. În calitate de manager, aș vrea să adaug sarcini de lucru pentru echipa mea, cu detalii despre criteriile de acceptanță, prioritate, dificultate și termen limită, pentru ca acestea să fie cât mai clare pentru membri echipei.

  2. În calitate de manager, aș vrea să pot vizualiza factorii de performanță precum disponibilitatea, volumul de muncă, calificarea, eficiența și atribuțiile fiecărui membru al echipei mele, pentru a evalua corect atribuirea de sarcini de lucru noi.

  3. În calitate de manager, aș vrea să pot modifica factorii de performanță precum disponibilitatea, volumul de muncă, calificarea, eficiența și atribuțiile fiecărui membru al echipei mele, pentru ca aceștia să fie mereu actuali.

  4. În calitate de manager, aș vrea să atribui automat sarcini de lucru, bazat pe detaliile sarcinii de lucru și pe factorii de performanță angajatului, pentru a le împărți cât mai echitabil și a maximiza productivitatea și șansele ca sarcina să fie îndeplinită până la termen.

  5. În calitate de manager, aș vrea să pot atribui manual sarcini de lucru, bazat pe analiza mea asupra situației, pentru a ajusta eventualele erori ale recomandă-

rilor, a adapta panoul la noi informații relative la disponibilitate, sau a încuraja creșterea/scăderea productivității pentru un anumit membru al echipei

- Employee:

  1. În calitate de angajat obișnuit, aș vrea să îmi vizualizez sarcinile de lucru împreuna cu detalii despre criteriile de acceptanță, prioritate, dificultate și termen limită, pentru a îmi putea organiza activitatea zilnică.

  2. În calitate de angajat obișnuit, aș vrea să pot actualiza statusul sarcinilor de lucru pentru ca managerul meu să le cunoască progresul.

  3. În calitate de angajat obișnuit, aș vrea să fiu notificat atunci când managerul îmi atribuie o sarcina de lucru nouă, pentru a percepe mai ușor schimbări ale panoului cu sarcinile mele.

- Administrator:

  1. În calitate de administrator, aș vrea să pot crea, actualiza sau elimina utilizatori și echipe, pentru a putea gestiona accesul la platformă și organizarea internă.

  2. În calitate de administrator, aș vrea să pot atribui manageri echipelor create, pentru ca fiecare echipă să fie gestionată de un manager.

2.2 Alegerea tehnologiilor

2.2.1 Typescript&React

Pentru partea de front-end a aplicației web am ales să utilizez Typescript ca limbaj și React ca bibliotecă, împreună cu componente reutilizabile din biliboteca de componente UI Ant Design.

Utilizarea limbajului Typescript, în defavoarea limbajului JavaScript, este explicată prin minimizarea erorilor cauzate de inconsistența adusă de faptul ca JavaScript este un limbaj „loosely typed” („dinamically typed”) și de incertitudinea adusă de acest fapt. Astfel, prezența interfețelor și tipurilor de date bine stabilite facilitează dezvoltarea rapidă și stabilă.

În plus, biblioteca React aduce o perspectivă structurată dezvoltării interfeței, prin definirea de componente reutilizabile în cadrul aplicației. O parte din componentele UI sunt utilizate din biblioteca Ant Design, folositoare prin aspectul vizual unitar și formal, acompaniat de o documentație bine pusă la punct. Un dezavantaj al abordării prin componente este comunicarea între acestea, motiv pentru care am folosit state managerul MobX.

2.2.2 Java&Spring

Pentru partea de back-end a aplicației web am optat pentru limbajul Java și framework-ul Spring și ecosistemul acestuia, având în vedere opțiunile robuste pentru gestionarea logicii de afaceri, a securității și a interacțiunilor cu baza de date.

Versiunea de Java utilizată este Java 21, pentru a avea acces la cele mai noi update-uri ale limbajului, cu cele mai noi feature-uri, precum clase Record, fire virtuale de execuție, și Sequenced Collections.

Versiunea de Spring folosită este 6.1.3, ce vine împreună cu Spring Boot 3.2.2, alături de alte dependințe, printre cele mai notabile numărându-se Spring Security și Jakarta Persistence API (JPA).

Cele mai majore utilități puse la dispoziție de Spring și de ecosistemul de dependințe sunt:

- securitatea asigurată prin generarea de JSON Web Tokens pentru autentificarea și autorizarea utilizatorilor și a cererilor HTTP trimise;

- crearea tabelelor și conectarea la baza de date facilitată de ORM-ul JPA;

- injectarea dependințelor prin constructorii claselor, în scopul reducerii codului boilerplate.

[detaliere spring boot, spring security, spring jpa]

2.2.3 PostgreSQL

Pentru persistența datelor am ales PostgreSQL, un sistem open-source de gestiune pentru baze de date, pentru sistemul său robust de asigurare a integrității datelor și abilitatea de a manevra ușor cantități mari de date.

Capitolul 3

Arhitectura aplicației

3.1 Microservicii

Microserviciile sunt o arhitectură care presupune segmentarea unei aplicații într-o suită de servicii specializate, independente, ce comunică prin protocoale light-weight, în mod popular transmițând date prin HTTP într-un context RESTful. Unul din avantajele principale ale acestei arhitecturi este decuplarea serviciilor implicate. Eșecul unui serviciu nu afectează întreaga aplicație, astfel încât problemele se manifestă izolat și sunt mai ușor de controlat. Un alt motiv pentru care am considerat că această arhitectură este potrivită aplicației este versatilitatea permisă în alegerea tehnologiilor și a limbajelor de programare.

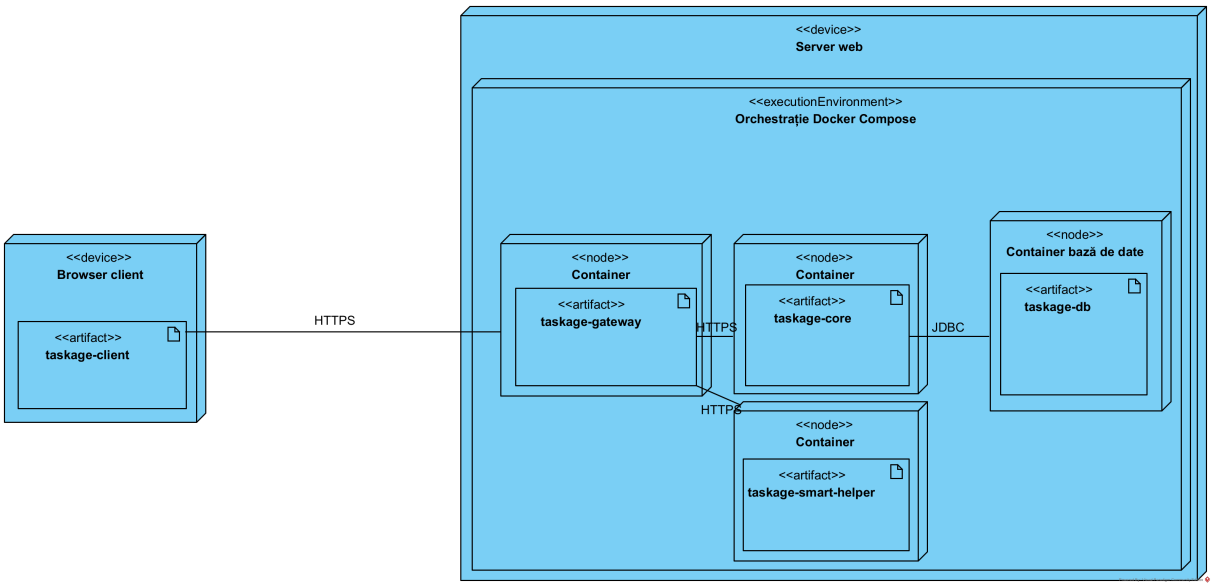

Figura 3.1: Arhitectura high-level

Figura 3.1 ilustrează caracteristica modulară a aplicației. Pe partea de front-end, artefactul aplicației React taskage-client rulează în browser-ul web al utilizatorului prin

codul transpilat din TSX în JS. Acesta comunică cu serverul prin apeluri HTTP, utilizând componenta Spring Cloud taskage-gateway ca punct de intrare către functionalitățile back-end. Serviciile de back-end comunică la rândul lor prin protocolul HTTP, iar conexiunea la baza de date este facilitată de JDBC.

[de completat legaturile taskage-smart-helper cu bd dupa ce ma hotarasc daca fac alt db sau continui pe acelasi]

3.2 taskage-core

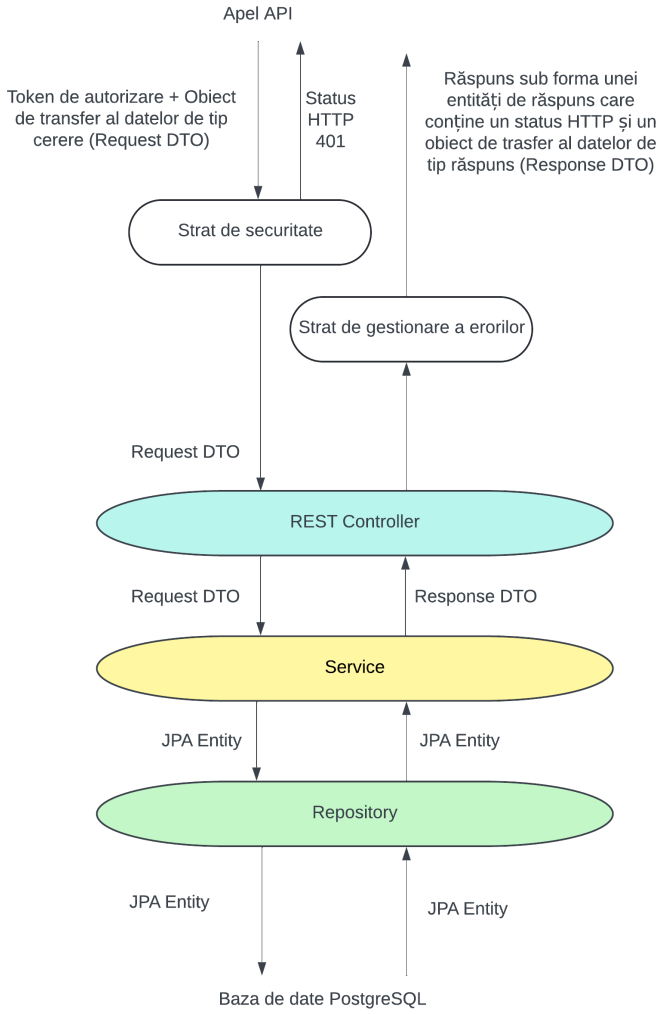

Figura 3.2: Fluxul datelor în componenta core

Componenta care se ocupă cu funcționalitățile cele mai de bază ale aplicației este taskage-core. Aici se realizează operațiile CRUD asupra entităților și se gestionează cererile uzuale ale platformei. Structura componentei urmează patternul Controller-Service-Repository, popular în aplicațiile Spring Boot. Acest model de structurare a proiectului

urmărește separarea sarcinilor unui sistem mare în părți mai ușor de manevrat. Fluxul datelor, reprezentat în figura 3.2, este controlat, iar fiecare strat are un scop bine definit.

Apelurile HTTP conțin un JWT token în header și trec prin filtrul de autorizare, implementat cu ajutorul Spring Securit. Dacă tokenul este aprobat ca fiind valabil și având criptat rolul corespunzător, apelul ajunge la end-point-ul asociat din controller. Altfel, aplicația intoarce un răspuns cu status HTTP 401(Unauthorized). Controllerele constituie API-ul aplicației și ca au ca unic scop expunerea unor funcționalități către agenți externi. Datele sunt primite sub forma unui obiect de transfer al datelor(Request DTO), care nu se traduce într-o entitate a bazei de date, fiind doar un mod de transmitere a informației între microservicii. Aceste DTO-uri trebuie de asemenea să respecte validările impuse câmpurilor prin Jakarta Validation.

Dacă DTO-ul a fost validat, acesta este trimis mai departe pentru procesare în zona cu logica de afaceri a aplicației, și anume serviciile. Serviciile procesează informația și eventual o mapează ca entități JPA spre a o transmite mai departe depozitelor(repositories). Depozitele reprezintă stratul de acces al datelor și se ocupă cu toate interacțiunile cu baza de date. Dacă totul se realizează cu succes, se va returna o entitate de răspuns cu statusul HTTP 200, eventual conținând un Response DTO cu datele cerute. Dacă se aruncă vreo excepție, aceasta va fi prinsă de clasa care se ocupă cu gestiunea erorilor la nivel global, GlobalExceptionHandler, și se va trimite înapoi un răspuns sugestiv.

3.3 taskage-client

Componenta care constituie front-end-ul aplicației este taskage-client. Aceasta este un proiect React care folosește MobX pentru gestionarea stării. Una din provocările principale la construcția unui proiect în React este lipsa de structură interentă tehnologiei. Prin comparație cu un framework popular, Angular, proiectele React nu au o structură de fișiere și un flux de date bine-definite, acestea rămânând responsabilitatea dezvoltatorului. Cele două mari abordări pentru structurarea aplicației sunt: abordarea orientată pe cateogorie și abordarea orientată spre funcționalitate. Prima din cele două este mai puțin scalabilă, organizând fișierele după tipul lor(componente, pagini, modele, etc...) în timp ce a doua abordare rafinează mai departe structura, sortând componentele în funcție de funcționalitatea efectivă pe care o implementează. Am ales cea de-a doua alternativă pentru că densitatea componentelor devenise prea mare pentru a putea fi gestionate la un loc.

Deși front-end-ul reprezintă un SPA(Single Page Application), încărcarea paginilor se face pe baza rutelor pentru eficiență. Rutele sunt atât publice, cât și private. Cele private se pot accesa pe baza unor criterii precum tipul utilizatorului, iar în cazul lipsei privilegiilor de acces, acestea vor redirecționa utlizatorul către o pagină cu un mesaj sugestiv. Rutele servesc pagini, implementate în folderul „pages”, iar paginile randează

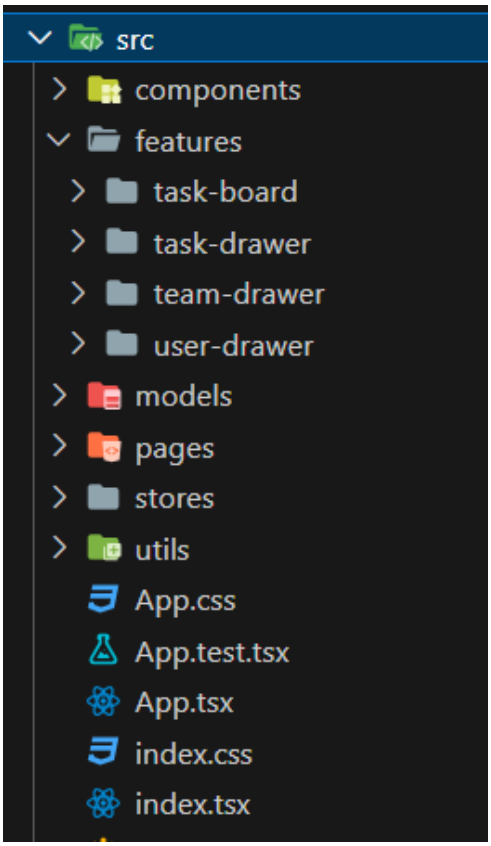

Figura 3.3: Structura de fișiere a aplicației React

diverse componente care, daca au logică complexă, sunt plasate într-un folder cu nume sugestiv funcționalității de care țin, în folderul „features". Folderul „components" este rezervat pentru componente reutilizabile, generice.

Stocarea datelor necesare în componente aflate la distanță mare pe ierarhia din DOM se face in „stores", concept specific MobX. Tot aici se fac si apelurile către server și gestionarea datelor utilizatorului spre setarea corespunzătoare a tokenului în header-ul cererilor.

3.4 taskage-smart-helper

[de adaugat capitol despre microserviciul pyhton dupa ce are o structura bine definita]

Capitolul 4

Implementarea sistemului

4.1 Spring Framework

Capitolul 5

Concluzii