

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Principiile metodologiei kanban în format electronic

propusă de

Mihai Corciu

Sesiunea: *Februarie, 2019*

Coordonator științific

Lector, Dr. Cristian Traian Vidrașcu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Principiile metodologiei kanban în format electronic

Mihai Corciu

Sesiunea: *Februarie, 2019*

Coordonator științific

Lector, Dr. Cristian Traian Vidrașcu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP,
absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de
..... specializarea, promoția
....., declar pe propria răspundere, cunoscând consecințele falsului în
declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr.
1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul:

_____elaborată sub îndrumarea dl. / d-na
_____, pe care urmează să o susțină în fața
comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin
orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la
introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări
științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei
lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie

răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am
întreprins-o.

Data azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Principiile metodologiei kanban în format electronic*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent *Mihai Corciu*

(semnătura în original)

Cuprins

Introducere.....	4
Contribuții.....	6
Motivație.....	8
1. Arhitectură.....	9
1.1 Securitate.....	9
1.2 Server.....	10
1.3 Client.....	13
1.4 Baza de date.....	15
2 Tehnologii utilizate.....	21
2.1 ReactJS.....	21
2.2 React Redux.....	22
2.4 Node.js.....	23
2.5 ExpressJS.....	23
2.6 Bootstrap.....	23
2.7 React DnD.....	24
3 Utilizarea aplicației.....	25
3.1 Înregistrare, autentificare și recuperare parolă.....	25
3.2 Pagina proiectului.....	26
3.2.1 Meniu.....	26
3.2.2 Proiect, componentă și tichet.....	28
3.2.3 Filtre.....	29
3.2.4 Pagină creare.....	30
3.2.4 Secțiune comentarii.....	31
4 Detalii tehnice.....	32
4.1 package.json.....	32
4.2 webpack.config.js.....	33
5 Direcții viitoare în dezvoltarea aplicației.....	35
5.1 Suport platformă cloud.....	35

5.2 Capacitate de ataşare documente şi fişiere.....	35
5.3 Încărcare progresivă.....	35
5.4 Secţiune specială de statistică.....	35
6 Concluzie.....	36
Bibliografie.....	37

Introducere

Kanban este un sistem de planificare a sarcinilor derivat din noțiunea de *lean software development*¹. Această metodologie implementează o mulțime de șapte principii: eliminarea timpului mort, punerea accentului pe specializarea individului (învățarea continuă), luarea de decizii cât mai târziu posibil, livrarea produsului cât mai rapid posibil, transferarea responsabilităților către echipa de dezvoltare, crearea unei imagini de ansamblu concrete și implementarea integrității și a intuiției.

Sistemul a fost propus pentru prima oară în cadrul firmei producătoare de mașini Toyota, de către inginerul industrial Taiichi Ohno, luând inspirație din modul în care magazinele alimentare își reîmprospătau rafturile bazat pe cererea creată de clienți. Acest model a devenit popular abia în timpul recesiunii globale din 1970, în mod inițial pentru a reduce costurile și utilizarea de mașinărie costisitoare. În prezent, compania Toyota folosește sistemul kanban pentru a detecta impedimentele din procesele de fabricație, dar și pentru a descoperi oportunități noi de optimizare. David J. Anderson a stabilit faptul că un sistem kanban poate fi împărțit în cinci componente: semnale vizuale, coloane, limită de lucru, un punct de comitere și un punct de livrare.

Semnalul vizual este reprezentat de către tichet. Echipa ce folosește sistemul scrie toate sarcinile pe care trebuie să le îndeplinească pe astfel de tichete, cât mai concis și mai concret, de obicei, un singur obiectiv reprezentând un singur tichet.

Cea de a doua componentă sunt coloanele care au rolul de a găzdui tichetele și reprezintă un tip de activitate din cadrul procesului de lucru. După cum vom vedea în cadrul acestei lucrări de licență, ele pot fi denumite precum “În Progres” sau “Completate”, și în mod natural, vor conține tichetele la care se lucrează în mod curent, sau care sunt completate.

Limita de lucru reprezintă numărul maxim de tichete care pot exista în același timp într-o coloană. Când se atinge această limită, echipa trebuie să-și schimbe focusul asupra lor și să îndeplinească obiectivul descris de către acestea, pentru a fi mutate în următorul stadiu al fluxului de lucru. Aceasta limită este esențială pentru a detecta impedimentele din cadrul fluxului și a descoperi dacă munca depășește efortul pe care echipa îl poate depune. În mod ideal, limita propusă de metodologia kanban este de maxim trei tichete, dar mecanismul implementat în această lucrare de

¹ Set de principii și practici din domeniul dezvoltării de programe creat în anul 2003 de către Marry și Tom Poppendieck

licență lasă la îndemâna utilizatorilor de a modifica această limită ori de câte ori este necesar pentru a acomoda nevoile unui proiect.

Punctul de comitere este locul unde clientul și echipa de dezvoltare își adaugă ideile și conceptele legate de proiect, de unde vor fi preluate mai departe de echipă și vor fi implementate.

Punctul de livrare este de obicei sfârșitul procesului de dezvoltare și etapa când produsul se află în mâna clientului.

Scopul echipei este să treacă tichetele din punctul de comitere, prin toate stadiile procesului de implementare și în final în punctul de livrare.

Peste toate aceste elemente se află tabla kanban, ustensila utilizată pentru a implementa paradigma kanban și care poate fi văzută ca și recipient. Rolul ei este de a oferi o imagine de ansamblu celui care o accesează, pentru a înțelege cât mai ușor stadiul în care se află proiectul.

Tablele kanban pot fi implementate atât fizic cât și electronic, fiecare având avantajele și dezavantajele lor. Tabelele fizice sunt ușor de construit, și există în mod perpetuu. Una sau mai multe echipe pot să contribuie la unul sau mai multe proiecte pe aceeași tabelă, iar membrii unei alte echipe pot observa modul de lucru al acesteia, și astfel să poată utiliza aceleași principii pentru a optimiza procesele din cadrul proiectului lor. Dezavantajul major este cel al erorii umane și dacă aceasta nu este observată din timp, fluxul de lucru este compromis. De exemplu, dacă o funcționalitate legată de securitatea unei baze de date este descrisă într-un tichet și acesta este mutat în coloana finală a tabelii, toți participanții unui proiect vor considera acea funcționalitate completă. În lipsa unui mecanism care să ateste că acel tichet a fost mutat în mod corect, cei care vor analiza tabela vor fi induși în eroare.

Pe de cealaltă parte, o tabelă implementată în mod electronic poate avea un astfel de mecanism de monitorizare și pot fi accesate la distanță oricând și de oricine este înscris la ea. Din păcate, de-a lungul timpului se poate dezvolta o practică de izolaționism, participanții punând din ce în ce mai mult accentul pe propriile tichete, excluzând imaginea de ansamblu. Cel mai mare avantaj este că o astfel de tabelă poate fi modificată și configurată după preferințele celor ce lucrează pe proiect, se pot adăuga filtre, se pot controla atât limita de lucru, cât și numărul de coloane, se poate oferi și elimina accesul unui anumit cont înregistrat în sistem.

Toate aceste mecanisme prezentate mai sus sunt implementate în aceasta lucrare de licență, și se va putea observa cum aceste concepte pot fi utilizate nu numai de un număr nelimitat de persoane, ci pot suporta și un număr mare de proiecte, coloane și tichete.

Contribuții

Primul pas în elaborarea acestei lucrări de licență a fost reprezentat de procesul de documentare și informare asupra metodologiei kanban și analizarea altor platforme care utilizează același sistem.

În momentul elaborării acestui proiect, există șapte platforme care oferă o implementare a paradigmei kanban, dar de obicei o regăsim utilizată în ansamblu cu metodologia *scrum*², cea mai utilizată aplicație de acest gen fiind JIRA, aplicație proprietară Atlassian.

Următoarea contribuție a fost conceperea unei arhitecturi care să stea la baza platformei și să ofere utilizatorilor să o folosească în mod eficient și intuitiv. Rezultatul este o structură compusă din trei elemente, puse în pereche printr-o relație de cardinalitate unul-la-mai-mulți³, denumite proiect, modul și tichet/raport problemă.

Cea mai importantă etapă în dezvoltarea platformei descrise a fost cercetarea tehnologiilor necesare. Conceptul a fost gândit pentru a oferi utilizatorilor acces de pe stațiile de lucru personale (dispozitivele *desktop*), cât și de pe dispozitivele mobile pe care aceștia le au asupra lor. O mare parte din funcționalitățile pe care le oferă aplicația *desktop* sunt accesibile și din aplicația mobilă, cu excepția unora care nu sunt suportate de niciun navigator de pe un dispozitiv mobil. Clientul și serverul sunt în totalitate dezvoltate în limbajul de programare JavaScript. Programul sursă a serverului utilizează biblioteca Node.js împreună cu cea ExpressJS, pentru o implementare completă a stilului arhitectural REST.

Pe partea de interfață, ReactJS este folosit pentru capacitatea acestuia de a crea și defini componente, care au o mare abilitate de reutilizare. Despre aceste tehnologii se va intra în detaliu în capitolele ce urmează.

Din punct de vedere al securității, se utilizează sistemul de JSON *web token*, mecanism ce oferă posibilitatea unui utilizator de a folosi mai multe aplicații prin metoda *single sign on*⁴. În această lucrare, acest obiect se utilizează mai ales pentru a securiza toate rutele prin care serverul și clientul schimbă informații. Criptarea parolilor utilizatorilor se face prin intermediul bibliotecii bcrypt, care dispune de o funcție de criptare unidirecțională.

² Metodologie asemănătoare kanban, bazată pe împărțirea muncii pe perioade de timp numite *sprint*-uri, care durează în medie două săptămâni

³ Relație între două entități în care un element de tip A poate fi legat de mai multe elemente de tip B, dar elementul de tip B este legat la un singur element de tip A

⁴ Modalitate ce face posibilă autentificarea pe mai multe platforme utilizând o singură mulțime de informații

Aplicația mai dispune de un administrator care poate crea și modifica orice resursă din sistem exceptând profilele celorlalți utilizatori. Cei care nu au acest drept sunt constrânși a modifica doar resursele pentru care sunt participanți.

Motivație

Ideea acestei aplicații a originat din experiența mea cu o platformă asemănătoare care implementează același principiu, pe care o utilizez la locul de muncă. Primul aspect pe care l-am observat a fost cât de eficient și de ușor este pentru fiecare angajat să urmărească propria muncă și pe cea a colegilor săi, dar această platformă aduce cu sine și dezavantaje.

Primul aspect observat a fost complexitatea mare pe care această platformă o conferă. Cele menționate mai sus încearcă să suporte o categorie mare de companii, fiecare folosind propriul sistem de management și de aici rezultă complexitatea ridicată a aplicației. Angajații care se confruntă pentru prima oară cu o astfel de platformă nu pot apela doar la intuiție, principiu discutat în introducere, ci au nevoie de o pregătire pentru a înțelege conceptele implementate și mecanismele de care pot dispune.

Pe de altă parte, chiar dacă o aplicație de acest gen propune o structură simplă și ușor de utilizat, există posibilitatea ca o mare parte din funcționalitățile pe care le oferă să nu fie pe deplin gratuite. Acest lucru reprezintă o provocare pentru o singură persoană care doar își dorește un mecanism prin care să-și planifice sarcinile și să-și înregistreze munca, sau chiar pentru o firmă mică de antreprenariat care are doar câțiva angajați.

Aplicația mea propune o modalitate prin care se oferă mult control pentru proiecte mici persoanelor care nu sunt experimentate cu metodologia kanban și un start în lumea managementului.

1. Arhitectură

Din punct de vedere arhitectural, există mai multe aspecte care trebuie luate în considerare.

Aplicația este alcătuită din trei părți, un server, o bază de date și un client. La rândul lor, acestea au o arhitectură internă complexă.

1.1 Securitate

La nivel de securitate, există trei mecanisme care asigură protecția datelor comunicate între server și client.

SSL(*Secure Sockets Layer*) este un standard de securitate care creează o conexiune securizată între părți, bazată pe două chei criptografice, una privată și una publică.

Certificatul conține diferite detalii despre domeniul pe care va fi activ: numele celui care deține domeniul, adresa sa și o dată de expirare. Atunci când se accesează pagina web, primul lucru verificat este termenul limită al certificatului, apoi se vor compara cele două chei pe care le conține, iar navigatorul va alerta utilizatorul în cazul în care nu se respectă standardul.

A doua metodă de securitate este standardul JSON *web token*(JWT) ce definește o metodă compactă de a transmite în mod securizat informații între server și client ca și obiect JSON, informație care este de fiecare dată semnată utilizând un secret sau pereche de chei criptografice ca și în cazul certificatelor SSL.

Diferența dintre cele două mecanisme este că SSL este utilizat pentru a transmite informațiile de natură sensibilă care se folosesc pentru autentificarea sau înregistrarea utilizatorului în sistem.

Ceea ce se transmite sunt e-mail-ul și parola, în format text simplu. E-mail-ul ajută la extragerea parolei criptate din baza de date, pentru a fi comparată cu cea introdusă în momentul autentificării.

JSON *web token* este folosit pentru a securiza toate rutele de acces de pe server. De fiecare dată când se cer sau se modifică date, o funcție *middleware*⁵ va verifica validitatea obiectului, și va da un răspuns pozitiv sau negativ în legătură cu integritatea acestuia.

Al treilea mecanism de securitate este biblioteca bcrypt menționată mai sus prin intermediul căreia se criptează parolele sau alte date înainte de a fi introduse în baza de date. Prin utilizarea

⁵ Funcție responsabilă cu traducerea informației dintre două părți participante

unei funcții de criptare unidirecțională se asigură securitate optimă pentru aceste detalii cu caracter sensibil.

1.2 Server

Toate datele care sunt introduse în baza de date sunt consumate prin intermediul unui server ce implementează stilul arhitectural REST.

Acest standard propune separarea implementării clientului de cea a serverului, cele două fiind create să ruleze în mod independent una față de cealaltă.

Spunem că sistemele care implementează această paradigmă nu au stare, ceea ce înseamnă că serverul nu necesită vreo informație despre starea pe care clientul o are în mod curent și vice versa. Astfel, serverul și clientul pot înțelege orice mesaj care se recepționează sau se transmite, fără a ține cont de mesajele anterioare.

Comunicația dintre server și client se face prin cereri și răspunsuri. Cererile făcute de client pot recupera sau pot modifica datele acestuia. În general, o cerere este descrisă de un verb HTTP, ce semnifică tipul operației ce trebuie îndeplinită, un antet(*header*), ce descrie tipul de informație conținută în cerere, o cale relativă către resursa cerută și, opțional, un mesaj ce conține datele de procesat.

Verbele HTTP cele mai comune sunt GET, POST, PUT și DELETE. Implementarea serverului este construită peste o arhitectură ce utilizează doar verbele GET și POST. Primul verb este de obicei utilizat pentru a recupera date din baza de date pe care serverul o accesează, iar cel de-al doilea verb reprezintă alte două posibile operațiuni. În cazul operațiunii de modificare, două tipuri de date sunt transmise, cea care are rolul de identificator și valoarea care va suprascrie ceea ce deja se regăsește în baza de date.

A doua operațiune reprezintă crearea de resurse noi, și de obicei, singurul lucru transmis este valoarea ce trebuie introdusă.

Un utilizator are posibilitatea de a se înregistra în sistem, utilizându-se de e-mail, parolă și alte detalii. Aceste două informații sunt transmise la server, unde se verifică dacă e-mail-ul există deja în baza de date. În acest caz, se va transmite un *status* și un mesaj de eroare către client, unde va fi afișat pe ecran. În caz contrar, parola va fi criptată, iar aceasta, împreună cu e-mail-ul, vor fi înregistrate în baza de date.

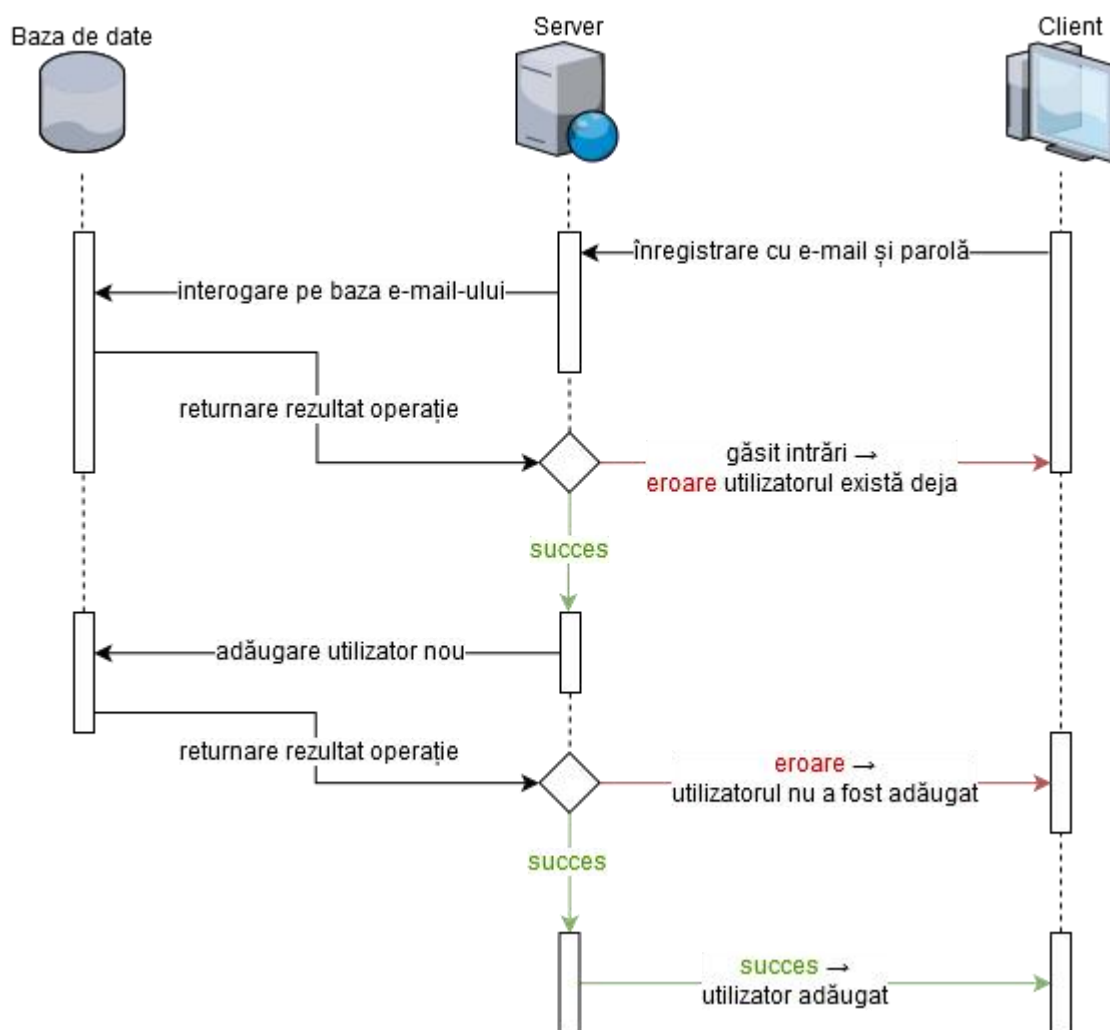


Fig. 1 - Înregistrarea unui utilizator

După ce un utilizator este adăugat în sistem, el se poate autentifica utilizând valorile pe care le-a specificat pe pagina de înregistrare. E-mail-ul și parola sunt trimise către server, unde se extrag informațiile din baza de date. Dacă nu există un rezultat pentru această interogare, un mesaj de eroare este trimis la client care specifică dacă e-mail-ul utilizatorului există sau nu în baza de date. În caz contrar, informația recuperată este parola criptată și este comparată cu valoarea de tip text simplu introdusă de către utilizator. În cazul în care acestea diferă, se trimite un mesaj de eroare către utilizator, pentru a reintroduce parola.

Altfel, se creează obiectul JSON de validare, utilizând ca informație de bază e-mail-ul, și alte informații care sunt necesare mai departe pe partea de *front-end*. Odată creat, acesta este trimis la client, unde trebuie memorat în mediul de stocare local (de exemplu: *cookies*⁶). Acesta va fi folosit de către server de fiecare dată când este accesată o rută pentru a se comunica informații.

⁶ Mediul de stocare local al unui navigator ce reține date într-o structură cu perechi cheie-valoare

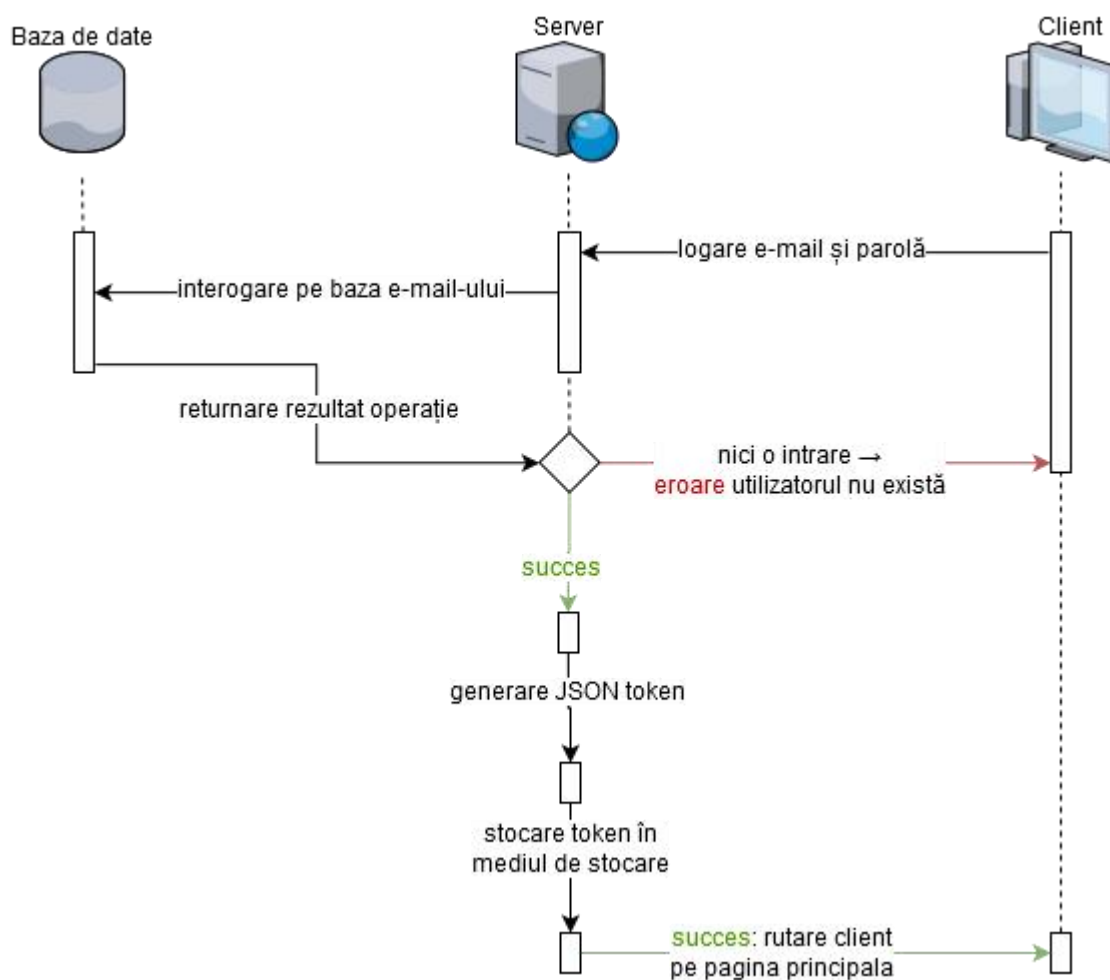


Fig. 2 - Secvența de autentificare

Ruta utilizată pentru deconectarea unui utilizator operează în mod simetric opus față de cea de autentificare. Atunci când utilizatorul se va deconecta, obiectul JSON care există în mediul de stocare al navigatorului se va configura cu un timp de expirare foarte scurt (de ordinul milisecundelor), ceea ce va duce la distrugerea acestuia, iar apoi utilizatorul va fi rutat către pagina de autentificare.

Restul rutelor sunt folosite pentru a recupera sau pentru a manipula datele din baza de date. O astfel de rută extrage date sau le modifică după natura rutei implementate.

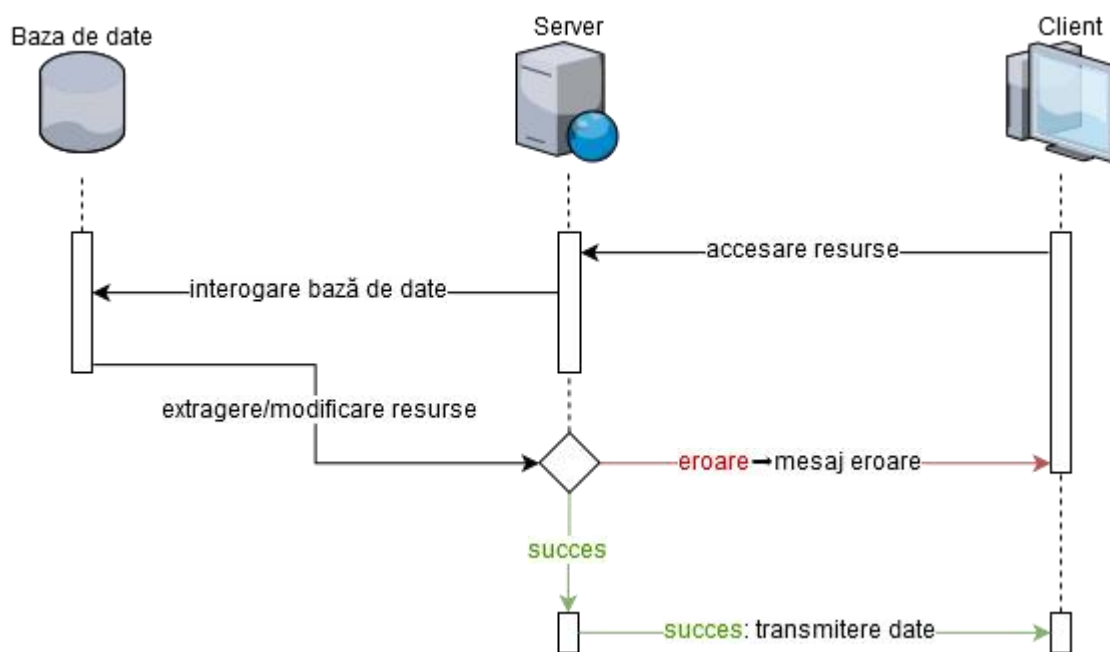


Fig. 3 - Cerere simplă de date de către client

1.3 Client

Clientul care interacționează cu serverul pentru a prelua sau modifica date este creat peste o arhitectură de tip *Single Page Application*, în care aplicația este doar o singură pagină HTML, construită pe bază de componente, cu ajutorul bibliotecii ReactJS.

Aceste componente pot avea o stare internă, dar primesc și un obiect JSON prin intermediul unei metode, numit *props*, iar alterarea uneia dintre aceste două elemente va conduce la rescrierea componentei în arborele DOM, cu noile valori.

Fiind construită ca și o singură pagină, aplicația se comportă dinamic la toate acțiunile realizate de către utilizator.

Componentele care fac parte din structura paginii vor exista în permanență pe toată perioada operării în cadrul aplicației, iar altele vor fi distruse și vor fi înlocuite odată ce își vor îndeplini rolul.

În mod normal, ele comunică în mod direct, printr-o relație de tip părinte-copil, în care părintele, la nevoie, va transmite componentei copil starea sa internă și/sau informațiile externe pe care și el, la rândul său, le primește de la o componentă superioară, dar există și necesitatea transmiterii de informații între componentele care nu comunică direct.

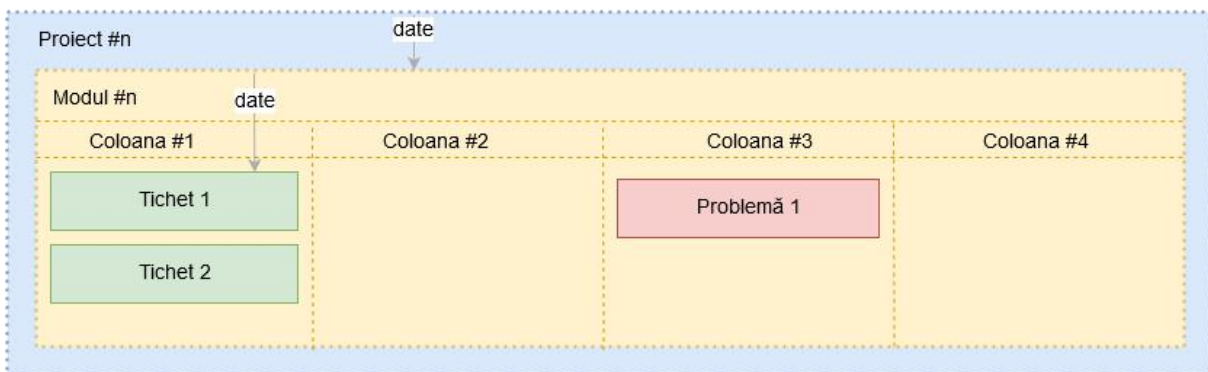
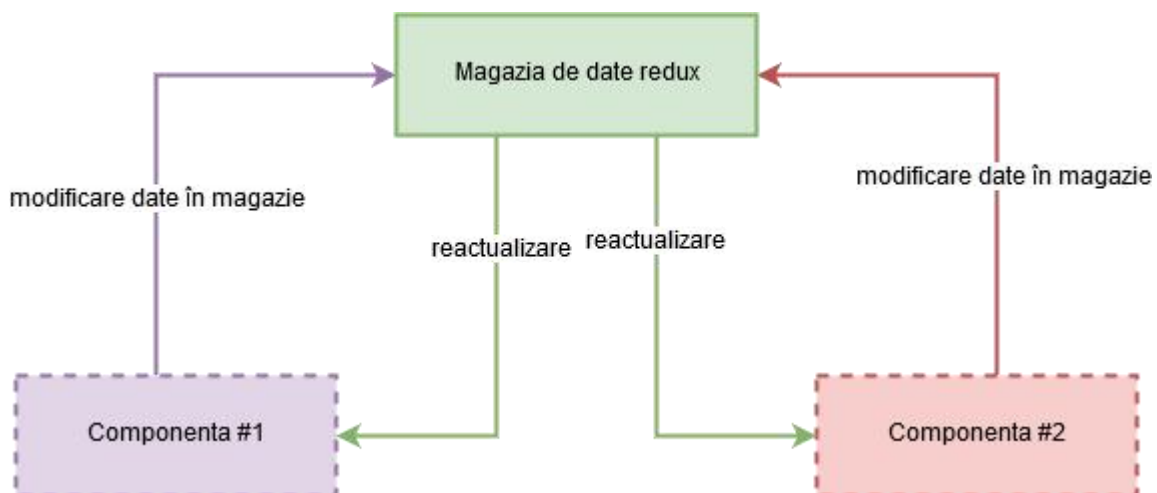


Fig. 4 - Componenta Proiect, relația dintre aceasta și copii săi, Modul și Tichet

În acest caz intervine un mecanism de înmagazinare a datelor implementat cu ajutorul bibliotecii React Redux. Prin intermediul acesteia se definește un obiect de tip JSON, care are rolul de a stoca date și care există dinafara componentelor descrise anterior, dar care poate fi accesat de oricare dintre ele, atât timp cât ele sunt conectate la acesta utilizând o funcție specială, care este definită un sub numele *connect* și încapsulează componenta din două motive. Unul dintre ele este pentru a injecta direct datele pe care le conține în obiectul *props* al componentei, aceasta operație îndeplinindu-se prin intermediul uneia dintre funcțiile de tip *callback* pe care funcția *connect* le poate accepta. Componenta poate alege să filtreze datele pe care le primește din magazie, accesând membrii obiectului JSON după numele cheii. Prin al doilea motiv se oferă posibilitatea componentei de a defini metode prin care poate modifica magazia de date.



Componenta principală este cea a unui proiect și din punctul de vedere al interfeței, ea nu reprezintă decât un recipient pentru elementele ce se vor afla în interiorul acesteia, dar din punct de vedere arhitectural, ea reprezintă primul nivel de granularitate a elementelor din pagină. Pe următorul nivel din punct de vedere ierarhic se află modulul. Acesta se regăsește în interiorul unui proiect, și reprezintă, în mod natural, un modul *software* ce trebuie dezvoltat în cadrul proiectului. La rândul lui el reprezintă un recipient pentru următoarele elemente ce se află în

interiorul său. Spre deosebire de proiect, un modul este împărțit în patru coloane, ce vor reprezenta progresul unui tichet sau al unui raport de problemă pe tot parcursul vieții proiectului. Aceste coloane sunt reprezentarea vizuală a elementului kanban cu același nume.

A treia componentă este tichetul. Acesta este elementul descris în metodologia kanban ca fiind semnalul vizual, și descrie în mod succint o sarcină de îndeplinit pentru a aduce implementarea modului la bun sfârșit.

A patra componentă este raportul de problemă. El se creează abia atunci când modulul este în decurs de testare. Odată găsită o neregulă, aceasta poate fi semnalată prin intermediul acestui mecanism. Ea poate fi atașată la un tichet, dacă aceasta este sursa problemei, sau nu, în cazul în care problema are mai multe surse.

Următoarea componentă este cea responsabilă pentru configurarea de elemente noi. În funcție de tipul elementului, vor apărea diferite câmpuri pe care un utilizator poate să le completeze cu informații. Se vor putea introduce detalii precum numele elementului, descrierea acestuia și ziua limită pentru îndeplinirea sarcinii. În cazuri speciale, se pot seta un proprietar sau participanți, putând alege dintre toți utilizatorii care există în sistem.

Toate aceste aspecte pot fi modificate mai târziu prin intermediul componentei de editare a unui element. Un element care apare în plus în această componentă, în cazul tichetului și raportului de problemă, este secțiunea de comentarii și raport muncă. În secțiunea comentarii, utilizatorii își pot exprima opinia despre acea sarcină, iar în secțiunea raport de muncă, un participant poate să înregistreze munca pe care a depus-o și numărul de ore petrecute.

Un alt rol al clientului este de a primi și utiliza elementele de securitate oferite de către server. Un prim astfel de element este cel al obiectului de autentificare JSON, pe care clientul trebuie să-l stocheze în mediul de stocare local (de exemplu: *cookies*), dar de asemenea trebuie să și extragă informații din acesta, cea mai importantă fiind e-mail-ul utilizatorului care este autentificat în mod curent.

1.4 Baza de date

În următoarele pagini voi descrie structura bazei de date, relațiile dintre tabelele acesteia și modul în care interacționează pentru a oferi utilizatorului posibilitatea de a utiliza platforma în mod cât mai sigur și mai intuitiv.

După cum se poate observa în Fig.1, un proiect joacă rolul de recipient pentru una sau mai multe componente.

Acesta are un identificator(*id*) unic în baza de date, un nume(*name*), care va fi afișat utilizatorului în interfața grafică a navigatorului, o descriere(*description*), ce este opțională, dar ar trebui să ofere informații despre proiectul respectiv, și o zi de început(*startDate*), care reprezintă momentul în care a fost creat. De precizat este faptul că doar utilizatorii care au drepturi de administrator pot să creeze un proiect nou, sau să modifice unul deja existent.

Pe nivelul următor se află modulul, element responsabil să găzduiască unul sau mai multe tichete sau rapoarte de problemă. Pe lângă câmpurile descrise în tabela proiectelor, un modul poate avea ca și cheie străină identificatorul unui proiect, unei versiuni și unei categorii.

Categoria reprezintă denumirea tipului sau naturii acestui element din proiectul respectiv, și are rol informativ pentru utilizator.

Versiunea reprezintă un punct izolat în timp a stadiului în care produsul se află, dar trebuie utilizat împreună cu un document de suport în care acestea sunt specificate cu datele limită și alte aspecte.

Ziua limită(*dueDate*) semnifică ziua până când tichetele care sunt conținute de această componentă ar trebui să fie completate, dar și acest detaliu este recomandat a fi utilizat cu rol îndrumător.

Proprietarul(*owner*) este utilizatorul care deține dreptul de a modifica detaliile unei componente, iar un alt utilizator poate prelua acest drept doar dacă îi este oferit de proprietarul curent. Singurul utilizator care totuși poate modifica detaliile componente, fără să fie proprietarul ei, este cel care deține drepturile de administrator.

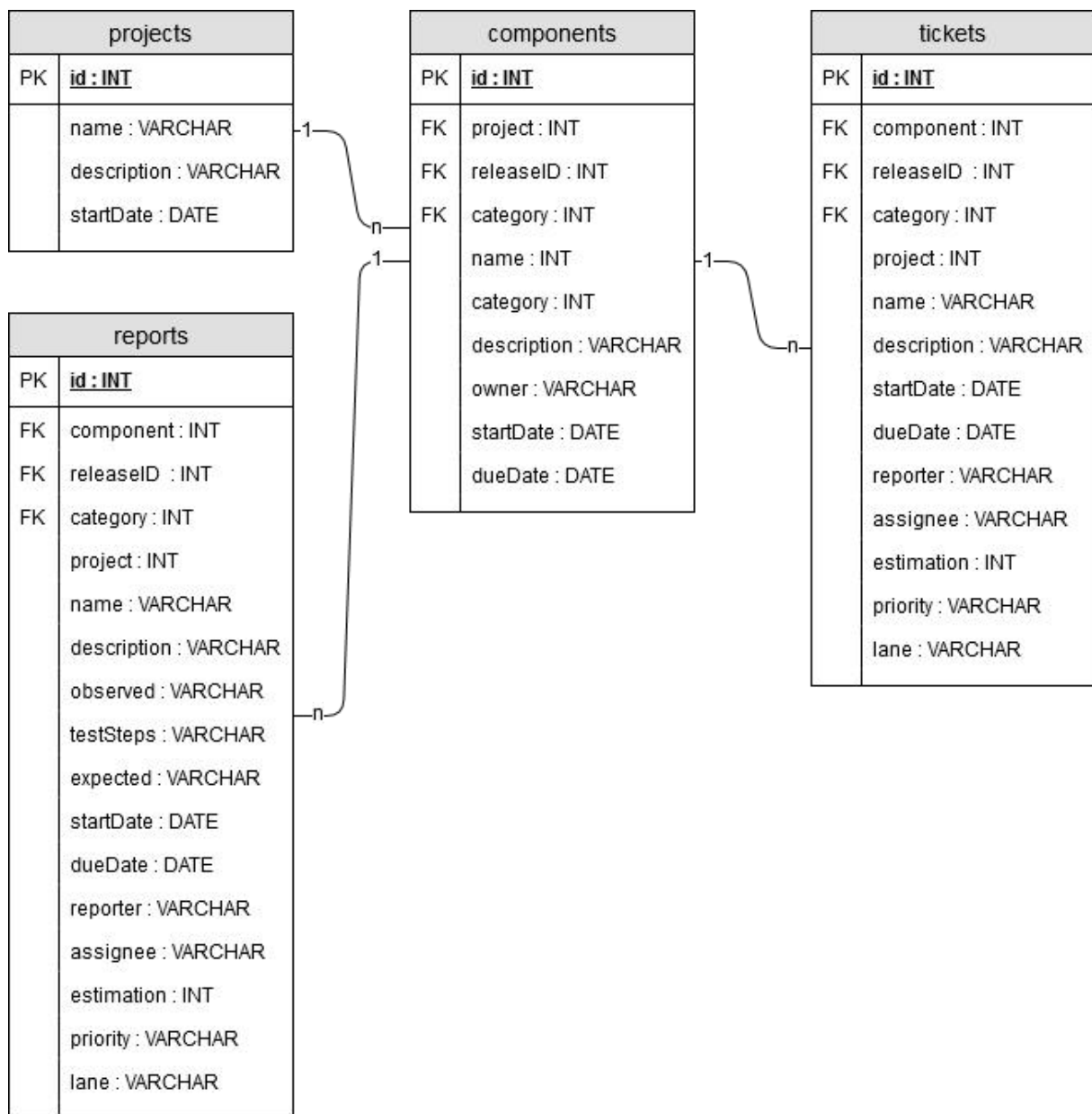


Fig. 5 - Relațiile unuia-la-mai-mulți dintre proiect, componentă, raport problemă și tichet

Tichetul și raportul de problemă au și ele câmpuri comune, mai puțin cele care se găsesc în raport cum ar fi pașii de testare(*testSteps*), în care un utilizator scrie etapele care au rezultat în comportamentul observat(*observed*) care trebuie să difere față de comportamentul așteptat(*expected*), ce este modul în care funcționalitatea descrisă de tichet ar trebui să opereze în mod normal.

Alte atribute de notat sunt supervisorul(*reporter*), cel care în mod original a creat tichetul, și asignatul(*assignee*), cel care va lucra la sarcina respectivă.

Toate aceste tabele sunt legate între ele printr-o constrângere pe operația de ștergere prin intermediul cheilor străine. Comportamentul propus este ca atunci când o intrare din această

coloană este eliminată, această operație să declanșeze ștergerea intrărilor din tabelele componentelor, a tichetelor și a rapoartelor. În acest mod se asigură că niciun element nu va rămâne în baza de date fără părinte, și nu va compromite modul în care aceste elemente vor fi afișate pe interfața grafică utilizatorului.

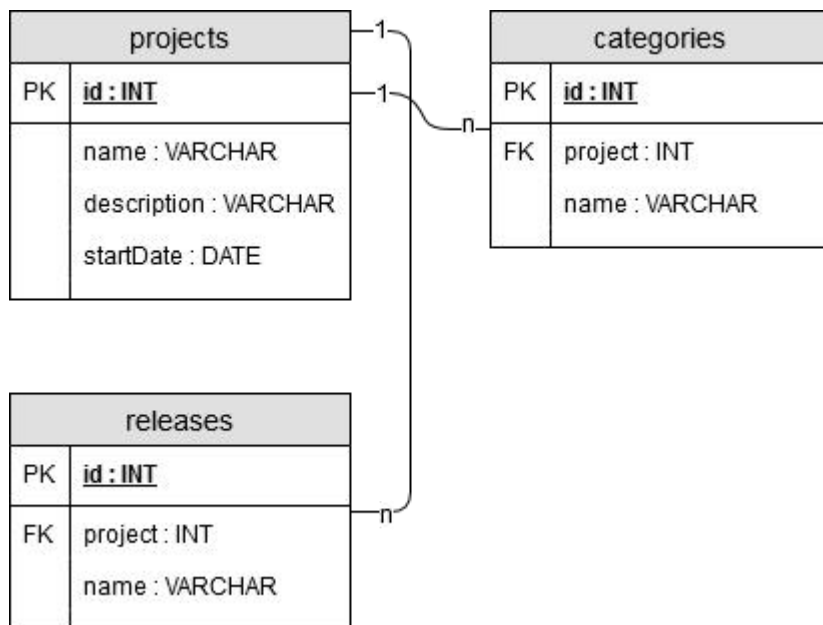


Fig. 6 - Relația dintre proiect, categorii și versiuni

Aceeași relație o putem descoperi între proiect, categorii(*categories*) și versiuni(*releases*).

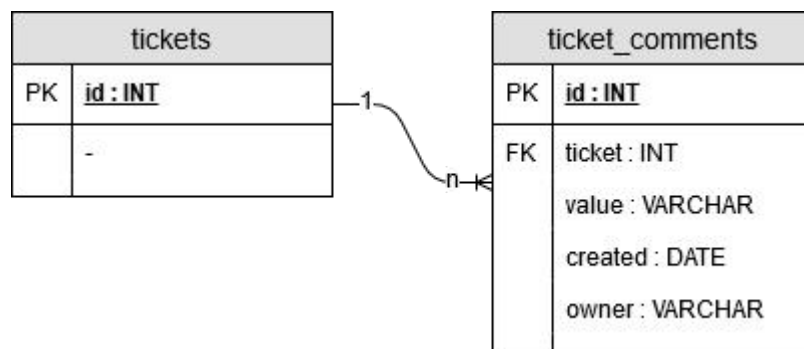


Fig. 7 - Relația dintre tichete și comentarii

Un tichet poate avea unul sau mai multe comentarii. Sunt folosite pentru a oferi utilizatorilor posibilitatea de a-și exprima opiniile în legătura cu orice aspect legat de sarcina descrisă în cadrul tichetului. Comentariul este identificat în baza de date folosind un identificator(*id*), care este unic. Valoarea(*value*) este textul introdus de către utilizator. Ziua de creare este generată în mod automat atunci când comentariul este introdus în baza de date, iar utilizatorul nu are acces la această valoare. Proprietarul(*owner*) este completat în mod automat prin extragerea utilizatorului care este autentificat în mod curent, și de asemenea, acest atribut nu poate fi influențat de cel ce introduce acest comentariu.

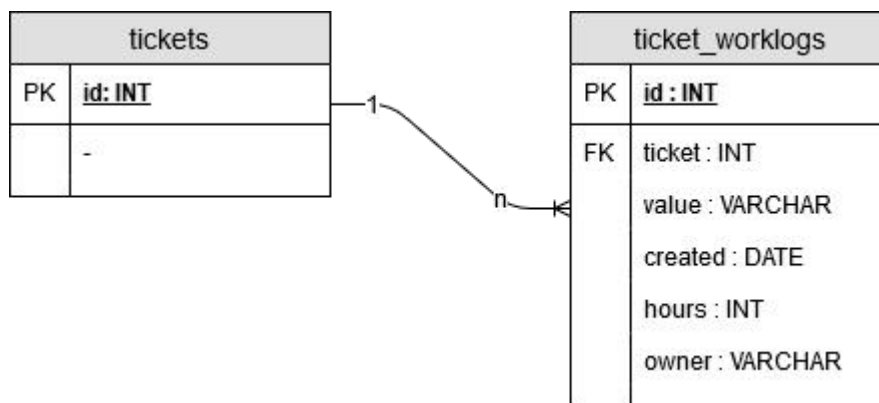


Fig. 8 - Relația dintre tichet și raport de muncă

Tabela rapoartelor de muncă(*ticket_worklogs*) au structura asemănătoare cu cea a tabelii comentariilor. Singura diferență este reprezentată de coloana orelor înregistrate(*hours*), care reprezintă orele de muncă pe care o persoană le-a depus asupra sarcinii descrise de către tichet. Aceste ore sunt reprezentate grafic pe tichetele care se găsesc pe tabela kanban, și sunt puse în comparație cu orele estimate.

Aceleași relații le regăsim și în cazul rapoartelor de muncă.

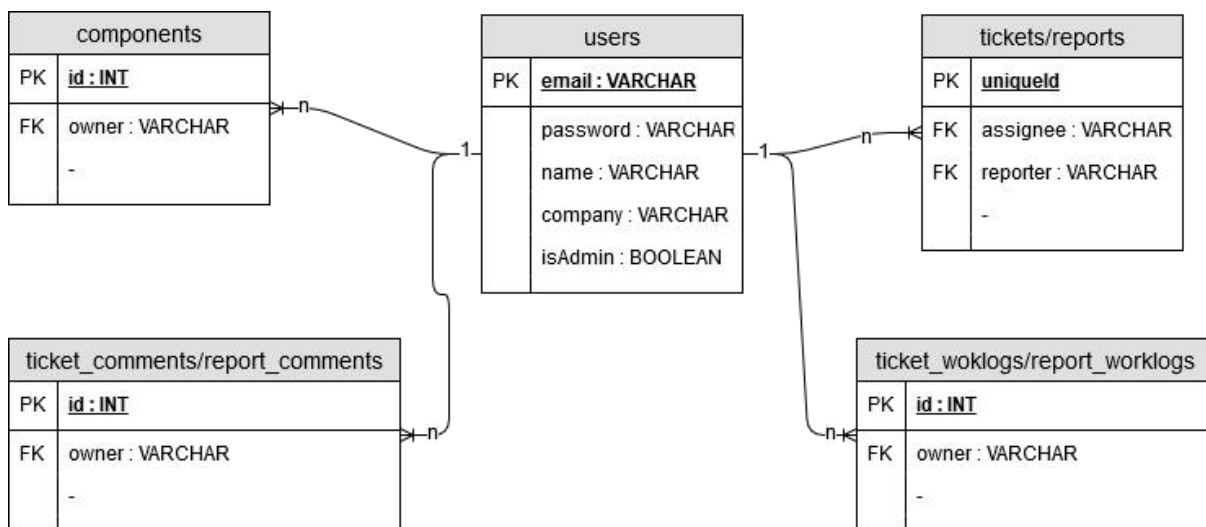


Fig. 9 - Utilizatorul și e-mail-ul ca și cheie străină

În figura de mai sus se prezintă toate tabelele care sunt influențate de tabela utilizatorilor(*users*). În baza de date, un utilizator este descris de un e-mail, care este cheie primară și unică, o parolă(*password*), un șir de caractere ce semnifică parola utilizatorului criptată de o funcție unidirecțională, un nume(*name*), care în mod ideal este numele real al utilizatorului și numele companiei(*company*) la care acesta lucrează. Ultimul câmp(*isAdmin*) semnalează dacă acest utilizator are sau nu drepturi de administrator, drepturi care îi oferă posibilitatea să modifice majoritatea câmpurilor din interfața grafică.

Atunci când se creează o componentă sau un tichet, în mod opțional putem să specificăm, în cazul componentei, care este proprietarul(*owner*) acesteia, iar în cazul tichetului, care sunt participanții, unul din ei fiind supervizorul(*reporter*), iar celalalt fiind asignatul(*assignee*). În ambele situații, dacă un utilizator a fost specificat într-un câmp din aceste două elemente, dorim ca atunci când este eliminat din sistem, câmpul să rămână eliberat, ca altă persoană să îi preia locul la un moment dat. Din punctul de vedere al bazei de date, câmpul din aceste tabele va fi setat pe valoarea NULL.

În cazul tabelor comentariilor și rapoartelor de muncă, intrările care au ca și cheie străină utilizatorul șters, vor fi și ele la rândul lor șterse.

2 Tehnologii utilizate

2.1 ReactJS

ReactJS este o bibliotecă care se bazează pe limbajele JavaScript și JSX pentru a oferi un mecanism puternic de creare de interfețe, utilizabile atât în navigatoarele bazate pe tehnologia HTML5, dar și pentru aplicații mobile.

```
<div id="aplicatieReact"></div>
<script type="text/babel">
  class Mesaj extends React.Component {
    render() {
      return <h1>{this.props.mesaj}</h1>
    }
  }
  ReactDOM.render(<Mesaj mesaj="Salut!" />,
document.getElementById('aplicatieReact'));
</script>
```

Tabela 1 - Exemplu rudimentar de utilizare a unei componente

Caracteristica principală este orientarea pe componente, fiecare fiind capabilă să primească obiecte JSON externe numite *props*. Acestea sunt informații pe care componentele le moștenesc între ele.

Componentele au o stare internă(*state*) care poate fi transmisă către componentele copil.

ReactJS propune un mecanism de DOM⁷ virtual, prin care se creează în memorie o structură de date ce reprezintă componenta, care, odată ce starea internă suferă schimbări, această structură este analizată, și toți membrii care utilizează obiectul stării vor fi actualizați în mod eficient.

Din faptul că o componentă are o stare internă rezultă faptul că aceasta dispune de un ciclu de viață, și există anumite funcții *hook*, care sunt apelate în anumite puncte cheie din timpul de viață a acesteia. Cele mai notabile funcții sunt cea apelată înainte ca o componentă să fie actualizată(*shouldComponentUpdate*), cea care este apelată după ce componenta a fost creată(*componentDidMount*, *componentWillMount*) sau va fi distrusă(*componentWillUnmount*) și funcția de randare(*render*).

⁷ Document Object Model


```

class ComponentaParinte extends React.Component {
  state = { color: 'green' };
  render() {
    return (
      <ComponentaCopil color={this.state.color} />
    );
  }
}

```

Tabela 2 - Componenta ce își transmite starea internă

După cum putem observa în Tabela 2, *ComponentaParinte* transmite toată starea lui internă, colorată cu roșu, către moștenitorul ei, *ComponentaCopil*.

Toată interfața de care utilizatorul dispune este implementată utilizând mecanismul de componente al acestei biblioteci.

2.2 React Redux

Pentru programe complexe, mecanismul propus de ReactJS nu este de ajuns, atunci când se dorește transmiterea de date nu numai de la componenta părinte la componenta copil, ci și între orice alte două componente din aplicație.

React Redux propune crearea unei magazii de date, care există dinafara interfeței implementate cu ajutorul React, iar componentele din interfață trebuie apoi conectate la această magazie prin funcția denumită *connect*. Aceasta primește ca parametri două metode de tip *callback*⁸, unul din ele fiind folosit pentru a injecta datele conținute în magazie direct în obiectul props al componentei, iar cealaltă este utilizată pentru a declara metode noi folosite pentru a altera magazia externă.

În acesta lucrare de licență acest mecanism este folosit pentru a îndeplini mai multe roluri. Cea mai importantă funcție pe care o îndeplinește este memorarea obiectului de validare JSON pe care clientul îl primește de la server, și de a extrage e-mail-ul utilizatorului care este autentificat. Aplicația mai dispune de un mecanism de schimbare a temei vizuale, utilizatorul putând opta între o temă de culoare deschisă, potrivita pentru lucrul în medii luminoase, și o tema de culoare închisă, pentru a proteja ochii atunci când se operează într-un mediu cu prea puțină lumină.

⁸ Funcții oferite ca și parametru, apelate la un moment dat

Un alt rol pe care React Redux îl îndeplinește în acest proiect este reținerea valorilor booleene ale filtrelor selectate de către utilizator, chiar dacă acesta navighează pe altă pagină din aplicație.

2.4 Node.js

Node.js este o bibliotecă *open-source*, dezvoltată peste compilatorul de limbaj JavaScript Google Chrome V8, scoasă din contextul navigatorului și modificată pentru a permite rularea acesteia direct pe servere care promovează unificarea aplicației web cu cea server, spre deosebire de folosirea mai multor tehnologii în cadrul aceluiași proiect.

Node.js are o arhitectură orientată pe evenimente capabile să recepționeze și să ofere date în mod asincron. Prin acest timp de model se urmărește optimizarea debitului de date și capacitate de scalabilitate crescută.

Utilizând modelul Observator, operează pe un singur fir de execuție, utilizând apeluri non-blocante, astfel fiind capabil să suporte zeci de mii de conexiuni în același timp.

Dezavantajul principal este că se oferă prea puțin suport pentru scalabilitatea verticală, ceea ce înseamnă că performanța nu va crește odată cu creșterea numărului de nuclee din procesor.

2.5 ExpressJS

Deși Node.js este o bibliotecă rapidă și eficientă, cantitatea de cod scris necesară pentru configurarea unei singure rute este mult prea mare și prea complexă.

Rolul bibliotecii ExpressJS este de a reduce considerabil efortul de implementare, și propune modularizare și minimalism, prin implementarea de funcții reduse, unele care pot fi apelate de fiecare dată, pentru cererile ce vin de la client, și altele doar pe anumite tipuri de rute.

2.6 Bootstrap

Bootstrap este cel mai popular *framework* care conține șabloane HTML și CSS pentru dezvoltarea de site-uri și aplicații web, și se ocupă doar de dezvoltarea pe partea de *front-end*. În momentul scrierii acestei documentații, ultima versiune, care este folosită și în elaborarea acestei lucrări de licență, este v.4.2, și suportă ultimele versiuni de Google Chrome, Firefox, Edge(Internet Explorer), Opera și Safari.

Principalul rol al acestui *framework* este că suportă modelul *responsive web*, ceea ce înseamnă că o aplicație web poate fi dezvoltată atât pentru navigatorul de pe dispozitivele *desktop*, cât și pe

cel de pe dispozitivele mobile, deoarece interfața se modifică în funcție de lățimea ecranului folosit.

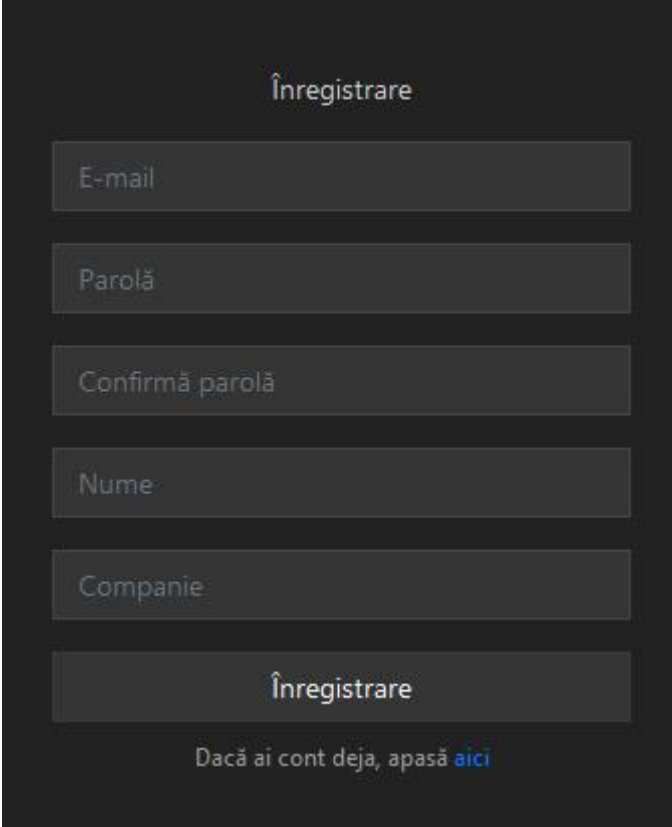
2.7 React DnD

Această bibliotecă este utilizată pentru a implementa mecanismul *drag-and-drop* pentru tichete și rapoarte problemă peste coloanele configurate în modul. Ideea de bază al acestui mecanism o reprezintă distrugerea tichetului de pe coloana sursă, și re-crearea lui pe coloana destinație, utilizând obiectul *props* al componentei React.

Exista două clase mai importante, ele fiind *DragSource*(sursa obiectului mutat), care încapsulează tichetul sau raportul și *DropTarget*, care este destinația pe care se va muta obiectul, adică coloana.

3 Utilizarea aplicației

3.1 Înregistrare, autentificare și recuperare parolă



The image shows a registration form titled "Înregistrare" (Registration) on a dark background. The form consists of several input fields and a submit button, all with a light gray background and rounded corners. The fields are labeled "E-mail", "Parolă" (Password), "Confirmă parolă" (Confirm password), "Nume" (Name), and "Companie" (Company). Below the fields is a large "Înregistrare" button. At the bottom, there is a link that says "Dacă ai cont deja, apasă aici" (If you already have an account, click here).

Înregistrare

E-mail

Parolă

Confirmă parolă

Nume

Companie

Înregistrare

Dacă ai cont deja, apasă [aici](#)

Fig. 10 - Formular înregistrare

The login form is titled "Autentificare" in white text on a dark background. It contains two input fields: "E-mail" and "Parolă", both with light gray placeholder text. Below these fields is a large, dark gray button labeled "Autentificare" in white. At the bottom, there is a blue link "Parolă uitată" and a line of text "Dacă nu ai cont, apasă [aici](#)" where "aici" is also a blue link.

Fig. 11 - Formular autentificare

The reset password form is titled "Resetare parolă" in white text on a dark background. It contains three input fields: "E-mail", "Parolă nouă", and "Confirmă parolă", all with light gray placeholder text. Below these fields is a large, dark gray button labeled "Resetare parolă" in white. At the bottom, there is a blue link "Înapoi la pagina de logare".

Fig. 12 - Formular resetare parola

Utilizatorii se pot înregistra, autentifica și modifica parola utilizând formularele din imaginile de mai sus. Cele mai multe detalii pe care utilizatorul le completează sunt în timpul înregistrării, unde trebuie să specifice, pe lângă e-mail și parola, date personale precum nume și compania la care este angajat. Aceste detalii vor fi folosite mai departe în anumite contexte din interfață precum profilul utilizatorului.

3.2 Pagina proiectului

3.2.1 Meniu

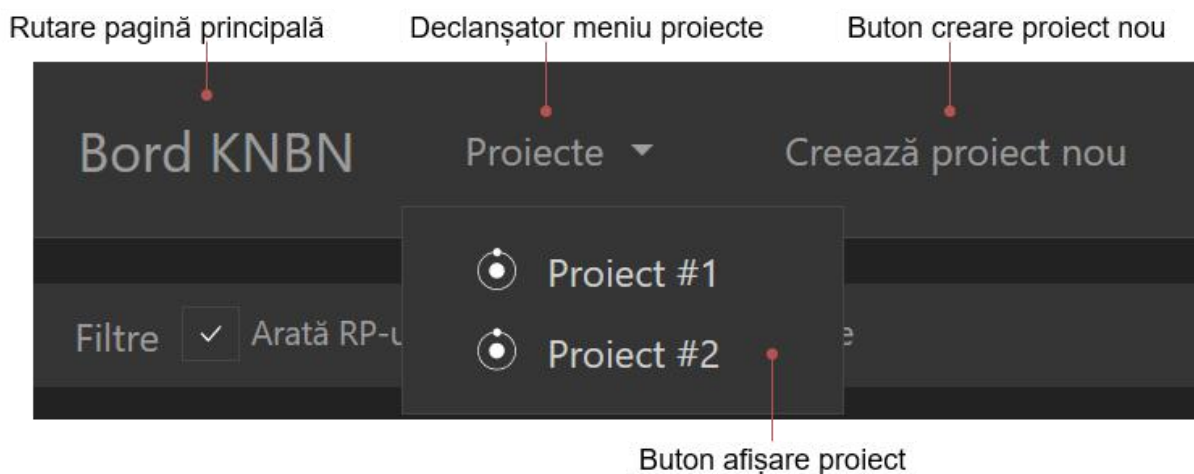


Fig. 13 - Secțiunea proiectelor

Primele trei butoane al meniului sunt dedicate vizualizării și creării de proiecte. Primul buton denumit **Bord KNBN** are rolul de a ruta rapid utilizatorul pe pagina principală.

Al doilea buton este un meniu cu deschidere în partea de jos. Acesta este populat cu numele proiectelor atunci când pagina este încărcată. Toate numele proiectelor din sistem vor fi prezente în acest meniu, iar selectarea unuia dintre acestea va trimite utilizatorul pe pagina de editare a acestuia.

Butonul **Creează proiect nou** este întotdeauna la îndemână pentru a crea un proiect, fiind accesibil doar utilizatorilor care au primit drept de administrator.

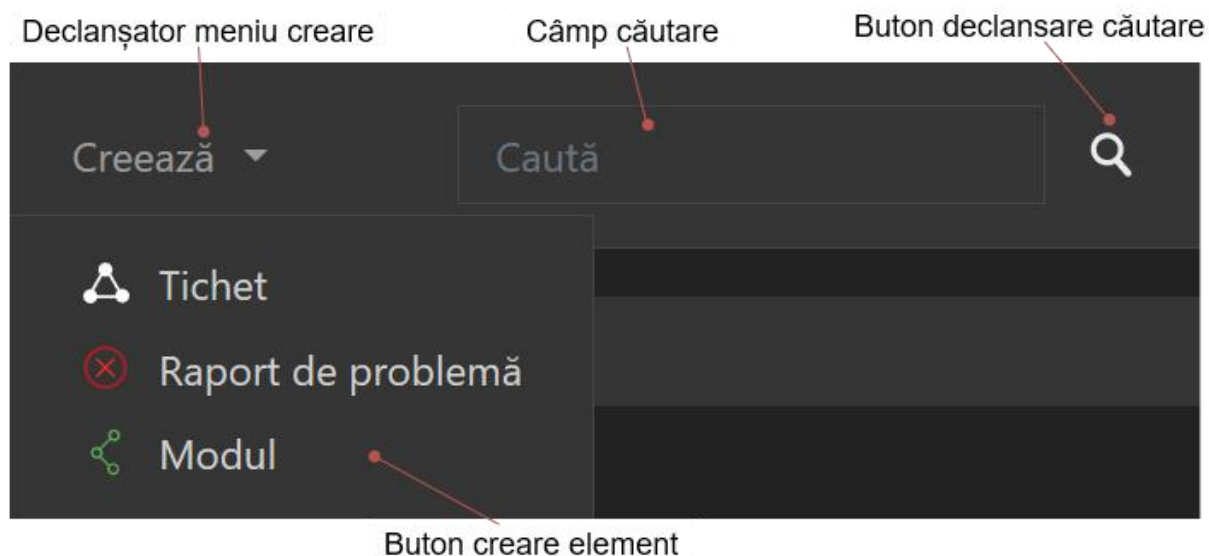


Fig. 14 - Secțiunea elementelor ce pot fi create și bara de căutare

În Fig. 14 putem vedea următoarele butoane din meniu. Butonul **Creează** are ca și opțiuni toate elementele pe care un utilizator le poate introduce în sistem.

Exista și un câmp de căutare care va ruta utilizatorul spre pagina elementelor găsite care conțin în nume textul introdus de acesta.

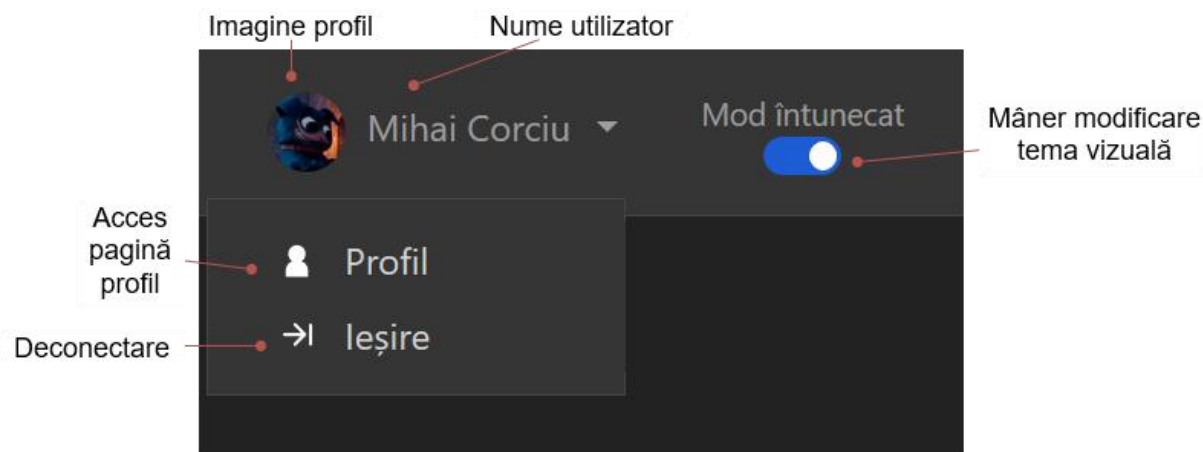


Fig. 15 - Profil și mânerul de schimbare temă vizuală

Fig. 15 afișează opțiunile pe care un utilizator le are în legătură cu contul său. El poate să se deconecteze de la aplicație, fiind rutat direct la pagina de autentificare, sau poate să acceseze pagina profilului dacă dorește să modifice detaliile sale personale.

Butonul **Mod întunecat** este un mâner utilizat pentru a schimba tema vizuală a întregii platforme.

3.2.2 Proiect, componentă și tichet

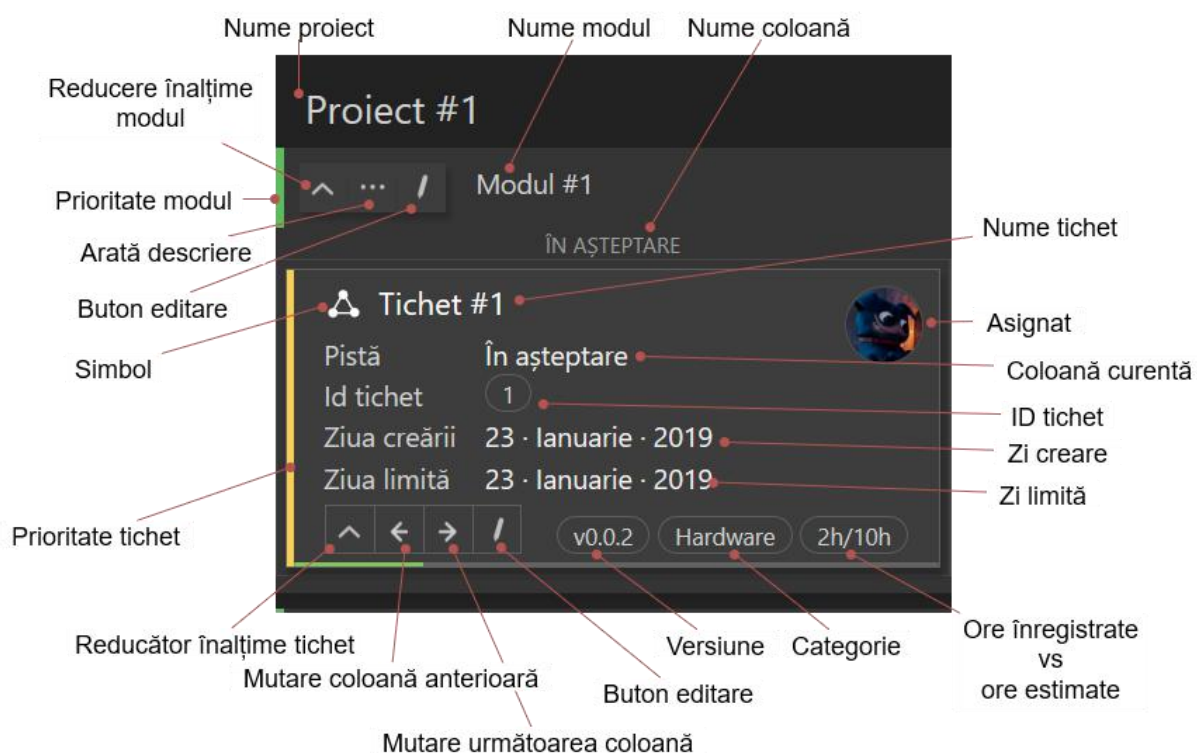


Fig. 16 - Proiect, Modul și Tichet

În Fig. 16 putem vedea principala structură care sta la baza acestei platforme. În această imagine, observăm relațiile dintre un proiect, un modul intern și sarcinile reprezentate ca și tichete.

Un proiect va fi afișat doar cu numele în cadrul acestui mediu de vizualizare. Modulul are în mod adițional un cadru ce conține butoane pentru reducerea înălțimii acestuia, afișarea descrierii și rutarea pe pagina de editare.

Un tichet conține mult mai multe informații decât cele două elemente prezentate mai sus. În acest mod, utilizatorului îi este mult mai ușor să găsească tichetul pe care îl interesează.

În această reprezentare, primele lucruri observate sunt simbolul, fiind o modalitate de a diferenția tichetul de raportul de problemă, numele și imaginea de profil a persoanei care are rolul de asignat.

În partea centrală, apar mai multe câmpuri descriptive despre coloana pe care tichetul se află în mod curent, identificatorul unic, ziua în care a fost creat și ziua limită până când trebuie completat.

În partea de jos, există două secțiuni, prima fiind utilă în cazul utilizării aplicației în varianta sa mobilă. Primul buton este utilizat pentru a reduce înălțimea tichetului, al doilea și al treilea sunt

utilizate pentru a muta tichetul între coloane, iar al patrulea buton este folosit pentru accesa pagina de editare a tichetului.

A doua secțiune reprezintă versiunea pentru care tichetul este relevant, categoria în care se află, și raportul între orele înregistrate și orele estimate.

3.2.3 Filtre

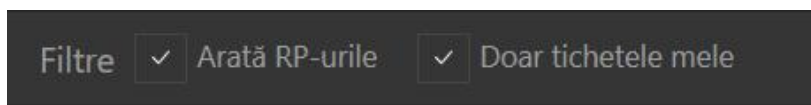


Fig. 17 - Filtre

Utilizatorul are la îndemână câteva filtre pe care sa le utilizeze pentru a discerne elementele de pe ecran, și poate selecta în a arata doar tichetele sau doar rapoartele de problemă, sau a vedea doar tichetele la care el participă.

3.2.4 Pagină creare

Creare Modul

Nume

Introdu nume

Numele modulului înregistrat

Atașează proiect

Introdu nume

Proiectul în care se va afla modulul

Prioritate

Minimă

Prioritatea modulului










Asignat

Alocă mie

Introdu nume persoană

Proprietarul curent al modulului

Descriere

B *I* U         

Descrierea modulului

Atașează versiune

Introdu nume

Versiune la care se atașează modulul

Atașează categorie

Introdu nume

Categoria modulului

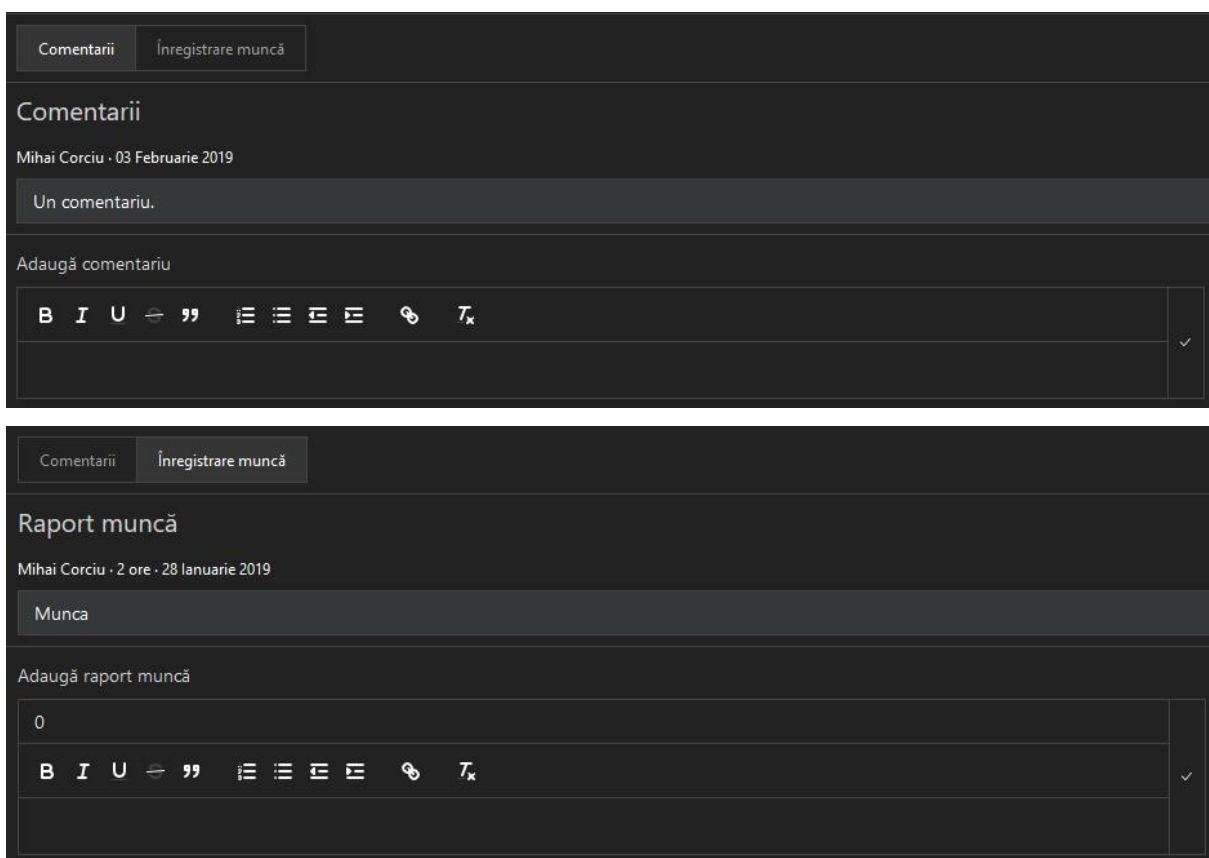
Adaugă modulul

Anulează

Fig. 18 - Exemplu pagină creare

În imaginea de mai sus vedem un exemplu de pagină de creare, mai exact fiind pagina de creare a unui modul. Există câteva câmpuri care întotdeauna sunt obligatorii, acestea fiind câmpul de nume, câmpul de proiect, iar în cazul tichetului și raportului de problemă, câmpul modulului. Câmpurile care specifică utilizatori vor fi întotdeauna completate utilizând persoana autenticată în mod curent, dacă aceasta decide să nu modifice această secțiune. Câmpul de prioritate este întotdeauna completat cu cea mai mică prioritate în cazul tichetului și modulului, și cu cea mai mare prioritate în cazul raportului de problemă. Restul câmpurilor sunt opționale, și pot fi modificate mai târziu din pagina de editare.

3.2.4 Secțiune comentarii



The image displays two screenshots of a web application interface, both featuring a dark theme. The top screenshot shows the 'Comentarii' (Comments) section, which includes a header with 'Comentarii' and 'Înregistrare muncă' buttons. Below the header, there is a text input field with the placeholder 'Un comentariu.' and a rich text editor with various formatting options. The bottom screenshot shows the 'Raport muncă' (Work Report) section, which also has a header with 'Comentarii' and 'Înregistrare muncă' buttons. Below the header, there is a text input field with the placeholder 'Munca' and a rich text editor with various formatting options. Both sections have a 'Adaugă' (Add) button next to the text input field.

Fig. 19 - Secțiune comentarii și raport muncă

Secțiunea de comentarii și înregistrare muncă este o caracteristică care apare doar în cazul tichetelor sau a raportului de muncă. Diferența dintre cele două este că în cazul înregistrării muncii, utilizatorul este obligat să specifice câte ore a petrecut pe sarcina respectivă.

4 Detalii tehnice

4.1 package.json

```
{
  "name": "knbn",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "dependencies": {
    "axios": "^0.17.1",
    "babel": "^6.23.0",
    "babel-loader": "^7.1.2",
    "babel-plugin-add-module-exports": "^0.2.1",
    "babel-plugin-react-html-attrs": "^2.1.0",
    "babel-plugin-transform-class-properties": "^6.24.1",
    "babel-plugin-transform-decorators-legacy": "^1.3.4",
    "babel-preset-env": "^1.6.1",
    "babel-preset-es2016": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.18.2",
    "cookie-parser": "^1.4.3",
    "cookies": "^0.7.1",
    "dateformat": "^3.0.3",
    "express": "^4.16.4",
    "express-react": "^1.0.2",
    "express-react-views": "^0.10.4",
    "express-session": "^1.15.6",
    "express-validator": "^4.3.0",
    "fs": "0.0.1-security",
    "jsonwebtoken": "^8.4.0",
    "mysql": "^2.15.0",
    "nodemon": "^1.12.1",
    "passport": "^0.4.0",
    "passport-local": "^1.0.0",
    "pug": "^2.0.0-rc.4",
    "react": "^16.4.0",
    "react-addons-update": "^15.6.2",
    "react-dnd": "^2.5.4",
    "react-dnd-html5-backend": "^2.5.4",
    "react-dom": "^16.2.0",
    "react-html-parser": "^2.0.2",
    "react-quill": "^1.3.3",
    "react-redux": "^6.0.0",
    "react-router-dom": "^4.2.2",
    "react-update": "^0.4.4",
    "react-validation": "^3.0.7",
    "react-view-engine": "0.0.1",
    "universal-cookie": "^3.0.7",
    "webpack": "^3.8.1"
  },
  "devDependencies": {
    "babel-plugin-transform-async-to-generator": "^6.24.1"
  }
}
```

```

},
"scripts": {
  "start": "webpack --watch"
},
"author": "Mihai Corciu",
"license": "ISC"
}

```

Tabela 3 - Conținutul fișierului de configurare package.json

Conținutul fișierului de mai sus reprezintă un obiect JSON, pentru a fi interpretat cu succes de managerul de pachete *npm* și cele mai importante aspecte care trebuie prevăzute sunt *name*, care este numele curent, *main*, care este fișierul sursă principal, *dependencies*, bibliotecile de care implementarea aplicației depinde și *author*, ce reprezintă autorul ei.

4.2 webpack.config.js

```

var debug = process.env.NODE_ENV !== "production";
var webpack = require('webpack');
module.exports = {
  context: __dirname,
  devtool: debug ? "inline-sourcemap" : null,
  entry: {
    reactRouter: "./public/routers/ReactRouter.js"
  },
  module:{
    loaders: [{
      test: /\.js?$/,
      loader: 'babel-loader',
      exclude: /(node_modules)/,
      query: {
        presets: ['react', 'es2016'],
        plugins: ['react-html-attrs', 'transform-class-properties',
'transform-decorators-legacy', "transform-async-to-generator"]
      }
    }
  ],
  output: {
    path: __dirname + "/public/js",
    filename: "[name].min.js"
  },
  node: {
    fs: "empty"
  },
  plugins: debug ? [] : [
    new webpack.optimize.DedupePlugin(),
    new webpack.optimize.OccurenceOrderPlugin(),
    new webpack.optimize.UglifyJsPlugin({ mangle: false, sourcemap:
false })],
  ]};

```

Tabela 4 - Conținutul fișierului de configurare webpack.config.js

Webpack reprezintă o bibliotecă denumită *module bundler*, și scopul ei principal este cel de a procesa tot codul aplicației în module. Acesta poate fi declarat în această configurație în secțiunea

entry, prin perechi de cheie și valoare, unde cheia este numele fișierului de ieșire după procesare, iar valoarea este calea relativă către fișierul de procesat.

În secțiunea *module*, atributul *loaders*, membrul *test*, putem denumi modelul denumirii fișierelor pe care dorim să le procesăm, iar membrul *exclude* este folosit pentru a ignora diferite directoare sau fișiere.

5 Direcții viitoare în dezvoltarea aplicației

5.1 Suport platformă *cloud*

O prima propunere în dezvoltarea pe o platforma *cloud* ar rezolva problema spațiului pentru un număr foarte mare de resurse și utilizatori. De asemenea, ar rezolva problema pierderii de informații în cazul unui dezastru natural, din cauza mecanismului de clonare și descentralizare a datelor pe care aceste platforme îl implementează.

5.2 Capacitate de atașare documente și fișiere

O altă particularitate ar fi capacitatea de atașare de documente diferitelor elemente din cadrul platformei, cum ar fi imagini și rapoarte care ar oferi explicații și modele pentru o mai bună implementare a funcției descrise de acestea.

5.3 Încărcare progresivă

O data cu creșterea numărului de elemente afișate pe ecran, trebuie implementat un mecanism de încărcare progresivă, în care ele să fie aduse în mod dinamic pe pagină, din baza de date, fără a afecta performanța navigatorului.

5.4 Secțiune specială de statistică

Unele date care se află în baza de date pot fi folosite pentru anumite elemente de tip statistic. De exemplu, se poate implementa un grafic care să descrie numărul de ore estimate versus orele înregistrate pentru un anumit proiect. Procedând astfel, se pot lua decizii de eficientizare a proceselor în cazul în care orele înregistrate le depășesc pe cele estimate.

6 Concluziile lucrării

Metodologia kanban este utilizată foarte mult în domeniul IT pentru modalitatea ușoară de a planifica sarcinile, dar poate fi folosită în orice mediu și de către oricine care dorește să-și organizeze munca printr-o metodă care pune accentul pe semnale vizuale.

Opinia mea personală este că aplicația descrisă în lucrarea de licență are capacitatea să devină un adevărat competitor pentru celelalte care există deja pe piață, aducând ca și element nou posibilitatea utilizării ei pe ecranele *desktop*, cât și cele mobile.

Biblioteca ReactJS schimbă direcția implementărilor paginilor web către una care se bazează pe reutilizarea resurselor, astfel fiind posibil realizarea, ca și în cazul acestui proiect, unei singure pagini în care să coexiste mai multe componente, care reacționează dinamic la acțiunile utilizatorului său, comportându-se exact ca o aplicație din domeniul mobil.

Baza de date MySQL este tehnologia cea mai utilizată în momentul actual în întreaga lume pentru modalitățile simple și eficiente de stocare a datelor.

Bibliotecile Nodejs și ExpressJS capătă teren în domeniul de dezvoltare al serverelor, captând din ce în ce mai mult atenția dezvoltatorilor din întreaga lume prin modul flexibil de a implementa stilul arhitectural REST și capacitatea de a suporta zeci de mii de conexiuni concurente, fiind posibilă extinderea aplicației în zona platformelor *cloud*.

Posibilitatea utilizării atât pe dispozitivele mobile cât și cele *desktop*, utilizând *framework*-ul Bootstrap reduce considerabil timpul de implementare.

În final, consider ca această platformă creată utilizând principiile din paradigma kanban oferă utilizatorilor săi o modalitate compactă și integrală de a-și crea sarcini și urmări munca pe care o depun, și de a fi capabili să își asume mai multă responsabilitate în cazul în care lucrează pe un proiect complex.

Bibliografie

1. John M. Gross, Kenneth R. McInnis - Kanban Made simple: Demistifying and Applying Toyota's Legendary Manufacturing Process
2. David J. Anderson - Kanban: Successful Evolutionary Change for Your Technology Business
3. Evan Hahn - Express in Action: Writing, Building, and Testing Node.js Applications
4. <https://www.inflectra.com/methodologies/kanban.aspx>
5. <https://www.atlassian.com/agile/kanban>
6. <https://getbootstrap.com/docs/4.2/getting-started/introduction>
7. <https://reactjs.org/docs/getting-started.html>
8. <https://webpack.js.org/concepts>
9. <https://nodejs.org/en/docs>
10. <https://www.codecademy.com/articles/what-is-rest>
11. <https://jwt.io/introduction>
12. <http://react-dnd.github.io/react-dnd/docs/api>